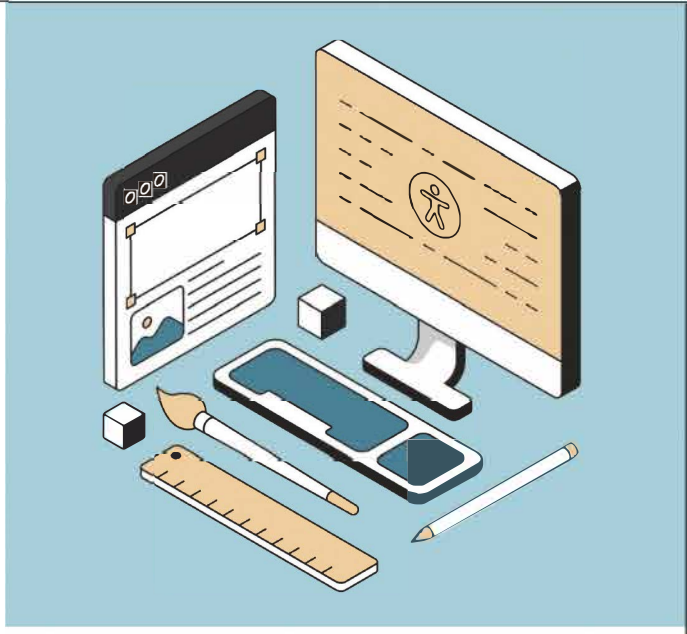
Uwe Mutz



Webseiten

programmieren und gestalten

Das umfassende Handbuch

Ohne Vorkenntnisse loslegen

- Alle Grundlagen: Programmierung, UX-Design, Datenbanken, Sicherheit
- Von der Idee über das erste HTML bis zur komplexen Website
- Mit vielen Codebeispielen, Übungen und Praxistipps

Alle Beispielprojekte zum Download

Rheinwerk Computing

Kapitel 3
Das liebe Internet – ein wenig Basiswissen

Was genau ist das Internet aus technischer Sicht? Wie viel technisches Verständnis brauche ich für meine Arbeit? Was ist eine Client-Server-Umgebung, und warum ist das für uns als Webdesigner wichtig?

Unser Ziel ist es, Websites zu erstellen. Für diese ist das Internet selbstverständlich die Grundlage. Was das Internet aus technischer Sicht genau ist, fragen wir uns eher selten. Für unsere Zunft ist es aber hilfreich, auch über Domains, Provider und Protokolle Bescheid zu wissen.

Beginnen wir ganz von vorne, sozusagen bei einer großen Menge Kabel, die quer über den Globus gezogen wurden. Sie bilden die Grundlage dessen, was wir als das *Internet* bezeichnen. Etwas technischer gesprochen, meint man einen »großen Verbund an Rechnernetzen, die autonom fungieren«. Gerade der Begriff der *Automie* ist sehr entscheidend für das Verständnis des Internets, denn die Idee für das eigentliche Verlegen dieser Kabel entstand der Legende nach zunächst einmal durch das Militär (konkret: das amerikanische Verteidigungsministerium), was in Wahrheit nicht ganz korrekt ist: Obwohl die Finanzierung des damaligen »ARPA-Net« von der DARPA (»Defense Advance Research Projects Agency«) übernommen wurde, wurden dennoch hauptsächlich zivile Projekte (und zwar insbesondere Projekte mit Forschungscharakter) finanziert. Dieses ARPA-Net setzte sich zum Ziel, Großrechner von Universitäten und anderen Forschungszentren miteinander zu verbinden, um einerseits die Verteilung von Forschungsergebnissen zu erleichtern und andererseits die Kommunikation zu verbessern.

Die Legende rund um das Internet

Die weitverbreitete Legende besagt, dass das US-amerikanische Verteidigungsministerium in Zeiten des Kalten Kriegs gegenüber einem atomaren Anschlag gerüstet sein wollte und deshalb ein Kommunikationsnetz aufbauen ließ.

Die Kommerzialisierung des Internets (im Rahmen des WWW, so wie wir es heute kennen) begann erst im Jahre 1990, als die National Science Foundation der USA beschloss, das Internet über die Nutzung durch Universitäten und Forschungszentren

hinaus der (damals noch nicht so) breiten Öffentlichkeit zur Verfügung zu stellen. Bereits im Jahr zuvor entwickelte ein Physiker namens Tim Berners-Lee am Kernforschungszentrum CERN (nahe Genf in der Schweiz) eine Auszeichnungssprache namens HTML, die er im Jahr 1991 veröffentlichte.

Wie wir heute wissen, wurde damals der Grundstein für das heutige Internet gelegt – wohlweislich, dass der damalige Gedanke weniger auf der Erstellung unserer heute bekannten Websites, sondern vielmehr auf der Präsentation (und Verknüpfung) wissenschaftlicher Daten lag. Zu diesem Zeitpunkt gab es jedoch noch keinen Browser für die Darstellung von Hypertex. Eine solche Software (namens Mosaik) kam erst einige Jahre später (1993) auf den Markt, verdrängte aber alles Dagewesene mit einem Schlag.

3.1 Dienste im Internet, Server und Client

Da jedoch ein jedes Netzwerk – wie wir heute wissen – nur so gut sein kann wie seine Nutzung, war eine der ersten wesentlichen Aufgaben die Definition dessen, was man eigentlich mit diesem neu geschaffenen Netzwerk anstellen wollte. Unter einem »Dienst« versteht man – einfach gesprochen – die Nutzung eines Netzwerkes zu einem bestimmten Zweck. So wurden in den Jahren mehr und mehr Dienste erdacht, die über das Internet nutzbar sein sollten und sollen:

- Steuerung von Geräten in einem Netzwerk: Telnet (ab dem Jahr 1969)
- Versand von Nachrichten: Mail-Dienst (1971)
- Übertragen von Daten: File Transfer (1971)
- Diskussionsforen: Usenet (1979)
- Chat/Messenging: IRC (Internet Relay Chat; 1988)
- Darstellung von Informationen: WWW – World Wide Web (1989)

Ein sehr wesentliches Prinzip von Diensten im Internet ist die Art und Weise, wie eine Kommunikation zwischen zwei beteiligten Stellen zustande kommt. Die Stelle, die etwas benötigt, nennen wir den *Client*, und die Stelle, die diesen Dienst oder diese Information bereitstellen kann, ist der *Server*:

- **Client**: Er initiiert eine Anfrage (*Request*) an einen Server.
- **Server**: Er antwortet auf die Anfrage mit einer Antwort (*Response*) an den Client.

An dieser Stelle ist es sehr wichtig zu erkennen, dass die Kommunikation zunächst immer vom Client ausgeht – der Server wird (bis auf wenige Ausnahmen) von sich aus nicht tätig. Wie die Kommunikation dann tatsächlich abläuft, müssen wir in den folgenden Abschnitten erst klären.

Ein jeder Dienst im Internet benötigt in der Regel einen auf ihn zugeschnittenen Client und ebenso einen auf ihn zugeschnittenen Server. Deutlich wird das, wenn wir beispielsweise an die Dienste WWW und E-Mail denken:

- **Um den WWW-Dienst zu benutzen**, verwendet ein User einen Browser (Google Chrome, Mozilla Firefox, Apple Safari oder Microsoft Edge), der auf dem lokalen Gerät des Users installiert wird. Demnach ist ein Browser ein Client für den WWW-Dienst. Um dann in weiterer Folge tatsächlich eine Website in diesem Client betrachten zu können, muss ein zugehöriger Server existieren, der dem User die Daten bereitstellt, die zu der Website gehören. Einen solchen Server nennen wir einen *Webserver*, und somit kann man sagen, dass der WWW-Dienst einen Browser als Client und einen Webserver benötigt.
- **Um den E-Mail-Dienst zu benutzen**, benötigt ein User ein Mail-Programm (MS Outlook, Thunderbird etc.), mit dessen Hilfe er E-Mails senden und empfangen kann – mit anderen Worten einen *Mailclient*. Klarerweise werden die zu sendenden Mails von einem Server versendet und zu empfangene Mails auf einem Server (zwischen-) gespeichert. Demnach hat man es mit einem *Mailserver* zu tun.

Diese Darstellung kann man auf sämtliche Dienste ausweiten.

3.2 Protokolle und Ports – eine Frage der Kommunikation

Um die oben angesprochenen Dienste (oder mit anderen Worten: Anwendung, Verwendung, Nutzung) zu ermöglichen, ist eine Technik erforderlich, mit deren Hilfe die jeweiligen Daten über ein Netzwerk übertragen werden können. An dieser Stelle ist es wichtig zu wissen, dass die Daten in Form sogenannter *Pakete* übertragen werden, was bedeutet, dass ein »Block an Daten« in kleine, handhabbare Stücke, eben die Pakete, zerteilt wird. Wie Sie im Folgenden sehen werden, ist das Internet-Protokoll für diese Filetierung der Daten zuständig – aber hierzu gleich mehr.

Man unterscheidet – ohne dass wir ganz ins Technische abdriften wollen – vier Nutzungsebenen bzw. (OSI-)Schichten:

1. **Netzzugangsschicht** (OSI-Schichten 1 und 2): Die unterste Ebene stellt den (physischen) Netzzugang dar. Zu dieser Schicht gehören keine Software-Lösungen, sondern sogenannte *Netzwerk-Topologien*, also Arten und Weisen, wie Geräte in einem Netzwerk miteinander verbunden werden können.
2. **Netzwerk- oder Internetschicht** (OSI-Schicht 3): Diese Ebene ist für die (Weiter-) Vermittlung von Datenpaketen und für die Wahl der am besten geeigneten Route (*Routing*) zuständig. Man spricht an dieser Stelle von einer *Punkt-zu-Punkt-Verbindung*, da ein Paket immer von einer Stelle zu einer anderen Stelle weitergeleitet

wird. Der Kern dieser Schicht ist das *Internet Protocol (IP)*. Dieses Protokoll ist an sich unzuverlässig, da es sich nicht darum kümmert, dass ein Datenpaket auch tatsächlich ankommt. Jedes (physische) Gerät erhält zudem eine IP-Adresse, um mit anderen Geräten kommunizieren zu können.

3. **Transportschicht** (OSI-Schicht 4): Anders als die Netzwerkschicht stellt die Transportschicht keine Punkt-zu-Punkt-, sondern eine *Ende-zu-Ende-Verbindung* her. Das bedeutet, dass eine Form der *Übertragungskontrolle* erfolgen kann, wenn Sender und Empfänger voneinander wissen. Der Kern dieser Schicht ist das *Transmission Control Protocol (TCP)*, das eine Übertragungskontrolle ermöglicht. Man spricht auch von einem verbindungsorientierten und zuverlässigen Protokoll.

   Nicht in jedem Fall ist man jedoch an einer Übertragungskontrolle interessiert, da eine jede Form der Übertragungskontrolle ein gewisses Maß an Overhead in Form von Wartezeit bedeutet. Aus diesem Grund existiert in dieser Schicht ein zweites Protokoll namens *User Datagram Protocol (UDP)*, das verbindungslos und demnach unzuverlässig arbeitet.

[i]

Übertragungskontrolle ja oder nein?

Übertragungskontrolle bedeutet, dass überprüft werden kann, ob ein jedes Datenpaket, das von einem Sender versendet wurde, auch tatsächlich beim Empfänger ankommt. Hierzu muss der Sender dem Empfänger eine gewisse Zeit einräumen, bis dieser beim Sender meldet, dass das Paket angekommen ist. Geschieht dies nicht in der vorgegebenen Zeit, schickt der Sender das Paket erneut.

Im Falle von Live-Übertragungen oder auch der Videotelefonie verzichtet man zumeist auf eine Übertragungskontrolle: Sollte das eine oder andere Datenpaket nicht »ankommen«, so bedeutet dies lediglich eine kurze Störung und stellt somit kein gravierendes Problem dar. Würde hingegen bei einer E-Mail ein Teil fehlen, so wäre das eher schmerzhaft.

4. **Anwendungsschicht** (OSI-Schichten 5 bis 7): Die Anwendungsschicht stellt Protokolle bereit, die die jeweiligen Anwendungen (Dienste) für die Verarbeitung ihrer Daten benötigen. Aus diesem Grund sind die Protokolle in dieser Schicht auch sehr zahlreich (und es werden mit jedem neuen Dienst im Internet auch mehr). Hierzu zählen:
   - **HTTP**: Hypertext Transfer Protocol
   - **FTP**: File Transfer Protocol
   - **SSH**: Secure Shell
   - **SMTP**: Simple Mail Transfer Protocol (Versand von E-Mails)
   - **POP3**: Post Office Protocol (Version 3) (Abruf von E-Mails)
   - **Telnet**: Login auf entfernten Rechnern (Remote Terminal)

- **DNS**: Domain Name System (Übersetzung eines Domainnamens in eine IP-Adresse)
- **SNMP**: Simple Network Management Protocol (Verwalten von Geräten in einem Netzwerk)

Das OSI-7-Schichten-Modell

In Wahrheit besteht das OSI-Modell aus sieben Schichten. Mit jeder Schicht werden an die Daten weitere sogenannte *Header* gehängt (bzw. ihnen vorangestellt), um die einzelnen Schichten voneinander zu trennen und deren Steuerinformationen zu speichern.

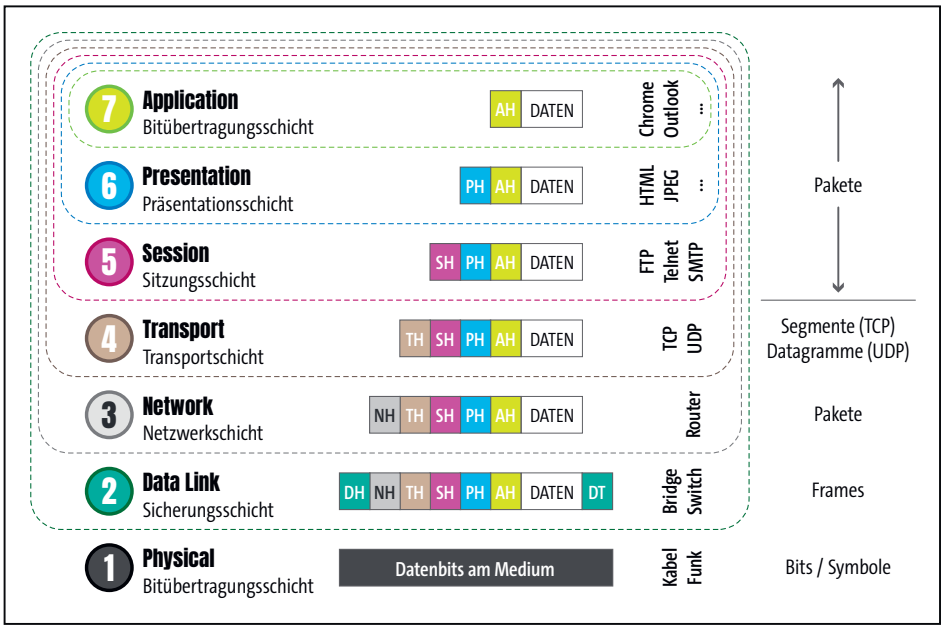


Abbildung 3.1 Die sieben Schichten des OSI-Modells

Sehr oft liest man im Zusammenhang mit der Datenübertragung im Internet vom *TCP/IP-Protokollstack*. Die obigen Ausführungen zeigen, dass die beiden Protokolle IP und TCP sehr eng miteinander verwoben sind, wenn es um eine ausfallsichere Übertragung von Daten geht. Unter einem »Stack« versteht man jedoch nicht mehr, als dass die einzelnen Protokolle übereinandergestapelt werden: Ein Protokoll baut auf dem anderen Protokoll auf.

Sie erkennen ebenso, dass einem jeden Dienst im Internet (mindestens) ein Protokoll zugewiesen ist. Dieses ist speziell auf die Anforderungen des Dienstes zugeschnitten. Protokolle sind im Wesentlichen nichts anderes als Vereinbarungen über die Form

der Kommunikation, also über die Art und Weise, wie die Datenübertragung zwischen einem Sender und einem Empfänger vonstatten gehen soll.

Die Begriffe »Sender« und »Empfänger« können allerdings nicht in die Begriffe »Client« und »Server« umgemünzt werden:

- Im Falle eines Requests ist der Client der Sender und der Server der Empfänger.
- Im Falle des (folgenden) Responses ist der Server der Sender und der Client der Empfänger.

3.2.1 HTTP/1.1, HTTP/2 oder HTTP/3 und die Statuscodes

HTTP (*Hypertext Transfer Protocol*) ist dasjenige Protokoll, das für die Bereitstellung/Darstellung von Websites benötigt wird. Während einer Anfrage eines Clients an einen Server (*Request*) oder während einer Antwort eines Servers an einen Client (*Response*) sorgt es dafür, dass die betreffenden Informationen übermittelt werden. Jede solche Information (oder auch *Nachricht*) ist in zwei Teile aufgeteilt: Der *Header* beinhaltet beispielsweise Informationen zur Datenkodierung oder zum Typ der übertragenen Daten (vergleiche hierzu auch Abschnitt 3.5), der *Body* (oder auch *Rumpf*) beinhaltet dann die eigentlichen Nachrichtendaten.

Es handelt sich bei HTTP außerdem um ein *zustandsloses* Protokoll, das mehrere Anfragen eines Systems (z. B. eines Browsers) an ein anderes System (z. B. einen Server) grundsätzlich als separate Transaktionen behandelt und sie somit nicht speichert.

Seit der Entwicklung bzw. Veröffentlichung von HTTP im Jahr 1991 und der Weiterentwicklung zur ersten wirklich verwendbaren Version im Jahr 1997 hat sich viel getan. Im Jahr 1999 wurde HTTP/1.1 veröffentlicht, das eines der wesentlichen Probleme von HTTP/1 löste: Während HTTP/1 nur zwei TCP-Anfragen zur selben Zeit an ein entferntes System ermöglicht, sind in HTTP/1.1 mehrere (üblicherweise sechs) gleichzeitige Anfragen möglich.

HTTP/2, veröffentlicht 2015, sollte auf der anderen Seite vor allem die Übertragung von Daten beschleunigen und optimieren – mit der Absicht, vollständig abwärtskompatibel zu HTTP/1.1 zu sein. Neben diesen sehr wesentlichen Eigenschaften sind weitere Möglichkeiten hinzugekommen:

- Zusammenfassen mehrerer Anfragen
- verbesserte Datenkompression sowie binärkodierte Übertragung von Inhalten
- Priorisierung einzelner Datenströme (Streams)
- Server-initiierte Datenübertragung (die sogenannte »Push-Methode«)

Ersteres ist der Grundstein für die Beschleunigung in der Datenübertragung; die Kompressionsmethode HPACK verbessert die Datenkompression, indem HPACK

nun auch die Kopfdaten mit in die Kompression integrieren kann, und die Server-initiierte Datenübertragung verringert die *Latenz* zwischen Client und Server.

Latenz

Unter der *Latenz* versteht man die Zeit zwischen dem Auftreten eines Ereignisses und dem Auftreten eines erwarteten Folgeereignisses.

Während HTTP/2 noch auf TCP aufsetzt, baut das 2018 entworfene und 2022 beschlossene HTTP/3 auf UDP auf. Das ist deshalb möglich, weil die Datenströme bei HTTP/3 getrennt verarbeitet werden.

Dennoch ist die Frage noch nicht beantwortet, ob HTTP/1.1, HTTP/2 oder HTTP/3 das Protokoll unserer Wahl ist. Die Antwort ist dennoch sehr einfach: Bei der Webentwicklung können wir uns das Protokoll nicht aussuchen, denn einerseits muss der Server das Protokoll verwenden und andererseits muss der Browser das Protokoll verstehen. Dies ist laut heutigem Stand (2023) vor allem HTTP/2 im Falle (halbwegs) moderner Provider und HTTP/1.1 im Falle etwas älterer Provider. Für HTTP/3 ist derzeit noch kein Platz, leider.

Allen Versionen des Protokolls ist jedoch gemeinsam, dass sie mit *Statuscodes* arbeiten, um den Erfolg einer Anfrage an ein System (Request) mit einer Antwort (Response) darzustellen. Einige dieser Statuscodes – wie etwa der bekannte Code 404 für »Dokument nicht gefunden« – sind bekannt. Recht wesentlich ist jedoch die Erkenntnis, dass es mehrere Spannen dieser Statuscodes gibt:

1. **1xx – Informationen**: Sehen Sie einen Statuscode, der mit 1 beginnt, dauert die Bearbeitung des Requests noch an, obwohl bereits eine Rückmeldung an den Client erfolgt. Man spricht von einer »Zwischenantwort«, die notwendig ist, da Clients nach einer gewissen Zeitspanne davon ausgehen, dass das angefragte System nicht mehr antwortet, da serverseitig ein Fehler aufgetreten ist. Die Statusinformation 100 besagt beispielsweise, dass die laufende Anfrage am Server noch nicht zurückgewiesen wurde.
2. **2xx – Erfolgreiche Operation**: Die Anfrage des Clients wurde erfolgreich bearbeitet, die Rückantwort des Servers an den Client ist erfolgt. Die meist erhoffte Operation ist 200; sie besagt, dass alles glattgegangen ist.
3. **3xx – Umleitung**: In manchen Fällen kann die Anfrage des Clients vorerst nicht beantwortet werden, da weitere Schritte des Clients notwendig sind. In vielen Fällen betrifft dies Anfragen nach Dokumenten, die unter der angefragten URL nicht mehr verfügbar sind. Der Server jedoch »weiß«, welche URL der Client anfragen müsste. Diese URL teilt er dem Client nun mit, um eine neuerliche Anfrage an den Server zu stellen. Beispielsweise besagt die Umleitung 302, dass die angeforderte

Resource (z. B. ein PHP-Dokument) nunmehr unter einer anderen Adresse bereitgestellt wird.

4. **4xx – Client-Fehler**: Man unterscheidet bei einer Anfrage eines Clients an einen Server zwei Arten von Fehlern: Fehler, die ein Client verursacht hat, und Fehler, die ein Server verursacht hat. Im Falle von 400er-Fehlern handelt es sich um clientseitige Fehler. Die bekanntesten Fehler sind die Fehler 404 (»Dokument nicht gefunden«), der gemeldet wird, wenn ein Client versucht, auf ein Dokument zuzugreifen, das nicht existiert; sowie der Fehler 403 (»Verboten«), wenn ein Client einen Zugriff auf ein Dokument versucht, auf das er keinen Zugriff hat (beispielsweise auf ein geschütztes Dokument, das nur serverseitig verfügbar ist).
5. **5xx – Server-Fehler**: Bei 5xx-Fehlern liegt das Problem auf Serverseite. Beispielsweise tritt dieser Fehler auf, wenn in einem serverseitigen Code ein Fehler existiert und der Programmcode nicht ausgeführt werden kann. Der Fehler 500 (»Serverfehler«) ist ein sehr unangenehmer Fehler, da er im Wesentlichen nur mitteilt, dass am Server (irgendein) Fehler aufgetreten ist.

Neben dem Statuscode wird im Header der Antwort des Servers an den Client eine (englischsprachige) Beschreibung des Status übermittelt. Bei einem Fehler 404 ist dies beispielsweise der Text »Not Found«, der gegebenenfalls clientseitig weiterverarbeitet wird.

3.2.2 Ports – die Türchen der Protokolle

Betrachtet man die Kommunikation zwischen dem Gerät des Users und einer Server-Hardware (Server), so fällt auf, dass es Situationen gibt, in denen ein und dasselbe Gerät mehrmals (und vielleicht sogar gleichzeitig/parallel) mit demselben Server kommunizieren möchte. Ohne bereits ins Detail der Kommunikation zu gehen: Es muss gewährleistet sein, dass dieser eine Server auf die unterschiedlichen Anfragen immer eindeutig antworten kann. Aus diesem Grund verwendet das Gerät des Users für die Anfrage an einen Server eine zusätzliche Information namens *Port*. Diese beschreibt im Wesentlichen nicht mehr als eine Art Postfach, an das der Server seine Antwort liefern darf. Diese Ports nennen wir *dynamische Ports*.

