Fachhochschule Südwestfalen
University of Applied Sciences
Standort Hagen
Fachbereich Technische Betriebswirtschaft

Webtechnologie

Die Lehre von den Techniken des World Wide Web und von ihren Auswirkungen auf Wirtschaft und Gesellschaft

— Teil I: Grundlagen —

Vorlesungsskript

Andreas de Vries

Version: 24. Januar 2023

Inhaltsverzeichnis

Vorwort

Der Titel dieses Skripts erscheint auf den ersten Blick anachronistisch, also irgendwie „aus der Zeit gefallen“. Er verwendet das alte Wort *Technologie*, das sich vom griechischen *τέχνη techně* „Kunst, Handwerk“ und *λόγος logos* „Wort, Lehre Wissenschaft“ ableitet und daher treffend mit „Lehre von den Techniken“ übersetzt werden kann. Genau so definierte der Philosoph Christian Wolff den Begriff 1740.[1] Nun wurde der Ausdruck im Laufe des 20. Jahrhunderts von dem englischen *technology* überlagert, das im Wesentlichen dem deutschen Wort *Technik* entspricht.[2] So konnte sich im Englischen dann auch die Pluralform entwickeln, die mit der Bedeutung einer „Lehre“ nicht vereinbar ist, und heute spricht man ganz selbstverständlich von *Technologien*, meint aber *Techniken*. Entsprechend sollten die *web technologies* im Deutschen auch *Webtechniken* heißen. Aber sei's drum, Sprache lebt.

Was sind nun die Inhalte *der* Webtechnologie, also der „Lehre von den Webtechniken“? Sie umfasst einerseits die informatische Ebene des Internets, also die Behandlung sowohl seiner technischen Infrastruktur mit den verschiedenen Schichten und Protokollen, als auch seiner Programmiersprachen, Algorithmen und Speichermechanismen. Webtechnologie beschäftigt sich andererseits aber auch mit den wirtschaftlichen und gesellschaftlichen Wirkungen des Webs, also den Chancen und Gefahren, den Möglichkeiten und Grenzen dieser weltumfassenden informationellen Revolution, deren Keim mit Alan Turings Konzept einer Rechenmaschine 1936 entstand[3] und nun mit Social Media, dem „Internet der Dinge“ und mobilen Techniken zur bedeutendsten wirtschafts- und kulturhistorischen Umwälzung seit der Industriellen Revolution führte und deren psychologische, politische und gesellschaftliche Auswirkungen bislang allenfalls visionäre Kulturwissenschaftler wie Marshall McLuhan[4] erahnt haben. So verstanden gehört die Webtechnologie also zu einem der Kerngebiete der Wirtschaftsinformatik.

Ein wesentliches Ziel der Webtechnologie ist es, durch die Vermittlung der technischen Grundlagen unserer digital vernetzten Welt zu deren Verstehen und zu deren Gestaltung beizutragen. Damit will die Webtechnologie einer derzeit zu beobachtenden gesellschaftlichen Entwicklung entgegen treten: Immer mehr von uns bedienen sich einer immer höher entwickelten Technik, aber das Verständnis für die Funktionsweise der Hard- und Software, mit deren Hilfe wir arbeiten, plaudern oder flirten, koppelt sich immer weiter davon ab. Dieser Trend zu einem „digitalen Analphabetismus“ ist gefährlich. Denn solange die allem unterlegte Welt der Algorithmen, Programmiersprachen und Speichertechniken des Netzes uns magisch und unerklärlich erscheint, solange bleiben wir durch sie manipulierbar. Diese Welt verstehen, kritisieren und gestalten zu können ist die dringlichste Herausforderung für den modernen Menschen. Jedem von uns müssen die Mittel an die Hand gegeben werden, die alltäglich verwendete Technik zu durchschauen und sie zu beherrschen, um nicht von ihr beherrscht zu werden.

---

[1] *„Est itaque* Technologia *scientia artium & operum artis.“* Etwa: „So ist *Technologie* die Wissenschaft von den Handwerken und den Handwerkserzeugnissen.“ C. Wolff (1740): *Philosophia rationalis sive logica*, S. 33. `https://books.google.com/books?id=9mYOAAAAQAAJ&pg=PA33`

[2] `http://www.merriam-webster.com/dictionary/technology`

[3] Turing (1936–1937).

[4] McLuhan (1962); McLuhan (1964 (reissued 2001)).

Zum Inhalt dieses Skripts

An der FH Südwestfalen in Hagen wird Webtechnologie über zwei Semester angeboten. Im vorliegenden Skript über die Grundlagen der Webtechnologie werden nach einer Einführung der grundlegenden Begriffe wie HTML, CSS, XML und JSON Datenbankanbindungen über das Internet mit PHP als serverseitiger Technik behandelt, sowie JavaScript als zentrale clientseitige Webtechnik. Beide Techniken spielen eine ökonomisch wichtige Rolle, nicht nur zur Intensivierung von Kundenanbindungen oder als Erweiterungspotenzial für Vertriebskanäle, sondern auch für unternehmensinterne Prozesse im Rahmen eines Intranets. Als zentraler Datenspeicher wird die relationale Datenbank MariaDB behandelt.

Warum PHP und JavaScript? Mitte der 1990er Jahre entstanden, sind beide Sprachen immerhin mittlerweile in die Jahre gekommene und zudem nicht unumstrittene Sprachen. Allerdings hat sich PHP wegen seiner relativ einfachen Handhabbarkeit und dem Rückgriff auf frei verfügbare *Open-Source*-Produkte, vor allem in Kombination mit der Datenbank MariaDB (früher MySQL), zu einem De-Facto-Standard der Webtechniken entwickelt. JavaScript auf der anderen Seite läuft auf allen modernen Browsern und ist eine der Säulen moderner Webanwendungen, vor allem Google trieb den weltweiten Einsatz von JavaScript seit den 2000er Jahren massiv voran. Den Stellenwert der beiden Programmiersprachen für die Entwicklung von Webanwendungen kann man also kaum überschätzen. Zudem sind nach W3Techs (`http://w3techs.com/`) beide Sprachen die im Web meist verwendeten, PHP mit über 80 % als serverseitige Sprache und JavaScript mit über 90 % als clientseitige Sprache.

Warum relationale Datenbanken? Seit ihrer Entstehung in den 1970er Jahren sind relationale Datenbanken der De-facto-Standard für die Speicherung großer und komplexer Datenmengen. Kunden- und Produktinformationen in Unternehmen werden ebenso in relationalen Datenbanken gespeichert wie die Artikeleinträge der Wikipedia. Zwar werden seit den 2010er Jahren gerade von großen IT-Konzernen zunehmend auch sogenannte NoSQL-Datenbanken entwickelt und eingesetzt.[5] Allerdings sind sie für Anwendungen mit besonderen Anforderungen an die Hochverfügbarkeit großer Datenmengen (BigData und verteilte Datenspeicherung) ausgelegt, auf Kosten der zu jedem Zeitpunkt garantierten Datenkonsistenz.[6] Datenkonsistenz ist nach wie vor die Stärke relationaler Datenbanksysteme.

Lernziel dieses Skripts ist es, dass die Studierenden die Prinzipien der Programmierung webbasierter Datenbankanwendungen verstehen und anwenden können, die Struktur einer typischerweise dreischichtigen Webanwendung verstehen und darauf basierende Softwareentwürfe solcher Systeme verstehen und entwickeln können. Diese Themen bilden die wesentlichen Elemente der in letzter Zeit häufig so genannten „Fullstack-Entwicklung“. Als deutschsprachige Darstellung dazu ist Ackermann (2021) geeignet. Insgesamt werden als Grundlage und zur Ergänzung des hier behandelten Stoffs beispielhaft Maurice (2014), Theis (2018) und Wenz und Hauser (2021) empfohlen, daneben die Internetquellen `php.net` und `stackoverflow.com`.

Ein Hinweis noch zur Handhabung dieses Skripts. Betrachten Sie es als Obermenge zu dem Stoff, der in der Vorlesung und den Praktika behandelt wird. Diejenigen Teile des Skripts, die in den Veranstaltungen nicht behandelt werden, sollen Ihnen als Materialsammlung und Nachschlagewerk zum Thema dienen. Sie sind mit einem Sternchen * markiert. Da mir oft die Frage gestellt wird, welche Teile des Skripts denn für die Prüfungen verlangt werden, lautet die Antwort also: Die Beherrschung der Inhalte des gesamten Skripts ist hinreichend zum Bestehen der Prüfungen, das Beherrschen des Stoffs aus Vorlesung und Praktika (also ohne die Sternchen) aber ist notwendig.

Danksagungen. Am Ende möchte ich denjenigen danken, die zur Entstehung und Vollendung

[5] Google: Bigtable, Amazon: DynamoDB, Facebook: Cassandra, Twitter: FlockDB, SAP: HANA

[6] Das ist die Aussage des CAP-Theorems, `https://de.wikipedia.org/wiki/CAP-Theorem`

dieses Skripts beigetragen haben. Insbesondere wäre es ohne meinen langjährigen Freund und Mitarbeiter Ingo Schröder und seine wichtigen Anregungen, Ideen und Initiativen nicht in dieser Form entstanden. Mein Dank gilt zudem Robert Schweda, der als in dem Bereich der Webentwicklung professionell Tätiger nicht nur wichtige Impulse gegeben, sondern auch die Inhalte des Kapitels über CSS beigesteuert hat.

Hagen,
im Januar 2023 Andreas de Vries

Teil I

Informatische Grundlagen des Web

1

* Überblick über TCP/IP

Dieses Kapitel fasst die wesentlichen technischen Grundlagen der Webtechnologie zusammen. Es geht um die Frage, wie Daten und Informationen zwischen einem Browser und einem Webserver über das Internet übertragen werden können. Basis dafür sind die TCP/IP-Protokolle. Da viele Studierende diese bereits behandelt haben und sie überdies kein eigenes Prüfungsthema dieser Veranstaltung sind, ist dieses Kapitel mit einem Sternchen markiert. Für Studierende jedoch, die davon noch nichts gehört haben oder die eine Wiederauffrischung ihres Wissens benötigen, ist es trotzdem hier aufgeführt.

TCP/IP ist eine Protokollfamilie, die die Verständigung von Rechnern in Netzwerken, also insbesondere im Internet ermöglicht. Die Abkürzung steht für *Transmission Control Protocol / Internet Protocol*.

1.1 Das Prinzip einer Schichtenarchitektur

Hauptziel eines Schichtenmodells ist die Standardisierung der Kommunikation zwischen technischen Systemen. Solche Systeme können Computer, mobile Endgeräte oder auch chipbasierte Sensoren und Aktoren sein. Wie in jeder Kommunikation ist stets eines der Systeme in der Rolle

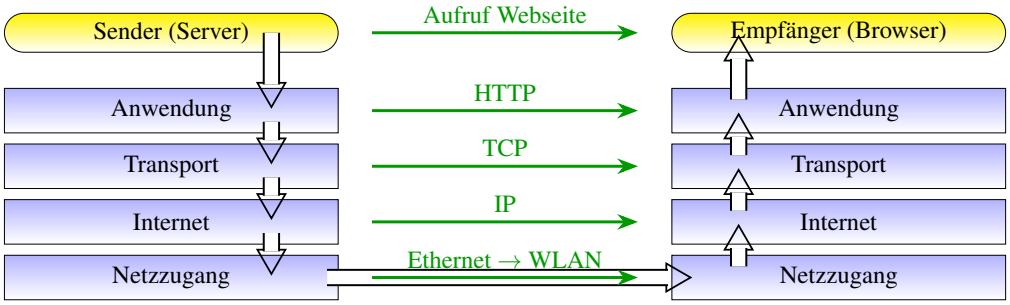


Abbildung 1.1: Schichten und Protokolle, hier am Beispiel der TCP/IP-Familie: die logische Kommunikation über Protokolle (→) verläuft horizontal innerhalb jeder Schicht, die reale physikalische Kommunikation (⇒) vertikal durch die Schichten.

des Senders und eines in der des Empfängers einer Nachricht. Im Laufe der Zeit können die Systeme ihre Rolle als Sender und Empfänger durchaus wechseln. In einer Schichtenarchitektur

werden aufeinanderfolgende Schichten (englisch: *layer*) mit jeweils spezifischen Aufgaben definiert. Sowohl Sender als auch Empfänger arbeiten in jeder Schicht nach festgelegten Regeln, die zu einem *Protokoll* zusammengefasst werden.

Entsprechend sind die verschiedenen Protokolle der TCP/IP-Familie in Abbildung 1.2 dargestellt. Zum Vergleich ist dort die OSI-Schichtung angegeben. ISO/OSI ist ein Standard für eine

<table>
<tr><th>OSI-Schicht</th><th>Internet-Schicht</th><th colspan="3">TCP/IP-basierte Protokolle</th></tr>
<tr><td>Anwendung</td><td rowspan="3">Anwendung</td><td>FTP
Telnet
HTTP
SMTP, XMPP, MQTT</td><td>SFTP
SSH
HTTPS</td><td></td></tr>
<tr><td>Darstellung</td><td></td><td></td><td></td></tr>
<tr><td>Sitzung</td><td></td><td></td><td></td></tr>
<tr><td>Transport</td><td>Transport</td><td>TCP</td><td>TLS</td><td>UDP</td></tr>
<tr><td>Vermittlung</td><td>Internet</td><td colspan="3">IP (IPv4, IPv6), ICMP</td></tr>
<tr><td>Sicherung</td><td rowspan="2">Netzzugang</td><td colspan="3">ARP</td></tr>
<tr><td>Bitübertragung</td><td colspan="3">IEEE 802.x, IEEE 811.xx, DSL, ISDN</td></tr>
</table>

Abbildung 1.2: TCP/IP-Protokollfamilie im Vergleich zum OSI-Referenzmodell

Protokollfamilie, die alle Schichten der Datenübertragung umfasst, von der Anwendungsschicht bis zur physikalischen. TCP/IP entstand früher als der ISO/OSI-Standard und ist daher unabhängig davon zu sehen. Im Sinne von ISO/OSI ist TCP/IP also keine vollständige Protokollarchitektur, es deckt nicht die unteren Schichten 1 und 2 ab, und an Stelle der drei Anwendungsschichten gibt es mehrere Protokolle, die OSI-Schicht übergreifend definiert sind, siehe Abbildung 1.2.

1.2 Adressierung mit Sockets und URL

Prozesse als laufende Programme auf Rechnern sind über Sockets mit Ports adressierbar. Ein Rechner, der einen Prozess zur Kommunikation mit einem Prozess auf einem anderen Rechner freigibt, öffnet einen bestimmten Port. Die Syntax eines Sockets lautet:

$$\underbrace{\textit{Protokoll}\texttt{://}\textit{IP-Adresse}\texttt{:}\textit{Port}}_{\textit{Socket}}$$

also z.B. `http://haegar.fh-swf.de:80`. (Genaugenommen ist ein *Socket* – „Steckdose“ – ein besonderer Datentyp im UNIX-Betriebssystem, der als Schnittstelle zwischen beliebigen Prozessen für sie Prozeduren zur Kommunikation bereitstellt.)

Zentral für jeden Internetdienst ist der *URL (Uniform Resource Locator)*,[1] der einen Dienst im Internet weltweit eindeutig aufruft. Das Adressformat ist wie folgt definiert:

protokoll`://`[*benutzer*[`:`*passwort*]`@`]*server*[`:`*port*] [`/`*pfad*[`?`*query*][`#`*fragment*]]

Kursiv Gedrucktes bezeichnet einzusetzende Variablen, die eckigen Klammern bezeichnen optionale Angaben, und die vorgegebenen Zeichen (`:`, `/`, `?`, `#`) sind in nichtproportionaler Schrift gedruckt. Hierbei bedeuten:

[1] `http://tools.ietf.org/html/rfc1738`

- *protokoll:* gewünschter Internetdienst (ftp, telnet, http, ...);
- *benutzer*:*passwort*`@` wird nur verwendet, wenn Passwortschutz gegeben ist (i.d.R. bei Telnet oder ftp)
- *server:* Domainname oder IP-Nummer des Servers
- *port:* die Nummer des Ports, d.h. des auf dem Server eingerichteten Verbindungspunktes (Socket);
- *pfad:* gibt den Pfadnamen des gewünschten Objekts

Ein URL ist nach RFC3986 (`http://tools.ietf.org/html/rfc3986`) ein spezieller URI (*Uniform Resource Identifier*), der auch allgemeinere, also nicht nur über das Internet verfügbare Ressourcen identifiziert.

1.3 IP

Das *Internet Protocol* (IP) ist für die logische Adressierung und Zustellung von Paketen zuständig. Das Paket wird über verschiedene physikalische Stellen weitergereicht, bis es die Zieladresse erreicht. IP arbeitet dabei *verbindungslos (connectionless)*, d.h. es kennt keine Zustände irgendeiner stehenden Verbindung („Session“), auch wenn sie in einer höheren Schicht oder einer Anwendung aufgebaut werden. Jedes Paket wird unabhängig von den anderen behandelt, IP weiß nichts davon, ob Pakete zusammengehören. IP ist also wie der Paketdienst eines Versandhandels. Gibt man im Versandhandel eine Bestellung auf, stellt der Versender die Lieferung zusammen. Es kann dabei durchaus vorkommen, dass mehrere Pakete einzeln auf den Weg geschickt werden. Der Paketdienst (IP) behandelt alle Pakete gleich, er weiß nicht, ob sie zusammen gehören.

Nach der IP-Adressierung Version IPv4 ist eine IP-Adresse 4 Byte (32 bit) lang, und 16 Byte = 128 bit nach der erweiterten Version IPv6. In IPv4 wird zur besseren Lesbarkeit die „Dotted Decimal“-Darstellung verwendet, die jedes der 4 Bytes getrennt als Dezimalzahl darstellt.

1. Byte	**2. Byte**	**3. Byte**	**4. Byte**
194	94	2	21

1.4 UDP

UDP *(User Datagram Protocol)* ist ein Transportdienst gemäß OSI-Schicht 4 wie TCP, jedoch *verbindungslos*. Wie IP schickt UDP Datenpakete ungeprüft „so wie sie kommen“ und weiß gar nicht, welche zusammen gehören und welche nicht. Insbesondere werden Fehler wie Datenverlust oder eine falsche Reihenfolge der Pakete nicht bemerkt.

Der Vorteil von UDP gegenüber TCP ist, dass es ein schnelles Protokoll ist. Es wird vor allem dort eingesetzt, wo es auf einen geringen Overhead ankommt. Typische Einsatzgebiete von UDP sind Anwendungen mit Frage-Antwort-Charakter, etwa ein Ping („schick mich zurück“), eine Anfrage beim Name-Server („Welche IP-Nummer gehört zu dem Domain-Name?“) oder eine Synchronisation der Netzwerkzeit von Servern (NTP, *network time protocol*). Hier geht es um einen simplen Austausch von Informationen, und wenn die Antwort nicht zurückkommt, wird einfach nochmal gefragt. Auch für Audio- und Videoübertragungen eignet sich UDP besser als TCP.

1.5 TCP

Während das Internetprotokoll IP eine logische Kommunikation zwischen zwei Rechnern ermöglicht, verbindet das *Transmission Control Protocol* TCP zwei Prozesse („Programme“) miteinander. Im OSI-Schichtenmodell findet sich TCP in der Transportschicht (OSI-Schicht 4) wieder, siehe Abb. 1.2 (S. 10), und wird über IP übertragen.

TCP ist verbindungsorientiert

TCP ist ein *verbindungsorientiertes* Protokoll, das heißt es wird eine bidirektionale Verbindung zwischen zwei Prozessen aufgebaut, über eine gewisse Zeit aufrecht erhalten und schließlich wieder geordnet beendet. Ohne einen Verbindungsaufbau ist erst gar kein Datentransfer möglich. Jeder Empfang eines Datenverkehrs wird quittiert (ähnlich wie bei einem Einschreiben), so dass der Sender genau weiß, welche Pakete auf jeden Fall angekommen sind.

Man kann sich eine verbindungsorientierte Kommunikation wie ein Telefonat vorstellen: Jemand ruft an, der Angerufene muss explizit abnehmen (bestätigen), erst dann kommt eine Verbindung zustande. Während des Verbindungsaufbaus werden noch keine Nutzdaten übertragen, und am Ende des Gesprächs wird die Verbindung explizit beendet, indem jeder Teilnehmer auflegt.

Betrachten wir uns die Eigenschaften einer TCP-Verbindung etwas genauer. Die Datenpakete bei TCP heißen *Segmente*. Der Verbindungsaufbau bei TCP erfolgt in drei Schritten und wird daher auch als *three way handshake* bezeichnet (Abb. 1.3).

1. Der Client initiiert die Verbindung, indem er dem Server ein TCP-Segment sendet, bei dem ein bestimmtes Bit, das SYN-Flag *(synchronize)*, gesetzt ist und das eine zufällige Sequenznummer (ISN *initial sequence number*) beinhaltet. Diese Sequenznummer ist ein 32-Bit-Feld wird bei jedem nachfolgend verschickten TCP-Segment der bestehenden Verbindung hoch gezählt.

2. Der Server sendet ein TCP-Segment zurück, bei dem das SYN- und das ACK-Flag *(acknowledge)* gesetzt sind und das eine zufällige ISN sowie die Acknowledge-Nummer enthält, die einfach die Client-ISN um eins erhöht.

3. Der Client bestätigt den Empfang des SYN-Flags des Servers durch ein Segment, bei dem das ACK-Flag gesetzt ist mit der ISN des Servers um eins erhöht.

Die ACK-Flags stellen also so etwas wie Empfangsbestätigungen dar, mit denen der Sender eines Segments genau weiß, was auf jeden Fall beim Empfänger angekommen ist. Der Verbindungsabbau funktioniert ähnlich wie der Aufbau, nur dass an die Stelle des SYN-Flags das FIN-Flag tritt. Allerdings gibt es bei einem Verbindungsabbau das Problem, dass der Client nicht einseitig

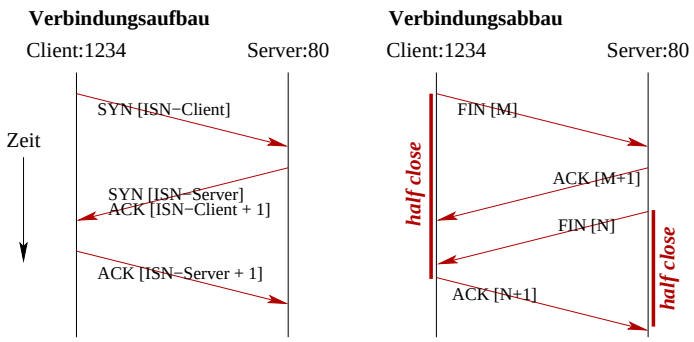


Abbildung 1.3: Verbindungsaufbau und -abbau bei TCP. Die Sequenznummer *M* beim Verbindungsabbau ergibt sich aus der Summe von ISN-Client und der Anzahl der vom Client in der Verbindung verschickten TCP-Segmente; entsprechend ist *N* die Summe von ISN-Server und der Anzahl der vom Server verschickten TCP-Segmente.

die Verbindung beenden kann (da hinkt unser Telefonbeispiel von oben, denn ein Telefonat kann

einseitig beendet werden!), wenn nämlich der Server noch nicht alle Daten versendet hat. Die Lösung ist ein Verbindungszustand *half close*, nachdem also eine Station das FIN-Segment verschickt hat und auf das ACK-Segment wartet; in diesem Zustand kann sie keine weiteren Daten senden außer einem ACK-Segment. Abb. 1.3 rechts zeigt einen Verbindungsabbau schematisch. Erst nach der letzten Bestätigung ist die Verbindung sauber beendet.

Es wird von TCP eine Fehlerüberprüfung und Flusskontrolle durchgeführt, die folgende Fehler bemerkt und ggf. durch erneutes Senden korrigieren lässt.

- Datenverlust beim Transport;
- fehlerhafte empfangene Daten;
- falsche Reihenfolge der Datenpakete;
- netzwerkbedingte Verzögerung der Datenlieferung.

Notwendig für diese Funktionen sind Sende- und Empfangspuffer, also temporäre Speicher, in die die Daten der Verbindung zwischengespeichert werden (Abb. 1.4).

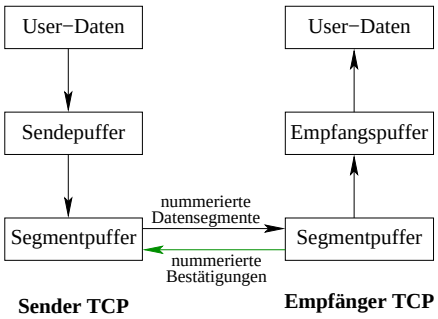


Abbildung 1.4: Sende- und Empfangspuffer für Datensegmente bei TCP.

Eine TCP-Verbindung ist bidirektional, man spricht daher auch von einer *Full-Duplex*-Fähigkeit von TCP. Um auch größere Datenmengen austauschen zu können, ohne gleich bei jedem kleinen Segment die Quittungen auszutauschen, gibt es das *TCP-Window*: Der Empfänger teilt über ein Feld im TCP-Segment seine TCP-Window-Größe mit, also wieviel Platz er aktuell in seinem Empfangspuffer hat.

1.6 HTTP und HTTPS

Der Anwender kommt mit TCP oder IP nicht in Berührung, denn die Anwendungsschicht ist nach Tabelle 1.2 die Schnittstelle und damit die „Vermittlungsschicht“ zwischen ihm und TCP und nachgelagert IP. Diejenigen Protokolle, die er verwendet, sind entsprechend die *Anwendungsprotokolle* oder *Internet-Dienste*. Es sind Dienste im Sinne der Client/Server-Architektur, die alle auf TCP oder UDP basieren.

Als endbenutzerorientiertes Anwendungssystem besteht das WWW aus Web-Servern, die HTML-Dokumente bereitstellen, die von Web-Clients, den Browsern, abgerufen und dargestellt werden. Das Übertragungsprotokoll für diese Anwendungsschicht ist HTTP *(hypertext transfer protocol)*. Es basiert auf TCP und ist standardmäßig auf Port 80 des Web-Servers. HTTP ist ein *zustandsloses (stateless)* Protokoll, d.h. es gibt mit HTTP nur isolierte Transaktionen, und Informationen jeder vorhergehenden Transaktion werden nicht gespeichert, sondern sofort vergessen. Insbesondere können über HTTP also keine Verbindungen oder Sessions vermittelt werden, dazu sind ergänzende Speichermechanismen am Browser oder am Server notwendig

HTTP ist zustandslos

(z.B. Cookies). Da ganz allgemein für eine Verbindung zwischen Kommunikationsteilnehmern Informationen über den Zustand der Verbindung nötig sind, ist ein zustandsloses Protokoll immer auch verbindungslos. Demgegenüber ist ein zustandsbehaftetes Protokoll nicht notwendig auch verbindungsorientiert: Beispielsweise ist IP zustandsbehaftet, aber verbindungslos[2].

Eine Transaktion in HTTP funktioniert nach dem Request-Response-Muster und besteht jeweils aus den folgenden vier Schritten:

1. Anfrage *(request)*, immer ausgehend vom Client,

2. Antwort *(response)* des Servers,

Die Kommunikationseinheiten in HTTP zwischen Client und Server heißen *Nachrichten (messages)*, von denen es also zwei Arten gibt, die Anfrage *(request)* vom Client und die Antwort (*response*) vom Server. Die Daten, die der Server bereitstellt, heißen *Ressource* und werden über einen eindeutigen URI (S. 11) adressiert. Jede Nachricht besteht aus zwei Teilen, dem Nachrichtenkopf *(message header)* mit Metadaten wie die Absendezeit, die verwendete Textkodierung oder Clientinformationen, und dem Nachrichtenrumpf *(message body)*, der die Nutzdaten enthält.

Eine Anfrage muss stets ihre Methode spezifizieren, die eine von neun gemäß HTTP definierten und in Tabelle 1.1 angegebenen Methoden sein muss.[3] Diese Methoden werden spezifiziert in

Methode	**Beschreibung der Anfrage**
OPTIONS	prüft, welche Methoden auf einer Ressource zur Verfügung stehen.
TRACE	gibt die Anfrage zurück, wie sie der Zielserver erhält.
HEAD	fordert Metadaten der angegebenen Ressource an. HEAD entspricht also GET, nur dass kein Nachrichtenrumpf erwartet wird.
GET	fordert die angegebene Ressource vom Server an. GET weist keine Nebeneffekte auf, d.h. der Zustand des Servers wird nicht verändert.
DELETE	fordert die Löschung der angegebenen Ressource.
PUT	fordert das Anlegen einer Ressource unter dem angegebenen URI mit den angehängten Daten. Wenn die Ressource bereits existiert, wird sie geändert. Ein erfolgreiches Anlegen einer neuen Ressource wird mit dem Statuscode 201 (Created) angezeigt, im Falle einer geänderten Ressource 200 (OK) oder 204 (No Content).
PATCH	ein Teil der angegebenen Ressource wird geändert. PATCH bewirkt also Nebeneffekte auf dem Server. (Definiert seit Juni 2010 RFC 5789.)
POST	übermittelt Daten an die angegebene Ressource, die der Server weiterverarbeiten soll. Das Ergebnis dieser Verarbeitung kann eine neue Ressource sein, deren URI der Server mit dem Statuscode 201 (Created) zurücksendet oder mit 200 (OK) bzw. 204 (No Content) quittiert.
CONNECT	fordert, die Verbindung durch einen TCP-Tunnel zu leiten. Wird meist eingesetzt, um eine HTTPS-Verbindung über einen HTTP-Proxy herzustellen.

Tabelle 1.1: Anfragemethoden gemäß HTTP.

idempotente Methoden und in sichere Methoden: Eine *idempotente Methode* kann Nebeneffekte *(side effects)* haben, d.h. den Zustand des Servers verändern, aber eine Wiederholung derselben Anfrage hat dann keinen weiteren Nebeneffekt mehr. Spezielle idempotente Methoden sind die *sicheren (safe) Methoden*, die auch beim ersten Aufruf keine Nebeneffekte bewirken.

Idempotente Methoden:	OPTIONS, TRACE, HEAD, GET, PUT, DELETE
Sichere Methoden:	OPTIONS, TRACE, HEAD, GET

[2] Sosinsky (2009):S. 476.

[3] `https://tools.ietf.org/html/rfc7231#section-4`, `https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods`

Obwohl GET in HTTP-Protokoll als sichere Methode definiert ist, kann nicht ausgeschlossen werden, dass sie dennoch Nebeneffekte erzeugt: Im Prinzip kann eine dynamische Ressource, z.B. ein PHP-Skript, durchaus den Zustand des Servers verändern. Entscheidend bleibt jedoch, dass sich der anfragende Client gemäß HTTP nicht darauf verlassen kann, eine GET-Anfrage gibt ihm die Ressource eben nur „zur Ansicht“.

PATCH und PUT unterscheiden sich dadurch, dass PATCH bei einer noch nicht vorhandenen Ressource diese nicht notwendig anlegt. Der grundlegende Unterschied zwischen POST und PUT ist, dass bei POST der URI die Ressource adressiert, die die Anfrage bearbeiten soll, während PUT anfragt, die Ressource unter diesem URI anzulegen. Entsprechend kann die Ressource einer POST-Anfrage ein Prozess (z.B. ein PHP-Skript, s.u.) sein, der die Daten serverseitig beliebig verarbeitet, wohingegen bei einer PUT-Anfrage die übermittelten Daten als Ressource unter der angegebenen URI auf dem Server gespeichert werden müssen.

Welche der Anfragemethode für einen gegebenen Webserver möglich ist, hängt von dessen Konfiguration ab. Üblicherweise sind nur GET und POST ermöglicht.

Jeder Antwort auf eine HTTP-Anfrage wird ein dreistelliger Statuscode mitgeliefert, der über Erfolg, Warnungen oder Fehler der Anfrage informiert. Eine Übersicht über wichtige HTTP-

Code	Nachricht	Bedeutung
101	Switching Protocols	Der Server erkennt in der Anfrage die Aufforderung zu einem Wechsel des Protokolls und führt ihn aus. Ein solcher Wechsel wird z.B. bei Echtzeitkommunikation (Videokonferenzen) notwendig.
200	OK	Die Anfrage wurde erfolgreich bearbeitet und das Ergebnis der Anfrage wird in der Antwort übertragen.
404	not found	Die angeforderte Ressource wurde nicht gefunden. Dieser Statuscode kann ebenfalls verwendet werden, um eine Anfrage ohne näheren Grund abzuweisen. Links, welche auf solche Fehlerseiten verweisen, werden auch als „Tote Links“ bezeichnet.
500	internal server error	Unerwarteter Serverfehler

Tabelle 1.2: Wichtige HTTP-Statuscodes einer HTTP-Antwort.

Statuscodes gibt Tabelle 1.2, für eine vollständige Auflistung sei auf Abschnitt 6 des RFC7231 verwiesen: `https://tools.ietf.org/html/rfc7231#section-6`.

HTTPS (*HTTP Secure* oder *HTTP over TLS*) ist die verschlüsselte Variante von HTTP, es setzt auf TLS auf als Zwischenschicht zwischen TCP und HTTP.[4] Die zwei wesentlichen

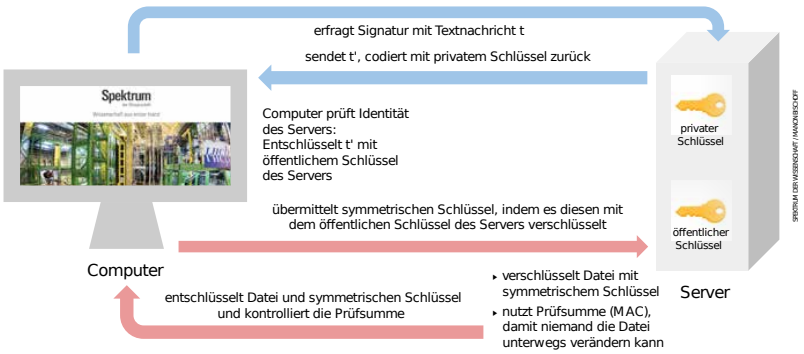


Abbildung 1.5: HTTPS ist mit TLS in 2 Phasen verschlüsseltes HTTP. Quelle: Spektrum der Wissenschaft.

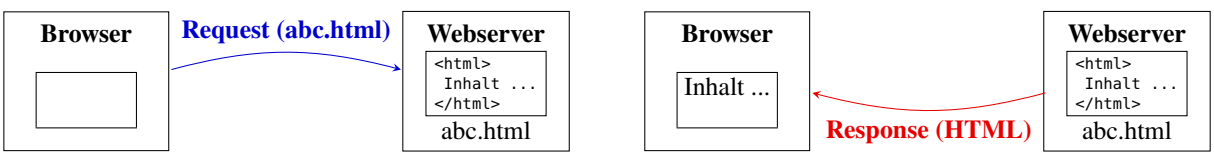
Vorbereitungsphasen von HTTPS sind in Abbildung 1.5 skizziert, also (1) die Authentifizierung des Servers und (2) die Schlüsselvereinbarung für die eigentliche Datenübertragung.

[4] `http://tools.ietf.org/html/rfc2818`

2 HTML

Kapitelübersicht

Was passiert eigentlich, wenn ein Browser eine Webadresse aufruft? Hinter jeder (gültigen) Webadresse verbirgt sich stets eine Datei oder eine Ressource auf dem Web-Server. Der Browser fordert diese Datei an (*request*). Der Webserver, der als eifriger Dienstleister permanent auf solche Requests horcht, antwortet auf sie umgehend mit einer *Response* (Antwort). Die ganze Kommunikation im Internet basiert auf diesem Frage-Antwort-Spiel. Stets fordert ein Browser eine Ressource an, erhält sie und stellt sie in seinem Fenster dar. Hat der Name der angeforderten Datei die Endung `.html` oder `.htm`, so interpretiert der Browser den Inhalt als HTML.



Dieses einfache Request-Response-Muster ist einerseits für den durchschlagenden Erfolg des Internets als Informationsmedium verantwortlich, da ein einzelner Server seine Daten nahezu gleichzeitig an Millionen Browser liefern kann. Andererseits aber sorgt für eine gewisse Starrheit, denn eine Server kann zunächst keine Informationen der Clients verarbeiten. Diesen Mangel zu

beheben ist nicht einfach, eine der möglichen Antworten darauf ist PHP. Doch bevor wir uns dieser Webtechnik zuwenden, wollen wir erst HTML kennen lernen.

2.1 Elemente und Tags, Inhalt und Attribute

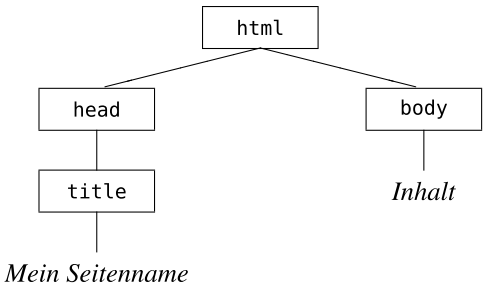
HTML ist eine einfache formale Sprache, die aus vordefinierten Formatierungsanweisungen besteht und von einem Interpreterprogramm, dem *Browser*, übersetzt und auf dem Bildschirm dargestellt wird. „HTML“ steht dabei als Abkürzung für *Hypertext Markup Language*, also etwa „Hypertext Markierungssprache“ oder „Hypertext Auszeichnungssprache“, wobei der Begriff *markup* ursprünglich aus dem Verlagswesen stammt und jene handschriftlichen Markierungen („Textauszeichnungen“) bezeichnet, die ein Layouter als Anmerkungen für den Setzer in ein Manuskript einfügt und die im fertigen Endausdruck natürlich nicht sichtbar sind.

Entsprechend bestehen HTML-Dokumente aus rein ASCII-textbasierten Anweisungen, den sogenannten *Elementen*. Es gibt eine bestimmte Menge von Elementen, die in einem HTML-Dokument von sogenannten *Tags* umschlossen sind. Tags tauchen also fast immer paarweise auf (Ausnahmen sind die leeren Elemente, s.u.). Man schreibt dabei das Anfangs-Tag, indem man den Namen des Elements in spitzen Klammern schreibt, <*name*>, und das Ende-Tag zusätzlich mit einem Schrägstrich *(slash)* </*name*> versieht, also z.B.

<*name*> . . . </*name*>

Der Inhalt „. . . “ des Elements steht innerhalb dieser Tags und kann wiederum Elemente enthalten. Wichtig ist dabei die Regel, dass ein inneres Element erst geschlossen sein muss, bevor ein äußeres geschlossen werden kann, also z.B.

```
<html>
  <head>
    <title>
       Mein Seitenname
    </title>
  </head>
  <body>
    Inhalt
  </body>
</html>
```



Auf diese Weise entsteht eine logische Baumstruktur ineinander geschachtelter Elemente, die rechts daneben dargestellt ist. Sie stellt in diesem Fall ein HTML-Dokument dar, das ein *head*- und ein *body*-Element als Kinder enthält, und *head* enthält wiederum als Kind das Element *title*. Die Elemente *title* und *body* enthalten als Inhalt *Mein Seitenname* und *Inhalt*.

Falls ein Element kein Kind und keinen Inhalt besitzt, heißt es *leer* und kann geschrieben werden als <*name*/>, d.h.

<*name*> </*name*> = <*name*/>.

Ein Element kann mehrere *Attribute* enthalten. Ein Attribut ist eine bestimmte Eigenschaft des Elements und kann einen Wert aus einer vorgegebenen Menge annehmen. Es wird in das Start-Tag des Elements geschrieben und sein Wert nach einem Gleichheitszeichen dahinter in Anführungszeichen:

<name attribut="wert"> ... </name>

In XML, dem allgemeinen „Dokumentenstandard“ des Web (siehe Abschnitt 4.1 ab Seite 38) spielt die Groß- und Kleinschreibung von Tags und Attributen eine Rolle, meist werden Elementnamen klein geschrieben.

2.1.1 Dateiendung `.html` und Kommentare

Als weitere allgemeine Information zu HTML-Dokumenten muss man wissen, dass sie eine Textdatei sind, erstellbar mit jedem einfachen Editor, und die Endung `.html` oder `.htm` haben müssen. Ferner können Kommentare an beliebiger Stelle nach dem Vorspann (s.u.) in ein HTML-Dokument eingefügt werden. Sie haben die Syntax

```
<!-- ... -->
```

Ein Kommentar ist also ein einzelner Tag, allerdings kein Element; es kann und darf daher nicht geschlossen werden. Kommentare werden vom HTML-Interpreter des Browsers ignoriert und also im Browserfenster nicht dargestellt. Das gleiche Schicksal ereilt auch jedes Leerraumzeichen *(Whitespace)*, also Leerzeichen, Tabulator, Zeilenumbruch usw., das auf ein Leerraumzeichen folgt.

2.2 Erstes Beispiel

Das folgende Beispiel gibt ein vollständiges HTML-Dokument wieder und verwendet einige derjenigen Elemente, die wir im Folgenden benötigen werden. Es ist eine reine Textdatei und *muss* als Endung `.html` oder `.htm` haben, um vom Browser bzw. vom Webserver als HTML-Dokument erkannt zu werden. Das Dokument wird dann wie in Abb. 2.1 dargestellt.

Listing 2.1: Einfaches HTML-Dokument zur Begrüßung

```
<!DOCTYPE html>
<html>
  <head>
    <title>Meine erste Seite</title>
  </head>

  <body>
    <h1>Hallo Welt!</h1>
    <p>
      Dies ist ein vollständiger Absatz.
    </p>
    Hier befindet sich nur Text, der hier
    <br/>
    umgebrochen wird,
  </body>
</html>
```

2.2.1 Der Vorspann, Zeichensatzkodierung

Die erste Zeile des HTML-Dokuments ist der Vorspann, der die genaue HTML-Version des Dokument bestimmt. Die derzeit übliche ist HTML5. Wie in jeder der gängigen Programmiersprachen ist die standardgemäße Textcodierung für HTML der ASCII-Code, kennt also keine

Abbildung 2.1: Die Ausgabe von Listing 2.1.

Sonderzeichen wie z.B. ä, ß, ç, usw. Möchte man Sonderzeichen darstellen, so gibt es zwei Möglichkeiten. Einerseits gibt es dafür in HTML Sonderbefehle, z.B.:

Zeichen	HTML-Befehl	Erläuterung
ä, Ä	`&auml; &Auml;`	„a Umlaut“, „A Umlaut“
ß	`&szlig;`	„sz Ligatur“
€	`&euro;`	
φ	`&#966; = &#x3C6;`	966 = `0x3C6` ist der Unicode von φ

(Achtung: das Semikolon gehört zu den Anweisungen!) Eine vollständige Liste aller in HTML darstellbaren Zeichen finden Sie unter

`https://wiki.selfhtml.org/wiki/Referenz:HTML/Zeichenreferenz`

Eine andere (und bequemere) Möglichkeit ist es, innerhalb des `<meta>`-Tags im `<head>`-Tag das Attribut `charset` auf die Kodierung einzustellen, also

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
  </head>
  ...
```

Üblich sind die beiden Werte `"UTF-8"` für Unicode und `"ISO-8859-1"` für den erweiterten ASCII-Zeichensatz. Die mächtigere von beiden Kodierungen ist natürlich UTF-8, mit dem alle international gebräuchlichen Schriftsätze dargestellt werden können, wie die Internetseite `https://wikipedia.org` eindrucksvoll zeigt.

Sie können Ihr HTML-Dokument auf Gültigkeit überprüfen lassen, indem Sie den HTML-Validator des W3-Consortiums (W3C) verwenden, `http://validator.w3.org/`

Bis auf die Angabe des Dokumententyps hat der Vorspann keinen Einfluss auf die Darstellung des HTML-Dokuments, also der „Webseite“. Das Dokument selber besteht aus einem einzigen `<html>`-Element. Hier muss der Inhalt des Dokuments festgelegt werden.

2.2.2 `<head>` und `<body>`

Die beiden Kindelemente eines `<html>`-Elements sind `<head>` und `<body>`. Das `<head>`-Element kann Zusatzinformationen („Metainformationen“) enthalten sowie im Element `<title>` den

Namen des Dokuments, der in der Titelzeile des angezeigten Fensters erscheint. Der Fensterinhalt wird im `<body>`-Element festgelegt. Jeder hier erstellte Text wird im Fenster dargestellt, es sei denn er ist in einem Tag.

2.2.3 Überschriften, Absätze und Zeilenumbrüche

Der im `<body>`-Element erstellte Text kann mit Tag-Formatierungen nun gestaltet werden. Eine Überschrift kann mit dem `<h1>`-Element erstelllt werden, wobei ‚h' für *header* steht und die 1 für die Ebene; es gibt weitere Elemente `<h2> ... <h6>`, die jeweils verschiedene Unterüberschriften erstellen. Der Inhalt der Überschrift ist dann ganz einfach der Text innerhalb der beiden Tags `<h1> ... </h1>`.

Ein Absatz wird in HTML durch das `<p>`-Element markiert, der Browser stellt seinen Inhalt passend im Fenster dar und bricht abhängig von der Fensterbreite automatisch um. Standardmäßig wird der Text linksbündig dargestellt. Andere Formatierungen kann man mit dem Attribut `style` erreichen:

HTML-Anweisung		Wirkung
`<p style="text-align:justify">`	`... </p>`	Absatz in Blocksatz
`<p style="text-align:center">`	`... </p>`	Absatz zentriert
`<p style="text-align:right">`	`... </p>`	Absatz rechtsbündig

Beachten Sie die Anweisung als Attribut-Wert-Paar. Der Wert ist hier übrigens eine CSS-Anweisung. (Auf CSS werden wir weiter unten noch etwas genauer eingehen.) Früher wurde zur Zentrierung oft das `<center>`-Element verwendet; es gilt heute jedoch als *deprecated* („abgelehnt") und sollte unbedingt vermieden werden. Eine Auflistung von veralteten HTML-Anweisungen finden Sie unter:

`http://wiki.selfhtml.org/wiki/HTML/deprecated`

Möchte man einen Zeilenumbruch im laufenden Text erzwingen, so kann man es mit dem `<br>`-Element (für *break*) erreichen. Da es ein leeres Element ist, sollte man es XML-konform als `<br/>` verwenden.

2.3 Pfade und Hyperlinks

Das ‚H' in HTML geht auf *Hypertext* zurück, eine Eigenschaft, die maßgeblich zum Erfolg des Webs beigetragen hat: Man kann nämlich an beliebiger Stelle im Text einen *Hyperlink*, oder kurz *Link*, setzen, also einen Verweis auf irgendeine Webadresse im Internet, auf die man durch Anklicken gelangen kann.

Verantwortlich dafür ist das `<a>`-Element (für *anchor* – Anker), das wir im folgenden Beispiel kennen lernen. Da wir dazu das Konzept der Pfade im Internet benötigen, das man auch zur Darstellung von Bildern einsetzen muss, ist gleich auch noch eine Grafik eingebaut.

Listing 2.2: Hyperlinks

```
<!DOCTYPE html>
<html>
  <head><title>Links</title></head>
  <body>
    Hier geht es zur <a href="http://www.fh-swf.de">FH Südwestfalen</a>
    mit einem absoluten Pfad,
```

```
    deren Logo so aussieht:
    <br/>
    <img src="http://haegar.fh-swf.de/grafiken/logo.gif" width="100"
         border="0" alt="Logo"/>.
    <br/>
    Hier geht es zu einem
    <a href="../programme/auszeichnungen.html">HTML-Dokument</a>
    mit einem relativen Pfad zum Nachbarverzeichnis <tt>/programme</tt>.
  </body>
</html>
```

Im Browser ergibt sich damit die Darstellung in Abb. 2.2.

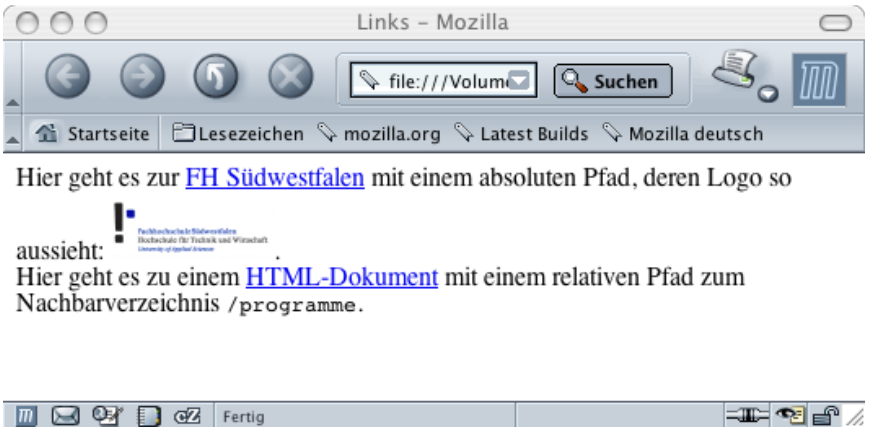


Abbildung 2.2: Die Ausgabe von Listing 2.2.

HTML-Dateien bestehen nur aus Text. Dennoch können sie Grafiken enthalten. Solche Elemente werden in HTML in Form eines Verweises auf eine Grafik notiert. Auch ein ausführbarer Verweis zu einer anderen eigenen oder fremden Web-Seite ist nur ausführbar, wenn er sein Verweisziel korrekt und genau benennt. Für all diese Zwecke wird das Referenzieren mit Pfaden in HTML benötigt.

Jede Datei im Internet hat eine weltweit eindeutige Adresse, den *URI (Uniform Resource Identifier)*. Das ist eine vollständige Webadresse wie

`http://haegar.fh-swf.de/grafiken/logo.gif`

Das ist ein *absoluter Pfad*. Ein Pfad ist üblicherweise schreibungssensitiv *(case sensitive)*, `/Pfad` verweist also auf etwas anderes als `/pfad`. Das Ende einer Hierarchiestufe – sei es der Rechner oder ein Verzeichnis – wird stets mit einem Slash `/` gekennzeichnet. (Beachten Sie: die DOS-Notation mit dem *Backslash* `\` funktioniert nicht im Internet!).

Im Gegensatz dazu gibt es *relative Pfade*, die den Ort einer Datei relativ zum aktuellen Dokument angeben. Hierbei gibt es die beiden Notationen

./Verzeichnispfad oder *../Verzeichnispfad*

Ein einzelner Punkt vor dem Verzeichnispfad (der zu einer Datei – manchmal auch nur zu einem Unterverzeichnis – führt) besagt, dass der Startpunkt sich im Verzeichnis des aktuellen Dokuments befindet. Zwei Punkte bedeuten, dass man zum Startpunkt des Verzeichnispfades vom aktuellen Dokument eine Verzeichnisstufe höher und von dort den angegebenen Verzeichnispfad entlang gehen muss.

Im Beispiel 2.2 sind die beiden Adressen

`http://www.fh-swf.de` und `http://haegar.fh-swf.de/grafiken/logo.gif`

als zwei absolute Pfade angegeben. Der erste zeigt nur auf das Hauptverzeichnis des Rechners `www.fh-swf.de`, der zweite auf die Datei `logo.gif` im Verzeichnis `/grafiken` im Hauptverzeichnis. Das Hauptverzeichnis eines Rechners ist üblicherweise das Verzeichnis `htdocs`.

Demgegenüber ist `.auszeichnungen.html` ein relativer Pfad, der auf die Datei `auszeichnungen.html` verweist, die sich neben dem Verzeichnis des aktuellen Dokuments befindet. Würde sich die Datei im gleichen Verzeichnis befinden, würde man schreiben `./auszeichnungen.html`.

2.3.1 Hyperlinks

Hyperlinks, also anklickbare Verweise auf andere Dateien, werden in HTML mit dem `<a>`-Element (*anchor* = Anker) erstellt. Der Start-Tag muss zwingend das Attribut `href` besitzen, das den Pfad zu der verlinkten Datei als Wert erhält. Innerhalb der beiden Tags `<a>` und `</a>` befindet sich der Text (oder auch das Bild), der markiert erscheint und den man anklicken kann, um den Link auszuführen. Der Pfad als Wert des `href`-Attributs kann absolut oder relativ zum aktuellen Dokument sein, wie im Beispiel:

```
<a href="http://www.fh-swf.de">FH Südwestfalen</a>
```

oder

```
<a href=".auszeichnungen.html">HTML-Dokument</a>
```

Bei dem ersten Link ist noch anzumerken, dass der Pfad ja gar nicht auf eine Datei verweist, sondern nur auf das Hauptverzeichnis. Wieso wird dann trotzdem eine Webseite angezeigt? Allgemein gibt der Webserver den Inhalt der Datei `index.html` an den Browser. Falls sich keine Datei mit diesem Namen in dem verlinkten Verzeichnis befindet, wird auch tatsächlich keine Webseite angezeigt! (Je nach Einstellung des Webservers sieht man dann tatsächlich nichts oder eine Liste aller Dateien des Verzeichnisses, die man dann einzeln anklicken kann.) Wenn sich eine Datei namens `index.html` in dem Verzeichnis befindet, sind daher die drei folgenden Pfade äquivalent:

```
http://www.fh-swf.de = http://www.fh-swf.de/ = http://www.fh-swf.de/index.html
```

2.3.2 Bilder

Bilder werden vom Browser durch das `<img>`-Element dargestellt, das zwingend das Attribut `src` (*source* = Quelle)[1] enthalten muss, dessen Wert der Pfad zu dem gewünschten Bild ist. Ähnlich wie der Zeilenumbruch `<br/>` ist auch das `<img>`-Element leer, besteht also aus nur einem Tag.

Gemäß Standard ist ebenso das Attribut `alt` erforderlich, dass das anzuzeigende Bild kurz beschreibt. Dieser Text wird angezeigt, wenn das Bild nicht dargestellt werden kann. Ein „minimales“ `<img>`-Tag hat also die folgende Gestalt,

```
<img src="http://haegar.fh-swf.de/garfik/logo.png" alt="Logo FH"/>
```

Optional sind die Attribute `width` und `height` für die Breite und Höhe des Bildes in Pixel, sowie `border` für die Dicke des Rahmens. Legt man nur eines der beiden Attribute `width` oder `height`

[1] Interessanterweise ist ein häufig gemachter Fehler, dass man das Attribut `scr` nennt, was die unangenehme Folge hat, dass der Browser das Bild nicht anzeigt (und natürlich auch sonst keine Fehlermeldung ausgibt); für diesen Fehler scheint es einerseits die Erklärung zu geben, dass man zu schnell tippt und den Dreher reinbekommt, oder dass man intuitiv bei Bildern an *Screen* denkt und so auf ‚scr' kommt. Daher: Es heißt `src`, von *Source*, *Source*, *Source*!

fest, so wird die andere Größe proportional errechnet; werden beide weg gelassen, so wird das Bild in Originalgröße angezeigt. Legt man beide fest, so wird das Bild entsprechend verzerrt.

Man sollte bei Bildern ein webfähiges Bildformat verwenden, das von allen Browsern dargestellt werden kann. Je nach Bildart[2] sind dies die Formate JPG, GIF und PNG. Verwenden Sie insbesondere keine BMP-Dateien.

2.4 Tabellen

Tabellen erlauben eine sehr übersichtliche Darstellung von Daten oder Information. Sie spielen daher insbesondere für die Präsentation von Daten aus Datenbanken eine wichtige Rolle. Da es in HTML möglich ist, Tabellen ohne Gitterlinien („blinde Tabellen“) zu erstellen, haben sich Tabellen auch als Gestaltungsmittel für Webseiten etabliert. Aus Gründen der Barrierefreiheit, also der Anforderung an Internetauftritte, die Information auch für behinderte Menschen, insbesondere Blinde, Sehbehinderte oder Farbenblinde, zugänglich zu halten, ist dies jedoch mit Bedacht zu tun. Empfehlungen zur Barrierefreiheit *(Web Content Accessability)* finden Sie unter `http://www.w3.org/TR/WAI-WEBCONTENT-TECHS/`.

2.4.1 Grundproblematik der Darstellung einer Tabelle im Browser

Eine Tabelle hat eine zweidimensionale Struktur, besitzt also zwei Richtungen. Bei Tabellen spricht man üblicherweise von Zeilen (in der Waagerechten) und Spalten (in der Senkrechten). Ein einzelner Eintrag, beispielsweise die zweite Zeile in der dritten Spalte, heißt Zelle.

Wir Menschen lesen Tabellen kontextabhängig. Zunächst überprüfen wir, wofür jede Zeile und Spalte steht, und gehen dann je nach Interesse spalten- oder zeilenweise durch die Zellen. So haben wir relativ schnell die Information aus einer Tabelle gewonnen.

Ein Browser kann solch ein bedeutungsabhängiges Verhalten natürlich nicht durchführen, er *versteht* sein Handeln ja nicht. Man muss sich also auf ein automatisches Vorgehen einigen, mit dem alle Browser eine Tabelle interpretieren: er geht zeilenweise durch die Tabelle (Abb. 2.3). Im

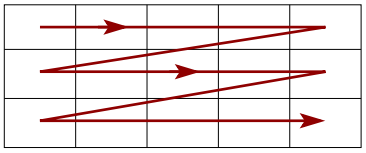


Abbildung 2.3: Eines Browser interpretiert eine Tabelle zeilenweise

Umkehrschluss bedeutet das, dass wir in HTML eine Tabelle zeilenweise aufschreiben müssen, unabhängig von der Bedeutung der Zeilen und Spalten.

2.4.2 Struktur einer Tabelle in HTML

Eine Tabelle wird mit dem `<table>`-Element erzeugt. Das `<table>`-Element enthält mindestens ein Kindelement `<tr>` (*table row* – Tabellenzeile), das `<tr>`-Element wiederum beinhaltet mindestens ein Kindelement `<th>` (*table head* – Tabellenkopf) oder `<td>` (*table data* – Tabellendaten, Zelle). `<th>` und `<td>`-Elemente sind gleichberechtigt, sie unterscheiden sich nur darin, dass `<th>`-Elemente den Text zentriert und fett darstellen, `<td>`-Elemente dagegen linksbündig und normal. Der Grundaufbau eines `<table>`-Elements ist also stets:

[2]Grob gesagt eignet sich JPG für Fotos oder Bilder mit vielen Farb- oder auch Grautönen, während PNG und GIF eher für Bilder mit wenigen Farbtönen, z.B. Logos, verwendet werden sollten.

```
<table>
  <tr>
    <th> ... </th>
     ...
  </tr>
  <tr>
    <td> ... </td>
     ...
  </tr>
</table>
```

Diese Verschachtelung muss unbedingt eingehalten werden. Die Anzahl der Zeilen ergeben sich aus der Anzahl der `<tr>`-Elemente, während die Spaltenanzahl sich aus der maximalen Anzahl von `<td>` und `<th>`-Elementen je Zeile ergibt. Die folgende Beispieltabelle hat also 3 Zeilen mit jeweils 2 Spalten, im Browser wird sie wie in Abb. 2.4 dargestellt.

Listing 2.3: Tabelle in HTML

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8"/>
    <title>Tabelle</title>
  </head>
  <body>
    <table style="margin:auto; width:500px" border="1">
      <tr>
        <th>Nummer</th>
        <th>Tipp</th>
      </tr>
      <tr>
        <td style="width:200px">Blumentip 1</td>
        <td>
          Vergessen Sie nie Ihre Blumen zu gießen!
        </td>
      </tr>
      <tr style="background:#FFC200">
        <td style="height:50pt">Blumentip 2</td>
        <td>Nachtschattengewächse im Dunkeln lagern</td>
      </tr>
    </table>
  </body>
</html>
```

Innerhalb des `<table>`-Tags können verschiedene Attribute zur Formatierung gesetzt werden. Drei häufig verwendete Attribute des `<table>` wurden im Quellcode bereits angewendet.