Demgegenüber werden den unterschiedlichen Diensten (respektive den Protokollen, die den Diensten zugewiesen sind) ebenso Ports zugewiesen. In diesem Fall spricht man von den *well-known Ports* und den *registrierten Ports* (je nach Portbereich).

Insgesamt existieren 65.536 Ports (16 Bit = 2^{16} = 65.536), von denen die ersten 1024 Ports (also diejenigen von 0–1023) die oben angesprochenen well-known Ports (oder auch *Standardports*) darstellen. In diesem Bereich befinden sich die Ports, die zu den Protokollen der oben genannten Dienste gehören (siehe Tabelle 3.1).

Port	Protokoll	Port	Protokoll	Port	Protokoll
20	FTP (Daten)	**25**	SMTP	**143**	IMAP
21	FTP (Verbindungsaufbau)	**53**	DNS	**194**	IRC
22	SSH Secure Shell	**80**	HTTP	**443**	HTTPS
23	Telnet	**110**	POP3	**993**	IMAPS

Tabelle 3.1 Protokolle und ihre Ports

Die Ports zwischen 1024 und 49.151 nennt man *registrierte Ports*. Bei den Ports mit noch höheren Nummern handelt es sich dann um die *dynamischen Ports*, also um diejenigen Ports, die von Clients für die Anfrage an einen Server verwendet werden.

Eine vollständige Auflistung aller Ports finden Sie bei Interesse unter *https://de.wikipedia.org/wiki/Liste_der_standardisierten_Ports*.

3.3 IP-Adressen – unsere Anschrift im Internet

Hat man nun einmal die Grundlagen geschaffen, dass Kommunikation funktionieren *kann*, so muss man sich anschließend dem Problem stellen, wie die Kommunikation funktionieren *wird*. Überdenken Sie Kommunikation im realen Leben, so ist völlig klar, dass Sie, wenn Sie mit einer Person – beispielsweise per Brief – kommunizieren möchten, auch deren Postadresse kennen müssen. Das gilt für beide Seiten:

1. Sie kennen die Postadresse einer Person und sind demnach in der Lage, an diese Postadresse einen Brief zu versenden.
2. Haben Sie auf dem Brief die eigene Adresse vermerkt (oder kennt der Empfänger Ihre Adresse), so kann der Empfänger auf Ihren Brief antworten.

(In etwa) so muss man sich auch die Kommunikation über Netzwerke vorstellen, wobei die »Adresse« in diesem Fall die sogenannte *IP-Adresse* ist. Sie basiert auf dem Internet Protocol (IP), wie in Abschnitt 3.2 beschrieben.

Um die gewünschte Kommunikation zwischen Geräten zu ermöglichen, muss einem jeden an der Kommunikation beteiligten Gerät eine eindeutige IP-Adresse zugewiesen werden. Ganz korrekt ist das nicht, da eine Kommunikation auch über »vermittelnde« Systeme wie beispielsweise private Netze, NAT (*Network Address Translation*) oder ähnliche möglich wäre. Da dieser einleitende Teil jedoch nicht zu technisch werden soll, verzichte ich (zunächst) auf eine Beschreibung von »Ausnahmen von der Regel«.

IP-Adressen benötigen einen vorgegebenen Aufbau, wie dies auch im Falle der Post-Adressen notwendig ist. Je nach Version des Internet-Protokolls (v4 oder v6) können unterschiedlich viele (eindeutige) Adressen abgedeckt werden.

3.3.1 IPv4 – sind 4,3 Milliarden Adressen zu wenig?

Die vierte Version des Internet-Protokolls wurde im Jahr 1981 eingeführt und war die erste wirklich verbreitete Version dieses Protokolls. Im Rahmen von IPv4 werden IP-Adressen als 32-Bit-Zahlen in vier 8-Bit-Blöcken (2^8 = 256 Möglichkeiten, also etwa Zahlen von 0–255) dargestellt:

0–255.0–255.0–255.0–255, also etwa 213.97.199.142

Hieraus ergeben sich 2^{32} = 4,3 Milliarden unterschiedliche IP-Adressen. Diese Zahl reduziert sich auf ca. 3,7 Milliarden Adressen, da manche Adressbereiche für andere Zwecke reserviert wurden und deshalb nicht zur Verfügung stehen:

- private Netze: 10.0.0.0/8, 172.16.0.0/12, 192.168.0.0/16
- Testnetze: 192.0.2.0/24, 198.51.100.0/24, 203.0.113.0/24
- andere Verwendungen

Subnetze

Vielleicht ist Ihnen bei der Angabe der IP-Adressen der obigen Netze aufgefallen, dass am Ende der Adresse ein Schrägstrich und eine Zahl folgen. Dies ist die Angabe eines *Subnetzes*, das verwendet wird, um Netzwerke zu strukturieren, zu segmentieren oder auch um Adressräume einzusparen. Wenn Sie darüber mehr wissen möchten, können Sie hier weiterlesen: *https://de.wikipedia.org/wiki/Subnetz*.

Die Vergabe der IP-Adressen an die Länder der Welt erfolgt zunächst einmal durch die IANA (*Internet Assigned Numbers Authority*), die eine Abteilung der ICANN (*Internet Corporation for Assigned Names and Numbers*) ist. Sie verteilt die zur Verfügung stehenden IP-Adressen an die RIRs (*Regional Internet Registries*), die eine regionale Weiterverteilung übernehmen. Das Wort »regional« ist hierbei etwas irreführend, da sich »regional« in diesem Fall auf ganze Kontinente bezieht (siehe Abbildung 3.2).

Die RIRs erhalten von der IANA IP-Adressblöcke zugewiesen, die diese an ihre lokalen Vertreiber (die LIRs – *Local Internet Registries*; in der Regel sind dies die Internet Service Provider) weitergeben. Diese LIRs bedienen den Endkunden.

Die ursprüngliche Intention war, dass die IP-Adressen in /8-Blöcken (CIDR-Notation; siehe etwa *https://de.wikipedia.org/wiki/Classless_Inter-Domain_Routing*) verteilt werden, die den RIRs und somit den Regionen zugeordnet sind, wobei solche /8-Blöcke 2^{24} = 16,8 Millionen Adressen beinhalten.

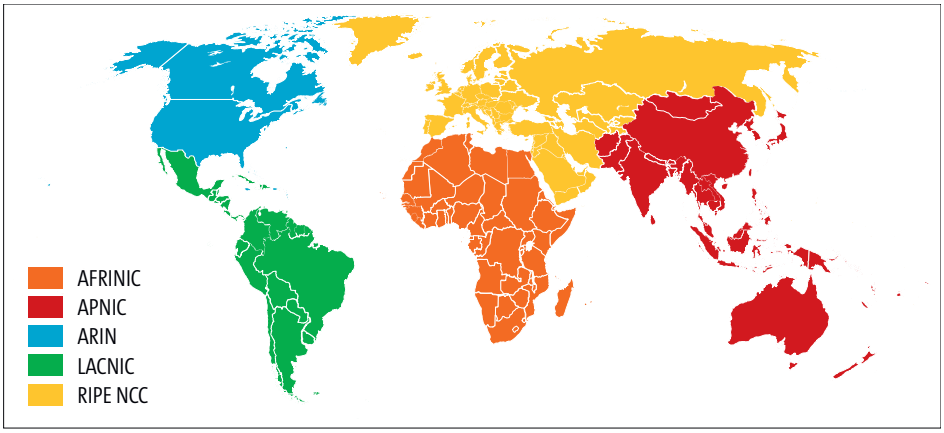


Abbildung 3.2 Die regionalen Verwaltungsbereiche der IP-Adressen.
(Quelle: Von Rir.gif: DorkBlankMap-World6,_compact.svg: Canuckguy et al.derivative work: Sémhur (talk) - Rir.gifBlankMap-World6,_compact.svg, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=5810575)

Da jedoch der Bedarf an IP-Adressen je RIR immer weiter wuchs, kann heute aus einer IP-Adresse nicht mehr geschlossen werden, aus welcher Region sie stammt. Man spricht von einer »Fragmentierung der IP-Adressbereiche«, die technisch wiederum einige Probleme aufwirft (lange Routingtabellen).

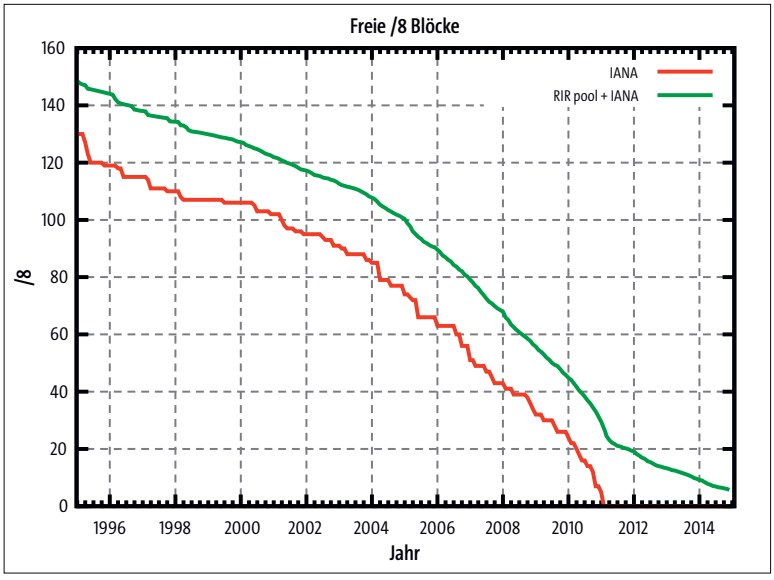


Abbildung 3.3 Freie IP-Adressen.
(Quelle: Von Mro - Based on data from http://www.iana.org/assignments/ipv4-address-space/ipv4-address-space.xml and http://bgp.potaroo.net/ipv4-stats/prefixes.txt, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=10593349)

Bereits Anfang der 1990er-Jahre war klar, dass diese Anzahl an Adressen zukünftig nicht mehr ausreichen wird. (Dies würde ja bedeuten, dass maximal 4,3 Milliarden Geräte im Internet aktiv sein dürften, was angesichts der Weltpopulation und der pro Person verwendeten Geräte eindeutig zu wenig wäre.) Deshalb wurde in mehreren Zwischenvarianten schlussendlich zu IPv6 gewechselt. Das Problem bei einem derartigen Wechsel von einem System zu einem anderen liegt jedoch leider weniger in der softwaretechnischen Umsetzung (die relativ einfach um- und einsetzbar wäre), als vielmehr in der für den Datenverkehr notwendigen Hardware. Die letzten /8-Blöcke wurden im September 2012 vergeben (vgl. auch Abbildung 3.3) – seitdem werden nur noch kleine /24-Blöcke, die aufgrund von Rückgaben an die IANA verfügbar sind, in Form von Wartelistenplätzen an die RIRs vergeben.

3.3.2 IPv6 – mehr als ausreichend viele Adressen

Um der stetig steigenden Nachfrage an IP-Adressen gerecht zu werden, wurde bereits im Jahr 1988 die sechste Version des Internet-Protokolls veröffentlicht. Obwohl die technische Realisierung ohne Probleme möglich wäre, verbreitet sich IPv6 nur zögerlich. Einer der Hauptgründe hierfür ist die Angst vor einem Verlust der Anonymität: Mit IPv4 ist es aufgrund der Adressknappheit nicht möglich, einem Gerät eines Users eine mehr oder weniger »statische« (also immer gleiche) IP-Adresse zuzuweisen; IPv6-Adressen sind von Haus aus jedoch statisch. Die Vorteile von Ipv6 gegenüber IPv4 sind hingegen mehr als deutlich:

- **Ungleich mehr IP-Adressen**: Anstatt von $2^{32} = 4{,}3 \cdot 10^9$ Adressen käme man nun auf $2^{128} = 3{,}4 \cdot 10^{38}$ Adressen, was eine Vergrößerung um den Faktor $7{,}9 \cdot 10^{28}$ bedeutet.
- **Vereinfachter/verbesserter Protokollrahmen**: Dies bringt eine Entlastung der Router in Hinblick auf deren Rechenaufwand.
- **Implementierung von IPSec**: Dadurch wird eine Verschlüsselung von IP-Paketen möglich.
- **Unterstützung weiterer Netzwerktechniken** (Multicast etc.)
- etc.

Allein die Erhöhung der Anzahl von IP-Adressen würde eine wesentliche Erleichterung bilden: Wird einer Privatperson heutzutage in der Regel nur eine IP-Adresse zugewiesen, die dann mittels Network Address Translation in ein Subnetz übergeführt wird, um mehreren Geräten den Zugang zum Internet zu ermöglichen, so wäre mit IPv6 der gleichzeitige Betrieb vieler Geräte/Dienste möglich. Auch das *Internet of Things* (IoT) würde damit ein Stück näher in Richtung einer sinnvollen und durchgängigen Nutzung gerückt.

Da im Rahmen von IPv6 sehr viel mehr Adressen abgedeckt sind, erweist sich die Schreibweise in dezimaler Notation als nicht praktikabel (die Adressen würden ein-

fach zu lang werden). Aus diesem Grund greift man auf eine hexadezimale Notation zurück, bei der die IP-Adresse in acht Blöcke á 16 Bit (vier hexadezimale Stellen) unterteilt wird und die einzelnen Blöcke durch Doppelpunkte voneinander getrennt werden. Das sieht dann so aus:

2001:0db8:85a3:08d3:1319:8a2e:0370:7344

Es existieren einige Regeln für abgekürzte Schreibweisen, die Sie beispielsweise unter *https://de.wikipedia.org/wiki/IPv6* nachschlagen können. Für unser Thema sind diese Regeln aber nicht von Belang.

3.3.3 Internet Service Provider sind das Tor zur Internetwelt

Üblicherweise werden die IP-Adressen von *Internet Service Providern* (*ISPs*; oft auch nur Provider genannt) an deren Endkunden weitergegeben. Jeder Haushalt (Kunde) erhält hierdurch eine eigene IP-Adresse, auf die das vom ISP bereitgestellte Modem konfiguriert wird. Neben dem Modem erhält ein Kunde auch einen Router, über den das Bilden eines Subnetzes möglich wird (Modem und Router sind üblicherweise in einem Gerät vereint). Dies wiederum ermöglicht, dass ein Haushalt mehrere Geräte im Internet nutzen kann, wobei der Router synchron eine Anfrage eines Gerätes nach dem anderen durchführt und somit das Nadelöhr zum Internet darstellt.

Je nach Größe des Providers unterscheidet man drei sogenannte *Tiers* (dt. »Ränge«):

- **Tier-1-Provider**: große autonome Netze, die keinen Transit (siehe »Tier-2-Provider«) kaufen, sondern nur mit anderen Tier-1-Providern kooperieren (»peeren«). Beispiele sind: Deutsche Telekom, AT&T und Lumen (USA) etc.
- **Tier-2-Provider** (»Transit-Provider«): Zu diesen zählen die meisten autonomen Systeme im Internet. Sie kaufen Konnektivität von Tier-1-Providern und verkaufen Konnektivität an Tier-3-Provider. Gleichzeitig peeren solche Provider mit anderen Tier-2-Providern. In seltenen Fällen verkaufen Tier-2-Provider Konnektivität direkt an den Endkunden.
- **Tier-3-Provider**: Provider dieser Klasse kaufen Konnektivität von Tier-2-Providern, peeren mit anderen Tier-3-Providern und verkaufen Konnektivität an Endkunden.

Die Kosten für den Endkunden werden an mehreren Faktoren gemessen (die angegebenen Werte beziehen sich auf den aktuellen Stand von 2023):

- **Downstream-Geschwindigkeit**: Sie bestimmt, wie schnell Daten aus dem Internet geladen werden können. Aktuelle Raten sind durchschnittlich bei 30 bis 300 MBit/s (Megabit pro Sekunde).
- **Upstream-Geschwindigkeit**: Sie bestimmt, wie schnell Daten in das Internet (hoch-)geladen werden können. Dieser Wert ist üblicherweise geringer als der

Downstream-Wert, da der Upload für die meisten Kunden weniger wichtig ist. Übliche Werte liegen bei 10 bis 100 MBit/s.

- **Synchrone oder asynchrone Anbindung**: »Synchron« bedeutet, dass die Upstream- und die Downstream-Geschwindigkeit identisch sind; »asynchron« bedeutet sinngemäß das Gegenteil.
- **Statische oder dynamische IP-Adresse**: Eine statische IP-Adresse bedeutet, dass ein Kunde eine eindeutige IP-Adresse für sich reserviert, die sich anschließend nicht mehr ändert. Da ein Provider jedoch nur über einen eingeschränkten IP-Adressbereich verfügt, ist eine statische Zuweisung für ihn nachteilig: Eine dynamische IP-Adresse könnte er einem Kunden B zuweisen, wenn Kunde A diese Adresse gerade nicht benötigt. Aus diesem Grund sind statische IP-Adressen teurer.

Die üblichen Anbindungskosten über einen ISP liegen bei etwa 40 EUR für eine Kupfer- und 60 EUR für eine Lichtleiteranbindung. Konkret spricht man hier von der *Last Mile*, also von dem letzten Stück von der Verteilerstelle des ISP zum Haushalt. Je neuer/jünger die Anbindung ist, umso eher erfolgt eine Anbindung über Lichtwellenleiter (LWL).

[i]

Lichtwellenleiter, Glasfaser? Was ist gemeint?

Es gibt viele Synonyme für einen Lichtleiter: *Lichtwellenleiter*, *Glasfaserkabel*, *optisches Kabel*. Lassen Sie sich davon nicht beirren: Es ist immer dasselbe gemeint.

Glasfaser oder Kupfer?

Jede Datenkommunikation erfolgt über Leitungen und/oder Funkverbindungen. Da selbst die Lichtgeschwindigkeit in Lichtwellenleitern nicht unendlich schnell ist, benötigt jedes gesendete Datenpaket eine gewisse Zeit für die Übertragung. In der Regel hat man es aber gar nicht ausschließlich mit Lichtwellenleitern zu tun, sondern sehr oft mit einfachen Kupferkabeln.

Dennoch muss man hier die Kirche im Dorf lassen, denn die Ausbreitungsgeschwindigkeit allein macht nicht die Leistung der Verbindung aus. Würden Sie die Geschwindigkeiten nämlich vergleichen, so würden Sie rasch feststellen, dass die Ausbreitungsgeschwindigkeit einer elektromagnetischen Welle in einem Kupferkabel 230.000 km/s beträgt, wohingegen die Lichtgeschwindigkeit in einem Glasfaserkabel (wo die Signale nach einer Art Morse-Prinzip übertragen werden) nur bei ca. 200.000 km/s liegt (im Vakuum wären es 300.000 km/s). Der Vorteil der Glasfaser liegt vielmehr in folgenden Punkten:

1. Die Bandbreite von Glasfaserkabeln ist aufgrund der Lichtmodulation (Licht an, Licht aus: Morse-Prinzip) wesentlich höher als bei Kupferkabeln: Die Lichtmodula-

tion in Glasfaserleitungen erfolgt bei wesentlich höheren Frequenzen (ca. 60 THz = 60 Billionen Schwingungen pro Sekunde) als die elektromagnetische Modulation in Kupferkabeln (bei G.Fast ca. 212 MHz = 212 Millionen Schwingungen pro Sekunde).

2. Ein elektromagnetisches Signal wird mit der Entfernung (also der Länge einer Leitung) schwächer, weshalb es öfter regeneriert werden muss. Die optische Datenübertragung ist von diesem Problem nur sehr bedingt betroffen.
3. Elektromagnetismus ist von Wechselwirkungen betroffen (parallel liegende Stromkabel, Magnete etc.), die optische Datenübertragung nicht.

Das Hauptproblem im Falle von Kupferkabeln (oder anderen Kabeln) liegt somit nicht in der Übertragungsgeschwindigkeit, sondern in der Fehlerrate, die sich mit der Kabellänge erhöht. Aus diesem Grund kann man wie folgt zusammenfassen:

- Kupferleitungen (oder ähnliche Leitungen) sind für kurze Distanzen sehr gut geeignet: Aus diesem Grund ist die Last Mile oftmals als Kupferleitung ausgeführt.
- Glasfaserleitungen sind für lange Distanzen geeignet: für Leitungen, die über Ozeane geführt werden oder Knotenpunkte miteinander verbinden.

3.4 Domains

Wie ich in den vorangegangenen Abschnitten beschrieben habe, wäre eine Kommunikation zwischen zwei Geräten rein über die IP-Adressen möglich. Dies bedeutet jedoch auch, dass die IP-Adressen bekannt sein müssen. Haben wir jedoch die Situation, dass ein User mithilfe seines Webclients (Browsers) einen Webserver kontaktieren muss, um die gewünschte Website zu sehen, ist das ebenfalls nichts anderes als eine Kommunikation zwischen zwei Geräten: zwischen dem Gerät des Users, auf dem der Browser installiert ist, und dem Gerät des Providers, auf dem die Webserver-Software des Website-Betreibers installiert ist.

Natürlich ist uns allen bekannt, dass wir in unserer Rolle als User wirklich sehr, sehr selten (bis wahrscheinlich nie) eine Website über ihre zugehörige IP-Adresse aufgerufen haben. Vielmehr bedienen wir uns einer *Domain*, die wir in unseren Browser eingeben, um die betreffende Website öffnen zu können. Dies bedingt jedoch, dass die Domain, die ein User eingibt, auf irgendeine Art mit der IP-Adresse des zugehörigen Servers in Verbindung steht.

Dies ist in der Tat der Fall, und die entsprechende Namensauflösung werden wir uns in den folgenden Abschnitten noch genauer ansehen. Es wäre in Wahrheit auch nicht praktikabel, müsste ein User die IP-Adresse eines Servers kennen. Wie Sie bereits aus den vorangegangenen Abschnitten wissen, ist bereits eine IPv4-Adresse schwer zu

merken. Müssten Sie sich eine IPv6-Adresse merken, so wäre das wohl ein unmögliches Unterfangen.

Die vorerst für unser Verständnis wichtigeren Fragen lauten: Wie sind Domains aufgebaut, welche Spielarten von Domains existieren und worauf müssen Sie achten?

Eine Domain ist hierarchisch aufgebaut, wobei die einzelnen Teile des Aufbaues durch Punkte voneinander getrennt sind. Des Weiteren wird eine Domain von rechts nach links gelesen: Der am weitesten rechts stehende Teil stellt die oberste Hierarchie dar usw. Betrachten wir das Beispiel der Domain *www.orf.at*:

- Die oberste Hierarchie stellt die Angabe **at** dar – sie wird mit dem Begriff *Top-Level Domain* beschrieben.
- Danach folgt der eigentliche Domainname **orf**, der als *Second-Level Domain* bezeichnet wird.
- Der Name des Dienstes **www** stellt in diesem Beispiel eine freiwillige Ebene dar, die als *Third-Level Domain* bezeichnet wird.

Zusammen bilden die einzelnen Teile einer Domain den *Fully Qualified Domain Name* (FQDN), wobei gilt:

- Jeder Abschnitt des FQDN darf maximal 63 Zeichen lang sein.
- Die Gesamtlänge des FQDN liegt bei 255 Zeichen.