- `style="width:500px"` : Die Tabellenbreite wird fest auf 500 Pixel gesetzt
- `border="1"` : Die Gitterlinien der Tabelle werden mit 1 Pixel Dicke dargestellt. Mit der Anweisung `border="0"` erhält man somit eine blinde Tabelle, also eine ohne sichtbare Gitterlinien.

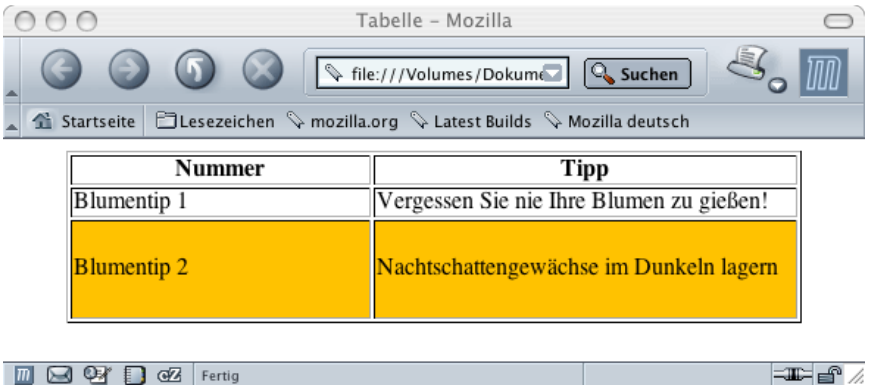


Abbildung 2.4: Die Ausgabe von Beispiel 2.3.

Die Standardvorgaben für die Tabelleneinträge sehen wie folgt aus:

- Die Breite der Spalten richtet sich nach der breitesten Tabellenzelle innerhalb dieser Spalte.
- Die Höhe richtet sich nach der höchsten Tabellenzelle innerhalb dieser Reihe.
- Die Ausrichtung des Zelleninhaltes ist linksbündig und vertikal zellenzentriert.

Entsprechend kann man – allerdings mit gewissen Beschränkungen – auch einzelne Zeilen oder Zellen formatieren, beispielsweise Breite, Höhe oder Hintergrundfarbe festlegen. Für die Darstellung von Zahlenspalten wird oft eine rechtsbündige Formatierung benötigt, die man mit CSS wie folgt erreicht:

```
    <td style="text-align:right"> 21,99 &euro;</td>
```

Weitere Hinweise zur Formatierung von Tabellen mit Hilfe von CSS siehe

`https://css-tricks.com/complete-guide-table-element/`

2.4.3 Spaltenformatierung mit `colgroup`

Mit dem `<colgroup>`-Element können die Spalten einer Tabelle einheitlich formatiert werden. Das Element kann in dem `<table>`-Element vor dem ersten `<tr>`-Tag erscheinen und legt mit `<col>-Tags` die Formatierungen für die einzelnen Spalten fest.

```
<table>
  <colgroup>
    <col style="background-color:red;">
    <col style="background-color:yellow;">
      ...
  </colgroup>
  <tr>
    ...
  </tr>
</table>
```

Dabei kann mit dem Attribut `span` die Anzahl der betroffenen aufeinanderfolgenden Spalten angegeben werden, wenn diese gleich zu formatieren sind. Beispielsweise werden in folgendem Beispiel die zweite und dritte Spalte gelb gefärbt, die erste rot:

```
<table>
  <colgroup>
    <col style="background-color:red">
    <col span="2" style="background-color:yellow">
  </colgroup>
  <tr>
    <th>ISBN</th>
    <th>Titel</th>
    <th>Preis</th>
  </tr>
  <tr>
    <td>987-3-4768-96</td>
    <td>My first HTML Table</td>
    <td>$53,--</td>
  </tr>
</table>
```

Weitere Informationen siehe `http://w3schools.com/tags/tag_colgroup.asp`.

2.5 Formulare

In HTML gibt als Benutzerschnittstelle zur Eingabe von Daten durch den Anwender die so genannten Formulare, das `<form>`-Element. Um ein Eingabeformular wie in Abbildung 2.5 anzeigen zu lassen, muss man in HTML den Quelltext 2.4 programmieren.

Listing 2.4: Ein Formular mit den wichtigsten Eingabeelementen

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8"/>
  <title>Formular</title>
</head>
<body>
  <h1>Eingabeformular</h1>
  <form action="http://haegar.fh-swf.de/spiegel.php" method="GET">
    <input type="text" name="anmeldename" placeholder="Anmeldename"/>
    <br/>
    <input type="password" name="passwort" placeholder="Passwort" />
    <br/>
    <input type="email" name="email" placeholder="Email"/>
    <p>
    Größe:
    <label><input type="radio" name="groesse" value="k"/>klein</label>
    <label><input type="radio" name="groesse" value="m"/>mittel</label>
    <label><input type="radio" name="groesse" value="g" checked/>groß</label>
    </p>
    <p>
    Blume:
    <label><input type="checkbox" name="blume[]" value="Rose"/>Rose</label>
    <label><input type="checkbox" name="blume[]" value="Tulpe" checked/>Tulpe</label>
```

```
    <label><input type="checkbox" name="blume[]" value="Gerbera"/>Gerbera</label>
    <label><input type="checkbox" name="blume[]" value="Lilie"/> Lilie</label>
    </p>
    <p>
    Farbe:
    <select name="farbe" size="1">
       <option value="rot"> rot </option>
       <option value="gelb" selected> gelb </option>
       <option value="blau"> blau </option>
    </select>
    </p>
    <p>
    <input type="hidden" name="kennung" value="verborgene Daten" />
    Frist: <input type="date" name="datum"/>
    Uhrzeit: <input type="time" name="zeit" step="900" value="12:00"/>
    </p>
    Beitrag: <br/>
    <textarea name="beitrag" rows="9" cols="45">
Da steh ich nun, ich armer Tor!
und bin so klug als wie zuvor;
heiße Magister, heiße Doktor gar,
und ziehe schon an die zehen Jahr
herauf, herab und quer und krumm,
meine Schüler an der Nase herum -
und sehe, dass wir nichts wissen können!
Das will mir schier das Herz verbrennen!
    </textarea>
    <hr/>
    <input type="submit" value="Abschicken"/>
    <input type="reset" value="zurücksetzen"/>
  </form>
</body>
</html>
```

Die meisten Eingabeelemente werden durch das Element `<input>` implementiert, dessen Attribut `type` über die Darstellung entscheidet. Allgemein sollte jedes Eingabeelement mit dem Attribut `name` einen Namen erhalten, der in dem Formular eindeutig ist. Für die Darstellung ist dieser Name zwar nicht wichtig, jedoch werden wir später sehen, dass dieses Attribut für die serverseitige Verarbeitung der Eingabedaten notwendig ist.

Der Standardtyp ist `type="text"` und zeigt ein Textfeld an.

```
<input type="text" name="abc" value="Vorgabetext"/>
```

Mit dem optionalen Attribut `value` kann hierbei ein Standardtext als Vorgabe angezeigt werden. Statt eines Wertes kann es auch einen Platzhaltertext beinhalten:

```
<input type="text" name="abc" placeholder="Platzhaltertext"/>
```

Er wird in dem Textfeld abgeschwächt angezeigt und verschwindet bei der ersten Eingabe. Er wird oft statt eines Textes vor dem Eingabefeld als Eingabehilfe verwendet.

Weitere Eingabeelemente sind durch die Typen `"checkbox"` und `"radio"` gegeben, die anklickbare Checkboxen bzw. Radiobuttons darstellen. Zusammengehörige Auswahloptionen gehören

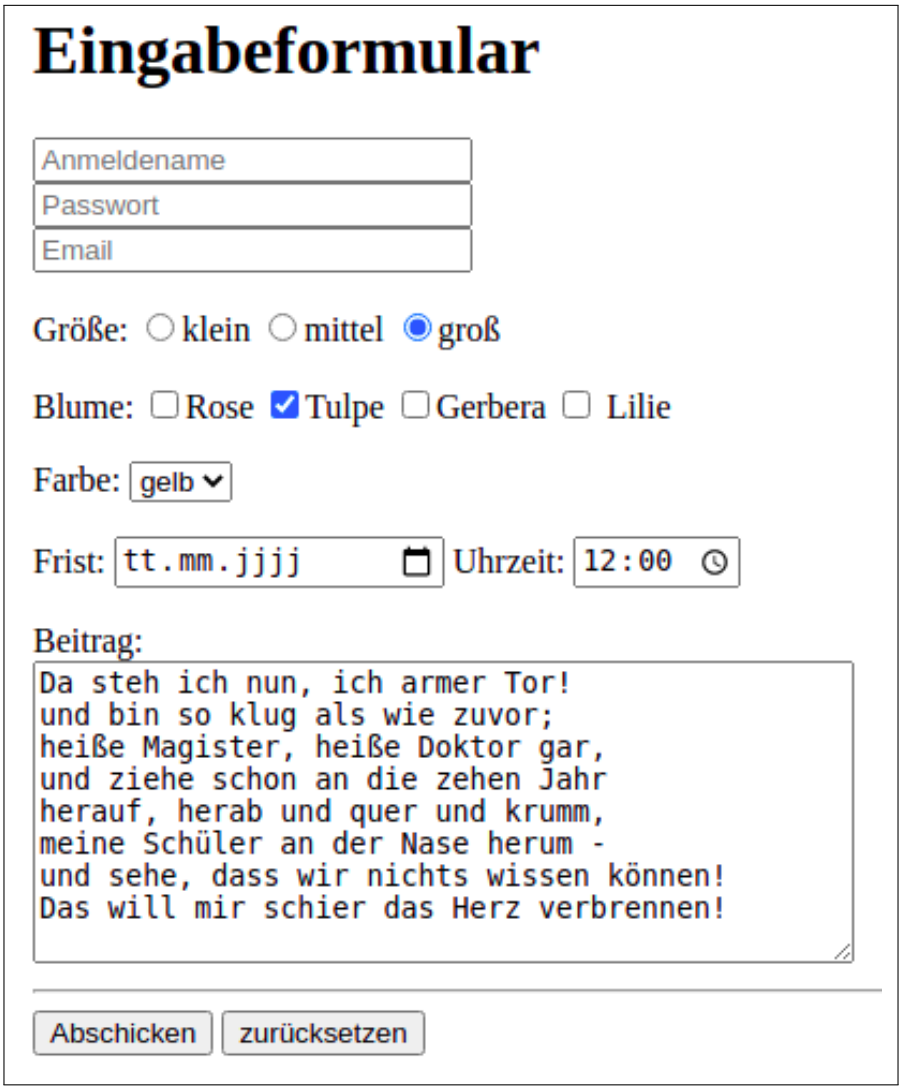


Abbildung 2.5: Die Ausgabe des HTML-Formulars aus Beispiel 2.4.

hierbei zu einer Gruppe und werden mit demselben Namen bezeichnet, erhalten jedoch mit dem Attribut `value` verschiedene Werte, die nicht mit den zu der jeweiligen Checkbox bzw. dem Radiobutton angezeigten Text übereinstimmen müssen. Der Unterschied der beiden Eingabeelemente ist, dass aus einer Gruppe von Radiobuttons nur höchstens ein Button angeklickt werden kann, während eine Checkbox eine Mehrfachauswahl ermöglicht. Um bei einer Mehrfachauswahl die korrekte Übertragung (und vor allem spätere Verarbeitung) zu ermöglichen, sollte der Name mit einem Paar eckiger Klammern enden:

```
<input type="checkbox" name="blume[]"/>
```

So werden die Daten in einem Array, also einer Art Datencontainer, übertragen und können vom Server „durchnummeriert“ werden (Arrays werden wir später im Zusammenhang mit PHP behandeln).

Um die Bedienerfreundlichkeit zu erhöhen ist übrigens empfohlen, insbesondere Checkboxen und Radiobuttons jeweils in `label`-Elemente zu umschließen, also beispielsweise:

```
<label> <input type="radio" name="groesse" value="k"/> klein </label>
```

Die Wirkung des `<label>`-Elements ist, dass der Anwender zur Auswahl nicht unbedingt den Button bzw. die Checkbox anklicken muss, sondern auch den in dem Element enthaltenen Text.

Neben den `<input>`-Tags gibt es noch die Auswahlbox

```
<select name="farbe">
  <option value="rot">rot</option>
  ...
</select>
```

die mit dem optionalen Attribut `size` die Anzahl der in dem Sichtfenster angezeigten Optionen vorgibt. Eine Option kann standardmäßig vorausgewählt angezeigt werden durch das Attribut `selected`. Ein längerer Text schließlich kann mitttels `<textarea>` eingegeben werden, dessen Höhe und Breite durch die Anzahl der Zeilen `rows` bzw. Spalten `cols` festgelegt.

Das spezielle `<input>`-Element mit `type="hidden"` wird *hidden Field* genannt und bezeichnet in einem Formular ein Datenfeld, das im Browser nicht angezeigt wird, dessen Wert aber wie alle anderen Eingabefelder an den Server übermittelt wird. Dazu muss es einen innerhalb des Formulars eindeutigen Namen und mit dem Attribut `value` einen Wert haben:

```
  <input type="hidden" name="kennung" value="verborgene Daten" />
```

Nach dem Abschicken wird aber das Feld `kennung` mit dem Wert `"verborgene Daten"` an den Server geschickt. Ein hidden Field wird also nicht im Browser angezeigt werden. (Allerdings sieht man es in der Quelltextanzeige der Seite im Browser.) Es ist daher eigentlich kein Eingabefeld. Wofür braucht man dann also ein solches Feld? Es wird sich als sehr praktisch erweisen, wenn wir später mit Datenbankanwendungen arbeiten.

Wichtig zu bemerken ist, dass niemals sicherheitsrelevante Informationen in einem hidden Field gespeichert werden dürfen, denn mit der Seitenquelltextfunktion des Browsers können sie immer angezeigt und mit den Entwickler-Tools verändert werden.[3]

Jedes `<input>`-Element kann als Attribut `required` erhalten. Damit wird bewirkt, dass der Browser das Formular nicht abschickt in dem Fall, dass das Eingabeelement nicht ausgefüllt bzw. ausgewählt wurde.

Das `action`-Attribut und der `submit`-Button

Wesentlich für die Übertragung der einegegebenen Daten zum Server sind das Attribut `action` im Formular und der Submit-Button

[3] vgl. `https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input/hidden`

```
<input type="submit"/>
```

Die Inhalte aller sonstigen `<input>`- und weiteren Eingabeelemente mit Namen (!) werden nach Anklicken des `submit`-Buttons an den URL im `action`-Attribut abgeschickt. Hier ist dies ein PHP-Programm `spiegel.php` auf dem Haegar-Server, dass einfach die empfangenen Daten in einer Tabelle wieder an den Browser zuücksendet.

2.5.1 Datenübertragung mit `GET` und `POST`

Wenn ein Browser eine Seite von einem Server anfordert, so wird die Kommunikation zwischen Browser und Server über das HTTP-Protokoll abgewickelt. Als Teil der Anfrage und damit auch des Protokolls überträgt der Browser eine „Methode“, mit der festgelegt wird, wie dem Server eventuelle Inhalte von Formularfeldern übermittelt werden. Standardmäßig ist diese Methode `GET`, also beispielsweise auch bei einem normalen Aufruf einer HTML-Seite nach einem Klick einen Hyperlink oder Eingabe eines URI Man kann in mit Hilfe der `GET`-Methode beliebige Daten als Attribut-Werte-Paare übermitteln, indem zunächst ein Fragezeichen an den URI gehängt wird und die Attribut-Werte-Paare (ohne Leer- und Anführungszeichen) und durch ein Kaufmanns-Und voneinander getrennt werden, beispielsweise:

```
http://haegar.fh-swf.de/spiegel.php?x=5&y=10.5&art=koordinate
```

Probieren Sie es aus!

Bei der HTML-Programmierung von Formularen legen wir über das `method`-Attribut des `<form>`-Tags fest, welche Methode beim Abschicken des Formulars genutzt werden soll. Hier gibt es zwei Möglichkeiten: `GET` und `POST`. Bei `POST` werden die Daten gewissermaßen dem Request angehängt und erscheinen nicht in der URI. Dementsprechend können per POST übermittelte Daten auch Binärdaten wie z.B. Dateien sein.

2.6 Zusammenfassung

In den zurückliegenden Seiten wurde ein Schnelleinstieg in HTML geliefert. Er soll dazu dienen, kurz diejenigen HTML-Elemente vorzustellen, die wir im Folgenden benötigen werden. Er kann keinen Anspruch auf Vollständigkeit stellen. Insbesondere das schwierige Thema der Gestaltung von Webseiten, vor allem unter dem Aspekt der Barrierefreiheit (Fußnote S. 23), kann im Rahmen dieses Skripts nicht vertieft werden. Auch können rechtliche Aspekte nicht angesprochen werden, die bei einem Webauftritt, insbesondere mit kommerziellen Absichten, zu beachten sind. Zur Vertiefung sei auf die selfhtml `http://de.sefhtml.org` empfohlen.

3
Cascading Stylesheets (CSS)

Kapitelübersicht

Cascading Stylesheets (deutsch: „geschachtelte Formatvorlagen“) dienen dazu, das Design und das Layout von HTML-Seiten getrennt von dem eigentlichen Inhalt zu programmieren. Das Design legt hierbei die Ästhetik einer Seite fest und ist häufig für einen vollständigen Webauftritt gültig, also „global“. Typischerweise bestimmt das Design die Farbgebung von Haupttext, Überschriften und Hintergrund. Das Layout einer Seite bezieht sich auf die Anordnung von Elementen, bestimmt also beispielsweise den Ort der Navigation.

3.1 Syntax: Deklarationen, Selektoren und CSS-Regeln

Den Kern von Formatierungsanweisungen in CSS bilden *Deklarationen*, die Attribut-Wert-Paare sind und gemäß der Syntax

```
eigenschaft1: wert1;
eigenschaft2: wert2;
```

aufgebaut sind. Die Deklarationen werden durch Semikolons voneinander getrennt. Die Formateigenschaften und die jeweils möglichen Werte sind hierbei vorgegeben. Eine gute Übersicht darüber findet man in dem Wiki von selfhtml.org:

`http://wiki.selfhtml.org/wiki/CSS/Eigenschaften`

Wie erfolgt nun genau die Verknüpfung zwischen den zu formatierenden HTML-Elementen und den definierten CSS-Deklarationen? Ein *Selektor* (oder einer Liste mehrerer durch Kommas getrennter Selektoren) bildet hierbei die Auswahlbedingung für diejenigen Elemente des HTML-Dokuments, für das die in geschweiften Klammern zu einem Block eingeschlossenen CSS-Regeln gelten sollen:

```
selektor1, selektor2 {
  eigenschaftA: wertA;
  eigenschaftB: wertB;
}
```

Das letzte Semikolon vor der schließenden geschweiften Klammer kann weggelassen werden. Ein solcher mit Selektoren versehener Block von CSS-Deklarationen heißt *CSS-Regel*. Wichtige Selektoren sind die folgenden:

- Ein *Typselektor*: wählt Elemente abhängig von einem angegebenen HTML-Elementtyp aus, also z.B. `<p>` (allerdings ohne die spitzen Klammern!).
- Ein *Klassenselektor* beginnt mit einem Punkt und seinem Namen dahinter (`.meineKlasse`) und wählt diejenigen Elemente des HTML-Dokuments aus, deren Attribut `class` diesen spezifizierten Wert haben, also beispielsweise

  ```
  <p class="meineKlasse"> ... </p>
  ```

- Ein *ID-Selektor* beginnt mit einem Doppelkreuz `#` und wählt das (eindeutige) Element eines Dokuments aus, das den Wert des ID-Attributes hat.

  ```
  <p id="otto"> ... </p>
  ```

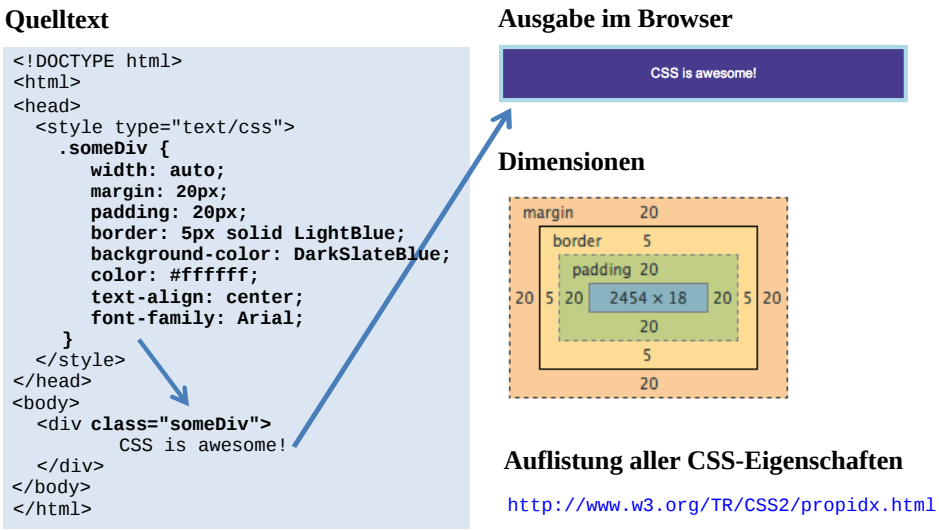


Abbildung 3.1: Funktionsweise eines Klassenselektors und der Begrenzungsregeln in CSS.

An beliebiger Stelle einer CSS-Regel können Kommentare eingefügt werden, z.B.:

```
/* Kommentare wie in C, Java, PHP, ...*/
selektor1, selektor2 {
  eigenschaftA: wertA;
  eigenschaftB: wertB;
}
```

Kommentare werden von `/*...*/` umschlossen.

Bereits hier erkennen wir, dass die Syntax von CSS völlig unterschiedlich zu derjenigen von HTML ist. Insbesondere wird in CSS kein Gleichheitszeichen verwandt, sondern der Doppelpunkt, und zwischen zwei Deklarationen muss stets ein Semikolon erscheinen, ganz im Gegensatz zu HTML. Wir haben also das grundsätzliche Problem, CSS-Anweisungen in ein HTML-Dokument „einzuschleusen“. Es gibt drei Arten der Einbindung, die im folgenden Abschnitt dargestellt werden.

3.2 Einbindung von CSS in HTML

CSS-Anweisungen müssen in ein HTML-Dokument eingebunden werden. Man unterscheidet drei Formen der Einbindung von CSS-Quelltext in ein HTML-Dokument:

- *Inline:* Bestehende HTML-Elemente werden um ein Attribut namens `style` erweitert, das CSS-Deklarationen für dieses Element enthält. Die Syntax dazu lautet (am Beispiel des `<div>`-Elements):

  ```
  <div style="eigenschaft1: wert1; eigenschaft2: wert2"> ... </div>
  ```

  Die Deklarationen dürfen hierbei nicht durch geschweifte Klammern zu einem Block zusammengefasst werden und dürfen keinen Selektor enthalten. Letzteres ist auch gar nicht nötig, das Style-Attribut betrifft nur das Element, in dem es definiert wird. Die Inline-Einbindung eignet sich für individuelle Adhoc-Formatierungen. Sehr nützlich ist hier das `<span>`-Tag, das in HTML grundsätzlich einen Textbereich `<span> ... </span>` markiert, ohne aber irgendwie sichtbar zu sein. Mit

  ```
  Dies ist ein <span style="color: rot; font-family:italic;">beliebiger</span>
  ```

  können so beispielsweise einzelne Wörter formatiert werden.

- *Embedded:* CSS-Regeln werden im `<head>`-Bereich eines HTML-Dokuments definiert.

  ```
  <head>
  <title>A title</title>
  <style>
  h1 {
    color: red;
  }
  </style>
  </head>
  <body>
    ...
  </body>
  ```

  Die CSS-Anweisungen gelten dann für das gesamte Dokument. Man kann das `<style>`-Element auch um das Attribut `type="text/css"` ergänzen, es ist jedoch für HTML5 (`<!DOCTYPE html>`) nicht mehr notwendig. Diese Einbindungsart eignet sich für generelle CSS-Regeln, die sich auf das gesamte Dokument beziehen.

- *Extern:* CSS-Regeln werden in einer externen Datei definiert und im `<head>`-Element als Link verknüpft.

  ```
  <head>
   <title>A title</title>
   <link rel="stylesheet" href=" ... Pfad zur CSS-Datei ..." />
  </head>
  ```

  Die CSS-Datei enthält ausschließlich CSS-Regeln mit Selektoren und Deklarationsblöcken sowie Kommentare. Diese Form der Einbindung eignet sich für generelle CSS-Regeln, die sich auf alle Dokumete des gesamten Webauftritts beziehen sollen.

Beispiel 3.1. Beispiele für die Arten der Einbindung von CSS-Anweisungen in ein HTML-Dokument.

(a) Inline

```
<h1 style="color:red;">FH SWF</h1>
```

(b) Eingebettet

```
<head>
<title>A title</title>
<style>
h1 {
  color: red;
}
</style>
</head>
<body>
  ...
</body>
```

(c) Externe Datei, z.B.:

```
<head>
 <title>A title</title>
 <link rel="stylesheet" href="./fhswf.css" />
</head>
```

mit der Datei `fhswf.css` im selben Verzeichnis:

```
  h1 {
    color:red;
  }
```

Die CSS-Anweisungen können dann in mehreren verschiedenen HTML-Dokumenten verwendet werden. □

Bei Inline-Styles ist nur das HTML-Element verknüpft, dessen `<style>`-Attribut es belegt. Eingebettete und externe Definitionen müssen mit Selektoren arbeiten.

3.3 Die Kaskadierung geschachtelter Regeln

Aufgrund der verschiedenen Einbindungsmöglichkeiten und der Verschachtelungen von formatierten HTML-Elementen können mehrere CSS-Regeln zur selben Formateigenschaft in einem bestimmten HTML-Element vorhanden sein. CSS verspricht durch das „C“ in seinem Namen eine Kaskadierung von Formatanweisungen. Wie genau sieht diese Kaskadierung der Regeln aus? Welcher Stil wird denn nun angewendet bzw. welche Farbe hat der Text in der Darstellung? Die Kaskadierung in CSS richtet sich hauptsächlich nach der direkten Verarbeitungsreihenfolge des Browsers, also nach den folgenden Prioritäten:

1. Die CSS-Regel des innersten formatierten HTML-Elements hat die höchste Priorität.
2. Die Priorität hängt dann ab von der Einbindungsart der CSS-Regel,

$$\text{inline} > \text{embedded} > \text{extern}.$$

(Also „inline“ gewinnt immer!)

3. Die Priorität hängt bei eingebetteten oder der externen Einbindung ab vom Selektor, gemäß

ID-Selektor > Klassenselektor > Typselektor.

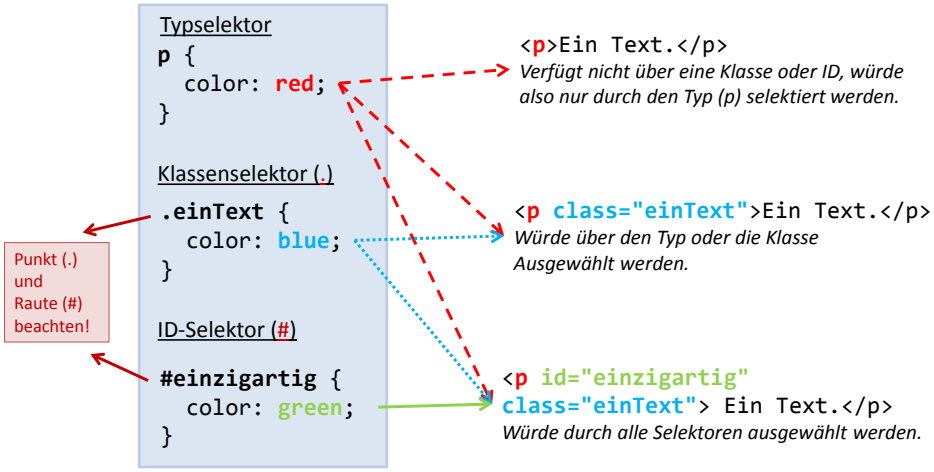


Abbildung 3.2: Selektoren in CSS. Welche Farbe hat der Text jeweils?

Diese Prioritätenregeln („Spezifitäten“) können explizit außer Kraft gesetzt werden, und zwar mit den `!important`-Anweisungen, siehe `http://www.w3.org/TR/CSS2/cascade.html#specifity`.

3.4 Media Queries

Webseiten müssen auf mobile Endgeräte reagieren („Responsive Webdesign“). In der Version 3 bietet CSS hierfür die sogenannten *Media Queries* an,

`http://www.w3.org/TR/#tr_CSS`.

Sie ermöglichen die Erkennung verschiedener Geräte und Geräteklassen anhand von deren Auflösung und Orientierung. Die Syntax für Media Queries ist wie folgt:

```
@media ( ... Bedingung ... ) {
    { ... CSS-Regeln ... }
}
```

Hinter dem Schlüsselwort `@media` folgt in runden Klammern eine Bedingung, die bei Erfüllung den nachfolgenden Block von CSS-Regeln gültig werden lässt.

Betrachten wir dazu das folgende Beispiel.

```
<!DOCTYPE html>
<html>
<head>
<style type="text/css">
  @media (max-width: 480px) {
    table {
      display: none;
```

```
    }
  }
</style>
</head>
<body>
  <table>
    <tr>
      <td>1</td>
      <td>2</td>
    </tr>
  </table>
</body>
</html>
```

Hierbei leitet `@media` ein Media Query ein, das in Alltagssprache ausformuliert etwa lauten würde: „Blende auf Geräten mit einer maximalen Auflösung von 480 Pixeln alle Elemente vom Typ `table` aus!“ Ist also der Bildschirm klein genug, greift die CSS-Deklaration `display: none`, die die Tabelle verschwinden lässt.

Mehrere Bedingungen können in einem Media Query mit dem Schlüsselwort `and` verknüpft werden.

```
<!DOCTYPE html>
<html>
<head>
  <title>Aktion</title>
  <style>
    @media (min-width: 800px) {
      div {
        background-color: green; color: white;
      }
    }
    @media (min-width: 600px) and (max-width: 800px) {
      div {
        background-color: yellow;
      }
    }
    @media (max-width: 600px) {
      div {
        background-color: #FF0000;
      }
    }
  </style>
</head>
<body>
  <div>
    Da steh ich nun, ich armer Tor
  </div>
  <div>
    Und bin so klug als wie zuvor.
  </div>
</body>
```

```
</html>
```

Bei diesem Media Query wechseln die `<div>`-Elemente ihre Hintergrundfarbe bei Veränderung der Fensterbreite, und zwar für die drei Fälle

$$800\text{ px} < \text{Breite}, \qquad 600\text{ px} < \text{Breite} < 800\text{ px}, \qquad \text{Breite} < 600\text{ px},$$

3.5 Weitere Informationen und Quellen

- `https://www.w3.org/standards/techs/css#w3c_all` – Die aktuellen Standards, formal und trocken; wenig Beispiele
- `https://wiki.selfhtml.org/wiki/CSS` – deutschsprachige Erklärungen und gute Beispiele
- `https://developer.mozilla.org/en-US/docs/Web/CSS` – englischsprachige Beispiele und Tutorien (teilweise auf Deutsch übersetzt)
- `https://css-tricks.com/snippets/css/` – viele Beispiele, auch ausgefallene

4

Formate zum Datenaustausch: XML & JSON

Kapitelübersicht

Das *World Wide Web Consortium* [W3C] ist ein 1994 gegründetes Gremium zur Standardisierung von Web-Techniken. Ein solcher durch das W3C festgelegter Standard zur Definition und zum allgemeinen Datenaustausch ist XML. Es ist gewissermaßen eine Obermenge zu HTML, oder anders ausgedrückt: HTML ist eine spezielle Instanz von XML. Obwohl XML seit den 2000er Jahren sich in seiner Allgemeingültigkeit zu einem weitverbreitetem Format zum textbasierten Datenaustausch entwickelt hat, gibt es andere Formate, die sich erfolgreich etablieren konnten. Die wichtigsten darunter sind JSON als ein weiteres textbasiertes Format, aber auch binäre Formate wie BSON oder CBOR.

4.1 XML

XML ist ein Akronym für *eXtensible Markup Language*, also eine erweiterbare Auszeichnungssprache.[1] XML ist eine Metasprache, d.h. es lassen sich mit XML wiederum Sprachen für bestimmte spezielle Zwecke definieren[2], beispielsweise XHTML (eine „strenge Variante“ von HTML). Obwohl XML in 1998 zunächst nur für elektronische Veröffentlichungen von Dokumenten konzipiert war, etablierte es sich zunehmend auch als allgemeines Datenaustauschformat. Die Sprachspezifikationen werden vom World Wide Web Consortium [W3C] festgelegt.

[1] Eine Auszeichnungssprache ist eine formale Sprache, die Teile eines Textes (meist ASCII oder Unicode) durch Markierungen beschreibt, so dass sie auf bestimmte Weise dargestellt werden können. Beispiele für Auszeichnungssprachen sind HTML, LaTeX oder Wikitext (der bei Wikipedia verwendet wird).

[2] Hauser (2006):S. 9.

4.1.1 Die Idee

Bevor wir auf die Spezifikationen genauer eingehen, ist es hilfreich, sich die grundlegende Idee von XML zu verdeutlichen. Da XML mit dem Anspruch entwickelt wurde, eine Rahmensprache zur Speicherung und Veröffentlichung von *Dokumenten* zu sein, muss zunächst einmal die Frage beantwortet werden: Woraus besteht eigentlich ein Dokument? Was muss demnach alles gespeichert werden?

Ein allgemeines Dokument kann beispielsweise ein Formular, ein Vertrag, ein Brief, eine Zeitung oder ein Buch sein. Ein Dokument besitzt immer einen *Inhalt*, beispielsweise als Text

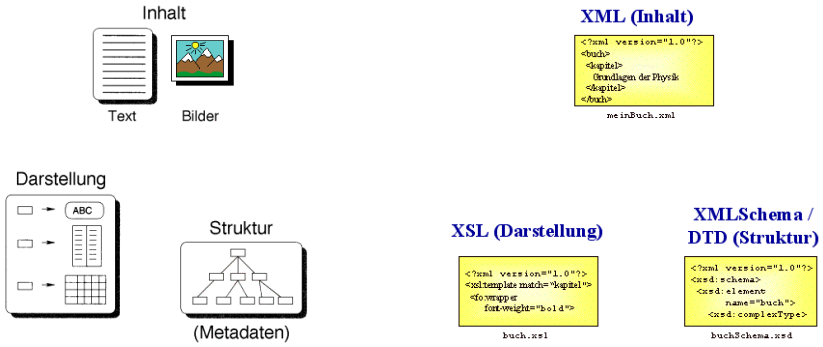


Abbildung 4.1: Die logischen Bestandteile eines Dokuments und das Prinzip von XML.

oder als Bilder, eine innere *Struktur*, wie Fließtexte, Paragraphen, Kapitel, und eine konkrete Form der *Darstellung*, z.B. Formatierungen, Wahl des Mediums wie Papier oder Karton. Diese drei Strukturbausteine sind in Abb. 4.1 links illustriert. Entsprechend trennt XML als Technologie prinzipiell den Inhalt, die Form und die Struktur eines Dokuments. Hierbei wird das konkrete Dokument mit seinem Inhalt als ein XML-Dokument gespeichert, die Struktur in einer XML-Schema-Datei (XSD, früher oft auch in einer DTD) und die Art der Darstellung in einem XSL-Programm, das die Formatierungen konkret darstellt (z.B. ein Browser, der XHTML-Dokumente im Fenster anzeigt), oder ein XSLT-Programm, das das XML-Dokument in ein spezielles Format (z.B. PDF) umformt.

Beispiel 4.1. Ein Beispiel für eine XML-Sprache ist XHTML, eine Art „strenges HTML“, nämlich HTML gemäß den noch zu nennenenden XML-Regeln. (Historisch entstand HTML vor XML.) Die Struktur XHTML ist in einer DTD-Datei, d.h. einer Strukturdatei, festgelegt. Mit dieser Struktur ist definiert, welche Tags überhaupt und in welchen Verschachtelungen erlaubt sind. □

4.1.2 Elemente und Tags, Inhalt und Attribute

Allgemein besteht ein XML-Dokument aus Unicode-Text. Die einzigen allgemein in XML reservierten Zeichen mit bestimmter Bedeutung sind die spitzen Klammern, das Kaufmanns-Und *(ampersand)*, das Semikolon, das Gleichheitszeichen, die Anführungszeichen oben und der Apostroph:

Zeichen	<	>	&	;	=	"	'
Unicode	&#x3C;	&#x3E;	&#x26;	&#x3B;	&#x3D;	&#x22;	&#x27;
XML-Entität	&lt;	&gt;	&amp;	&#x3B;	&#x3D;	&quot;	&apos;

(4.1)

Allerdings haben diese Zeichen nur in bestimmten Kombinationen eine Bedeutung, beispielsweise hat das Semikolon nur nach einem Kaufmanns-Und Auswirkungen, oder das Gleichheits-

zeichen und die Anführungszeichen nur innerhalb von spitzen Klammern. Die ausführlichen Syntaxregeln für die Grundbausteine finden Sie zum Beispiel in[3].

Die Bausteine von XML-Dokumenten sind *Elemente*. Sie werden von *Tags* (Auszeichnungen) umschlossen. Tags tauchen also stets paarweise auf, nämlich als Start-Tag und als Ende-Tag. Man schreibt dabei das Start-Tag, indem man den Namen des Elements in spitzen Klammern schreibt, und das Ende-Tag zusätzlich mit einem Schrägstrich *(slash)* versieht, also etwa

<*name*> . . . </*name*>

Das Innere „. . . “ des Elements zwischen den Tags kann einen beliebigen Unicode-Text, den *Inhalt* des Elements, oder wiederum Elemente enthalten.

Falls ein Element kein Kindelement und keinen Inhalt besitzt, heißt es *leer* und kann kurz geschrieben werden als <*name*/>, d.h.

<*name*/> = <*name*></*name*>.

Ein Element kann mehrere *Attribute* enthalten. Ein Attribut ist eine bestimmte Eigenschaft des Elements und kann einen Wert aus einer vorgegebenen Menge annehmen. Es wird in das Start-Tag des Elements geschrieben und sein Wert nach einem Gleichheitszeichen dahinter in Anführungszeichen:

<*name attribut*="*wert*"> . . . </*name*>

Eine *Prozessinstruktion (processing instruction, PI)* oder *Verarbeitungsanweisung* ist in XML ein spezielles Tag, das mit einem Fragezeichen beginnt und direkt dahinter einen Namen aufweist,

<?*name* . . . ?>

siehe[4]. Beispielsweise ist `<?php ... ?>` eine solche Prozessinstruktion, die wir im Zusammenhang mit PHP noch kennen lernen werden. Zu Beginn eines XML-Dokuments muss eine spezielle Prozessintruktion stehen, die XML-Deklaration:

```
<?xml version="1.0"?>
```

Sie gibt an, welche XML-Version verwendet wird. Aktuell gibt es nur die zwei Versionen 1.0 und 1.1, allerdings verstehen die meisten Browser verlässlich nur Version 1.0.[5] Daneben kann es ein Attribut `encoding` geben, das die Textkodierung festlegt, sowie ein Attribut `standalone`, das anzeigt, ob das vorliegende Dokument keine externe Strukturdatei (XML Schema, DTD, . . .) benötigt,

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
```

Das Attribut für die Version ist obligatorisch, die anderen beiden sind optional. Sind sie jedoch vorhanden, so ist die Reihenfolge `version` → `encoding` → `standalone` vorgeschrieben.

Ferner können Kommentare an beliebiger Stelle nach dem Vorspann (s.u.) in ein XML-Dokument eingefügt werden, und zwar wie in HTML mit der Syntax

```
<!- ... ->
```

Ein Kommentar ist also genau wie eine Prozessinstruktion nur ein einzelner Tag, kein Element. Beides kann und darf daher nicht wie eine leeres Element geschlossen werden.

[3] St. Laurent und Fitzgerald (2006):S. 12.

[4] St. Laurent und Fitzgerald (2006):S. 28.

[5] `https://www.w3.org/XML/Core/`. Der wesentliche Unterschied zwischen den Versionen 1.0 und 1.1 ist dass 1.1 stets automatisch die aktuellste Version des Unicode erlaubt.

4.1.3 Wohlgeformte XML-Dokumente

Ein *wohlgeformtes (well-formed)* XML-Dokument hat folgende Kriterien zu erfüllen:

- Es enthält genau ein Wurzelelement, das *Dokumentelement*, dessen Start- und Ende-Tag das gesamte Dokument (bis auf die Prozessinstruktion am Anfang) umschließt.

- Jedes Element muss geschlossen werden, auch wenn es leer ist. In HTML übliche Konstrukte wie `<p> bla <p> blubb`, `bla <br> blubb`, `<input ...>` oder `<img ...>` sind damit *kein* wohlgeformtes XML.

- Elemente dürfen nur streng ineinander verschachtelt werden. Das impliziert, dass ein inneres Element erst geschlossen sein muss, bevor ein äußeres geschlossen werden kann.

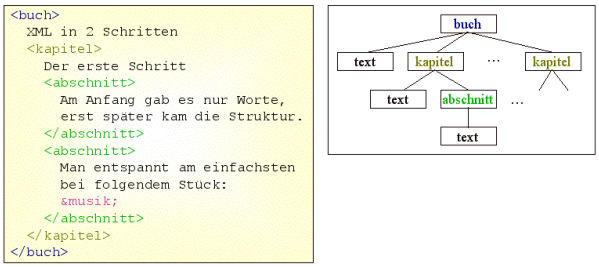


Abbildung 4.2: Die logische Baumstruktur eines XML-Dokuments.

  Auf diese Weise entsteht eine streng logische Baumstruktur ineinander geschachtelter Elemente wie in Abb. 4.2.

- Groß- und Kleinschreibung für Element- und Attributnamen muss streng beachtet werden. `<p> ... </P>` ist also *nicht* wohlgeformtes XML.

- Attributwerte müssen in Anführungszeichen stehen. `<img border=0/>` ist also *nicht* wohlgeformt.

- Ein Attributname darf nur einmal im Start-Tag vorkommen.

- Die in (4.1) aufgelisteten Sonderzeichen dürfen nicht innerhalb von Element- oder Attributnamen auftauchen, die Zeichen `<`, `"` und `&` nicht einmal in Attributwerten. Auch in Elementinhalten darf `<` gar nicht und `&` nur in definierten Fällen in Kombination mit einem Semikolon (also Begrenzer von „Entitäten“) erscheinen. Man kann beliebige Zeichenfolgen (außer `]]>`) in CDATA-Sektionen schreiben (CDATA steht für *Character Data*):

```
<![CDATA[ <hier kann <alles & stehen>!]]>
```
(4.2)

  als Elementinhalt wäre also wohlgeformtes XML. Alternativ können die Zeichen mit den aus HTML bekannten Entitäten `<`, `>` und `&` dargestellt werden oder durch ihre Unicode-Werte `&#x3C;`, `&#x3E;` und `&#x26;` gemäß Tabelle (4.1).

Die Wohlgeformtheit eines XML-Dokuments können schon Browser wie Firefox und Internet Explorer überprüfen. (Probieren Sie es mal aus!) Das XML-Dokument in Beispiel 4.1 ist wohlgeformtes XML.

Listing 4.1: Das XML-Dokument `Buch.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<buch>
  XML in zwei Schritten.
  <kapitel>
    Der erste Schritt
    <abschnitt ausrichtung="Blocksatz">
      Am Anfang gab es nur Worte, erst später
      kam die Struktur.
    </abschnitt>
    <abschnitt>
      Die meisten entspannen bei Musik, nur wenige auch bei mathematischen
      Ungleichungen der Form x + 1 &#x3C; x*x.
    </abschnitt>
  </kapitel>
</buch>
```

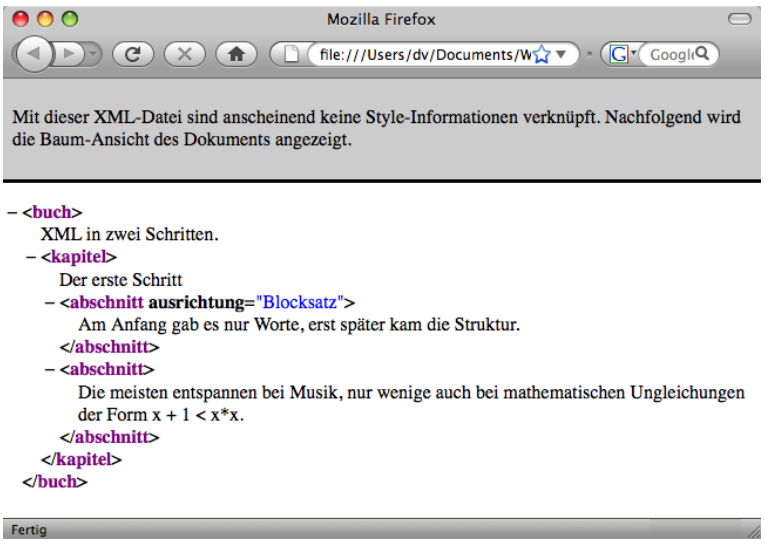


Abbildung 4.3: Die Ausgabe von Beispiel 4.1 im Firefox.

Generell können Element- oder Attributnamen in XML jedes Zeichen des spezifizierten Zeichensatzes enthalten, bis auf die Sonderzeichen von (4.1). Wohlgeformt gemäß XML wären also die Elemente `<tele.fon nummer="0123/>` oder `<e-mail>xyz@abc.de</e-mail>`.

4.1.4 Gültige XML-Dokumente

Oft ist ein XML-Dokument schon einsetzbar, wenn es wohlgeformt ist, beispielsweise für den Datenaustausch zwischen Datenbanken. Die Wohlgeformtheit sagt jedoch noch nichts über die Frage aus: Welche Elemente und Attribute sind überhaupt möglich, und in welcher Kombination? Ein XML-Dokument heißt *gültig (valid)*, wenn seine Elemente der in einer (meist separat gespeicherten) Strukturdatei festgelegten Elementenstruktur entsprechen. In einer Strukturdatei wird genau beschrieben, welche Elemente in der XML-Sprache überhaupt existieren, welche Attribute sie haben können oder müssen, und wie die Elemente verschachtelt sein dürfen.

Für XML gibt es mehrere Strukturen. Die älteste ist die DTD *Document Type Definition)*, sie ist die Strukturen für XHTML. Da DTD selbst allerdings keine XML-Sprache ist,

entwickelte das W3C als Alternative XML Schema. In einer DTD sind die beiden grundlegenden Datentypen Strings (in DTD-Syntax: `#PCDATA`, für *parsed character data*) und Notationen (`<!NOTATION ...>`), die nicht verarbeitet (geparst) werden sollen und als Referenz auf externe Dateien dienen. Eine detaillierte Beschreibung von DTD und ihrer Syntax findet man in z.B.[6] oder[7].

Namensräume

Das Konzept der *Namensräume (name spaces)* gibt es in vielen Programmiersprachen. In Java ist es beispielsweise über Pakete implementiert. Generell ist der Sinn von Namensräumen, Datei- oder Objektnamen eindeutig referenzierbar zu halten bei gleichzeitig größtmöglicher Freiheit bei der Namensvergabe eigen erstellter Dateien oder Objekte. So gibt es beipielsweise in Java zwei Klassen `Date`, die allerdings in verschiedenen Paketen liegen (`java.util` bzw. `java.sql`).

Ohne Namensräume müsste ein Programmierer also jedesmal überprüfen, ob der Name, den er vergeben möchte, bereits belegt ist, um gegebenenfalls einen anderen Namen wählen zu müssen. Damit würde natürlich ein wichtiger Aspekt der Namensvergabe, nämlich „sprechende Namen“ zu verwenden, insbesondere für große Systeme erheblich ausgehöhlt. *Innerhalb* eines Namensraums müssen die Namen selbstverständlich unterschiedlich sein.

Auch in XML gibt es Namensräume. Ein Namensraum in XML ist (meist) ein allgemeiner URI.[8] Das spezielle Attribut `xmlns` spezifiziert eine Namensraum-Deklaration in XML. Es gibt zwei Möglichkeiten, in XML Namensräume zu definieren: Entweder mit der *Standardnamensraum-Deklaration*

```
<unternehmen id="4711" xmlns="http://example.org/">
  <name>XYZ GmbH</name>
  <telefon>
    <anlage>Siemens</anlage>
    <technologie>Voice Over IP</technologie>
  </telefon>
</unternehmen>
```

für die der Namensraum in allen Unterelementen gilt, oder mit der *qualifizierten Namensraum-Deklaration*,

```
<it:unternehmen id="4711" xmlns:it="http://example.org/">
  <it:name>XYZ GmbH</it:name>
  <it:telefon>
    <it:anlage>Siemens</it:anlage>
    <it:technologie>Voice Over IP</it:technologie>
  </it:telefon>
</it:unternehmen>
```

mit einem Präfix (hier: `it`), durch das in jedem Unterelement gezielt ein Namensraum ausgewählt werden kann. Qualifizierte Namensräume werden insbesondere dann unabdingbar, wenn in einem XML-Dokument mehrere Namensräume benötigt werden.

---

[6] Hauser (2006):S. 30ff.

[7] St. Laurent und Fitzgerald (2006):S. 38ff.

[8] *URI (Uniform Resource Identifier)*: ein nach dem Prinzip *Schema: schemaspezifischer Teil* aufgebauter Identifikator einer abstrakten oder physischen Ressource wie Webseiten, Dateien, Webservices, E-Mail-Empfängern.[RFC3986] Beispielsweise ist *http://haegar.fh-swf.de* ein URI, oder *doi:10.1002/piuz.200401040*. Spezielle URIs sind *URL (Unified Resource Locators)*, also Internetquellen. Allerdings wird die Bezeichnung URL von dem W3C als *deprecated* eingestuft [RFC3305, §2.2].

XML Schema

XML Schema, oft auch als XSD bezeichnet, stammt ursprünglich von Microsoft und wurde 2001 zum W3C-Standard [XSD]. Im Unterschied zu DTD basiert XML-Schema komplett auf XML und hat wesentlich mehr Datentypen als DTD. Allerdings sind komplexe Strukturen in einer DTD erheblich kürzer beschreibbar.

Beispiel 4.2. Das XML Schema *Buch.xsd*:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://example.org/Buch" xmlns="http://example.org/Buch" >

 <xs:element name="buch" type="buchTyp" />

 <xs:complexType name="buchTyp" mixed="true">
  <xs:sequence>
   <xs:element name="kapitel" type="kapitelTyp"
               minOccurs="0" maxOccurs="unbounded" />
  </xs:sequence>
 </xs:complexType>

 <xs:complexType name="kapitelTyp" mixed="true">
  <xs:sequence>
   <xs:element name="abschnitt" type="abschnittTyp"
               minOccurs="0" maxOccurs="unbounded" />
  </xs:sequence>
 </xs:complexType>

 <xs:complexType name="abschnittTyp" mixed="true">
  <xs:attribute name="ausrichtung">
   <xs:simpleType>
    <xs:restriction base="xs:string">
     <xs:enumeration value="Blocksatz"/>
     <xs:enumeration value="linksbündig"/>
     <xs:enumeration value="zentriert"/>
    </xs:restriction>
   </xs:simpleType>
  </xs:attribute>
 </xs:complexType>

</xs:schema>
```

und ein entsprechendes XML-Dokument:

```
<?xml version="1.0" encoding="UTF-8"?>
<p:buch xmlns:p="http://www.example.org/Buch"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://www.example.org/Buch Buch.xsd ">
  XML in zwei Schritten.
  <kapitel>
    Der erste Schritt
```

```
    <abschnitt ausrichtung="Blocksatz">
        Anfang gab es nur Worte, erst später
        kam die Struktur.
    </abschnitt>
    <abschnitt>
        Die meisten entspannen bei Musik, nur wenige auch bei mathematischen
        Ungleichungen der Form x + 1 &#x3C; x*x.
    </abschnitt>
  </kapitel>
</p:buch>
```

Die Verschachtelung der Elemente des XML-Dokuments ist in dem XML-Schema vordefiniert. □

Ein XML-Schema besteht aus einem `schema`-Element als Wurzel, mit einem fest vorgegebenem Namensraum (im Beispiel mit `xs` bezeichnet). Es enthält im Wesentlichen die Elemente `element`, die wiederum als obligatorisches Attribut `name` besitzen. Mit diesen Elementen definiert man genau die Elemente, die in einem XML-Dokument erlaubt sind.

Jedes Element muss von einem angegebenen Datentyp sein. Diesen Datentyp kann man anonym direkt anlegen, indem man einfach die beinhalteten Elemente als Kindelemente einbaut, oder aber, indem man die *Datentypen* als Elemente definiert. Es gibt eine ganze Reihe elementarer Datentypen in XML-Schema, die wichtigsten sind:

- *boolean* mit den mögliche Werten `"true"` (oder `"1"`), und `"false"` (oder `"0"`);
- *double* gemäß IEEE 754;
- *hexBinary* für binäre Daten mit Wörtern aus den Zeichen 0–9, A–F, a–f;
- *int* für Ganzzahlwerte mit 4 Byte;
- *integer* für beliebige ganze Zahlen (ohne Speicherbegrenzung);
- *string* für alle Unicode-Wörter ohne die Zeichen `<`, `>` und `&`.

Eine vollständige Liste aller in XML-Schema vorgesehenen Datentypen befindet sich in[9].

Daneben kann man mit XML-Schema komplexe Datentypen definieren, nämlich durch das Element *complexType*. Es beinhaltet weitere Elemente, allerdings benötigt es als direktes Kindelement (u.a.) *choice* (für eine beliebige Reihenfolge der umschlossenen Elemente) oder *sequence* (für eine durch die Reihenfolge ihres Auftretens festgelegte Reihenfolge der umschlossenen Elemente). Ein komplexer Typ, der neben inneren Elementen auch Text als Inhalt enthalten kann, muss das Attribut *mixed* mit dem Wert `"true"` besitzen. Jedes so definierte Element muss genau einmal im XML-Dokument auftauchen, es sei denn, die beiden Attribute *minOccurs* and *maxOccurs* sind mit Werten `"0"`, `"1"`, `...` oder `"unbounded"` angegeben (voreingestellter Wert jeweils: `"1"`). Natürlich muss $minOccurs \leqq maxOccurs$ gelten.

Ferner werden die möglichen Attribute eines Elements definiert durch das Element *attribute*, und zwar innerhalb des betreffenden Elements als Kindelemente. Es enthält als Attribute zwingend *name* und optional *use*, das vorbelegt den Wert `"false"` hat und bei `"required"` als Attribut im XML-Dolument auftauchen muss. Soll ein Attribut nur bestimmte Werte annehmen, so kann man das durch das Kindelement *simpleType*, dann *restriction* und schließlich *enumeration* erreichen:

[9] St. Laurent und Fitzgerald (2006):S. 91ff.

```
<xs:simpleType>
   <xs:restriction base="xs:string">
      <xs:enumeration value="Blocksatz"/>
      <xs:enumeration value="linksbündig"/>
      <xs:enumeration value="zentriert"/>
   </xs:restriction>
</xs:simpleType>
```

Hierbei legt das Attribut *base* im *restriction*-Element den Datentyp der Grundmenge fest, und das Attribut *value* in *enumeration* definiert jeweils einen Wert der Grundmenge.

Namensräume in XSD. Der Umgang mit Namensräumen in XSD kann beliebig kompliziert werden, insbesondere wenn man nichtqualifizierte Elemente und gleichzeitig qualifizierte Attribute verwendet. Eine Grundregel, die sich erfahrungsgemäß bewährt hat, lautet:

Merkregel. Arbeitet man mit Namensräumen in einem XML-Schema, so sollte man die Elemente qualifizieren, nicht aber die Attribute.

Einbindung von XSD in ein XML-Dokument Eine XML-Schema-Datei wird als Instanz in das Wurzelelement eines XML-Dokuments eingebunden.

Validierung eines XML-Schema-Dokuments

Die meisten XML-Editoren haben eingebaute Validierungsmöglichkeiten, um ein XML-Dokument gegen ein XML-Schema oder eine DTD zu validieren. Daneben gibt es jedoch auch einen Online-Validator vom W3C, `http://www.w3c.org/2001/03/webdata/xsv/`

4.2 JSON

JSON, kurz für *JavaScript Object Notation* und gesprochen wie der englische Name Jason, ist ein schlankes textbasiertes Format zum Datenaustausch zwischen Software-Anwendungen und zum RPC (*Remote Procedure Call*, d.h. Aufruf von Methoden auf anderen Rechnern). Wie XML verwendet JSON Unicode.

Das JSON-Format ist spezifiziert in RFC7159[10], der offizielle Internet Media Type (früher „MIME“) für JSON ist `application/json`. JSON kennt als Attributnamen Strings (in Anführungszeichen!) und als Werte Strings (in Anführungszeichen), Zahlen und die Konstanten `null`, `true` und `false`. Diese Elemente können auf zwei Arten als Datenstruktur zusammengefasst werden, einerseits als (numerisch indiziertes) Array

```
["Hallo", 123, null, false, true, "John Doe"]
```

in eckigen Klammern, oder als Attribut-Werte-Paare (assoziatives Array, Map)

```
{"key":"Abc", "key2":123, "key3":null, "flag":true}
```

in geschweiften Klammern. Üblicherweise werden einfache Listen in JSON durch ein Array dargestellt, also zum Beispiel die Liste der ersten 5 Primzahlen als

```
[2, 3, 5, 7, 11, 17]
```

[10] `http://tools.ietf.org/html/rfc7159`

Vornehmlich wird JSON in AJAX-Webapplikationen eingesetzt als Alternative zu XML, allerdings gibt es eine ganze Reihe von Programmen, die JSON in Objekte anderer Programmiersprachen wie Java transformieren und umgekehrt, vgl. `http://json.org`, so beispielsweise eine Java-API von Google[11] Ein einfaches Beispiel, welches eine Struktur in JSON mit der entsprechenden XML-Struktur vergleicht, ist in Tabelle 4.1 angegeben. Hierbei wurde die Liste der

Besucher in XML

```
<besucher>
  <id>481048</id>
  <name>Peter Müller</name>
  <email>pm@example.com</email>
  <gebuchteKurse>5</gebuchteKurse>
  <gebuchteKurse>21</gebuchteKurse>
  <gebuchteKurse>22</gebuchteKurse>
  <gebuchteKurse>23</gebuchteKurse>
  <gebuchteKurse>40</gebuchteKurse>
  <gebuchteKurse>44</gebuchteKurse>
</besucher>
```

Besucher in JSON

```
{
  "id" : 481048,
  "name" : "Peter Müller",
  "email" : "pm@example.com",
  "gebuchteKurse" : [
    5,
    21,
    22,
    23,
    40,
    44
  ]
}
```

Tabelle 4.1: Beispielstruktur *Besucher* in XML und JSON

gebuchten Kurse in JSON kurz als ein Array implementiert.