Gemeinhin ist eine Unterteilung der Domain in drei Teile nicht notwendig – lediglich die ersten beiden Teile sind von Bedeutung. In der Praxis verwendet man die Third-Level Domain gern als sogenannte *Sub-Domain*, um beispielsweise unterschiedliche Dienste, die über dieselbe Domain laufen sollen, zu unterteilen:

- Webserver: **www**.orf.at
- Mailserver: **mail**.orf.at
- FTP-Server: **ftp**.orf.at

Das Root-Label

Streng genommen ist die obige Schreibweise nicht vollständig korrekt, da vor einer Top-Level Domain noch das *Root-Label* angegeben wird. Da jedoch das Root-Label immer leer ist, würde sich folgender FQDN ergeben:

www.orf.at

Man beachte den Punkt am Ende der Domain, der das Root-Label von der Top-Level Domain trennt.

3.4.1 Top-Level Domain (TLD) – der Ausgangspunkt

Die Top-Level Domains sind in zwei Hauptgruppen und einen Sonderfall unterteilt:

- Generische TLDs, die wiederum in gesponserte, nicht gesponserte und allgemeine TLDs unterteilt werden
- länderspezifische TLDs
- Sonderfall: Infrastruktur-TLDs wie .arpa und .root

Generische Top-Level Domains

Wie oben erwähnt werden die generischen Top-Level Domains in zwei Untergruppen unterteilt, wobei die (sehr viel wichtigeren) ungesponserten TLDs (*uTLD*) von der ICANN und der Internet Society, die gesponserten TLDs (*sTLD*) von unabhängigen Organisationen verwaltet werden. Letztere sind in der Lage, eigene Vergaberichtlinien festzulegen: Beispielsweise wird die TLD **mil** vom amerikanischen Militär verwendet und ist somit nicht öffentlich zugänglich/registrierbar. Tabelle 3.2 zeigt Beispiele.

uTLD			sTLD (Auswahl)		
Kürzel	**Bedeutung**	**Zweck**	**Kürzel**	**Bedeutung**	**Zweck**
biz	business	ausschließlich für kommerzielle Verwendung, de facto jedoch frei zugänglich	.edu	educational	seit 2001 auf Bildungseinrichtungen beschränkt (fast ausschließlich US-amerikanische Universitäten und Colleges)
.com	commercial	ursprünglich für Unternehmen erdacht, seit längerer Zeit jedoch frei zugänglich	.gov	government	Regierungsorgane der USA
.info	information	ursprünglich für Informationsanbieter gedacht, jedoch seit jeher frei zugänglich	.mil	military	militärische Einrichtungen der USA

Tabelle 3.2 Gesponserte Domains sind nicht wenige, werden aber dennoch in ihrer Anzahl von den nicht gesponserten Domains übertroffen.

uTLD			sTLD (Auswahl)		
Kürzel	**Bedeutung**	**Zweck**	**Kürzel**	**Bedeutung**	**Zweck**
.name	name	für natürliche Personen oder Familien, de facto jedoch frei zugänglich	.xxx	sex	erotische und sexuelle Inhalte
.net	network	für Netzverwaltungs-Einrichtungen, jedoch mittlerweile frei zugänglich			
.org	organization	für nichtkommerzielle Organisationen (Non-Profit Organizations), jedoch seit 2003 frei zugänglich			
.pro	professionals	für »qualifizierte Fachkräfte« erdacht			

Tabelle 3.2 Gesponserte Domains sind nicht wenige, werden aber dennoch in ihrer Anzahl von den nicht gesponserten Domains übertroffen. (Forts.)

Neben den oben dargestellten TLDs existiert seit 2012 ein Vergabeverfahren, in dem Anträge auf neue TLDs eingebracht werden können. Etwa alle zwei Jahre wird auf Basis dieser Anträge entschieden, welche neuen TLDs zugänglich gemacht werden. Dies können ebenso Städtenamen (.berlin) wie auch Berufs- oder Spartenbezeichnungen (.radio) oder allgemeine Ausdrücke (.rocks, .sports, .io) sein. Das Ziel hierbei ist es, ein breiteres Spektrum an TLDs anbieten zu können.

Länderspezifische Top-Level Domains

Es existieren etwa 200 länderspezifische TLDs, wobei diese in zweistelliger Notation laut dem ISO-Standard 3166 ausgeführt sind. Von dieser Regel existieren einige wenige Ausnahmen wie etwa die TLD von Großbritannien (.uk anstatt .gb) oder auch die TLD der EU (die bekanntermaßen kein Staat, sondern ein Staatenbund ist). Bekannte Beispiele wären etwa .at, .de, .it, .es usw.

Jeder Staat hat das Recht, eigene Vergaberichtlinien für die ihm zugewiesene TLD zu treffen. Zumeist entscheidet sich der Staat dafür, die Nutzung der TLD den eigenen Bürgern wie auch den ansässigen Unternehmen und Institutionen zur Verfügung zu stellen. Auch ist es einem jeden Staat selbst überlassen, ob die Nutzung der TLD auch auswärtigen Personen oder Unternehmen ermöglicht wird oder nicht. Beispielsweise war die Nutzung der TLD von Frankreich (.fr) bis 2011 nur denjenigen Personen oder Unternehmen vorbehalten, die einen Wohn- oder Firmensitz (also eine Postadresse) in Frankreich besaßen. Seit 2011 dürfen alle in der EU ansässigen Personen und Unternehmen eine solche TLD beantragen.

Es gibt zahlreiche Beispiele für die unterschiedlichen Herangehensweisen von Staaten. Dies geht so weit, als dass sich manche Staaten entschlossen, die eigene TLD zu verkaufen. Das Paradebeispiel stellt die TLD .tv dar, die eigentlich dem Zwergstaat Tuvalu gehört. Wie Sie sich leicht vorstellen können, ist diese TLD jedoch gerade bei Televisionsunternehmen sehr begehrt. Diese Tatsache machte sich Tuvalu zunutze und vermarktet die eigene Domain an Fernsehsender. Die Gewinne waren beträchtlich: Tuvalo nahm auf diese Weise mehr als 50 Millionen Dollar ein, was es ihm ermöglichte, einerseits IT-Infrastruktur für die wichtigsten staatlichen Einrichtungen zu beschaffen und andererseits die Aufnahmegebühr für die Vereinten Nationen zu begleichen.

Die zweite Namensebene

Zusätzlich zu den Länderkennzeichnungen existieren noch weitere Unterteilungen auf zweiter Namensebene, die eine genauere Charakterisierung der Nutzung ermöglichen. Beispiele sind:

- .ac.at (akademische Nutzung in Österreich),
- .co.at (kommerzielle Nutzung in Österreich),
- .gv.at (behördliche Nutzung in Österreich),
- .or.at (gemeinnützige Organisationen in Österreich) usw.

3.4.2 Second-Level Domain (SLD) – die Qual der Wahl

Die Second-Level Domain stellt die »eigentliche Domain« dar und obliegt (mit gewissen Einschränkungen) dem Betreiber selbst, wobei die Restriktionen hinsichtlich der Namenswahl gering sind:

- **mögliche Buchstaben, Ziffern und Sonderzeichen**: Beispielsweise sind Umlaute bei gewissen länderspezifischen Domains erlaubt, bei generischen Domains jedoch nicht.

- **minimale und maximale Länge**: Beispielsweise musste eine .de-Domain lange Zeit aus mindestens drei Zeichen bestehen, wobei das erste Zeichen noch dazu ein Buchstabe sein musste.
- **Rechte Dritter, Verletzung von Markenrechten**: Achten Sie bei der Domainauswahl darauf, keine Rechte zu verletzen! Das kann unter Umständen teuer werden.

Im Allgemeinen ist man als potenzieller Betreiber einer Domain dennoch sehr frei in der Wahl ihrer Bezeichnung – das wesentlich größere Problem stellt die Verfügbarkeit einer Domain dar. Gerade diese eingeschränkte Vielfalt war einer der Gründe, warum sich die ICANN 2012 entschied, alle zwei Jahre neue allgemeine generische TLDs zu veröffentlichen.

3.4.3 Um's Eck zum Kiosk: Vergabestellen von Domains

Haben Sie sich für eine Top- und Second-Level Domain in Kombination entschieden und sichergestellt, dass diese noch verfügbar ist, so ist der nächste Schritt die Registrierung dieser Domain. Hierzu existieren unterschiedliche Vergabestellen:

- **Betreiber der TLD, offizielle Vergabestelle** (»Registry«): Dies sind beispielsweise die Verwalter länderspezifischer TLDs, die üblicherweise in einem jeden Land existieren (Österreich: *https://nic.at* für .at, .or.at und .co.at; Deutschland: *https://denic.de*).
- **Provider**:
  - **Registrare**: Dies sind spezielle Kooperationspartner der offiziellen Vergabestellen, denen es erlaubt ist, Domains zu verwalten (Beispiel: domainfactory GmbH, *www.domainfactory.de*).
  - **Reseller**: Wiederverkäufer von Domains; diese arbeiten in der Regel mit Registraren zusammen und treten selbst nicht in Erscheinung (Beispiel: World4You Internet Services GmbH, *www.world4you.at*).

[i]

Whois-Dienste

Um zu prüfen, ob eine Wunschdomain noch frei ist, bedient man sich in der Regel eines *Whois-Dienstes*, der im Allgemeinen ein auf einem jeden Betriebssystem zugänglicher Dienst ist und über ein Terminal aufgerufen werden kann (z. B. mittels `whois syne.at`). Viele Anbieter im Internet betreiben eigene Websites, die die Nutzung dieses Dienstes ermöglichen (beispielsweise *https://www.whois.com*).

In den meisten Fällen werden Sie sich an einen Provider wenden, nur in seltenen Fällen an die offizielle Vergabestelle (denn das ist umständlich und teuer). Wesentlich bei der Registrierung einer Domain ist, dass Sie diese nicht kaufen, sondern lediglich

die Nutzungsrechte erwerben. Dies geschieht üblicherweise pro Jahr, weshalb die Kosten für die Domain (konkret: für die *Nutzungsmöglichkeit* der Domain) jährlich mit ca. 15 bis 40 EUR zu Buche schlagen, sofern es sich um eine länderspezifische Domain handelt.

Um eine Domain zu registrieren, müssen drei Personen- oder Unternehmensinformationen zu unterschiedlichen Informationszwecken angegeben werden:

- Inhaber der Domain
- administrative Ansprechperson
- technische Ansprechperson

Selbstverständlich müssen auch Kontaktinformationen bekannt gegeben werden. Diese können jedoch aus Sicht des Datenschutzes mittlerweile vor öffentlichen Aufrufen geschützt (verborgen) werden.

3.4.4 Nameserver und Domain Name Service (DNS) – einmal IP-Adresse und wieder zurück

Um eine Domain nun tatsächlich auch nutzen zu können, muss die Domain in einen *Nameserver* eingetragen werden. Hierbei werden die Domain und die IP-Adressen (sowie die Art der Nutzung) der zuständigen Server in Verbindung gebracht.

Nameserver sind hierarchisch aufgebaut, was bedeutet, dass nicht ein einziger Nameserver existiert, der die Daten sämtlicher Domains und IP-Adressen kennt, sondern dass es ein verzweigtes Netz an Nameservern gibt, die miteinander diese Informationen austauschen.

Man unterscheidet zwischen *autorativen Nameservern* (»Primary Nameserver«), die für eine Zone verantwortlich sind und somit gesicherte Informationen bereitstellen können, und *nichtautorativen Nameservern*, die ihre Informationen wiederum von autorativen Nameservern beziehen. (Die Informationen eines nichtautorativen Nameservers werden als nicht gesichert angesehen, da er diese umgangssprachlich »aus dritter Hand« bezieht).

Bevor ein User eine Domain verwenden und sie über den gewünschten Dienst (etwa www) mithilfe seines Clients nutzen kann, muss zunächst eine *Namensauflösung* erfolgen, mit deren Hilfe die Domain in eine IP-Adresse übersetzt wird. Weiterführende Informationen zur Funktionsweise von Nameservern finden Sie z. B. unter *https://de.wikipedia.org/wiki/Domain_Name_System*.

Bevor eine Domain also überhaupt nutzbar wird, muss ein Betreiber für einen jeden gewünschten Dienst (*www*, *mail* etc.) über einen entsprechenden Provider verfügen, der diesen Dienst (in der Regel kostenpflichtig) zur Verfügung stellt. Gerade dieser

Punkt der Nutzbarmachung einer Domain stellt normalerweise eine nicht unerhebliche technische Hürde dar, weshalb bei der Domain-Registrierung gerne auf einen Provider zurückgegriffen wird, der die notwendigen Hilfsmittel (oder sogar einen Automatismus) bereitstellt; die offizielle Vergabestelle ermöglicht dies in der Regel nicht. Auch geht das benötigte technische Wissen im Normalfall über die Anforderungen an Webentwickler hinaus. Fakt ist dennoch, dass der Betreiber einer Domain (und damit meint man normalerweise den Betreiber einer Website) auf entsprechende Provider zurückgreifen muss:

- auf einen Domain-Provider zur Nutzbarmachung einer Domain
- auf einen Hosting-Provider, um eine Website unter der Domain betreiben zu können
- auf einen Mail-Provider, um über die eigene Domain Mails versenden und empfangen zu können

Aus diesem Grund bedient man sich gerne eines »Universal-Providers«, der neben der Registrierung der Wunsch-Domain (fungierend als Reseller von Domains) eben gerade die Hosting-Dienste für Website und Mails anbieten kann. Beispiele sind die bereits genannte World4You Internet Services GmbH mit Sitz in Linz, die Host Europe GmbH mit Sitz in Köln, der DreamHost aus Brea/USA usw.

3.5 Mime Types und Content Types – Sein und Schein

MIME- bzw. Content-Typen geben an, welchen Inhalt Dokumente besitzen und wie diese von einem darzustellenden System (beispielsweise einem Browser) verarbeitet werden sollen. Entgegen der landläufigen Meinung stellt das Suffix (die Datei-Endung) einer Datei keine ausreichende Information dar. Beispielsweise kann man sich bei einer Datei mit dem Namen *urlaubsbild.jpg* nicht sicher sein, ob es sich tatsächlich um eine JPEG-Datei (also eine Bilddatei) handelt oder ob diese Datei nicht einfach von *virus.exe* in *urlaubsbild.jpg* umbenannt wurde. Aus diesem Grund werden im Inhalt von Dateien üblicherweise die verwendeten Content-Typen festgehalten, sodass ein darzustellendes System das Suffix ignorieren kann.

Der Begriff *MIME* ist eine Abkürzung für *Multipurpose Internet Mail Extension* und steht synonym für den *Internet Media Type* wie auch für den *Content Type* einer Datei. Diese Information steht wie erwähnt im Dateiinhalt und dort insbesondere im Header des (gesendeten) Dokuments. MIME Types deklarieren ebenso den Inhalt in diversen Internetprotokollen (wie etwa HTTP). Beispiele hierfür sind etwa die folgenden Content-Typen für die jeweiligen Dokumente:

- HTML-Dokument: `text/html`
- JPEG-Datei: `image/jpeg`

Es existieren folgende Medientypen, die umgangssprachlich auch als die »Haupttypen« bezeichnet werden (plus mehr als 130 Subtypen):

- `application`: für uninterpretierte binäre Daten, Mischformate (z. B. Textdokumente mit eingebetteten nichttextuellen Daten) oder Informationen, die von einem bestimmten Programm verarbeitet werden sollen
- `text`: für Text
- `video`: für Videomaterial
- `audio`: für Audiodaten
- `image`: für Grafiken
- `multipart`: für mehrteilige Daten

und außerdem (für die Webentwicklung weniger von Interesse):

- `example`: Beispiel-Medientyp für Dokumentationen
- `message`: für Nachrichten, beispielsweise `message/rfc822`
- `model`: für Daten, die mehrdimensionale Strukturen repräsentieren
- `chemical`: für z. B. Moleküle (inoffiziell)

Insbesondere die Medientypen `text` und `application` sind voneinander zu unterscheiden, da `text`-basierte Dokumente Fehler (z. B. HTML-Fehler) enthalten dürfen, `application`-basierte Dokumente jedoch fehlerfrei sein müssen. Auf diese Unterscheidung gehe ich insbesondere in Abschnitt 5.1 »Ein bisschen in der Geschichte von HTML stöbern« noch genauer ein.

Fehlt in einer Datei die Angabe des MIME-Typs, so erfolgt ein sogenanntes *MIME-Sniffing*. Dabei werden die ersten Bytes des Dokuments gelesen und es wird versucht, den Inhalt der Datei zu erkennen. Bilder im Format JPEG oder PNG sind beispielsweise relativ einfach an ihrem Header zu unterscheiden. Schlägt die Analyse fehl, wird die Dateiendung herangezogen. Danach wird entweder eine Standarddarstellung versucht (beispielsweise die Wiedergabe als HTML-Datei) oder die Wiedergabe wird mit einer Fehlermeldung abgebrochen. Die Reihenfolge der Analyse wird in einer Protokolldokumention festgelegt. So fordert beispielsweise das HTTP 1.1-Protokoll, dass immer der angegebene Content-Typ zu verwenden ist. Nur wenn er fehlt, darf MIME-Sniffing den Typ erraten.

3.6 Ein paar notwendige Begriffsdefinitionen

Bereits in den ersten Kapiteln habe ich immer wieder von einer »Website« gesprochen. Sie haben sich vielleicht gefragt, ob ich damit dasselbe meine wie mit den Begriffen »Webseite« oder »Homepage«. Einigen wir uns auf folgenden Sprachgebrauch:

- **Website**: Unter einer Website versteht man das »Gesamte«, also sämtliche Dateien. Üblicherweise spricht man von *www.orf.at* als von einer Website und meint damit, dass ein Aufruf in einem Browser (formatierten) Inhalt samt weiterer Medien darstellt und dass man auch mittels Klick (oder Ähnlichem) andere Inhalte (Webseiten – siehe den folgenden Punkt) aufrufen kann. Dafür existiert auch der Begriff *Internetpräsenz*.
- **Webseite (Web Page)**: Unter einer Webseite verstehen wir ein einzelnes Inhaltsdokument, das HTML-Code beinhaltet. Dieses Dokument verweist mitunter auf weitere Inhalte wie Bilder, Videos etc., die jedoch nicht Teil des Dokuments sind. Beispiele sind die unterschiedlichen Seiten einer Website, die man (üblicherweise durch Klicks auf die Navigation) aufrufen kann. Besitzt eine Website etwa eine Navigation mit den Navigationspunkten *Home* | *Über uns* | *Referenzen*, so wären die Seiten, die sich nach einem Klick auf diese Punkte öffnen, die Webseiten oder *Web Pages*. Ein Beispiel für eine Webseite des ORF wäre etwa *www.orf.at/stories/1234567/index.html*.
- **Homepage**: Die Homepage ist eine ganz besondere Webseite (Web Page), nämlich die Startseite der Website. Sie stellt üblicherweise den Einstiegspunkt in die Website dar und besitzt deshalb eine sehr wesentliche Aufgabe, soll sie dem User doch vermitteln, dass er auf der gewünschten Seite gelandet ist. Die Homepage des ORF ist beispielsweise unter *www.orf.at/index.html* zu finden.

Sehr oft wird im täglichen Sprachgebrauch der Begriff »Homepage« synonym für »Website« verwendet. Wie Sie jedoch gesehen haben, ist dies nicht ganz korrekt, und wir sollten uns bemühen, diese Begriffe sauber zu trennen.

3.7 Wie gelangen unsere Daten auf den Server? FTP macht's vor

Um eine Website schlussendlich funktional zu machen, müssen Sie natürlich den Grundstock der Website in Form einzelner Dateien liefern: Um diese Punkte kümmern wir uns in allen folgenden Kapiteln. Sind diese Dateien einmal entwickelt, müssen sie ihren Weg zum Webhosting-Dienst des gewählten Hosting-Providers finden.

Dies erfolgt üblicherweise, indem Sie die Dateien mittels FTP (*File Transfer Protocol*) auf den Webserver in ein dafür bereitgestelltes Verzeichnis hochladen. Um dies zu ermöglichen, müssen Sie die Host-Adresse und das Host-Verzeichnis kennen sowie die für einen Login notwendigen Zugangsdaten besitzen. (Details hierzu finden Sie in Abschnitt 9 »Veröffentlichung und Versionierung«. Sind die Dateien einmal bereitgestellt und ist der Webserver (korrekt) mittels eines Nameserver-Eintrags verewigt, so steht dem Betrieb der Website nichts mehr im Wege.

3.8 Wie ein Webserver funktioniert

Bevor wir uns nun an die Entwicklung einer Website machen, muss zuvor noch klar sein, was die Funktionsweise und Aufgabe eines Webservers ist:

Ein Webserver ist eine Software, die aus Sicht des Users nicht mehr erledigt, als eine Anfrage (Request) nach einem konkreten Dokument (Datei) entgegenzunehmen, diese Anfrage abzuarbeiten und den Inhalt des gewünschten Dokuments als seine Antwort (Response) an den Anfragenden (Client) zurückzuschicken.

In dieser Aussage stecken mehrere kleine Stolpersteine:

1. »Anfrage nach einem *konkreten* Dokument«: Dies bedeutet, dass ein Webserver nur Anfragen verarbeiten kann, die sich auf Dokumente beziehen. Dies steht zunächst in einem Widerspruch zu einer Anfrage wie *https://www.orf.at*, denn hierin ist keine Angabe eines Dateinamens (Dokuments) zu sehen. Eine Anfrage nach einem Dokument wäre beispielsweise *https://www.orf.at/abc.html*, denn bei *abc.html* handelt es sich um eine Datei.
2. »den *Inhalt* des gewünschten Dokuments als seine Antwort«: Sehr wesentlich ist, dass ein Webserver keine Dateien versendet, sondern lediglich den (gegebenenfalls durch serverseitige Programmierung generierten) Inhalt einer Datei zurückliefert. Dies wird weitreichende Konsequenzen für das Kapitel über serverseitige Programmierung (siehe Abschnitt 11, »Die Serverseite oder auch das Backend – Programmieren mit PHP«) haben.

Bleibt jedoch zwingend der erste Punkt zu klären, denn als User wissen wir ja, dass eine Anfrage an einen Webserver via *https://www.orf.at* korrekt ist, obwohl wir nicht nach einem konkrete Dokument gefragt haben. Was genau passiert aufseiten des Servers, dass die Anfrage dennoch klappt? Die Antwort ist folgende:

1. Der Webserver erhält zunächst keine Anfrage nach einem Dokument, und er könnte diese sofort ablehnen (was er in der Regel jedoch nicht tut).
2. Der Webserver geht nun in seiner Konfiguration auf die Suche nach dem »Standarddokument«, das er immer dann aufrufen würde, wenn keine Anfrage nach einem speziellem Dokument erfolgt ist. Üblicherweise sind die Standarddokumente als *index.html*, *index.php* oder so ähnlich benannt; wir als User kennen sie auch unter der Bezeichnung »Homepage« – eine *Homepage* stellt die Startseite einer Website dar.

Die Namensgebung des Startdokuments

Grundsätzlich dürfen wir davon ausgehen, dass der Dokumentname *index.xxx* lautet, wobei das xxx für viele mögliche Endungen stehen darf. Was aber, wenn mehrere *index*-Dateien existieren: eine *index.html*, eine *index.php*, eine *index.aspx* usw.? Ganz

allgemein ist das erlaubt, und es kann sich hierbei sogar um eine größere Liste an unterschiedlichen Dokumenten handeln. Dabei besitzen die Dokumente eine Reihenfolge, die in der Konfigurationsdatei des Webservers abgelegt ist: Das erste Dokument dieser Liste, das der Webserver findet, wird verwendet.

3. Alsdann geht der Webserver auf die Suche nach einem dieser genannten Standarddokumente in demjenigen Verzeichnis, das ihm »übermittelt« wurde (im Fall von *https://www.orf.at* ist dies das Hauptverzeichnis). Wird er bei seiner Suche fündig, so verarbeitet er nun dieses Dokument und liefert den Inhalt dieses Dokuments an den Aufrufer zurück.
4. Sollte der Webserver bei seiner Suche nach einem Standarddokument im gegebenen Verzeichnis jedoch nicht fündig werden, so hat er wiederum mehrere Möglichkeiten, darauf zu reagieren:
   - Er lehnt die Anfrage final ab und liefert dem Client eine Fehlermeldung (typischerweise den Fehler 403: »Forbidden«) zurück.
   - Er listet den Inhalt desjenigen Verzeichnisses auf, das angefragt wurde. Dies nennt man ein *Directory Listing*. Diese Variante trifft man auf (professionellen) Websites normalerweise nicht an, da dies einen sogenannten *Exploit* bedeutet: Es werden dem User Informationen dargestellt, die nicht für ihn bestimmt sind (Thema Sicherheitslücke). Directory Listings werden jedoch im Falle lokaler Installationen von Entwicklungsumgebungen dargestellt, da es dort sinnvoll ist.

3.8.1 Die Qual der Wahl: Apache vs. NGINX

Zunächst einmal erscheint es aus Entwicklersicht irrelevant, mit welcher Art von Serversoftware man es zu tun hat, solange alles wie gewünscht funktioniert – bei einem Textverarbeitungsprogramm macht man sich ja auch wenig Gedanken, wo das eine Programm besser oder schlechter ist als das andere. Oder doch nicht? Die Antwort hängt oft davon ab, wie »tief« man in die Materie eindringen möchte oder wie hoch die Anforderungen (an eine Serversoftware) sind.

Betrachtet man die historische Entwicklung von Webservern, so stellten Webserver in den Anfangsjahren ausschließlich statische Informationen bereit, wobei man unter »statischen Dateien« solche versteht, in denen der bereitzustellende Inhalt (beispielsweise der eines HTML-Dokuments) bereits fix fertig vorliegt. Im Falle von Websites wären dies Webseiten, die ein Webdesigner bereits vollständig mit HTML-Code befüllt hat und in denen keine (dynamischen) Änderungen am Inhalt mehr vorgenommen werden.

Im Gegensatz zu statischen Dokumenten stehen dynamische Dokumente. Bei diesen Dokumenten steht der bereitzustellende Inhalt noch nicht fest, sondern muss erst er-

zeugt werden. Dies trifft beispielsweise immer dann zu, wenn Daten, die auf einer Webseite dargestellt werden sollen, aus einer Datenbank gelesen werden müssen. Klassische Vertreter solcher Websites sind Onlineshops, soziale Plattformen, Content-Management-Systeme oder eben all jene Websites, die den Inhalt entsprechend den Anforderungen der User »zusammenstellen« wollen bzw. müssen.

Dynamische Websites mit CGI-Scripts

Die dynamischen Dokumente prägen mittlerweile die weitaus größere Anzahl der Websites, denn fast keine Website arbeitet mehr mit statischen Inhalten. Webserver sind aber von sich aus nicht in der Lage, dynamische Inhalte zu generieren, denn die wesentliche Aufgabe eines Webservers ist lediglich das Entgegennehmen und Abarbeiten eines Requests und das Generieren eines Response. Das alles hat mit der *Erzeugung* der bereitzustellenden Daten nichts zu tun.

Aus diesem Grund musste das vorhandene Muster, nach dem ein Webserver ausschließlich den Inhalt eines angeforderten Dokuments bereitstellen muss, erweitert werden. An dieser Stelle kam CGI (*Common Gateway Interface*) als eine Schnittstelle eines Webservers mit einem compilierten Programm ins Spiel. Es wurden (und die Betonung liegt auf »wurden«) also kleine Programme in einer mehr oder weniger beliebigen Programmiersprache (beispielsweise C oder auch Java) geschrieben, die über CGI als Vermittler aufgerufen wurden und Daten für die Auslieferung als HTML-Dokument wiederum an CGI bereitstellten. Das HTML-Dokument wurde anschließend wieder dem Webserver übergeben, der somit eine Antwort an den Client zurücksenden konnte.

Der erste große Nachteil von CGI-Scripts liegt in der geringen Performance, da für jeden Aufruf eines CGI-Scripts ein eigener Prozess auf dem Server initiiert werden muss. Da die meisten CGI-Scripts in Perl geschrieben wurden, musste bei jedem Aufruf eines CGI-Scripts der recht umfangreiche Perl-Interpreter geladen werden, wodurch ein großer Overhead entstand, bei dem das Laden des Interpreters oft mehr Zeit in Anspruch nahm als die eigentliche Ausführung des aufzurufenden Programms. Dem nicht genug: Das Ganze musste bei einem jeden (parallelen) Aufruf eines CGI-Scripts erfolgen. In Verbindung mit einer in der Regel geringen Leistungsfähigkeit des Systems, das dem Webservers zugrunde lag, führte dies dazu, dass Webserver nur eine geringe Anzahl an CGI-Anfragen gleichzeitig erlaubten, wodurch wiederum eine Art »Warteschlange« eingeführt werden musste.

Der zweite wesentliche Nachteil von CGI-Scripts ist die Zusammenarbeit mit compilierten Programmen, die über die CGI-Scripts aufgerufen werden. Der Einsatz compilierter Programme hat zwar den Vorteil, dass compilierte Programme in der Regel (wesentlich) schneller ausgeführt werden können als interpretierte Programme; sie müssen jedoch einerseits bei jeder Veränderung des dahinterliegenden Codes neu compiliert werden (was für Sie bei der Entwicklung einen Mehraufwand bedeutet),

und andererseits sind sie nicht plattformunabhängig und müssen somit auf das Betriebssystem der Serverhardware zugeschnitten sein. Eine jede Veränderung am Betriebssystem kann ein neuerliches Compilieren nach sich ziehen. Interpretierte Scripts unterliegen diesem Problem nur bedingt: Lediglich der Interpreter muss in Harmonie mit dem Betriebssystem ablaufen, nicht jedoch das von Ihnen programmierte System.

FastCGI als Performance-Boost

Um die Performance-Probleme in den Griff zu bekommen, wurde FastCGI ins Leben gerufen, bei dem der Perl-Interpreter nur ein einziges Mal geladen wurde und beliebigen Requests (auch von unterschiedlichen Clients) zur Verfügung steht.

Serverseitige Programmiersprachen als bessere Alternative

Heutzutage werden compilierte Programme im Rahmen von Websites nur noch sehr, sehr (sehr) selten eingesetzt, da in der Generierung dynamischer Websites mittlerweile bessere Alternativen existieren:

- **PHP** (*Hypertext Pre-Processor*) ist eine serverseitige Programmiersprache, die auf Serverseite über einen PHP-Interpreter verarbeitet wird, der üblicherweise als Erweiterung zu einem Webserver installiert wird. PHP ist weiters die einzige serverseitige Programmiersprache, die rein für Webanwendungen entwickelt wurde, weshalb sie auch eine sehr hohe Verbreitung aufweist.

  PHP ist die meistgenutzte serverseitige Sprache mit einer Verbreitung von knapp 77,4 %. Der Aufruf des PHP-Interpreters erfolgt nach wie vor über ein (Fast)CGI-Script (oder über eine ISAPI-Schnittstelle von Microsoft). Nach wie vor besteht also der Nachteil, dass – wie im Fall von Perl auch – der PHP-Interpreter bei jedem Aufruf durch den Webserver geladen werden muss und somit Ressourcen beansprucht bzw. die Verarbeitung der Anfrage verlangsamt. Um PHP für einen Webserver verfügbar zu machen, stehen mittlerweile drei Varianten zur Einbindung von PHP bereit:

  - **mod_php** ist ein Modul, das speziell für den Apache Webserver geschrieben wurde und den PHP-Interpreter in den Webserverprozess integriert, wodurch (zumindest) kein eigener Prozess für den PHP-Interpreter erzeugt werden muss, sondern alles vom Apache-Prozess verarbeitet wird. Der Performance-Gewinn liegt gegenüber CGI bei 300 bis 500 % aufgrund diverser Caching-Methoden, jedoch existiert *mod_php* nur für den Apache Webserver.
  - **CGI/FastCGI** mit seinen obig genannten Vor- und Nachteilen
  - **FPM** (*FastCGI Process Manager*) ist eine Alternative zu FastCGI, bei der immer ein zu einem Webserverprozess paralleler FPM-Prozess existiert, an den PHP-Anfragen vom Webserver weitergeleitet werden.

Abgesehen von PHP existieren natürlich auch weitere serverseitige Sprachen:

- **Python** kann auf der Serverseite genauso wie PHP als eine interpretierte Sprache verwendet werden. Jedoch wurde Python nicht rein für Webanwendungen erdacht, sondern ist vielmehr eine universelle Sprache. Das hat zwar den Vorteil, dass Sie an vielen unterschiedlichen Stellen auf Python treffen werden, dafür müssen Sie aber andererseits auch darauf achten, welche Funktionalitäten der Programmiersprache in welcher Umgebung (Webserver, Applikation etc.) verfügbar sind. Für die Integration von Python in einen Apache Webserver steht ein Modul namens *mod_python* bereit.
- **ASP und ASP.Net** sind Microsoft-Entwicklungen. ASP war lange Zeit nur für den *Internet Information Server* (ISS; einen Webserver von Microsoft) verfügbar, wurde dann jedoch auch für Apache veröffentlicht. Nichtsdestotrotz ist ASP passé, da es im Jahr 2002 von ASP.Net abgelöst wurde. ASP.Net ist Teil des Microsoft .Net-Frameworks, das auf dem IIS läuft. Mit einer Verbreitung von 7,6 % ist es die am zweithäufigsten verwendete Sprache auf Webservern.

Damit sind wir zwar der Antwort auf die Frage nach der Serversoftware unserer Wahl noch keinen Schritt näher gekommen, aber Sie haben (hoffentlich) eine bessere Idee von der Funktionsweise dynamischer Webseiten.

Was haben CGI & Co. mit Apache vs. NGINX zu tun?

Nun, diese Frage erscheint zunächst nicht unberechtigt. Bis dato haben wir zwei Fragen geklärt:

1. Was ist der Unterschied zwischen statischen und dynamischen Webseiten?
2. Wie verarbeitet ein Webserver ein statisches und wie ein dynamisches Dokument?

Also weiter im Text: Zunächst lohnt einmal ein Blick in die Statistik zur Webserver-Verbreitung, wie wir sie etwa unter *https://w3techs.com/technologies/overview/web_server* finden: NGINX (33,9 %) und Apache (32,8 %) liegen etwa gleichauf, Node.js ist mit 2 % weit abgeschlagen. Wo liegen die Gründe hierfür?

Apache Webserver

Der Apache Webserver (oder auch Apache HTTP-Server/Apache HTTPD) ist eine Open-Source Webserver-Software, die HTTP-Anfragen eines Clients mit einem entsprechenden Response beantwortet. Für ihn spricht die hohe Verbreitung von kapp 33 % (vgl. *https://w3techs.com/technologies/details/ws-apache*), wodurch er aktuell die Nummer Zwei der am weitesten verbreiteten Webserver auf dem Markt ist. Zudem spricht für ihn, dass er für alle gängigen Betriebssysteme (Plattformen) verfügbar ist und somit in vielen Softwarepaketen für Entwickler (wie XAMPP, LAMP, MAMP) gefunden wird. Ein großer Vorteil des Apache Webservers ist außerdem seine

Modularität, wodurch Erweiterungen (wie PHP oder Python) einfach hinzugefügt werden können.

Die Architektur des Apache sieht vor, dass eingehende Anfragen *prozessorientiert* verarbeitet werden, was bedeutet, dass für eine jede Anfrage an den Server ein eigener Thread (Hintergrundprozess) erzeugt wird, was wiederum sehr resourcenfressend ist. Glücklicherweise existieren mehrere *Multi-processing Modules* (MPMs), die auf diese Problematik unterschiedlich eingehen – wir werden es an dieser Stelle damit belassen.

Ob statische oder dynamische Daten ausgeliefert werden, ist dem Apache in der Regel einerlei: Statische Dateien kommen seinem »file-based approach« sehr entgegen, und mithilfe seiner Modularität können serverseitige Scriptsprachen (wie PHP oder Python) einfach integriert werden.

Eine weitere Stärke von Apache ist die verzeichnisbasierte Konfigurationsmöglichkeit, die es Ihnen erlaubt, mittels einer *.htaccess*-Datei jedes Verzeichnis individuell zu konfigurieren und so beispielsweise vor unerwünschten Zugriffen zu schützen.

Für den Apache spricht auch seine (sehr) gute Erweiterbarkeit mittels Modulen, die dynamisch integriert werden können. Das bedeutet, dass Sie – sofern ein administrativer Zugriff möglich ist – die Module ohne Neu-Installation des Apache aktivieren oder deaktivieren können. Aktuell existieren über 50 Module, die dem Apache zu mehr Funktionalität verhelfen. Weil jede Apache-Installation bereits vorkonfiguriert ausgeliefert wird, empfiehlt es sich, einen genauen Blick auf diese Module zu werfen und nicht benötigte Module zu deaktivieren.

Das Thema Security wird bei der *Apache Software Foundation* sehr groß geschrieben. Die Mitglieder dieser Open-Source-Organisation arbeiten ständig daran, auftretende Sicherheitsprobleme so schnell wie möglich zu beheben. Dabei hilft ihnen einerseits die große Community, andererseits betreiben sie die *Apache Server Announcement*-Mailingliste, um Administratoren rasch zu informieren, sollten Sicherheitslücken entdeckt werden. Insbesondere Denial-of-Service-(DOS-)Attacken sind für Webserver ein wesentliches Thema, und es wird versucht, sie über diverse Timeout-Funktionalitäten in den Griff zu bekommen.

NGINX (sprich: »Engine eX«)

NGINX ist ebenfalls Open-Source, kann jedoch auf keine so lange Historie wie Apache zurückblicken. Es gilt dafür aber (in manchen Bereichen) als sehr performant und gut skalierbar. Die Verbreitung von NGINX liegt bei knapp 34 %, was diesen Webserver an die erste Position in der Statistik hievt (*https://w3techs.com/technologies/details/ws-nginx*). Eine Stärke von NGINX ist das Bereitstellen statischer Dateien aufgrund seines hervorragenden Cachings.

Neben der reinen Anwendung als Webserver wird NGINX auch gern als Loadbalancer oder Reverse Proxy bzw. für das Cachen von Daten eingesetzt.

NGINX arbeitet im Gegensatz zu Apache nicht prozessorientiert, sondern ereignisbasiert. Die Anfragen sind asynchron und nichtblockierend (*non-blocking*) angelegt, was es NGINX ermöglicht, mehrere Anfragen parallel in einem einzigen Prozess zu verarbeiten, und ihm somit zu einer hohen Performance und Skalierbarkeit (Vergrößerung des Systems, beispielsweise durch Verbessern der Hardware) verhilft. Aus diesem Grund wird NGINX auch sehr gerne für Websites eingesetzt, auf denen viele Anfragen zu verarbeiten sind.

Wie bereits erwähnt, kann NGINX sehr gut mit statischem Content umgehen, mit dynamischem Content jedoch weniger gut. NGINX bietet keine Modul-Integration für Skriptsprachen wie PHP oder Python an und muss dynamische Inhalte deshalb mittels FPM-PHP oder FastCGI einbetten.

In puncto individuelle Konfiguration bietet NGINX keine Möglichkeiten an. Dies mag vielleicht im ersten Moment – und sofern zwingend erforderlich – als Nachteil angesehen werden, jedoch ermöglicht der Verzicht auf diese Individualkonfiguration dem NGINX wiederum höhere Performance und Skalierbarkeit.

Auch NGINX stellt mit ca. 100 Modulen eine breite Palette an Modulen bereit, jedoch können diese nicht dynamisch integriert werden, weshalb die Integration zusätzlicher Module nur über eine Neu-Installation möglich ist. Hierdurch wird das Verwenden zusätzlicher Module unmöglich für Webentwickler, die keine Installationsrechte am Server haben (was in der Praxis meist der Fall ist). Erst die kostenpflichtige Version *NGINX Plus* ermöglicht ein dynamisches Laden von Modulen.

Selbstverständlich liegt die Security auch den Entwicklern von NGINX am Herzen, was sie durch *Rate-Limiting* oder im Fall der Verhinderung von *Distributed Denial-of-Service*-(DdoS-)Attacken durch Erlauben oder Verhindern des Zugriffs auf die IP-Adressen der Clients unter Beweis stellen. Auch unterstützt NGINX die aktuellste Version von TLS (*Transport Layer Security*), was eine gute Verschlüsselung für die Übertragung von Daten bedeutet.

Und Node.js als Webserver?

Node.js ist eine sogenannte *Runtime* (Laufzeitumgebung) und kein Webserver per se. Selbstverständlich bietet eine solche Runtime auch die Möglichkeit, einen Webserver bereitzustellen, und das an sich auf relativ einfache Art. Die Sache ist jedoch die: Sie als Entwickler*in müssen einen Webserver *programmieren*, anstatt ihn einfach ausführen und gut ist es.

Dennoch wird Node.js auch gerne im Zusammenhang mit Webservern erwähnt, und das nicht ohne Grund: PHP in seiner Funktion als die meist genutzte serverseitige Programmiersprache bietet so gut wie keine Möglichkeit an, sogenannte *Web-Sockets*

zu nutzen (es sei denn, man würde den Server als permanent laufenden Application-Server betreiben). Die Web-Sockets werden aber im Zusammenhang mit Chat- oder ähnlichen Kommunikationssystemen benötigt, um ein System nicht zu überlasten.

Zwar werden Sie – sofern Sie Interesse daran haben – auf Begriffe wie *Long Polling* oder ähnliche stoßen, doch keine dieser Alternativen kann das HTTP-Problem lösen, dass es ein zustandsloses Protokoll ist. Und solange man dieses Problem am Hals hat, bedeutet dies, dass ein Client entweder in regelmäßigen Abständen den Server kontaktieren muss, um abzufragen, ob neue Informationen bereitstehen; oder der Client stellt eine einmalige Anfrage, die auf Serverseite mit einer Endlosschleife und einem Sleep-Zustand beantwortet wird, während der wiederum in regelmäßigen Abständen abgefragt wird, ob neue Informationen existieren. Schlussendlich ist es dann jedoch relativ egal, ob der Client oder der Server mit dieser regelmäßigen Anfrage beauftragt wird: Es ist in jedem Fall nicht die korrekte Vorgehensweise, denn sie erzeugt einen Overhead an Anfragen, der nach einiger Zeit dazu führen wird, dass der Server crasht.

Ein weiteres Problem ist, dass Node.js als Runtime auf einem Server installiert werden muss, was ohne entsprechende administrative Rechte (über die ein normalsterblicher Webentwickler normalerweise nicht verfügt) einfach nicht möglich ist. Node.js ist demnach ein Thema für (weit) fortgeschrittene Entwickler und Entwicklerinnen, die über entsprechende Möglichkeiten auf der Serverseite verfügen, indem sie beispielsweise eigene Server betreiben.

Es sei an dieser Stelle aber auch erwähnt, dass Node.js eher mit PHP als mit einem Webserver wie Apache oder NGINX verglichen werden kann, da Node.js JavaScript als Programmiersprache nutzt.

Welcher Webserver soll's nun werden?

Halten wir noch einmal die wesentlichen Vor- und Nachteile der unterschiedlichen Systeme mit einer Skala von 1 bis 10 (1: schwach, 10: supergut) fest:

	Apache	NGINX
Verbreitung	10	10
Multi-Funktionalität	3 (Webserver, mod_proxy, mod_cache)	4 (Webserver, Reverse Proxy, Loadbalancer, Caching-System)
Skalierbarkeit	6	10
Statische Inhalte	8	10

Tabelle 3.3 Der Versuch einer Bewertung von Apache vs. NGINX

	Apache	NGINX
Dynamische Inhalte	9	5
Verzeichnisbasierte Konfiguration	10	1
Modul-Handling	9	5 (NGINX Plus: 10)
Sicherheitsthemen	8	7 (NGINX Plus: 8)
Unterstützung	10	9

Tabelle 3.3 Der Versuch einer Bewertung von Apache vs. NGINX (Forts.)

Das Fazit lautet somit (sofern Sie eine Wahl haben):

- Wählen Sie den Apache, wenn Sie auf Shared-Hosting-Systemen arbeiten, bevorzugt dynamische Inhalte ausliefern und wenn sich die parallelen Zugriffe auf Ihr System in Grenzen halten (wenn Sie also keine Suchmaschine oder Ähnliches betreiben).
- Wählen Sie NGINX, wenn Sie hoch performante Systeme entwickeln wollen, die bevorzugt statischen Content ausliefern. Auch eignet sich NGINX hervorragend für die Skalierung von Systemen.

3.8.2 HTTP vs. HTTPS – unsicher oder doch sicher?

Wie ich bereits in Abschnitt 3.2 über Protokolle und Ports angemerkt habe, können Websites über HTTP oder über HTTPS betrieben werden. Das Kürzel HTTP steht für *Hypertext Transfer Protocol*. Das »S« in HTTPS steht für *secure*. Der Unterschied zwischen HTTP und HTTPS liegt in der Art und Weise, wie die Daten zwischen Client und Server transferiert werden: Während im Falle von HTTP die Daten als *Plain Text* (also im Klartext) und somit unsicher transferiert werden, setzt HTTPS auf eine Verschlüsselung der Daten mittels SSL auf Basis einer asymmetrischen Technik. (Nähere Informationen zur asymmetrischen Verschlüsselung finden Sie in Abschnitt 13.5.)

Die Entscheidung, ob Daten zwischen einem Client und einem Server verschlüsselt über HTTPS oder unverschlüsselt über HTTP transferiert werden, kann – anders als man vielleicht vermuten mag – von uns beim Webdesign nicht beeinflusst werden. Diese Entscheidung wird im Rahmen des Hostings von Websites getroffen. Websites können grundsätzlich auf beide Arten betrieben werden, wobei der Trend jedoch zu HTTPS geht.

Grundsätzlich sollte eine jede Website mit HTTPS betrieben werden, sobald User-Eingaben an den Server übermittelt werden (Formulare, Passwörter, Kreditkarteninfor-

mationen etc.). Und darüber hinaus kennzeichnen die gängigen Browser eine mit HTTP betriebene Website als unsicher, was mit Sicherheit keinen guten Eindruck bei den Usern hinterlässt.

3.8.3 Ein paar Worte zum Cloud-Hosting

War es in den letzten Jahrzehnten noch üblich, auf einen Hosting-Provider zurückzugreifen, so haben sich im letzten Jahrzehnt mehr und mehr Cloud-Hoster am Markt etabliert. Hierbei befindet sich die Website (oder auch die App) auf virtuellen Servern in der Cloud. Dies bietet gegenüber dem Webhosting vor allem einen wesentlichen Vorteil: Benötigt eine Website – etwa aufgrund erhöhter Nachfrage – auf einmal mehr Ressourcen, so muss beim Webhosting auf ein anderes Paket umgestellt werden. Im Falle von Cloud-Hosting kann der Website-Inhalt auf mehrere virtuelle Server verteilt werden, wodurch sich die Last pro Server verringert. Aus diesem Grund spricht man auch davon, dass Cloud-Hosting robuster und zuverlässiger als Webhosting ist.

Cloud-Hoster gibt es mittlerweile recht viele am Markt: Google Cloud, Azure (Microsoft), AWS (Amazon Web Services) und viele weitere.