Es ist sofort ersichtlich, dass Dateien in JSON weniger Speicherplatz einnehmen. Ein wichtiger Unterschied zwischen den beiden Formaten XML und JSON ist ferner, dass XML-Dateien über die Forderung nach Gültigkeit eines Dokuments erzwingen kann, dass sie einem festen vorgegeben Schema genügen müssen. Für bestimmte Anwendungsfälle, z.B. Speicherung strukturierter Daten mit häufigen Suchzugriffen, ist diese Eigenschaft vorteilhaft. Oft sind auch, gerade in der Speicherung von Massendaten und großen Netz-Communities, dynamische schemalose Speicherstrukturen gefragt, für die JSON besser geeignet ist.[12]

4.3 * Binäre Austauschformate

Textbasierte Austauschformate wie XML oder JSON haben neben ihren Vorteilen der Lesbarkeit und einfachen Erstellbarkeit einen großen Nachteil: Sie verbrauchen oft mehr Speicherplatz als ihr eigentlicher Informationsgehalt benötigt. Beispielsweise erfordert der 20-stellige String

```
"18446744073709551615"
```

in UTF-8 kodiert 20 Bytes. Als vorzeichenloser Integer allerdings erfordert diese Zahl nur 8 Bytes, denn

$$18446744073709551615 = \texttt{0xffffffffffffffff} = 2^{64} - 1 \tag{4.3}$$

hat in Hexadezimaldarstellung 16 Stellen à 4 bit. Zudem kann ein längerer Text durch Datenkompressionen wie ZIP oder RAR beträchtlich speicherreduziert werden.

[11] `https://github.com/google/gson` [2015-10-05]

[12] `http://www.codeproject.com/Articles/604720/JSON-vs-XML-Some-hard-numbers-about-verbosity`

Gerade für die Übertragung sehr großer Datenmengen oder für einen effizienten Datenaustausch zwischen *cyberphysikalischen Systemen*, d.h. „intelligenten“ Gegenständen mit geringer Rechenleistung wie Uhren, Wearables, Sensoren, selbstfahrende Autos, lernende Thermostate, Kühlschränke, usw. sind daher binäre Austauschformate als Alternative sinnvoll. So war insbesondere das Ziel der schnellen Übertragung sehr großer Datenmengen – „*Big Data*“ – der Hauptgrund für die Entwicklung und den Einsatz solcher Formate bei Google und Facebook (Protocol Buffers und Apache Thrift). Dagegen wird ganz aktuell die Vernetzung cyberphysikalischer Systeme das sogenannte „Internet der Dinge“ *(Internet of Things, IoT)* bilden, das wiederum allgemein als die technische Grundlage für die künftige Logistik, Produktion, Gebäude- und Verkehrsinfrastruktur angesehen wird.

4.3.1 BSON

Binary JSON (BSON `http://bsonspec.org`) wurde für die Speicherung JSON-ähnlicher Datenstrukturen (Maps) entwickelt. Inhalte einer BSON-Datei können nachträglich geändert werden, ohne sie komplett neu schreiben zu müssen. Das unterscheidet BSON von anderen binären Austauschformaten. BSON wird in der NoSQL-Datenbank MongoDB zur Datenspeicherung verwendet.

Die Inhalte dieses Abschnitts stammen zu einem großen Teil aus[13].

4.3.2 CBOR

Concise Binary Object representation (CBOR) ist ein auf Datenkompaktheit hin optimiertes Austauschformat und eignet sich daher gut für Anwendungen im Internet der Dinge. Es ist 2013 im RFC 7049 (`http://tools.ietf.org/html/rfc7049`) und zielt darauf ab, in langlebigen Anwendungen auf veränderte Anforderungen reagieren zu können. Als Ausgangspunkt diente das JSON-Datenmodell, es werden jedoch weitere Datentypen unterstützt. Weitere Informationen sind unter `http://cbor.io/` verfügbar.

[13] Wojcieszak (2015).

Teil II

Serverseitige Programmierung am Beispiel von PHP

5 PHP Grundlagen

Kapitelübersicht

In diesem Kapitel werden die grundlegenden Eigenschaften der Programmiersprache PHP und die wesentlichen Syntaxregeln behandelt. Damit werden Sie in der Lage sein, erste Programme mit einfachen Berechnungen in PHP zu schreiben und mit Browser und Webserver ablaufen zu lassen.

PHP wird 1995 veröffentlicht

PHP wurde Anfang 1995 von Rasmus Lerdorf als eine Programmiersprache speziell für den Einsatz auf Webservern zur dynamischen Generierung von Webseiten entwickelt. Die Syntax von PHP ähnelt derjenigen von Java oder C sehr, angereichert mit einigen Elementen aus Perl (z.B. „$“, „#“ und „.“).

PHP = PHP Hypertext Preprocessor

PHP ist eine sogenannte *Skriptsprache*, also eine Interpretersprache, die erst zur Laufzeit im Arbeitsspeicher aus dem Quelltext übersetzt und ausgeführt wird.[1] Die Abkürzung PHP steht als rekursives Akronym[2] für ***PHP Hypertext Preprocessor*** (nach dem Vorbild von GNU = ***GNU's not Unix***). Ursprünglich allerdings bezeichnete der Entwickler von PHP, Rasmus Lerdorf, seine Programmiersprache als „Personal Home Page / Form Interpreter“, weshalb die erste PHP-Version 1995 noch „PHP/FI“ hieß; am 8. Juni 1995 wurde sie von ihm dann aber als „Personal Home Page Tools (PHP Tools)“ veröffentlicht.[3]

PHP ist eine quelloffene Software (*open source*), d. h. alle zugrunde liegenden Quelltexte sind

[1] Der Interpreter verwendet seit PHP 8 aus Performance-Gründen intern einen JIT-Compiler, der den Quelltext nach dem Vorbild von Java compiliert (Wenz und Hauser, 2021:§1.4).

[2] Ein *Akronym* ist ein Kunstwort, das sich aus Anfangsbuchstaben verschiedener Worte zusammensetzt, z.B. USA, NATO oder Azubi

[3] `https://en.wikipedia.org/wiki/PHP#Early_history`

offen zugänglich und unter Aufsicht eines Kontrollgremiums auch veränderbar. Insbesondere ist PHP damit kostenlos erhältlich, es gibt keine Lizenzgebühren.

5.1 Installation eines Webservers mit PHP und Datenbank

Ein PHP-Programm wird durch einen Interpreter als Modul eines Webservers ausgeführt, die *PHP-Engine*. Ein PHP-Skript kann nur von einem Browser aus über eine Webadresse aufgerufen werden. Entsprechend muss zwingend ein Webserver zusammen mit der PHP-Engine installiert sein. Um zusätzlich auf eine Datenbank zuzugreifen, muss entsprechend auch ein Datenbankserver vorhanden sein. Für alle gängigen Betriebssysteme gibt es das Installationsprogramm XAMPP, das die notwendigen Programme (Apache, PHP, MariaDB und PHPMyAdmin) direkt installiert. Es ist unter dem Link

Installation mit XAMPP

```
https://www.apachefriends.org/
```

verfügbar. Zu beachten ist, dass Ihre PHP-Dateien und HTML-Dokumente sich innerhalb des Wurzelverzeichnisses des Webservers befinden müssen, bei XAMPP unter dem Verzeichnis `htdocs` im Verzeichnis `xampp`. Am besten legen Sie darin ein eigenes Verzeichnis `programme` oder ähnlich an. In diesem Fall können Sie sie nach dem Start von XAMPP mit Ihrem Browser durch Eingabe der Webadresse

```
http://localhost/programme/
```

sehen und aufrufen.

5.2 Prinzipieller Ablauf von PHP-Programmen

Zunächst sei erwähnt, dass die Standardreferenz für PHP unter der Webadresse

```
https://php.net
```

zu finden ist und die API dokumentiert. Eine *API (Application Programming Interface)* ist allgemein die Bezeichnung für die Gesamtheit aller Funktionen, Variablen und Objekte, die eine Programmiersprache für die Programmierung standardmäßig zur Verfügung stellt. Das selbständige Arbeiten mit dieser Referenz wird dringend empfohlen.

Der Quelltext eines PHP-Skripts kann nur über eine HTTP-Anfrage an einen Webserver aufgerufen werden, der ihn dann über die PHP-Engine, also einen PHP-Interpreter als Unterprogramm, ausführt. Ein PHP-Skript muss daher immer im Dokumentenverzeichnis des Webservers gespeichert sein (z.B. `htdocs` bei Apache) und die Extension `.php` haben: Es kann nur von einem Browser aus über HTTP aufgerufen werden, also auf dem eigenen Rechner über

Dateiendung `.php`, PHP-Tag `<?php ... ?>`

```
http://localhost/...      oder      http://127.0.0.1/...
```

Da ein PHP-Skript ein vom Server ausgeführtes Programm ist, muss der Webserver es als solches erkennen. Mit der Extension `.php` verschickt er die Datei zunächst nicht über das Internet an den Client zurück, sondern ruft mit ihr die PHP-Engine auf, die den Programmcode zwischen den Tags `<?php` und `?>` übersetzt und Anweisung nach Anweisung ausführt. Das Ergebnis des Programmlaufes, also das, was das PHP-Modul zurückgibt, wird anstelle des PHP-Codes in die analysierte Datei eingefügt. Das so entstandene Ergebnis schickt der Webserver dann an den Browser. Kurz gesagt ist also das von HTML her bekannte Request-Response-Prinzip leicht verändert, der Browser fordert eine PHP-Datei an und erhält dynamisch erzeugtes HTML, wie

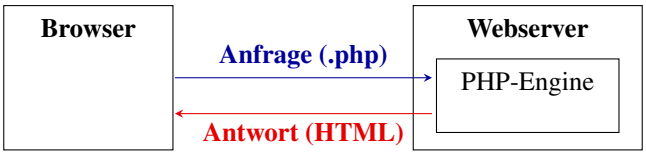


Abbildung 5.1: Eine PHP-Datei wird angefordert und erzeugtes HTML geliefert.

in Abbildung 5.1 dargestellt. Die PHP-Engine wird oft auch PHP-Interpreter oder PHP-Modul genannt.

Damit der Server erkennt, wann die Engine aufzurufen ist, muss der PHP-Quelltext in der Datei stets mit dem PHP-Tag `<?php ... ?>` umschlossen werden:

```
<?php
    ... PHP-Quelltext ...
?>
```

Ein solches PHP-Tag nennt man auch „Markierung nach dem XML-Stil“. Die Dateiendung `.php` und das PHP-Tag sind also beide notwendig, um ein PHP-Skript auszuführen.

Leider funktioniert das alles nicht, wenn man den XML-Vorspann `<?xml ...>` verwendet. Die Ursache ist offenbar, dass die aktuellen PHP-Interpreter bereits die beiden Zeichen `<?` als PHP-Tags interpretieren. Daher muss man den XML-Vorspann bei einem PHP-Dokument vermeiden. In den folgenden Beispielskripten werden wir meistens sowieso auf den Vorspann verzichten, die PHP-Skripte laufen auch ohne ihn. Oder man verwendet einfach den HTML-Vorspann `<!DOCTYPE html>`.

5.3 Ausgaben in PHP

Wir beginnen (wie bei Einführungen in Programmiersprachen üblich) mit einem Programm, welches „Hello World“ ausgibt.

Beispiel 5.1. Das Programm „Hello World“ in PHP

```
<!DOCTYPE html>
<html>
  <head><title>Hello World</title></head>
  <body>
<?php
   echo "<h2> Hello World </h2>";
?>
  </body>
</html>
```

Machen wir uns das Prinzip der HTML-Antwort an Beispiel 5.1 klar. Hier sehen wir in Zeile 6 den einleitenden `<?php`-Tag. In Zeile 7 folgt die einzige PHP-Anweisung des Programms:

```
   echo "<h2> Hello World </h2>";
```

`echo` ist einer der PHP-Befehle für die Ausgabe. Text, der hinter `echo` steht, wird ausgegeben, Zeichenketten müssen dabei in Anführungszeichen gesetzt werden. Jede Anweisung in PHP wird mit einem Semikolon beendet. In der nächsten Zeile wird durch `?>` die Programmausführung beendet. Die Server-Software ersetzt also

```
<?php
   echo "<h2> Hello World </h2>";
?>
```

durch `<h2> Hello World </h2>` und schickt den so entstandenen Ausgabetext an den Client, was zu der Darstellung in Abbildung 5.2 führt. □

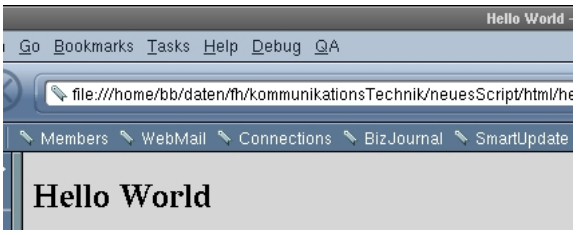


Abbildung 5.2: Ausgabe von Beispiel 5.1

Bei einem PHP-Skript wird also das *Ergebnis des Programmlaufs* auf dem Server an den Client übertragen. Der Browser „weiß“ gar nicht, dass er eine dynamisch erzeugte (d.h. „berechnete“) Seite erhält. Für den Browser besteht kein Unterschied zu statischen HTML-Seiten, insbesondere bedarf es keiner besonderen Einstellungen oder Browser-Plugins, um die Ausgaben von PHP-Programmen im Browser zu sehen.

5.4 Variablen in PHP

Betrachten wir als ein weiteres Beispiel ein Programm, das mit Hilfe von Variablen die Konstanten 9 und 10 addiert.

Beispiel 5.2. Ein erstes Additionsprogramm in PHP

```
<!-- Das Programm addiert 9 und 10 (oha!). Dateiname: addition1.php -->
<h2>Addition zweier Zahlen</h2>
<?php
   $ersterSummand = 9;
   $zweiterSummand = 10;
   $summe = $ersterSummand + $zweiterSummand;
   echo "Die Summe von $ersterSummand und $zweiterSummand
         ist $summe";
?>
```

□

Zunächst ist zu bemerken, dass Variablen in PHP mit dem `$`-Zeichen beginnen müssen. Variablen sind Referenzen, um Werte zu speichern und im Verlauf des Programms zu verwenden, also zu lesen oder zu verändern. Als nächstes ist zu erkennen, dass es in PHP keine Deklaration von Variablen gibt, also keine Festlegung des Datentyps. Zum Schluss des Skripts erfolgt die Ausgabe mit einer `echo`-Anweisung. Im Gegensatz zu den meisten Programmiersprachen können wir hier Variablen und normalen Text in den Anführungszeichen mischen. Der Grund ist, dass der PHP-Interpreter anhand des $-Zeichens erkennt, dass kein normaler Text, sondern der *Wert* einer Variable auszugeben ist. (Das $-Zeichen selbst wird durch `"\$"` ausgegeben.)

Man kann in Strings zur Deutlichkeit oder zur Abgrenzung auch geschweifte Klammern um die Variable verwenden, also

```
    echo "Die Summe von {$ersterSummand} und {$zweiterSummand} ..."
```

Manchmal ist dies sogar notwendig, z.B. wenn man ausgeben möchte `"Entfernung {$x}km"`.

Interessant ist der Blick auf den HTML-Text, den der Browser letztendlich vom Server empfängt. Man kann ihn sich anzeigen lassen, indem man die Funktion „Seitenquelltext anzeigen“ des Browsers aufruft. So erkennt man auch noch mal, dass der Programmcode von dem PHP-Modul *des Servers* ausgeführt wird und nur die Ausgaben über `echo` an den Browser übergeben werden. Der Browser erhält also reines HTML und kann dies direkt darstellen. *Alle Anweisungen eines PHP-Skripts werden von dem Webserver ausgeführt.*

5.4.1 Konkatenation von Strings mit dem Punktoperator

Der Operator zur Verknüpfung (Konkatenation oder Stringaddition) von Zeichenketten ist in PHP der Punkt (`.`). (In vielen Programmiersprachen wie Java, C# oder Python ist es das +-Zeichen.) Möchten wir zum Beispiel in unserem obigen Programm auf die Variable `$summe` verzichten und die Summe bei Ausgabe des Strings zu berechnen, so wäre eine mögliche Variante:

```
    echo "Die Summe von $ersterSummand und $zweiterSummand
          ist: " . ($ersterSummand + $zweiterSummand);
```

Wir werden in Zukunft noch oft den verwandten Operator

```
.=
```

verwenden, der der Variablen links von ihm den String rechts von ihm anhängt, oder „konkateniert“:

```
$text  = "Es gilt $ersterSummand + $zweiterSummand = ";
$text .= $ersterSummand + $zweiterSummand;  // Konkatenation
echo $text;  // Ausgabe an den Browser
```

Dies ergibt dieselbe Ausgabe wie unsere originale Version in Beispiel 5.2.

5.4.2 Regeln und Konventionen für Variablennamen

Abschließend noch einige Regeln zu Variablennamen.

- Variablennamen beginnen in PHP mit dem $, auf das $-Zeichen muss ein Buchstabe oder der Unterstrich _ folgen.
- Variablennamen können nach dem ersten Buchstaben Ziffern und Buchstaben in beliebiger Reihenfolge enthalten.
- Variablennamen dürfen außer dem Unterstrich _ keine Sonderzeichen enthalten, insbesondere Umlaute.
- Variablennamen können beliebig lang sein.
- Groß- und Kleinschreibung spielt eine Rolle (`$summe` ≠ `$Summe` ≠ `$suMME` ≠ `$SUMME`). Variablennamen sollten klein geschrieben werden.
- Variablennamen sollten einen Bezug zu den Daten haben, die sie speichern sollen.

5.5 Kommentare in PHP

Kommentare werden beim Programmablauf ignoriert. Kommentare sind dazu da, unseren Programmcode verständlicher zu machen.

Beispiel 5.3. Kommentare in PHP

```php
<?php
   /** Autor: Andreas de Vries
    *  Version: 17.06.2022
    *  Zweck: Addition zweier Zahlen
    */
   $ersterSummand  = 9;
   $zweiterSummand = 10;
   // Aufbereiten der Ausgabe:
   $text  = "Es gilt $ersterSummand + $zweiterSummand = ";
   $text .= $ersterSummand + $zweiterSummand;
   echo $text;  // Ausgabe an den Browser
?>
```

□

In PHP gibt es drei Arten von Kommentaren:

- Sämtlicher Text zwischen `/*` und `*/` wird ignoriert. In dem Beispielprogramm wird diese Kommentarform benutzt, um einige allgemeine Informationen zum Programm in die Datei aufzunehmen.
- Auf `//` folgender Text wird bis zum Zeilenende ignoriert. In dem Beispielprogramm wird diese Kommentarform benutzt, um den Sinn der einzelnen Programmzeilen zu dokumentieren.
- Auf `#` folgender Text wird bis zum Zeilenende ignoriert.

Für dieses kleine Programm sind die Kommentare sicherlich unnötig. Doch schon bei geringfügig größeren Programmen sind Kommentare notwendig. Sie dienen Ihnen dazu, Ihr eigenes Programm auch noch nach einigen Wochen zu verstehen. Anderen können Kommentare erleichtern, den Algorithmus nachzuvollziehen. Kommentare können auch außerhalb der PHP-Tags `<?php ... ?>` geschrieben werden. Allerdings muss dann der HTML-Kommentar `<!- ... ->` verwendet werden.

5.6 Datentypen und Operatoren

Viele Programmiersprachen bieten die Möglichkeit, durch eine Deklaration einer Variablen einen Datentyp zuzuweisen und damit festzulegen, welche Dinge auf dieser Variablen abgespeichert werden können. Skriptsprachen wie PHP bieten diese Möglichkeit nicht. Solche Sprachen nennt man *untypisiert*, oder wie PHP *schwach typisiert*. Grundsätzlich unterstützt PHP die folgenden Datentypen:

- *Strings* (`string`): Zeichenketten, beispielsweise `"Abc"` oder `"Er sagte: \"Ich gehe!\" und ging."`. In PHP wird ein String durch die doppelten Anführungszeichen `"` markiert, aber auch durch Apostrophs `'`. Da aber Apostrophs einerseits das $-Zeichen nicht als Markierung für eine Variable erkennen und andererseits wir es später für SQL-Befehle benötigen, ist von ihrer Benutzung eher abzuraten.

- *Ganze Zahlen* (`int` oder `integer`) z. B. -1, 1000, 3, 7, 12, -1000
- *Reelle oder Fließkommazahlen* (`double` oder `real`): z.B. `-1.2`, `-100.001`, `2.0e-12`. Merke: . (Punkt) statt , (Komma) in PHP als Dezimaltrenner.
- *Wahrheitswerte* (`bool`): `true` oder `false`, bzw. 1 oder 0.

Strings sind gewissermaßen der „Hauptdatentyp“ in PHP, alle Variablen sind prinzipiell Strings, es sei denn sie werden mit numerischen oder logischen Operatoren (s.u.) verknüpft.

In PHP ist es dementsprechend nicht möglich, Variablen zu deklarieren. Man schreibt einfach den Namen einer Variablen in ein Programm und PHP erzeugt die Variable. Der Datentyp wird abhängig von den Operatoren automatisch zugeordnet.

Will man prüfen, ob der aktuelle Wert einer Variablen `$x` numerisch ist, so kann man das mit der Funktion

```
is_numeric($x)
```

tun, sie gibt `true` zurück, wenn `$x` eine Zahl ist. Beispielsweise ergeben `is_numeric(3)`, `is_numeric (3.14)`, `is_numeric(.14)` oder `is_numeric(-3)` stets `true`, `is_numeric("3")` oder `is_numeric("A")` dagegen `false`.

Entsprechend gibt es die folgenden Funktionen, mit denen man den Datentyp des aktuellen Werts einer Variablen prüfen kann:

```
is_bool($x),
is_int($x), is_integer($x), is_long($x),
is_double($x), is_float($x), is_real($x),
```

Der Vollständigkeit halber sei erwähnt, dass eine Variable in PHP ansonsten nur noch ein Array oder ein Objekt sein kann.

5.6.1 Undefinierte Variablen und `isset`

Hat eine Variable in einem PHP-Skript noch keinen Wert erhalten, ist sie noch undefiniert. Will man zu einem bestimmten Zeitpunkt in einem PHP-Skript prüfen, ob eine gegebene Variable `$x` noch undefiniert ist, so kann dies mit der Anweisung `isset($x);` geschehen.

Beschreibt eine Variable `$a` ein Array, so zeigt `isset($a)` zwar an, ob das Array selbst initialisiert wurde, nicht aber, ob es Werte *enthält*. Diese Information liefert die Funktion `empty`, d.h. `empty($a)` ergibt `true` genau dann, wenn das Array `$a` leer ist.

5.6.2 Operatoren

Einer der wichtigsten Operatoren ist der Zuweisungsoperator `=`, der in der Variablen links von ihm den auf seiner rechten Seite festgelegten oder berechneten Wert speichert:

```
$x = 1.19 * 1000;
```

Arithmetische Operatoren

PHP kennt und unterstützt die gängigen arithmetischen Operatoren. Wie auch in der Schulmathematik besitzen diese Operatoren verschiedene Präzedenz (Vorrang), z. B. „Punkt- vor Strichrechnung“. Wie in der Mathematik kann man die Präzedenz der Operatoren durch Klammerung ändern, d. h. die runden Klammern haben die höchste Präzedenz. In Tabelle 5.1 sind die arithmetischen Operatoren und ihre Präzedenz aufgelistet.

Operator	Operation	Präzedenz
`()`	Klammern	werden zuerst ausgewertet. Sind Klammern von Klammern umschlossen, werden sie von innen nach außen ausgewertet.
`**`	Potenz	wird als zweites ausgewertet
`*`, `/`, `%`	Multiplikation, Division, Modulus	werden danach ausgewertet
`+`, `-`	Addition, Subtraktion	werden zuletzt ausgewertet

Tabelle 5.1: Die Präzedenz der arithmetischen Operatoren in PHP. Mehrere Operatoren gleicher Präzedenz werden stets von links nach rechts ausgewertet.

Beispiel 5.4. *Arithmetische Operatoren.* Das folgende PHP-Skript führt die gängigen arithmetischen Operationen durch, das jeweilige Ergebnis steht als Kommentar daneben.

```
<?php
    $i = 2;
    $j = 3;
    $l = $i + $j;        // 5;
    $l = $i - $j;        // -1
    $l = $j % $i;        // 1 ( modulo-Rechnung )
    $l = $i * $j;        // 6
    $l = $j / $i;        // 1.5 ( Division )
    $l = $i + $i * $j;   // 8
    $l = ($i + $i) * $j; // 12
    $l = $i * $j ** $i;  // 18 ( Potenz- vor Punktrechnung )
?>
```

□

Beispiel 5.5. *Kombinierte Zuweisungsoperatoren.* Wie alle C-ähnlichen Sprachen erlaubt PHP in einigen Fällen abkürzende Schreibweisen mit kombinierten Zuweisungsoperatoren.

```
$i += $j  // $i = $i + $j
$i -= $j  // $i = $i - $j
$i *= $j  // $i = $i * $j
$i /= $j  // $i = $i / $j
$i %= $j  // $i = $i % $j
$a .= $b  // $a = $a . $b
$i++      // $i = $i + 1
$i--      // $i = $i - 1
```

Die beiden letzten Operatoren, `++` und `-`, werden auch *Inkrement-* bzw. *Dekrementoperator* genannt. □

Beispiel 5.6. *(Plus- und Punktoperator)* PHP benutzt den Punkt (`.`), um zwei Strings miteinander zu verbinden, wie wir bereits in Abschnitt 5.4.1 gesehen haben. Das folgende Beispiel zeigt, dass zwei Strings aber auch mit dem Pluszeichen verknüpft werden können.

```
<!-- Dateiname: plusOperator.php -->
<?php
  $s1 = "ab";
  $s2 = "cd";
```

```
  $i  = 5;
  $j  = "6";
  $text = $s1 + $s2 . "<br/>";  // ergibt "0"
  $text .= $s1 . $s2 . "<br/>"; // ergibt "abcd"
  $text .= $s1 + $i . "<br/>";  // ergibt "5"
  $text .= $s1 . $i . "<br/>";  // ergibt "ab5"
  $text .= $j . $i . "<br/>";   // ergibt "65"
  $text .= $j + $i . "<br/>";   // ergibt "11"
  echo $text;
?>
```

Wir sehen, dass PHP zuweilen unschöne Dinge tut. Strings, die sich nicht in Zahlen umwandeln lassen, werden bei der Addition mit + auf 0 gesetzt. Es gibt keine Fehlermeldung, an der man erkennen könnte, dass diese Operation mit den beteiligten Operanden wenig Sinn macht. □

Vergleichsoperatoren

Tabelle 5.2 zeigt die in PHP verfügbaren Vergleichsoperatoren.

Symbol	Operator in PHP	Beispiel	Bedeutung
$=$	`==`	x `==` y	x hat den gleichen Wert wie y
$\neq$	`!=, <>`	x `!=` y	x hat nicht den gleichen Wert wie y
$=$	`===`	x `===` y	x und y haben gleichen Datentyp und Wert
$\neq$	`!==`	x `!==` y	Datentyp oder Wert von x und y sind ungleich
$>$	`>`	x `>` y	x ist größer als y
$<$	`<`	x `<` y	x ist kleiner als y
$\geqq$	`>=`	x `>=` y	x ist größer gleich y
$\leqq$	`<=`	x `<=` y	x ist kleiner gleich y

Tabelle 5.2: Vergleichsoperatoren in PHP.

Beispielsweise ist `"5" === 5` in PHP `false`, aber `"5" == 5` ist `true`.

Vergleichsoperatoren werden in Verbindung mit bedingten Anweisungen und Schleifen genutzt. Beide Programmierkonstrukte wurden bisher noch nicht behandelt. Darum wirken die jetzt vorgestellten Beispiele etwas konstruiert. Sie sollen nur die Ergebnisse der Vergleiche zu zeigen. Besondere Beachtung verdient der Operator ==. Er darf keinesfalls mit dem Zuweisungsoperator = verwechselt werden, ist aber etwas „lockerer“ bei seinen Vergleichen als ===. Beispiel 5.7 zeigt die Wahrheitswerte einiger Vergleiche und die Boole’schen Werte `true` und `false`. Betrachten wir das folgende Beispiel.

Beispiel 5.7. *(Vergleichsoperatoren in PHP)* Das folgende Programm mit verschiedenen Vergleichsoperatoren ergibt die Ausgabe rechts:

```php
<Dateiname: vergleichsOeratoren.php -->
<?php
   $s1 = "ac";
   $s2 = "ad";
   $i  = 5;
   $j  = "5";

   echo "$s1 != $s2: " . ($s1 != $i ) . "<br/>";
   echo "$s1 < $s2: "  . ($s1 < $s2 ) . "<br/>";
   echo "$s1 == $s2: " . ($s1 == $s2) . "<br/>";
   echo "$s1 = $s2: "  . ($s1 = $s2)  . "<br/>";
   echo "$i === $j: "  . ($i === $j ) . "<br/>";
   echo "$j !== $i: "  . ($j !== $i)  . "<br/>";
   echo "$i == $j: "   . ($i == $j )  . "<br/>";
   echo "$j != $i: "   . ($j != $i )  . "<br/>";
   echo "$j <> $i: "   . ($j <> $i )  . "<br/>";
   echo "$j = $i: "  . ($j = $i )   . "<br/>";
   echo "$i === $j: "  . ($i === $j ) . "<br/>";
   echo "(5 == 3) === false: " . ((5 == 3) === false) . "<br/>";
?>
```

```
ac != ad: 1
ac < ad: 1
ac == ad:
ac = ad: ad
5 === 5:
5 !== 5: 1
5 == 5: 1
5 != 5:
5 <> 5:
5 = 5: 5
5 === 5: 1
(5 == 3) === false: 1
```

Ein Vergleich wie `($s1 < $s2)` ergibt `true` oder `false`, je nach zugrunde liegendem Ordnungsbegriff. Die Konstanten `true` und `false` (übrigens auch in Groß- und Kleinbuchstaben geschrieben, auch wild gemischt!) werden von `echo` nur als leerer String `""` (false) oder 1 (true) dargestellt. □

Der Operator == wird *Gleichheitsoperator* genannt, der Operator === *Identitätsoperator*. Die erste Bezeichnung ist etwas missverständlich, denn in typisierten Programmiersprachen wie C und Java versteht man unter dem Gleichheitsoperator das, was in PHP === ist, das „PHP-==“ gibt es in diesen Sprachen gar nicht. Werte, die nur mit == false sind, werden auch „falsy“ genannt, und Werte, die nur mit == true sind, entsprechend „truthy“.

Ansonsten sehen wir, dass sich Vergleichsoperatoren so verhalten, wie wir das erwarten. Eine Ausnahme besteht, wenn wir den Zuweisungsoperator verwenden: Das Ergebnis einer Zuweisung ist der Wert der zugewiesenen Variable. Dies wird insbesondere von besonderer Bedeutung sein, wenn wir uns im nächsten Kapitel mit Kontrollstrukturen (der if-Anweisung) beschäftigen.

Logische Operatoren

Logische Operatoren werden benutzt, um Vergleiche zu kombinieren oder zu negieren und um logische Aussagen zu bilden.

- *Logisches AND (`&&` oder `AND`):* Der `&&`-Operator führt die logische UND-Verknüpfung durch. Er ergibt true, wenn seine beiden Operanden true ergeben, ansonsten false.
- *Logisches OR (`||` oder `OR`):* Der `||`-Operator führt die logische ODER-Verknüpfung durch. Er ergibt true, wenn einer seiner beiden Operanden true ist, ansonsten false.
- *XOR = exklusives ODER (`!=`, `XOR` oder `^`):* Der XOR-Operator, führt die logische Verknüpfung des Exklusiven ODER durch. Hierbei ist *A* `!=` *B* wahr, wenn *A* und *B* *unterschiedliche* Wahrheitswerte haben, und falsch, wenn sie gleiche Wahrheitswerte haben.
- *Logische Negation (`!`)* Der Negations-Operator `!` wird auf nur einen Operanden angewendet. Er ergibt true, wenn sein Operand false ist und umgekehrt. Beispiel 5.8 zeigt einige durch logische Operatoren verkettete Vergleiche und deren Ergebnisse.

Die Wahrheitstabellen in Tab. 5.3 zeigen die möglichen Ergebnisse der logischen Operatoren, wobei 0 = `false` und 1 = `true` (Tab. 5.3).

a	b	a && b	a || b	a != b	!a
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1	0	0

Tabelle 5.3: Wahrheitstabellen für verschiedene logische Operatoren.

Beispiel 5.8. *(Logische Operatoren in PHP)* Das folgende Programm mit logischen Operatoren ergibt die Ausgabe rechts:

```
<!-- Dateiname: logischeOperatoren.php -->
    <?php
      $i=5;
      $j=6;
      $k=6;
      $l=5;
      $ergebnis = ($i == $j) && ($j == $k); // false, da i nicht j
      $text  = "ergebnis1: $ergebnis <br/>";
      $ergebnis = ($i == $j) || ($j == $k); // true, da j gleich k
      $text .= "ergebnis2: $ergebnis <br/>";
      $ergebnis = ($i == $l) && ($j == $k); // true
      $text .= "ergebnis3: " . $ergebnis . "<br/>";
      $ergebnis = ($i == $l) || ($j == $k); // true
      $text .= "ergebnis4: $ergebnis <br/>";
      $ergebnis = !($i == $j);              // true
      $text .= "ergebnis5: $ergebnis <br/>";
      $ergebnis = !($i != $j);              // false
      $text .= "ergebnis6: $ergebnis <br/>";
      $ergebnis = ($i != $j) ^ ($k == $l);  // true
      $text .= "ergebnis7: $ergebnis <br/>";
      echo $text;
    ?>
```

```
ergebnis1: 0
ergebnis2: 1
ergebnis3: 1
ergebnis4: 1
ergebnis5: 1
ergebnis6:
ergebnis7: 1
```

Die Zeile

```
$ergebnis = ($i == $j) && ($j == $k)
```

ähnelt hier sehr den Ausdrücken des vorherigen Kapitels. Der einzige Unterschied besteht darin, dass wir auf der rechten Seite des Ausdrucks zwei Vergleiche sehen. Beide sind durch einen logischen Operator (in diesem Beispiel `&&`) verbunden. Der Interpreter wertet zunächst beide logischen Ausdrücke aus und danach die Vernüpfung. Das Ergebnis wird der Variable `$ergebnis` zugewiesen. Entsprechend funktionieren die anderen ANweisungen. □

5.7 Zusammenfassung

- Die Endung einer PHP-Datei muss `.php` lauten, damit sie von dem Webserver als PHP-Skript erkannt und der PHP-Interpreter aufgerufen wird.

- PHP-Code muss stets von dem PHP-Tag `<?php` und `?>` umschlossen sein. Der Interpreter übersetzt („parst“) die Anweisungen innerhalb des Tags und führt jede Anweisung sofort aus.
- Nach jeder Anweisung muss ein Semikolon gesetzt sein.
- Einzeilige Kommentare werden mit `//` begonnen, mehrzeilige mit `/* ... */` umschlossen.
- Die Ausgabe geschieht mit dem Befehl `echo`.
- Ruft der Browser eine PHP-Datei auf, so wird das Skript ausgeführt und er erhält als Antwort HTML, (meist) erzeugt mit `echo` (Abb. 5.1).
- Strings werden mit dem Konkatenationsoperator `.` verknüpft.
- Variablen beginnen stets mit dem Dollarzeichen `$`. Sie werden nicht deklariert, sondern erhalten ihren Datentyp mit einer Wertzuweisung.

6 Kontrollstrukturen in PHP

Kapitelübersicht

6.1 Die if-Anweisung

Oft sollen Anweisungen nur unter bestimmten Bedingungen ausgeführt werden. Die Lösung solcher Probleme ist die `if`-Anweisung. Sie hat die Form:

```
if ( bedingung ) {
    anweisung;
      ...
} else {
    anweisung;
      ...
}
```

`bedingung` ist ein logischer oder Boole'scher Ausdruck, der einen Wahrheitswert (`true` oder `false`) annimmt. Vergleiche z.B. sind solche Boole'schen Ausdrücke (vgl. Kapitel 5.6.2).

Ergibt der logische Ausdruck den Wert `true`, werden die Anweisungen im `if`-Teil (oder `if`-Block) ausgeführt, andernfalls (der logische Ausdruck ergibt `false`) werden die Anweisungen

im `else`-Teil (oder `else`-Block) ausgeführt.
Regeln:

- Vor `else` darf kein `;` stehen.
- `if`-Anweisungen können beliebig tief geschachtelt werden.
- Der `else`-Block ist optional. Gibt es den `else`-Block nicht, ist dies gleichbedeutend mit: Ansonsten tue nichts. Ist ein `else`-Teil vorhanden, sprechen wir auch von einer verzweigten Selektion.

Prinzipiell können zwar die geschweiften Klammern im `if`-Block oder im `else`-Block weggelassen werden, wenn der Block nur aus einer Anweisung besteht; allerdings ist davon dringend abzuraten, denn insbesondere durch das „dangling-else“-Problem ist dies sehr unübersichtlich und fehleranfällig. Das Problem des *dangling else* ist die scheinbare Mehrdeutigkeit einer verschachtelten if-Anweisung:

```
$a = 0; $b = 1;
if ($a == 1)
   if ($b == 1)
      $a = 42;
else
   $b = 42;
```

Ist `$b = 1` oder `$b = 42`? Lösung:

Der else-Zweig gehört zum inneren `if`, nicht zum äußeren, wie es die Einrückung suggeriert; daher ist `$b = 1`.

6.1.1 Mehrfachoptionen: Verschachtelte `if`-Anweisungen

Zur Verarbeitung mehrfacher Optionen (z.B. Monat ist *entweder* 1 *oder* 2 *oder* . . .) wird eine Konstruktion verschachteler if-Anweisungen benötigt. In PHP gibt es dazu das reservierte Wort `elseif`.

Beispiel 6.1. *(Die if-else-if-Leiter)* Eine Abfolge von mehreren if-else-Anweisungen wird auch *if-else-if-Leiter* genannt. Betrachten wir dazu das folgende Programm, das uns die Anzahl der Tage eines Monats eines Nichtschaltjahres errechnet.

```
$monat = 2;    // Beispielwert
$jahr  = 2022; // Beispielwert
if( ( $jahr % 4 == 0 ) && ( $jahr % 200 == 0 ) ) {
   $schaltjahr = true;
} else {
   $schaltjahr = false;
}

if (
   $monat === 1 || $monat === 3  || $monat === 5 ||
   $monat === 7 || $monat === 10 || $monat === 12
) {
   $tage = 31;
} elseif ($monat === 4 || $monat === 6 || $monat === 9 || $monat === 11) {
   $tage = 30;
```

```
} elseif ($monat === 2 && $schaltjahr) {
   $tage = 29;
} elseif ($monat === 2 && !$schaltjahr) {
   $tage = 28;
} else { // ungültiger Monatswert
   $tage = 0;
}
```

Die Konstruktionen `elseif` in Zeile 4 und 6 sind eigentlich verschachtelte if-Anweisungen, aber ohne geschweifte Klammern. Da hier kann kein Dangling-else passieren kann, können wir auch `else if` statt `elseif` schreiben. □

Oft wird für Alternativen die `switch-Anweisung` verwendet. Sie ist jedoch logisch ein Spezialfall der if/else-if-Konstruktion und wird daher hier nicht vorgestellt. Für ihre Syntax sei auf die Literatur verwiesen,[1] oder auf die PHP-Dokumentation:

`https://php.net/manual/de/control-structures.switch.php`

6.1.2 Der Bedingungsoperator

Der *Bedingungsoperator (conditional operator)* ist eine sehr kompakte Variante der if-else-Anweisung. Er hat die Form

$$A \ ? \ B \ : \ C$$

mit den drei Operanden A, B, C. Der erste kommt vor das Fragezeichen, der zweite zwischen Fragezeichen und Doppelpunkt, der dritte hinter den Doppelpunkt. Der erste Operand A ist eine Bedingung, B und C sind jeweils Werte (oder Anweisungen). Ergibt A `true`, so wird der Wert B (der zweite Operand) angenommen, ansonsten C (der dritte Operand), vgl. Beispiel 6.2.

Beispiel 6.2. Der Bedingungsoperator □

```
<!-- Der Fragezeichenoperator. Dateiname: fragezeichen.php //->
    <?php
       $i = 1;
       $j = 2;
       $min = ($i < $j) ? $i : $j;
       echo " min = $min"; // Ausgabe: "min = 1"
    ?>
```

Die Wirkung ist also äquivalent zu

```
    if ( $i < $j ) {
       $min = $i;
    } else {
       $min = $j;
    }
```

Ein Operator mit drei Operanden heißt *ternärer Operator*.

[1] z.B. Wenz und Hauser (2021):§5.2.2.

6.2 Schleifen

Schleifen oder Loops gehören neben den `if`-Anweisungen zu den üblichen Kontrollstrukturen. Durch Schleifen kann ein Programmblock mehrfach durchlaufen werden. Die Anzahl der Wiederholungen kann fest oder variabel definiert werden und hängt von den Bedingungen und der Schleifenstruktur ab.

In PHP gibt es vier Schleifenstrukturen, die abweisende `while`-Schleife, die nichtabweisende `do/while`-Schleife, die zählergesteuerte `for`-Schleife, und als letzte die Array-bezogene `foreach`-Schleife, die wir in Kapitel 7 kennen lernen.

6.2.1 `while`-Schleife

Die `while`-Schleife prüft zunächst eine Bedingung, die Schleifenbedingung. Solange diese erfüllt ist, wird der Anweisungsblock durchlaufen, anschließend wird die Bedingung erneut geprüft.

```
while ( Bedingung ) {
   anweisungen;
}
```

Als Beispiel berechnen wir näherungsweise die Quadratwurzel einer Zahl z mit Hilfe des *Newton'schen Iterationsverfahrens*.[2] Bei jedem Schleifendurchlauf wird anfangs geprüft, ob das Quadrat der Wurzel w bereits hinreichend nah bei der Zahl liegt, hier z.B. ob $|w^2 - z| < \varepsilon z$ mit ε = 1e-12 = 10^{-12}. Ist das Ergebnis nicht genau genug, wird mit der Formel

$$w \leftarrow \frac{1}{2}\left(w + \frac{z}{w}\right) \qquad \text{bzw.} \qquad w_{k+1} = \frac{1}{2}\left(w_k + \frac{z}{w_k}\right)$$

($k \geqq 0$) die nächste Iteration berechnet. Übrigens: Dieses Verfahren konvergiert extrem schnell für einen beliebigen Anfangswert und wird zur Berechnung der Quadratwurzel in Programmiersprachen verwendet.

```
<!-- Berechnet die Wurzel einer Zahl. Dateiname: wurzel.php -->
  <h2>Wurzelberechnung nach dem Newton-Verfahren</h2>
  <?php
      $zahl = 10;
      $wurzel = $zahl/2;
      $epsilon = 1e-12 * $zahl;

      while ( abs($wurzel*$wurzel - $zahl) >  $epsilon ) {
          $wurzel = ($wurzel + $zahl/$wurzel)/2;
          echo " $wurzel<br/>"; // zum Debuggen
      }
      echo " Wurzel von $zahl = $wurzel";
  ?>
```

Hierbei liefert die Standardfunktion `abs($x)` den Absolutbetrag $|x|$ von x. PHP hat weitgehend dieselben mathematischen Standardfunktionen wie z.B. die Klasse `Math` in Java. Die Ausgabe des Programms zeigt eine immer bessere Näherung an die Quadratwurzel der angegebenen Zahl, hier z.B. 10:

[2] Das angegebene Verfahren zur näherungsweisen Berechnung der Quadratwurzel wird dem griechischen Mathematiker Heron von Alexandria (wohl um 100 n. Chr.) zugeschrieben und daher manchmal als *Heron'sches Verfahren* bezeichnet. Mathematisch ist es jedoch ein Spezialfall des Newton'schen Verfahrens $x_{k+1} = x_k - f(x_k)/f'(x_k)$, und zwar mit $f(x) = z - x^2$.

```
3.5
3.1785714285714
3.1623194221509
3.1622776604441
3.1622776601684
```

Problematisch bei der `while`-Schleife ist, dass die Schleifenbedingung unter Umständen immer wahr bleiben kann, damit läge eine Endlosschleife vor. Das ist insbesondere äußerst kritisch, da in diesem Falle der Webserver behindert wäre, der das PHP-Skript ja ausführt. Um diesen Effekt zu verhindern, ist die Laufzeit eines PHP-Skripts auf maximal 30 Sekunden beschränkt. Daher sollten laufzeitintensive Algorithmen *nicht* in PHP implementiert werden.

6.2.2 `do/while`-Schleife

Die `do/while`-Schleife führt zunächst seinen Anweisungsblock aus und prüft seine Schleifenbedingung erst zum Schluss. Solange diese erfüllt ist, wird der Anweisungsblock erneut durchlaufen. Syntax:

```
do {
   anweisungen;
} while ( bedingung );
```

Wie bei der `while`-Schleife kann es auch hier zu einer Endlosschleife kommen.

6.2.3 `for`-Schleife

Die `for`-Schleife fasst Initialisierungsanweisung (*start*), Schleifenbedingung (*test*) und Iterationsanweisung (*update*) in dem Schleifenkopf zusammen.

```
for (start; check; update) {
   anweisungen;
}
```

Hier bezeichnet *start* also die einmalig zum Start der Schleife durchzuführenden Initialisierungen von Variablen, *check* die vor jeder Iteration zu überprüfende Schleifenbedingung und *update* stets am Ende des Schleifenrumpfs auszuführenden Anweisungen. Die `for`-Schleife eignet sich besonders zur Implementierung einer Zählschleife, also einer Schleife, die eine Zählvariable oder einen „Index“ von einem Startwert bis zu einen Endwert hochzählt:

```
for ($i = 0; $i < 10; $i++) {
   echo "$i, ";
}
```

oder runterzählt:

```
for ($i = 10; $i > 0; $i--) {
   echo "$i, ";
}
```

In dieser Schleife werden also im Gegensatz zu den `while`-Schleifen die für die Verwaltung einer Schleife wichtigen Anweisungen übersichtlich im Schleifenkopf aufgelistet. Auch mit diesem Schleifenkonstrukt kann man eine Endlosschleife programmieren, wenn nämlich die Update-Anweisung (und die Anweisungen im Schleifenrumpf) nicht dazu führen, dass die Schleifenbedingung falsch wird. Beispielsweise: `for($i=10; $i>0; $i++)` ...

6.2.4 foreach-Schleife

An dieser Stelle sei auf eine weitere wichtige Schleifenkonstruktion hingewiesen, die allerdings nur mit der Kenntnis von Arrays oder Listen zu verstehen ist und daher erst im Abschnitt 7.2 auf Seite 75 und mit einer wichtigen Variante in Abschnitt 7.3 ab Seite 77 behandelt wird.

6.3 Funktionen

Wie in anderen Programmiersprachen ist es in PHP möglich, häufig benutzte Programmteile als *Funktionen* zu definieren und von anderen Programmteilen oder ganz anderen Programmen aus aufzurufen. An Funktionen können Parameter übergeben werden, von denen der Ablauf der Funktion abhängt. Eine Funktion wird mit einer `return`-Anweisung beendet, wenn sie einen Wert an den aufrufenden Programmteil zurückgeben soll. Die Syntax der Deklaration einer Funktion beginnt in PHP mit dem reservierten Wort `function`, gefolgt von einem Block in geschweiften Klammern:

```
function myFunction ($x1, $x2, ...) {
   anweisungen;
   ...;
   return $ergebnis;
}
```

Hierbei können Standardwerte für Eingabeparameter vergeben werden, so dass bei einem Funktionsaufruf die Parameter „von hinten beginnend“ weggelassen werden können. Mit diesem Konzept der *optionalen Parameter* kann man in PHP Funktionen in eingeschränkter Weise „überladen“.[3]

optionale Parameter

Beispiel 6.3. Betrachten wir als Beispiel für Funktionen das folgende PHP-Skript:

```
<!-- Einige Funktionen. Dateiname: funktionen.php -->
<h2>Einige Funktionen</h2>
<?php
   function f($x) {
     return $x * $x;
   }

   function pythagoras($x, $y = 12) {
     return sqrt($x*$x + $y*$y);
   }

   /** Text wird gelb grundiert */
   function gelbHG($text) {
     return "<span style='background-color:yellow'>$text</span>";
   }

   echo " f(2) = " . f(2) . ", f(4) = " . f(4) . "<br/>";
   $x = 3; $y = 4;
   echo " pythagoras($x,$y) = " . pythagoras($x,$y);
```

[3] In der Programmierung nennt man die Deklaration von Funktionen mit gleichem Namen, aber unterschiedlicher Parameterliste, ein *Überladen* der Funktion. Da in PHP aufgrund der dynamischen Typisierung von Variablen eine wohldefinierte Signatur von Funktionen nicht existiert, ist ein echtes Überladen wie in Java nicht möglich.

```
   $x = 9;
   echo "<br/> pythagoras($x) = " . pythagoras($x) . "<br/>";
   echo " Dieser " . gelbHG("Textteil") . " wird gelb grundiert.";
?>
</body>
</html>
```

□

Es sind drei Funktionen deklariert, zwei mathematische und eine, die einen String formatiert. Es wird eine der mathematischen Standardfunktionen von PHP aufgerufen, `sqrt`. Die Funktion `pythagoras` ist mit dem zweiten optionalen Parameter `$y` deklariert, der bei einem Aufruf mit nur einem Parameter mit dem Standardwert 12 belegt wird. Die Ausgabe des Programms in

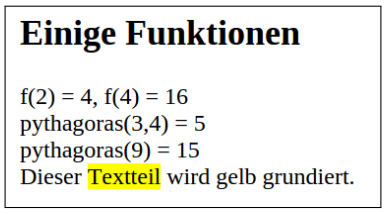


Abbildung 6.1: Die Browseranzeige von Beispiel 6.3.

Beispiel 6.3 im Browserfenster ist in Abbildung 6.1 dargestellt. Beachten Sie bei der Analyse des Aufrufs `pythagoras(9)`, dass $9^2 + 12^2 = 15^2$.

6.3.1 Lokale, globale und superglobale Variablen

Wird eine Variable in PHP innerhalb einer Funktion verwendet, so ist sie eine *lokale Variable*, ihr Geltungsbereich ist also nur die Funktion.

```
$temp = 5;
function lokalTest($x) {
   echo "In Funktion (vorher): \$temp = $temp<br/>";
   $temp = 2*$x;
   echo "In Funktion (nacher): \$temp = $temp<br/>";
}

lokalTest(50);
echo "\$temp = $temp";
```

Man kann in PHP zwar auch „globale“ Variablen verwenden, indem man sie *in der Funktion* als `global` deklariert. Mit `global` zu arbeiten, ist aus zwei Gründen jedoch nicht ratsam. Erstens sind die so deklarierten Variablen nicht echt global, denn man muss ja in jeder Funktion separat festlegen, dass sie es sind. (`global $x;` außerhalb einer Funktion nutzt es *nichts*!) Zum zweiten kann es bei etwas umfangreicheren Systemen dazu kommen, dass in irgendeiner von tausenden von Funktionen eine Variable als global deklariert ist, deren Namen auch irgendwo im Hauptskript verwendet wird. Die Funktion ändert ihren Wert, und die Ursache ist sehr schwer zu finden.

lokale Variablen gelten in Funktionen, superglobale sind vordefiniert

Die einzigen echt globalen Variablen heißen in PHP *superglobal*. Sie gelten an jeder Stelle eines PHP-Skripts, also auch in Funktionen (und dürfen dort nicht als `global` deklariert werden!). Superglobale Variablen sind meist Arrays, die externe Daten enthalten, die wichtigsten sind `$_GET`, `$_POST`, `$_SERVER`, `$_FILES`, `$_ENV`, `$_SESSION`, `$_COOKIE`.

6.4 Auslagern und Einbinden von PHP-Code

Es können externe PHP-Dateien eingebunden werden. Dies erfolgt mit den Befehlen `require_once` oder `include_once`. Die Wirkung beider Befehle ist identisch, zumindest solange kein Fehler beim Laden der Datei auftritt: `include_once` gibt in diesem Falle lediglich eine Fehlermeldung aus, während `require_once` das komplette Skript mit einem „fatalen Fehler“ abbricht.

Listing 6.1: Die PHP-Datei zu Beispiel 6.2

```
<!-- Dateiname: addition2.php -->
<h2>Aufsummierung natürlicher Zahlen (Arithmetische Reihe)</h2>
<?php
  require_once("./includes/addition2.inc.php");
  $zahl = 6;
  echo "Die Summe bis $zahl ist " . summeBis($zahl);
?>
```

Die Datei mit der zugehörigen eingebundenen Datei:

Listing 6.2: Die PHP-Datei zu Listing 6.1

```
<!-- Datei addition2.inc.php im Verzeichnis ./includes/ -->
<?php
   function summeBis($n) {
      return ($n + 1) * $n / 2;
   }
?>
```

Wir erkennen, dass in der eingefügten PHP-Datei ebenfalls die PHP-Tags stehen müssen. Die PHP-Datei heißt `addition2.inc.php` und befindet sich im Verzeichnis includes unterhalb des Verzeichnisses, in dem die HTML-Datei (`addition2.php`) beheimatet ist. Deshalb wird im `require_once`-Befehl der relative Pfad zu `addition2.inc.php` angegeben. Pfad- und Dateinamen werden in Anführungszeichen und runden Klammern eingeschlossen

Die Extension `.inc.php` ist nicht zwingend notwendig, hat sich aber eingebürgert. Das Kürzel `.inc` weist darauf hin, dass es sich um eine eingebettete Datei handelt, die weitere Extension `.php` ist dann aus Sicherheitsgründen notwendig. Dateien mit der Extension `.inc` werden, wenn sie alleine aufgerufen werden, vom Server nicht ausgeführt. Errät ein Benutzer aber Pfad und Namen einer solchen Datei, kann er dies als URL direkt im Browser eingeben; der Server führt den Code nicht aus, sondern überträgt ihn als Quelltext an den Browser, der ihn dann darstellt. Endet die Datei aber auf `.php`, wird sie auf jeden Fall vom Server ausgeführt.

`.inc.php`: Konvention für Dateiendung

Für eine Auslagerung von PHP-Programmen in eigene Dateien sprechen die folgenden Argumente.

- Allgemeine Funktionen, Variablen oder Standardeinstellungen können von mehreren PHP-Skripten verwendet werden. Beispielsweise kann man Standardeinstellungen wie User und Passwort für einen Datenbankzugriff als Datei auslagern und kann sie so flexibel und zentral verwalten.
- Die PHP-Datei mit dem Hauptprogramm wird kleiner und besser überschaubar.
- Die Erstellung von HTML- und PHP-Datei kann mit unterschiedlichen Werkzeugen erfolgen.

6.4.1 `require` und `include`

Es gibt zwei ähnliche Funktionen zum Einbinden von ausgelagerten Dateien, `include` und `require`. Sie werden häufig in kleinen Projekten oder einfachen Systemen verwendet. Sie haben dieselbe Wirkung wie die „`_once`“-Varianten, verhindern aber nicht eine Mehrfacheinbindung derselben Datei. In komplexen Systemen mit einer tiefen Einbindunghierarchie kann eine solche Mehrfacheinbindung schnell vorkommen; beispielsweise wird in dem Fall

A bindet B und C ein, B bindet C ein,

die Datei C von A zweimal eingebunden. Mit `require_once` und `include_once` wird genau das verhindert. Insbesondere in komplexeren Systemen sind letztere daher zu empfehlen.

6.5 Zeit- und Datumsfunktionen in PHP

Mehrere PHP-Funktionen arbeiten mit Datums- und Uhrzeitangaben. Meistens erzeugen sie einen Unix-Zeitstempel *(UNIX timestamp)* oder formatieren einen UNIX Zeitstempel so, dass er für Menschen lesbar wird.

Ein Unix-Zeitstempel (oder „Systemzeit“) zeigt die Anzahl der Sekunden an, die seit dem (willkürlich gewähltem) Datum 1.1.1970, 0:0:0 Uhr Greenwich Mean Time vergangen sind. Die meisten Systeme stellen einen Zeitstempel mit einem vorzeichenbehafteten 32-Bit-Integer dar, was einem Datumsumfang vom 13.12.1901 bis 19.1.2038 entspricht. Bei einem Datum außerhalb dieser Bereiche kann es also Probleme geben.

UNIX-Zeitstempel

6.5.1 Erzeugen eines Zeitstempels

Es gibt im wesentlichen zwei Funktionen, die in PHP einen Zeitstempel erzeugen. Die erste,

```
int time()
```

wird stets ohne Parameter aufgerufen und liefert den Zeitstempel der aktuellen Systemzeit des Servers. Um den Zeitstempel eines bestimmten Datums zu erzeugen, gibt es die Funktion

```
int mktime(int stunde, int min, int sek, int monat, int tag, int jahr)
```

Sie wird in der amerikanischen Datumsschreibweise (Monat/Tag/Jahr) aufgerufen. (PHP findet selbständig heraus, ob es sich um Winter- oder Sommerzeit handelt.) Will man also den Zeitstempel des 23. Februar 2008, 12 Uhr mittags (Winterzeit) erzeugen, so programmiert man

```
$zeit = mktime(12, 0, 0, 2, 23, 2008); // ergibt $zeit = 1203764400
```

Die Funktion geht auch mit Parameterwerten korrekt um, die eigentlich kein korrektes Datum sind, beispielsweise liefert `mktime(0, 0, 0, 2, 30, 2008)` die Systemzeit des 1.3.2008 0:0:0 Uhr, oder `mktime(26, 0, 0, 14, 29, 2004)` die vom 2.3.2005 2:0:0 Uhr. Auf diese Weise kann man gut Fristen oder Zeiträume addieren oder subtrahieren:

```
$frist = 10; // 10 Tage Frist
$fristEnde = mktime(12, 0, 0, 2, 23 + $frist, 2008); // ergibt 1204628400
```

6.5.2 Formatierte Zeitangaben

Der Unix-Zeitstempel ist sehr praktisch zum Programmieren, aber er hat kein brauchbares Anzeigeformat. Die Funktion `date` liefert eine formatierte Ausgabe des Zeitstempels:

`int date(string` *Formatmuster*`, int` *zeitstempel*`)`

Hierbei ist das Formatmuster ein String, in dem festgelegte Buchstaben als Parameter übergeben werden, beispielsweise erzeugt

```
$zeit = mktime(10, 0, 0, 5, 20, 1964);
echo date("D d.m.Y u\m H:i:s \U\h\\r", $zeit); //=> "Wed 20.05.1964 um 10:00:00 Uhr"
```

Soll ein Buchstabe ausgegeben werden, der einen Parameter darstellt (wie `m`, `U`, `h` oder `r`), so muss ein Backslash verwendet werden. Ein ganz fieser Sonderfall sind jedoch die Buchstaben `r` und `n` in diesem Zusammenhang: `\r` und `\n` sind Escape-Zeichen, nämlich die Steuerzeichen für Rücksprung und Zeilenumbruch, und daher müssen sie mit *zwei* Schrägern versehen werden: `\\r` und `\\n`.