3.9 Ach ja, und wie funktioniert nun eine Website?

Fasst man die bis dato gesammelten Informationen nun zusammen, so ergibt sich für eine Website nun folgendes Bild:

1. **Der Website-Betreiber**
   - registriert eine Domain.
   - stellt (vor allem) einen Webserver (Web-Host) bereit, auf dem die Website gehostet werden soll.
   - stellt üblicherweise einen Mailserver (Mail-Host) bereit, über den Mails gesendet und empfangen werden können.
   - stellt Zugangsdaten zu den jeweiligen Servern (Hosts) bereit, die an den Webentwickler weitergegeben werden.
2. **Der Webentwickler / die Webentwicklerin**
   - erhält Hosting-Informationen vom Website-Betreiber (insbesondere FTP-Zugangsdaten).
   - lädt die Dateien (sowie gegebenenfalls Datenbankdaten) auf den Host hoch und stellt somit die Website den potenziellen Usern zur Nutzung bereit.
3. **Die User**
   - öffnen einen Browser (WWW-Client) und geben den Domainnamen ein.

- Der Browser stößt im Hintergrund eine Namensauflösung an (Übersetzung der Domain in eine IP-Adresse), indem er eine DNS-Anfrage an denjenigen DNS-Server stellt, der durch den Internet Service Provider des Users bereitgestellt wird. (Dies ist üblicherweise ein Eintrag in den Konfigurationseinstellungen des Modems.) Er erhält dann die IP-Adresse desjenigen (Web-)Servers, der die Daten zur gewünschten Domain bereithält.
- Der Browser des Users kontaktiert nun diesen Webserver in Form eines »Requests« und erhält vom Server die gewünschten Daten in Form eines »Response«.

Der Request an den Server: eine unvollständige Darstellung

An dieser Stelle sei angemerkt, dass die Beschreibung der Anfrage eines Browsers an einen Server hier noch nicht ganz vollständig und exakt ist. Jedoch ist sie an dieser Stelle präzise genug, um ein gewisses Grundverständnis für die Funktionsweise einer Website zu erhalten. In den folgenden Kapiteln werden wir auf diese Funktionsweise noch einige Male zu sprechen kommen und die Darstellung Schritt für Schritt verfeinern.

Abbildung 3.4 fasst die einzelnen Schritte noch einmal zusammen:

1. Nachdem der User in seinem Browser die Domain eingegeben hat, die er besuchen möchte, hängt der Browser zunächst einmal das zu erwartende Protokoll (hier: HTTP) an diese Anfrage an. Bevor der Browser jedoch den zugehörigen Server kontaktieren kann, muss dessen IP-Adresse erst einmal ermittelt werden. Hierzu kontaktiert der Browser einen Nameserver-Dienst und übermittelt diesem die Domain (samt Dienst).
2. Der Nameserver löst (gegebenenfalls unter Zuhilfenahme weiterer Nameserver) die Domain in eine IP-Adresse auf, die er an den Browser zurückliefert.
3. Nun kann der Browser eine Anfrage an den Server stellen (konkret: an die vom Nameserver zurückgelieferte IP-Adresse, der er das Protokoll voranstellt) und dem Server sowohl die eigene IP-Adresse wie auch einen Port schicken, an die der Server antworten kann. Dies nennt man den *Request an den Server*.
4. Der Server verarbeitet nun den Request, indem er das vom Client gewünschte Dokument einliest. Ist ein solches Dokument nicht mit angegeben worden, nimmt der Server stattdessen das Standarddokument (*index.html* oder ein ähnliches; siehe Abschnitt 3.8).
5. Der Server liefert den Inhalt des gewünschten Dokuments in Form seines *Response an den Client* zurück.
6. Der Client verarbeitet in weiterer Folge die Antwort des Servers, indem er aus dem übermittelten Inhalt zunächst den HTML-Code abarbeitet (»parst«) und daraus

den sogenannten *Dokumentbaum* erzeugt. Währenddessen fragt der Browser alle noch benötigten Daten vom Server an. Alsdann macht er sich über die Formatierungs- und Scriptdateien her und arbeitet diese ab.

7. Sobald alles verarbeitet wurde und auch alle noch angefragten Daten vom Server geliefert wurden, ist der Client fertig.

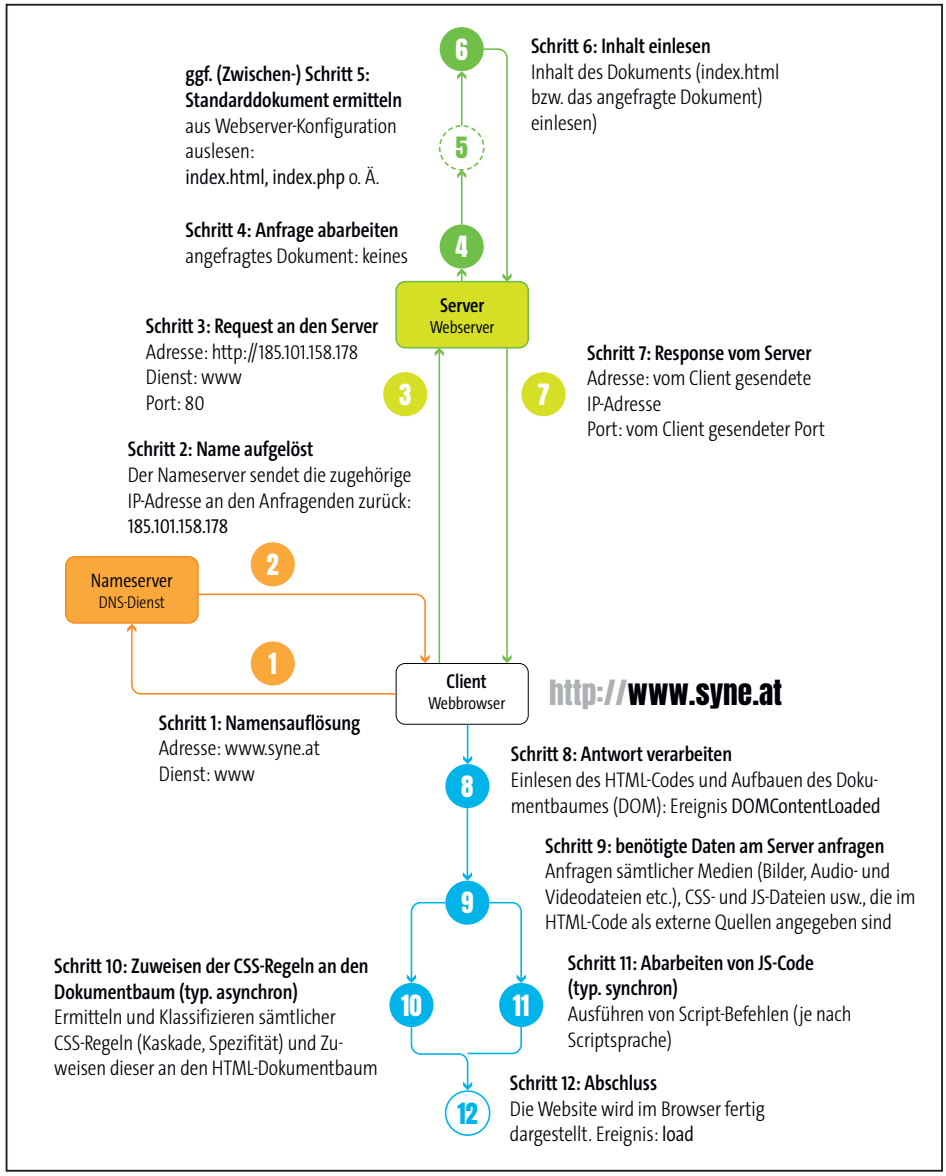


Abbildung 3.4 Der Ablauf beim Aufruf einer Website (ohne Angabe eines Dokuments) im Browser (Client) des Users

In den folgenden Kapiteln werden wir noch mehrere Male auf diese Grafik zurückkommen und uns einzelne Teile detaillierter ansehen. Nichtsdestotrotz liefert diese Grafik einen Überblick über die wesentlichen Schritte bei der Abarbeitung einer Website durch Client, Name- und Webserver. Sie ist die Grundlage für das weitere Verständnis der Webentwicklung.

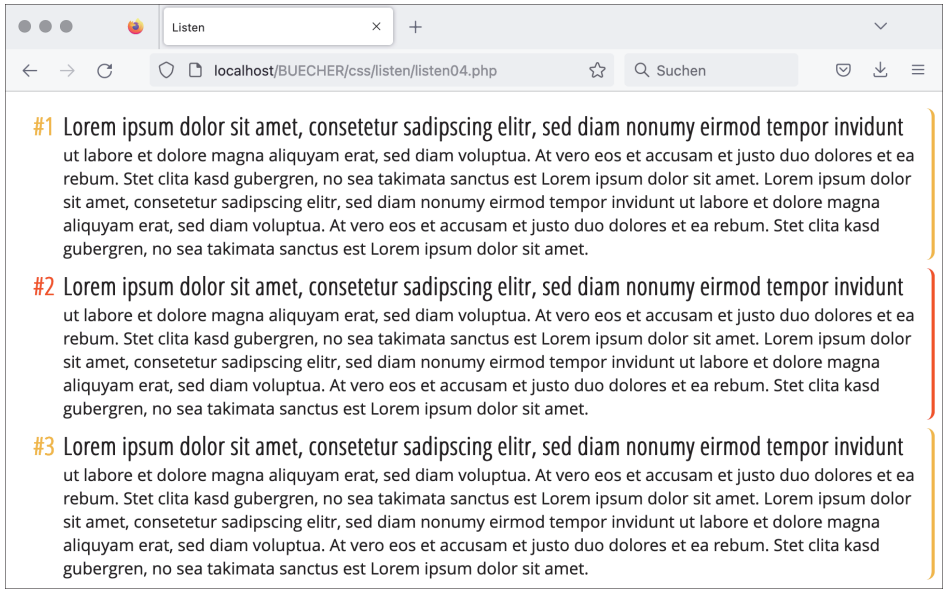


Abbildung 6.61 Die nummerierte Liste aus dem vorangegangenen Abschnitt erhält nun auch eine Individualnummerierung. Bewegt der User den Mauszeiger über einen Listenpunkt, so wird das Hovering aktiv und der Listenpunkt rot markiert.

6.17 Fortgeschrittenes Layout – Jetzt wird's fein

6.17.1 Flexboxen – eindimensionales Denken

Flexboxen ermöglichen eine saubere Platzierung von Elementen innerhalb eines Containers, und zwar in einer Dimension (horizontal oder vertikal). Wie Sie noch sehen werden, ist die Unterscheidung zwischen *horizontal* und *vertikal* genau der Grund, warum man nicht von *»von links nach rechts«* (es hätte auch *»von rechts nach links«* sein können) oder *»von von oben nach unten«* (bzw. *»von unten nach oben«*) spricht. Aber alles der Reihe nach ...

Grundsätzlich haben Sie es mit zwei Dingen zu tun:

- mit einem umgebenden Container, der die *Flexbox* darstellt: Diesem wird die notwendige Eigenschaft `display:flex;` zugewiesen.
- mit den Elementen innerhalb der Flexbox (den *Flex Items*), die entsprechend der Vorgaben ausgerichtet werden. Hierzu sei angemerkt, dass Elemente innerhalb einer Flexbox automatisch zu Block-Elementen werden.

Der entsprechende HTML-Code sieht also recht einfach aus:

```
<div class="flexbox">
  <div>Box A</div>
  <div>Box B</div>
  <div>Box C<br>Box C</div>
  <div>Box D<br>Box D<br>Box D</div>
  <div>Box EeeeeeEeeeeeEeeeee</div>
  <div>Box F Box F Box F Box F</div>
  ...
</div>
```

Natürlich ist es unerheblich, ob

- es sich um `div`-Elemente oder andere Container handelt.
- die Formatierung mithilfe einer Klasse `flexbox` oder auf eine andere Art und Weise (z. B. durch Verwendung einer ID) erfolgt.
- welcher Inhalt sich innerhalb der geflexten Elemente befindet.

In diesem Beispiel verwende ich zur optischen Veranschaulichung folgenden (zusätzlichen, aber für Flexboxen nicht benötigten) CSS-Code:

```
.flexbox {
  padding:0.2em;
  border:2px solid #ccc;
  background-color:#eee;
  border-radius:0.2em;
  margin-bottom:1em;
  width:100%;
}
.flexbox > * {
  padding:1em;
  border-radius:0.2em;
  border:2px solid white;
}
...
```

Ohne Einsatz einer Flexbox würde das obige Beispiel so wie in Abbildung 6.62 aussehen.

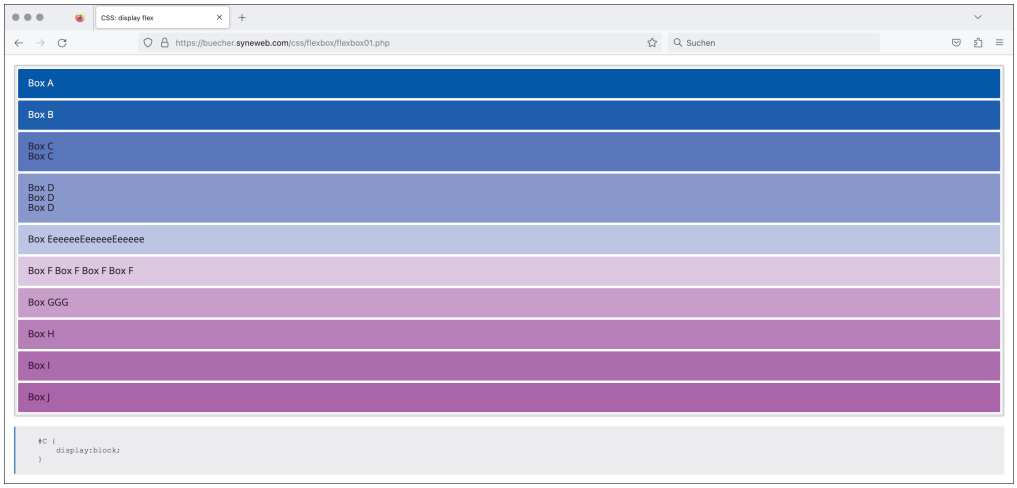


Abbildung 6.62 Die Darstellung von normalen Block-Elementen war zu erwarten.

Um die Elemente nun entsprechend als Flex-Elemente darzustellen, wird der Klasse `flexbox` die Eigenschaft `display:flex;` zugewiesen. In diesem Zusammenhang kommen auch gleich einige Standardeinstellungen für Flexboxen zum Tragen (dazu jedoch später mehr). Das Resultat ergibt das, was Sie in Abbildung 6.63 sehen.

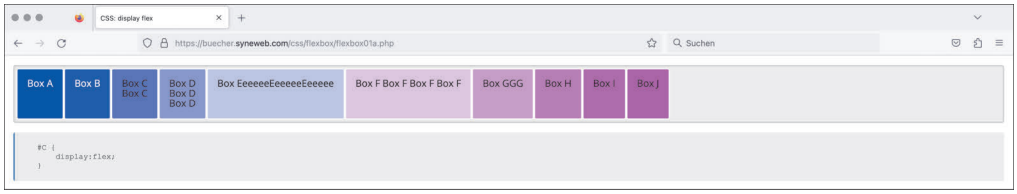


Abbildung 6.63 Die Elemente innerhalb der Flexbox richten sich nun neu aus.

Folgendes fällt auf:

- Die geflexten Elemente werden horizontal nebeneinander angeordnet. Der Grund ist, dass die horizontale Darstellung die Standardrichtung ist. Diese können Sie mithilfe der Eigenschaft `flex-direction:`**`row`** `| row-reverse | column | column-reverse;` festlegen.
- Die Elemente sind nur so breit wie nötig. Die dahinterliegende Eigenschaft ist `flex-shrink:0 | 1;` womit angegeben werden kann, ob ein Element sich verkleinern darf (Wert `1`) oder nicht.

- In der Höhe nehmen alle Elemente den gesamt zur Verfügung stehenden Platz ein (dieser wird im aktuellen Beispiel durch das höchste Element (*Box D*) bestimmt). Die zugehörige Eigenschaft lautet:

```
align-items: stretch | flex-start | flex-end | center | baseline ...;
```

Die Standardwerte der Eigenschaften sind jeweils fett gekennzeichnet.

Wir haben es bei Flexboxen ja mit zwei Richtungen zu tun: einerseits mit der horizontalen Ausrichtung, andererseits mit der vertikalen Ausrichtung. Oder eben umgekehrt! Je nach `flex-direction` bestimmt diese entweder die horizontale Ausrichtung (`flex-direction:row` oder `flex-direction:row-reverse`) oder die vertikale Ausrichtung (`flex-direction:column` oder `flex-direction:column-reverse`). Die zweite Eigenschaft, `align-items`, übernimmt dann die jeweils andere Dimension:

- `flex-direction:row | row-reverse` legt eine horizontale Ausrichtung der Elemente fest, wodurch `align-items` die vertikale Ausrichtung übernimmt.
- `flex-direction:column | column-reverse` legt eine vertikale Ausrichtung der Elemente fest, wodurch `align-items` wiederum die horizontale Ausrichtung übernimmt.

In Abbildung 6.64 und Abbildung 6.65 finden Sie eine grafische Zusammenfassung einiger der oben genannten Einstellungen.

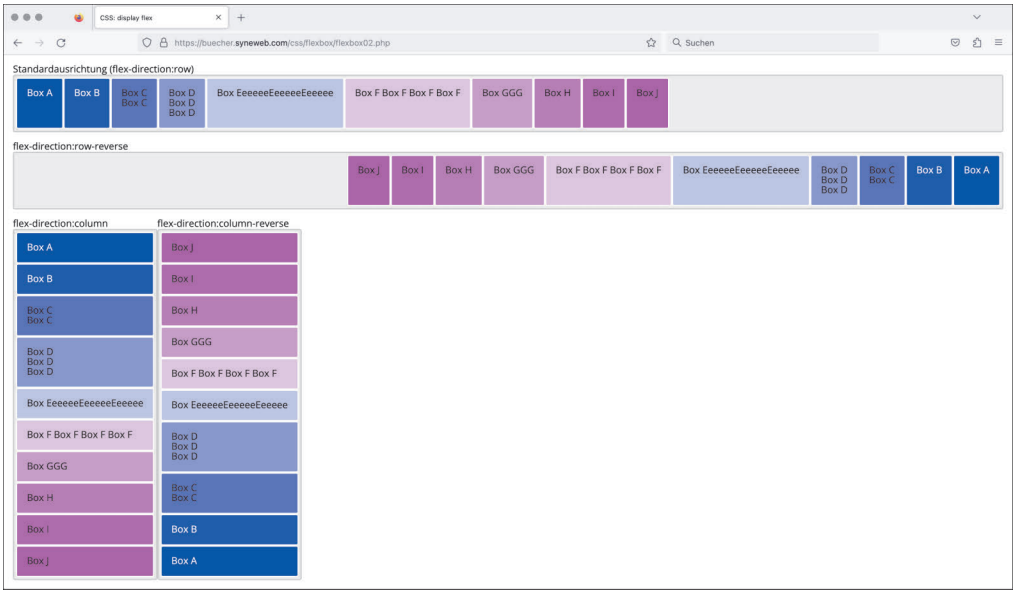


Abbildung 6.64 Die erste Flexbox zeigt die Standarddarstellung von »flex-direction (row)«; die zweite Flexbox verwendet »row-reverse«; die beiden darunterstehenden Flexboxen setzen »column« bzw. »column-reverse« ein.

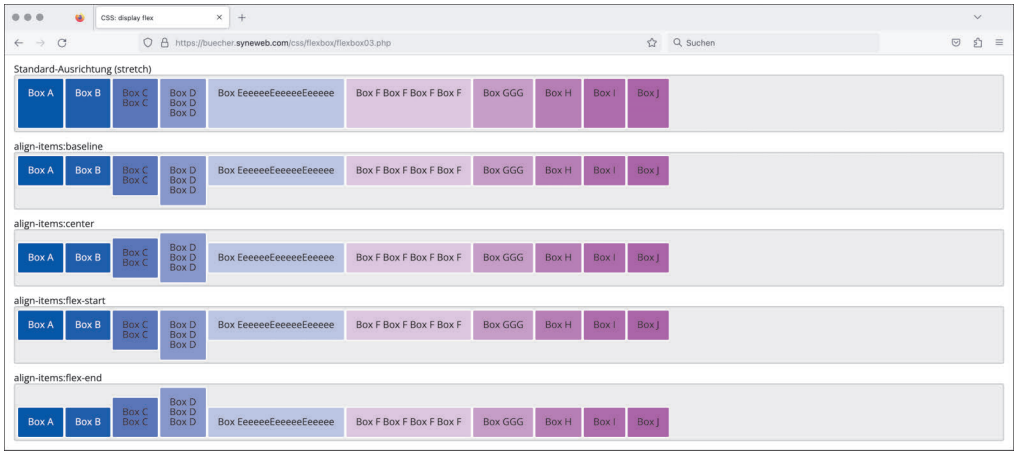


Abbildung 6.65 »align-items« schlagen sich im Falle von »flex-direction:row« (oder »flex-direction:row-reverse«) in der vertikalen Richtung nieder. Würde man »flex-direction:column« (oder »column-reverse«) verwenden, so würde »align-items« die horizontale Richtung betreffen.

Den Flex-Elementen können Sie über die Eigenschaft `flex-basis` den Platzbedarf zuweisen, den sie einnehmen sollen. Folgende Situationen können auftreten:

- Es bleibt noch Platz übrig, da die Elemente nicht so viel Platz benötigen, wie zur Verfügung steht. Somit stellt sich die Frage: Was geschieht mit dem freien Platz?
- Es ist nicht genügend Platz vorhanden, da die Elemente mehr Platz benötigen, als zur Verfügung steht. Die Frage ist: Soll umbrochen werden oder soll der Inhalt über die Flexbox hinaus stehen?

Grundsätzlich werden die Flex-Elemente alle nebeneinander (untereinander) dargestellt, ein Umbruch findet also nicht statt – Abbildung 6.66 zeigt dies.

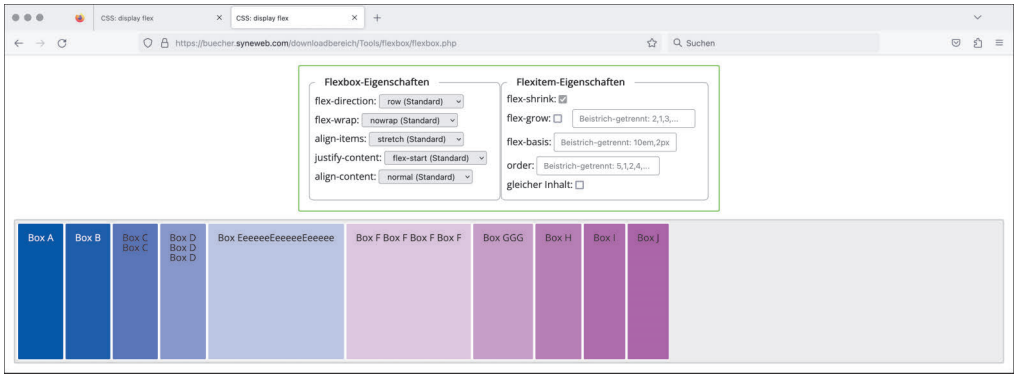


Abbildung 6.66 Sämtliche Boxen stehen nebeneinander – es ist ja auch ausreichend Platz zur Verfügung.

Sollte der Platz jedoch nicht ausreichen, da etwa die `flex-basis`-Eigenschaft zu große Elemente erzeugt, so muss der Inhalt über den Container (die Flexbox) hinausschießen. Wir haben das abgefangen, indem wir das `overflow-x` auf `auto` gestellt haben und somit immer dann einen Scrollbar erzeugen, wenn dieser notwendig wird (siehe Abbildung 6.67).

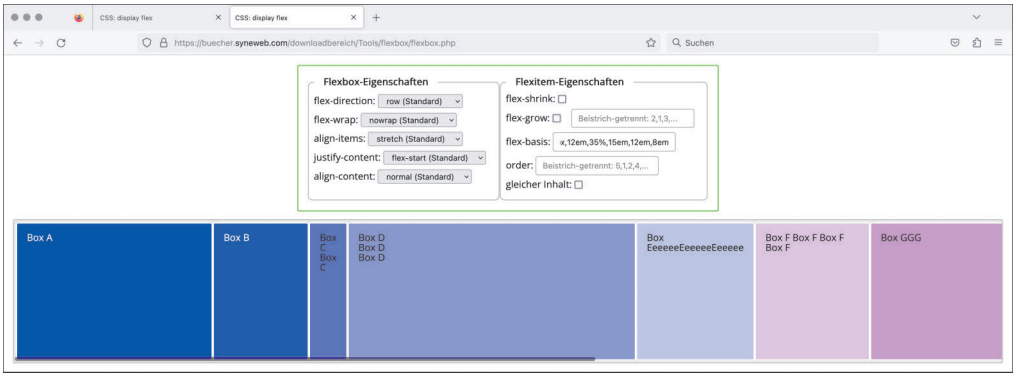


Abbildung 6.67 Die Flexbox ist zu klein für den Inhalt, weshalb der Inhalt über die Box hinausschießt. Lediglich der Scrollbar ermöglicht es, den Inhalt noch betrachten zu können.

Wie Sie gut erkennen können, benötigen die Flex-Items eigentlich gar nicht so viel Platz, wie wir ihnen zugewiesen haben. Um ein Hinausschießen über die Flexbox zu verhindern, könnten Sie in diesem Fall ein `flex-shrink` erlauben, wodurch sich die Elemente – unabhängig von der Einstellung der `flex-basis` – so weit verkleinern, dass sie doch wieder in die Flexbox passen. Abbildung 6.68 zeigt diesen Fall.

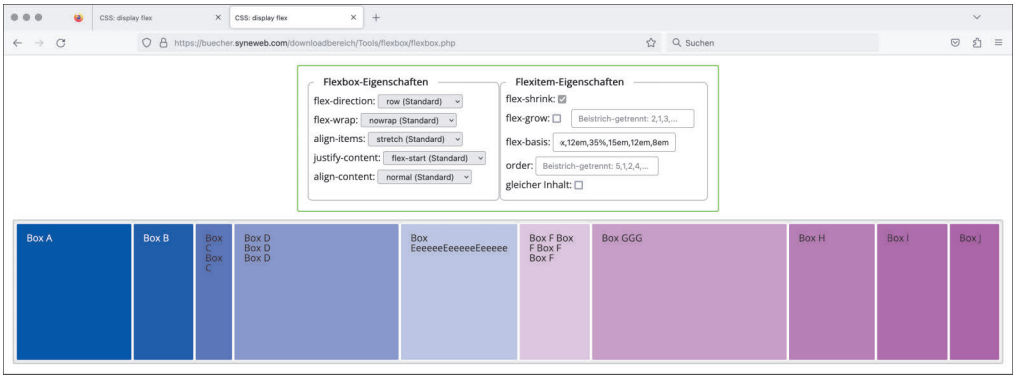


Abbildung 6.68 Die Eigenschaft »flex-shrink« erlaubt, dass sich die Flex-Items auf den benötigten Platz verkleinern, sollte dies notwendig sein.

Kommt anstatt von `flex-shrink` die Eigenschaft `flex-wrap:wrap` (oder `wrap-reverse`) zum Einsatz, erlaubt die Flexbox einen Umbruch der Flex-Items in eine zweite Zeile (oder in eine dritte oder vierte usw., sofern notwendig) – Abbildung 6.69 zeigt dies.

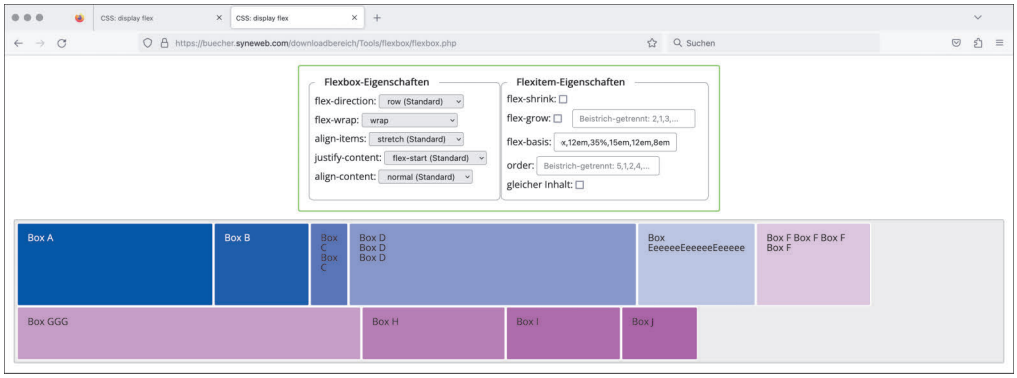


Abbildung 6.69 Mithilfe von »flex-wrap« wird es möglich, dass die Flex-Items eine neue Zeile bilden (oder auch eine neue Spalte im Falle von »flex-direction:column;«).

Wie viel Platz ein jedes Flex-Element erhält, wird pro Flex-Element definiert, also über die Eigenschaft `flex-basis`. Die Eigenschaften `flex-shrink` und `flex-grow` definieren *je Element* (Flex-Item), ob ein *Kleinerwerden* (`flex-shrink`) oder ein *Größerwerden* (`flex-grow`) erlaubt sein soll, sofern dies benötigt wird.

Weitere Eigenschaften von Flexboxen sind:

- `justify-content`: Gibt an, wie sich die Flexelemente den noch freien Platz innerhalb einer Flexbox aufteilen. (Im Fall der `row`-Richtung teilen sie sich den horizontalen Platz; im Fall der Spalten-Richtung den vertikalen Platz.) Die möglichen Werte sind:

  `flex-start` | `flex-end` | `space-around` | `space-between` | `space-evenly` | `center` | `start` | `end` | `left` | `right`

  Zwei Beispiele sehen Sie in Abbildung 6.70 und Abbildung 6.71.

- `align-content`: Gibt an, wie sich die Flex-Elemente im Fall von `flex-wrap` den freien Platz in der jeweils nicht genutzten Dimension (bei `row`: die `col`-Richtung; bei `col`: die `row`-Richtung) aufteilen. Die möglichen Werte sind:

  `normal` | `flex-start` | `flex-end` | `space-around` | `space-between` | `space-evenly` | `center` | `start` | `end` | `stretch` | `baseline` | `first-baseline` | `last-baseline`

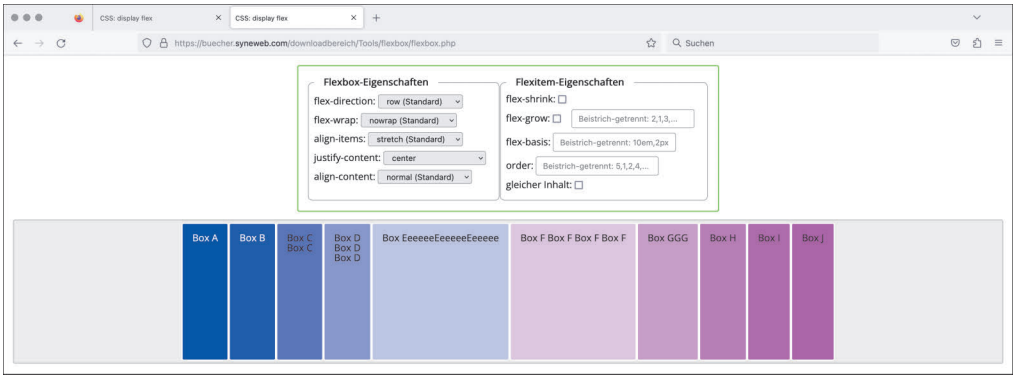


Abbildung 6.70 Die Eigenschaft »justify-content« definiert, wie die Elemente horizontal ausgerichtet werden. Sollte die »flex-direction« auf »column« (oder »column-reverse«) gesetzt sein, gilt dies natürlich für die vertikale Ausrichtung.

Beispiel 2 für `justify-content`:

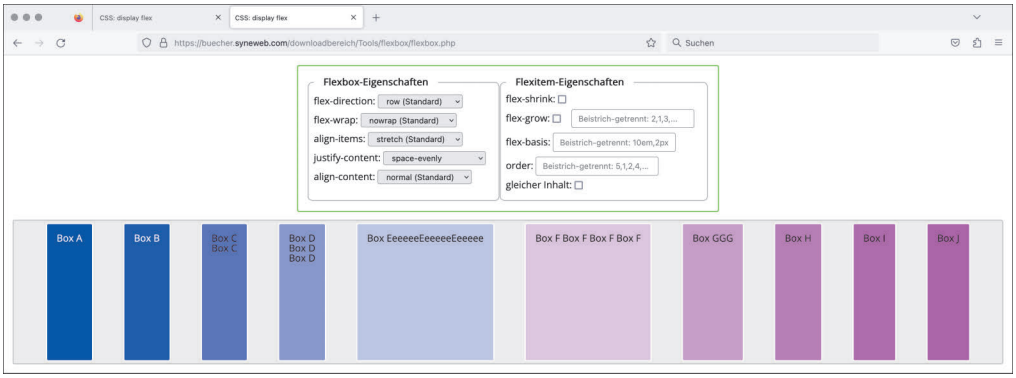


Abbildung 6.71 »justify-content:space-evenly« teilt den verbleibenden Platz gleichmäßig auf.

Um mit Flexboxen zu experimentieren, finden Sie im Downloadbereich zum Buch einen HTML- und CSS-Code für die oben dargestellte Spielwiese.

6.17.2 Grids – zweidimensionales Denken

Das *CSS-Grid* ist eine logische Weiterentwicklung der eindimensionalen Flexboxen hinsichtlich eines zweidimensionalen Rasters. Flexboxen sind von ihrer Natur her eigentlich nicht in der Lage, einen Gestaltungsraster aufzubauen, da untereinander stehende Boxen (anders als beispielsweise eine Tabelle) keinen Zusammenhang zu den darüber liegenden Boxen aufweisen. Somit könnte es bei einem solchen Flexbox-

Grid passieren, dass zwei untereinanderliegende Spalten unterschiedlich breit sind. Ein solches Problem darf jedoch bei einem Grid nicht auftreten!

Der Grundaufbau im HTML ist sehr ähnlich zum Aufbau von Flexboxen: Sie benötigen einerseits einen umgebenden Container, der die Eigenschaft `display:grid;` erhält, sowie die darin befindlichen Grid-Items. Die Reihenfolge der Grid-Items im HTML-Code ist grundsätzlich nicht relevant, da diesen in weiterer Folge Positionen im Grid zugewiesen werden können.

Neben der Grundeigenschaft `display:grid;` müssen Sie dem Grid-Container noch mitteilen, wie sein Aufbau hinsichtlich Spalten und Zeilen ist, was mit den Eigenschaften `grid-template-columns` und `grid-template-rows` erfolgt. Den Items wird – sofern erwünscht – mit `grid-column` und `grid-row` mitgeteilt, wohin sie sich positionieren müssen, was der Grund ist, warum die Reihenfolge im HTML-Code irrelevant ist.

Die Terminologie eines CSS-Grids sieht wie folgt aus:

- **Grid-Container**: Das ist der übergeordnete Container, der die Eigenschaft `display:grid;` erhält.
- **Grid-Items**: die Elemente, die ins Grid platziert werden sollen; sie sind direkte Kinder des Grid-Containers.
- **Grid-Line**: Sie stellt die Grenze zwischen Grid-Cells dar und kann naturgemäß horizontal oder vertikal sein.
- **Grid-Track**: Ein Track ist der Bereich zwischen zwei *aufeinanderfolgenden* Grid-Lines, also entweder eine Zeile zwischen horizontalen Grid-Lines oder eine Spalte zwischen vertikalen Grid-Lines.
- **Grid-Cell**: Sie wird durch horizontal und vertikal *aufeinanderfolgende* Grid-Lines begrenzt, vergleichbar mit einer Zelle einer Tabelle.
- **Grid-Area**: Sie stellt einen Bereich in einem Grid dar, der durch horizontale und vertikale Grid-Lines begrenzt ist. Anders als bei einem Track müssen diese jedoch *nicht aufeinanderfolgen*. Sie sind vergleichbar mit Zellen einer Tabelle, die mit `colspan` oder `rowspan` zusammengefasst wurden.

Gehen wir von dem Beispiel aus, das wir auch bei den Flexboxen verwendet hatten (mit dem Unterschied, dass ich die CSS-Klasse nun `grid` getauft habe):

```
<div class="grid">
  <div>Box A</div>
  <div>Box B</div>
  <div>Box C<br>Box C</div>
  <div>Box D<br>Box D<br>Box D</div>
  <div>Box EeeeeeEeeeeeEeeeee</div>
  <div>Box F Box F Box F Box F</div>
```

```
  <div>Box GGG</div>
  <div>Box H</div>
  <div>Box I</div>
  <div>Box J</div>
</div>
```

Wie zu erwarten war, passiert mit unseren Elementen zunächst noch nichts Großartiges (siehe Abbildung 6.72), da unsere Klasse `grid` noch keine Regeln beinhaltet:

```
.grid {
}
```

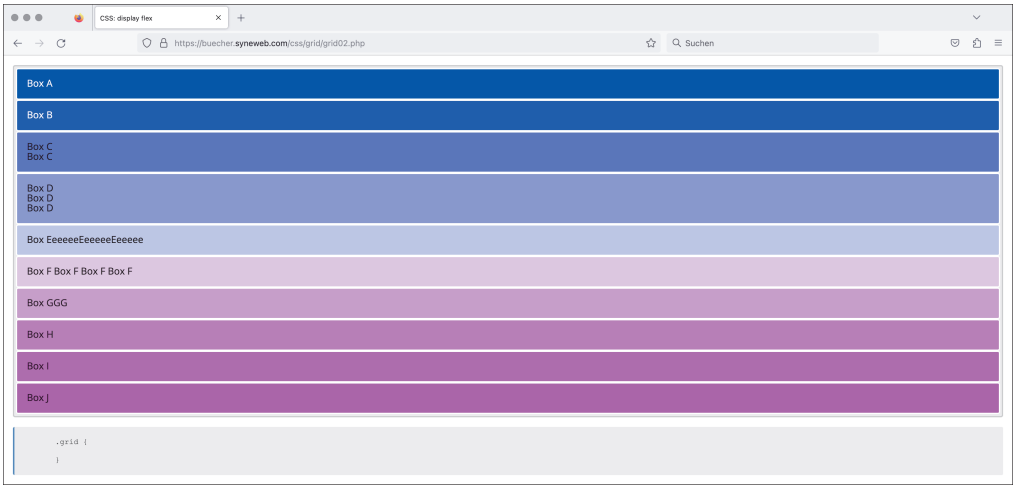


Abbildung 6.72 Noch ist kein Grid zu erkennen.

Dies ändert sich jedoch schnell, wenn wir die Eigenschaft `display` auf `grid` setzen (`display:grid;`) und ein dreispaltiges Design entwerfen: Die erste Spalte im Grid soll `5em` breit sein, die zweite Spalte soll sich einen Anteil vom übrig gebliebenen Rest nehmen (`1fr`), die dritte Spalte bekommt zwei Anteile davon (`2fr`). Die erste Zeile soll `8em` hoch sein, und alle folgenden Zeilen passen sich in der Höhe automatisch an (da keine weiteren Höhen angegeben sind):

```
.grid {
  height:70vh;

  display:grid;
  grid-template-columns: 5em 1fr 2fr;
  grid-template-rows: 8em;
}
```

Zur Erklärung:

1. `grid-template-columns` definiert die Breite der Spalten, wobei diese auf vielfältige Weise definiert werden können.
2. `grid-template-rows` definiert sinngemäß die Höhe der Zeilen, ist jedoch optional; alternativ kann man die Anzahl der Zeilen offen lassen (also nicht explizit angeben) – dann wird die Anzahl der Rows automatisch erzeugt. Die Höhe dieser Rows könnten Sie mit `grid-auto-rows` angeben.
3. Die Höhenangabe ist selbstredend optional und wurde nur aufgrund einer besseren Darstellung gewählt.

Als Einheiten für sowohl Spalten als auch Zeilen können Sie alle in CSS möglichen Einheiten verwenden, also Pixel (`px`), Prozent (`%`), Schriftgrößeneinheiten (`em`, `rem`) sowie ein paar besondere:

- `auto` stellt den »übrig bleibenden« Platz dar.
- `fr` (Fraction) stellt einen »Teil des übrig bleibenden« Platzes dar. Hiermit können Sie etwa den verbleibenden Teil des Platzes gleichmäßig aufteilen. Im obigen Beispiel bleibt also zu den 100 % der Breite nach Abzug von 5 em noch ein Platz frei. Dieser wird nun in diesem Fall in drei (1+2) Teile aufgeteilt und entsprechend an die Spalten vergeben.

Das Ergebnis ist in Abbildung 6.73 dargestellt.

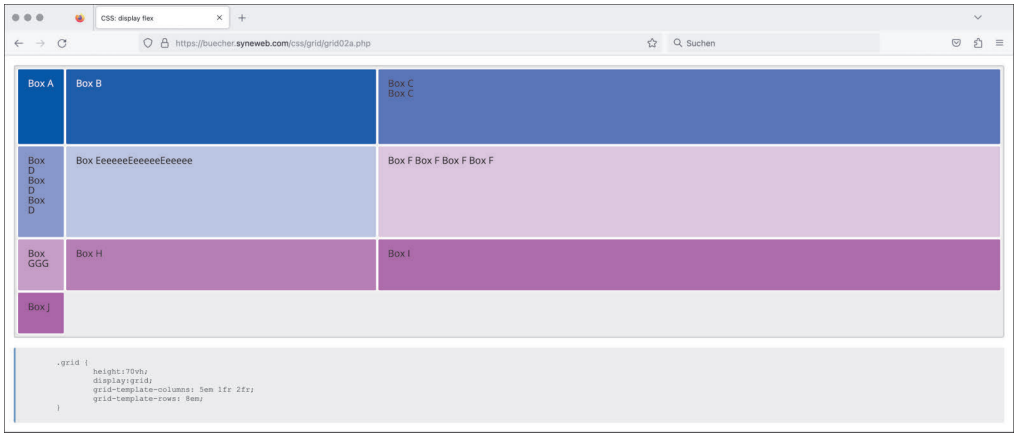


Abbildung 6.73 Weil sowohl die Spalten als auch die Zeilen teilweise relativ angegeben sind, nutzt das Grid den gesamten ihm zur Verfügung stehenden Platz aus: 100 % in der Breite, 70 vh in der Höhe.

Sollte man mehr Grid-Items als Grid-Plätze haben (in unserem Beispiel hat es den Anschein, dass das Grid aus elf Plätzen besteht, wobei einer frei zu bleiben scheint), so geschieht die Aufteilung automatisch auf Basis der `grid-template-rows`. Gehen wir

vom obigen CSS-Code aus und stellen wir jedoch nicht zehn Grid-Items, sondern 15 Grid-Items zur Verfügung, so werden weitere Zeilen angefügt. Welche Darstellung diese bekommen, wird über die Eigenschaften `grid-auto-columns` und `grid-auto-rows` definiert (siehe hierzu etwa *https://developer.mozilla.org/en-US/docs/Web/CSS/grid-auto-columns* sowie *https://developer.mozilla.org/en-US/docs/Web/CSS/grid-auto-rows*).

Zusätzlich zu den Spalten und Zeilen besteht auch die Möglichkeit, zwischen beiden Abstände zu definieren:

- `grid-column-gap` | `column-gap`: Abstand zwischen den Spalten
- `grid-row-gap` | `row-gap`: Abstand zwischen den Zeilen
- `grid-gap` | `gap`: gemeinsamer Abstand für sowohl Zeilen als auch Spalten

Neben der Darstellung des Grids geschieht im Hintergrund auch eine Nummerierung der Grid-Lines, und zwar sowohl positiv als auch negativ:

- In diesem Beispiel existieren vier vertikale und fünf horizontale Grid-Lines.
- Die erste vertikale Grid-Line erhält die Nummerierungen 1 und -4, die zweite 2 und -3, die dritte 3 und -2 und die vierte 4 und -1, also einmal von vorne und einmal von hinten gezählt.
- Die erste horizontale Grid-Line erhält die Nummerierungen 1 und -5, die zweite 2 und -4, die dritte 3 und -3, die vierte 4 und -2 und die fünfte 5 und -1.

Bei Bedarf können den Grid-Lines auch Namen zugewiesen werden, die in der Definition der Grid-Columns und -Rows in eckigen Klammern angegeben werden. Möchten Sie etwa die vertikalen Linien mit »Linie_1«, »Linie_2«, »LineA« und »BB« sowie die horizontalen Grid-Lines mit »h1«, »horiz« und »CX« benennen (Sie sehen schon: die Namen haben keine Bedeutung, dürfen jedoch keine Leerzeichen beinhalten), so würden Sie diese für das obige Beispiel wie folgt notieren:

```
.grid {
  ...
  grid-template-columns: [Linie_1] 5em [Linie_2] 1fr [LineA] 2fr [BB];
  grid-template-rows: [h1] 8em [horiz] auto [CX];
}
```

Der Grund für eine derartige Vorgehensweise ist, dass Sie in weiterer Folge die Definition der Grid-Areas durch Angabe von etwa »von Linie_1 bis LineA« usw. vornehmen können. Es bestünde sogar die Möglichkeit, einer Linie zwei Namen zu geben, indem Sie innerhalb der eckigen Klammern zwei Namen getrennt durch ein Leerzeichen schreiben: `[Linie_1b Linie_2a]`. Außerdem dürfen Linien auch denselben Namen haben – unterschieden werden sie dann durch eine (bei 1 beginnende) fortlaufende Zahl.

Grid-Template-Areas

Ein sehr wichtiger Punkt bei CSS-Grids ist die Erkenntnis, dass das Grid nur einen gedanklichen Raster über den Container legt. Das bedeutet, dass die durch die Grid-Lines begrenzten Grid-Cells nur im ersten Moment die eigentlichen Plätze für die Grid-Items darstellen – genau das haben wir im letzten Beispiel auch gesehen. Arbeitet man jedoch mit Grid-Areas (also der Zusammenfassung mehrerer Grid-Cells zu einem gemeinsamen Bereich (engl. *area*)), so stellt die Area den Platz für den Inhalt bereit.

Eine solche Area definiert man über die Eigenschaft `grid-template-area`:

```
.grid {
  ...
  grid-template-areas:
    "header header header header header"
    "nav nav nav nav nav"
    ". adv1 main adv2 ."
    ". adv1 nav2 adv2 ."
    "footer footer footer footer footer";
}
```

Folgende Areas werden definiert:

- `header`: Diese Area erstreckt sich über fünf Cells (der ersten Row).
- `nav`: erstreckt sich ebenfalls über fünf Cells (der zweiten Row).
- `main`: definiert die dritte Cell der dritten Row.
- `adv1`: definiert die zweite Col der dritten und vierten Row.
- `adv2`: definiert die vierte Col der dritten und vierten Row.
- `footer`: erstreckt sich über alle Cells der letzten Row.

Vier Cells werden hierbei ausgelassen: Diese werden mit einem `.` gekennzeichnet und bleiben bei der Belegung durch Inhalte unberücksichtigt, sofern keine weiteren Cells benötigt werden.