`\\r` und `\\n` für r und n

Eine vollständige Liste aller Parameter für das Formatmuster finden Sie im PHP-Manual online unter

`http://www.php.net/manual/de/function.date.php`

7 Arrays

Kapitelübersicht

Arrays gehören zu den wichtigsten Datenstrukturen der Informatik überhaupt. Ein *Array* (dt. „Ansammlung“, „Anordnung“) ist eine indizierte Anordnung von Datenelementen oder *Einträgen*. D.h., man kann auf einen Eintrag des Arrays, also dessen Dateninhalt, mit Hilfe eines Indexes zugreifen. Ein Array ist also kein einzelner Wert, sondern eine Liste von Werten oder Elementen. Z.B. kann man sich ein Array namens `$farbe` wie folgt vorstellen.

`$farbe`	rot	grün	blau	gelb
Index	0	1	2	3

(7.1)

Wie in den meisten modernen Programmiersprachen ist der erste Array-Index 0, d.h. die Indizes eines Arrays `$daten` mit 4 Einträgen laufen von 0 bis 3,

```
$farbe[0], $farbe[1], $farbe[2], $farbe[3].
```

In PHP gibt es zwei verschiedene Arten von Arrays, indizierte (oder numerische) Arrays und assoziative Arrays. Ein assoziatives Array erweitert gewissermaßen den Begriff des Index und lässt einen Datenzugriff über Strings (und nicht nur numerische Werte) zu.

7.1 Indizierte Arrays

Ein *indiziertes* oder *numerisches Array* spricht seine Elemente über einen durchnummerierten Index an. Ein Array selber ist durch eine Variable, also ein Dollarzeichen `$` und einen Namen ansprechbar, also z.B. `$farbe`.

Ein Array kann in PHP durch das Schlüsselwort `array()` mit den runden Klammern erzeugt werden. Mit `$farbe = array()` wird also ein leeres Array erzeugt, dass dann im Verlauf des Programms gefüllt werden kann.

Um ein Element eines Arrays gezielt anzusprechen, wird sein Index in eckigen Klammern hinter den Arraynamen geschrieben, z.B. `$farbe[1] = "grün"`.

Beispiel 7.1. (3 Arten, Arrays zu bilden)

```
<!-- Ein Array von Farben. Dateiname: farben.php -->
<h2>Ein Array von Farben</h2>
<?php
   /* 1. Methode, um ein Farbe-Array zu bilden: */
   $farbe1 = array();
   $farbe1[] = "rot";
   $farbe1[] = "grün";
   $farbe1[] = "blau";
   $farbe1[] = "gelb";

   echo "farbe1:";
   echo "<table border=\"1\">";
   echo "  <tr> ";
   for( $i=0; $i < sizeof($farbe1); $i++ ) {
      echo "<td>$farbe1[$i]</td>";
   }
   echo "  </tr>";
   echo "</table>";

   /* 2. Methode, um ein Farbe-Array zu bilden: */
   $farbe2 = array("rot", "grün", "blau", "gelb");

   echo "<hr/>farbe2:";
   echo "<table border='1'>";
   echo "  <tr> ";
   foreach ( $farbe2 as $element ) {
      echo "<td>$element</td>";
   }
   echo "  </tr>";
   echo "</table>";

   /* 3. Methode, um ein Farbe-Array zu bilden: */
   $farbe3 = ["rot", "grün", "blau", "gelb"];

   echo "<hr/>farbe3:";
   echo "<table border='1'>";
   echo "  <tr> ";
   foreach ( $farbe3 as $element ) {
      echo "<td>$element</td>";
   }
   echo "  </tr>";
   echo "</table>";
?>
```

□

Die Browseranzeige des Skripts sehen Sie in Abb. 7.1. Das Skript zeigt drei Arten, ein

Abbildung 7.1: Die Browseranzeige von Beispiel 7.1.

numerisches Array zu bilden, sowie zwei Wege, ein Array mit einer Schleife zu durchlaufen. In jeder dieser Schleifen erstellen die `echo`-Befehle eine einzeilige Tabelle, in deren Zellen jeweils ein Array-Eintrag steht. Man erkennt die erste Art, ein Array zu bilden:

```
$farbe1 = array();
$farbe1[] = "rot";
$farbe1[] = "grün";
$farbe1[] = "blau";
$farbe1[] = "gelb";
```

Beim Aufbau eines Arrays in PHP werden die Werte bzw. Elemente wie in Regalfächern von unten nach oben immer höher abgelegt. Die Indexnummer entspricht entspricht einer Fachnummer der Regale und wird automatisch beginnend bei 0 gezählt.

Der Aufbau eines Arrays mit expliziter Indexangabe ist natürlich ebenso möglich, also

```
$farbe1 = array();
$farbe1[0] = "rot";
$farbe1[1] = "grün";
$farbe1[2] = "blau";
$farbe1[3] = "gelb";
```

Da PHP die Indexverwaltung jedoch automatisch durchführen kann, braucht man diesen Aufbau normalerweise nicht zu machen. Auch könnten bei einer solchen manuellen Indexssteuerung Lücken oder doppelte Zuweisungen auftreten. Manchmal kann es aber sinnvoll sein, eine explizite Indexzuweisung zu machen, oder vielleicht ist dem Einen oder Anderen beim Programmieren wohler, wenn er die Indexzuordnung selber sehen kann.

Das gleiche Array wird in einer Anweisung erzeugt durch eine zweite Art:

```
$farbe2 = array("rot", "grün", "blau","gelb");
```

Dieser Aufbau ist sinnvoll, wenn die Werte aller Arrayelemente zum Zeitpunkt der Erzeugung bekannt sind. Eine weitere Variante, in diesem Fall ein Array zu erzeugen, ist wie folgt:

```
$farbe3 = ["rot", "grün", "blau", "gelb"];
```

Diese Variante mit den eckigen Klammern `[...]` ist die Syntax eines JSON-Arrays und als *„short syntax array“* bekannt.

Gleichgültig wie man ein Array aufgebaut hat, der Zugriff auf ein Element geschieht mit der Indexnummer; so ergäbe z.B. `echo $farbe1[3];` die Ausgabe *gelb*. Die Anzahl der Elemente eines Arrays, d.h. die Länge des Arrays, kann man mit der Funktion `sizeof` oder `count` bestimmen.

Die Ausgabe des Arrays `$farbe1` in dem Beispielskript geschieht mit einer `for`-Schleife:

```
for($i=0; $i < sizeof($farbe1); $i++) {
   echo "<td>$farbe1[$i]</td>";
}
```

Der Laufindex ist `$i`; er beginnt bei 0 und geht bis zum Ende des Arrays, das man mit der Standardfunktion `sizeof` herausfindet. Da die 0 mitgezählt wird, muss `$i` *vorher* aufhören (daher also `$i<sizeof($farbe1)` als Endebedingung).

Die Ausgabe des zweiten Arrays `$farbe2` zeigt eine weitere komfortablere Möglichkeit, Arrays in PHP zu verarbeiten, die `foreach`-Schleife.

7.2 Die `foreach`-Schleife

In dem Skriptbeispiel 7.1 ist eine spezielle Schleife, die `foreach`-Schleife. Sie ist speziell für Arrays anwendbar und hat einen „kürzeren“ Schleifenkopf als die für Arrays verwendete `for`-Schleife. Ihre Syntax für ein Array namens `$array` lautet:

```
foreach ($array as $element) {
  anweisungen;
}
```

Hierbei wird das Array `$array` von Anfang bis Ende durchlaufen und bei jedem Schleifendurchlauf wird das aktuelle Element der Variablen `$element` zugewiesen. Der Variablenname `$element` ist frei wählbar, er sollte allerdings keine existierende Variable überschreiben. Auf diese Weise können Sie den Ausgabeteil in Beispiel 7.1 leicht nachvollziehen. Auf eine wichtige Erweiterung der foreach-Schleife kommen wir auf Seite 77 zu sprechen.

7.3 Assoziative Arrays

Assoziative Arrays werden nicht über einen durchnummerierten Index verwaltet, sondern über Strings, so genannte *Schlüssel (keys)*. Dieser Schlüssel muss das jeweilige Element eindeutig beschreiben. Damit entsteht eine assoziative Verbindung zwischen Schlüssel und Wert.

Beispiel 7.2. (Adresse als assoziatives Array)

```
<!-- Ein Adressen als Array Dateiname: adresse1.php -->
<html>
<head><title>Adresse</title></head>
<body>
<h2>Eine Adresse als Array</h2>
<?php
   $adresse = array();
   $adresse["vorname"] = "Andreas";
   $adresse["nachname"] = "de Vries";
   $adresse["email"] = "de-vries@fh-swf.de";
   $adresse["web"] = "www2.fh-swf.de/fbtbw/deVries.htm";
   $adresse["strasse"] = "Haldener Straße 182";
   $adresse["plz"] = "D-58095";
   $adresse["ort"] = "Hagen";
```

```
   echo "<table border=\"1\"> \n";
   echo " <tr> \n";
   foreach ( $adresse as $key => $value ) {
      if ( $key == "email" ) {
         echo "   <td><a href=\"mailto:$value\">$value</a></td>\n";
      } else if ( $key == "web" ) {
         echo "   <td><a href=\"http://$value\">$value</a></td>\n";
      } else {
         echo "   <td>$value</td> \n";
      }
   }
   echo "  </tr>\n";
   echo "</table>";
?>
</body>
</html>
```

□

Das PHP-Skript ergibt die Ausgabe Abb. 7.2 im Browser. Interessant ist der von dem PHP-

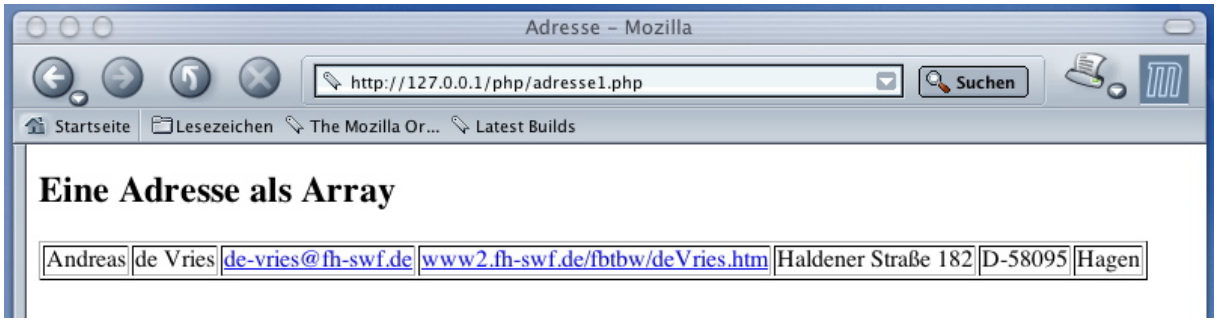


Abbildung 7.2: Die Browseranzeige von Beispiel 7.2.

Skript erzeugte HTML-Code in Abb. 7.3, den man in der Quelltextanzeige des Browsers sehen kann.

Abbildung 7.3: Die Browseranzeige des HTML-Quelltextes von Beispiel 7.2.

Aus dem Skript wird ersichtlich, wie man assoziative Arrays erzeugen kann, nämlich indem man einfach den Schlüssel in eckige Klammern nach dem Arraynamen setzt und die Werte der Elemente diesem zuordnet:

```
$adresse = array();
$adresse["vorname"] = "Andreas";
$adresse["nachname"] = "de Vries";
$adresse["email"] = "de-vries@fh-swf.de";
$adresse["web"] = "www2.fh-swf.de/fbtbw/deVries.htm";
```

Entsprechend kann auf jeden Array-Wert mit seinem Schlüssel zugegriffen werden:

`echo $adresse[nachname];` ergibt *de Vries*

Ähnlich der Array-Erzeugung eines numerischen Arrays in einem Schritt wie bei `$farbe2` kann man auch assoziative Arrays mit einer einzigen Anweisung erzeugen. Dazu wird der *Assoziationsoperator* verwendet:

```
$adresse = array(
             "vorname"  => "Andreas",
             "nachname" => "de Vries",
             "email"    => "de-vries@fh-swf.de",
             "web"      => "www2.fh-swf.de/fbtbw/deVries.htm",
             "strasse"  => "Haldener Straße 182",
             "plz"      => "D-58095",
             "ort"      => "Hagen"
           );
```

Alle Schlüssel-Wert-Paare *(key-value pairs)* werden nacheinander aufgeführt, d.h. das Aufbauprinzip ist stets

```
$myArray = array(
             "key_0" => "value_0",
                ...
             "key_n" => "value_n"
           );
```

Die `foreach`-Schleife kann man auch auf ein assoziatives Array anwenden. Zuende überlegt ist sie sogar *die einzige* Schleifenkonstruktion, die man zum Durchlaufen eines assoziativen Arrays verwenden kann! (Denn keine der anderen Schleifen hat bei nicht-indizierten Arrays eine Chance, an die Schlüssel zu kommen . . .) Zusätzlich hat man mit der foreach-Schleife sogar die Möglichkeit, auf die Schlüssel zuzugreifen, und zwar mit Hilfe des Assoziationsoperators:

Durchlaufen assoziativer Arrays nur mit Foreach

```
foreach ($myArray as $key => $element) {
  anweisungen;
}
```

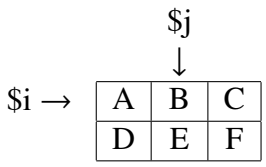
Wir werden im nächsten Skriptbeispiel eine Anwendung kennen lernen.

7.4 Mehrdimensionale Arrays

Mehrdimensionale Arrays sind Arrays in Arrays. Ein zweidimensionales Array kann man sich vorstellen als eine Tabelle oder Matrix, in der eine Zeile ein Array von Spalten ist, und eine Spalte wiederum ein Array von Zellen. Z.B. stellt das Array

```php
$tabelle = array(
              array("A", "B", "C"),
              array("D", "E", "F")
           );
```

eine 2×3-Tabelle dar:



Auf einen bestimmten Wert des Arrays greift man entsprechend mit mehreren Indizes zu, also hier

```
$tabelle[$i][$j].
```

Hierbei bezeichnet i die Nummer der Zeile (beginnend bei 0!) und j die Spalte, wie in der Tabelle angedeutet. D.h. in unserem Beispiel:

```
echo $tabelle[0][2];
```
ergibt C.

Man kann i und j auffassen als Koordinaten, die jeweils den Ort einer Datenzelle angeben, so ähnlich wie A-2 oder C-5 Koordinaten eines Schachbretts (oder des Spielfeldes von Schiffeversenken oder einer Excel-Tabelle usw.) sind. In einem assoziativen Array nehmen die Koordinaten die Werte der Schlüssel an.

Höherdimensionale Arrays werden entsprechend gebildet. Sie heißen *Tensoren*.[1] Ein dreidimensionales Array ist beispielsweise ein Array von Arrays von Arrays von Elementen; man kann es sich vorstellen als einen Quader von Würfeln als Zellen, oder als Array von Tabellen. Jede Zelle kann durch 3 Koordinaten angesprochen werden,

```
$tensor[$i][$j][$k]
```

Zwar kann man sich Arrays mit mehr als 3 Dimensionen nicht mehr anschaulich vorstellen, aber formal lassen sie sich entsprechend aufbauen. Man braucht dann genau so viele Indizes, wie man Dimensionen hat.

Das folgende Beispielskript erzeugt ein zweidimensionales Array von Adressen und gibt es als Tabelle aus, wie Abb. 7.4 zeigt. Zu beachten ist, dass der Tabellenkopf die Schlüssel des

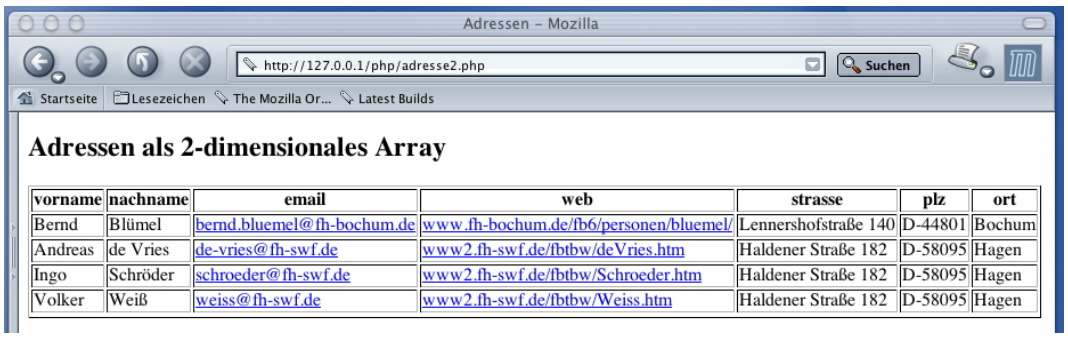


Abbildung 7.4: Die Browseranzeige von Beispiel 7.1.

inneren Arrays enthält. Das PHP-Skript bindet eine separate Datei ein, die das 2-dimensionale Array erzeugt.

[1] Der Begriff des *Tensors* ist in der Physik und den Ingenieurwissenschaften sehr geläufig.

Beispiel 7.3. Erzeugung eines zweidimensionalen Arrays in externer Datei

```php
<?php
   // Dateiname: adressen.inc.php
   /* Es wird ein zweidimensionales Array von Adressen aufgebaut */
   $adresse =
        array(
          array(
            "vorname"  => "Bernd",
            "nachname" => "Blümel",
            "email"    => "bernd.bluemel@fh-bochum.de",
            "web"      => "www.fh-bochum.de/fb6/personen/bluemel/",
            "strasse"  => "Lennershofstraße 140",
            "plz"      => "D-44801",
            "ort"      => "Bochum"
           ),
          array(
            "vorname"  => "Andreas",
            "nachname" => "de Vries",
            "email"    => "de-vries@fh-swf.de",
            "web"      => "www3.fh-swf.de/fbtbw/deVries.htm",
            "strasse"  => "Haldener Straße 182",
            "plz"      => "D-58095",
            "ort"      => "Hagen"
          ),
          array(
            "vorname"  => "Ingo",
            "nachname" => "Schröder",
            "email"    => "schroeder@fh-swf.de",
            "web"      => "www3.fh-swf.de/fbtbw/Schroeder.htm",
            "strasse"  => "Haldener Straße 182",
            "plz"      => "D-58095",
            "ort"      => "Hagen"
          ),
          array(
            "vorname"  => "Volker",
            "nachname" => "Weiß",
            "email"    => "weiss@fh-swf.de",
            "web"      => "www3.fh-swf.de/fbtbw/Weiss.htm",
            "strasse"  => "Haldener Straße 182",
            "plz"      => "D-58095",
            "ort"      => "Hagen"
          )
   );
?>
```

□

Der Source-Code des Skripts selber ist in Listing 7.1 angegeben.

Listing 7.1: Adressen als zweidimensionales Array

```
<!-- Adressen als 2-dimensionales Array. Dateiname: adresse2.php -->
```

```
<h2>Adressen als 2-dimensionales Array</h2>
  <table border="1">
<?php
   require_once( "./adressen.inc.php" );

   // Tabellenkopf:
   echo "   <tr>\n";
   foreach ( $adresse[0] as $key => $value ) {
      echo "    <th>$key</th>\n";
   }
   echo "   </tr>\n";

   // Tabelleneinträge:
   foreach ( $adresse as $adressEintrag ) {
     echo "  <tr>\n";
     foreach ( $adressEintrag as $key => $value ) {
       if ( $key == "email" ) {
          echo "    <td><a href=\"mailto:$value\">$value</a></td>\n";
       } else if ( $key == "web" ) {
          echo "    <td><a href=\"http://$value\">$value</a></td>\n";
       } else {
          echo "    <td>$value</td> \n";
       }
     }
     echo "   </tr>\n";
   }
?>
  </table>
```

Die inneren Arrays sind die assoziativen Adress-Arrays, die wir aus dem vorigen Abschnitt kennen. Das äußere Array `$adresse` ist ein numerisches Array. Direkt nach der Erzeugung des zweidimensionalen Arrays folgt seine Ausgabe. Dabei ist zu beachten, dass das `table`-Element vor den PHP-Markierungen geöffnet wurde.

```
   <body>
   <h2>Adressen als 2-dimensionales Array</h2>
     <table border="1">
       <tr>
   <?php
       ...
```

Der Tabellenkopf wird mit einer `foreach`-Schleife eingeleitet,

```
   foreach ( $adresse[0] as $key => $value ) {
      echo "   <th>$key</th>\n";
   }
   echo "</tr>\n<tr>\n";
```

Was passiert hier? Mit dem Index 0 wird auf das erste Element des Arrays `$adresse` zugegriffen, also eine Adresse, die ein assoziatives Array ist. Da wir für den Tabellenkopf die Schlüssel dieses Arrays benötigen, machen wir sie uns mit der „Maske“ `$adresse[0] as $key => $value` als Variable `$key` zugänglich – um sie dann als *table head* innerhalb der `<th>`-Tags auszugeben.

7.5 Nützliche Funktionen für Arrays

PHP liefert einige nützliche Standardfunktionen für Arrays, sie sind vollständig in der PHP-Dokumentation `http://php.net/manual/de/ref.array.php` verfügbar. Die wichtigsten dieser Funktionen werden in diesem Abschnitt beschrieben.

Beispielsweise mit `in_array` überprüfen, ob ein bestimmter Wert in einem Array existiert, oder mit `array_search`, welchen Schlüssel ein gegebener Wert in dem Array hat.[2]

7.5.1 Debuggen mit `print_r` oder `var_dump`

Sehr nützlich beim Programmieren mit Arrays in PHP sind die Funktionen `print_r`, und `var_dump`, die eine lesbare Kurzdarstellung von Arrays (und übrigens auch Objekten) liefern und direkt im Browserfenster ausgeben. Allerdings ist diese Ausgabe nicht in HTML formatiert, sondern in reinem Text, man sollte sie sich also im Browser als Quelltext anschauen. Ein kleines Programm zur Demonstration dazu lautet:

```
<?php
  $array = array("a", "b", "", "d", null, "f");
  print_r($array);
?>
```

7.5.2 Funktionen zum Arrayinhalt: `count`, `sizeof`, `empty`

Um die Größe eines Arrays zu bestimmen, gibt es die Funktionen `count` und `sizeof`. Zwar ist in PHP `sizeof` ein Alias von `count`, letztere also „das Original“, aber vom Sinn her klarer und auch in besserem Einklang mit ähnlichen Funktionen in anderen Programmiersprachen ist `sizeof` zu bevorzugen. Beide Funktionen zählen die eingetragenenen Elemente des Arrays, auch wenn diese ein leerer String oder `null` sind.

```
<?php
  $array = array("a", "b", "", "d", null, "f");
  echo "count: " . count($array) . ", sizeof: " . sizeof($array); // => 6
  if (empty($array)) {
     echo " leer!";
  } else {
     echo " gefuellt!";
  }
?>
```

Mit der Funktion `empty` kann man prüfen, ob das Array leer ist.

7.5.3 `explode` und `implode`

Zwei sehr nützliche Standardfunktionen für Arrays sind `explode` und `implode`, die einen String in ein Array umwandeln bzw. umgekehrt. Sie eignen sich insbesondere zum Export oder Import von `csv`-Dateien *(comma separated values)*, die von Excel und anderen Tabellenkalkulationsprogrammen erstellt oder gelesen werden können.

[2] `array_search` liefert `false` zurück, wenn der gesuchte Wert gar nicht in dem Array existiert. Da nun der gesuchte Wert durchaus mit dem Index 0 in dem Array sein könnte, in PHP jedoch `false` und 0 nur schwer zu unterscheiden sind, sollten Abfragen auf den Misserfolg der Suche unbedingt mit dem Identitätsoperator `===` erfolgen.

Funktion	Beispiel	Beschreibung
`explode()`	`$arr=explode(";",$str);`	zerlegt den übergebenen String (hier `$str`) an bestimmten Trennzeichen (hier Semikolons ";") und gibt das Ergebnis als Array zurück
`implode()`	`$str=implode(";", $arr);`	verbindet alle Array-Elemente des übergebenen Arrays (hier `$arr`) zu einem String, in dem sie durch die übergebenen Trennzeichen (hier Semikolons ";") unterteilt sind.

Beispiel 7.4. *(`implode` und `explode`)* Betrachten wir dazu das folgende Beispielprogramm.

```
<!-- implode und explode. Dateiname: imexplode.php -->
<?php
   // erstelle Adressen als csv-Format:
   require( "./adressen.inc.php" );
   foreach ($adresse as $adressEintrag ) {
      $adressEintrag = implode(";", $adressEintrag) . "\n";
      echo $adressEintrag;
   }
?>
```

In dem Skript wird unser zweidimensionales Adress-Array aus Beispiel 7.1 eingelesen, und in einer `foreach`-Schleife wird jeder Adresseintrag (wieder ein Array!) mit `implode` in das csv-Format konvertiert. Zu beachten ist, dass in diesem Format die Datenzellen einer Zeile standardmäßig durch ein Semikolon getrennt sind, während die Zeilen wiederum durch einen Zeilenumbruch (`\n`) getrennt sind. Die Ausgabe im Browser sieht hier nicht so gut aus, da wir keine HTML-Formatierungen ausgeben. Aber in der Quelltextanzeige des Browsers erkennt man gut die csv-Struktur. □

7.5.4 Sortierfunktionen

Für die Sortierung von Arrays stehen neun Funktionen zur Verfügung. Sortiert werden kann vorwärts (aufsteigend), rückwärts oder mit Hilfe einer individuell festgelegten Funktion. Für numerische Arrays, Schlüssel von assoziativen Arrays und Inhalte assoziativer Arrays werden unterschiedliche Funktionen verwendet.

Array	vorwärts	rückwärts	funktionsabhängig
numerisch	`sort($array)`	`rsort($array)`	`usort($array)`
assoziativ (Schlüssel)	`ksort($array)`	`krsort($array)`	`uksort($array)`
assoziativ (Inhalt)	`asort($array)`	`arsort($array)`	`uasort($array)`

Zu beachten ist, dass die Vorwärtssortierung Großbuchstaben vor Kleinbuchstaben sortiert, also Z < a. Daneben gibt es die Sortierfunktionen `natsort` und `natcasesort`: beide sortieren „natürlich“, d.h. Zahlen werden als Zahlen und nicht als Ziffernfolgen sortiert. `sort` sortiert 125 < 34, `natsort` und `natcasesort` sortieren 34 < 125. Sie werden wie `sort($array)` aufgerufen. `natcasesort` sortiert außerdem ohne Berücksichtigung der Groß- und Kleinschreibung.

Beispiel 7.5. *(Namen sortieren)*

```
<!DOCTYPE html>
<html><head><meta charset="UTF-8"/></head><body>
<!-- Dateiname: sortListe.php -->
<?php
```

```
    $liste = array("Schröder", "Blümel", "de Vries", "Weiß");
    echo implode(", ", $liste) . "<br/>";
    natcasesort($liste);
    echo implode(", ", $liste);
?>
</body></html>
```

In dem Beispielskript wird eine unsortierte Namensliste als String sortiert, indem sie zunächst mit `explode` in ein Array konvertiert wird, welches dann mit `natcasesort` sortiert und schließlich mit `implode` wieder in einen String verwandelt wird. □

7.6 Zusammenfassung

- Es gibt zwei Arten von Arrays in PHP, numerische Arrays à la `$farbe[0]`, `$farbe[1]`, ..., und assoziative Arrays, `$adresse["nachname"]`, `$adresse["vorname"]`, ..., ansprechbar über Schlüssel *(keys)*,
- Numerische Arrays können in einem Schritt erzeugt werden mit der `array`-Funktion

  ```
  $myArray = array("value_0", "value_1", ... );
  ```

  oder mit eckigen Klammern (der „short syntax“ Notation)

  ```
  $myArray = ["value_0", "value_1", ... ];
  ```

  Sie können aber auch sukzessive (als „Regalstapel“) gefüllt werden,

  ```
  $myArray = array();
  $myArray[] = "value_0";
  $myArray[] = "value_1";
   ...
  ```

  (wobei der Index auch auftreten kann, `$myArray[2] = "blau"];`).
- Assoziative Arrays werden entsprechend mit dem Assoziationsoperator `=>` durch

  ```
  $myArray = array(
               "key_0" => "value_0",
                  ...
               "key_n" => "value_n"
             );
  ```

  oder sukzessive durch

  ```
  $myArray = array();
  $myArray["key_0"] = "value_0";
  $myArray["key_1"] = "value_1";
   ...
  ```

- Sehr geeignet für Arrays ist die `foreach`-Schleife,

  ```
  foreach ($myArray as $element) {
    anweisungen;
  }
  ```

Auf die Schlüssel in einem assoziativen Array kann man mit dem Assoziationsoperator `=>` zugegriffen werden,

```
foreach ($myArray as $key => $element) {
  anweisungen;
}
```

- Zur Konvertierung eines Strings mit Trennzeichen in ein Array und zurück gibt es die beiden nützlichen Funktionen `explode` und `implode`.
- Es gibt eine Reihe nützlicher Sortierfunktionen für Arrays, (S. 82). Vorsicht bei der Sortierung von Zahlen mit `sort`, es sortiert 124 < 35 und Z < a. Abhilfe schaffen die speziellen Sortierfunktionen `natsort` (Sortierung „Zahlen-statt-Ziffern“) und `natcasesort` (Sortierung „Zahlen-statt-Ziffern“ + A < a < B).

8 Eingaben

Kapitelübersicht

Da die Eingabe von Daten über den Browser erfolgen muss und die Daten dann zum Webserver kommen müssen, benötigen wir einen Mechanismus, der die Eingabedaten über HTTP übermittelt. Diesen Mechanismus liefern HTML-Formulare (`<form`), die ihre Eingabedaten mit einem Submit-Button `<input type="submit"/>` an das im `action`-Attribut angegebene PHP-Skript schickt.

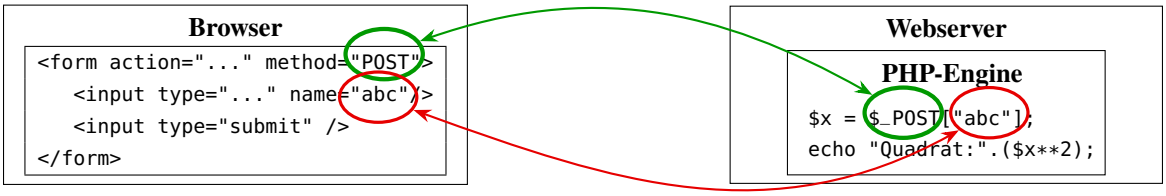
Hier werden nur die `<input>`-Elemente übertragen, die mit dem Attribut `name` einen Namen erhalten haben und die sich innerhalb desjenigen `<form>`-Elements befinden, in dem der verwendete Submit-Button `<input type="submit"/>` ist. Abhängig von der Übertragungsmethode `<form method=...>` sind die Daten in PHP über eine der folgenden „superglobalen Variablen“ verfügbar.

- `$_GET`: Diese Variable enthält die vom Browser übermittelten Daten einer `GET`-Anfrage über ein Formular mit dem Attribut `method="GET"` oder über den URL, wobei direkt hinter die Adresse des Skripts ein Fragezeichen gefolgt von durch ein Kaufmanns-Und (`&`) getrennte Schlüssel-Wert-Paare, also z.B.:

```
?polynom=x^2-4&x=2
```

- `$_POST`: Diese Variable enthält die vom Browser übermittelten Daten einer `POST`-Anfrage über ein Formular mit dem Attribut `method="POST"`. Hierbei werden die Daten vom Browser wie bei einer E-Mail an den HTTP-Request „angehängt“.
- `$_COOKIE`: Diese Variable enthält vom Browser übermittelte Cookie-Informationen.
- `$_FILES`: Diese Variable enthält Informationen über vom Browser per `POST`-Anfrage hochgeladene Dateien.

Alle diese Felder sind assoziative Arrays und enthalten die übermittelten Daten unter dem Schlüssel, der bei `$_GET` und `$_POST` von dem `<input>`-Tag des aufrufenden Formulars abhängt. In folgender Skizze ist die HTTP-Methode `"POST"` und das Formularfeld `"abc"` festgelegt:



Damit kann in PHP mit `$_POST["abc"]` auf die nach dem Abschicken übertragenen Daten zugegriffen werden.

Zusammengefasst wird also der URL im `action`-Attribut nach Drücken des Submit-Buttons mit den Formulardaten und der spezifizierten Methode dorthin. Wird `action` gar nicht angegeben, so ruft der Browser die Datei des Formulars erneut auf; wir werden diesen Mechanismus weiter unten genauer besprechen.

Betrachten wir das folgende Beispielskript, das von einem Formular über dessen `action`-Attribut aufgerufen wird. Es macht nichts anderes, als die Daten, die es empfangen hat, dem Browser zu spiegeln.

Beispiel 8.1. Ein einfaches Spiegelskript

```
<!-- Spiegelt die empfangenen Daten. Dateiname: request-response.php -->
<p>Folgende Daten wurden empfangen:</p>
<?php
  echo "\$_GET: <pre>"; print_r($_GET); echo "</pre>";
  echo "<br/>\$_POST: <pre>"; print_r($_POST); echo "</pre>";
?>
```

Hier wird mit `print_r` der Inhalt der `<input>`-Tags mit Namen ausgegeben, also z.B. im Array `$_GET`, wenn ein Formular mittels `<form ... method="GET">` mit der Methode GET abgeschickt wurde. □

Bei der Datenübertragung mit HTTPS übrigens werden die Parameterdaten aller HTTP-Methoden, insbesondere für GET, verschlüsselt; vgl. dazu

`https://https.cio.gov/faq/#what-information-does-https-protect.`

8.1 Eingaben in ein PHP-Skript

Als erstes einfaches Beispiel betrachten wir wieder eines unserer ersten Programme, die Addition von Zahlen. Der Benutzer soll über seinen Browser die Zahlen eingeben, unser Programm soll die Summe der eingegebenen Zahlen ausrechnen und das Ergebnis ausgeben. Wir erstellen zunächst eine HTML-Seite, die ein Formular enthält, in das wir die beiden Summanden eingeben können.

Beispiel 8.2. Eingabeformular für zwei Zahlen, die addiert werden sollen.

```
<!-- Datei addition-2.html -->
<form action="./addition-2.php" method="POST">
  x = <input type="text" name="x" size="3" /> <br/>
  y = <input type="text" name="y" size="3" /> <br/><br/>
  <input type="submit" value="Rechne!"/>
</form>
```

Das Klicken auf den Submit-Button führt dazu, dass der Inhalt der Eingabefelder an den Server übertragen wird, der Server das im `action`-Attribut des `form`-Tags angegebene Programm startet und diesem Programm die Inhalte der Eingabefelder übergibt. Bleibt also noch, diese PHP-Datei zu implementieren. □

Wie kommt PHP an die eingegebenen Zahlen? Dazu sind die Definition der beiden Eingabefelder in Beispiel 8.2 wesentlich:

```
    <input type="text" name="x" size="3"/>
    <input type="text" name="y" size="3"/>
```

Die `name`-Attribute der `input`-Tags geben den einzelnen Eingabefeldern die Namen, die als Schlüssel eines der superglobalen Arrays `$_GET` oder `$_POST`, abhängig von der Übertragungsmethode des Formulars, und mit den durch das Eingabefeld festgelegten Werte an das aufgerufene PHP-Skript übergeben werden. Bei der Übertragung des Formulars mit der `POST`-Methode liegen die Daten aller Input-Felder (außer den Buttons) in dem `$_POST`-Array vor:

`<input ... name="x" value="7"/>` ⟹ `$_POST["x"] = "7"`

Zusammengefasst erhält man also die Summe der einegegeben Zahlen mit dem Programm aus dem folgenden Beispiel.

Beispiel 8.3. Berechnung der Eingaben von Beispiel 8.2.

```
<!-- Datei addition-2.php -->
<?php
  if (isset($_POST["x"]) && isset($_POST["y"])) {
     $x = $_POST["x"];
     $y = $_POST["y"];
     echo "$x + $y = " . ($x + $y);
  }
?>
```

□

Um Interaktivität in PHP zu realisieren, müssen wir also offenbar zwei Dateien programmieren: eine Seite, in der die Benutzer ihre Eingaben machen können, und eine Seite, die die Eingaben verarbeitet. Tatsächlich geht es aber auch mit nur einer einzigen PHP-Datei, wie wir im nächsten Abschnitt 8.2 sehen werden.

8.2 Eingabe und Verarbeitung in einem einzigen Skript

Durch das `if`-Kommando können wir unsere PHP-Programme vereinfachen. Bislang haben wir, wenn wir in PHP Benutzereingaben lesen wollten, immer zwei Dateien benötigt: Eine mit einem Formular, in das wir unsere Daten eingeben konnten und eine, die dann die Aktionen

durchgeführt hat. Dies können wir nun ändern. Betrachten wir dazu das folgende Beispiel, in dem zwei einzugebende Zahlen dividiert werden, jedoch der Fall der Division durch 0 abgefangen wird.

Beispiel 8.4. Division in PHP, Eingabe und Ablauflogik in einem einzigen Skript.

```
<!-- Das Programm dividiert 2 Zahlen. Dateiname: division.php -->
<?php
   if(!empty($_POST)) { // Daten über POST, d.h. Formular abgeschickt?
      if($_POST['nenner'] == 0) { // Division durch 0 oder NaN?
         echo "Versuch durch Null zu teilen!";
      } else {
         $quotient = $_POST['zaehler'] / $_POST['nenner'];
         echo " $_POST[zaehler] / $_POST[nenner] = $quotient";
      }
   } else { // keine Daten über POST, d.h. das Skript wird erstmals aufgerufen
?>
    <form name="division" action="./division.php" method="post">
       Zaehler: <br/>
       <input type="text" name="zaehler" size="5"/> <br/>
       Nenner: <br/>
       <input type="text" name="nenner" size="5"/> <br/>
       <input type="submit" value="Abschicken"/>
    </form>
<?php
   }
?>
```

Neues beginnt schon in Zeile 3:

```
    if(!empty($_POST))
```

Hier wird festgestellt, ob aus dem Formular Daten in der `$_POST`-Variablen übermittelt wurden. Ist dies nicht der Fall, handelt es sich um den ersten Aufruf der Seite und das Array ist leer, was als Bedingung `false` bedeutet. Da der `if`-Zweig nur in dem Fall durchgeführt wird, wenn Daten in `$_POST` enthalten sind, können wir über die Namen der Input-Felder der Formulare auf die Eingaben der Benutzer zugreifen und die Berechnung durchführen. Im `else`-Zweig dagegen wird das Formular erzeugt. Die PHP-Anweisungen werden hier durch `?>` beendet, und es folgt reines HTML. Man kann PHP in einer Datei also beliebig oft an- und abschalten, sogar innerhalb von geschweiften Klammern. Die HTML-Zeilen erzeugen den Rest unseres Formulars, der Browser verarbeitet den Text original. Die Ausführung von PHP-Code setzt erst wieder mit den Zeilen

```
<?php
   }
```

ein. Die geschweifte Klammer schließt den `else`-Block ab und muss daher wieder im `<?php ... ?>`-Tag stehen. Natürlich kann man das gesamte Formular auch über `echo`-Befehle ausgeben, nur wird in diesem Fall der Quelltext unübersichtlicher, insbesondere wegen der vielen Anführungszeichen in HTML, die ja dann alle als `\"` geschrieben werden müssten (oder `'`, was aber nicht XML-konform ist).

In dem Formular nun ist das `action`-Attribut weggelassen, was zur Folge hat, dass das aktuelle Skript mit Submit aufgerufen wird. Dieselbe Wirkung hätten wir erreicht, wenn wir stattdessen den Namen des Skripts angegeben hätten, also hier `action="./division.php"`. Wir

können auch PHP dynamisch herausfinden lassen, was das aktuelle Skript ist, und zwar mit der Anweisung

```
<form action="<?php echo $_SERVER['PHP_SELF'];?>" method="POST">
```

Machen wir uns noch einmal den zeitlichen Ablauf der Anwendung klar. Wird das PHP-Skript zum ersten Mal aufgerufen, so sind in `$_POST` noch gar keine Daten, und es wird der `else`-Zweig ausgeführt. Schickt der Anwender dagegen das Formular ab, so enthält `$_POST` wegen `method = "post"` nun Daten (übrigens auch, wenn die Werte alle leere Strings sein sollten), und da im `action`-Attribut des Formulars das Skript selbst angegeben ist, ruft es sich selber wieder auf. d.h. der `if`-Zweig wird durchgeführt und der Quotient wird berechnet. □

8.2.1 Euro-Dollar-Umrechnung

Wir wollen nun ein Programm schreiben, das Eurobeträge in Dollarbeträge und umgekehrt umrechnet. In das Eingabeformular soll der Anwender die Zielwährung aus einer `select`-Box auswählen können, nach dem Abschicken soll dann das Ergebnis der Umrechnung angezeigt werden, und das Formular soll eine erneute Eingabe ermöglichen. Die Lösung des Problems in PHP ist in Beispiel 8.5 dargestellt.

Beispiel 8.5. Umrechnung von Euro in Dollar und umgekehrt.

```
<!-- Euro-Dollar Umrechnung. Dateiname: euro.php //->
<form action="<?php echo $_SERVER['PHP_SELF'];?>" method="POST">
  Kurs: <br/>
  <input type="text" name="kurs" value="1.10"/> ($ / &euro;)
  <br/>
  Betrag: <br/>
  <input type="text" name="betrag"/>
  Zielwaehrung:
  <select name="zielwaehrung">
    <option value="euro" selected="selected">&euro;</option>
    <option value="dollar">$</option>
  </select>
  <br/>
  <input type="submit" value="Abschicken"/>
</form>

<?php
if (isset($_POST['betrag']) && $_POST['betrag'] != "") {  // Betrag übermittelt?
   $kurs = $_POST['kurs'];
   if ($_POST['zielwaehrung'] == "dollar") {
      $zielbetrag = round( $kurs * $_POST['betrag'], 2);
      $zielsymbol = "\$";
      $symbol = "&euro;";
   } else if ($_POST['zielwaehrung'] == "euro" && $kurs != 0) {
      $zielbetrag = round( (1/$kurs) * $_POST['betrag'], 2);
      $symbol = "\$";
      $zielsymbol = "&euro;";
   } else {
      $zielbetrag = "- kein Umrechnungskurs angegeben -";;
      $symbol = "\$";
```

```
        $zielsymbol = "&euro;";
    }
    echo "$_POST[betrag] $symbol entsprechen $zielbetrag $zielsymbol";
}
  ?>
```

Hier wird der Dateiname des aufzurufenden Skripts von PHP ermittelt, und zwar mit der superglobalen Variablen `$_SERVER?`. Das ist ein Array und enthält recht viele Informationen zur Anfrage, zum Skript, zu den Pfaden und zu den Headern. Den Pfad des PHP-Skriptes ermittelt man dabei, indem man den Array-Eintrag `PHP_SELF` aus `$_SERVER` in eckigen Klammern aufruft,

```
$_SERVER['PHP_SELF']
```

□

Die wichtigsten Array-Einträge von `$_SERVER` sind:

- `HTTP_HOST`: Server-URL, z.B. `haegar.fh-swf.de`
- `HTTP_USER_AGENT`: Informationen zum anfragenden Browser, z.B. Mozilla/4.0 ...
- `PHP_SELF`: Der Dateiname des aktuell ausgeführten PHP-Skripts relativ zum Homeverzeichnis der Website, z.B. `/programme/euro.php`
- `REMOTE_ADDR`: IP-Adresse der Anfrage, z.B. 193.132.49.193
- `REQUEST_METHOD`: Methode der Anfrage, z.B. `POST`
- `SERVER_ADDR`: IP-Adresse des Webservers, z.B. 194.94.2.20
- `SERVER_PORT`: Port der Anfrage, normalerweise 80 (`http`) bzw. 443 (`https`)
- `SERVER_SOFTWARE`: verwendete Webserver-Software, z.B. Apache/1.3.27 (Unix)

8.3 Eingabe und formatierte Ausgabe von Kommazahlen

Kommazahlen haben bei Ein- und Ausgaben zweierlei Arten von Problemen zur Folge. Einerseits ist ihr internes Datenformat, wie in allen Programmiersprachen, mit einem Punkt als Bruchtrennzeichen, also `3.1415`, aber die Eingabe von Kommazahlen geschieht oft mit dem Komma. Andererseits sollen sie oft entsprechend mit Komma ausgegeben werden, zusätzlich aber auch mit einer festen Nachkommastellen, beispielsweise bei Geldbeträgen mit 2 Nachkommastellen, bei großen Beträgen mit dem Punkt oder Leerzeichen als Tausendertrenner.

Zur Konvertierung einer Float-Zahl, die mit einem Komma eingegeben wurde, verwendet man zweckmäßigerweise die Stringfunktion `str_replace`

```
string str_replace(nadel, ersatz, $heuhaufen)
```

Die Eingabeparameter sind Strings oder Arrays, wobei darauf zu achten ist, dass `nadel` und `ersatz` gleiche Struktur haben. Für Strings ist der erste Parameter der zu ersetzende Teilstring, der zweite die stattdessen erwünschte Ersetzung, und der dritte der String, in dem ersetzt werden soll. Es werden also alle Vorkommen von `nadel` in `heuhaufen` durch `ersatz` ersetzt. So kann beispielweise ein eingegebenes Komma in einer Zahl zu einem Punkt gewandelt werden:

```
$x = str_replace(",", ".", "3,1415")  // ergibt $x = 3.1415
```

Ob der Anwender jedoch überhaupt eine Zahl eingegeben hat, kann diese Funktion natürlich nicht prüfen, das kann nur `is_numeric`. Die Variante mit Arrays als Eingabeparameter kann dazu dienen, mit nur einem Aufruf alle Eingabefelder zu verändern:

```
$nadeln = array(",", " ");
$ersatz = array(".", "_");
$_POST = str_replace($nadeln, $ersatz, $_POST);
```

Dabei sollten die zu ersetzenden Strings in dem `$nadeln`-Array in entsprechender Reihenfolge mit den ersetzenden Strings in `$ersatz` stehen. Auf diese Weise wird in jedem Eintrag von `$_POST` jedes Komma durch einen Punkt und jedes Leerzeichen durch einen Unterstrich ersetzt.

Zur formatierten Ausgabe von Kommazahlen stellt PHP die Funktion `number_format` zur Verfügung,

`string number_format( float` *x*, `int` *stellen*, `string` *dezimaltrenner*, `string` *tausendtrenner*)

Der erste Parameter `x` ist die zu formatierende Kommazahl, danach kommt als int die Anzahl der Nachkommastellen, dann jeweils als String das Dezimal- und das Tausendertrennzeichen. Beispielsweise ergibt

```
$x = 3.1415;
number_format($x, 3, ",", ".");  // ergibt: "3,142"
```

8.3.1 Datenübertragung mit hidden Fields

Das spezielle `<input>`-Element mit `type="hidden"` wird *hidden Field* genannt und bezeichnet in einem Formular ein Datenfeld, das im Browser nicht angezeigt wird, dessen Wert aber wie alle anderen Eingabefelder an den Server übermittelt wird. Dazu muss es einen innerhalb des Formulars eindeutigen Namen und mit dem Attribut `value` einen Wert haben:

```
<form action="http://haegar.fh-swf.de/spiegel.php" method="POST">
  <input type="hidden" name="abc" value="123" />
  <input type="submit" />
</form>
```

Hier sehen wir im Browser nur den Submit-Button, nach dem Abschicken wird aber das Feld `abc` mit dem Wert `123` an den Server geschickt und kann mit `$_POST["abc"]` in PHP verarbeitet werden.

Ein hidden Field wird also nicht im Browser angezeigt und kann auch nicht verändert werden. (Allerdings sieht man es in der Quelltextanzeige der Seite im Browser.) Es ist daher eigentlich kein Eingabefeld. Wofür braucht man dann also ein solches Feld? Es wird sich als sehr praktisch erweisen, wenn für aufeinanderfolgende Datenbanken Nutzdaten mit den anderen Eingabedaten übertragen werden müssen. Bestätigt beipielsweise ein Kunde seinen Einkauf, so kann die Kunden-ID aus der Datenbank verborgen in das Formular eingefügt werden und würde damit bei der Bestätigung mit an den Server geschickt. So kann der Einkauf eindeutig und effizient dem Kunden zugeordnet werden.

Zweck von hidden Fields

Wichtig zu bemerken ist, dass allerdings niemals sicherheitsrelevante Informationen in einem hidden Field gespeichert werden, denn mit der Seitenquelltextfunktion des Browsers können sie immer angezeigt und mit den Entwickler-Tools verändert werden.[1]

[1] vgl. `https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input/hidden`

8.4 Zusammenfassung

- Benutzer-Eingaben können nur über ein HTML-Formular erfolgen und mit den superglobalen Variablen `$_GET` und `$_POST` ermittelt werden, abhängig von der Aktionsmethode des Formulars.
- Dabei wird der Wert eines Input-Feldes durch Anhängen seines Namens in eckigen Klammern an die Variable abgegriffen.
- Man braucht nicht unbedingt zwei Dateien (eine HTML-Datei für das Formular und eine PHP-Datei für die Verarbeitung der Eingabedaten), sondern kann sie in einer einzigen PHP-Datei zusammenfassen. Allerdings muss man dazu geeignete Einträge des `$_GET` bzw. des `$_POST`-Arrays abfragen, um den Ablauf des zu steuern und zu entscheiden, ob das Skript erstmalig aufgerufen wurde oder ob bereits Eingabedaten zur Verarbeitung übermittelt wurden.
- Um sich den aktuellen Namen des Skripts für diesen Fall automatisch in das `action`-Attribut des Eingabeformulars schreiben zu lassen, kann man den Wert von

  ```
  $_SERVER["PHP_SELF"]
  ```

  verwenden, oder das `action`-Attribut einfach ganz weglassen.

9
Cookies und Sessions

Kapitelübersicht

9.1 Cookies

Cookies sind eine Möglichkeit der HTTP-Protokolls, Daten des Webservers auf dem Client-Rechner zu speichern und bei einem späteren Aufruf wieder abzurufen. Dieser Mechanismus verletzt in gewisser Weise also das Grundpinzip von HTTP, denn der Client bekommt nun nicht nur die Informationen und Daten, die er angefragt hat, sondern der Server speichert Daten willkürlich auf dem Client. Allerdings sind Cookies Bestandteil des Protokollstandards und werden sehr häufig verwendet.

Die Einsatzzwecke von Cookies sind vielfältig. Mit ihnen kann ein Programm auf dem Webserver bei dem aufrufenden Browser die bei vorherigen Besuchen gespeicherten Daten abrufen und aktualisieren. Zum Beispiel ermöglicht dies ein Webtracking, also ein Nachverfolgen der Aufrufe wiederkehrender Besucher*innen einer Website, indem geeignet gestaltete Daten serverseitig in einer Datenbank gespeichert werden. Neben anderen Techniken können damit Unternehmen Marketingaktionen starten, z. B. Empfehlungslisten anhand früher betrachteter Produkte oder Dienstleistungen. Ein weiteres wichtiges Einsatzgebiet von Cookies sind das Speichern von Produkten in virtuellen Einkaufskörben. Aus Gründen des Datenschutzes und des in Deutschland geltenden Rechts auf informationelle Selbstbestimmung und gemäß der in der EU geltenden Datenschutz-Grundverordnung (DVGO) muss dem Einsatz von Cookies immer explizit zugestimmt werden.

Wir werden in diesem Kapitel betrachten, wie Cookies mit PHP realisiert werden können.

9.1.1 Cookies und HTTP

Ein *Cookie* (Keks, Plätzchen) bzw. *Magic Cookie* ist eine kleine Datei, üblicherweise ASCII-Text, mit der ein Programm Daten seines aktuellen Zustands auf einem anderen Rechner speichert, um diesen Zustand bei einen späteren Aufruf wiederherstellen zu können. Mit den gespeicherten Daten des Cookies kann nur das Programm etwas anfangen, das es gesetzt hat. Man kann ein Cookie vergleichen mit der Garderobenmarke einer öffentlichen Garderobe in Museen, Theatern oder Konzertsäälen, das ein Besucher bei der Abgabe seines Mantels erhält. Mit der Information auf der Marke kann nur das Garderobenpersonal etwas anfangen, am Ende des Besuchs kann es damit den korrekten Mantel zurück geben.

Entsprechend ist ein *HTTP-Cookie* ein Cookie, das der Webserver nach einem Aufruf einer Webseite als Schlüssel-Wert-Paar `name=content` durch den Browser auf dem Clientrechner speichern lässt und das der Browser bei einem späteren HTTP-Request an den Server sendet. Der

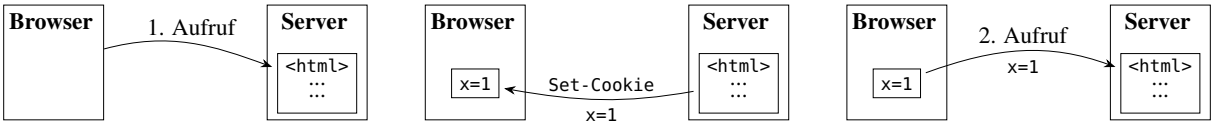


Abbildung 9.1: Prinzipieller Ablauf einer Cookie-Speicherung auf dem Browser: Nach dem ersten Aufruf einer geeignet programmierten Webseite speichert die HTTP-Response ein Cookie im Browser, hier `x=1`. Bei einem späteren Aufruf der Seite wird das Cookie dann zum Server übertragen.

prinzipielle Ablauf des Setzens und Übermittelns eines Cookies ist in Abbildung 9.1 schematisch skizziert.

So kann ein HTTP-Cookie unter Anderem dazu dienen, dem Anwender eine Sitzung (Session) zu ermöglichen (siehe S. 96). Prinzipiell können Cookies jedoch auch dazu verwendet werden, ein Profil des Internetverhaltens eines Anwenders zu bestimmen. So erstellen beispielsweise Google und Facebook individuelle Identitätsnummern (`PREF` oder `datr`, siehe Abbildung

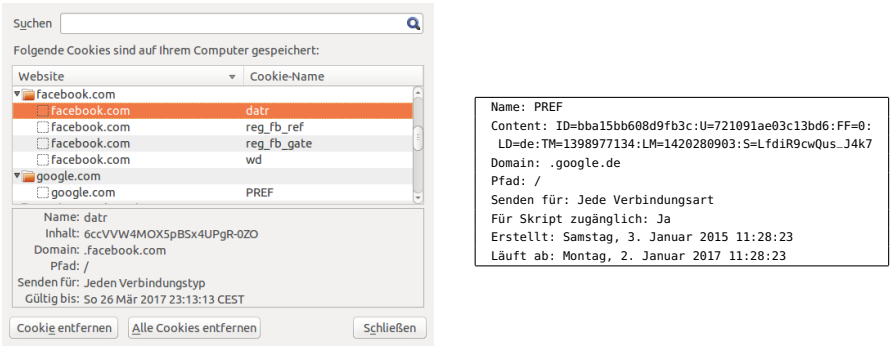


Abbildung 9.2: Cookies von Facebook und Google, im Firefox (links) und im Chrome (rechts). Die Cookiedaten sind in beiden Browsern über (Erweiterte) Einstellungen → Datenschutz einsehbar.

9.2), die bei jedem Aufruf des jeweiligen Servers übermittelt werden und somit dort eine eindeutige Spur der aufgerufenen Seiten über die Zeit hinterlassen.

Die Übermittlung von Cookiedaten an den Webserver geschieht ohne die explizite Betätigung des Anwenders, allerdings werden Cookies generell vom Browser verwaltet. So sind Cookies nicht versteckt oder geheim, sondern können vom Anwender über seinen Browser jederzeit eingesehen und auch gelöscht werden (Abbildung 9.2). Auch kann in den Datenschutzeinstellungen des Browsers das Speichern von Cookies teilweise oder komplett abgeschaltet werden. Das Verhalten von Google und Facebook ist demnach laut der aktuellen Rechtsprechung in Europa und den USA nicht rechtswidrig.

Spezifikation von HTTP-Cookies

„An HTTP cookie is opaque data that can be assigned by the origin server to a user agent by including it within a Set-Cookie response header field, with the intention being that the user agent should include the same cookie on all future requests to that server until it is replaced or expires.“[1] Gemäß RFC 6265 (`http://tools.ietf.org/html/rfc6265`) der IETF enthält ein Cookie Text einer Länge von maximal 4 Kilobyte (4096 Byte) und besteht aus einem Namen und Attributen, also Schlüssel-Wert-Paaren. Bei der Definition eines Cookies muss mindestens ein Attribut angegeben werden.

```
"Set-Cookie: Name=" Wert (";" key "=" value)+
```

`Name` und `Wert` sind Folgen von ASCII-Zeichen, wobei einige Sonderzeichen ausgeschlossen sind. Die Syntax von `Name` verwendet einen eingeschränkten Zeichensatz, wie er auch bei anderen HTTP-Kopfzeilen gemäß RFC 2616 verwendet wird. Für `Wert` sind Semikolon, Komma, Leerraum-Zeichen und Backslash als Zeichen ausgeschlossen. Um beliebige Daten als Cookie-Wert zu speichern, kann eine Kodierung wie Base64 oder die URL-Kodierung mit `%xx` verwendet werden. Im linken Beispiel in Abbildung 9.2 hat der Domainserver `facebook.com` zur Definition des Cookies den folgenden HTTP-Header gesendet:

```
Set-Cookie: Name=datr;Content=6ccVVW4MOX5pBSx4UPgR-0ZO;
            Domain=facebook.com;
            Path=/;Expires=Sun, 26-Mar-2017 23:13:13 CEST
```

Daraufhin hat der Browser das Cookie lokal gespeichert.

Das Attribut `HttpOnly` soll den Zugriff auf Cookies mittels JavaScript verhindern. Dies stellt einen möglichen Schutz gegenüber XSS dar, sofern der jeweils genutzte Browser dieses Attribut unterstützt.

Der Browser Firefox stellt Add-On zum Editieren von Cookies bereit, beispielsweise *Cookies Manager+*: `https://addons.mozilla.org/de/firefox/addon/92079`

9.1.2 Cookies in PHP

Cookies werden in PHP über das superglobale assoziative Array `$_COOKIE` implementiert. Ein Cookie wird mit der Funktion

`setcookie(`*Name*`,`*Wert*`,`*Verfallszeit*`,`*Pfad*`,`*Domäne*`,`*https*`,`*httponly*`)`

gesetzt, wobei die ersten beiden und der vierte Parameter Strings sind und nur der erste obligatorisch ist. *Name* und *Wert* bezeichnen hier den Inhalt des Cookies, *Verfallszeit* ist ein UNIX-Zeitstempel (Abschnitt 6.5), *Pfad* und *Domain* bestimmen den Gültigkeitsbereich des Cookies, (standardmäßig wird der aufgerufene URI verwendet), *https* ist ein Boole'scher Wert, der bei `true` eine HTTPS-Verbindung erfordert, und *httponly* ist ein Boole'scher Wert, der bei `true` nur über das HTTP(S)-Protokoll ausgelesen wird und nicht beispielsweise per JavaScript. Eine mögliche Wertbelegung aller Parameter ist wie folgt:

```
setcookie("Sprache", "englisch", time()+3600);
```

Hier bedeutet der Eintrag `time()+3600`, dass die Systemzeit des Servers in Sekunden UNIX-Zeitstempel plus 1 Stunde eingestellt wird, d. h. das Cookie gilt nur noch eine Stunde. In Listing 9.1 wird das Setzen und das Auslesen von Cookies an einem einfachen Beispiel gezeigt.