Die Items selbst können alsdann durch Angabe von `grid-area` in die jeweiligen Areas platziert werden:

HTML-Code:

```
<div class="grid2">
  <header>Seiten-Header</header>
  <nav>Navigation</nav>
  <main>Hauptbereich</main>
  <nav>Footer-Navigation</nav>
```

```
  <footer>Seiten-Footer</footer>
  <aside>Werbung 1</aside>
  <aside>Werbung 2</aside>
</div>
```

CSS-Code:

```
.grid2 {
  display:grid;
  grid-template-columns:1fr 10em minmax(fit-content,71.25rem) 10em 1fr;
  grid-template-rows:8em 3em auto 3em 10em;
  grid-template-areas:
    "header header header header header"
    "nav nav nav nav nav"
    ". adv1 main adv2 ."
    ". adv1 nav2 adv2 ."
    "footer footer footer footer footer";
}
.grid2 header {
  grid-area:header;
}
.grid2 nav:first-of-type {
  grid-area:nav;
}
.grid2 nav:last-of-type {
  grid-area:nav2;
}
.grid2 main {
  grid-area:main;
}
.grid2 footer {
  grid-area:footer;
}
.grid2 aside:first-of-type {
  grid-area:adv1;
}
.grid2 aside:last-of-type {
  grid-area:adv2;
}
```

Wenn nun Elemente hinzukommen, denen kein Platz zugewiesen wurde, werden diese in die mit Punkten gekennzeichneten Cells gelegt, wie Abbildung 6.75 zeigt.

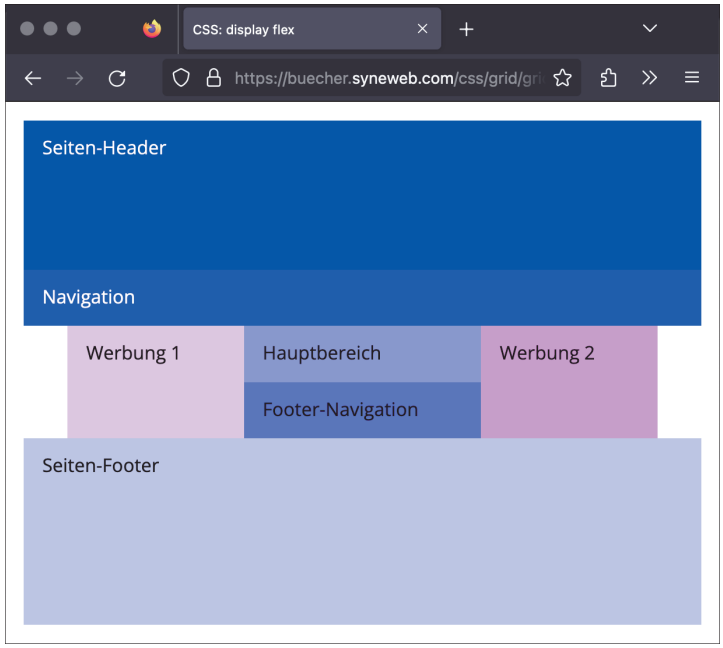


Abbildung 6.74 Ein Grid mit definierten Areas. Zwei Plätze sind frei geblieben.

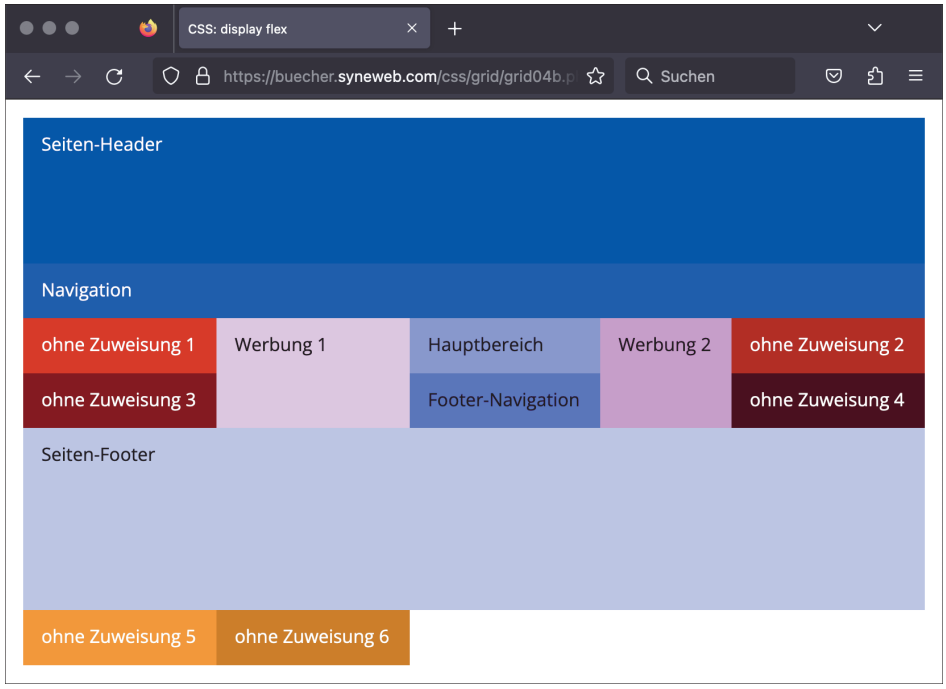


Abbildung 6.75 Die unbenannten Plätze werden von den Elementen belegt, denen keine Area zugewiesen wurde.

Was wäre aber nun, wenn keine Plätze mehr frei sind, weil es mehr Grid-Items als Plätze (Areas plus freie Cells) gibt? Nun, dann würde eine weitere Zeile eröffnet werden, in der die restlichen Items platziert werden (siehe Abbildung 6.76).

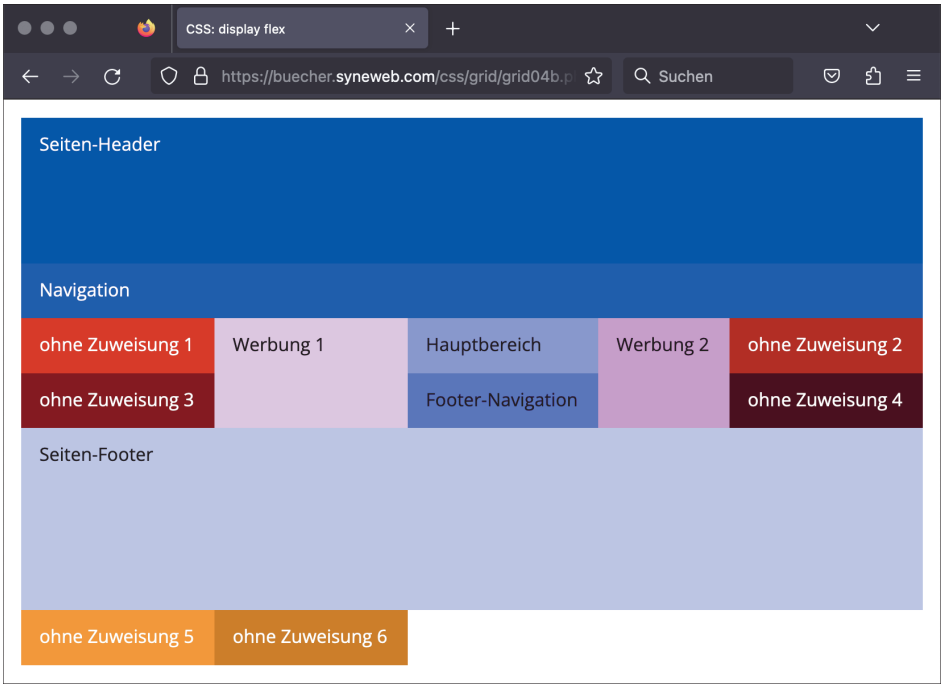


Abbildung 6.76 Elemente, die keinen Platz mehr finden, eröffnen neue Cells und somit gegebenenfalls auch Rows

Grid-Items versus Grid-Cells/Grid-Areas

Nach allem, was wir bisher gesehen haben, ist es vernünftig anzunehmen, dass ein Grid-Item den gesamten Platz, den ihm eine Grid-Cell bzw. eine Grid-Area zur Verfügung stellt, auch einnimmt. Dies muss jedoch nicht der Fall sein, denn ausschlaggebend hierfür sind die Eigenschaften `justify-items` und `align-items` (ähnlich wie bei den Flexboxen – vergleiche hierzu Abschnitt 6.15, »Listenartiges Design mit CSS«). Da für beide Eigenschaften gilt, dass der Standardwert `stretch` ist, sieht es so aus, als würde der gesamte Platz immer ausgefüllt werden.

Die möglichen Werte sind:

- `justify-items: start | end | center | stretch;`
- `align-items: start | end | center | stretch | baseline;`

Würde man etwa `justify-items:start;` und `align-items:center;` ansetzen, so sähe das obige Beispiel wie in Abbildung 6.77 aus.

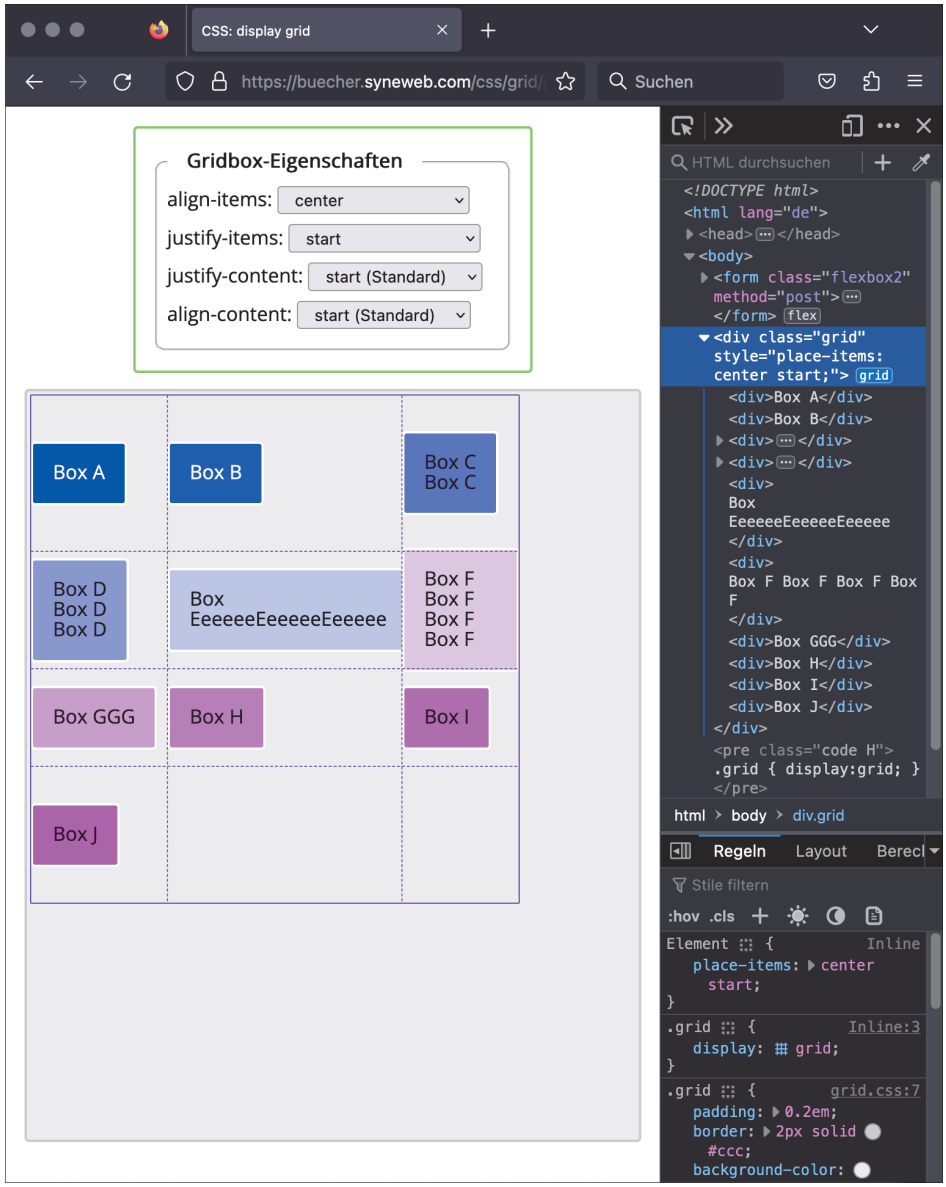


Abbildung 6.77 Bei eingeblendetem Grid (Tipp: Entwicklerkonsole • Grid-Element anwählen und auf »grid« klicken) sieht man gut, wie sich die Grid-Items bei den ausgewählten Eigenschaften verhalten.

Diese Spielwiese zum Testen der Grid-Eigenschaften finden Sie wie auch im Falle der Flexboxen im Downloadbereich des Buchs unter *Tools*.

einem Dokument aus (üblicherweise ganz am Anfang des Codes), so gilt diese Einstellung ab der Zeile, in der sich dieser Befehl befindet. Nicht selten legen Programmierer diese Einstellungen *global* fest. Das bedeutet, dass sie ein zentrales Dokument erzeugen, das von allen anderen Dokumenten eingebunden wird (ähnlich wie die Verlinkung auf gemeinsam zu nutzende CSS-Dateien mittels `<link rel="stylesheet" href="...">` oder der `@import`-Direktive). In dem zentralen Dokument nehmen sie dann diese Einstellungen vor. Wie dies genau geschieht, sehen Sie in Abschnitt 11.8.

Welche Fehler (gemessen an deren Schwere) dann tatsächlich auch *darstellt* werden, entscheidet wiederum der Befehl `error_reporting` in PHP. Hier stehen viele Möglichkeiten zur Verfügung, wobei man üblicherweise *während der Entwicklung alle Fehler* angezeigt bekommen möchte (`error_reporting(E_ALL);`), *im Live-Betrieb* einer Seite jedoch *keinen* (`error_reporting(0);`). Eine vollständige Auflistung aller Fehlerausgaben finden Sie in der offiziellen Dokumentation von PHP unter *https://www.php.net/manual/de/errorfunc.constants.php*.

Zusammenfassend kann man also sagen:

- Während der Entwicklung:

```
ini_set("display_errors",1);
error_reporting(E_ALL);
```

- Im Live-Betrieb einer Website:

```
ini_set("display_errors",0);
error_reporting(0);
```

Fehler-Konstanten

`E_ALL` ist eine Konstante, wobei allen Fehlerkonstanten ein numerischer Wert zugewiesen ist. Beispielsweise ist `E_ALL` der Wert `32767` zugewiesen, `E_NOTICE` etwa der Wert `8`. Unter dem Wert `0` versteht man die Ausgabe keines Fehlers.

11.6 Die Unterschiede in der Schreibweise von JavaScript und PHP

Weil die erste Programmiersprache, mit der Sie bei der Webentwicklung üblicherweise in Kontakt kommen, JavaScript ist, lohnt sich ein Vergleich der Schreibweise von JavaScript und PHP. In den folgenden Abschnitten stelle ich also die Unterschiede zwischen den beiden genannten Sprachen dar, wobei ich von PHP ausgehe.

Im Weiteren gehe ich davon aus, dass Sie sich mit den Grundlagen der Programmierung im Rahmen von JavaScript beschäftigt haben, dass Sie mit den Begriffen *Variable*, *Bedingung*, *Schleife*, *Array*, *Objekt* und *Funktion* vertraut sind und dass Sie bereits ein gewisses Maß an Erfahrung in der Programmierung mitbringen.

11.6.1 Variablen und Konstanten

Zunächst gilt, dass auch PHP keine explizite Typdeklaration erfordert. Das bedeutet also, dass einer Variablen kein fest definierter Wertetyp (wie eine Ganzzahl (`int`), eine Kommazahl (`float`), ein Text (`string`) oder dergleichen) zugewiesen werden muss. Das halten erfahrene Programmierer oft für den schwerwiegendsten Nachteil, den PHP mit sich bringt, ist jedoch der Umgebung geschuldet, in der sich PHP bewegt: Bei der Webentwicklung hat man es grundsätzlich immer mit Textwerten zu tun, da die Web-Welt textbasiert ist. (Beispielsweise sind sämtliche HTML-Formularfeldinhalte eigentlich immer nur Text, auch wenn sie z. B. als `<input type="number">` oder `<input type="date">` deklariert sind.) Aus diesem Grund ist es oftmals notwendig, eine Umwandlung von einem Wertetyp in einen anderen Wertetyp vorzunehmen (z. B. von Text in ein Datum), *nachdem* der Wert der Variablen zugewiesen wurde. Würde man sich jedoch explizit auf einen Wertetyp für eine Variable festlegen, so wäre das nicht möglich:

- Eine Variablendeklaration mit `var` bzw. `let` existiert in PHP nicht: Variablen werden *implizit* deklariert, indem man einer Variablen einen Wert zuweist. Lediglich bei Funktionen und Klassen gibt es Ausnahmen.
- Jeder Variablenname beginnt mit einem `$`-Zeichen.
- Die Zeichenverkettung (*Konkatenation*) wird in PHP nicht mit einem `+` (Pluszeichen), sondern mit einem `.` (Punkt) realisiert.
- In PHP existiert kein globaler Gültigkeitsbereich für Variablen. Benötigen Sie globale Variablen, so müssen Sie sich behelfen, indem Sie Variablen innerhalb von z. B. Funktionen als global betrachten. Dies wird mit der Angabe `global` innerhalb von Funktionen realisiert; alternativ steht das Array `$GLOBALS` zur Verfügung, in dem globale Werte abgelegt werden können.

Bitte beachten Sie, dass in PHP keine *explizite* Variablendeklaration (mittels `var`/`let`) existiert, sondern dass eine Variable *implizit* (d. h. bei ihrer Verwendung) deklariert wird:

JavaScript	PHP

`let a = "Uwe", b = "Mutz", c;`
`c = a + " " + b; //"Uwe Mutz"`
	`$a = "Uwe";`
	`$b = "Mutz";`
	`$c = $a . " " . $b; //"Uwe Mutz"`

11.6.2 Bedingungen

Bedingungen (`if`/`switch`) werden in JavaScript und PHP identisch verwendet. Somit ergeben sich an dieser Stelle keine Unterschiede.

11.6.3 Schleifen

Auch hier entdecken wir im Großen und Ganzen keine Unterschiede: Sowohl die bewährte `for`-Schleife als auch die `while`- und die `do-while`-Schleife gibt es in PHP. Eine Änderung ergibt sich jedoch für die `for-in`-Schleife: Während Sie in JavaScript mit einer `for-in`-Schleife arbeiten, definiert PHP die `foreach`-Schleife. Eine solche Schleife wird immer dann verwendet, wenn man von einem gegebenen Konstrukt (Array, Objekt etc.) sämtliche Informationen auslesen möchte – ganz im Sinne einer `for`-Schleife, die über alle Einträge eines Arrays oder Objekts läuft.

Es bleibt jedoch nicht bei einer simplen Änderung des Namens von `for-in` in `foreach`, sondern auch die Verwendung der Schleifen ist etwas unterschiedlich, wie das folgende Beispiel zeigt.

JavaScript

```
let person = {
  vn: "Uwe",
  nn: "Mutz"
};
for(const eintrag in person) {
  console.log("Eintrag: " + eintrag +
", Wert: " + person[eintrag]);
}
```

PHP

```
$person = [
  "vn" => "Uwe",
  "nn" => "Mutz"
];
foreach($person as $eintrag=>$wert) {
  echo('Eintrag: ' . $eintrag . ',
Wert: ' . $wert . '<br>');
}
```

Wie Sie sehen, ist die Herangehensweise etwas anders. Ich werde in Abschnitt 11.13, »Formulare aufgebohrt: Datei-Upload«, auf die `foreach`-Schleife zurückkommen.

11.6.4 Arrays und Objekte

Die größten Unterschiede in der Schreibweise von PHP und JavaScript bestehen bei Arrays und Objekten. Wie Sie bereits wissen, gibt es in PHP keine explizite Deklaration, weshalb ein `var` bzw. `let` entfällt. Die zwei Möglichkeiten zur Array-Deklaration, die Sie in JavaScript kennengelernt haben, die Objekt- und die Literalschreibweise, gibt es auch in PHP – wenngleich die Objektschreibweise mit `array` etwas anders ist als diejenige in JavaScript mit `new Array`: `$arr = [ ]` oder `$arr = array()`.

Das, was wir in JavaScript unter Objekten verstehen, wird in PHP als *assoziative Arrays* bezeichnet. Arrays dürfen in PHP somit zusätzlich auch *Stringschlüssel* anstatt Indizes haben.

Weiters gilt:

- Ans Ende von Arrays kann sehr einfach ein neuer Wert geschrieben werden, indem man den Array-Namen mit hinten anstehenden eckigen Klammern schreibt:

`$arr[] = 7;` fügt am Ende des Arrays `$arr` einen neuen Eintrag (mit dem Wert 7) hinzu. In JavaScript hätte man hierzu die Funktion `push` benötigt; alternativ kann in PHP aber auch die Funktion `array_push` verwendet werden.

- Die Eigenschaft der Länge eines Arrays aus JavaScript (`arr.length`) wird in PHP durch die Funktion `count($arr)` ersetzt. Dabei ist zu beachten, dass `count` lediglich die benutzten Stellen im Array zählt, JavaScript jedoch auch die unbenutzten Stellen. Betrachten Sie hierzu die Beispiele weiter unten.
- In PHP unterscheidet man zwischen indizierten und assoziativen Arrays:
  - **Indizierte Arrays** sind solche, die die Einträge im Array (beginnend mit 0) durchnummerieren.
  - **Assoziative Arrays** wiederum kennen wir als die Objekte in JavaScript, in denen die Einträge nicht nummeriert, sondern benannt werden.

Arrays in PHP dürfen auch *gemischt assoziativ* sein. Das bedeutet, dass ein Array sowohl nummerierte als auch benannte Einträge besitzen darf.

JavaScript

```
let arr0 = new Array();
let arr1 = [];
arr1.push(17);
arr1[4] = 9;
```

PHP

```
$arr0  = array();
$arr1 = [];
$arr1[] = 17; // ans Ende des Arrays arr1 wird der
// Wert 17 angehängt
$arr1[4] = 9;
```

Umgemünzt in ein Beispiel, ergäbe dies den Inhalt von Abbildung 11.4, in der die Arrays grafisch dargestellt ausgegeben werden.

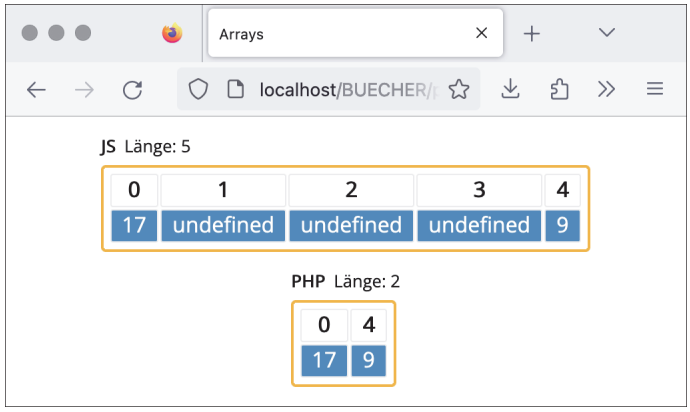


Abbildung 11.4 Arrays in JavaScript und PHP verhalten sich ein wenig unterschiedlich.

Wie Sie gut sehen können, zählt JavaScript auch die unbenutzten Stellen im Array (Länge = 5), in PHP gibt es solche unbenutzten Stellen jedoch gar nicht (weshalb die `count`-Funktion auch nur zwei Stellen zählt; siehe Abbildung 11.5).