[1] Fielding (2000):§6.3.4.2.

Listing 9.1: Cookies setzen und auslesen

```
<?php
  setcookie("Autor", "de Vries");
  setcookie("Titel", "WebTech");

  if (isset($_COOKIE['Autor'])) {
    echo $_COOKIE['Autor'] . ": " . $_COOKIE['Titel'];
  } else {
    echo " -- kein Cookie! --";
  }
?>
```

Es ist bei der Programmierung von Cookies sehr hilfreich, sich die jeweils aktuellen Werte der existierenden Cookies mit den Anweisungen

```
echo "<br/>\$_COOKIE: <pre>"; print_r($_COOKIE); echo "</pre>";
```

am Ende des PHP-Skripts anzeigen zu lassen.

Löschen eines Cookies

Wie löscht man ein Cookie? Das Löschen von Cookies ist in PHP grundsätzlich nicht möglich. Die Ursache ist, dass PHP ja auf dem Server läuft, das Cookie aber auf dem Browser gespeichert ist. Durch den in Abbildung 9.1 skizzierten Mechanismus kann man mit PHP allerdings ein eventuell existierendes Cookie auf dem Browser überschreiben, indem eines mit gleichem Namen gesetzt wird. Damit hat man es eigentlich zwar nicht gelöscht; setzt man allerdings die Verfallszeit des neuen Cookies auf irgendeinen positiven Wert echt kleiner als die aktuelle (Browser-) Zeit, so wird es beim nächsten Aufruf der Seite sofort gelöscht. Möchten wir beispielsweise das Cookie `"test_cookie"` auf dem Browser „löschen“, so erreichen wir dies mit dem Befehl

```
setcookie("test_cookie", "Stirb", 1);
```

Hier wurde die Verfallszeit auf den 1.1.1970 um 0:00:01 Uhr gesetzt. Das Cookie wird also beim nächsten Aufruf der Seite sofort gelöscht, egal in welcher Zeitzone der Erde sich Browser und Server jeweils befinden.

9.2 Sessions

In diesem Kapitel werden wir eine spezielle Technik von PHP behandeln, um stehende Verbindungen eines Browseranwenders zu implementieren. Solche Verbindungen, sogenannte Sessions oder Sitzungen, ermöglichen es, Zustände der Verbindung über ihre Lebensspanne über beliebig viele Aufrufe und Eingaben der PHP-Seiten zu speichern. Die Verwaltung der Session übernimmt hier der Webserver über eine eindeutige Session-ID. Wir werden genauer betrachten, auf welche Weisen diese für die Sitzung notwendige ID über ihren Verlauf gespeichert werden kann.

Wofür werden Sessions überhaupt gebraucht? Stehende Verbindungen sind immer notwendig, wenn eine Kommunikation oder ein mehrschrittiger Prozess programmiert werden soll. Ein typisches Beispiel aus dem Alltag dafür ist ein Telefonanruf: Hier wird der Verbindungsaufbau mit dem Wählen der Nummer begonnen, die Verbindung selbst mit dem Abheben des Angewählten geöffnet und mit dem Auflegen eines der Beteiligten beendet. Wichtige Beispiele aus dem Bereich der Internettechnologien sind entsprechend Log-In-Prozesse oder Einkäufe mit virtuellen Warenkörben.

Da wir eine Sitzung neben PHP-Sessions grundsätzlich auch mit Cookies ermöglichen könnten, werden wir dieses Kapitel in einer Gegenüberstellung von Cookies und Sessions

in PHP beschließen. Dabei wird sich ergeben, dass PHP-Sessions einfacher und intuitiver zu programmieren sind als Cookies, da man mit den sogenannten „Sessionvariablen“ fast wie mit normalen Variablen arbeiten kann.

9.2.1 Eigenschaften von Sitzungen

Eine *Session* oder *Sitzung* bezeichnet in der Informatik eine stehende Verbindung zwischen zwei oder mehreren Kommunikationspartnern. In einem Client-Server-System besteht eine Sitzung üblicherweise zwischen einem Client und einem Server, also im Internet zwischen einem Browser und einem Webserver. Der Anfang einer Sitzung geschieht dabei über ein explizites Login des Clients oder einfach mit dem Aufruf einer Seite. Während der Sitzung können Informatio-

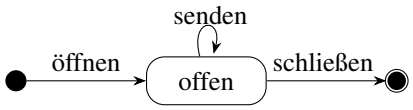


Abbildung 9.3: Zustandsdiagramm einer Sitzung.

nen zwischen den Kommunikationspartnern übermittelt werden. Eine Sitzung kann durch eine vorgegebene maximale Sitzungsdauer zeitlich begrenzt sein, wird üblicherweise aber von einem der Kommunikationspartner beendet, beispielsweise durch einen Logout. Ein expliziter Logout einer Browsersitzung geht dabei meist vom Client aus, bei zeitlich begrenzten Sitzungen oder im Fehlerfall auch vom Server.

Da im Web eine stehende Verbindung über das zustandslose Protokoll HTTP nicht möglich ist (Abbildung 9.4), ist eine Sitzung hier eine Verbindung aus mehreren logisch zusammengehörenden Aufrufen (den HTTP-Requests) und muss in einer auf HTTP basierenden Anwendung implementiert werden, also *über* der Anwendungsschicht des Protokollstacks. (Siehe dazu auch § 1.1, S. 9ff) Das können daher nur der Browser oder der Webserver sein. Die für die Sitzung

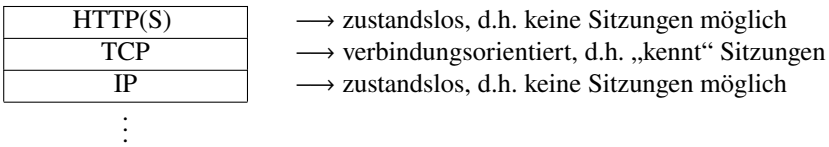


Abbildung 9.4: Die TCP/IP-Familie und Sitzungen, vgl. §1.6, S. 13ff.

und deren jeweiligen Zustand notwendigen Daten müssen dabei Client und Server zu jedem Zeitpunkt bekannt sein. Ganz grundsätzlich können dabei die Sitzungsdaten vom Client allein, vom Server allein, oder koordiniert von beiden verwaltet werden. Wird die Verwaltung der Sitzungsdaten vollständig dem Client übertragen, der sie mit jedem Request über einen URI mit den Nutzdaten übermittelt und abhängig von der Response verändert, so heißt die Implementierung „REST-konform“.

Meist werden die Sitzungsdaten jedoch vom Webserver verwaltet und über eine eindeutige *Session-ID* hergestellt, die entweder auf dem Client und dem Server koordiniert (meist mit Hilfe von Cookies) oder auf dem Server allein gespeichert sind. Beide Varianten der serverseiten Sitzungsverwaltung bietet PHP standardmäßig an.

9.2.2 Sitzungen in PHP

Eine Sitzung in PHP wird mit dem Aufruf `session_start()` gestartet. In diesem Moment sucht PHP anhand einer übermittelten Session-ID, ob die Sitzung existiert, oder erzeugt eine neue.

Jede Sitzung hat in PHP den einen, immer gleichen Namen `PHPSESSID`, eine eindeutige Session-ID und optionale Sitzungsdaten in dem superglobalen assoziativen Array `$_SESSION`, in dem weitere im Verlauf der Sitzung erzeugten Daten als Schlüssel-Wert-Paare gespeichert werden können. Die Funktion `session_start()` muss aufgerufen werden, bevor irgendein Text (noch nicht einmal ein Leerzeichen) an den Browser versendet wird,[2] auf jeden Fall aber vor der ersten Anweisung mit einem Abruf der Sitzungsdaten. Nach Beginn einer Sitzung können Name und ID mit den Funktionen

`session_name()` und `session_id()`

ausgelesen werden. Eine Sitzung wird automatisch mit dem Schließen des Browserfensters beendet. Die eigentlichen Nutzdaten der Sitzung können nun mit Hilfe des superglobalen Arrays `$_SESSION` erstellt und gespeichert werden, indem ein Schlüssel-Wert-Paar `key="value"` als `$_SESSION["key"]="value"` gespeichert wird,

`key="value"` $\Rightarrow$ `$_SESSION["key"]="value"`

Ein einfaches Beispiel, das als Zustand einer Sitzung die Anzahl der bisherigen Aufrufe anzeigt, ist durch das folgende Beispielprogramm gegeben:

Listing 9.2: Eine Sitzung starten und Werte speichern

```php
<?php
  session_start();

  if (!isset($_SESSION["aufrufe"])) {
    $_SESSION["aufrufe"] = 1;  // erster Aufruf: Initialisierung
  } else {
    $_SESSION["aufrufe"]++; // weitere Aufrude: hochzählen ...
  }
  echo "Sitzungszaehler: " . $_SESSION["aufrufe"];
  echo "<br/>Sitzung: Name = ".session_name().", ID = ".session_id();
?>
```

Die Zustandsvariable ist hier `aufrufe` und wird als Eintrag im Array `$_SESSION` gespeichert.

9.2.3 Sitzungen ohne Cookies

Für den Fall, dass der Client keine Cookies zulässt, müssen die Sitzungsdaten über den URI übermittelt werden, also über GET, oder aber über POST. Das sind die einzigen (!) Möglichkeiten, Informationen an den Server zu übermitteln. Ein Beispiel ist das folgende Programm, in dem die Session-ID dem Client per als Hyperlink mitgegeben wird und die der Anwender beim Anklicken übermittelt.

Listing 9.3: Eine Sitzung ohne Cookies

```php
<?php
  ini_set("session.use_cookies", false);
  ini_set("session.use_only_cookies", false);
  session_start();
```

[2] Zwar wird die gesamte Sitzungsverwaltung durch die PHP-Engine abgewickelt und kommt im Wesentlichen ohne Modifikationen der HTTP-Header aus, jedoch wird je nach Variante zu Sitzungsbeginn ein Cookie gesendet. Je nach Servereinstellung kann das allerdings auch eine separat geschickte Response sein, womit die Position der Funktion im Skript dann wieder keine Rolle spielt.

```
  if (!isset($_SESSION["aufrufe"])) {
    $_SESSION["aufrufe"] = 1;
  } else {
    $_SESSION["aufrufe"]++;
  }
  echo "Anzahl Aufrufe dieser Sitzung: " . $_SESSION["aufrufe"] . "<br/>";
  echo "<a href='$_SERVER[PHP_SELF]?".session_name()."=".session_id().
       "'>Sitzung aufrechterhalten</a>";
?>
```

Mit den beiden Aufrufen der Funktion `ini_set` wird der PHP-Interpreter so konfiguriert, dass er keine Cookies zur Sitzungsdatenverwaltung verwendet. Zur Aufrechterhaltung der Sitzung muss allerdings der Link angeklickt werden. (Zumindest beim zweiten Aufruf, danach könnte man die Reload-Funktion des Browsers nutzen, da der URI von da an die Session-ID enthält.)

9.2.4 Sicherheitsapekte: Erneuerung der Session-ID

Da die Übermittlung einer Session-ID generell eine bekannte Sicherheitslücke darstellt, die *Session Fixation*, sollte die Session-ID mit der Funktion

```
session_regenerate_id(true)
```

bei jedem Seitenaufruf stets neu generiert werden, so dass die Kenntnis der vorherigen Session-IDs für einen Angreifer wertlos werden.

Listing 9.4: Eine Sitzung ohne Cookies mit erneuerter Session-ID

```
<?php
  ini_set("session.use_cookies", false);
  ini_set("session.use_only_cookies", false);
  session_start();
  session_regenerate_id(true);

 if (!isset($_SESSION["aufrufe"])) {
    $_SESSION["aufrufe"] = 1;
  } else {
    $_SESSION["aufrufe"]++;
  }
  echo "Anzahl Aufrufe dieser Sitzung: " . $_SESSION["aufrufe"] . "<br/>";
  echo "<a href='$_SERVER[PHP_SELF]?".session_name()."=".session_id().
       "'>Sitzung aufrechterhalten</a>";
?>
```

Allerdings büßt die Webanwendung durch diese Maßnahme etwas an Bedienbarkeit ein, da der Benutzer zur Aufrechterhaltung der Sitzung dann nicht den Zurück-Button seines Browsers verwenden darf, sondern nur noch über Hyperlinks oder Submit-Buttons.

9.3 Gegenüberstellung Cookies und Sessions

Zwar gibt es gewisse Gemeinsamkeiten können Sessions auf Cookies basieren, dennoch sind ihr Funktionsprinzip und ihre Funktionalität ganz verschieden. Gemeinsam ist Cookies und

Sessions in PHP, dass die Nutzdaten, also die Daten, die nicht zum notwendigen technischen Overhead gehören, in einer superglobalen Variablen gespeichert werden. Bei Cookies ist dies das Array `$_COOKIES`, bei Sessions das Array `$_SESSION`. Gemeinsam ist beiden ebenso, dass die Verwaltung der Daten wie Lesen, Ändern oder Löschen durch den Server geschieht.

Der wesentliche Unterschied zwischen Cookies und Sessions allerdings ist der Speicherort der Nutzdaten. Die Daten von Cookies werden bei dem Browser persistent (d.h. hier über die Laufzeit des PHP-Skripts hinaus) gespeichert, die Nutzdaten von Sessions dagegen auf dem Server. Dieser Sachverhalt wird in Abbildung 9.5 skizziert. Daraus ergeben sich zwangsläufig

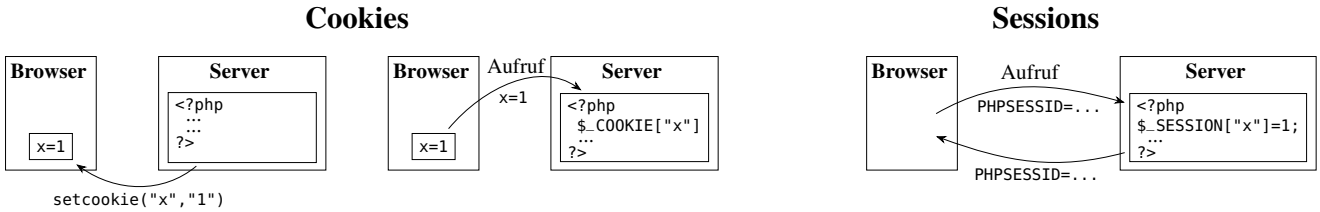


Abbildung 9.5: Die Nutzdaten werden bei Cookies im Browser gespeichert, bei Sessions auf dem Server.

alle weiteren Unterschiede zwischen Cookies und Sessions in PHP. So wird bei Cookies eine Variable, beispielsweise `x`, durch `setcookies("x", 1, ...)` mit einem Wert (hier 1) initialisiert, bei Sessions dagegen durch Einführung einer neuen Sessionvariablen mit `$_SESSION["x"]=1`. Bei Cookies kann PHP über das Array `$_COOKIE` die aktuell übertragenen Cookies nur auslesen, die originalen Cookiedaten auf dem Browser aber nur durch ein neues Setzen mit `setcookie` ändern. Das ist bei Sessionvariablen ganz anders, sie können – wie andere Variablen in PHP auch – einfach geändert werden, z.B. mit `$_SESSION["x"]++`. Ein weiterer Unterschied ist die Verfallszeit der Nutzdaten: Während sie bei Cookies frei einstellbar ist und mehrere Jahre in die Zukunft reichen kann, ist sie bei Sessions auf die Dauer der aktuellen Browsersitzung beschränkt, endet also mit Schließung des Browserfensters. Zusammengefasst erhalten wir die folgende Gegenüberstellung der Eigenschaften von Cookies und Sessions in PHP.

Eigenschaft	**Cookies**	**Sessions**
PHP-Variable	`$_COOKIE`	`$_SESSION`
Datenverwaltung	Server	Server
Speicherort (Nutzdaten)	Browser	Server
Verfallszeit (Nutzdaten)	frei einstellbar	Ende Browsersitzung
Dateninitialisierung `x=1`	`setcookie("x", 1, ...);`	`$_SESSION["x"] = 1`
Datenänderung `x++`	`setcookie("x", $_COOKIE["x"]+1, ...);`	`$_SESSION["x"]++;`

9.4 Zusammenfassung

In diesem Kapitel haben wir zwei Möglichkeiten kennengelernt, mit PHP Daten dauerhaft zu speichern. Einerseits können Daten mit Hilfe von Cookies auf dem Rechner des Browsers gespeichert werden, andererseits serverseitig mit Sessions.

Mit Cookies werden Strings als Schlüssel-Wert-Paare auf dem Server erzeugt und mit dem Aufruf des Clients an den Browser geschickt. Beim nächsten Aufruf der Seite wird dieses Cookie wieder an den Server zurückgeschickt.

PHP-Sessions ermöglichen stehende Verbindungen einer Webanwendung ermöglichen. Die Verwaltung der Sessions übernimmt der Webserver über eine eindeutige Session-ID, die wiederum mittels eines Cookies oder eines URL's dem passenden Browser zugeordnet werden kann.

In einer Sitzung können in PHP Variablen als Einträge des superglobalen Arrays `$_SESSION` in der Form `$_SESSION["variable"]` gespeichert und imVerlauf der Sitzung abgerufen und verändert werden.

Da wir eine Sitzung neben PHP-Sessions grundsätzlich auch mit Cookies ermöglichen können, haben wir Cookies und Sessions in PHP gegenübergestellt. Die Programmierung mit Sessions erweist sich dabei etwas einfacher und intuitiver, da man mit Sessionvariablen wie mit lokalne Variablen arbeiten kann und nicht umständlich die Cookie-Methode `setcookie` verwenden muss.

10 Objekte

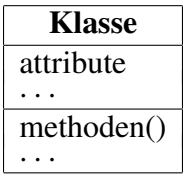
Kapitelübersicht

Im Rahmen der objektorientierten Programmierung sind Objekte Datenstrukturen, die zusammengehörige Daten zu einer Einheit zusammenfassen und die erlaubten Zugriffe darauf regeln. Das Konzept eines Objekts in diesem Sinne ist also eng verwandt mit demjenigen des Datensatzes einer Datenbank, allerdings ohne die Zugriffskontrolle auf die Daten.

Wir werden in diesem Kapitel behandeln, wie Objekte in PHP implementiert und angewendet werden können. Wie in den meisten Programmiersprachen werden die Struktur der Daten eines Objekts und die Zugriffskontrolle sowie Algorithmen darauf durch eine Klasse bestimmt. Auch werden wir betrachten, wie die Konzepte der Vererbung von Klassen und der *Traits* zur Vererbung von Funktionen in PHP realisiert werden.

10.1 Das Konzept eines Objekts in der Programmierung

Ein *Objekt* im Sinne der Objektorientierung ist eine programmiertechnische Speichereinheit von zusammengehörigen und strukturierten Daten sowie von auf sie operierenden Funktionen. Die Daten heißen dabei *Attribute*, die Funktionen *Objektmethoden* oder nur kur *Methoden*. In der objektorientierten Programmierung ist die grundlegende Programmiereinheit dafür die Klasse, die den Datentyp oder Bauplan der aus ihr erzeugten Objekte darstellt und die Attribute und Objektmethoden definiert. Für die Darstellung oder den Entwurf einer Klasse ist ein *Klassendiagramm* sehr nützlich, das in drei abgeteilten Bereichen eines Rechtecks den Klassennamen, die Attribute und die Methoden auflistet:



In Objekten können somit komplexe Einheiten von Daten und Funktionen programmiert und für Andere zur Wiederverwendung verfügbar gemacht werden. So sind viele PHP-Frameworks objektorientiert programmiert und können nur als Objekte verwendet werden, so zum Beispiel die Template-Engine Smarty (`http://smarty.net`) oder die PDF-Bibliothek FPDF (`http://fpdf.org`).

Die Syntax von PHP zur Programmierung von Klassen und Objekten ist ähnlich wie in Java oder C++, allerdings gibt es ein paar wesentliche Unterschiede, beispielsweise:

- Klassen dürfen in PHP nicht **`public`** sein.
- Attribute müssen einen Modifizierer haben, also **`private`**, **`protected`** oder **`public`**; empfohlen ist **`protected`**, wenn das Attribut vererbbar sein soll, ansonsten **`private`**. (Ein **`public`** Attribut widerspricht zwar der Idee der Datenkapselung, also einer der Grundideen der Objektorientierung, wird aber oft in Objekten der API oder der Frameworks verwendet.)
- Methoden und Konstruktoren sind Funktionen.
- Es kann maximal einen Konstruktor geben, und er muss dann `__construct(...)` heißen (egal wie die Klasse heißt).
- Die Referenzierung auf Attribute und Objektmethoden muss mit dem Pfeiloperator `->` geschehen (Der Punkt ist in PHP ja schon für die Stringkonketanation vorgesehen!)
- Demgegenüber müssen statische Attribute (Konstanten) oder Methoden (Klassenfunktionen) mit dem Doppel-Doppelpunktoperator `::` referenziert werden.
- Attribute müssen in Objektmethoden mit **`$this`**`->attribut` referenziert werden;
- **`public`** Methoden werden von außen mit `$objekt->methode(...)` aufgerufen, ebenso **`public`** Attribute.

Das folgende Programm möge die Syntax für Klassen und Objekte erläutern:

Listing 10.1: Die Klasse Gasthaus

```
<!DOCTYPE HTML><html lang="de-DE"><head><meta charset="UTF-8"></head><body>
<!-- Gasthaus.php -->
<?php
  class Gasthaus {
     protected $name;
     protected $betten;
     protected $frei;

     public function __construct($name, $betten = 2) {
        $this->name   = $name;
        $this->betten = $betten;
        $this->frei   = $betten;
     }
     public function __toString() {
        return "$this->name, $this->frei freie Betten";
     }
     public function buchen($betten) {
        if ($this->frei >= $betten) {
```

```
            $this->frei -= $betten;
            return "Buchung von $betten Betten in $this->name bestätigt!";
         } else {
            return "Nicht genügend Betten frei!";
         }
      }
   }

   $herberge = new Gasthaus("Jugendherberge A", 250);
   $pension  = new Gasthaus("Pension B");
   echo "$herberge,<br/> $pension<br/>";
   echo $herberge->buchen(2);
   echo "<br/>$herberge,<br/> $pension";
?>
</body></html>
```

Das Klassendiagramm dieser Klasse lässt sich also wie folgt darstellen:

Gasthaus
name betten frei
__toString() buchen(integer)

Die Ausgabe des Programms lautet entsprechend:

```
Jugendherberge A, 250 freie Betten,
Pension B, 2 freie Betten
Buchung bestätigt: 2 Betten in Jugendherberge A
Jugendherberge A, 248 freie Betten,
Pension B, 2 freie Betten
```

10.2 Vererbung

Ein wichtiges Programmierkonzept der Objektorientierung ist die Vererbung. Durch Vererbung wird eine Basisklasse (Superklasse) erweitert um zusätzliche Attribute und Funktionen.

Listing 10.2: Die Klasse Hotel

```
<!DOCTYPE HTML><html lang="de-DE"><head><meta charset="UTF-8"></head><body>
<!-- Hotel.php -->
<?php
   require_once("Gasthaus.php");
   class Hotel extends Gasthaus {
      protected $zimmer;

      public function __construct($name, $zimmer) {
         parent::__construct($name, 2*$zimmer);
         $this->zimmer = $zimmer;
         $this->frei   = $zimmer;
      }
```

```
    public function __toString() {
       return "$this->name, $this->frei freie Zimmer";
    }
    public function buchen($zimmer) {
       if ($this->frei >= $zimmer) {
          $this->frei -= $zimmer;
          return "Buchung von $zimmer Zimmern in $this->name bestätigt!";
       } else {
          return "Nicht genügend Zimmer frei!";
       }
    }
  }

  $hotel   = new Hotel("Hotel C", 450);
  $pension = new Gasthaus("Pension D");
  echo "<br/><br/>$hotel,<br/> $pension<br/>";
  echo $hotel->buchen(1) . "<br/>";
  echo "$hotel,<br/> $pension";
?>
</body></html>
```

Die Ausgabe dieses Programms lautet:

```
Hotel C, 450 freie Zimmer,
Pension D, 2 freie Betten
Buchung bestätigt: 1 Zimmern in Hotel C
Hotel C, 449 freie Zimmer,
Pension D, 2 freie Betten
```

10.3 Traits

Ähnlich wie eine Klasse ist ein *Trait* eine benannte Gruppe von Methoden, Konstanten und Klassenvariablen. Anders als von einer Klasse kann von einem Trait jedoch kein Objekt instanziiert werden, und es können keine Subklassen gebildet werden. Traits werden in PHP verwendet, um Namensräume für statische Methoden und Konstanten zu erstellen oder Mixins zu programmieren, d.h. Einheiten von Objektmethoden und Objektvariablen, die man mit `use` in eine Klasse „hineinmischen“ *(mix in)* kann, so dass deren Objekte diese Methoden auch verwenden können:

Listing 10.3: Ein Trait

```
<!-- Traits.php -->
<?php
  trait Mathematik {
     public function mehrwertsteuer($brutto) {
        return 0.19/1.19 * $brutto;
     }
  }

  class Rechnung {
     use Mathematik;
  }
```

```
  $rechnung = new Rechnung();
  echo $rechnung->mehrwertsteuer(29.90);
?>
```

Prinzipiell ist damit in PHP die Mehrfachvererbung von Verhalten möglich, allerdings bemerkt der PHP-Interpreter eine solche Mehrfachvererbung und verlangt die explizite Angabe der zu verwendenden Methode[1].

10.4 Zusammenfassung

In diesem Kapitel haben wir behandelt, wie Objekte in PHP implementiert und die Struktur ihrer Daten und die Zugriffskontrolle sowie Algorithmen darauf durch eine Klasse bestimmt werden. Auch die Konzepte der Vererbung von Klassen und der *Traits* zur Vererbung von Funktionen in PHP haben wir kennengelernt.

[1] vgl. Maurice (2014):§9.14.2.

11 Datenbankzugriffe übers Web: HTML & PHP & SQL

Kapitelübersicht

In diesem Kapitel werden die prinzipielle Architektur webbasierter Datenbankanwendungen und die grundlegenden Programmierschritte und Befehle behandelt. Erfahrungsgemäß machen dabei den Studierenden gar nicht so sehr die eigentlichen Programmieranweisungen Probleme, sondern eher das Verständnis für das Zusammenspiel der verschiedenen Programmiersprachen und Systeme. Zunächst ist wichtig zu verstehen, dass die klassische Client-Server-Architektur erweitert wird zu einer oder einer *Drei-Schichten-Architektur (3-tier architecture)*,[1] siehe Abbildung 11.1. Man spricht hier auch oft von einem *Software Stack*[2]

Webanwendungen sind dreischichtig

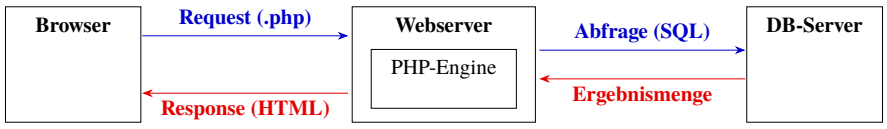


Abbildung 11.1: Die drei Schichten einer webbasierten Datenbankanwendung

Grundsätzlich laufen die Prozesse in jeder Schicht separat ab. Sie könnten daher auf getrennten Rechnern verteilt ablaufen, verbunden miteinander nur über das Internet. Üblicherweise werden webbasierte Datenbankanwendungen aber mit Umgebungen wie XAMPP entwickelt, wo alle drei Schichten auf demselben Rechner ablaufen.

[1] `https://de.wikipedia.org/wiki/Schichtenarchitektur#Drei-Schichten-Architektur`
[2] `https://encyclopedia2.thefreedictionary.com/software+stack`

Prinzip: Komplexität auf mehrere Schichten verteilen

Jede Schicht kann ausschließlich mit einer direkt benachbarten kommunizieren. So entstehen gewissermaßen zwei separate Client-Server-Architekturen: Ein Browser ist Client für den Webserver (wie wir es ja schon kennen), der Webserver mit seiner PHP-Engine wiederum ist Client für den Datenbankserver. Insbesondere kann ein Browser nicht direkt auf eine Datenbank zugreifen, sondern nur indirekt über einen Webserver. Auf den ersten Blick wirkt diese Struktur etwas umständlich: Browser und Webserver sollten separat laufen, klar, aber: Wieso nicht Webserver und Datenbankserver zu einem einzigen „Serversystem“ zusammenfassen? Nun, da jede Schicht ihre individuellen Aufgaben und Möglichkeiten hat, kann so die recht hohe Komplexität einer Webdatenbankanwendung auf die einzelnen Schichten verteilt werden.[3] Wir werden in Kapitel 13 genauer betrachten, wie das aussehen kann.

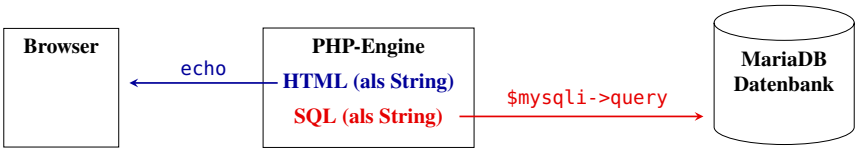


Abbildung 11.2: PHP als Schnittstelle, die über reinen Text kommuniziert

Schauen wir uns die einzelnen Kommunikationswege einer Datenbankabfrage etwas detaillierter an. Nehmen wir dazu als typisches Beispiel an, ein Kunde möchte die aktuell verfügbaren Artikel eines Webshops angezeigt bekommen, die in einer zentralen Datenbank gespeichert sind. Dazu ruft er im Browser eine geeignete PHP-Datei auf dem Webserver auf, die wiederum für ihre HTML-Antwort die eigentliche Datenbankabfrage startet und die Ergebnismenge am Ende in HTML formatiert an den Browser zurückschickt. Die PHP-Engine des Webservers ist daher die zentrale Schnittstelle einer webbasierten Datenbankanwendung. Sie kann natürlich nur PHP-Programme ausführen, die SQL-Anweisungen schickt sie als String an die Datenbank und die HTML-Formatierungen ebenfalls als String an den Browser. Die entsprechenden PHP-Befehle zum Verschicken der Strings an die jeweiligen Schichten sind `echo` und `$mysqli->query()`, wie in Abbildung 11.2 skizziert. Hierbei ist der Datenbankserver durch MariaDB[4] realisiert, das in der Webprogrammierung weit verbreitet ist.
Beachten Sie also: In einem PHP-Skript kann weder HTML- noch SQL-Code *verarbeitet* werden. Beides kann nur dynamisch als String erzeugt und entweder per `echo` an den Browser oder, nach geeigneter Vorbereitung, per `$mysqli->query()` an die Datenbank geschickt werden.

11.1 Zugriff von PHP auf MariaDB oder MySQL

Ein Datenbankserver arbeitet mit einer mehrbenutzerfähigen Datenbank, also nicht mit einer lokalen Datenbank wie Access, die nur einen Benutzer zulässt. Beispiele für Datenbankserver sind MariaDB, MySQL, Oracle oder MSSQL. Um mit PHP auf Daten eines allgemeinen Datenbankservers zuzugreifen, benötigt man im Wesentlichen die folgenden Schritte:

1. Verbindung zur Datenbank eines Datenbankservers erstellen;
2. Abfrage (*query*) zur Datenbank per SQL absetzen;
3. Datensätze von Auswahlabfragen (*result*) aus der Datenbank in eine geeignete Datenstruktur von PHP umformen.

[3] Diese Art der Komplexitätsreduktion liegt übrigens jeder Schichtenarchitektur zugrunde, so beispielsweise auch den TCP/IP-Protokollen.

[4] Die Open-Source Software MySQL gehört seit Januar 2010 zu Oracle. 2009 spaltete sich davon das freie System MariaDB ab. Die Schnittstellen von PHP zu beiden Systemen sind identisch.

Für eine Webanwendung in PHP, die Zugriffe auf eine MariaDB-Datenbank aus dem Web ermöglichen soll, müssen wir diese Schritte programmieren können. Dazu stehen in PHP Funktionen zur Verfügung, die wir im folgenden kennen lernen werden. Stellen wir zunächst einmal den Schritten für einen Datenzugriff die dazu verwendeten PHP-Funktionen gegenüber; in den nächsten Abschnitten werden wir diese näher betrachten:

Arbeitsschritt	**PHP-Anweisung**
1. Verbindung zur Datenbank erstellen	`$mysqli = new mysqli(...)`
2. SQL-Abfrage zur Datenbank senden	`$result = $mysqli->query(...)`
3a. Abfrageergebnis als Objekt in PHP einlesen	`$result->fetch_object()`
3b. Abfrageergebnis als assoziatives Array in PHP einlesen	`$result->fetch_assoc()`
3c. Abfrageergebnis als numerisches Array in PHP einlesen	`$result->fetch_row()`

Die Schritte 3a, 3b und 3c der Ergebniseinlesung nach PHP sind hierbei als Alternativen anzusehen, sie unterscheiden sich lediglich in der Datenstruktur, in der die Datensätze der Ergebnismenge der Datenbankabfrage in PHP zur Verfügung stehen.

Haben wir die Verbindung zu einer Datenbank eingerichtet, so können wir u.a. die folgenden Abfragefunktionen über SQL an die Datenbank schicken:

1. Eintragen von Daten mit `INSERT`. Die Syntax dazu lautet:

```
INSERT INTO tabelle (spalte1, spalte2, ...)
VALUES ('wert1', 'wert2', ...)
```

2. Ansehen gespeicherter Daten mit `SELECT`:

```
SELECT spalte_x, spalte_y FROM tabelle WHERE filterbedingung ...
```

   (Die `WHERE`-Klausel ist optional.) Will man alle Daten einer Tabelle erhalten, so kann man statt der einzelnen Spaltennamen auch einfach die Wildcard $*$ angeben.

3. Ändern von gespeicherten Daten mit `UPDATE`:

```
UPDATE tabelle SET
  spalte1 = 'neuerWert1',
  spalte2 = 'neuerWert2',
  ...
WHERE Primärschlüssel = 'x'
```

4. Löschen gespeicherter Daten mit `DELETE`:

```
DELETE FROM tabelle WHERE bedingungen
```

   (Achten Sie hier auf die `WHERE`-Klausel, sonst ist *alles* gelöscht . . .)

11.1.1 Verbindung zum Datenbanksystem: `new mysqli(...)`

Der Verbindungsaufbau zu einer Datenbank geschieht durch die Erzeugung eines mysqli-Objekts, `$mysqli = new mysqli(...)`, eines Standardobjekts in PHP:

```
$mysqli = new mysqli("localhost", "user", "passwort", "db");
```

Der Konstruktor des PHP-Objekts `mysqli` erwartet vier Eingabeparameter:

1. Den Namen des Datenbankservers: Dies ist in der Regel `"localhost"`. Damit wird der Rechner, auf dem die PHP-Engine läuft, selber angesprochen. Alternativ kann hier die IP-Adresse eines anderen Rechners stehen; allerdings muss das Serversystem dazu entsprechend konfiguriert sein.
2. Den Namen des Benutzers, hier `"user"`. Hier kann der Name eines jeden beim Datenbankserver registrierten Nutzers stehen.
3. Das Passwort des Benutzers, hier: `"passwort"`.
4. Den Namen der Datenbank, hier `"db"`.

Standard User & Passwort bei XAMPP

Optional kann als fünfter Parameter der Port des Datenbankservers angegeben werden, sein Wert ist üblicherweise 3306. Bei der Standardinstallation von XAMPP ist der User der Datenbank `root` und das Passwort leer `""`.

Es ist zu empfehlen, zur besseren Übersicht und zur Vereinfachung bei Änderungen der Zugriffsdaten die Aufrufparameter in Variablen zu speichern:

```
$server   = "localhost";
$user     = "user";
$password = "passwort";
$database = "datenbank";

$mysqli   = new mysqli($server, $user, $password, $database);
$mysqli->set_charset("utf8");
```

Die letzte Anweisung ist sehr ratsam und veranlasst das mysqli-Objekt, alle Strings von und zur Datenbank in der UTF8-Kodierung zu verarbeiten. Dies ist die Standardkodierung von MariaDB. Natürlich muss dazu Text aus einem HTML-Formular, der über PHP in der Datenbank gespeichert werden soll, dieselbe Kodierung haben, d.h. die Webseiten zur Eingabe sollten entsprechend das `<meta>`-Element

```
  <meta charset="utf-8"/>;
```

enthalten.

Um Fehler beim Zugriff ausgeben zu lassen, was gerade bei der Entwicklung einer Anwendung sehr hilfreich sein kann, sollte man nach Erzeugung des mysqli-Objekts und vor Einstellung der Kodierung das Attribut `$mysqli->connect_ errno` verwenden:

```
$server   = "localhost";
$user     = "user";
$password = "passwort";
$database = "datenbank";

$mysqli   = new mysqli($server, $user, $password, $database);
if ($mysqli->connect_errno) {
   echo "Fehler beim Zugriff auf MySQL: (" .
```

Listing 11.1: Die Datei `db_login.inc.php`

```php
<?php
/** Erstellt eine Datenbankverbindung mit zentralen
 *  Konfigurationsdaten des DB-Servers.
 *  @param $database die Datenbank
 *  @return mysqli-Objekt der Datenbankverbindung
 */
function login($database) {
   $server   = "localhost";
   $user     = "user";
   $password = "passwort";

   $mysqli   = new mysqli($server,$user,$password,$database);
   if ($mysqli->connect_errno) {
      echo "Keine DB-Verbindung: ({$mysqli->connect_errno}) "
         . $mysqli->connect_error;
      exit();   // beendet das Programm
   }

   $mysqli->set_charset("utf8");
   return $mysqli;
}
?>
```

```
      $mysqli->connect_errno . ") " . $mysqli->connect_error;
}
$mysqli->set_charset("utf8");
```

Es gibt die Fehlernummer der aktuellen Datenbankverbindung an und ist null, wenn der Zugriff erfolgreich war; in der `if`-Anweisung wird dies von PHP als `false` betrachtet und die `echo`-Anweisung daher nicht ausgeführt. Ist jedoch ein Fehler aufgetreten, so ergibt in PHP die `if`-Anweisung `true` und das Attribut `$mysqli->connect_error` enthält eine Beschreibung des eingetretenen Fehlers. (Eine solche Fehlerausgabe sollte man bei produktiv eingesetzten und veröffentlichten Systemen allerdings deaktivieren, denn ein böswilliger Hacker kann damit wertvolle Informationen erlangen.) Weitere Hinweise und Informationen finden Sie auf `php.net` unter dem Stichwort mysqli.[5]

DB Login-Daten in zentrale Datei auslagern

Eine weitere Empfehlung ist, die Konfigurationsdaten des Zugriffs auf den Datenbankserver in eine eigene Datei auszulagern, z.B. `db_login.inc.php`, und zu Beginn eines PHP-Skripts per `require_once` einzulesen. So können mehrere PHP-Skripte die Einstellungen zentral einlesen und bei etwaigen Änderungen der Zugriffsparameter die aktuellen Werte erhalten, ohne dass sie selbst geändert werden müssen. Da meist verschiedene PHP-Skripte eines Webauftritts zwar auf denselben Datenbankserver zugreifen sollen, aber auf verschiedene Datenbanken, sollte in der Zugriffsdatei zentral eine Funktion definiert werden, die den Namen der Datenbank als Parameter erfordert und das mysqli-Objekt der Datenbankverbindung zurück gibt, wie in Listing 11.1 dargestellt. Diese Funktion kann dann von einem beliebigen PHP-Skript importiert werden, um eine Datenbankverbindung zu erstellen:

```php
   require_once("db_login.inc.php");
```

[5] `http://php.net/manual/de/mysqli.quickstart.connections.php`

```
$mysqli = login("AdressDB");
```

Danach ist bei erfolgreichem Zugriff auf den Datenbankserver und die Datenbank ein `mysqli`-Objekt über die Variable `$mysqli` verfügbar. Bei einem Verbindungsfehler allerdings wird hier wegen der Funktion `exit` das Skript sofort abgebrochen.

11.1.2 Abfragen mit `$mysqli->query`

Mit der Objektmethode `$mysqli->query` wird ein SQL-Kommando an die Datenbank geschickt und dort ausgeführt. Die Methode erwartet als Eingabeparameter die auszuführende SQL-Anweisung in Form eines Strings. Ein Beispiel für das Abschicken einer `SELECT`-Anweisung:

```
$result = $mysqli->query("SELECT * FROM adressen");
```

Der SQL-Befehl selektiert alle Datensätze der Tabelle `adressen`. Natürlich ist der SQL-Befehl für PHP nur ein String, also ohne weitere Bedeutung. Ähnlich wie PHP ein HTML-Dokument auch nur als einen String aufbaut und per `echo` an den Browser schickt, muss ein SQL-Befehl als String an die Datenbank geschickt werden. Genau das macht `$mysql->query`: es schickt die Abfrage an die Datenbank, die sie dort als SQL interpretiert und ausführt. Die Ergebnismenge (*result set*) wird üblicherweise in einer Variablen `$result` gespeichert.

Eine weitere Variante, die oft in der Literatur oder in Programmbeispielen zu sehen ist, ist die Methode `mysqli_query()` nach dem „prozeduralen Stil“. Sie hat als obligatorischen Parameter eine gültige Datenbankverbindung und erlaubt damit den Zugriff auf mehrere Datenbanken innerhalb eines PHP-Skripts, ohne sie immer wieder neu aufbauen zu müssen. Weitere Informationen dazu siehe unter `http://php.net/manual/en/mysqli.query.php`.

11.1.3 Übernahme der Daten nach PHP

Die Ergebnismenge `$result` einer erfolgreichen Abfrage hat mehrere Metoden, um auf deren Datensätze zuzugreifen. Die wichtigsten sind:

```
$result->fetch_object()
```

und

```
$result->fetch_assoc()
```

Die erste Methode liefert die Ergebnismenge der Datenbankabfrage als Objekt zurück, die zweite als assoziatives Array. Hierbei wird genauer der jeweils *nächste Datensatz* der Ergebnismenge `$result` gelesen und in einem Objekt mit den Spalten als Attributnamen bzw. einem assoziativen Array mit den Bezeichnungen der *Tabellen*spalten als Schlüssel gespeichert.

Die Wirkungen dieser beiden Methoen ist fast gleich und am besten anhand eines einfachen Beispiels zu erklären: Hat man nach einer Datenbankabfrage eine Ergebnismenge `$result` mit der Spalte `spalte` und dem Eintrag `$eintrag`, so erzeugt die Anweisung `$result->fetch_object()` je Datensatz ein Objekt

```
$wert = $eintrag->spalte
```

Entsprechend erzeugt die Anweisung `$result->fetch_assoc()` je Datensatz ein assoziatives Array

```
$wert = $eintrag["spalte"]
```

Eine weitere Möglichkeit der Konvertierung der Ergebnismenge einer Auswahlabfrage der Datenbank ist mit der Anweisung

```
$result->fetch_row()
```

gegeben, die ein numerisches Array liefert, in dem jedoch die Spaltennamen der Datenbanktabelle nicht mehr verfügbar sind. Der Zugriff auf die Datenbankinhalte geschieht hier über Indizes 0, 1, ... Die assoziative Variante bzw. die `fetch_object`-Methode sind flexibler, da sie auch nach späteren eventuellem Einfügen neuer Felder in die Datenbanktabelle korrekte Resultate liefern.

Um nach einem `SELECT`-Befehl die Ergebnismenge zu durchlaufen, kann man eine der folgenden Schleifenkonstrukte verwenden:

```php
while ($datensatz = $result->fetch_object()) {
   $datensatz->spalte ... // $datensatz ist ein Objekt
}
```

oder

```php
while ($datensatz = $result->fetch_assoc()) {
   $datensatz["spalte"] ... // $datensatz ist ein assoz. Array
}
```

oder

```php
while ($datensatz = $result->fetch_row()) {
   $datensatz[0] ... // $datensatz ist ein numerisches Array
}
```

Bei allen Varianten wird jeweils der nächste Datensatz in der Ergebnismenge `$result` als Objekt bzw. Array eingelesen, solange es einen solchen gibt. Nach dem letzten Datensatz liefert die Zuweisung ein leeres Ergebnis, d.h. sie ist `false`, und die Schleife wird beendet. Ist die Ergebnismenge leer oder enthält sie eine Fehlermeldung, so wird die Schleife gar nicht erst begonnen.

11.2 Anzeigen von Daten im Browser

Das folgende Programmbeispiel lehnt sich an das Adressbeispiel 4.2 aus dem Studienbrief *Programmierung 3: Programmierung mit PHP* an, indem es die verfügbaren Adressen aus einer Datenbank liest und anzeigt. Voraussetzung ist, dass bereits eine Datenbank mit einigen Einträgen existiert, siehe Abb. 11.3.

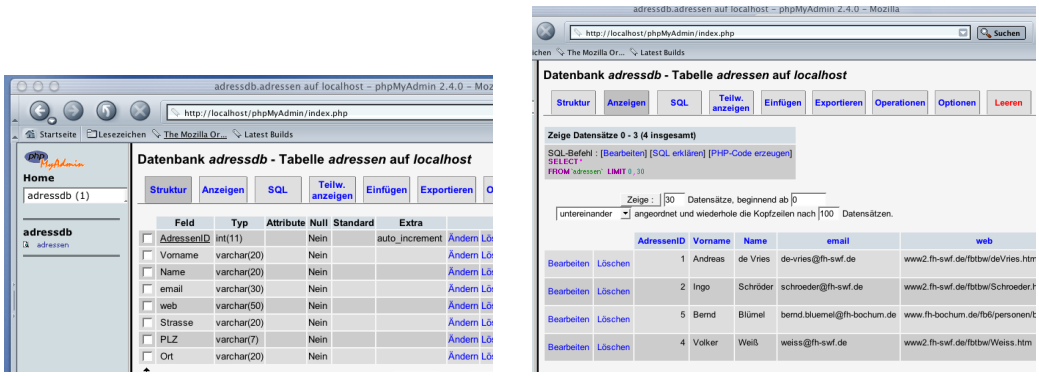


Abbildung 11.3: Die Struktur der Adressdatenbank (links) und einige Einträge (rechts), mit phpMyAdmin betrachtet.

Um den Inhalt der Datenbank als Tabelle im Browser darzustellen, muss also mit einem `SELECT`-Befehl als SQL-Statement auf die Datenbank zugegriffen werden, die Ergebnismenge mit `$mysqli->fetch_object` in ein Objekt in PHP transformiert und in HTML als Tabelle erzeugt werden.

Wir werden zwei Versionen eines solchen PHP-Skripts betrachten. Die erste zeigt den Datenbankinhalt einfach zeilenweise im Browser an und erläutert konkret die notwendigen Programmzeilen eines allgemeinen PHP-Skripts zur Datenanzeige. Das zweite Programm wird dann eine formatierte Anzeige in einer Tabelle vornehmen und die Feldnamen als Spaltenbezeichnung verwenden. Ferner wird die `foreach`-Schleife eingesetzt, um die Datensatzfelder automatisch zu durchlaufen.

Listing 11.2: Einträge einer Datenbank anzeigen

```
<!-- Adressen aus DB. Dateiname: adresseDB.php -->
<?php
  require_once("db_login.inc.php");
  $mysqli = login("adressDB");

  $result = $mysqli->query("SELECT * FROM adressen");
  while ($adresse = $result->fetch_object()) {
     echo "$adresse->Name, $adresse->Vorname, $adresse->email<br/>";
  }
?>
```

11.2.1 Die wesentlichen Schritte des PHP-Skripts

Schauen wir uns die wesentlichen Programmschritte an, um den aktuellen Inhalt einer Datenbank im Browser darzustellen. Zunächst wird in Zeile 3 die Verbindung zur Datenbank `AdressDB` auf dem Datenbankserver `localhost` erstellt. Dazu muss eine Datei `db_login.inc.php` mit den Login-Daten für den Datenbankzugriff vorhanden sein, wie in Abschnitt 11.1.1 beschrieben. Als nächstes wird in Zeile 6 mit der Anweisung

```
  $result = mysql_query( "SELECT * FROM adressen" );
```

eine SQL-Anweisung, hier ein `SELECT`, an den Datenbankserver geschickt und das das Ergebnis in der Variable `$result` gespeichert. Somit enthält `$result` die vollständige Ergebnismenge der SQL-Anweisung. Die Ergebnismenge kann je nach Abfrage und Datenbank tausende von Einträgen enthalten, oder vielleicht leer sein, oder aber einen Fehlercode liefern. Die für die Anzeige der Datensätze aus der Datenbank nächste wesentliche Anweisung ist die folgende Schleife:

```
  while ( $adresse = mysql_fetch_assoc($result) ) {
    echo "$adresse->Name ... <br/>";
  }
```

Die `while`-Schleife wiederholt sich hier, solange noch ein nächster Datensatz in der Ergebnismenge `$result` existiert. Falls nicht, so gibt die Anweisung in der `while`-Klammer nämlich `false` zurück, und der Schleifenrumpf wird nicht weiter ausgeführt.

Während eines Schleifendurchlaufs steht also in der Variablen `$adresse` ein bestimmter Datensatz aus der Datenbank als Objekt, mit den Feldnamen der Datenbanktabelle als Attribute und den Datenbankeinträgen des Datensatzes als Werte.

11.2.2 Formatierte Ausgabe

Zum Anzeigen der Daten reichen die obigen Anweisungen völlig aus. In dem folgenden Beispiel werden die Datensätze zeilenweise in einer HTML-Tabelle dargestellt.

Beispiel 11.1. Formatierte Ausgabe von Adressen aus Datenbank

```
<!-- Adressen aus DB. Dateiname: adresseDB1.php -->
<?php
  require_once("db_login.inc.php");
  $mysqli = login("adressDB");
?>

<h2>Adressen als 2-dimensionales Array</h2>
<table border="1">
<?php
   $result = $mysqli->query("SELECT * FROM adressen");

   // Tabellenkopf mit den Feldnamen als Spaltenbezeichnungen:
   echo "   <tr>\n";
   while ( $field = $result->fetch_field() ) {
      echo "    <th>$field->name</th>\n";
   }
   echo "   </tr>\n";

   // Datensätze der Datenbank, spezielle Ausgabe für email und web:
   while ( $adresse = $result->fetch_object() ) {
     echo "   <tr>\n";
     foreach ( $adresse as $key => $value ) {
       if ( $key === "email" ) {
          echo "    <td><a href=\"mailto:$value\">$value</a></td>\n";
       } else if ( $key === "web" ) {
          echo "    <td><a href=\"http://$value\">$value</a></td>\n";
       } else {
          echo "    <td>$value</td> \n";
       }
     }
     echo "   </tr>\n";
   }
?>
</table>
```

Die Datensätze werden hier in einer Tabelle ausgegeben. Die Ergebnismenge der SELECT-Anweisung enthält alle Spalten der Datenbanktabelle. Mit der Methode `fetch_field()` wird der Name der jeweils nächsten Spalte zurückgegeben. So wird in der ersten while-Schleife die Kopfzeile der HTML-Tabelle mit den Spaltennamen aus der Datenbank automatisch erstellt. Die `foreach`-Schleife läuft das Array sukzessive durch und ermöglicht den Zugriff auf die Inhalte über die Variable `$value`. In dem Beispiel wird der Wert von `$value` in ein `<td>`-Tag von HTML gepackt, so dass jeder Datensatz in `<tr>`-Elementen steht und so als eine Tabellenzeile im Browser angezeigt wird. □

Ermittlung der Feldnamen aus der Datenbank

Möchte man zusätzlich, so wie in diesem Beispiel, die Feldnamen der Datenbank als Spaltenbezeichner anzeigen, so muss man irgendwie an die Bezeichnungen der Felder aus der Datenbank gelangen. Die Information ist aber bereits in der Ergebnismenge einer erfolgreich abgeschlossenen `SELECT`-Anweisung enthalten, in unserem Falle also in der Variablen `$result`. Der eleganteste Weg, an diese Information zu kommen, ist mit Hilfe der Funktion `$result->fetch_field()`. Sie liefert ein Objekt zurück, dessen Attribute die Metadaten des jeweils nächsten Feldes der Datenbanktabelle(n) ist, wie z.B. Feldname oder Feldtyp. So kann man sich also mit der Anweisung

```
while ($spaltenname = $result->fetch_field()->name) {
   echo "$spaltenname<br/>";
}
```

die Feldnamen anzeigen lassen.

11.3 Informationen über gerade abgesetzte SQL-Befehle

In diesem Abschnitt werden zwei sehr nützliche öffentliche Attribute des mysqli-Objekts bzw. der Ergebnismenge vorgestellt, die Informationen über den jeweils letzten abgesetzte SQL-Befehl speichern.

- `$mysqli->affected_rows` speichert die Anzahl der durch die jeweils letzte SQL-Anfrage an den Server betroffenen Datensätze. Beispielsweise findet man wie folgt die Anzahl gelöschter Datensätze heraus:

  ```
  $mysqli->query("DELETE FROM mytable WHERE id < 10");
  $anzahl = $mysqli->affected_rows;
  echo "$anzahl Datensätze gelöscht";
  ```

  Die prozedurale Variante davon, `mysqli_affected_rows($link)`, ist eine Funktion und benötigt als Parameter die Verbindungskennung `$link`; Näheres dazu siehe unter `http://php.net/manual/de/mysqli.affected-rows.php`.

- `$result->num_rows` speichert die Anzahl der Datensätze einer Ergebnismenge `$result`. Dieses Attribut ist nur gültig nach einem `SELECT`-Befehl.

  ```
  $result = $mysqli->query("SELECT * FROM table1");
  $num_rows = $result->num_rows;
  echo "$num_rows Zeilen\n";
  ```

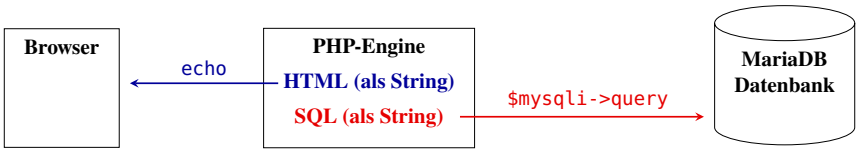
  Direkt nach einer `SELECT`-Anfrage mit `$mysqli->query()` ergibt `$result->num_ rows` dasselbe wie `$mysqli->affected_rows`.

11.4 Zusammenfassung

- Ein netzwerkbasiertes System, das Datenbankzugriffe über das Internet ermöglicht, kann mit HTML, PHP und SQL programmiert werden.

- Insgesamt stellt PHP eine Schnittstelle zwischen Browser und einer Datenbank dar.

- In einem PHP-Skript kann weder HTML- noch SQL-Code verarbeitet werden. Beides muss als String erzeugt werden und entweder per `echo` an den Browser oder per `$mysqli->query` an die Datenbank geschickt werden.



- Die Funktionen zur Durchführung der drei wesentlichen Schritte zum Datenzugriff auf eine Datenbank via PHP sind:

  1. `$mysqli = new mysqli(...)` — erstellt die Verbindung zur Datenbank auf einem Datenbankserver
  2. `$mysqli->query(...)` — sendet SQL-Befehle zur Datenbank (`SELECT`, `INSERT`, `UPDATE`, `DELETE`)
  3. `$result->fetch_object()`, `$result->fetch_assoc()`, `$result-> fetch_row()` — liest die Ergebnismenge einer SQL-Abfrage als Objekt bzw. als assoziatives oder numerisches Array in PHP ein

- Der Zugriff auf eine Datenbank sollte der Übersichtlichkeit und Wartbarkeit halber mit Variablen parametrisiert werden:

```php
$server   = "localhost";
$user     = "user";
$password = "passwort";
$database = "datenbank";

$mysqli   = new mysqli($server, $user, $password, $database);
$mysqli->set_charset('utf8');
```

  Zudem sollten diese Anweisungen in eine eigene Datei ausgelagert werden, so dass mehrere PHP-Skripte die Einstellungen einlesen können.

- Zum Anzeigen von Informationen aus einer Datenbanktabelle benötigt man mindestens die zwei Schritte

```php
$result = $mysqli->query("SELECT * FROM Tabelle");
```

  zur Selektion der Daten und die Schleife

```php
while ( $eintrag = $result->fetch_assoc() ) {
   foreach ($eintrag as $value) {
      echo "$value<br/>";
   }
}
```

12

Ablauflogik von Datenbankinteraktionen programmieren

Kapitelübersicht

PHP kann nicht auf Eingaben „warten“

Üblicherweise besteht eine Webanwendung aus einer Interaktion mit einer Datenbank, also aus einem komplexen Ablauf von Aus- und Eingabeaktionen, die jeweils von den vorherigen Aktionen abhängen. Ablauf und Logik aus Sicht des Anwenders müssen aufgrund der starren Request-Response-Architektur des Webs in dem PHP-Skript allerdings umgeschrieben werden. Die Schwierigkeit liegt abstrakt gesprochen darin, dass ein PHP-Skript immer nur auf einen einkommenden Request reagieren, aber keinen permanenten Prozess darstellen kann, der „anhält“ und auf eine Usereingabe „wartet“. Typischerweise läuft ein PHP-Skript nur Bruchteile von Sekunden. Es muss stattdessen bei jedem Aufruf den aktuellen Zustand des Prozessablaufs von Neuem feststellen. Ein typisches Beispiel ist der Fall, dass die Information benötigt wird, welchen Button der Anwender in einem vorherigen Prozessschritt gedrückt hat. Mit der Lösung dieser Problematik beschäftigen wir uns in diesem Kapitel.

Wir möchten dazu im Folgenden ein Skript schreiben, das nicht nur das Ansehen der Daten in der Adressdatenbank ermöglicht, sondern auch das Eintragen neuer Daten über den Browser. Grob soll dabei folgender Ablauf stattfinden:

1. Beim ersten Aufruf soll der aktuelle Inhalt der Datenbank gezeigt werden. Allerdings soll es nun einen Submit-Button geben, siehe Abb. 12.1 (links).

2. Anklicken dieses Buttons ruft dasselbe Skript auf, das nun aber ein Eingabeformular zeigt, in das die neuen Daten eingetragen werden können, siehe Abb. 12.1 (rechts).

3. Das Anklicken des Submit-Buttons soll erneut das Skript aufrufen, um nun die Formulardaten in die Datenbank zu speichern.

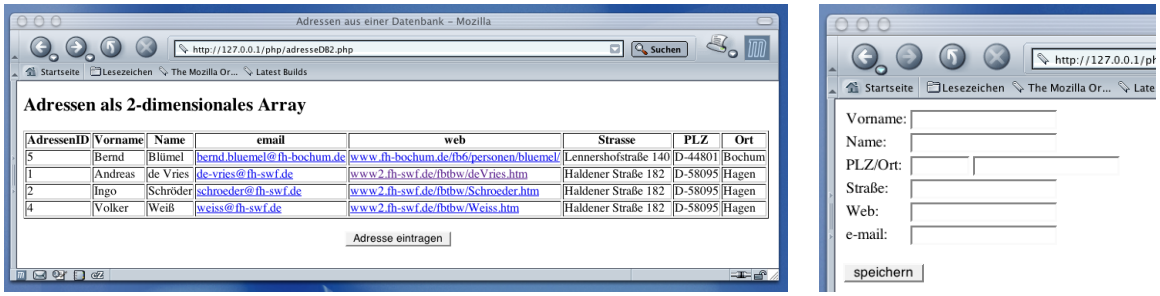


Abbildung 12.1: Die Seite beim Aufruf und beim Drücken des Eintragen-Buttons.

12.1 Eintragen-Skript mit unformatierten Ausgaben

Wir werden zunächst ein PHP-Skript betrachten, das zwar alle geforderten Funktionalitäten liefert, aber die Browserausgaben nicht formatiert. Schauen wir uns dazu zunächst den vollständigen Quelltext des Programms an, bevor wir den groben Ablauf und die einzelnen Schritte genauer untersuchen.

Listing 12.1: Adressen aus Datenbank editieren, unformatierte Ausgabe

```
<!-- Adressen in DB eintragen. Dateiname: adresseDB2.php -->
<form action="<?php echo $_SERVER['PHP_SELF'];?>" method="POST">
<?php
require_once("db_login.inc.php");
$mysqli = login("adressDB");

if (isset($_POST['eintragen'])) { // Eingabeformular erstellen
?>
     Vorname: <input type="text" name="vorname"/><br/>
     Name:    <input type="text" name="name"/><br/>
     PLZ:     <input type="text" name="plz" size="7"/>
     Ort:     <input type="text" name="ort"/> <br/>
     Strasse: <input type="text" name="strasse"/><br/>
     Web:     <input type="text" name="web"/><br/>
     e-mail:  <input type="text" name="email"/><br/>
     <input type="submit" name="speichern" value="Speichern"/>
<?php
} else {
   if (isset($_POST['speichern'])) { // Formulardaten speichern
      $name    = $_POST["name"];
      $vorname = $_POST["vorname"];
      $plz     = $_POST["plz"];
      $ort     = $_POST["ort"];
      $strasse = $_POST["strasse"];
      $web     = $_POST["web"];
      $email   = $_POST["email"];
```

```
      $sql  = "INSERT INTO adressen ";
      $sql .= "(Name, Vorname, PLZ, Ort, Strasse, web, email)";
      $sql .= " VALUES ";
      $sql .= "('$name', '$vorname', '$plz', '$ort', '$strasse',";
      $sql .= " '$web',  '$email')";
      $mysqli->query($sql);  // an Datenbank schicken
   } // -- Ende Formulardaten speichern -----------------

   //Adressen auflisten + Eintragen-Button anzeigen:
   $sql = "SELECT * FROM adressen ORDER BY Name";
   $result = $mysqli->query($sql);

   while ($adresse = $result->fetch_object()) {
      echo "$adresse->Name, $adresse->Vorname, $adresse->email<br/>";
   }
?>
     <input type="submit" name="eintragen" value="Adresse eintragen"/>
<?php
} // -- Ende else -------------------------------
?>
</form>
</body>
</html>
```

Zunächst fällt auf, dass das gesamte Skript durch ein Form-Tag umschlossen wird, das beim Submit die Seite selbst wieder aufruft (Zeilen 2 und 48).

12.1.1 Der Ablauf des Skripts

Ablauflogik durch if-Anweisungen

Der eigentliche Ablauf des Skripts aus Sicht des Benutzers wird durch `if`-Anweisungen gesteuert (Zeilen 7–18, 18–46 & 19–34). Sie haben hier die folgende Struktur:

```
if (isset($_POST['eintragen'])) {
   ... Eingabeformular und Speichern-Button erstellen ...
} else {
   if (isset($_POST['speichern'])) {
      ... Eingabedaten in DB speichern ...
   }
   ... DB-Inhalt auflisten und Eintragen-Button erstellen ...
}
```

Wie läuft das Skript also aus Benutzersicht ab? Wird es das erste Mal aufgerufen, so sind `$_POST["eintragen"]` und `$_POST["speichern"]` undefiniert. In dem Skript werden also nur die Anweisungen des else-Zweigs ohne die innere `if`-Anweisung ausgeführt, so dass der Benutzer den aktuellen Datenbankinhalt und den Eintragen-Button sieht.

Klickt er nun auf den Eintragen-Button, so wird das Skript erneut aufgerufen, doch diesmal ist der Inhalt von `$_POST["eintragen"]` definiert (wesentlich ist hierbei, dass der Eintragen-Button auch den Namen `eintragen` hat!). Somit werden die Anweisungen des äußeren `if`-Zweigs ausgeführt und das Eingabeformular für die Daten sowie der Speichern-Button angezeigt.

Schickt der Benutzer nun dieses Formular ab, so wird das Skript ein drittes Mal aufgerufen, diesmal ist jedoch `$_POST` mit den Eingabedaten gefüllt und `$_POST["speichern"]` ist nun ebenfalls definiert. Daher wird der innere `if`-Zweig ausgeführt, d. h. der neue Eintrag wird gespeichert, und der aktualisierte Stand der Datenbank angezeigt.

12.1.2 Das Eingabeformular

Das Eingabeformular wird aufgebaut, wenn zuvor der Submit-Button namens `eintragen` gedrückt worden ist. Jedes Input-Element hat hier als Namen den entsprechenden Spaltennamen aus der Datenbanktabelle. Das muss man zwar nicht so machen, ist aber sehr ratsam:

Bemerkung. Die Namen der Input-Elemente in Formularen sollten den Namen der Spaltennamen der Datenbanktabellen gleichen, denen sie entsprechen. Auf diese Weise hat man eine leicht erkennbare Verbindung zwischen Formularen und Datenbank und erleichtert die Programmierung, auch bei späteren Änderungen der Datenbankstruktur.

Wesentlich für den korrekten Ablauf des Skripts ist, dass der Submit-Button dieses Eingabeformulars `speichern` heißt, denn die steuernden `if`-Anweisungen benötigen zum korrekten Ablauf des Skripts diesen Namen.

12.1.3 Das Abspeichern der Daten

Zum Abspeichern der Daten aus dem Eingabeformular muss ein `INSERT`-Befehl in SQL auf die Datenbank abgesetzt werden. (Hier erkennt man einen der Vorteile, wenn man die Input-Elemente des Formulars gleich nennt wie die Felder der Tabelle!) Ein häufig gemachter Fehler ist das Fehlen der Apostrophs bei der Wertzuweisung.

Der SQL-Befehl wird zunächst in eine String-Variable `$sql` gespeichert. (Zu beachten sind dabei die Leerzeichen bei Zeilenumbrüchen!). Das hat einerseits den Vorteil, dass man den relativ langen Befehl übersichtlich aufbauen kann. Andererseits kann man im Fehlerfall leicht den SQL-Befehl anzeigen („debuggen“) lassen, um zu prüfen, ob es sich um einen SQL-Fehler handelt. Debugging ist das zeilenweise Verfolgen der Werte eines Programms während der Laufzeit. Es gibt dafür, ähnlich wie für andere Programmiersprachen, einige ausgereifte Tools, jedoch in so einfachen Umfeldern wie hier reicht auch ein einfaches Ausgeben der Werte, etwa vor dem `$mysqli->query`-Befehl mit:

```
echo "<br/> $sql <br/>";
```

Damit kann man sich während des Skriptablaufs anzeigen lassen, welches SQL-Statement genau abgesetzt worden ist. Oft kann man auf diese Weise Syntaxfehler in SQL schnell entdecken, insbesondere das Vergessen der Apostrophs (`'`). Möglich ist in diesem Zusammenhang auch das Debuggen mit etwaigen Fehlerausgaben der Funktion `mysqli_error` nach einer SQL-Anfrage:

```
$result = $mysqli->query($sql);
if ($mysqli->error) {
   echo "SQL error: " . $mysqli->error;
}
```

Mit der if-Bedingung wird die echo-Anweisung nur im Fehlerfall aufgerufen.

Wie auch immer, am Ende muss ein SQL-Befehl natürlich auch mit `$mysqli->query` an die Datenbank gesendet werden. Dieser Fehler – eine geniale SQL-Anweisung wird erzeugt, aber nicht abgesetzt – wird erfahrungsgemäß amAnfang oft gemacht. Denken Sie immer an Abbildung 11.2, PHP ist nur das Medium!

12.1.4 Die Anzeige

Die Anzeige des aktuellen Inhalts der Datenbank wird durch eine `SELECT`-Anweisung implementiert. Sie lautet in unserem Listing 12.1, Zeile 37, 38

```
SELECT * FROM adressen ORDER BY Name
```

und liefert eine nach Namen sortierte Ergebnismenge `$result`. Im Rest des Programms wird entsprechend die Ergebnismenge ausgegeben.

Bemerkung. Die Anzeige von Inhalten einer Datenbank sollten in einem PHP-Programm immer am Schluss erfolgen, da sonst von dem Programm durchgeführte Änderungen der Datenbank (mit `INSERT`, `UPDATE`, `DELETE`) nicht aktualisiert angezeigt werden.

12.2 Version 2.0: Eintragen-Skript mit Funktionen und Formatierungen

Modifizieren wir unser Programm nun, indem wir die Ablauflogik mit Funktionen implementieren und die Ausgaben strukturierter formatieren. Mit Funktionen kann man das eigentliche PHP-Skript, als „Hauptprogramm“, relativ kurz gestalten.

Listing 12.2: Adressen aus Datenbank editieren, Hauptprogramm

```
<!-- Adressen in DB eintragen. Datei: adresseDB3.php -->
<form action="<?php echo $_SERVER['PHP_SELF']?>" method="POST">
<?php
  require_once("./adressDBFunktionen.inc.php");
  require_once("db_login.inc.php");
  $mysqli = login("adressDB");

  if (isset($_POST['eintragen'])) {
     eintragen();
  } else {
     if (isset($_POST['speichern'])) {
        speichern($mysqli, "adressen", $_POST);
     }
     auflisten($mysqli);
  }
?>
</form>
```

Im Wesentlichen wird also ein Formular erzeugt und nach dem Aufbau der Datenbankverbindung je nach Inhalt der `$_POST`-Variablen eine der Funktionen `auflisten`, `eintragen` oder `speichern` aufgerufen. Diese Funktionen sind ausgelagert in der Datei

`adressDBFunktionen.inc.php`

in der gleichen Verzeichnisebene wie das PHP-Skript, zu erkennen am Pfad des `require_once`-Befehls: Beginnt der Verzeichnispfad mit einem Punkt vor dem Slash (`./`), sucht der Server in derselben Verzeichnisebene, beginnt er mit zwei Punkten (`../`), wechselt er in eine Ebene darüber, genau wie die Verlinkung relativer Pfade in HTML. (Wie dort kann man aber `./` auch einfach weglassen.)

Ein so gestaltetes Hauptprogramm hat den Vorteil, dass sein Ablauf leicht zu erkennen ist und die Komplexität auf einzelnen Funktionen verteilt wird. Dafür müssen allerdings die Funktionen die notwendigen Daten auch als Parameter übergeben bekommen. Die Funktion `eintragen` benötigt keine Eingabedaten und hat daher keinen Parameter. Die Funktion `auflisten` dagegen muss den aktuellen Verbindungsdaten zur Datenbank kennen und bekommt daher als Eingabeparameter das Objekt `$mysqli` übergeben. Die Funktion `speichern` schließlich benötigt die meisten Informationen, sie bekommt als Eingabe die aktuelle Datenbankverbindung durch `$mysqli`, die Tabelle, in die zu speichern ist, und die zu speichernden Daten `$_POST` aus dem Formular.

Schauen wir uns die Funktionen nun im Detail an.

Listing 12.3: Die Funktionen der Adressdatenbank aus Listing 12.2

```
<!-- Funktionen der Adress-DB. Datei: adressDBFunktionen.inc.php -->
<?php
function auflisten($mysqli) {
?>
  <h2>Adressen als 2-dimensionales Array</h2>
  <table border="1">
<?php
  $sql = "SELECT * FROM adressen ORDER BY Name";
  $result = $mysqli->query($sql);

  // Tabellenkopf mit den Feldnamen als Spaltenbezeichnungen:
  echo "   <tr>\n";
  while ( $attribut = $result->fetch_field() ) {
     echo "    <th>$attribut->name</th>\n";
  }
  echo "   </tr>\n";

  // Datensätze aus der Datenbank, spezielle Ausgabe für email und web:
  while ( $adresse = $result->fetch_assoc() ) {
    echo "   <tr>\n";
    foreach ( $adresse as $key => $value ) {
      if ( $key == "email" ) {
         echo "    <td><a href=\"mailto:$value\">$value</a></td>\n";
      } else if ( $key == "web" ) {
         echo "    <td><a href=\"http://$value\">$value</a></td>\n";
      } else {
         echo "    <td>$value</td> \n";
      }
    }
    echo "   </tr>\n";
  }
  ?>
  </table>
  <p style="text-align:center">
    <input type="submit" name="eintragen" value="Adresse eintragen"/>
  </p>
<?php
}  // Ende function auflisten
```

```
function eintragen() {
?>
  <table border="0">
   <tr><td>Vorname:</td><td><input type="text" name="Vorname"/></td></tr>
   <tr><td>Name:</td><td><input type="text" name="Name"/></td></tr>
   <tr>
    <td>PLZ/Ort:</td>
    <td>
     <input type="text" name="PLZ" size="7"/>
     <input type="text" name="Ort"/>
    </td>
   </tr>
   <tr><td>Strasse:</td><td><input type="text" name="Strasse"/></td></tr>
   <tr><td>Web:</td><td><input type="text" name="web"/></td></tr>
   <tr><td>e-mail:</td><td><input type="text" name="email"/></td></tr>
 </table>
 <p><input type="submit" name="speichern" value="Speichern"/></p>
<?php
} // Ende function eintragen

function speichern($mysqli, $tabelle, $daten) {
  $spalten = array();
  $werte = array();
  foreach ( $daten as $key => $value ) {
    if ( $key != "speichern" ) {
        $spalten[] = $key;
        $value = $mysqli->escape_string($value);
        $werte[] = "'$value'";
    }
  }
  $sql  = "INSERT INTO $tabelle ";
  $sql .= "(" . implode(",", $spalten ) . ")";
  $sql .= " VALUES ";
  $sql .= "(" . implode(",", $werte )   . ")";
  $mysqli->query($sql);
}
?>
```

Die Funktion `auflisten` entspricht im Wesentlichen dem Programm in Listing 11.2. Einzige Änderungen sind das `SELECT`-Statement, das jetzt

```
SELECT * FROM adressen ORDER BY Name
```

lautet und eine nach Namen sortierte Ergebnismenge `$result` liefert. Außerdem ist nun ein Submit-Button namens `eintragen` eingefügt, der an sich nichts anderes bewirkt, als dass das Hauptskript wieder aufgerufen wird. Die `$_POST`-Variable enthält nun als einziges Datenfeld `$_POST[eintragen]`, ist also nicht mehr leer.

Damit wird nun in dem Skript der zweite `if`-Zweig durchlaufen, d.h. die Funktion `eintragen` aufgerufen. Diese Funktion baut nun ein einfaches Eingabeformular auf, in dem die Adressdaten

eingegeben werden können. Der Submit-Button heißt hier nun `speichern`, und damit wird nach Anklicken das Hauptskript wieder aufgerufen.

Diesmal wird der dritte `if`-Zweig durchlaufen und die Funktion `speichern` aufgerufen. Diese Funktion übernimmt die Dateninhalte der `$_POST`-Variablen des Eingabeformulars, um daraus ein `INSERT`-Statement zu bilden. Mit `$mysqli->query` wird dieser SQL-Befehl an die Datenbank geschickt, in der dann der Datensatz gespeichert wird. Diese Variante bildet zwei Arrays, je eins mit den Spaltennamen und eins mit den Inhalten, und verwendet den `implode`-Befehl, um daraus die Klammerausdrücke für ein `INSERT` zu erzeugen. Nachdem dieser `INSERT`-Befehl als String gebildet worden ist, wird er mit `$mysqli->query` an die Datenbank geschickt, dort von dem Datenbanksystem als SQL interpretiert und ausgeführt (wenn denn kein Syntaxfehler in SQL vorliegt ...). Danach wird direkt die Funktion `auflisten` aufgerufen, die die nun aktualisierte Tabelle an den Browser schickt. Viola!

12.3 SQL Injection entschärfen mit `escape_string`

Ein Sicherheitsrisiko jeder Webdatenbank ist die „SQL-Injektion“, also der Versuch eines Angreifers, über eine Formulareingabe die Datenbank abzufragen oder gar zu manipulieren. Ein Schutz vor solchen Angriffen ist nicht trivial, allerdings bietet PHP mit der Funktion `escape_string` einen einfachen Abwehrmechanismus an. Diese Funktion durchläuft den übergebenen String und ersetzt sicherheitskritische Zeichen, die zu einer SQL-Injektion führen könnten. Man sollte generell alle Eingaben analog folgendem Mechanismus entschärfen:

```
$eingabe = $mysqli->escape_string($_POST["eingabe"]);
// irgendeine SQL-Anweisung:
$sql = "SELECT * FROM tabelle WHERE spalte='$eingabe'";
$result = $mysqli->query($sql);  // zur Datenbank senden
```

SQL-Injektionen werden wir in Kapitel 14 genauer behandeln.

12.4 Zusammenfassung

- Der Ablauf eines Skripts, das sich mehrmals selbst aufruft, wird mit `if`-Anweisungen gesteuert, die den Inhalt der Submit-Buttons prüfen. Wesentlich ist also in diesem Fall, dass jeder Submit-Button einen eigenen Namen trägt.

- Der zeitliche Ablauf des Skripts aus Sicht des Benutzers entspricht nicht der Struktur der `if`-Anweisungen, denn zuerst werden spezielle Belegungen der Submit-Buttons abgefragt, dann erst kommt der allgemeine Teil.

- Die Anzeige des aktuellen Datenbankinhalts sollte erst am Ende des Skripts erfolgen, nachdem etwaige Änderungen in der Datenbank erfolgt sind. Würde man in Listing 12.1 die Anzeige der Datenbankinhalte *vor* der Speicherung der Eingabedaten programmieren, wäre zwar der neue Eintrag korrekt in der Datenbank gespeichert, der Anwender sähe aber irreführenderweise noch den alten Stand der Datenbank.

- Man sollte die Namen der `<input>`-Tags in einem Eingabeformular genauso nennen wie die entsprechenden Feldnamen der Datenbanktabelle. Das erleichtert einerseits die Programmübersicht, andererseits die Synchronisation und Verwaltung bei Änderungen der Datenbankstruktur.

- Erstellt man ein SQL-Statement als einen String (z.B. namens `$sql`), so muss er irgendwann mit `$mysqli->query` an die Datenbank gesendet werden. ;-)

13

Geschäftslogik nach SQL verlagern

Kapitelübersicht

Wir haben in den letzten Kapiteln betrachtet, wie webbasierte Datenbanken und Zugriffe darauf mit PHP programmiert und komplexere Abläufe umgesetzt werden können. In diesem Abschnitt werden wir untersuchen, wo Informationen in einer webbasierten Datenbankanwendung grundsätzlich verarbeitet werden können und wo die Geschäftslogik der Anwendung am effizientesten implementiert wird. Wir werden sehen, dass die Geschäftslogik wesentlich auf dem Datenmodell der Anwendung basiert und daher so weit wie möglich auf dem Datenbankserver implementiert werden sollte.[1] Also ganz nach dem Motto: *Lass die Datenbank arbeiten!*

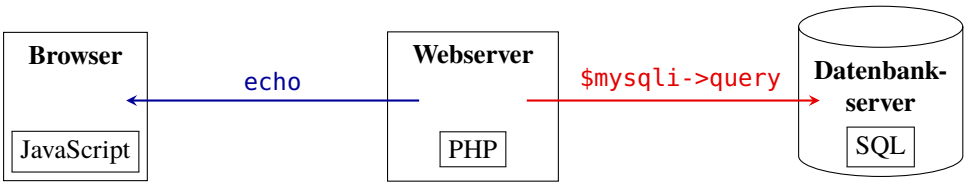
Wir betrachten dazu in diesem Kapitel als Fallbeispiel eine einfache datenbankgestützte Online-Auktion. Wir werden die eigentliche Geschäftslogik der Auktion in Datenbankabfragen verlagern, also die Überprüfungen, ob ein Gebot rechtzeitig kommt und ob sein Betrag hoch genug ist, sowie die Speicherung eines gültigen Gebots. In PHP wird nur noch die reine Ablauflogik der Auktion und die Darstellung der Anwendung im Browser programmiert.

Durch kluge SQL-Anweisungen lässt sich so der Quelltext in PHP stark verkürzen und die Laufzeit der Anwendung beschleunigen. Zudem sind die gängigen Datenbankmanagementsysteme mit optimierten Such- und Sortierfunktionen ausgestattet, die man getrost ausnutzen sollte.

[1] Das Framework Angular verfolgt mit seinem Konzept der Single-Page-Anwendungen (SPA) einen radikal anderen Ansatz und verlagert wesentliche Teile der Geschäftslogik *und* der Ablauflogik weg vom Server auf den Client. Bei diesen Anwendungen liegt die Priorität allerdings auf dem *Look-and-Feel* aus Sicht des Users und auf der Entlastung des Servers. Vgl. dazu z. B. Malcher et al. (2020).

13.1 Strukturierung der Verarbeitung von Informationen

Die Architektur webbasierter Datenbanken ist in drei Schichten gegliedert, wie wir bereits in Abbildung 11.1 gesehen haben. Daher gibt es ganz prinzipiell auch drei Stellen, an denen Informationen verarbeitet und Logik programmiert werden kann: (a) im Browser auf Basis von JavaScript, (b) im Webserver mit PHP, oder (c) im Datenbankserver mit SQL.



Aufgabenverteilung anhand domänenspezifischer Informationen

Alle drei Programmiersprachen werden in der Praxis auch eingebunden. Allerdings hat jede ihre eigene Anwendungsdomäne mit ihren spezifischen Informationen. So kann der Browser durch JavaScript direkt auf Usereingaben reagieren, bevor Abfragen und Daten an den Webserver geschickt werden. Der Webserver wiederum verfügt durch seine zentrale Vermittlerrolle vor allem über Informationen zur Steuerung des Ablaufs. Die Datenbank schließlich bildet durch ihr Datenmodell die Geschäftslogik ab und hat damit alle Informationen über die Entitäten der Anwendung. Zusammengefasst ergibt sich damit die Übersicht in Tabelle 13.1.

	Browser	**Webserver**	**Datenbankserver**
Informationen	User-Eingaben	Ablauflogik	Geschäftslogik
Aufgaben	Plausibilitäts-prüfungen bei der Eingabe	Weiterleitung von Daten	fachliche Konsistenzprüfungen

Tabelle 13.1: Verteilung der Aufgaben auf die drei Schichten einer webbasierten Datenbankanwendung anhand ihrer domänenspezifischen Informationen.

Verdeutlichen wir uns dies an einem typischen Beispiel. In einem Webshop sollte der Einkauf eines Artikels natürlich nur erfolgen, wenn er noch im Bestand vorrätig ist. Würden wir diese Überprüfung den Webserver machen lassen, so müsste dieser erst beim Datenbankserver „anfragen“, bevor er den Einkauf durchführen lässt oder verweigern muss. Praktisch wären also zwei Datenbankzugriffe nötig, erst ein SELECT, dann ein INSERT bzw. UPDATE. Gleiches gilt, wenn der Browser diese Überprüfung machen würde, nur muss dazu der Webserver noch zusätzlich aktiv werden. Die Datenbank dagegen kann durch einen einzigen SQL-Befehl mit Hilfe der WHERE-Klausel den Bestand überprüfen und den Einkauf korrekt verarbeiten.

Wie die Programmierung einer solchen Anwendung aussehen kann, werden wir in diesem Kapitel betrachten. Wir werden uns jedoch dabei auf die Implementierung der Geschäftslogik konzentrieren und Plausibiltätsprüfungen von User-Eingaben hier außer Acht lassen.

13.2 Eine einfache Auktion

Wir werden in diesem Abschnitt eine kleine Anwendung beschreiben, die Gebote geordnet nach ihrem Laufzeitende anzeigt und die Eingabe eines neuen Gebots ermöglicht, siehe Abbildung 13.1. Die Eingabe der Auktionen sowie allgemein die Administration der Tabelle erfolge über ein anderes PHP-Skript, das wir an dieser Stelle nicht programmieren wollen.

Es sollen nur gültige Auktionen angezeigt werden, also nur solche, die beim Aufruf der Seite noch nicht abgelaufen sind. Hat ein Bieter sein Gebot abgegeben und sollte inzwischen bereits ein höheres Gebot erfolgt oder die Auktion insgesamt schon abgelaufen sein, so soll der

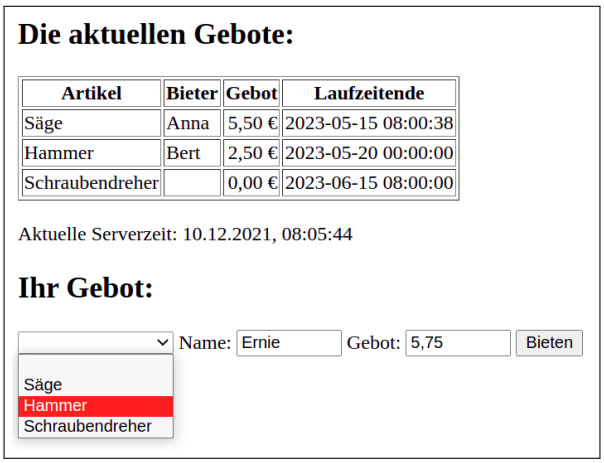


Abbildung 13.1: Eingabemaske einer einfachen Online-Auktion

Anwender am unteren Ende der Seite eine entsprechende Meldung erhalten und das Gebot nicht gespeichert werden. Ist ein Gebot korrekt abgegeben, soll es in der Tabelle gespeichert und die aktualisierte Seite angezeigt werden.

13.2.1 Das Datenmodell

Wir betrachten die Anwendung als Implementierung eines Geschäftsprozesses der Auktion, das Bieten. Dieser Prozess sieht aus Anwendersicht vor, dass ein Bieter aus einer Auswahl von Artikeln wählen kann, für die es laufende Gebote gibt. Gibt er für einen der Artikel ein Gebot ab, das hoch genug ist, wird er als Bieter für diesen Artikel eingetragen.

Für das Datenmodell bedeutet das, dass jede Auktion als eigener Datensatz in einer zentralen Tabelle gespeichert werden und deren Bieter veränderbar sein muss. Daher kann er nicht Teil des Primärschlüssels dieser Tabelle sein. Was aber muss überhaupt in dieser Tabelle gespeichert sein? Natürlich der Artikel, um den es geht, der aktuelle Bieter, das Gebot und das Laufzeitende des Auktion. Der Relationentyp der Tabelle könnte also wie folgt aussehen:

Geschäftslogik und Datenmodell

gebote(artikel, bieter, gebot, ende)

Der Primärschlüssel ist hier der Artikel. Das ist hier auch sinnvoll, da Auktionen für zwei gleiche Artikel die Konsistenz des Geschäftsproesses zerstören würden. Wir sehen hier exemplarisch, wie die Geschäftslogik bereits auf das Datenmodell wirkt.

Das Datenmodell ist hier recht schlank geraten. Für die vorgesehene Anwendung reicht eine einzige Tabelle aber auch aus. In der Praxis ist das natürlich sehr unrealistisch, denn es werden üblicherweise viel mehr und komplexere Daten zu verarbeiten sein. So wären beispielsweise für eine echte Auktion sicher eigene Tabellen für die Bieter und die Artikel angebracht, so dass die Attribute `artikel` und `bieter` hier als Fremdschlüssel auf deren Primärschlüssel verweisen.

Datenmodelle in der Praxis sind komplexer

Zur Erstellung der Tabelle müssen noch die Datentypen der Attribute festgelegt werden. Artikel und Bieter sollten hier Strings sein, das Gebot eine Zahl mit zwei Nachkommastellen, und das Laufzeitende ein Datum mit Uhrzeit. Wir können die Tabelle also mit folgender SQL-Anweisung erstellen:

```
CREATE TABLE gebote (
  artikel varchar(30) PRIMARY KEY,
  bieter varchar(30),
  gebot decimal(10, 2),
  ende timestamp NULL DEFAULT NULL
```

```
);
```

Eine Anmerkung zu den Initialwerten der Attribute: Bei Eintragung eines neuen Artikels können die anderen Attribute leer sein. Das ist aus Sicht der Geschäftslogik sicher sinnvoll, denn am Anfang gibt es ja noch kein Gebot. Eine Anmerkung zu dem Attribut `ende`: Wollen wir die Eintragung eines Artikels ermöglichen, ohne das Laufzeitende der Auktion festzulegen, müssen in MariaDB hinter dem Datentyp `timestamp` die Anweisungen `NULL DEFAULT NULL` stehen, da er für diese Datenbank standardmäßig nicht `NULL` sein darf. Wollen wir jedoch erzwingen, dass ein Artikel nur mit einem Laufzeitende eingestellt werden kann, so müssen wir stattdessen `NOT NULL` schreiben. Sie sehen: die Geschäftslogik kann Konsequenzen bis in die technische Feinheiten haben!

Ist die Tabellenstruktur festgelegt, so können wir mit den folgenden SQL-Anweisungen erste Testfälle eintragen:

```
INSERT INTO gebote (artikel, bieter, gebot, ende) VALUES
  ('Hammer', 'Bert', 2.5, '2023-05-20 00:00');
INSERT INTO gebote (artikel, bieter, gebot, ende) VALUES
  ('Säge', 'Anna', 5.5, '2023-05-15 08:00:38');
INSERT INTO gebote (artikel, bieter, gebot, ende) VALUES
  ('Zange', 'Cindy', '9.80', '2021-03-01 12:00');
INSERT INTO gebote (artikel, ende) VALUES
  ('Schraubendreher', '2023-06-15 08:00');
```

Dabei sollten die Testdaten ein möglichst repräsentatives Spektrum abdecken, hier also Auktionen mit Laufzeiten, die erst in der Zukunft enden oder die bereits abgelaufen sind.

13.2.2 Der grobe Ablauf des PHP-Skripts

Bevor wir uns den vollständigen Quelltext des PHP-Skripts anschauen, das diese Anwendung ausführt, überlegen wir uns den groben Ablauf. Gemäß Abbildung 13.1 soll der Anwender stets die Liste der aktuell gültigen Gebote sehen und am unteren Teil des Fensters seine Eingaben machen können. Da dies auch beim ersten Aufruf passieren soll, muss also die Anzeige des Datenbankinhalts und des Eingabeformulars ohne Bedingung erfolgen. Schickt der Anwender jedoch sein Gebot ab, so soll die Datenbank entsprechend geändert werden, *bevor* sie angezeigt wird. Die grobe Struktur des Skripts sieht also wie folgt aus.

```
if ( isset($_POST[...] ) {
   // ... in Tabelle eintragen, wenn Gebot gültig & hoch genug ...
}
// ... Anzeige der aktuell gültigen Gebote ...
// ... Formular zur Eingabe des neuen Gebots ...
```

Ein Grobentwurf muss nicht alle Details klären

Hierbei repräsentiert `$_POST[...]` die relevanten Informationen des Eingabeformulars zum Abgeben eines Gebots. Welche das sinnvollerweise sein sollten, werden wir uns zu einem späteren Zeitpunkt überlegen. Merke: *Für den ersten Grobentwurf einer Anwendung muss nicht jedes Detail schon geklärt sein.*

Ein kleines Problem stellen die Fehlermeldungen bei einem ungültig abgegebenen Gebot dar. Sie sollen am unteren Rand unter dem Formular erscheinen, zu diesem Zeitpunkt kommen wir in PHP aber so ohne Weiteres nicht an die Information, ob ein Fehler passiert ist. Was tun? Die einfachste Lösung ist, an derjenigen Stelle im Programm, an der der Fehler auftritt, eine entsprechende Fehlermeldung in einer Variablen `$fehlermeldung` zu speichern und diese dann am Ende auszugeben.

13.2.3 `SELECT`-Anweisungen mit `WHERE`-Klausel

Die Anzeige der aktuell gültigen Gebote, geordnet nach ihrem Ablaufzeitpunkt, ist die erste Hürde unseres Programms. Eine mögliche Lösung wäre, zunächst per `SELECT`-Anweisung alle Datensätze auszuwählen und mit einer `if`-Bedingung in PHP nur diejenigen auszugeben, die aktuell gültig sind. Zwar würde das im Prinzip funktionieren, wäre jedoch keine so gute Lösung. Insbesondere bei einer größeren Anzahl von Datenbankeinträgen und einer gut besuchten Auktion kann dadurch der Arbeitsspeicher des Webservers gehörig in Anspruch genommen werden.

Lassen Sie die Datenbank arbeiten! Sorgen Sie insbesondere dafür, dass bei einer `SELECT`-Anweisung auch nur diejenigen Daten ausgewählt werden, die sie benötigen. Damit wird der Webserver entlastet, und der Programmieraufwand in PHP wird geringer. Wir sollten also nur aktuell gültige Auktionen selektieren. Was heißt aber „aktuell gültig“? Der Ablaufzeitpunkt der Auktion in der Spalte `ende` muss größer sein als der aktuelle Zeitpunkt, den die Datenbank mit der Funktion `NOW()` bestimmt:

```
$sql = "SELECT * FROM gebote WHERE ende > NOW()";
```

Statt `NOW()` kann auch der Ausdruck `CURRENT_TIMESTAMP` verwendet werden:

```
$sql = "SELECT * FROM gebote WHERE ende > CURRENT_TIMESTAMP";
```

Das Ergebnis könnten wir nun in PHP sortieren und dann ausgeben. Aber auch hier: Lassen Sie die Datenbank arbeiten! Mit der Option `ORDER BY` kann man schon die Selektion von der Datenbank sortieren lassen. Entsprechend könnte eine vereinfachte Ausgabe also wie folgt aussehen.

```
$sql = "SELECT * FROM gebote WHERE ende > NOW() ORDER BY ende";
$result = $mysqli->query($sql);

while ( $auktion = $result->fetch_object() ) {
   echo "Artikel: $auktion->artikel, Bieter: $auktion->bieter ";
   echo "Gebot: ".number_format($auktion->gebot, 2, ",", ".")." &euro; ";
   echo "Ablauf: ".$auktion->ende."<br>";
}
```

Das liefert zwar noch nicht die schöne Tabellenform aus Abbildung 13.1, aber es funktioniert. Insbesondere werden alle Gebote mit zwei Nachkommastellen und einem Komma als Dezimaltrenner ausgegeben, und der Zeitstempel erscheint in einem lesbaren Format.

Das Formular stellt bis auf die Select-Box keine Schwierigkeiten dar, wir nennen die Input-Tags genau wie die Tabellenfelder. Für die Select-Box müssen wir kurz einen Schritt zurück, denn es sollen nur diejenigen Gebote auswählbar sein, die auch angezeigt werden, diese Information haben wir jedoch zur Laufzeit an dieser Stelle nicht mehr. Wir müssen uns also in der obigen `while`-Schleife für jeden Eintrag die Artikelbezeichnung (hier der Primärschlüssel) in einem Array speichern:

```
while ( $row = $result->fetch_object) ) {
   // ... echos (s.o.) ...
   $artikel[] = $row->artikel;  // Speichern der Artikelbezeichnungen
}
```

13.2.4 Die `UPDATE`-Anweisung

Die letzte Hürde unserer Anwendung ist die Eintragung eines neuen Gebots. Zunächst ist klar, dass wir eine `UPDATE`-Anweisung verwenden müssen, denn die Auktion besteht ja schon, es müssen nur die Daten des Gebots aktualisiert werden. Zur Optimierung der `UPDATE`-Anweisung müssen wir noch überlegen, welche Kriterien der zu ändernde Datensatz erfüllen muss. Zunächst natürlich muss die Bezeichnung der Eingabe, also `$_POST["artikel"]`, mit dem Wert des Tabellenfeldes `artikel` als Primärschlüssel übereinstimmen, aber daneben muss die Auktion zum Zeitpunkt der Eintragung immer noch gültig sein, und das Gebot `$_POST[gebot]` der Eingabe muss echt größer als das Gebot der Datenbank sein. Verknüpfen wir diese drei Bedingungen einfach mit `AND`, und verwenden diese Bedingung als `WHERE`-Klausel, so erhalten wir unsere `UPDATE`-Anweisung:

```
// Ersetzt eingegebenes Komma durch Punkt:
$gebot = str_replace(",", ".", $_POST[gebot]);
$sql  = "UPDATE gebote SET ";
$sql .= "bieter = '$_POST[bieter]', ";
$sql .= "gebot = '$gebot' ";
$sql .= "WHERE artikel = '$_POST[artikel]' ";
$sql .= "AND ende > NOW() ";
$sql .= "AND gebot < '$_POST[gebot]'";
```

Beim Eintragen des Gebotsbetrags wird hier eine vom Anwender mit Komma eingegebene Zahl in eine Float-Zahl mit Punkt umgewandelt. Gibt der Anwender hier etwas ein, was trotz dieser Maßnahme keine Zahl ergibt, so endet der `UPDATE`-Befehl mit einem Fehler und ändert den alten Eintrag nicht. Alles in Allem ergibt sich damit das PHP-Skript in Listing 13.1.

Listing 13.1: Eine einfache Auktion

```
<!DOCTYPE html>
<html>
<head><title>Auktion</title></head>
<body>
<?php
require_once("db_login.inc.php");
$mysqli = login("auktion");

if (isset($_POST["artikel"]) && isset($_POST["bieter"]) && isset($_POST["gebot"])) {
   // Ersetzt eingegebenes Komma durch Punkt:
   $gebot = str_replace(",", ".", $_POST["gebot"]);

   $sql  = "UPDATE gebote SET ";
   $sql .= "bieter = '$_POST[bieter]', ";
   $sql .= "gebot = '$gebot' ";
   $sql .= "WHERE artikel = '$_POST[artikel]' ";
   $sql .= "AND ende > NOW() ";
   $sql .= "AND gebot < '$gebot'";
   $mysqli->query($sql);
   if ($mysqli->affected_rows == 0 ) { // falls kein Gebot gefunden wurde:
      $fehlermeldung = "Ihr Gebot ist abgelaufen oder unterboten!";
   }
}

// -- Anzeige der aktuell gültigen Auktionen: --------
```

```
echo "<h2>Die aktuellen Gebote:</h2>";
echo "<table border=\"1\"><tr>";
echo "<th>Artikel</th><th>Bieter</th><th>Gebot</th><th>Laufzeitende</th>";
echo "</tr>";
$sql  = "SELECT * FROM gebote WHERE ende > NOW() ORDER BY ende";
$result = $mysqli->query($sql);

while ( $auktion = $result->fetch_object() ) {
   echo "<tr>";
   echo "<td>$auktion->artikel</td><td>$auktion->bieter</td>";
   echo "<td style=\"text-align:right\">";
   echo number_format($auktion->gebot, 2, ",", ".") . " &euro;</td>";
   echo "<td>" . $auktion->ende . "</td>";
   echo "</tr>";
   $artikel[] = $auktion->artikel;  // Speichern der Artikelbezeichnungen
}
echo "</table>";
$systemzeit = time();
echo "<br/>Aktuelle Serverzeit: " . date("d.m.Y, H:i:s", $systemzeit);
?>
   <form action="<?php echo $_SERVER['PHP_SELF'];?>" method="POST">
     <h2>Ihr Gebot:</h2>
     <select name="artikel"> <option></option>
<?php
for ($i=0; $i < sizeof($artikel); $i++) {
   echo "<option value=\"$artikel[$i]\"";
   if ( isset($_POST["artikel"]) && $artikel[$i] == $_POST["artikel"] ) {
      echo " selected=\"selected\"";
   }
   echo ">$artikel[$i]</option>\n";
}
?>
     </select>
     Name: <input type="text" name="bieter" size="5" value="<?php
       echo isset($_POST["bieter"]) ? $_POST["bieter"] : "";
     ?>"/>
     Gebot: <input type="text" size="5" name="gebot" />
     <input type="submit" value="Bieten"/>
   </form>
<?php
   if (isset($fehlermeldung)) {
      echo "<p style=\"color:red\">$fehlermeldung</p>";
   }
?>
</body>
</html>
```

Da dieses Programm für die Auktion – in der Terminologie der Wirtschaftswissenschaften – den Geschäftsprozess des Bietens darstellt, sollte es in einer Datei mit dem Namen `bieten.php` oder ähnlich gespeichert sein. Wollen wir weitere Geschäftsprozesse implementieren, so können sie in eigenen, entsprechend benannten Dateien programmiert werden, beispielsweise `artikel_eintragen.php` für das Einpflegen von neuen Artikeln. Auf diese Weise können wir die Abläufe aller notwendigen Geschäftsprozesse über die Dateistruktur der PHP-Skripte abbilden.

Geschäftsprozesse in Dateistruktur abbilden

Haben wir zudem die Geschäftslogik in SQL implementiert, bleibt damit am Ende auch eine komplexe Anwendung noch wartbar.

13.3 Zusammenfassung

Wir haben in diesem Kapitel anhand eines einfachen Beispiels wichtige Ansätze zum Entwurf und zur Implementierung wartbarer webbasierter Datenbankanwendungen kennengelernt. Eine Grundlage dazu ist die Trennung von Ablauflogik und Geschäftslogik eines Geschäftsprozesses. Die Ablauflogik wird in PHP programmiert, die Geschäftslogik in SQL.

Bei der Entwicklung einer webbasierten Datenbank steht am Anfang das Datenmodell. Es ist die Basis für die gesamte Anwendung, eine Änderung dort kann massive Änderungen der PHP-Quelltexte erfordern. Es sollte daher gut durchdacht sein. Es muss die Geschäftslogik der Anwendung konsistent abbilden, die sich bis in technische Feinheiten wie die Datentypen oder die Primär- und Fremdschlüssel der Tabellen auswirken kann.

Steht die Datenbank, kann schrittweise der Ablauf eines Geschäftsprozesses in PHP implementiert werden. Auch hier sollte die Geschäftslogik so weit wie möglich auf die Datenbank verlagert werden, also in SQL programmiert werden. In einer einzelnen PHP-Datei sollten nicht mehrere Geschäftsprozesse implementiert werden, sondern höchstens einer oder nur einzelne Aktivitäten, je nach Komplexität. Die Namen der Dateien sollten die darin implementierten Prozesse widerspiegeln. Damit bildet die Datei- und Verzeichnisstruktur die Struktur der Geschäftsprozesse ab und das System bleibt wartbar und versionisierbar.

Zum Schluss kann man die Darstellungsschicht verfeinern, sie z. B. um Formatierungen erweitern, die das Erscheinungsbild im Browser bestimmen.

14 Informationssicherheit

Kapitelübersicht

Informationssicherheit bezeichnet Eigenschaften eines informationsverarbeitenden Systems, die seine Vertraulichkeit, Verfügbarkeit und Integrität schützen. Informationssicherheit dient insbesondere dem Schutz vor Gefahren bzw. Bedrohungen, der Vermeidung von wirtschaftlichen Schäden und der Minimierung von Risiken.

Jedes Informationssystem hat Schwachstellen und ist damit verwundbar. Ein *Angriffsvektor* bezeichnet einen möglichen Angriffsweg oder eine Angriffstechnik, die ein unbefugter Eindringling nehmen kann, um ein fremdes System zu kompromittieren, das heißt unbefugt einzudringen und es danach entweder zu übernehmen oder zumindest für eigene Zwecke zu missbrauchen. Meist werden dazu bekannte Sicherheitslücken des angegriffenenen Systems ausgenutzt. Ein solches Ausnutzen bezeichnet man als *Exploit*.

14.1 SQL-Injektion: Angriffe auf Daten

Unter einer *SQL-Injektion* versteht man das Absetzen von SQL-Befehlen über ein Texteingabefeld, wie z.B. für Benutzername oder Passwort, oder über die `GET`-Methode. Der Angreifer versucht damit, eigene Datenbankbefehle über das Eingabeformular einzuschleusen und so die Kontrolle über die SQL-Datenbank oder gar den Datenbankserver zu erlangen.

Wie kann das gelingen? Kern dieser Angriffe sind die drei Funktionszeichen in SQL,

```
; , \ , ' ,
```

die Kommentarzeichen `--` oder `#`, sowie (bei SQL-Injektion über `GET`) die URL-Codierung bestimmter Sonderzeichen, z. B. `%20` für das Leerzeichen.

14.1.1 Einfaches Beispiel

Nehmen wir an, wir hätten ein einfaches Eingabeformular mit zwei Textfeldern namens `name` und `passwort` als Login-Maske, welches nach dem Absenden ein Skript aufruft, das die folgende SQL-Anweisung ausführt:

```
$sql  = "SELECT * FROM user";
$sql .= "WHERE name = '$_POST[name]'";
$sql .= AND password = '$_POST[password]}'";
```

Wenn dieses SQL-Kommando ohne weitere Prüfung an die Datenbank geschickt wird, kann ein Angreifer sich ohne Kenntnis eines Usernamens oder Passwortes in das Systen einloggen, indem er einfach in das Textfenster eingibt:

Name:	`' OR 1=1 --`
Passwort:	

(14.1)

Daraus setzt das aufgerufene PHP-Skript dann nämlich die folgende SQL-Anweisung zusammen:

```
SELECT * FROM user WHERE name = '' OR 1=1 -- ' AND password = ''
```

Damit werden *alle User* der Datenbanktabelle mit ihrem Passwort angezeigt! Denn nun ist die `WHERE`-Bedingung eine ODER-Verknüpfung der Bedingung `name = ''`, die sicher falsch ist, und der Bedingung `1=1`, die sicher wahr ist, und damit lautet das SQL-Kommando effektiv `SELECT * FROM user`.

Auf ähnliche Weise kann es einem Angreifer gelingen, Kontrolle über die Datenbank zu bekommen und Daten zu manipulieren oder gar die ganze Datenbank zu löschen. Fatalerweise wäre es beispielsweise möglich, die Preise für in einem Webshop angebotene Artikel zu verändern und ihn dann zu bestellen.

14.1.2 Gegenmaßnahmen

Es ist nicht schwer, PHP-Programme vor SQL-Injektionen zu schützen. Wie so oft ist das hauptsächliche Problem die Unwissenheit oder Arglosigkeit des Programmierers über diese Art von Angriffen. Eine Möglichkeit, sich in PHP gegen SQL-Injektionen zu schützen, ist die Verwendung der Funktion

```
$mysqli->escape_string(string)
```

Diese Funktion stellt den folgenden Zeichen einfach einen Backslash voran: `NULL`, `\x00`, `\n`, `\r`, `\`, `'`, `"` und `\x1a`. Diese Voranstellung heißt *Maskierung (escaping)*. Damit lautet eine sichere Variante der obigen Erstellung einer SQL-Anweisung:

```
$sql  = "SELECT * FROM user ";
$sql .= "WHERE name='".  $mysqli->escape_string($_POST["name"])    ."' ";
$sql .= "AND password='".$mysqli->escape_string($_POST["password"])."'";
```

Die Eingabe (14.1) läuft nun ins Leere, sie ergibt nun nach dem Absenden:

```
SELECT * FROM user WHERE name = '\' OR 1=1 --' AND password = ''
```

Einen User mit dem Namen `\' OR 1=1 --` ohne Passwort wird es jedoch wohl kaum geben, und so klappt die Anmeldung auf diese Weise nicht.

Ein kleines Programm, das die Wirkung der Maskierung demonstriert, lautet wie folgt.

```
<h1>Demonstration SQL-Injektion</h1>

<form action="<?php echo $_SERVER['PHP_SELF'];?>" method="post">
   User:     <input type="text" name="user"/>
   Passwort: <input type="text" name="passwort"/>
   <input type="submit" value="Anmelden"/>
</form>

<p style="color:red">
<?php
   require_once("db_login.inc.php");
   $mysqli = login("login");

   if (!empty($_POST["user"]) && !empty($_POST["passwort"])) {
      $user     = $mysqli->escape_string($_POST["user"]);
      $passwort = $mysqli->escape_string($_POST["passwort"]);

      $sql  = "SELECT * FROM users WHERE user='$user' "
      $sql .= "AND passwort='$passwort'";
      echo "SQL mit Maskierung: $sql<br/>";
   }
?>
</p>
```

14.1.3 Prepared Statements mit PDO

PHP Data Objects (PDO) ist eine PHP-Erweiterung, die eine Schnittstelle für verschiedene Datenbanken darstellt, um mit PHP auf Datenbanken zuzugreifen.[1] Ein PDO-Objekt ähnelt dem mysqli-Objekt, mit dem wir bisher gearbeitet haben. Es speichert die für den Datenbankzugriff wesentlichen Daten, nämlich Server, User, Passwort und Datenbankname. Zusätzlich stellt es eine Funktion bereit, mit der man sogenannte *Prepared Statements* absetzen kann. Was ist das?

Prepared Statements können wir uns als eine Art kompilierte Vorlage für SQL-Anweisungen vorstellen, in die variable Parameterwerte eingesetzt werden können. Prepared Statements haben zwei wichtige Vorteile:

- *Laufzeit:* Die Abfrage muss nur einmal geparst (also vorbereitet) werden, kann dann aber mehrere Male mit denselben oder anderen Parametern ausgeführt werden. Komplexe Abfragen können damit merklich beschleunigt ablaufen, insbesondere wenn dieselbe Abfrage oft mit verschiedenen Parametern wiederholt wird. Mit einem Prepared Statement vermeidet die Anwendung einen immer wieder neu startenden Zyklus von Analyse – Kompilierung – Optimierung.

- *Vermeidung von SQL-Injektion:* Die Parameter für Prepared Statements müssen nicht extra maskiert werden. Der Treiber für die vorliegende Datenbank übernimmt das automatisch. Wenn eine Anwendung ausschließlich Prepared Statements benutzt, ist damit garantiert, dass keine SQL-Injektion auftreten wird.

Wir werden auf den Aspekt der Parametrisierung bei den Prepared Statements nicht weiter

[1] https://php.net/manual/de/book.pdo.php

eingehen. Für Anwendungen mit sehr großen Datenbeständen und komplexen Zugriffen sollte dies jedoch in Betracht gezogen werden.

Um mit PDO auf Daten einer Datenbank zuzugreifen, benötigen wir im Wesentlichen die folgenden Schritte.

Aktionsschritt	**PHP-Anweisung**
1. Verbindung zur Datenbank erstellen	`$pdo = new PDO(...)`
2. SQL-Abfrage vorbereiten	`$stm = $pdo->prepare(... SQL ...)`
3. SQL-Abfrage zur Datenbank senden	`$stm->execute()`
4a. Abfrageergebnis als Objekt in PHP einlesen	`$stm->fetch(PDO::FETCH_OBJ)`
4b. Abfrageergebnis als assoziatives Array in PHP einlesen	`$stm->fetch(PDO::FETCH_ASSOC)`
4c. Abfrageergebnis als numerisches Array in PHP einlesen	`$stm->fetch(PDO::FETCH_NUM)`

Ein PDO-Objekt wird durch die Anweisung

```
$pdo = new PDO("mysql:host=$server;dbname=$database", $user, $password);
```

erzeugt, wobei `$server` den Datenbankserver bezeichnet, `$database` die Datenbank und `$user` und `$password` die Zugangsdaten. Verwendet man eine andere Datenbank als MariaDB oder MySQL, so muss der Datenbanktreiber `mysql:host` entsprechend angepasst werden.[2] Vergleichen wir die entsprechenden Schritte für die Datenbankzugriffe mit mysqli in Abschnitt 11.1, so gibt es hier durch den Vorbereitungsschritt einen Schritt mehr, aber die sonstigen Funktionalitäten sind nahezu gleich.

Zu beachten ist ferner, dass die Erzeugung eines PDO-Objekts eine Exception werfen kann, also zu einem Laufzeitfehler führen kann. Soll das PHP-Programm daher nicht einfach abbrechen, sondern geordnet weiterlaufen (z. B. mit einer Fehlermeldung), so muss die Erzeugung in eine try-catch-Anweisung eingepackt werden:

```
try {
   $pdo = new PDO(
      "mysql:host=$server;dbname=$database;charset=utf8",
      $user, $password
   );
} catch (PDOException $e) {
   exit("Error: " . $e->getMessage() . "<br/>");
}
...
```

Vorteilhaft ist es hierbei, ähnlich wie mit Listing 11.1 eine Funktion `login` zentral für die Website in eine Datei `db_login_pdo.inc.php` mit den geeigneten Zugangsdaten auszulagern:

Listing 14.1: Login mit PDO

```
function login($database) {
   try {
      $server   = "localhost";
```

[2] `https://php.net/manual/de/book.pdo.php`

```
        $user     = "user";
        $password = "passwort";
        return new PDO(
           "mysql:host=$server;dbname=$database;charset=utf8",
           $user, $password
        );
    } catch (PDOException $e) {
        exit("Error: " . $e->getMessage() . "<br/>");
    }
}
```

und sie in den PHP-Programmen entsprechend aufzurufen:

```
require_once("db_login_pdo.inc.php")
$pdo = login($database);
```

14.2 Session Fixation

Session Fixation (etwa: „Sitzungsfestlegung“) ist ein Angriff auf eine verbindungsbehaftete Datenkommunikation zwischen zwei Computern, die auf einer Session-ID basiert. Wie in Kapitel 9.2 über Sessions dargestellt, muss ein Webserver für jeden Nutzer eine individuelle Session-ID vergeben, die dieser Nutzer dann mit jedem weiteren Aufruf, gemeinsam mit den eigentlichen Nutzdaten, übermitteln muss. Technisch kann dies mit Cookies oder mit einem URL Link realisiert werden. Eine Session ist solange gültig, bis entweder der Nutzer sie per Logout schließt oder der Server sie – in der Regel nach einer festgelegten Zeitüberschreitung – abbricht. Eine Session Fixation besteht dann aus drei Schritten:[3]

1. Der Angreifer verschafft sich eine gültige Session-ID, z. B. indem er eine Anmeldung bei dem Server startet.
2. Er schiebt diese Session-ID einem anderen Benutzer unter.
3. Loggt sich nun der Benutzer mit der untergeschobenen Session-ID mit seinen Userdaten auf dem Server ein, so kann der Angreifer dort über die Session-ID mit dessen Rechten Zugriff erlangen. Denn aus Sicht des Servers handelt es sich ja um eine Session, für die sich der echte Nutzer bereits authentifiziert hat.

Der schwierigste Schritt dieses Angriffs ist die Unterschiebung der Session-ID an den arglosen Benutzer. In dem Fall, dass die Session-ID über einen Link übertragen wird, muss der Angegriffene dazu gebracht werden, diesen Link anzuklicken, beispielsweise indem der Link ohne die Session-ID angezeigt wird:

```
<a href="http://example.com/login.php?session_id=3f5z7">example.com</a>
```

Falls die Session über Cookies identifiziert wird, sind entsprechende Links über Header-Response bzw. clientseitige Funktionen möglich, siehe `https://owasp.org/www-community/attacks/Session_fixation`. Um den Inhalt und insbesondere die Parameter des URL zu verschleiern, könnte der Angreifer auf einen Kurz-URL-Dienst zurückgreifen. Der URL kann dem Opfer beispielsweise per E-Mail zugesendet als Link oder als Ziel (`action`-Wert) eines HTML-Formulars auf einer Webseite hinterlegt werden. Bei diesen Methoden des Unterschiebens ist der Angreifer aber immer darauf angewiesen, dass sein Opfer den URL öffnet bzw. ein Formular absendet.

[3] Maurice (2014):§8.2.2.

Gegenmaßnahmen

Eine wirksame Gegenmaßnahme gegen eine Session Fixation ist, dass der Server die Session-ID eines Benutzers bei einem Login direkt ändert. So müsste ein Angreifer die neue Session-ID kennen, die untergeschobene und ihm bekannte Session-ID wäre dann wertlos.

Auch eine Verbindung über HTTPS verhindert Session Fixation, denn in diesem Falle würde der Dienstanbieter feststellen, dass die Identitäten von Opfer und Angreifer unterschiedlich sind.

15
* Dateien

Kapitelübersicht

Für das Lesen und Schreiben von Dateien gibt es in PHP zahlreiche Funktionen. Hier offenbaren sich gleichzeitig die Vorteile und die Risiken eines serverseitig ablaufenden Programms. Im Gegensatz zu clientseitigen Programmen mit JavaScript kann man mit PHP lesend und schreibend auf das Dateisystem des Servers zugreifen. Das ist einerseits eine enorme Erleichterung, da sehr oft Dateien gespeichert werden müssen, um auf sie dauerhaft zugreifen zu können. So können beispielsweise Logdateien geschrieben werden, die Abläufe und Nutzerbewegungen protokollieren, oder Dateien in Webseiten aufgenommen werden, die Nutzer vorher als Textdatei oder Grafiken hochgeladen haben.

Andererseits stellen Zugriffe auf das Dateisystem des Servers eine Sicherheitslücke dar. Man kann zwar viele dieser Lücken skriptseitig abfangen, jedoch ist das Wesentliche über die Dateizugriffsrechte des Servers zu regeln.

15.1 Zugriffsrechte

In den Dateisystemen aller modernen Server-Betriebssysteme, insbesondere Unix-Systemen, wird der Zugriff auf Verzeichnisse und Dateien über ein 3-Bit-Flag gesetzt: Das erste Bit entscheidet über das Leserecht (`r` wie *read*) der Datei oder des Verzeichnisses, das zweite über das Schreibrecht (`w` für *write*, beinhaltet das Erstellen und Löschen) und das dritte über Ausführbarkeit (`x` für *executable*, das bedeutet für Verzeichnisse das Öffnungsrecht). Für einen User bedeuten damit z.B.

`r-`, bzw. `rwx`,

dass er die Datei nur lesen darf bzw. dass er alle Rechte auf sie hat.

Wichtig zu wissen ist, dass die Rechte auf das Verzeichnis, in dem sich die HTML-Dateien auf dem Webserver befinden, für die User `r-` (bzw. für Unterverzeichnisse `r-x`) lauten, sie also nur Leserechte haben. Damit können sie keine Dateien in das Verzeichnis schreiben oder löschen (was ja auch sinnvoll ist).

Um Dateien zu erstellen oder hochzuladen, benötigt das PHP-Skript Schreibrechte auf das Verzeichnis, in das die Dateien gespeichert werden sollen. Es empfiehlt sich daher, dazu ein eigenes Verzeichnis anzulegen, das Schreibrechte für den Webserver als User einrichtet. In diesem Skript ist dies das Verzeichnis namens `uploads` direkt in dem Homeverzeichnis des Webservers. Diese Einstellungen können nur von dem Systemadministrator durchgeführt werden.

Sehr empfehlenswert ist außerdem, den Speicherplatz für Uploads zu begrenzen und/oder auf bestimmte Nutzer per Passwort einzuschränken.