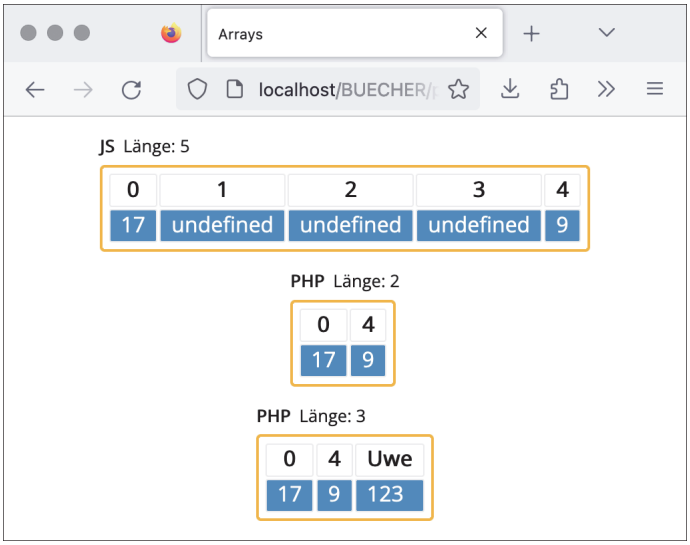


Abbildung 11.5 Gemischt-assoziative Arrays sind nur in PHP möglich – in JavaScript müssen Sie hierzu auf Objekte zurückgreifen.

Fügt man in PHP auch noch einen assoziativen Eintrag ein, so ergibt sich ein gemischt-assoziatives Array:

```
$arr1["Uwe"] = 123; //an der Stelle "Uwe" steht nun der Wert 123
```

Beachten Sie bitte, dass man in PHP nicht sagen kann, dass die assoziativen Einträge vor oder nach den indizierten Einträgen erfolgen – eine Reihenfolge hierfür gibt es einfach nicht.

Genauso wie in JavaScript dürfen Arrays in PHP bereits vorab mit Werten befüllt werden:

```
$arr  = [47,-9,false,"Uwe",11];
```

Beachten Sie den Wert `false`, den JavaScript und PHP unterschiedlich handhaben: In der Ausgabe von `false` auf dem Bildschirm ist dies in PHP ein leeres Wort, in JavaScript erscheint jedoch der Text »false« (siehe Abbildung 11.6).

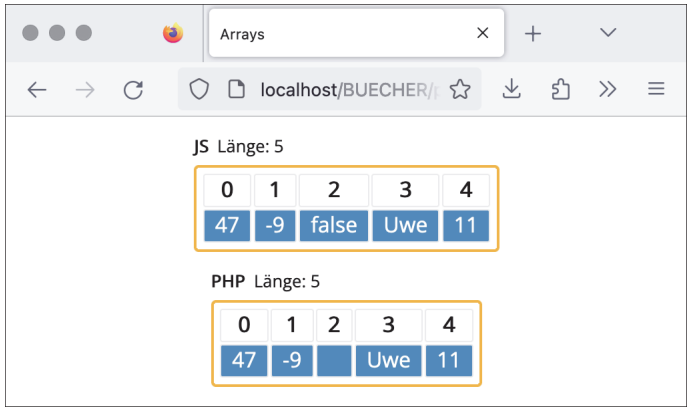


Abbildung 11.6 Vorab befüllte Arrays sind natürlich auch in PHP möglich.

Assoziative Arrays werden vorab wie folgt befüllt:

JavaScript (Objekt)

```
let obj = {
  Vorname: "Uwe",
  Nachname: "Mutz",
  Startnummer: 47
};
```

PHP (assoziatives Array)

```
$arr  = [
  "Vorname" => "Uwe",
  "Nachname" => "Mutz",
  "Startnummer" => 47
];
```

Beachten Sie die Anführungszeichen bei der Benennung der Einträge in PHP. Abbildung 11.7 zeigt die Ausgabe auf dem Bildschirm.

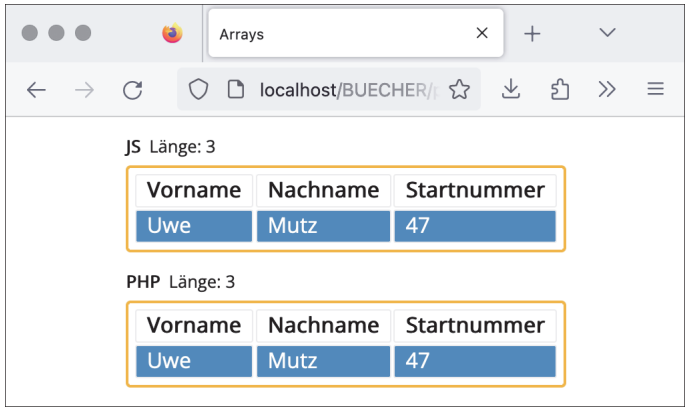


Abbildung 11.7 Assoziative Arrays in PHP und Objekte in JavaScript können gut miteinander verglichen werden, wenngleich sie auch nicht ganz dasselbe sind.

Die Verwendung mehrdimensionaler Arrays ist natürlich auch möglich.

11.6.5 Funktionen

Funktionen sind in PHP und JavaScript aus Sicht der Deklaration sehr ähnlich (jede Sprache hat so ihre Feinheiten). Jedoch haben Sie in JavaScript das `arguments`-Array kennengelernt, das in PHP nicht existiert (eine Funktion, die ähnlich arbeitet, ist die Funktion `func_get_args()`). Da Sie hierfür jedoch bereits die Rest-Parameter in JavaScript kennengelernt haben, ist das zu verkraften, denn diese Parameter (im Falle von PHP nennt man sie *Variable-length Argument List*) samt der zugehörigen Schreibweise mit den drei Punkten (*Splat-Operator*) finden wir auch in PHP.

Eine Sache unterscheidet PHP jedoch stark von JavaScript: PHP verwendet per se keine globalen Variablen (es sei denn, sie sind explizit über das `GLOBALS`-Array definiert, auf das wir noch stoßen werden), weshalb Variablen in PHP-Funktionen immer lokal sind. In manchen Fällen ist es aber dennoch notwendig, auf Variablen zuzugreifen, die »außerhalb der Funktion« definiert wurden. Dies erfolgt dann über das Schlüsselwort `global` innerhalb der Funktion:

JavaScript

```
var vn = "Uwe", nn = "Mutz";
function person_show() {
  console.log("Name: " + vn + " "
+ nn);
}
person_show();
```

PHP

```
$vn = "Uwe";
$nn = "Mutz";
function person_show() {
  global $vn;
  global $nn;
  echo("Name:" . $vn . " " . $nn);
}
person_show();
```

Ein weiterer – toller! – Unterschied zu JavaScript ist die Möglichkeit der Typdeklaration von Funktionsparametern und des Rückgabewertes. Diese Möglichkeit stand zuerst in PHP 7 zur Verfügung und wurde danach mit PHP 8 vollständig eingeführt. Werfen wir also einen Blick auf die in PHP möglichen Daten-/Wertetypen:

- `null`: leerer Wert
- `void`: kein Wert
- skalare Datentypen:
  - `bool`: Wahrheitswert
  - `int`: Ganzzahl
  - `float`: Kommazahl
  - `string`: Text
- `array`: assoziatives, indiziertes oder gemischt-assoziatives Array

- `object`: Standardklasse, die mit `new stdClass()` instanziiert wird
- `resource`: Speicherquelle (wird beispielsweise bei der Bildgenerierung benötigt)

Eine weitere mögliche Deklaration ist der gemischte Datentyp `mixed`, bei dem mehrere Wertetypen erlaubt sind.

Datentypen in aller Tiefe

Ich muss an dieser Stelle ehrlich mit Ihnen sein: Wir gehen hier nicht in die Tiefe, sondern bewegen uns eher an der Oberfläche. Zu Datentypen gibt es wesentlich mehr zu sagen, als es zu diesem Zeitpunkt notwendig ist. Mein Ziel ist es, dass Sie einen guten Einstieg in die Thematik finden – Profi können Sie immer noch werden.

Warum ist diese Typdeklaration nun so toll? Die Antwort ist relativ einfach: Je stärker man sich in der Programmierung einschränkt, umso weniger Fehler kann man machen. Wenn wir also eine Variable so deklarieren können, dass sie beispielsweise nur Zahlenwerte aufnehmen darf, so würde die Zuweisung eines Strings sofort zu einem Fehler (einem sogenannten *Type Error*) führen. Oder wenn wir beispielsweise eine Funktion definieren wollen, die als Ergebnis (Rückgabewert) eine Zahl liefert, so darf diese nicht fälschlicherweise einen booleschen Wert retournieren, der eventuell unsere weitere Programmierung fehlerhaft machen würde. Denken Sie immer daran: Weder Sie noch ich sind fehlerfrei in unserer Programmierung. Fehlerfreiheit ist ein Wunschtraum, der nie in Erfüllung geht.

Betrachten wir also ein erstes Beispiel: Wir wollen eine Funktion definieren, die zwei Zahlen addiert und die Summe der beiden Zahlen zurückgibt. Nach unserem bisherigen Wissen würden wir die Funktion wie folgt definieren und aufrufen:

```
function addiere($a,$b) {
  $c = $a+$b;
  return $c;
}
$erg0 = addiere(2,3);
echo('<p>Das Ergebnis der Addition von 2 und 3 ist ' . $erg0 . '</p>');
```

Es ist völlig klar, wie das Ergebnis aussieht (siehe Abbildung 11.8).

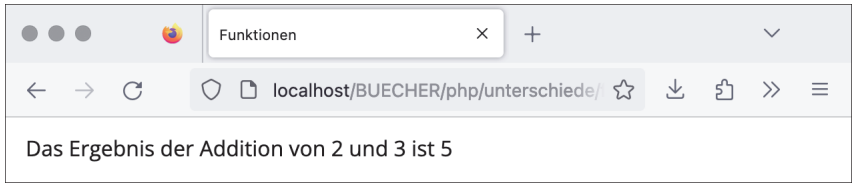


Abbildung 11.8 Zahlen addieren, »the easy way«: 2 plus 3 ergibt 5.

Was aber wäre, wenn wir beim Funktionsaufruf keinen Wert oder einen Textwert als zweiten Übergabeparameter angeben? Probieren wir's aus:

```
$erg1 = addiere(2,null);
echo('<p>Das Ergebnis der Addition von 2 und null ist ' . $erg1 . '</p>');
$erg2 = addiere(2,"Uwe");
echo('<p>Das Ergebnis der Addition von 2 und "Uwe" ist ' . $erg2 . '</p>');
```

Beim ersten Aufruf übergeben wir den Wert `null` (»leerer Wert«), beim zweiten Aufruf den Textwert »Uwe«. Beides sind keine Zahlen, die wir für die Addition verwenden können – das Ergebnis ist jedoch unterschiedlich, wie Abbildung 11.9 zeigt.

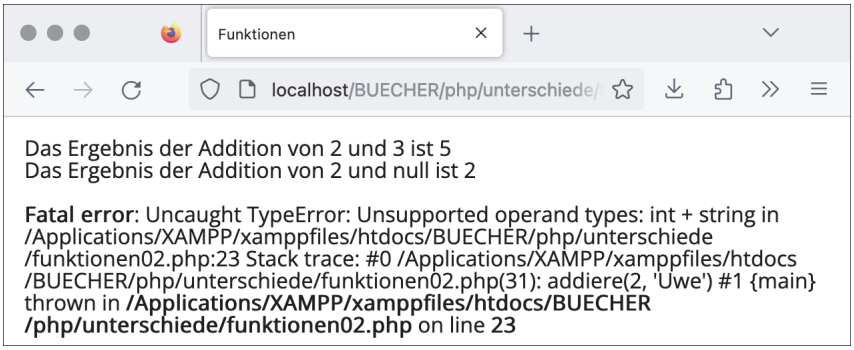


Abbildung 11.9 Je nach Übergabewert arbeitet unsere Funktion unterschiedlich, aber beide Male nicht so, wie wir es uns wünschen.

Im Klartext bedeutet das: Unsere Funktion hat die beiden Werte »ohne nachzudenken« übernommen und hat versucht, diese miteinander zu addieren. Dies kann jedoch zu ungewollten Ergebnissen führen, wie der erste Aufruf mit null zeigt. Dass der zweite Aufruf dann schlussendlich auch einen Fehler generiert hat, ist positiv und gut für uns. Dennoch wollen wir schon allein den Versuch der Addition unterbinden, sollten die Typen nicht passen. Genau das erreichen wir, indem wir Typen deklarieren.

Eine solche Typdeklaration erfolgt in der Funktionsdefinition an der Stelle, wo wir die Übergabevariablen benennen: einfach den gewünschten Typ (aus der obigen Liste) voranstellen, und gut ist es. Selbiges können wir mit einem vorangestellten Doppelpunkt für den Rückgabewert vornehmen. In unserem Fall erlauben wir demnach (Ganz- und) Kommazahlen als Übergabewerte und generieren eine Kommazahl als Rückgabewert. Beachten Sie an dieser Stelle bitte, dass der Kommazahlentyp `float` natürlich auch mit ganzen Zahlen (sogenannten »Integer-Zahlen«) arbeiten kann, da eine Ganzzahl ja im Wesentlichen nichts anderes ist als eine Kommazahl ohne Kommastellen:

```
function addiere(float $a, float $b):float {
  $c = $a+$b;
  return $c;
}
```

Rufen wir unsere abgeänderte Funktion wie bereits zuvor auf, so erhalten wir das Ergebnis aus Abbildung 11.10.

```
$erg0 = addiere(2,3);
echo('<p>Das Ergebnis der Addition von 2 und 3 ist ' . $erg0 . '</p>');
$erg1 = addiere(2,null);
echo('<p>Das Ergebnis der Addition von 2 und null ist ' . $erg1 . '</p>');
$erg2 = addiere(2,"Uwe");
echo('<p>Das Ergebnis der Addition von 2 und "Uwe" ist ' . $erg2 . '</p>');
```

Listing 11.2 Dieses Listing erzeugt beim Aufruf der Funktion »addiere()« einen Fatal Error.

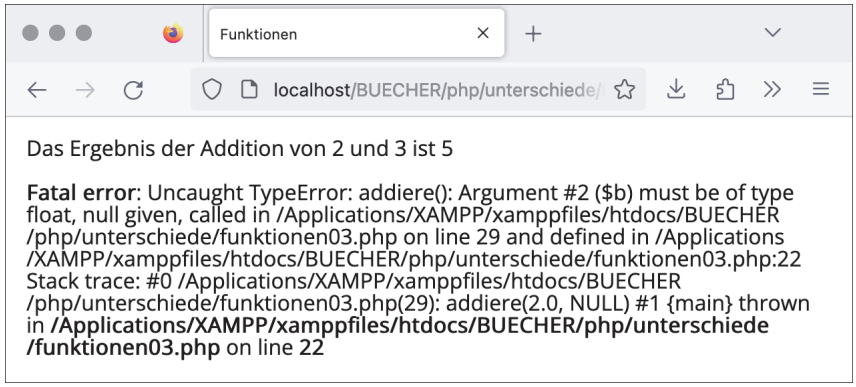


Abbildung 11.10 Unsere abgeänderte Funktionsdefinition bewirkt, dass die Funktion bereits beim ersten fehlerhaften Aufruf einen entsprechenden (Fatal) Error erzeugt.

Wie Sie sehen, bricht unsere Funktion nun bereits beim ersten fehlerhaften Aufruf ab. Genau so soll das auch sein, da wir es vermeiden wollen, dass die Funktion mit falschen Wertetypen versucht, die Addition vorzunehmen. Ganz allgemein gesagt sind Typdeklarationen also ein Schutz für uns selbst, damit wir keine (oder weniger) Fehler machen.

In manchen Fällen kann es notwendig sein, dass man mehrere mögliche Datentypen erlaubt. Insbesondere im Fall von Rückgabewerten ist das oft der Fall: Beispielsweise kann eine Funktion so definiert werden, dass sie entweder das tut, wofür sie erdacht wurde (z. B. ein Array sortieren), oder den Wert `false` zurückliefert (sollte das Array beispielsweise leer sein). Um mehrere Datentypen zu erlauben, trennt man diese durch einen senkrechten Strich (»Pipe«) voneinander:

```
//erwartet als Übergabewert eine Ganzzahl oder einen String und liefert als
//Ergebnis einen String oder einen booleschen Wert zurück
function test(int|string $a):string|bool { ... }
```

Sollte man wirklich nicht definieren können, welcher Datentyp an eine Funktion übergeben werden muss (ja, das kann in der Tat der Fall sein), so bedient man sich des `mixed`-Typs:

```
//erwartet als Übergabewert einen beliebigen Typ und liefert als Ergebnis eine
//Kommazahl zurück
function test(mixed $a):float { ... }
```

Bitte beachten Sie, dass `mixed` bereits (fast) alle Datentypen einschließt, weshalb eine Angabe wie etwa `function test(mixed|int $a):void { ... }` nicht erlaubt wäre.

Ein Datentyp sei noch besonders erwähnt: der Datentyp `void`. Dieser kommt immer dann zum Einsatz, wenn eine Funktion keinen Wert zurückliefert (was ja nicht selten der Fall ist). Die folgende (zugegebenermaßen sehr einfache) Funktion verbindet zwei Strings und gibt dies mittels `echo` auf dem Bildschirm aus – einen Rückgabewert sucht man jedoch vergeblich:

```
function test(string $a, string $b):void {
  echo($a . $b);
}
```

11.6.6 Konstanten und globale Variablen

Die Syntax zur Definition einer Konstanten lautet:

```
define("KONSTANTENNAME",Wert);
```

Die Verwendung einer Konstanten erfolgt genauso wie die Verwendung einer Variablen, jedoch mit dem Unterschied, dass Konstanten kein führendes `$`-Zeichen besitzen:

```
define("PI",3.1415);
$r = 20;
$umfang = 2*$r*PI;
```

define vs. const

Ähnlich wie in JavaScript können Konstanten in PHP auch mit `const` deklariert werden. Dies ist jedoch nur für die skalaren Datentypen erlaubt (`int`, `float`, `string`, `bool` – vergleiche die Auflistung der Datentypen aus dem vorangegangenen Abschnitt). Die Konstantendeklaration mittels `define` ist jedoch nicht auf die skalaren Datentypen beschränkt, weshalb ihr in der Regel der Vorzug gegeben wird.

Konstanten besitzen einen globalen Gültigkeitsbereich. Neben den Konstanten existieren auch noch globale Variablen, die in einem (assoziativen) Array `$GLOBALS` verwaltet werden. Um diesem Array einen neuen Eintrag hinzuzufügen, wird folgender Code benötigt:

```
$GLOBALS["Variablenname"] = Wert;
```

Man spricht im Falle eines solchen Array-Eintrags auch von einer *globalen Variable*.

Hierzu ein Beispiel:

```
$GLOBALS["erlaubeLeerePasswoerter"] = true;
```

Das Array `$GLOBALS` ist (selbstredend) ein globales Array, das somit wiederum von überall aus verfügbar ist. Der Unterschied zwischen globalen Variablen und Konstanten ist vor allem der Fakt, dass globale Variablen veränderbar sind, Konstanten jedoch nicht.

Sie werden in weiterer Folge noch auf einige ähnliche Arrays stoßen, die dieselben Eigenschaften (assoziativ, global) aufweisen – diese werden wir *Superglobals* nennen. Beispiele hierzu sind etwa die Arrays `$_POST`, `$_GET` und `$_REQUEST` aus Abschnitt 11.10, `$_SESSION` aus Abschnitt 11.12, `$_FILES` aus Abschnitt 11.14 sowie weitere Arrays wie `$_SERVER`, `$_ENV` usw. Einen vollständigen Überblick über alle Superglobals finden Sie unter *http://php.net/manual/de/language.variables.superglobals.php*.

11.7 Trial and Error: try-catch

Bereits in JavaScript haben Sie gesehen, dass Funktionen nicht immer nur mittels boolescher Rückgabewerte einen Fehler zum Ausdruck bringen, sondern dass sie auch *Exceptions* erzeugen, die wir wie folgt verarbeiten können:

- Wir könnten eine individuelle Fehlermeldung generieren.
- Wir könnten den aufgetretenen Fehler in ein Fehler-Log speichern und nicht ausgeben.
- usw.

Hier eröffnen sich viele Möglichkeiten. Die einzige Einschränkung ist, dass nicht jede Funktion auch eine Ausnahme erzeugt. Auf der anderen Seite können wir zu jeder Zeit auch manuell eine Ausnahme erzeugen.

Der folgende Code zeigt das Prinzip:

```
mysqli_report(MYSQLI_REPORT_ERROR | MYSQLI_REPORT_STRICT); //aktiviert die
//Möglichkeit, dass MySQL-Fehler eine Exception erzeugen
try {
```

Inhalt

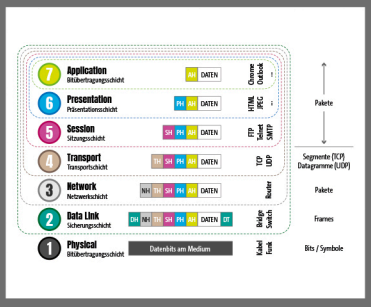
7 JavaScript – die logische Abteilung auf der Clientseite 415

Webseiten programmieren und gestalten

»Die umfassende Ausbildung«

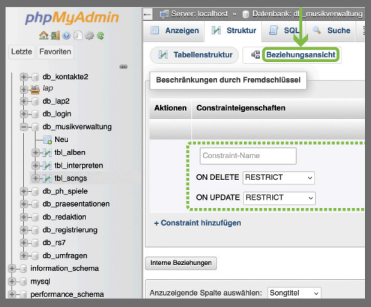
Code & Design fürs Web

Werden Sie zum Profi für gelungene Websites! Dieses Buch bietet Ihnen die vielseitige Ausbildung, die dafür sinnvoll ist. Ob Programmierung für Frontend und Backend, effektive Gestaltung, Usability oder Einbindung von Datenbanken: mit vielen Anwendungsbeispielen, ideal zum Lernen, zum Auffrischen und zum Nachschlagen.



Ausführliche Grundlagen

Nutzerorientiertes Design



Backend-Technologien

Grundlagen zuerst

Beginnen Sie mit einem guten Fundament aus Konzepten rund um Gestaltung und Technologie. Bei der Installation der Arbeitsumgebung nimmt dieses Buch Sie ebenso an die Hand wie beim Verständnis von Datenübertragung, Webserver und Co.

Technologie und Mensch

Farbeinsatz, User Experience, Responsive Webdesign, Inhaltsarchitektur … immer geht es im Kern um die Nutzenden. Eignen Sie sich technisches Know-how an und blicken Sie über den Tellerrand – im Dienste wirklich guter Webentwicklung!

Den Fullstack im Blick

Sie beginnen mit HTML, CSS und JavaScript, den unverzichtbaren Sprachen für das Frontend. Hinzu kommen Datenbanken und PHP, die beliebteste Backend-Sprache. Von »Hallo Welt« bis zur komplexen Webanwendung erweitern Sie nach und nach Ihr Repertoire.

Mit allen Codebeispielen zum Download

Uwe Mutz hat Informatik und Physik studiert, ist Geschäftsführer der SYNE Marketing & Consulting GmbH und Vortragender an der Universität Krems. Er entwickelt Online-Systeme mit Fokus auf Nutzerbedürfnisse.

Aus dem Inhalt

Grundlagen

Arbeitsumgebung einrichten
HTML, CSS, JavaScript
Gestaltungsprinzipien
Grundlagen der Programmierung

Webanwendungen

Usability, UX, Barrierefreiheit
Responsive Webdesign
Programmieren mit PHP
Datenbanken einbinden
Projekte: Bibliotheksverwaltung, Dienstpläne u. v. m.

Technologiethemen

Ein Datenbankschema entwerfen
SQL-Injection verhindern
Verschlüsselungsstandards
Den passenden Webserver wählen

Rheinwerk Computing

Gedruckt in Deutschland
Papier aus nachhaltiger Waldwirtschaft
Mineralölfreie Druckfarben

€ 49,90 [D] € 51,30 [A]

Webseiten
ISBN 978-3-8362-9788-2