15.2 Lesen und Schreiben von Dateien

Der Zugriff auf eine Datei erfolgt mit Hilfe eines so genannten *Datei-Handles*. Ein solches Handle stellt die Verbindung zwischen dem Programm und dem Dateisystem dar. Es arbeitet mit verschiedenen *Dateimodi*, nämlich Lesen (`r`), Schreiben (`w`) und Anfügen (`a`, für *append*) und einem Dateizeiger, der die aktuelle Lese- und Schreibposition innerhalb der Datei markiert.

Die wichtigsten PHP-Funktionen zum Lesen und Schreiben von Dateien sind:

- `$fh = fopen( $dateiname, $modus )`
  `fopen` öffnet eine Datei und gibt ein Handle `$fh` (wie *file handle*) zurück. Falls die Datei nicht geöffnet werden kann, wird `false` zurückgegeben. Der Dateiname kann ein relativer oder ein absoluter Pfad sein, aber auch eine FTP- oder HTTP-Datenquelle. So kann auch der Inhalt einer Webseite oder eines XML-Dokuments als Datei gelesen werden. Für den Zugriffsmodus `$modus` gibt es sechs Auswahlvarianten:
  - `"r"`: Lesemodus
  - `"r+"`: Lese- und Überschreibmodus
  - `"w"`: Schreibmodus; falls die Datei noch nicht existiert, wird eine neue angelegt, andernfalls wird die bestehende Datei neu geschrieben und der alte Inhalt gelöscht.
  - `"w+"`: wie bei `"w"`, zusätzlich kann aber gelesen werden
  - `"a"`: wie bei `"w"`, jedoch wird eine bestehende Datei nicht gelöscht, sondern am Ende weiter geschrieben
  - `"a+"`: wie bei `"a"`, zusätzlich sind Leseoperationen möglich

- `fclose( $fh )`
  `fclose` schließt eine Datei. Wichtig ist, dass als Parameter nicht der Dateiname, sondern das Datei-Handle übergeben wird. Da beim Beenden eines PHP-Skripts offene Dateien automatisch geschlossen werden, ist ein explizite Schließen einer Datei zwar nicht zwingend erforderlich, aber zur Vermeidung von Zugriffskonflikten während des Skriptablaufs dringend zu empfehlen.

- `fwrite( $fh, $string, [, $laenge] )`
  `fwrite` schreibt eine Zeichenkette (hier `$string`) in eine Datei; auch hier ist nicht der Dateiname als Parameter zu übergeben, sondern das Datei-Handle. Der Parameter `$laenge` ist dabei optional und begrenzt die Anzahl der Zeichen. Die Funktion gibt die Anzahl der geschriebenen Zeichen zurück.

- `$string = fread( $fh, $laenge )`
  `fread` liest eine durch `$laenge` bestimmte Anzahl von Bytes, maximal bis zum Dateiende. Die vollständige Datei wird durch die folgende Anweisung eingelesen: `$string = fread($fh, filesize($fh));`

- `readfile( $dateiname )`
  `readfile` öffnet die Datei `$dateiname`, schreibt den kompletten Inhalt in das Browserfenster (bzw. die Standardausgabe) und schließt die Datei wieder.

- *array* `file( $dateiname )`
  öffnet die Datei `$dateiname`, liest sie zeilenweise in ein Array und schließt sie wieder. Bei einer Textdatei steht dann jeweils eine Zeile in einem Array-Element. Mit der Anweisung

  ```
  $string = implode("", file( $dateiname ));
  ```

  wird der gesamte Textinhalt in den String `$string` gespeichert. Die Wirkung dieser Anweisung entspricht also `readfile`, nur dass der Ausgabestring nicht direkt in der Standardausgabe landet.

15.3 CSV-Dateien abspeichern

Betrachten wir zunächst ein Beispielprogramm, das ein PHP-Array als CSV-Datei auf dem Webserver abspeichert.

Listing 15.1: `implode` und csv-Export

```
<!-- Export als csv-Datei Dateiname: csv.php //->
<!DOCTYPE html>
<html>
<head>
  <title>csv-Export</title>
</head>
<body>
<h2>CSV-Export eines 2-dimensionalen Arrays mit <code>implode</code></h2>
<?php
   require_once( "./adressen.inc.php" );
   $datei = "../uploads/adressen.csv";
   $fh = fopen($datei, "w+"); // erstelle/überschreibe Datei
   foreach ($adresse as $adressEintrag) {
      $adressEintrag = implode(";", $adressEintrag) . "\n";
      fwrite($fh, $adressEintrag);
   }
   fclose($fh);
   echo " $datei gespeichert: <hr/>";
   readfile( $datei);
?>
</body>
</html>
```

Zum Export eines Arrays in eine csv-Datei wird zunächst das Array erzeugt (was in der eingebundenen Datei per `require` geschieht) und sodann der Pfad der zu schreibenden Datei als

Variable `$datei` gespeichert. Das Datei-Handle `$fh` wird mit `$fopen` erzeugt, wobei eine bereits existierende Datei gelöscht würde. Dann wird in einer `foreach`-Schleife jeder Adresseintrag des Adress-Arrays per `implode` als String erzeugt und als Zeile in die Datei geschrieben, (das Steuerzeichen `\n` bewirkt einen Zeilenwechsel) bevor die Datei sauber geschlossen wird und zur Kontrolle mit einer einfachen `readfile`-Anweisung den Dateiinhalt unformatiert an das Browserfenster übergibt.

15.4 Hochladen von Dateien

Eine sehr nützliche Möglichkeit eines serverseitigen Skripts ist das Hochladen *(upload)* von Dateien auf der lokalen Festplatte des Clients auf ein Verzeichnis des Webservers. Der Dateityp kann dabei im allgemeinen beliebig sein, einige können erkannt werden und eine spezifische Verarbeitung folgen. So können Sie Bilder hochladen und ggf. weiter verarbeiten lassen, PDF-Dokumente ins Web stellen usw.

Betrachten wir zunächst den groben Ablauf eines Uploads. Insgesamt ist das ein nichttrivialer Prozess, an dem die drei Systeme Client (Browser), Webserver und Dateisystem des Servers beteiligt sind. Daher läuft der Prozess grob in vier Schritten ab:

Schritt	Aktion	beteiligtes System
1.	Auswahl der Datei	Browser
2.	Abschicken der Datei	Browser
3.	Empfangen der Datei	Webserver
4.	Abspeichern der Datei	Dateisystem des Servers

Damit wird sofort klar, dass die Aktionen der Dateiauswahl und des Verschickens der Datei nicht von einem serverseitigen PHP-Skript gesteuert werden *kann*. Gehen wir die Aktionen sukzessive durch.

15.4.1 Auswahl und Abschicken der Datei

Wie bringt man den Browser dazu, die beiden clientseitigen Aktionen *Auswahl* und *Abschicken der Datei* durchzuführen? Die Antwort ist, wie bei jeder User-Interaktion im Web, ein Formular, und zwar für ein Upload mit dem Input-Typ `file`. Das Formular für einen Upload sieht wie folgt aus.

```
<form method="post" enctype="multipart/form-data" action="/...php">
  <input type="file" name="datei"/>
  <input type="submit"/>
</form>
```

Für ein Upload müssen die Attribute `method` und `enctype` des `form`-Elements gesetzt sein, und zwar `method="post"` und `enctype="multipart/form-data"`. Der URL zu dem PHP-Skript, das die hochgeladene Datei physisch auf dem Server speichern wird, kann als Wert für `action` angegeben werden. (Lässt man es weg, wird die Formulardatei selbst wieder aufgerufen.)
Das Formular enthält zwei Buttons Durchsuchen... und Datei absenden. (Die Beschriftungen der Buttons hängen vom Browser ab, können aber auch wie in HTML üblich mit dem Attribut `value="..."` festgelegt werden.) Wird der Durchsuchen-Button geklickt, öffnet sich ein Dateifenster auf dem lokalen Rechner des Browsers, und nach Auswahl einer Datei wird deren Name angezeigt. Mit Drücken des zweiten Buttons wird die Datei zur weiteren Verarbeitung an den Server geschickt und kann dort im nächsten Schritt weiterverarbeitet werden.

Abbildung 15.1: Browserfenster bei einem Upload. Links: Startanzeige des Formulars. Rechts: Anzeige nach Auswählen der Datei über „Durchsuchen ...”

15.4.2 Empfangen und Abspeichern der Datei

Der Webserver, auf dem das aufgerufene PHP-Skript liegt, speichert die abgeschickte Datei zunächst in einem temporären Verzeichnis. Diese wird aber nach Ablauf des Skripts sofort gelöscht. Erst jetzt kommt das PHP-Skript zum Zuge, es muss nun das Dateisystem ansprechen und die Datei in das Zielverzeichnis kopieren.

Die wesentlichen Informationen über die Datei stehen dabei in dem zweidimensionalen assoziativen Array `$_FILES` zur Verfügung, einer superglobalen Standardvariablen. Es enthält pro Input-Tag des Eingabeformulars ein inneres assoziatives Array mit dem Namen des Tags als Schlüssel, in unserem Beispiel also nur ein Element mit dem Schlüssel `datei`. Der Inhalt dieses Arrays ist jeweils wiederum ein assoziatives Array, das die Informationen über die mit der Anfrage hochgeladene Datei enthält, nämlich die folgenden fünf Elemente:

Schlüssel	Inhalt
`"name"`	Dateiname (ohne Pfad)
`"type"`	Dateityp
`"tmp_name"`	absoluter Pfad zur temporären Datei
`"size"`	Größe der Datei in Byte
`"error"`	Fehlernummer (0 = OK)

Die temporäre Datei ist dabei der Name der hochgeladenen Datei dar und muss zur dauerhaften Speicherung in das gewünschte Verzeichnis verschoben werden. Um eine Datei zu verschieben, gibt es in PHP die Funktion `move_uploaded_file( $from, $to )`, die den Pfad `$from` der zu kopierenden Datei als ersten Parameter erhält und als zweiten den Pfad `$to` der kopierten Datei.

Eine Schwierigkeit ergibt sich für Dateinamen, die Leerzeichen enthalten. Ein Leerzeichen im Dateinamen bringt das Skript zum Absturz (!), muss also auf jeden Fall abgefangen werden. Eine einfache Lösung ist die Ersetzung aller Leerzeichen durch den Unterstrich _ mit Hilfe der Funktion

```
strtr($string, $str1, $str2)
```

(für *string-trim*). Sie ersetzt in dem String `$string` alle Teilstrings `$str1` durch den String `$str2`. Als letztes Problem sollte man leere Dateinamen ("") und leere Dateien (also 0 Byte Größe) verhindern.

Als minimalen Algorithmus zum Hochladen der Dateien können wir also folgende Schleife benutzen.

```
foreach ($_FILES as $datei) {
  $name = strtr( $datei["name"], " ", "_" ); // Leerzeichen ersetzen
  if ( $datei["name"] != "" && $datei[size] > 0 ) {
    $geklappt = move_uploaded_file($datei[tmp_name], "../uploads/$name" );
    if ($geklappt) {
       echo " Datei empfangen!";
    }
  }
}
```

Fügen wir das alles zu einem Skript zusammen, so erhalten wir das folgende Beispielskript.

Listing 15.2: Datei hochladen

```
<h2>Datei hochladen</h2>
<?php
$uploadPfad = "../uploads/";  // Pfad des Upload-Verzeichnisses
if (empty($_FILES)) {         // Dateien noch nicht verschickt?
?>
<form action="<?php echo $_SERVER['PHP_SELF'];?>"
      method="POST" enctype="multipart/form-data">
  Dateipfad:
  <input type="file" name="datei"/>
  <br/> <br/>
  <input type="submit" value="Datei hochladen"/>
</form>
<?php
} else {
   foreach ($_FILES as $datei) {
      echo " Datei $datei[name] wird geladen ... <br/>";
      $name = strtr($datei["name"], " ", "_");  // Leerzeichen ersetzen
      echo " ... und nach $uploadPfad$name gespeichert ... <br/>";
      if ( $datei["name"] != "" && $datei["size"] > 0 ) {
         $geklappt = move_uploaded_file(
            $datei["tmp_name"], "$uploadPfad$name"
         );
         if ($geklappt) {
            echo "Datei empfangen!";
         } else {
            echo "Der Upload hat leider nicht geklappt!";
         }
      } else {
         echo "Datei nicht ok! (zu groß?)";
      }
   }
}
?>
```

Zu beachten ist, dass die maximale Größe von mit PHP hochzuladenden Dateien in der Konfigurationsdatei `php.ini` festgelegt wird, und zwar durch die Parameter `upload_max_filesize` und `post_max_size`. Ersterer muss kleiner gleich dem zweiten sein, üblicherweise sind sie auf 2MB und 8MB eingestellt. Weitere Informationen siehe `https://php.net/manual/de/features.file-upload.post-method.php`.

15.4.3 Dateien von Verzeichnissen anzeigen

Wie können wir uns die Dateien innerhalb eines Verzeichnisses anzeigen lassen? Grundsätzlich ist dazu die Funktion `dir`[1]

```
$dir = dir("... Verzeichnispfad ...");
```

[1] `https://php.net/manual/de/function.dir.php`

vorgesehen, die ein Objekt `$dir` der Klasse `Directory` liefert und als Parameter dessen absoluten Pfad erwartet. Das Objekt `$dir` hat eine Funktion `read`, mit der der jeweils nächste Eintrag dieses Verzeichnisses (genauer: des „Verzeichnis-Handles“) zurückgegeben wird. Auf diese Weise durchläuft die Schleifenkonstruktion

```
while ($datei = $dir->read()) {
   ...
}
```

also alle Einträge des Verzeichnisses. Allerdings befinden sich darunter stets die speziellen Verzeichniseinträge . (aktuelles Verzeichnis) und .. (Elternverzeichnis des aktuellen Verzeichnisses).[2] Sollen alle Verzeichniseinträge bis auf diese behandelt werden, so können sie durch eine **`continue`**-Anweisung direkt zu Beginn der Schleife wie folgt ausgeschlossen werden:

```
while ($datei = $dir->read()) {
   if ($datei === "." || $datei === "..") continue;
   ...
}
```

(**`continue`** beendet den aktuellen Schleifendurchlauf und führt die Schleife fort.) Am Ende des Programms sollte das Directory-Objekt mit der Anweisung

```
$dir->close();
```

geschlossen werden. Betrachten wir als Anwendungsbeispiel ein Programm, das alle Dateien seines Verzeichnisses als Hyperlink in einer Liste anzeigt. Dazu benötigen wir zumächst den absoluten Pfad dieses Verzeichnisses. Dazu verwenden wir den Eintrag `$_SERVER["SCRIPT_FILENAME"]` des superglobalen Arrays `$_SERVER`, das uns den absoluten Pfad des PHP-Programms liefert. Um daraus den Pfad des Verzeichnisses zu ermitteln, müssen wir den Teilstring bis ausschließlich des letzten Zeichens `/` bestimmen, was mit den Funktionen `substr` und `strrpos` gelingt. Damit lautet das vollständige Skript wie folgt:

Listing 15.3: Verzeichnis anzeigen

```
<?php
$verzeichnis = $_SERVER["SCRIPT_FILENAME"]; // Pfad zur aktuellen PHP-Datei
// String bis zum letzten Auftreten von /":
$verzeichnis = substr($verzeichnis,0,strrpos($verzeichnis,"/"));

echo "Das Verzeichnis $verzeichnis enthält die Dateien und Verzeichnisse:";
echo "<ul>";
$dir = dir( $verzeichnis );// erzeugt ein Verzeichnisobjekt
while ( $datei = $dir->read() ) {
   if ($datei === "." || $datei === "..") continue;
   $link = "./$datei";
   echo "<li><a href=\"$link\">$datei</a></li>";
}
$dir->close();
echo "</ul>";
?>
```

Im Browserfenster wir damit eine Liste aller Dateien des Vereichnisses angezeigt, vgl. 15.2. Hier ist das Skript `verzeichnisinhalt_auflisten.php` selber ebenfalls aufgelistet.

[2] siehe z. B. `https://help.ubuntu.com/community/LinuxFilesystemTreeOverview`

Abbildung 15.2: Ergebnis des PHP-Skripts in Listing 15.3.

15.5 Zusammenfassung

- Mit PHP-Skripten kann man auf das lokale Dateisystems des Webservers zugreifen. Dafür müssen systemseitig Zugriffsrechte auf bestimmte Verzeichnisse und Dateien eingerichtet werden.
- Dateien werden i.a. durch ihren URL angesprochen, d.h. durch relative oder absolute Pfade, oder vollständige Webadressen (`http://www`... oder `ftp://`...).
- Zentraler Begriff für die Verarbeitung von Dateien ist das Datei-Handle. Es wird in PHP mit der Funktion `fopen` erzeugt. Erst dann kann eine Datei gelesen (`fread`) oder geschrieben (`fwrite`) werden.
- Mit `implode` kann ein Array als csv-Datei gespeichert werden.
- Das Hochladen von Dateien ist ein komplexes Zusammenspiel von Client und Server. Zum Verschicken einer Datei benötigt der Browser das Input-Element vom Typ "file" eines Formulars, das PHP-Skript die Daten des `$_FILES`-Arrays und die `copy`-Funktion zum kopieren der temporären hochgeladenen Datei. Zu achten ist ggf. auf eine Modifikation der Dateinamen, da insbesondere Leerzeichen zum Abbruch des PHP-Skriptes führen.
- Dateien und Verzeichnisse eines Verzeichnisses anzeigen lassen gelingt mit der Funktion `dir`, die ein Directory-Objekt erzeugt und die dafür vorgesehene Funktion `read` liefert.

Teil III

Clientseitige Programmierung mit JavaScript

16 JavaScript

Kapitelübersicht

JavaScript ist eine wichtige Programmiersprache, da es die Sprache der Browser ist. Nach C, mit dem alle wichtigen Betriebssystemkerne programmiert sind, ist JavaScript damit die

wohl am weitesten verbreitete höhere Programmiersprache der Welt. JavaScript ist, ähnlich wie PHP, eine Skriptsprache, deren Anweisungen der Interpreter direkt in den Arbeitsspeicher liest und ausführt. Üblicherweise versteht man unter dem Begriff JavaScript eine clientseitige Sprache, deren Interpreter ein Modul des Browsers ist. JavaScript ermöglicht es, durch Zugriff auf bestimmte Funktionen des Browsers, also „lokal“ auf dem Rechner des Webclients, eine HTML-Seite dynamisch und interaktiv zu gestalten. Im wesentlichen lassen sich mit JavaScript Formulareingaben lokal (also noch vor Abschicken der Daten an den Server) überprüfen, Bilder animieren, Mausereignisse verarbeiten, Pop-Up-Fenster öffnen, oder (insbesondere im Rahmen von AJAX) asynchrone HTML-Requests für einzelne HTML-Elemente durchführen.

JavaScript wurde 1995 von Brendan Eich bei Netscape Communications entwickelt. Der Name suggeriert eine Verwandtschaft mit der Programmiersprache Java, was aber bis auf gewisse Syntaxähnlichkeiten überhaupt nicht der Fall ist. Als Reaktion auf den Erfolg von JavaScript entwickelte Microsoft kurz darauf eine sehr ähnliche Sprache unter dem Namen JScript. Die europäische Industrievereinigung ECMA *(European Computer Manufacturers Association)* mit Sitz in Genf erarbeitete 1997 den Standard ECMA-Script (ECMA-262, [JS]). ECMA-262 ist seit 1998 internationaler Standard (ISO/IEC 16262). Aktuelles findet man unter [JS] insbesondere zu den verschiedenen Browserimplementierungen auf der Unterseite About_JavaScript.

Insgesamt ist JavaScript eine im Kern sehr schöne und mächtige Programmiersprache, die allerdings auch einige üble Eigenschaften hat. Zudem existieren im Web unzählige unsäglich schlechte Programmbeispiele von Programmierlaien und – ja, auch Idioten. Glauben Sie niemals blind irgendeinem Beispiel aus dem Netz, prüfen Sie stets die Programme und Ausschnitte auf Nachvollziehbarkeit! Douglas Crockford, einer der Kämpfer für die Nutzung der guten Möglichkeiten von JavaScript[1], hat die Webseite JSLint [JSL] eingerichtet, auf der JavaScript-Quelltext auf seine Qualität geprüft werden kann. In einer umfangreichen Optionsliste kann man dabei die Toleranz des Prüfprogramms einstellen. Die Quelltexte dieses Skriptes sind mit den Einstellungen in 16.1 geprüft (bis auf die Methode `document.write`).

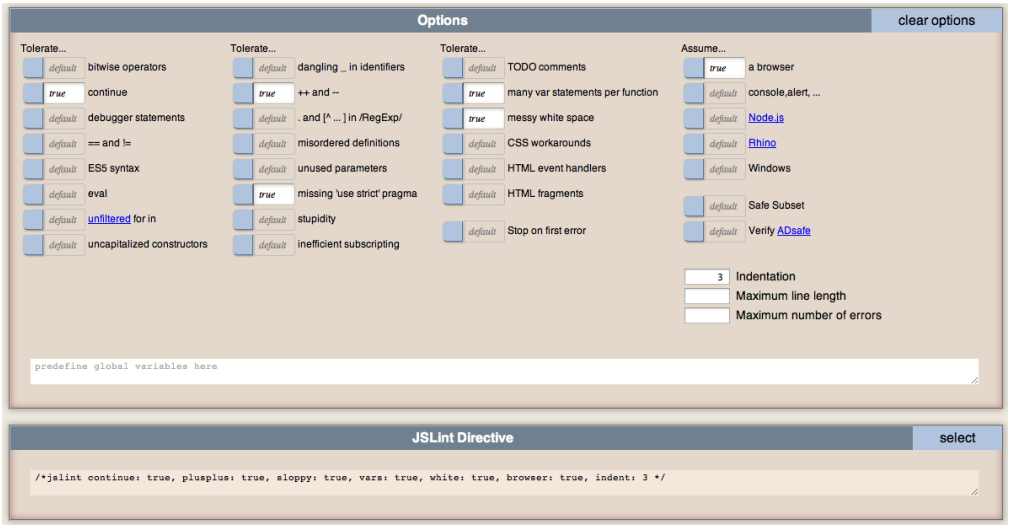


Abbildung 16.1: Optionen der Toleranz von JSLint [JL]

[1] Crockford (2008).

16.1 JavaScript als clientseitige Programmiersprache

Bevor wir in die Programmierung mit JavaScript einsteigen, sollten wir ein paar Worte zum Ort der Durchführung von JavaScript verlieren. Wir werden JavaScript hier vorwiegend als „clientseitige“ Programmiersprache des Web behandeln. Was heißt das?

Sehen wir uns die dreischichtige Struktur unserer Webanwendungen an, so erkennen wir, dass jede Schicht ihre spezifische Programmiersprache hat. Während SQL nur auf dem Datenbankserver laufen kann und PHP auf dem Webserver, wird ein JavaScript Programm auf dem Browser ausgeführt, also auf der Seite des Webclients. JavaScript ist mittlerweile die einzige

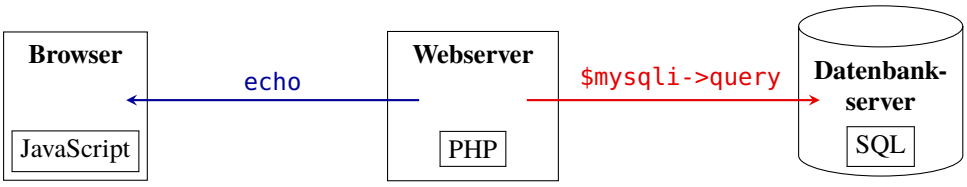


Abbildung 16.2: Ablauforte der Programmiersprachen in der dreischichtigen Architektur des Web

Programmiersprache, die im Browser ausgeführt wird. Früher gab es auch Java-Applets und Flash-Anwendungen, diese werden von den aktuellen Browsern jedoch längst nicht mehr unterstützt. Da JavaScript umgekehrt auf allen aktuellen Browsern ausführbar ist, ist sie in diesem Sinn nach C die meistverbreitete Programmiersprache überhaupt.

Während PHP und SQL ausschließlich auf dem Webserver bzw. dem Datenbankserver laufen, wird JavaScript inzwischen auch als Webserversprache eingesetzt, nämlich bei dem Projekt `Node.js` (`https://node.js`.

16.2 Einleitende Beispiele

Nach alter Tradition soll unser erstes Programm eine kleine Begrüßungsformel ausgeben. Als Vorbemerkung halten wir zunächst fest, dass ein JavaScript-Programm stets in einem HTML-Dokument eingebettet ist. Anders als bei PHP ist also der Quelltext von JavaScript in einer Textdatei mit der Endung `.html` oder `.htm`.

Listing 16.1: Hello World mit document.write

```
<!DOCTYPE html>
<html>
  <head><meta charset="UTF-8"/></head>
  <body>
    <script>
      document.write("<h2>Grüße Dich, Welt!</h2>");
    </script>
  </body>
</html>
```

Achtet man auf die korrekte Textcodierung des HTML-Dokuments, so kann man problemlos Umlaute oder Sonderzeichen ausgeben.

16.2.1 Das <script>-Tag

Gehen wir das obige Programm kurz durch. Zunächst bemerken wir, dass das Programm ein HTML-Dokument ist: Die ersten vier Zeilen sind Standard-HTML. In der fünften Zeile beginnt mit dem Tag <script> die Deklaration des JavaScript-Programms. Das HTML-Tag

```
<script> ... </script>
```

sagt dem Browser, dass nun JavaScript-Code folgt. Man kann JavaScript-Quellext auch in eine eigene Datei auslagern, was mit Hilfe des Attributs src geschieht:

```
<script src="./quelltext.js"></script>
```

Das Auslagern von Quelltext ist für kleine Programme zwar etwas aufwendiger, dafür aber bei größeren Projekten übersichtlicher. Zudem kann so einmal erstellter JavaScript-Code in anderen Webseiten verwendet werden.

Wo kann ein <script> erscheinen?

In unserem ersten Programm wurde der Quelltext im body-Tag des HTML deklariert. Muss das immer so sein? Prinzipiell kann JavaScript durch ein script-Tag an jeder Stelle des HTML-Dokuments – also im head-Element oder im body-Element – eingebunden werden, solange die HTML-Syntax nicht verletzt wird. Insbesondere kann man beliebig viele script-Tags verwenden. Der Browser interpretiert sie alle der Reihe nach durch.

Als Faustregel sollte man jedoch beachten, dass Deklarationen von Funktionen (s.u.) eher im head vorgenommen werden, das ist eine übliche und sinnvolle Konvention. JavaScript-Anweisungen, die vor oder während des Aufbaus der HTML-Seite geschehen sollen, *müssen* sogar im head deklariert werden.

Aus Performanzgründen allerdings wird für große Programme oft geraten, die <script>-Elemente am Ende des <body>-Elements zu laden, da somit das Laden des JavaScript-Quelltextes und der HTML-Elemente sich nicht gegenseitig blockieren können.[2] Sicher sollte man diesen Hinweis bei Ladezeitproblemen großer Programme auch beachten, jedoch werden in diesem Skript die <script>-Elemente bevorzugt im <head> erscheinen.

16.2.2 Anweisungen

Jede Anweisung sollte mit einem ; abgeschlossen werden. (Solange jede Anweisung in einer eigenen Zeile steht, ist dies zwar nicht notwendig, aber es kann zu Mehrdeutigkeiten und somit zu unvorhergesehenen Effekten führen.)

16.2.3 Ausgaben

Ausgabe mit document.write()

Der JavaScript-Befehl document.write übergibt direkt Text an den Browser, den dieser gemäß HTML interpretiert und im Fenster darstellt. Der Browser empfängt zum Beispiel im Programm des Listings 16.1 einfach <h2>Hello World</h2> innerhalb des body-Tags. Eine Ausgabe mit document.write sollte also stets im body-Tag an der Stelle stehen, wo auch der Text erscheinen soll.

[2] https://docs.angularjs.org/guide/bootstrap

Ausgabe in HTML-Elementen mit ID

Eine weitere Möglichkeit zur Ausgabe in JavaScript zeigt das folgende Beispiel. Es ist ein wenig komplizierter, verwendet aber bereits eine sehr bedeutsame Möglichkeit von JavaScript, auf einzelne Elemente eines HTML-Dokuments zuzugreifen, wenn sie jeweils dokumentweit eindeutige ID's erhalten haben. Dazu wird das Attribut `id` verwendet, das bereits im Zusammenhang mit CSS angesprochen wurde. Mit der Funktion `getElementById` kann dann aus JavaScript mit dieser ID auf das Element referenziert werden. Das so erhaltene Element ist ein Objekt in JavaScript, das als ein Attribut `innerHTML` besitzt, das wiederum auf den Textinhalt verweist. (Beachten Sie dabei die Groß- und Kleinschreibung!) Betrachten wir dazu das Beispielprogramm 16.3, in dem eine Ausgabe nach Anklicken eines Buttons geschieht.

Abbildung 16.3: Ausgabe per Element-ID

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8"/>
  <script>
    let ausgeben = function() {
       document.getElementById("ausgabe").innerHTML = "I say hello!";
    }
  </script>
</head>
<body>
  <p id="ausgabe">You say good-bye.</p>
  <input type="button" value="OK" onclick="ausgeben();"/>
</body>
</html>
```

Hier hat das `<p>`-Element die ID `"ausgabe"` und wird so mit der Anweisung in Zeile 7 mit einem Text beschrieben. Zu beachten ist, dass die Funktion `getElementById` zum Zeitpunkt ihrer Ausführung das Element kennen muss. Da sie in unserem Programm im `<head>`-Element deklariert ist, das `<p>`-Element in Zeile 13 also noch nicht bekannt ist, wird hier ein in JavaScript üblicher Mechanismus verwendet, der die Ausführung erst nach Anklicken des Buttons ausführt; zu diesem Zeitpunkt ist das `<p>`-Element also schon bekannt. (Diesen Mechanismus werden wir unter dem Stichwort Ereignisbehandlung bzw. *Event Handling* näher kennen lernen.): Mit dem Attribut `onclick` wird festgelegt, was bei Anklicken der Schaltfläche passieren soll. Hier wird die Funktion `ausgeben` ausgeführt, die den Inhalt `innerHTML` des `<p>`-Elements überschreibt. Die Deklaration der Funktion ab Zeile 6 mit dem Schlüsselwort **`function`** ist JavaScript typisch und wird weiter unten erklärt.

Debugging: Ausgabe auf der JavaScript-Konsole

Bei der Entwicklung von JavaScript-Programmen kann man zum Debuggen seiner Programme auch Text auf der JavaScript-Konsole ausgeben, und zwar mit dem Befehl

```
console.log("Hallo Welt!");
```

Die Konsole ist bei den Browsern Firefox oder Chrome sehr einfach einzuschalten.

16.2.4 Eingabe von Daten in JavaScript

Eine Erweiterung des Programmbeispiels 16.3 ist das folgende Programm mit der Möglichkeit, Daten in ein JavaScript-Programm einzulesen. Bei dieser Erweiterung definiert man im `<head>`-Element eine Funktion `einlesen`, die beim Klicken des OK-Schaltknopfes aufgerufen wird und den eingegebenen Text verarbeitet. Hierzu müssen also zwei HTML-Elemente mit einer ID versehen werden, so dass sie aus JavaScript angesprochen werden können, ein `<input>`-Element für die Eingabe und ein `<span>`-Element für die Ausgabe:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8"/>
  <script>
  let einlesen = function() {
    let text = document.getElementById("eingabe").value;
    document.getElementById("ausgabe").innerHTML = text;
  };
  </script>
</head>
<body>
  <input type="text" id="eingabe" value="Anna"/>
  <input type="button" value="OK" onclick="einlesen();"/>
  <p>Sie haben gerade <span id="ausgabe"></span> eingegeben.</p>
</body>
</html>
```

Um auf den eingegeben Wert des Eingabefeldes `eingabe` zuzugreifen, muss man sein `value`-Attribut ansprechen, das auf aktuell den eingegegeben Inhalt verweist.

16.3 Grundsätzliche Syntaxregeln

JavaScript ist schreibungssensitiv (case sensitive), d.h. es wird strikt unterschieden zwischen Groß- und Kleinschreibung. Das reservierte Wort `null` ist etwas völlig anderes als `Null`.

Wie in Java beginnen *Kommentare* entweder mit einem `//` und enden mit dem Zeilenende, oder sind eingeschlossen zwischen `/*` und `*/`.

Eine Anweisung wird mit einem Semikolon beendet. Zwar besitzt JavaScript einen Mechanismus, der fehlende Semikolons einfügt, d.h. man muss sie nicht notwendig selber setzen. Allerdings sollte man sich auf diesen Mechanismus nicht verlassen, er kann zu logischen Programmfehlern führen[3].

Anweisungen werden durch Umschließen von geschweiften Klammern `{...}` zu *Blöcken* zusammengefasst. Empfohlen wird in JavaScript der 1TBS-Stil,[4] bei dem die öffnende geschweifte Klammer immer *am Ende* einer Zeile steht, nie am Anfang[5].

[3] Crockford (2008):S. 109f, 127f.

[4] `http://de.wikipedia.org/wiki/Einrückungsstil` [2016-07-17]

[5] Crockford (2008):S. 103.

16.4 Dynamisch und implizite Typisierung

Eine Variable in JavaScript kann während ihrer Laufzeit verschiedene Datentypen annehmen. Programmiersprachen, die das ermöglichen, heißen *dynamisch typisiert*. Demgegenüber heißt eine Programmiersprache *statisch typisert*, wenn jede Variable einen Datentyp hat, den sie lebenslänglich behält. Konsequenterweise ist in JavaScript eine Variable stets *implizit* definiert, d.h. der Datentyp wird bei jeder Zuweisung während der Laufzeit automatisch bestimmt. Java ist demgegenüber statisch und explizit typisiert. In JavaScript kann eine Variable die folgenden 7 möglichen Datentypen annehmen oder eine Funktion oder ein Objekt sein.

- `string`: eine in Anführungszeichen `"..."` oder Apostrophs `'...'` eingerahmte Zeichenkette (mit Unicode-Zeichen), z.B. `"¡Howdy!"` oder `'Allô'`; um Unicode zu ermöglichen, sollte das einbettende HTML-Dokument die Zeichenkodierung UTF-8 im `<meta>`-Tag vorsehen werden, wie in den obigen einleitenden Beispielen.

- `number`: eine Zahl gemäß IEEE754 (`double` in Java oder C), also `42` oder `3.14159`. Spezielle Werte sind `Infinity` für den Wert Unendlich (∞) und `NaN` für „not a number“. Einen expliziten Datentyp für ganze Zahlen (wie `int` in Java) kennt JavaScript nicht. Eine Zahl in JavaScript hat eine Objektmethode[6] `toString(radix)` mit `radix` $\in \{2, 3, \ldots, 36\}$, die sie bezüglich der Basis `radix` darstellt, also beispielsweise

  ```
  document.write((71).toString(36));   // => 1z
  document.write(Math.PI.toString(7)); // => 3.066365143203613411
  ```

  Eine weitere nützliche Standardmethode einer Zahl in JavaScript ist die Methode `toFixed`, die als Parameter die Anzahl n der darzustellenden Dezimalstellen erwartet, wobei $0 \leqq n \leqq 20$.[7]

  ```
  document.write((123.4567).toFixed(3));  // => 123.457
  ```

- `boolean`: einer der beiden logischen Werte `true` oder `false`. Wichtige Boole'sche Operatoren sind `===` und `!==`, hierbei ergibt `x === y` den Wert `true` genau dann, wenn Datentyp *und* Wert der beiden Operanden `x` und `y` gleich sind, und umgekehrt liefert `x !== y` den Wert `true` genau dann, wenn Typ *oder* Wert verschieden sind.[8] Um den Unterschied zwischen `===` und `==` zu illustrieren, sei auf die Webseite

  `http://haegar.fh-swf.de/Webtechnologie/equalities-js.html`

  verwiesen. Hier erkennt man, dass `===` die erwartete Gleichheit zeigt (nur die Diagonale ist besetzt). Auch die „falsy“ Werte, also Werte `== false`, sind einigermaßen übersichtlich (`0`, `"0"`, `NULL`, leere Arrays oder Arrays mit Eintrag 0); Ähnliches gilt für „truthy“, also `== true`, nur mit 1 statt 0. (Vergleiche die Situation in PHP).

- `function`: Der Datentyp für eine Funktion.

- `object`: Der Datentyp für ein Objekt. Für den Operator `typeof` ist auch ein Array vom Typ `object`[9].

---

[6] `https://developer.mozilla.org/en/JavaScript/Reference/Global_Objects/Number/toString` [2016-07-17]

[7] `http://www.hunlock.com/blogs/The_Complete_Javascript_Number_Reference#toFixed`

[8] Die auch in JavaScript vorhandenen Operatoren `==` und `!=` sind nicht zu empfehlen, sie erzwingen bei Typenungleichheit unerwartete Ergebnisse; insbesondere ist `==` nicht transitiv (Crockford (2008):S. 117): `""==0`: `true`, `0=="0"`: `true`, `""=="0"`: `false`.

[9] Crockford (2008):S. 18, 66.

- `null`: ein spezieller Wert für das Null-Objekt;
- `undefined`: ein spezieller Wert für „undefiniert“.

Der Datentyp eines gegebenen Wertes kann mit dem Operator `typeof` ermittelt werden. Vorsicht ist dabei nur geboten, wenn man auf das Nullobjekt prüfen möchte, denn `typeof null` ergibt `object`.

In JavaScript wird mit dem Schlüsselwort **`let`** eine *Variable* deklariert.[10]

```
let x = 1;
let a = 2, b, c = "Hallo?";
document.write("a=" + a + ", b=" + b+ ", c=" + c); // a=2, b=undefined, c=Hallo?
```

Dabei können in einer `let`-Anweisung auch mehrere durch Kommas getrennte Variablen deklariert werden. Auch wenn eine Variablendeklaration in JavaScript nicht notwendig und prinzipiell immer eine einfache Wertzuweisung wie `x = 1` für eine neue Variable `x` möglich ist, wird sie aus Gründen der Softwarequalität dringend empfohlen,[11], [JSL].

16.5 Objekte

Im Unterschied zu einer *klassenbasierten* objektorientierten Sprache wie Java oder C++ gibt es in JavaScript keine Klassen, sondern nur *Prototypen*. Klassenbasierte Objektorientierung unterscheidet zwischen Klassen, die die Struktur seiner Attribute und Methoden festlegen, und den Objekten oder Instanzen, die daraus gebildet werden. Ein Objekt kann niemals die Struktur seiner eigenen Klasse ändern.

In einer prototypenbasierten Programmiersprache wie JavaScript gibt es keine Klassen, sondern *nur* Objekte. Sie sind prototypisch in dem Sinne, dass ihre Struktur von anderen Objekten übernommen und vererbt werden kann und sie ihre eigene Struktur dynamisch erweitern können. Ein Objekt in JavaScript ist eine Menge von Attribut-Wert-Paaren, ganz ähnlich wie HashMaps in Java oder assoziative Arrays in PHP. Die Attribute heißen in JavaScript auch *Properties*. JavaScript eröffnet grundsätzlich zwei Möglichkeiten, ein Objekt zu erzeugen, eine Deklaration in einer pseudoklassischen Notation[12] einerseits, und als Objektliteral andererseits. Ist man an eine klassenbasierte Programmiersprache wie Java gewöhnt, ist die pseudoklassische Notation zwar einsichtig, allerdings sind Objektliterale eleganter und werden allgemein empfohlen[13]. Bei einem *Objektliteral* werden mit einem Doppelpunkt jeweils ein Attribut-Wert-Paar bestimmt und diese verschiedenen Paare durch Kommas getrennt:

```
let gemüse = {
  "name": "Möhre",
  "details": {
    "color": "orange",
    "size": 12
  },
  "toString": function() {return this.name + ", " + this.details.color;}
}
```

[10] Häufig findet man zur Variablendeklaration auch das reservierte Wort **`var`**. Es wird aber nicht empfohlen, da es nicht flexible Gültigkeitbereiche (Scopes) ermöglicht wie **`let`** wird in `https://stackoverflow.com/questions/762011/` erläutert. JavaScript nutzt bei `var` nicht Blöcke als Geltungsbereich für Variablen, sondern Funktionen, vgl. (Crockford (2008):S. 40, 109).

[11] Crockford (2008):S. 125, 130.

[12] Crockford (2008):S. 52.

[13] Crockford (2008):§3.

```
document.write(gemüse.name + ", " + gemüse.details.color);  // => Möhre, orange
document.write("<br>" + gemüse["name"] + ", " + gemüse["details"]["color"]);
document.write("<br>" + gemüse); // => Möhre, orange
```

Diese Syntax liegt dem wichtigen Format JSON zugrunde, das mittlerweile zu einem De-Facto-Standard der Datenübertragung im Web geworden ist.[14] In den write-Funktionen erkennt man die verschiedenen Abrufmöglichkeiten der Objekteigenschaften, nämlich mit der Punktnotation `objekt.property` (wie in Java oder C++), mit der Array-Notation mit eckigen Klammern, oder über definierte Objektmethoden wie `toString`. Die `toString`-Methode braucht dabei (als einzige Methode) nicht explizit genannt zu werden.

Die zweite Möglichkeit der Objekterzeugung ist wie in Java mit dem Schlüsselwort `new`; dazu muss ein *Konstruktor* als Funktion definiert werden, die mit der Selbstreferenz `this` die Eigenschaften *(properties)* des Objekts festlegt, also dessen Attribute und Funktionen (Methoden):

```
let Angestellter = function(name, abteilung) {
   this.name = name;
   this.abteilung = abteilung;
   this.toString = function() {return name + " in " + abteilung;}
}
emil = new Angestellter("Emil", "IT");  // Objekterzeugung
document.write(emil.name + ", " + emil.abteilung); // => Emil, IT
```

Der Konstruktor übernimmt somit in gewisser Hinsicht die Rolle der Klasse in einer klassenbasierten Sprache wie Java.

Für Objekte gibt es den Operator `in` mit der Syntax `"xyz" in obj`, der `true` zurückgibt, wenn das Objekt `obj` die Eigenschaft `xyz` besitzt, ansonsten `false`:

```
"details" in emil;   // => false
"details" in gemüse; // => true
```

Der in-Operator kann auch für Arrays (s.u.) verwendet werden, im Falle eines numerischen Indizes müssen dabei allerdings die Anführungszeichen weggelassen werden.

Notiz. JavaScript ermöglicht auch das Konzept der „prototypischen Vererbung“, eine Möglichkeit zur dynamischen Veränderung und Erweiterung der Property `prototype` bestehender Objekte. Allerdings ist das Konzept aus meiner Sicht unklar und nur begrenzt brauchbar (es ermöglicht z.B. kein Überschreiben bestehender Properties), daher wird es in diesem Skript, im Gegensatz zu früheren Versionen, nicht erläutert. Auch die ansonsten ja sehr guten Quellen `https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Global_Objects/Object/prototype` oder `http://www.w3schools.com/js/js_object_prototype.asp` brachten mir nicht viel. (Gerne nehme ich das Konzept wieder auf, wenn mich jemand vom Gegenteil überzeugt!)

16.6 Mathematische Funktionen und Konstanten

Das für mathematische Funktionen und Operationen zentrale Standardobjekt in JavaScript ist `Math` [JS, §15.8]. Es besitzt als Attribute die Konstanten

```
Math.E      // Euler'sche Zahl e = ∑_{k=0}^{∞} 1/k! ≈ 2,718281828...
```

[14] Für „lupenreines“ JSON kann es als Wert allerdings keine Funktionen geben.

```
Math.LN2    // Natürlicher Logarithmus von 2: ln 2 ≈ 0,6931471 ...
Math.LN10   // Natürlicher Logarithmus von 10: ln 10 ≈ 2,302585092994 ...
Math.LOG2E  // log_2 e ≈ 1,44269504 ...
Math.LOG10E // log_10 e ≈ 0,4342944819 ...
Math.PI     // Kreiszahl π = Σ_{k=0}^{∞} 1/16^k (4/(8k+1) − 2/(8k+4) − 1/(8k+5) − 1/(8k+6)) ≈ 3,14159 ...
Math.SQRT2  // √2 ≈ 1,41421356237 ...
Math.SQRT1_2// 1/√2 ≈ 0,707106781 ...
```

und als Methoden die mathematischen Funktionen

```
Math.abs(x)     // |x|
Math.acos(x)    // arccos x
Math.asin(x)    // arcsin x
Math.atan(x)    // arctan x
Math.atan2(x,y) // arctan(x/y)
Math.ceil(x)    // ⌈x⌉
Math.cos(x)     // cos x
Math.exp(x)     // e^x (e hoch x)
Math.floor(x)   // ⌊x⌋
Math.log(x)     // ln x (natürlicher Logarithmus)
Math.max(x,y)   // Maximum von x und y; kann auch mehrere Argumente bekommen
Math.min(x,y)   // Minimum von x und y; kann auch mehrere Argumente bekommen
Math.pow(x,y)   // x^y (x hoch y)
Math.random()   // Pseudozufallszahl z mit 0 ≤ z < 1
Math.round(x)   // kaufmännische Rundung von x auf die nächste ganze Zahl
Math.sin(x)     // sin x
Math.sqrt(x)    // √x
Math.tan(x)     // tan x
```

16.7 Kontrollstrukturen

16.7.1 Die `if`-Anweisung

JavaScript hat dieselbe bedingten Verzweigung wie Java:

```
if(ausdruck) { code } [else { code_2 }]
```

Hierbei bezeichnet *ausdruck* einen Boole'schen Wert und *code* eine Anweisung oder einen Block von Anweisungen, der `else`-Zweig ist optional (und daher in eckigen Klammern). *code* wird genau dann ausgeführt, wenn der Ausdruck nicht die Werte `true` annimmt, ansonsten wird $code_2$ ausgeführt, oder eben nichts, wenn der else-Zweig nicht existiert.

Fallunterscheidungen mit switch-case

JavaScript verwendet zur Unterscheidung mehrerer optionaler Fälle den Befehl `switch`. Die Syntax und Funktionsweise ähnelt der `switch`-Anweisung in Java.

```
compute = (operator, x, y) => {
  let z;
  switch (operator) {
```

```
    case "+": z = x + y;   break;
    case "-": z = x - y;   break;
    case "*": z = x * y;   break;
    case "/" | ":": x / y; break;
    case "%": z = x % y;   break;
    default: z = NaN;
  }
  return z;
};
document.write(compute("+", 4, 3));   // => 7
document.write("<br/>" + compute("%", 4, Math.PI));   // => 0.8584073464102069
```

16.7.2 Schleifen

Schleifen gehören als iterative Kontrollstrukturen in die Welt der imperativen Sprachen, rein funktionale Sprachen ermöglichen die Wiederholungen ausschließlich über Rekursionsaufrufe. JavaScript ist jedoch nicht rein funktional, es hat die vier Schleifenkonstrukte **`while`**, **`do/while`**, **`for`** und **`for-of`** wie in Java. Die Syntax ist bei den ersten drei Schleifenkonstrukten identisch zu der in Java. Bei der `for`-Schleife ist zu beachten, dass die Zählvariable mit `let` deklariert wird, so wie im folgenden Beispiel, dass die Euler'sche Zahl iterativ berechnet:

```
let n = 5, y = 0;
for (let k = 0; k <= n; k++) {
  y += 1/factorial(k);
  document.write("<br/>" + k + ". Iteration für e: " + y);
}
```

Hier bezeichnet `factorial` die Fakultätsfunktion auf S. 168. Die Ausgaben lauten dabei sukzessive 2, 2.5, 2.66666. . . , 2.70833. . . , 2.71666. . . .

Mit der `for-of`-Schleife kann man auf die Einträge einer Liste zugreifen, ähnlich wie die `foreach`-Schleife in PHP. Hierbei kann eine Liste ein Array sein oder ein JSON-Objekt. Die Syntax der Schleife lautet:

```
for(let key of liste) {
   ... liste[key] ...;
}
```

Hierbei sind `for`, `let` und `of` reservierte Wörter, während *key* die durch die Schlüssel der Liste *liste* laufende Variable bezeichnet. Für ein Array ist *key* einfach der Index der Einträge. Auf beide Werte von *key* und *liste* kann dann in der Schleife zugegriffen werden. Mit der `for-in` Schleife mit `in` statt `of` kann man durch die Attribute (Properties) eines Objekts laufen. Das Beispielprogramm

```
let gemüse = {
   name: "Möhre",
   for: "Max",  // funktioniert nicht im IE 8! (aber in IE 9)
   details: {
      color: "orange",
      size: 12
   },
   toString: function() {return this.name + ", " + this.details.color;}
}
```

```
for(let schlüssel in gemüse) {
   document.write("<br/> gemüse[" + schlüssel + "] = " + gemüse[schlüssel]);
}
```

ergibt die Ausgabe

```
gemüse[name] = Möhre
gemüse[for] = Max
gemüse[details] = [object Object]
gemüse[toString] = function () {return this.name + ", " + this.details.color;}
```

Man sieht, dass Werte von Attributen mit einfachem Datentyp angezeigt werden, Objekte als Attribut jedoch nicht. Bei einer Funktion wird der Queltext angezeigt, d.h. im Programmablauf kann man auf den definierenden Quelltext von Objektfunktionen lesend zugreifen.

16.8 Wichtige Standardobjekte

Es gibt einige vorgegebene Objekte in JavaScript, so die drei Objekte `String`, `Number`, `Boolean` zu den entsprechenden Datentypen, die Objekt `Math`, `Date` und `RegExp`, das `Function`-Objekt und einige Error-Objekte, sowie die Datenstrukturen `Array` und `JSON` [JS, §15].

16.8.1 Arrays

Die einzige Datenstruktur oder *Kollektion* in JavaScript ist das Array. Ein Array ist in JavaScript ein Objekt mit dem Konstruktor `Array` und kann daher mit dem new-Operator erzeugt werden. Empfohlen ist jedoch stattdessen die Erzeugung mit `[]` (dem *Literalausdruck*):

```
let a = [];
a[0] = 10;
a[1] = 20;
a[5] = "Unterschied"; // Array mit unterschiedlichen Datentypen
document.write("Länge von a: " + a.length + "<br/>");   // => 6
document.write("a = "+a+"<br/>");    // => 10,20,,,,Unterschied
```

Arrays sind, wie in Skriptsprachen allgemein, dynamisch in ihrer Größe. Daher muss die Größe nicht bei der Erzeugung angegeben werden wie beispielsweise in Java. Standardmäßig sind Arrays in JavaScript numerisch. Der Array-Index wird von 0 an hochgezählt. Die Elemente eines Array müssen nicht vom gleichen Typ sein. Das Attribut `length` gibt die Länge des Arrays zurück, die sich bei numerischen Arrays aus dem größten belegten Indexwert plus Eins ergibt. Ferner muss man das Array nicht sukzessive auffüllen, sondern kann durchaus nach Index 1 den Index 3 belegen; der Eintrag mit Index 2 bleibt dann entsprechend undefiniert.

Ein Array kann ohne `new` und den Konstruktor mit einer einzigen Anweisung erzeugt werden, indem die Einträge in eckigen Klammern eingeschlossen und durch Kommas getrennt dem Arraynamen zugewiesen werden:

```
let b = [5,7,12];
document.write("Länge von b: " + b.length + "<br/>"); // => 3
document.write("b = " + b + "<br/>");              // => 5,7,12
```

Damit ist das Array automatisch indiziert. (Eine Einzelbelegung der Arrayplätze ohne einen Index mit leeren eckigen Klammern, wie in PHP, funktioniert in JavaScript jedoch nicht.)

Assoziative Arrays. In JavaScript sind auch assoziative Arrays möglich. Als Schlüssel *(keys)* können dabei Strings verwendet werden, aber auch beliebige Objekte wie ein Array.

```
let tel = new Array();
tel["Meier"] = "9330-723";   // Arrays können assoziativ sein
tel["Müller"] = "0815/4711";
tel[a] = 42;  // a ist das Array von oben!
document.write("<br/>Länge: " + tel.length + "<br/>"); // lenght=0!
for(let key in tel) {
   document.write("tel[" + key + "] = " + tel[key] + "<br/>");
} // tel[Meier] = 9330-723, tel[Müller] = 0815/4711, tel[10,20„„Unterschied] = 42
```

Allerdings kann man bei einem assoziativen Array nicht mehr das Attribut `length` verwenden, es hat stets den Wert 0. Im Grunde sind Objekte in JavaScript nichts anderes als assoziative Arrays, bis auf die Tatsache, dass für ein Objekt im allgemeinen das Attribut `length` undefiniert ist, während es für ein assoziatives den Wert 0 hat.

Eine wichtige Funktion eines Arrays ist `forEach`,[15] die als Argument eine Funktion `callback` erhält, die sie auf jedes Array-Element anwendet. `callback` muss dabei drei Argumente `(value, index, array)` vorsehen, wobei `value` der beim Durchlaufen aktuell bearbeitete Elementwert ist, `index` der entsprechende Array-Index, und `array` das aktuelle Array.

```
let callback = function(value, index, array) {
   document.write("a[" + index + "] = " + value + ", ");
}
[2, 3, 5].forEach(callback); // => a[0] = 2, a[1] = 3, a[2] = 5,
```

Falls der dritte Parameter `array` nicht benötigt wird (wie in diesem Beispiel), kann er auch einfach weggelassen werden.

16.8.2 JSON

Die Abkürzung JSON steht für *JavaScript Object Notation* und bezeichnet zunächst ein weitverbreitetes Format zum Datenaustausch, also insbesondere eine allgemeine Syntax zur Serialisierung (Speicherung) und Übermittlung von Werten oder Objekten der sieben Datentypen auf S. 156, bis auf die Funktionen und `undefined`. Dabei ist zu beachten, dass in JSON nur Objekte erlaubt sind, deren Attribute Strings sind; es muss also heißen `"name"="Müller"` statt `name="Müller"`. In JavaScript ist JSON ein spezielles globales Objekt, das die beiden entgegengesetzt wirkenden („komplementären“) Methoden `parse` und `stringify` bereitstellt. Mit dem Befehl `JSON.parse("...")` wird der JSON-String in ein Objekt oder einen Wert in JavaScript umgewandelt. Ist der String nicht im JSON-Format, so wird das Programm mit einem Syntaxfehler beendet. Mit `JSON.stringify(...)` wird ein Objekt oder ein Wert in einen String in JSON-Format umgewandelt.

```
let gemüse = {
   name: "Möhre",
   details: {
      color: "orange",
      size: 12
   },
   toString: function() {return this.name + ", " + this.details.color;}
}
```

[15] `https://developer.mozilla.org/en-US/docs/JavaScript/Reference/Global_Objects/Array/forEach`

```
document.write("<br>" + gemüse);  // Möhre, orange
//let obj = JSON.parse(gemüse); // => Uncaught SyntaxError: Unexpected token M
let jsn = JSON.stringify(gemüse);
document.write("<br>" + jsn); // "name":"Möhre","details":"color":örange",ßize":12
let obj = JSON.parse(jsn);
document.write("<br>" + obj.name); // Möhre
let a = [2,3,5,7,11,13,17]; // Array von Primzahlen
jsn = JSON.stringify(a)
document.write("<br>" + jsn); // [2,3,5,7,11,13,17]
let b = JSON.parse("[2,3,5,7,11,13,17]");
document.write("<br>" + b); // 2,3,5,7,11,13,17
let c = JSON.parse("2357111317");
document.write("<br>" + c); // 2357111317
```

Man erkennt an dem Quelltextbeispiel, dass `stringify` alle Attribute in Strings verwandelt und Funktionen (wie hier die `toString`-Methode) einfach ignoriert. Weitere Details, insbesondere zur Syntax von JSON, findet man im MDN.[16]

16.8.3 Ereignisbehandlung (Events und Event-Handler)

JavaScript ermöglicht die Ausführung von Anweisungen, die durch Eingabeereignisse, sogenannte Events, ausgelöst werden und in speziellen Methoden, den zuständigen *Event-Handlern*, implementiert sind. Events und Event-Handler stellen auf diese Weise ein wesentliches Bindeglied zwischen dem HTML-Dokument und JavaScript-Funktionen dar. Grundsätzlich funkioniert die Ereignisbehandlung dabei wie folgt. Jede Anwenderaktion, also eine Mausbewegung, ein Mausklick, ein Tastendruck, löst ein *Ereignis* oder *Event* aus. Ist für das Element, in dem das Ereignis erzeugt wird, also beispielsweise eine Schaltfläche oder ein Textfeld, ein Ereignisbehandler registriert, so ruft dieser diejenige, auf die er referenziert, und führt sie aus.

Das Grundkonzept der Ereignisbehandlung in JavaScript ist klar und einfach: Für ein HTML-Element kann ein Event-Handler als ein Attribut festgelegt werden, das auf die JavaScript-Funktion oder die Anweisungfolge verweist, die bei Auslösen des entsprechenden Ereignisses auszuführen ist. Die Attribute der Ereignisbehandler sind daran zu erkennen, dass sie mit `on...` beginnen. Beispielsweise ist der Ereignisbehandler für einen Mausklick `onclick` oder für das Loslassen einer Taste `onkeyup`. Im allgemeinen kann nicht jeder Ereignisbehandler in jedem HTML-Element registriert werden, eine Auflistung der Ereignisbehandler und der HTML-Elemente, für die sie definiert sind, findet man auf selfhtml.org[17], eine umfangreiche Liste von Ereignissen in der Mozilla Event Reference[18]. Das folgende Beispielprogramm zeigt einige Ereignisbehandler für Maus- und Touchscreen-Ereignisse:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8"/>
  <script>
let machma = (event) => {
  alert(event + " ausgelöst!");
  console.log(event + " ausgelöst!");
```

[16] `https://developer.mozilla.org/en-US/docs/JavaScript/Reference/Global_Objects/JSON`; weitere Infos zu JSON: `https://developer.mozilla.org/en-US/docs/JSON`

[17] `http://de.selfhtml.org/javascript/sprache/eventhandler.htm`

[18] `https://developer.mozilla.org/en-US/docs/Mozilla_event_reference`

```
};
  </script>
</head>
<body>
  <p onclick="machma('Klick');">
    Wer hier klickt, kann was erleben!
  </p>
  <p ondblclick="machma('Doppelklick');">
    Wer hier doppelklickt, auch!
  </p>
  <p onwheel="machma('Mausrad');">
    Hier müssen Sie das Mausrad drehen!
  </p>
  <p ontouchmove="machma('Touchmove');">
    Hier müssen Sie wischen!
  </p>
</body>
</html>
```

Das Programm ist für Ihr Endgerät abrufbar über `http://haegar.fh-swf.de/JavaScript/event-handler.html`.

16.8.4 Date

Das Objekt `Date` regelt in JavaScript alle Berechnungen mit Datum und Zeit. Grundlage ist die Unix-Zeit in Millisekunden mit dem Nullpunkt 1.1.1970, 0:00 Uhr UTC (Universal Coordinated Time). Es gibt mehrere Konstruktoren für ein Date-Objekt

```
let today = new Date();  // liefert das aktuelle Datum und Uhrzeit
let birthday = new Date("December 17, 1985 03:24:00");
let geburtstag = new Date(1985,11,17);
let anniversaire = new Date(1985,11,17,3,24,0);
```

Daneben gibt es noch einen Konstruktor, der als Argument die Unixzeit in Millisekunden erwartet. Zwar gibt es eine Reihe von Methoden für das Objekt, insbesondere zur Ausgabe von Datum und Uhrzeit, jedoch sind die Ausgabeformate nur wenig spezifiziert und daher in ihrer Darstellung browserabhängig.[19] Mit den folgenden Methoden kommt man in der Regel jedoch hin:

```
let tag = ["Sonntag","Montag","Dienstag","Mittwoch","Donnerstag","Freitag","Samstag"];
let jetzt = new Date();
document.write("Heute ist " + tag[jetzt.getDay()] + ", der ");
document.write(jetzt.getDate() + "." + (jetzt.getMonth()+1) + "." + jetzt.getFullYear());
let minuten = Math.floor(jetzt.getMinutes() / 10) + "" + jetzt.getMinutes() % 10;
document.write(", " + jetzt.getHours() + "." + minuten + " Uhr");
document.write(", oder: " + jetzt.toISOString());
```

Eine Liste der möglichen Methoden für ein Date-Objekt findet man beispielsweise auf selfhtml.de.[20]

[19] Eine der Methoden, `getYear()`, ist sogar deutlich schlecht programmiert, sie gibt die Differenz der Jahreszahl mit 1900 zurück!

[20] `http://de.selfhtml.org/javascript/objekte/date.htm` [2016-07-17]

16.9 Funktionen und Callbacks

JavaScript ermöglicht funktionale Programmierung. In einer *funktionalen Programmiersprache* können Funktionen definiert werden und es gibt keinen Unterschied zwischen Daten und Funktionen, Funktionen können also auch in Variablen gespeichert und als Parameter übergeben werden[21]. Die funktionale Programmierung steht damit im Gegensatz zur *imperativen Programmierung*, denn die Trennung von Daten und Programmlogik wird aufgehoben und jede Variable kann nur einmal einen Wert bekommen (`final` in Java). Rein funktionale Sprachen haben sogar gar keine Variablen oder Schleifen, denn ein Algorithmus wird hier durch eine Abfolge von Funktionsaufrufen ausgeführt; für Wiederholungen werden also keine Schleifen verwandt, sondern ausschließlich Rekursionen. So werden mathematisch unsinnige Ausdrücke der imperativen Programmierung wie `x=x+1` vermieden. Andererseits widerspricht die funktionale Programmierung dem objektorientierten Paradigma, gemäß dem Objekte im Laufe ihres Lebenszyklus verschiedene Zustände annehmen können, die Attribute also Veränderliche sein müssen. JavaScript ist ein Kompromiss aus beiden Ansätzen, indem es beide Programmierparadigmen ermöglicht.

Funktionen gehören zu einem der elementaren Datentypen von JavaScript und bilden die modularen Einheiten der Sprache[22]. Eine Funktion wird definiert durch das Schlüsselwort `function`, ihren Namen, eine von runden Klammern eingeschlossene Liste von durch Kommas getrennten Parametern, und den durch geschweifte Klammern umschlossenenen Funktionsrumpf, der die Anweisungen der Funktion enthält. Der Rückgabewert der Funktion wird mit dem Schlüsselwort `return` ausgegeben:

```
function f(x,y) {
  return x*x - y*y;
}
document.write("f(5,3) = " + f(5,3)); // => 16
```

Als funktionale Programmiersprache ermöglicht JavaScript die Definition anonymer Funktionen genannt:

```
let g = function(x,y) {return x*x + y*y}
document.write("<br>g(5,3) = " + g(5,3)); // => 34
```

Damit kann eine Funktion in einer Variablen gespeichert werden; oder genauer gesagt als Referenz auf eine anonyme Funktion. Solche Funktionsreferenzen werden *Rückruffunktionen* oder *Callbacks* genannt, wenn sie in andere Funktionen als Argument eingesetzt und von diesen aufgerufen werden.

Etwas kürzer kann eine Funktion auch als „Lambda-Ausdruck“ oder „Pfeilfunktion“ wie $(x, y) \mapsto x^2 + y^2$ in der Mathematik geschrieben werden:

```
let f = (x,y) => x**2 + y**2;
```

(Lies: „(x, y) wird abgebildet auf x^2+y^2“.) Sie kann auch die geschweiften Klammern enthalten, muss aber dann auch ein `return` haben:

```
let f = (x,y) => {return x**2 + y**2};
```

(Falls eine Funktion aus mehreren Anweisungen besteht, braucht sie natürlich geschweifte Klammern.) Im Unterschied zu den „normalen“ Funktionen in JavaScript hat eine Pfeilfunktion kein eigenes `this`. Wird es als Referenz gebraucht, so wie in manchen populären Bibliotheken wie

[21] Piepmeyer (2010):S. 6.
[22] Crockford (2008):S. 29.

jQuery, so darf man nicht mit Pfeilfunktionen arbeiten. Für die meisten anderen Fälle aber sind sie äquivalent zu Funktionen.[23]

16.9.1 Rückruffunktionen (Callbacks)

Rückruffunktionen sind ein wichtiges Programmierkonzept der funktionalen Programmierung. Die Programmierung mit Rückruffunktionen folgt dem Entwurfsmuster der Kontrollflussumkehr *(Inversion of Control, IoC)*, denn in ein bereits vorher definiertes Programmmodul (die aufrufende Funktion) bzw. eine gegebene Infrastruktur (z.B. die API) kann „nachträglich“ auszuführender Quelltext definiert werden. Viele JavaScript-Bibliotheken und Frameworks basieren auf diesem Prinzip. Als ein einfaches Beispiel einer Rückruffunktion soll das folgende kurze Programm dienen:

```
let callback = (x) => {return x*x;};
let quadrate = [1, 2 , 3, 4, 5].map(callback);
document.write(quadrate); // => 1,4,9,16,25
```

Hier wird die Funktion `callback`$(x) = x^2$ von der API-Funktion `map` eines Arrays aufgerufen und ausgeführt. Ihr „Rückruf“ ergibt am Ende ein Array, das in jedem Eintrag den Quadratwert des

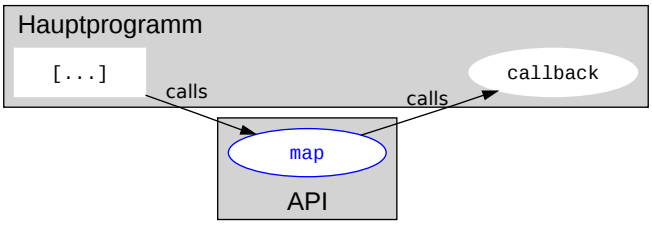


Abbildung 16.4: Aufrufstruktur einer Rückruffunktion. (Modifiziert nach `https://en.wikipedia.org/wiki/File:Callback-notitle.svg`)

originalen Arrays enthält. Die Aufrufstruktur ist in Abbildung 16.4 skizziert.

Es gibt zwei Arten von Rückruffunktionen. Eine Rückruffunktion heißt *blockierend*, oder auch *synchron*, wenn sie mit ihrer Referenz aufgerufen wird und ihr Ausführungsergebnis abgewartet wird, um weiterverarbeitet zu werden. Das obige Programmierbeispiel ist eine solche synchrone Rückruffunktion.

Bei einer *verzögerten (deferred)* oder *asynchronen Rückruffunktion* wird die Rückruffunktion in einem anderen Prozess oder Thread ausgeführt, das Hauptprogramm wartet aber nicht auf das Ergebnis der Rückruffunktion. Stattdessen wird das Ergebnis der Rückruffunktion „möglichst schnell“ verarbeitet, wenn es eingetroffen ist. Typische Anwendungsfälle solcher verzögerten Rückruffunktionen sind Ereignisbehandler, die nach einem gefeuerten Ereignis die Rückruffunktion ausführen, oder Aufrufe serverseitiger Skripte im Web.

In der nebenläufigen Programmierung spricht man bei asynchronen Rückruffunktionen auch oft von *Promises*.[24]

16.9.2 Closures

Im Gegensatz zu vielen Programmiersprachen wie Java können in JavaScript Funktionen auch innerhalb von Funktionen definiert werden. Eine solche innere Funktion hat wie jede Funktion Zugriff auf ihre eigenen Variablen, aber auch auf die Variablen der Funktion, in der sie

[23] Für nähere Details siehe `https://developer.mozilla.org/de/docs/Web/JavaScript/Reference/Functions/Arrow_functions`

[24] `https://docs.angularjs.org/api/ng/service/$q#the-promise-api`

definiert ist. Eine innere Funktion heißt *Closure*.[25] Mit Closures kann man Funktionenscharen oder „Funktionenfabriken“ programmieren, also Funktionen, die parameterabhängig Funktionen zurückgeben:

```
let fabrik = (name) => {
  return (instrument) => {
    return name + " spielt " + instrument;
  }
};
let a = fabrik("Anna");
let b = fabrik("Bert");

document.write(a("Geige") + ", " + b("Bass")); // Anna spielt Geige, Bert spielt Bass
```

Die äußere Funktion definiert hier eine Variable `name`, auf die die innere anonyme Funktion auch Zugriff hat. Bei jedem Aufruf der äußeren Funktion wird der jeweilige Wert der Variablen in der inneren Funktion „eingefroren“ und mit ihr zurückgegeben. Daher gibt die Funktion `anna()` einen anderen Wert zurück als die Funktion `bert()`.

Wofür braucht man Closures? Ein wichtiger Anwendungsfall für Closures ist das Verbergen von Hilfs- oder Arbeitsfunktionen, die nach außen vor dem Anwender gekapselt werden sollen. (In Java bis Version 7 war das nur mit `private` Objektmethoden möglich.) Als Beispiel sei die folgende effiziente Berechnung der Fibonacci-Zahlen $f_0 = 0$, $f_1 = 1$, $f_2 = 1$, $f_3 = 2$, ..., $f_n = f_{n-1} + f_{n-2}$ durch eine Rekursion innerhalb der aufzurufenden Funktion angegeben:

```
let f = (n) => {
   let c = function (f1, f2, i) {      // innere "Arbeitsfunktion" (Closure)
      if (i == n) return f2;
      return c(f2, f1+f2, i+1);
   }
   if (n.toFixed(0) != n) return NaN;  // nur für ganze Zahlen definiert!
   if (n <= 0) return 0;
   return c(0,1,1);
}
```

Ruft man nun die Funktion f mit einer natürlichen Zahl n auf, so wird die innere Arbeitsfunktion `c` in Zeile 8 mit den Startwerten $f_0 = 0$ unf $f_1 = 1$ und dem Zähler $i = 0$ aufgerufen, die sich wiederum in Zeile 6 selbst aufruft, falls keine der Abbruchkriterien erfüllt ist. Bei jedem Aufruf enthält der zweite Parameter von `c` die Summe der letzten beiden Fibonacci-Zahlen, die

Zudem finden Closures Einsatz bei dem Funktionalkalkül, also der Algebra mit Funktionen. Man kann beispielsweise die Summe $f + g$ zweier Funktionen $f, g : \mathbb{R} \to \mathbb{R}$ definieren, indem man $(f + g)(x) = f(x) + g(x)$ definiert. In JavaScript könnte der Additionsoperator für zwei Funktionen beispielsweise so aussehen:

```
let add = (f, g) => {
   return (x) => {return f(x) + g(x);};
};

let f = (x) => {return x*x - 1;};
let g = (x) => {return x*x + 1;};
let x = 2;
document.write(add(f,g)(x) + ", " + max(f,g)(x)); // => 8, 5
```

[25] `http://jibbering.com/faq/notes/closures/#clClose` [2016-07-17]

16.9.3 IIFE

Ein von JavaScript-Entwicklern oft verwendetes Programmiermuster ist ein „sofort aufgerufener Funktionsausdruck“ (*immediately invoked function expression, IIFE*) meist mit Closures. Hierbei wird eine anonyme Funktion deklariert und sofort aufgerufen[26]:

Listing 16.2: Ein einfaches IIFE

```
(function() {
   let tuwas = () => {
      document.write("a="+ a);
   }

   let a = 1;
   tuwas();
})();  // => "a=1"
```

Dieser Ausdruck tut zunächst nichts Besonderes, die Funktion `tuwas` wird ausgeführt und verarbeitet die Variable `a`. Funktionsdeklaration und Anweisungen sind in eine anonyme Funktion (Zeile 1) gekapselt, die sofort ausgeführt wird (letzte Zeile), also schematisch dargestellt:

```
(function() {...}) ();
```

Die Funktion `tuwas` ist hier also eine Closure, die auf die Variable `a` zugreifen kann, auch wenn die äußere anonyme Funktion bereits abgelaufen ist. Das Programm würde aber dasselbe ausgeben, wenn man die erste und letzte Zeile einfach wegließe. Nur wären dann `tuwas` und die Variable `a` global definiert, und genau das ist das Problem: Globale Variablen sollten in JavaScript in komplexen Systemen möglichst vermieden werden. Durch IIFEs lässt sich deren Anzahl deutlich reduzieren. Denn in Listing 16.2 beispielsweise sind die Variable `a` und die Funktion `tuwas` außerhalb der anonymen Funktion unbekannt.[27]

16.9.4 Rekursionen

In rein funktionalen Sprachen können Wiederholungen nicht durch Schleifen, sondern nur durch Rekursionen durchgeführt werden, also durch Selbstaufrufe von Funktionen. Als ein einfaches Beispiel sei die Berechnung der Fakultät angegeben:

```
let factorial = (n) => {
   if (n < 0) return NaN;
   if (n === 0) return 1;
   return n * factorial(n-1);
}
document.write("<br>factorial(5) = " + factorial(5)); // => 120
```

Hierbei ist `NaN` in JavaScript ein globales Objekt und wird in der Regel als Ergebnis von im Reellen nicht erlaubten mathematischen Operationen wie Division durch 0 oder $\sqrt{-1}$ zurückgegeben.[28]

In JavaScript wird häufig eine Rekursion mit einer definierten Verzögerung aufgerufen. Dazu muss die Funktion `setTimeout` verwendet werden,[29] deren erstes Argument die aufzurufende Funktion und deren zweites die Verzögerungszeit in Millisekunden ist:

[26] Wenz (2015).

[27] vgl. `https://developer.mozilla.org/en-US/docs/Glossary/IIFE`.

[28] `https://developer.mozilla.org/en-US/docs/JavaScript/Reference/Global_Objects/NaN`

[29] `http://developer.mozilla.org/en/DOM/window.setTimeout`

```
let rekurs = (n) => {
  if (n <= 0) return;
  setTimeout("rekurs("+(n-1)+")",1000);
  console.log("Aufruf rekurs("+n+")");
}
rekurs(5);
```

Um eine Funktion mit Argumenten aufzurufen, muss sie als String mit dem gewünschten Argumentwert aufbereitet werden.[30] Man erkennt beim Ausführen des Skripts insbesondere in der Konsole des Firefox, in der die Uhrzeiten der Ausgaben protokolliert werden, dass `setTimeout` zwar den Aufruf der angegebenen Funktion verzögert, den weiteren Ablauf der *aufrufenden* Funktion jedoch nicht aufhält: Die Ausgabe erscheint stets direkt nach Aufruf der Funktion, aber eine Sekunde vor dem Rekursionsaufruf der nächsten Funktion.

16.9.5 Funktionen höherer Ordnung

Eine *Funktion höherer Ordnung* ist eine Funktion, die als Argument eine Funktion erwartet oder deren Ergebnis eine Funktion ist. Beispielsweise kann die Kepler'sche Fassregel (oder auch Simpson-Regel)

$$I_f(a,b) \approx \frac{b-a}{6}\left(f(a) + 4f\left(\tfrac{a+b}{2}\right) + f(b)\right) \tag{16.1}$$

als Näherungsformel für das Integral $I_f(a,b) = \int_a^b f(x)\mathrm{d}\,x$ einer integrierbaren Funktion $f : [a,b] \to \mathbb{R}$ mit $a, b \in \mathbb{R}$ in JavaScript wie folgt implementiert werden:

```
let I = function (f, a, b) {
   return (b-a)/6 * (f(a) + 4*f((a+b)/2) + f(b));
}

let f = function(x) {return 3*x*x;}
let g = function(x) {return 1/x;}
let h = function(x) {return Math.sqrt(1 - x*x);}

document.write("I(f,0,1) = " + I(f, 0, 1)); // => 1
document.write("<br/>I(g,1,e) = " + I(g, 1, Math.E)); // => 1.0078899408946134
document.write("<br/>I(h,0,1) = " + I(h, 0, 1)); // => 0.7440169358562924
```

Die tatsächlichen Werte sind

$$\int_0^1 3x^2\mathrm{d}\,x = x^3\Big|_0^1 = 1, \qquad \int_1^e \frac{\mathrm{d}\,x}{x} = \ln x\Big|_1^e = 1 \tag{16.2}$$

und[31]

$$\int_0^1 \sqrt{1-x^2}\,\mathrm{d}\,x = \frac{\arcsin x + x\sqrt{1-x^2}}{2}\Big|_0^1 = \frac{\pi}{4} \approx 0{,}78539816339745. \tag{16.3}$$

(Für die Herleitung dieser Gleichung mit Integration durch Substitution siehe z.B.[32]; der Graph der Funktion $\sqrt{1-x^2}$ beschreibt in dem Intervall $[0,1]$ einen Viertelkreis mit Radius 1.)

[30] `http://javascript.about.com/library/blrecursive.htm` [2016-07-17]
[31] Zeidler (1996):S. 163.
[32] Forster (2008):§19.15.

16.10 DOM

Das DOM *(Document Object Model)*, ist eine Schnittstellenbeschreibung für API's, die Parser oder Browser zur Darstellung von HTML- oder XML-Dokumenten verwenden können. Für JavaScript (aber auch für jede andere Programmiersprache) ist im DOM festgelegt, welche HTML-Elemente als Objekte in zur Verfügung stehen und wie man dynamisch neue Elemente hinzufügen oder löschen kann, aber auch, welche Event-Handler für eine clientseitige Interaktion implementiert werden können. Obwohl das Grundkonzept des DOM als eine Baumstruktur von Knoten einfach ist, sind die Bezeichnungen und Dokumentationen zunächst etwas verwirrend. Neben der Standardreferenz sei daher die DOM-API des Mozilla-Projekts,

`https://developer.mozilla.org/en-US/docs/Glossary/DOM`

empfohlen. Bevor wir das DOM in seinen für uns wesentlichen Details untersuchen sei noch seine Historie erwähnt. Das DOM erschien sukzessive in mehreren Versionen, bis zur Version 3 Level genannt. Level 1 wurde im Oktober 1998 veröffentlicht, Level 2 im November 2000 und Level 3 im April 2004 .

Das DOM trägt seinen Namen „Objektmodell" vor allem, da es HTML- oder XML-Elemente als Objekte darstellt und in gegenseitige Beziehungen setzt, nämlich in eine Baumhierarchie wie in Abbildung 16.5 skizziert. In jedem Browser ist eine Objekthierarchie mit dem Wurzelobjekt

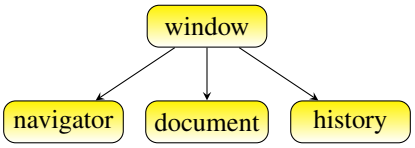


Abbildung 16.5: Objekt-Hierarchie des Browsers mit wichtigen Objekten, insbesondere das `document`-Objekt.

`window` vorhanden (Abbildung 16.5). Eines seiner Kindelemente ist `document` und bestimmt den im Browserfenster sichtbaren Inhalt. Das Objekt `navigator` liefert Informationen über den aktuellen verwendeten Browser, und `history` über die im aktuellen Tab oder Frame bisher aufgerufenen Webseiten.[33]

16.10.1 Web API des DOM: `Node`, `Document` und `Element`

Das zentrale Interface des DOM ist `Node`,[34] das einen allgemeinen Knoten des DOM-Baumes beschreibt. Ein Node im DOM hat u.a. stets einen `parentNode` und eine Liste von `childNodes` als Attribute und Methoden zum Hinzufügen oder Löschen von Kindknoten (`appendChild`, `removeChild`). Die wichtigsten Implementierungen von `Node` sind `document` und `element`. Ein `element`[35] ist ein allgemeines Element eines HTML- oder XML-Dokuments.

Das `document`-Objekt ist von der Klasse `Document`[36] und repräsentiert den Inhalt des darzustellenden Dokuments, also den Inhalt des Browser-Fensters. Mit der Methode `write()` des `document`-Objekts können wir mit JavaScript in das Browser-Fenster schreiben. Eine weitere wichtige Methode des `document`-Objektes ist `getElementById("ID")`, die ein Element des Dokuments als Ergebnis liefert, dessen Attribut `"id"` als Wert den übergebenen String hat. Dieses

[33] `https://developer.mozilla.org/en-US/docs/Web/API/Window`

[34] `https://developer.mozilla.org/en-US/docs/Web/API/Node`, `http://de.selfhtml.org/javascript/objekte/node.htm`

[35] `https://developer.mozilla.org/en-US/docs/DOM/Element` [2016-01-18]

[36] `https://developer.mozilla.org/en-US/docs/Web/API/Document`

Element ist von der Klasse `Element`. Beide Methoden haben wir bereits als einführende Beispiele für JavaScript-Ausgaben im Abschnitt 16.2 kennen gelernt.

In der Web-API des DOM sind `Document` und
codeElement implementieren das Interface `Node` und besitzen daher alle dort aufgeführten Properties und Methoden.

CSS-Anweisungen mit der Eigenschaft `style`

Ein `element`-Objekt hat unter anderem die Eigenschaft `style`, mit der CSS-Deklarationen über JavaScript angewiesen werden können.[35] Hierbei kann zu jedem HTML-Element dessen `style`-Attribut per JavaScript verändert werden, indem das zu jeweilige CSS-Attribut mit einem neuen Wert belegt wird. Dabei ist zu beachten, dass ein CSS-Attribut mit einem Bindestrich in JavaScript mit der Binnenmajuskel-Notation (*CamelCase*) übernommen wird, da dort der Bindestrich als Minuszeichen eine arithmetische Rechenoperation darstellen würde. In dem folgenden Beispielprogramm wird die Hintergrundfarbe des Textes in einem `span`-Element per Schaltfläche verändert, indem die Funktion `aendereFarbe` mit der gewünschten Hintergrundfarbe aufgerufen wird.

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8"/>
  <title>CSS-Anweisungen über JavaScript und DOM</title>
  <script>
    let aendereFarbe = function(farbe) {
       document.getElementById("text").style.backgroundColor = farbe;
    }
  </script>
</head>
<body>
  Hintergrundfarbe ändern:
  <button onclick="aendereFarbe('green');">Grün</button>
  <button onclick="aendereFarbe('red');">Rot </button>
  <span id="text">Die Zeichen an der Wand.</span>
</body>
</html>
```

Die entscheidende Anweisung dazu ist in Zeile 8, wo das CSS-Attribut `background-color` in Binnenmajuskelnotation in JavaScript als Attribut `backgroundColor` des Elementattributs `style` festlegt.

Das folgende auf Crockford[37] zurückgehende Programm beispielsweise verändert mit einer Closure langsam die Hintergrundfarbe des Browserfensters (das Element `document.body`) von gelb (`#ffff00`) auf weiß (`#ffffff`):

```
let fade = function(element, delay) {
   let level = 0;
   let step = function() {
      let hex = level.toString(16);     // level in hexadezimal
      element.style.backgroundColor = "#ffff" + hex + hex;
```

[37] Crockford (2008):S. 42.

```
      if (level < 15) {
         level += 1;          // Variable der äußeren Funktion!
         setTimeout(step, delay);  // Rekursion nach delay [ms]
      }
   };
   step();                           // erster Aufruf von step
};

fade(document.body, 100);
```

Hier rufen wir in Zeile 14 die Funktion `fade` auf und übergeben ihr das DOM-Objekt `document.body`, also den Knoten des HTML-Dokuments, der durch das Tag `<body>` erzeugt wurde, und die Verzögerungszeit `delay` in Millisekunden. Die Methode `fade` setzt zunächst die Variable `level` auf 0 und definiert als Closure eine rekursive Funktion `step`, in der `level` bei jedem hochgezählt wird und der Rekursionsaufruf mit Hilfe von `setTimeout` um `delay` verzögert wird. In Zeile 11 wird dann `step` erstmals aufgerufen und die Rekursion gestartet. Bei jedem weiteren Aufruf von `step` wird der Wert der Variablen `level` erhöht und als neuer Wert in `step` verwendet. Die Variablen `element`, `delay` und `level` sind hier aus Sicht der Closure also eine „globale“ Variable. Eleganter wäre sicherlich eine Rekursion, die ohne solche grlobalen Variablen auskommt, z.B. indem sie sie als Parameter übergibt:

```
   let step = function(element, delay, level) {
      ...
      if (level < 15) {
         step(element, delay, level + 1);
      }
   };
```

Leider ermöglicht `setTimeout` jedoch nur den verzögerten Aufruf von Funktionen ohne Parametern.

Hinzufügen und Löschen von Elementen

In dem folgenden Beispielprogramm wird mit Hilfe der Standardmethoden des DOM `createElement` in Zeile 9 und `appendChild` in Zeile 10 durch JavaScript dynamisch ein `<li>`-Element an das `<ol>`-Element mit der `id` angehängt.

Listing 16.3: DOM-Manipulation

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8"/>
  <title>DOM-Elemente</title>
  <script>
    let hinzufuegen = () => {
       let wurzel = document.getElementById('liste');
       let neu = document.createElement('li');
       let neuerText = document.createTextNode("Die Zeichen an der Wand.");
       neu.appendChild(neuerText);
       wurzel.appendChild(neu);
    }
    let loeschen = () => {
```

```
      let element = document.getElementById('liste');
      if (element.hasChildNodes()) {
         element.removeChild(element.lastChild);
      }
    }
  </script>
</head>
<body>
  <button onclick="hinzufuegen();">Neuer Text</button>
  <button onclick="loeschen();">Löschen</button>
  <ol id="liste"></ol>
</body>

</html>
```

Entsprechend wird in Zeile 18 das letzte Kindelement des `<ol>`-Elements mit der `id` mit der Methode `removeChild` gelöscht. Damit es nicht zu einem Laufzeitfehler kommt, wenn kein Kindelement des `<ol>`-Elements mehr vorhanden ist, wird das Löschen nur durchgeführt, wenn die Prüfung auf Existenz von Kindknoten mit der Methode `hasChildNodes` in Zeile 17 positiv ausgeht. Der DOM-Baum des Dokuments ist in Abbildung 16.6 skizziert. Die gestrichelt ge-

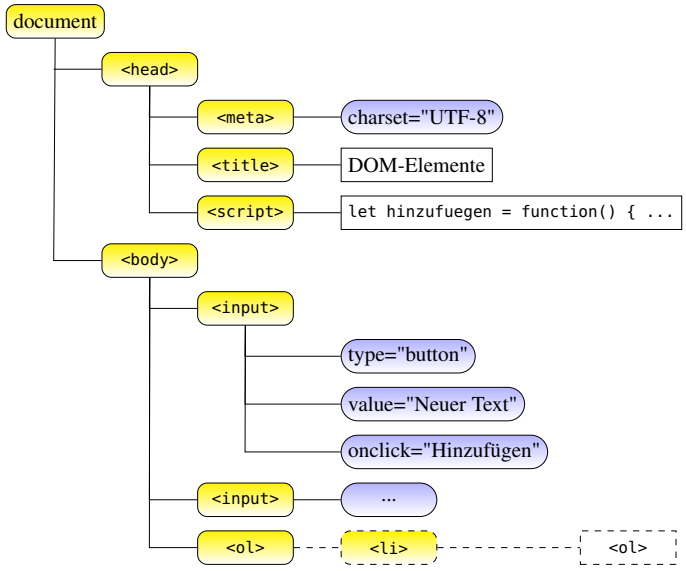


Abbildung 16.6: Der DOM-Baum des HTML-Dokuments aus Listing 16.3.

zeichneten Knoten werden durch das JavaScript-Programm je nach Eingabe hinzugefügt oder gelöscht.

16.10.2 Das window-Objekt

Das Browser-Fenster, in dem die HTML-Seite, die den JavaScript-Code enthält, dargestellt wird, wird durch ein `window`-Objekt repräsentiert.

Das `window`-Objekt verfügt über Methoden wie `prompt()` `alert()` oder `confirm()`, mit denen eigene Fenster zur Ein- und Ausgabe geöffnet werden können.

Darüberhinaus hat das `window`-Objekt Attribute, die sich von JavaScript aus steuern lassen. Dazu gehört die Statuszeile (Attribut `status`), der URL, der gerade dargestellt wird (`location`) oder die URL's der Seiten, die bis jetzt dargestellt wurden (`history`).

Tatsächlich sind sind sogar alle anderen JavaScript-HTML-Objekte entweder selbst Attribute des `window`-Objekts oder Attribute von Attributen des `window`-Objekts; so auch das `document`-Objekt, welches wir bereits öfter zum Schreiben in das Browser-Fenster benutzt haben. Das `window`-Objekt ist also die Wurzel der Objekthierarchie in JavaScript.

Das `window`-Objekt verfügt über drei Methoden, mit denen Benutzerinteraktion möglich ist. Die erste (`prompt()`) haben wir bereits kennengelernt. `prompt()` öffnet ein eigenes Fenster oberhalb des Browser-Fensters. Das Fenster enthält zwei Bereiche: im ersten wird eine Meldung dargestellt (erster Parameter von `prompt()`), im zweiten kann der Benutzer Eingaben tätigen (der zweite Parameter von `prompt()` ist eine Vorgabe im Eingabefeld).

`confirm()` erwartet vom Benutzer eine Bestätigung. Der einzige Parameter von `confirm()` ist eine Meldung, die im von `confirm()` aufgespannten Fenster dargestellt wird. Der Benutzer kann mit einem OK-Button bestätigen (`confirm()` liefert dann `true` zurück) oder mit einem Abbruch-Button abbrechen (`confirm()` liefert `false` zurück). Betrachten wir dazu folgendes Programm:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8"/>
  <script>
  let factorial = function (n) {
    if (n < 0) return NaN;
    if (n === 0) return 1;
    return n * factorial(n-1);
  }

  let einlesen = function () {
    let n = document.getElementById("eingabe").value;
    let wirklich = true;
    if (n > 150) {
      wirklich = confirm("Wirklich "+n+"! berechnen? Das kann lange dauern . . .");
    }
    if (wirklich) {
      document.getElementById("label").innerHTML = n + "! = " + factorial(n);
    } else {
      document.getElementById("label").innerHTML = "Berechnung von "+n+"! abgebrochen.";
    }
  }
  </script>
</head>
<body>
  <input type="text" id="eingabe"/>
  <input type="button" value="Rechnen" onclick="einlesen();"/>
  <p>Ergebnis: <span id="label"></span>.</p>
</body>
</html>
```

Hier muss der Anwender die Frage bestätigen, ob die Zahl, deren Fakultät berechnet werden soll, größer als 150 sein darf. In diesem Fall wird ein `confirm`-Fenster aufgebaut. Wählt der Benutzer „OK“, gibt `confirm` einfach `true` zurück. Der Boole'schen Variablen `wirklich` wird damit der Wert `true` zugewiesen und die Fakultäten werden berechnet. Klickt der Benutzer hingegen

Abbrechen, wird von `confirm` `false` zurückgegeben, `wirklich` wird damit `false` zugewiesen, und die Berechnung findet nicht statt.

16.11 XMLHttpRequest und AJAX

Ein weiteres wichtiges Objekt in JavaScript ist `XMLHttpRequest`. Es ermöglicht, über einen aus JavaScript ausgelösten Request, also eine Netzanfrage, Daten von einem beliebigen URL zu laden. Es spielt eine Schlüsselrolle bei AJAX-Anwendungen. Anders als der Name suggeriert, kann `XMLHttpRequest` nicht nur XML-Daten laden, sondern beliebige Formate, also auch Binärdateien wie Bilder und Streams. Ebenso kann es nicht ausschließlich HTTP-Requests auslösen, sondern auch andere Anfragen über andere Protokolle wie FTP oder FILE.

Als Objekt besitzt es als wichtigste Attribute `readystate`, `response` und `status`, als wichtigste Methoden `open` und `send` sowie den Event-Handler `onload`:[38]

XMLHttpRequest
`readystate` : int `response` : String `status` : int

`onload()`
`open(method, url [, ...])`
`send([data])`

Um einen HTTP-Request auszulösen, muss zunächst ein `XMLHttpRequest`-Objekt mit `new` instanziiert werden:

```
let request = new XMLHttpRequest();
```

Mit der Methode `open` wird ein Request initialisiert. Sie benötigt dafür mindestens zwei Parameter vom Typ String, sowie drei optionale Parameter:

```
void open(String method, String url, [boolean async, String user, String password])
```

Hierbei bezeichnet `method` die HTTP(S)-Methode, die einen der Werte `"GET"`, `"POST"`, `"PUT"` oder `"DELETE"` annehmen kann und `url` der angefragte URL, also der Pfad zur angefragten Datei. Der optionale Boole'sche Parameter `async` bestimmt, ob die Anfrage asynchron ablaufen oder das Programm die Antwort abwarten soll; er ist im Default auf `true` gesetzt. Die beiden optionalen Parameter `user` und `password` schließlich ermöglichen eine etwaig notwendige Authentifizierung für Zugang zu dem URL. Mit der Methode `send` wird die Anfrage abgeschickt. Dazu muss sie vorher mit `open` initialisiert sein. Ist sie asynchron initialisiert, so läuft das Programm weiter, andernfalls wird das Ende der Methode abgewartet. Die Methode kann optional mit zu übertragenden Daten als Parameter aufgerufen werden, z.B. Formulardaten. Die Funktion `onload`, ähnlich wie `onreadystatechange`, ist ein Event Handler, der auf Änderungen des Attributs `readyState` reagiert, das wiederum einen der 5 möglichen Werte 0, 1, 2, 3, 4 annehmen kann:

`readyState`	Zustand	Bedeutung
0	`UNSENT`	`open` ist noch nicht aufgerufen
1	`OPENED`	`send` ist noch nicht aufgerufen
2	`HEADERS_RECEIVED`	`send` ist aufgerufen und Headers und Status des Requests sind verfügbar
3	`LOADING`	Daten werden empfangen
4	`DONE`	Die Anfrage ist erledigt

[38] `https://xhr.spec.whatwg.org/#event-handlers`, `https://developer.mozilla.org/en/DOM/XMLHttpRequest`

Die beiden ersten Zustände dienen der Vorbereitung der Kommunikation, der Request wird erst durch die `send()`-Methode ausgelöst. Der Event-Handler `onload` reagiert nur, wenn der Request beendet und die Response korrekt angekommen ist, also wenn

`readyState === 4` und `status === 200`

Diese Kommunikation läuft übrigens *asynchron* ab, d.h. die `send()`-Methode blockiert oder beeinflusst nicht andere JavaScriptprogramme der Seite. Diese Möglichkeit, asynchrone Anfragen an den Server zu stellen, ohne dass die dargestellte Webseite blockiert wird oder gar neu geladen werden muss, bildet die technische Grundlage für AJAX-Anwendungen, die wir in Abschnitt 16.11.2 behandeln werden.

Der Event Handler ist zunächst nur eine „leere“ Funktion und muss mit einer Callback-Funktion implementiert werden, ähnlich wie wir es ja auch von den Event-Handlern des DOM bereits kennen. Üblicherweise wird hier die Antwort der Anfrage verarbeitet, die wiederum in dem Attribut `response` gespeichert ist. Im Überblick ergibt sich damit das Zustandsdiagramm eines

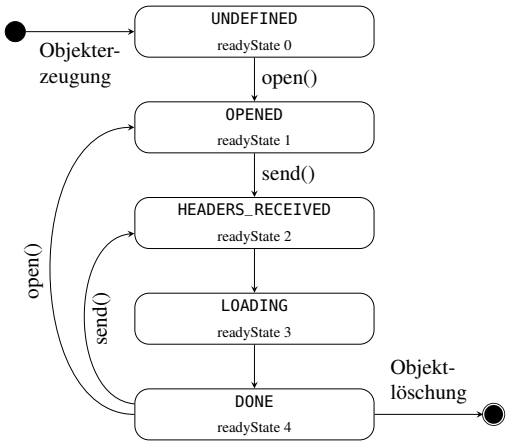


Abbildung 16.7: Zustandsdiagramm eines XMLHttpRequest-Objekts. Vgl. Gorski et al. (2015:S. 26).

XMLHttpRequest-Objekts nach Abbildung 16.7. Die Details sind im technischen Report unter `https://xhr.spec.whatwg.org/` der Arbeitsgruppe WHATWG des W3-Konsortiums festgelegt.

Beginnen wir mit einem einfachen Beispiel, das die vier Schritte Erzeugung eines Requests, Implementierung des Event Handlers, Öffnen und Senden eines Requests zeigt:

```
let request = new XMLHttpRequest();
request.onload => () { // Event Handler
   document.write(request.response);
}
request.open("GET", "http://localhost/");
request.send();
```

Die Reihenfolge der Anweisungen bei der Ausführung des Programms entspricht dabei nicht ganz der Reihenfolge ihrer Implementierung: Der Event-Handler `onreadystatechange` wird für das Request-Objekt zwar zu Beginn des Objektes implementiert, aber erst nach Beenden der Anfrage verarbeitet, also *nach* dem `send`. Er sollte aber natürlich bereits implementiert sein, wenn der Request gesendet wird.

Zu beachten ist, dass bei den Standardeinstellungen des Webservers eine Anfrage in JavaScript nur für denselben Server möglich ist, auf dem sich auch das JavaScript-Programm befindet. Möchte man über `XMLHttpRequest` einen anderen Server aufrufen, so müssen die Einstellungen

dieses Servers entsprechend angepasst sein. Man spricht in diesem Falle von *Cross Origin Resource Sharing (CORS)*.[39] Damit JavaScript einen XMLHttpRequest weiter verarbeitet, muss der Server nämlich die Antwort mit dem HTTP-Header

```
Access-Control-Allow-Origin: *
```

senden. Möglich ist auch die Variante `Access-Control-Allow-Origin: http://foo.bar`, wobei `foo.bar` die Domain ist, von der die Anfrage, also das JavaScript-Programm, kommt. Ein PHP-Skript kann diesen Header individuell senden, wenn es mit der Zeile

```
header("Access-Control-Allow-Origin: *");
```

beginnt.

Mit dem Standardobjekt `FormData` kann man dynamisch erstellte Formulardaten versenden.[40] Ein einfaches Beispiel übermittelt die Formulardaten `x=2` an ein PHP-Skript:

```
let request = new XMLHttpRequest();
request.onreadystatechange => () { // Event Handler
   document.write(request.response);
};
request.open("POST", "http://haegar.fh-swf.de/AJAX/spiegel.php");
let data = new FormData();
data.append("x",2);
request.send(data);
```

Mit der folgenden Änderung der Zeile 9 erhält man sogar die Übertragung eines dynamisch in JavaScript erzeugten Arrays `prim`:

```
data.append("prim[]",2); data.append("prim[]",3); data.append("prim[]",5);
```

16.11.1 Aktionen nach dem Request: `onload` und `JSON.parse`

Bislang haben wir nur Beispiele von XMLHttpRequest-Objekten kennengelernt, deren Event-Handler `onload` einfach nur die empfangene Antwort des Servers, d.h. den String `request.response`, ausgibt. Häufig werden allerdings komplexere Daten übermittelt, die in dem Event-Handler weiter verarbeitet werden sollten. Grundsätzlich können diese Daten als beliebiger Text vorliegen, üblich sind aber Strings in den drei folgenden Formaten HTML, JSON und XML. Die Verarbeitung

- HTML-Strings werden dabei in der Regel nicht weiterverarbeitet und unverändert dargestellt, also so wie in den obigen XMLHttpRequest-Beispielen mit `request.response`.
- Ein JSON-String kann mit der Standardmethode `JSON.parse()` direkt in ein JSON-Objekt umgewandelt werden, also beispielsweise

  ```
  let objekt = JSON.parse(request.respnse);
  ```

  Dieses Objekt kann nun in JavaScript beliebig weiterverarbeitet werden.
- Ein XML-String kann manuell geparst werden. Ist die Serverantwort beispielsweise ein SVG-String, so kann die Grafik direkt im Browser dargestelllt werden.

[39] `http://www.w3.org/wiki/CORS_Enabled`

[40] `https://developer.mozilla.org/en-US/docs/DOM/XMLHttpRequest/FormData/` [2016-07-17]; siehe auch: `https://developer.mozilla.org/en/DOM/XMLHttpRequest/FormData/Using_FormData_Objects`

Betrachten wir als Beispiel das folgende Programm, das die Daten des aktuellen Sondernangebots im JSON-Format erhält und im Event-Handler verarbeitet:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8"/>
  <script>
let ladenJSON = () => {
   let request = new XMLHttpRequest();
   request.onload = () => {
      let artikel = JSON.parse(request.response);
      let element = document.getElementById("angebot");
      let ausgabe = "Kaufen Sie unseren " +
         artikel.name + " mit der Nummer " +
         artikel.id + " für " + artikel.preis + " &euro;!";
      element.innerHTML = ausgabe;
   };
   request.open("GET", "http://haegar.fh-swf.de/json-artikel.php");
   request.send();
};
  </script>
</head>
<body>
  <h1>Unser Preisknaller!</h1>
  <p id="angebot">Klicken Sie, um unser heutiges Sonderangebot zu sehen:</p>
  <input type="button" value="JSON Eingabe" onclick="ladenJSON();";>
</body>
</html>
```

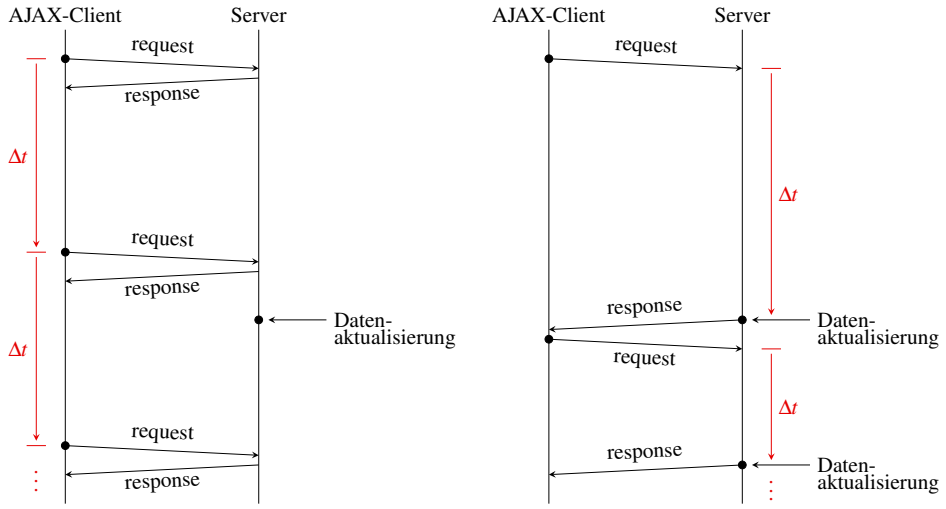
Bei Aufruf der HTML-Datei erscheint unter der Überschrift zunächst der Text „Klicken Sie, um unser heutiges Sonderangebot zu sehen:“. Klickt man den Button, so wird sein Event-Handler `onclick` aufgerufen, der wiederum die Funktion `ladenJSON` aufruft. In ihr wird ein XMLHttpRequest `request` erzeugt, das ein PHP-Skript auf dem Hägar-Server aufruft und einen JSON-String empfängt, der in dem Event-Handler `onload` zu einem vollständigen Text „Kaufen Sie unseren Prozessor mit der Nummer 123 für 99.99 €!“ ausformuliert wird und damit den vorherigen Text im `<p>`-Element überschreibt, allerdings nur unter der Bedingung, dass die HTTP-Antwort eingetroffen und der Status-Code OK lautet. (Der Status-Code 200 besagt, dass die HTTP-Anfrage erfolgreich verlief.) Das PHP-Skript auf Hägar ist übrigens nur ganz kurz und gibt einen immer gleichen String aus:

```
<?php
   header("Access-Control-Allow-Origin: *");
   echo '{
     "id"   : "123",
     "name" : "Prozessor",
     "preis": 99.99
   }';
?>
```

Wir können uns jedoch leicht vorstellen, dass das PHP-Programm stattdessen einen Datenbankzugriff ausführen könnte und so das Tagesangebot aktuell aus der Datenbank selektiert.

16.11.2 AJAX

Viele moderne Webanwendungen basieren auf AJAX (*Asynchronous JavaScript and XML*), das wiederum auf dem XMLHttpRequest-Objekt beruht. Ziel vieler AJAX-Anwendungen ist es, das starre Architekturmuster des Request-Response von HTTP zu überwinden, bei dem Anfragen nur vom Client ausgehen und der Server nur reagieren kann. Haben sich jedoch zentrale Daten auf dem Server aktualisiert, so kann der Server die Clients nicht darüber informieren. Bei einer Internetauktion beispielsweise würde ein Bieter nicht erfahren, wenn ein neues Gebot eines anderen vorläge, wenn er nicht selber immer wieder nachfragt. Ebenso würde bei einem Chatserver beispielsweise ein angemeldeter User nicht erfahren, wenn eine neue Nachricht eines Anderen eingetroffen wäre. Zwei Verfahren, den Client mit Hilfe eines asynchronen XMLHttpRequest-Objekts automatisiert von einer Datenaktualisierung in Kenntnis zu setzen, sind das Polling und das Long Polling, siehe Abbildung 16.8. Beim *Polling* wird in einem regelmäßigen Zeitabstand



(a) Polling (b) Long Polling

Abbildung 16.8: Polling und Long Polling als Methoden zur Synchronisation eines zentralen Datenbestandes. Vgl. Gorski et al. (2015:S. 27f).

Δt eine Anfrage an den Server gestellt, auf die er unverzüglich antwortet. Beim *Long Polling* dagegen stellt der Client eine asynchrone Anfrage, auf die der Server jedoch erst nach einer Aktualisierung seiner Daten antwortet. Empfängt der Client eine Antwort, stellt er sofort eine neue Anfrage, auf die er wiederum eine Antwort erst nach erfolgter Datenaktulisierung auf dem Server erhält.

Das Polling ist ein in JavaScript clientseitig relativ einfach zu implementierendes Verfahren, siehe Listing 16.4. Allerdings hat es den großen Nachteil, dass eine große Anzahl unnützer Anfragen gestellt werden, wenn die Anfragefrequenz höher als die typische Änderungsfrequenz der Daten ist. Vor allem Server mit sehr vielen Usern können dadurch unnötig belastet werden. Das Long-Polling dagegen lastet den Server optimal aus, da er nur noch zu den notwendigen Zeitpunkten an die betroffenen Clients Nachrichten sendet.

Listing 16.4: Polling durch Rekursion

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8"/>
    <script>
```

```
let periode = 2000; // Polling-Periode in ms
let data;
let request = new XMLHttpRequest();

request.onreadystatechange = function() { // Event Handler
  if (request.readyState === 4) {
    document.getElementById("ausgabe").innerHTML = request.response;
  }
}

let poll = function(x) {
   if (x > 5) return;  // Basisfall: beende Rekursion
   request.open("POST", "http://haegar.fh-swf.de/AJAX/spiegel.php");
   data = new FormData();
   data.append("x", x);   // x-Werte
   data.append("y", x*x); // y-Werte
   request.send(data);
   setTimeout("poll("+(x+1)+")", periode); // Rekursionsaufruf
}

poll(0); // Start der Rekursion
    </script>
  </head>

  <body>
    <div id="ausgabe"></div>
    <p>(<i>y</i> = <i>x</i><sup>2</sup>)</p>
  </body>
</html>
```

16.12 Web Workers: Nebenläufigkeit in JavaScript

Im Gegensatz zu Java oder C++ bietet JavaScript keine Möglichkeit, einzelne Threads zu programmieren, ein JavaScript-Programm wird grundsätzlich in nur einem einzigen Thread ausgeführt[41]. Bei rechenintensiven Algorithmen bewirkt diese Eigenart, dass das Programm solange blockiert und keine weiteren Anweisungen ausführt oder auf Benutzereingaben reagiert, bis die Berechnungen beendet sind.

Allerdings bietet JavaScript zur nebenläufigen Programmierung das Konzept der *Workers* oder *Web Workers* an.[42] Ein `Worker` in JavaScript ist ein Standardobjekt, das als Teilprozess im Hintergrund läuft, also nachrangig zum Hauptprozess des eigentlichen JavaScript-Programms, und isoliert für sich ausgeführt wird, insbesondere unabhängig von weiteren Benutzereingaben. Interaktionen sind ausschließlich mit dem Hauptprozess über die Methoden `postMessage` und `terminate` sowie den Event-Handler `onmessage` möglich. Um ein Worker-Objekt zu erzeugen, muss Quelltext des Workerprogramms in einer externen JavaScript-Datei ausgelagert vorliegen. Dies sei beispielsweise `worker.js`; dann wird im Hauptdokument das Worker-Objekt wie folgt

[41] Lau (2012).

[42] `https://dev.w3.org/html5/workers/`, `https://developer.mozilla.org/en-US/docs/DOM/Worker` [2021-10-21]

erzeugt:

```
let worker = new Worker("worker.js");
```

Die Kommunikation zwischen dem Hauptprozess und dem Workerprozess verläuft nun über den Austausch von Nachrichten. Wie in Abbildung 16.9 dargestellt reagiert der Event-Handler

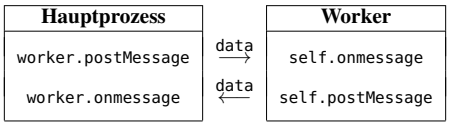


Abbildung 16.9: Kommunikation zwischen Hauptprozess und Worker.

des einen Prozesses stets auf die von `postMessage` gesendeten Nachricht des jeweils anderen Prozesses. Ein einfaches Beispiel ist das folgende HTML-Dokument, das einen bidirektionalen Nachrichtenaustausch bewirkt:

```
<!DOCTYPE HTML>
<html>
<head>
  <meta charset="UTF-8"/><title>Worker example: Send Messages</title></head>
<body>
  <p>
  <script>
   let worker = new Worker('worker.js');
   worker.onmessage = function (event) {
     document.getElementById('result').innerHTML = event.data;
   };
  </script>
  <input type="text" id="message" value="Hallo Welt!"/>
  <input type="button" value="Send"
         onclick="worker.postMessage(document.getElementById('message').value);"/>
  </p>
  <p>Message: <output id="result"></output></p>
</body>
</html>
```

Die zugehörige Worker-Datei lautet:

```
let count = 0;
self.onmessage = function (event) {
  count++;
  self.postMessage(event.data + ' (No ' + count + ')');
};
```

16.13 DOM Storage

DOM Storage, oder *Web Storage* nach W3C,[43] ist eine Web-Technik, mit der über einen Browser Daten von mehreren MB Größe auf einem Rechner gespeichert werden können. DOM Stora-

[43] Die Namensgebung ist einigermaßen kompliziert, nach W3C heißt es Web Storage, aber auch DOM Storage ist möglich, allerdings setzt es nicht die Existenz eines tatsächlichen Dokument-Objekts voraus `http://dev.w3.org/html5/webstorage/#terminology`.

ge ermöglicht damit eine persistente Datenspeicherung ähnlich wie ein Cookie, allerdings mit der entscheidenden Einschränkung, dass Datenzugriffe nicht über das HTTP-Protokoll geschehen, sondern vollständig vom Browser gesteuert und durch clientseitige Skripte (in der Regel JavaScript) durchgeführt werden.

DOM Storage speichert Daten in einem assoziativen Array, in dem die Schlüssel und Werte Strings sind. Sie können also in JavaScript einfach mit `JSON.parse` als Objekt erzeugt und mit `JSON.stringify` in DOM Storage gespeichert werden.

16.13.1 Lokale und sessionspezifische Speicherung

Daten, die lokal gespeichert werden, die sogenannten *Local Shared Objects (LSO)*, sind mit einer Domain verknüpft und bleiben auch nach Beenden des Browsers bestehen. Alle Skripte einer Domain, von der aus die Daten gespeichert wurden, können auf die Daten zugreifen. Bei Mozilla Firefox werden die Daten in der Datenbankdatei webappsstore.sqlite gespeichert.

Session-spezifisch gespeicherte Daten sind mit dem erzeugenden Browser-Fenster verknüpft und auf dieses beschränkt. Gespeicherte Daten werden beim Schließen des Browser-Fensters gelöscht. Diese Technik bietet die Möglichkeit, mehrere Instanzen derselben Anwendung in verschiedenen Fenstern laufen zu lassen, ohne dass es zu einer gegenseitigen Beeinflussung kommt, was von Cookies nicht unterstützt wird.

In JavaScript werden beide Speicherarten durch die Standardobjekte `localStorage` und `sessionStorage` realisiert und bieten die Methoden `setItem` und `getItem` zum Speichern und Lesen der Daten:

```
// Speichert ein Key-Value-Paar im localStorage
localStorage.setItem('schluessel', 'wert');
// Liest den eben gespeicherten Wert aus und zeigt ihn an
alert(localStorage.getItem('schluessel')); // => wert

// Speichert ein Key-Value-Paar im sessionStorage
sessionStorage.setItem('key', 'value');
// Liest den eben gespeicherten Wert aus und zeigt ihn an
alert(sessionStorage.getItem('key')); // => value
```

Für weitere Informationen zu DOM Storage mit JavaScript siehe `http://www.jstorage.info/`

Literatur

Ackermann, P. (2021). *Webentwicklung. Das Handbuch für Fullstack-Entwickler*. 1. Aufl. Rheinwerk: Bonn.

Crockford, D. (2008). *Das Beste an JavaScript*. O'Reilly: Köln.

Fielding, R. T. (2000). „Architectural Styles and the Design of Network-based Software Architectures". Diss. Irvine: University of California. URL: http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm.

Forster, O. (2008). *Analysis 1*. 9. Aufl. Vieweg: Wiesbaden.

Gorski, P. L., L. L. Iacono und H. V. Nguyen (2015). *WebSockets. Moderne HTML5-Echtzeitanwendungen entwickeln*. Hanser: Müchen.

Hauser, T. (2006). *XML-Standards. schnell + kompakt*. entwickler.press: Frankfurt.

Lau, O. (2012). „Verteiltes Rechnen mit JavaScript". In: *c't kompakt Programmieren* 3. http://www.ct.de/cs1207014.

Malcher, F., J. Hoppe und D. Koppenhagen (2020). *Angular. Grundlagen, fortgeschrittene Themen und Best Practices – inkl. RxJS, NgRx und PWA*. 3. Aufl. iX-Edition. dpunkt.verlag: Heidelberg.

Maurice, F. (2014). *PHP 5.5 und MySQL 5.6*. 3. Aufl. dpunkt.verlag: Heidelberg.

McLuhan, M. (1962). *The Gutenberg Galaxy*. https://books.google.com/books?id=7K7rKIWaXXIC. University of Toronto Press: Toronto.

— (1964 (reissued 2001)). *Understanding Media. The Extensions of Man*. https://books.google.com/books?id=K4AWBwAAQBAJ. Routledge: Oxon.

Piepmeyer, L. (2010). *Grundkurs Funktionale Programmierung mit Scala*. Carl Hanser Verlag: München Wien.

Sosinsky, B. (2009). *Networking Bible*. https://books.google.com/books?id=3DOREqRZejcC. Wiley: Indianapolis.

St. Laurent, S. und M. Fitzgerald (2006). *XML kurz & gut*. 3. Aufl. O'Reilly Verlag: Köln.

Theis, T. (2018). *Einstieg in PHP 7 und MySQL*. Rheinwerk: Bonn.

Turing, A. M. (1936–1937). „On computable numbers, with an application to the Entscheidungsproblem". In: *Proc. London Math. Soc.* 42(2). http://www.turingarchive.org/browse.php/B/12, S. 230–265.

Wenz, C. (2015). „Die 10 Gebote. Zehn Dinge, die man über JavaScript wissen muss". In: *Entwickler Magazin Spezial* 5. https://entwickler.de/entwickler-magazin/, S. 10–16.

Wenz, C. und T. Hauser (2021). *PHP 8 und MySQL. Das umfassende Handbuch*. 4. Aufl. Rheinwerk: Bonn.

Wojcieszak, M. (2015). „Binäre Austauschformate". In: *Java Magazin* 6.

Zeidler, E., Hrsg. (1996). *Teubner Taschenbuch der Mathematik. Teil 1*. B. G. Teubner: Leipzig.

Internetquellen

[DOM] https://w3.org/DOM – Spezifikation des W3C des DOM.

[GMP] http://www.masterplanthemovie.com/ – Ozan Halizi und Jürgen Mayer: *„The Google Master Plan"*, Film nach ihrer Bachelor-Thesis an der FH Ulm 2007. Kritische Betrachtung zu Google und dessen Wachstum. Der Film ist auch unter http://www.youtube.com/watch?v=9zKXCQpUnMg zu finden.

[JS] `https://developer.mozilla.org/en-US/docs/Web/JavaScript` – JavaScript-Dokumentation des Mozilla Developer Network.

[JSL] `http://www.jslint.com/` – JSLint, Programm zur Überprüfung der Qualität von JavaScript Quelltext.

[OWASP] `https://owasp.org/` – Open Web Application Security Project (OWASP), gemeinnützige („501(c)(3)“) Organisation mit dem Ziel, die Sicherheit von Software im WWW zu verbessern und durch Transparenz Entscheidungskompetenz bei Endanwendern und Organisationen in Fragen von Sicherheitsrisiken zu schaffen. Empfehlenswert ist die Wikiseite zu Sicherheitslücken (*vulnerability*): `https://www.owasp.org/index.php/Category:Vulnerability`

[PHP] `http://php.net/manual/de/` – PHP-Handbuch

[PS] `http://sandbox.onlinephpfunctions.com` – *PHP Sandbox*, Möglichkeit zum Testen von PHP-Quelltextfragmenten (Snippets). Die Sandbox wird bereitgestellt von Jereon (John) Post, die Website liefert Informationen und Tutorials über PHP.

[SuMa] `http://suma-ev.de/` – SuMa eV, gemeinnütziger Verein zur Förderung der Suchmaschinentechnologie und des freien Wissenszugangs. Hauptziel: Aufbau einer dezentralen und kooperativen Suchmaschinen-Struktur in Deutschland, die schwer monopolisierbar ist.

[TF] `https://www.tensorflow.org/` – TensorFlow, quelloffene Softwarebibliothek, die die Programmierung neuronaler Netze auf verteilten Systemen ermöglicht.

[W3C] `http://www.w3c.org/` – World Wide Web Consortium, Gremium zur Standardisierung der Web-Techniken; eine deutsche Übersetzung wichtiger Teile dieses Webauftritts findet man unter `http://www.edition-w3c.de/`, auf die auch das Deutsch-Österreichische Büro W3C.DE/AT `http://www.w3c.de/` bzw. `http://www.w3c.at/` verlinkt.

[XSD] `http://www.w3c.org/TR/xmlschema-0/` – Einführung in XML Schema, auf deutsch unter `http://www.edition-w3c.de/TR/2001/REC-xmlschema-0-20010502/`

Index