Deutsche Hochschule
für Prävention und Gesundheitsmanagement
University of Applied Sciences

Studienbrief

Programmierung II

Fortgeschrittene Programmierung

Inhaltsverzeichnis

1 Datenformate

Lernziele

Nach Bearbeitung des Kapitels *Datenformate* ...

- kennen Sie die gängigsten Datenformate und können diese zur Datenspeicherung und -verarbeitung nutzen.
- kennen Sie den Unterschied zwischen proprietären und standardisierten Datenformaten.
- wissen Sie, was der Unterschied zwischen menschen- und maschinenlesbaren Formaten ist.
- können Sie Daten aus Text-, JSON-, CSV- und XML-Dateien lesen und schreiben.
- kennen Sie die Vor- und Nachteile von CSV, JSON und XML-Dateien.

In diesem Teil lernen Sie, wie digitale Daten dauerhaft gespeichert werden können. Dabei werden Sie verschiedene Dateiformate zur Speicherung von Daten kennenlernen. Darüber hinaus lernen Sie, was ein *Datenbanksystem* ist, welche Vor- und Nachteile es bietet und wie Sie damit arbeiten können.

In *Programmierung 1* haben Sie bereits gelernt, dass ein Computer unterschiedliche Möglichkeiten bietet Daten zu speichern. Diese unterscheiden sich grundsätzlich in der Dauer der Speicherung. Während flüchtige Speichermedien, wie zum Beispiel Arbeitsspeicher, Informationen nur so lange speichern, bis der Computer ausgeschaltet wird, bleiben Daten auf nicht-flüchtigen Medien, wie zum Beispiel Festplatte oder USB-Sticks, auch erhalten, nachdem Sie den Strom ausgeschaltet haben.

Im Codebeispiel *Taschenrechner* (aus *Programmierung 1*) haben Sie Daten im flüchtigen Arbeitsspeicher angelegt. Jede Nutzereingabe sowie die Berechnung und Ausgabe eines Ergebnisses wurden nur temporär gespeichert. Wenn Sie das Programm beenden oder den Computer ausschalten, sind diese Daten für immer verloren. Wollen Sie dauerhaft eine Historie Ihrer Berechnungen speichern, müssen Sie ein nicht-flüchtiges Speichermedium wählen. Im folgenden Kapitel werden Sie verschiedene Möglichkeiten dazu kennenlernen.

2.1.3 Sicherheitsprobleme

Sicherheitsprobleme können dann entstehen, wenn Benutzer verschiedene Rechte zugeteilt bekommen sollen. Mit einfachen Dateien ist es nicht möglich, einem Benutzer nur eingeschränkten Zugriff auf bestimmte Daten zu erlauben. Stellen Sie sich vor, Sie möchten eine weitere Anwendung für Ihre Praktikanten schreiben. Diese sollen die Namen und trainingsrelevanten Daten eines Kunden lesen, allerdings nicht ohne Zustimmung ändern können. Außerdem wäre es besser, wenn nur Sie und die Buchhaltung Zugriff auf Kontoinformationen von Kunden haben.

Alle genannten Probleme lassen sich durch den Einsatz eines sogenannten *Datenbanksystems* lösen oder deutlich verbessern. Im Folgenden werden Sie lernen, woraus ein solches System besteht und wie Sie es einsetzen können.

2.2 Bestandteile eines Datenbanksystems

Ein Datenbanksystem (DBS) wird verwendet, wenn größere Mengen digitaler Daten dauerhaft, effizient und widerspruchsfrei gespeichert werden sollen. Um die Probleme einfacher Dateien zu umgehen, besteht ein Datenbanksystem aus einer Datenbank (DB), also den zu speichernden Daten und einem zusätzlichen Verwaltungsprogramm, das den Zugriff auf diese Daten regelt. Dieses Programm wird auch *Datenbankmanagementsystem* (DBMS) genannt. Für das Lesen und Schreiben von Daten, sowie die Definition eines Datenschemas, bietet ein solches Datenbanksystem eine *Datenbanksprache* an.

2.3 3-Ebenen-Architektur

Die meisten heutigen Datenbanksystem sind nach demselben Schema aufgebaut. Dieses Schema besteht aus drei Ebenen und wird deshalb auch *Drei-Ebenen-Modell* genannt (siehe Abbildung 1). Es wurde vom Standards Planning and Requirements Committee (SPARC) des American National Standards Institute (ANSI) festgelegt und wird deshalb auch als *ANSI-SPARC-Architektur* bezeichnet.

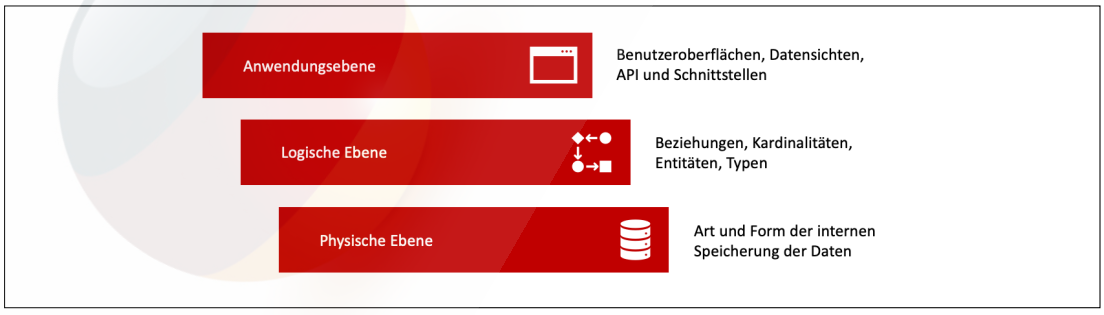


Abbildung 1: Drei-Ebenen-Architektur. (©BSA/DHfPG)

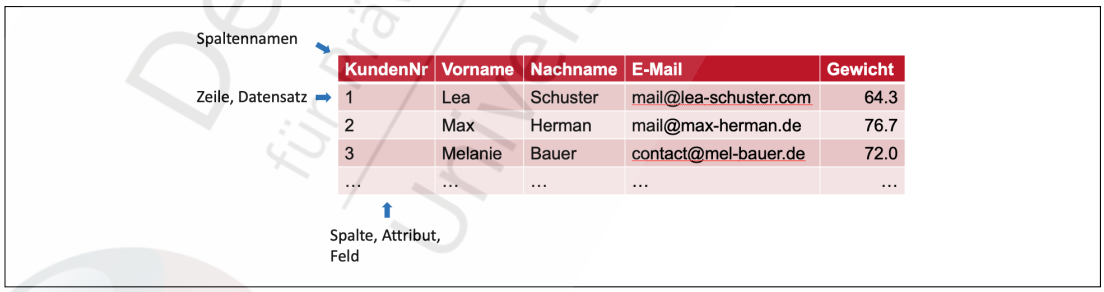
2.4.2 Logische Modelle: relationales Modell

Nachdem mithilfe eines konzeptionellen Modells ein Ausschnitt der realen Welt definiert wurde, der in der Datenbank abgebildet werden soll, kann dieses Modell in ein konkreteres, logisches Modell überführt werden. Diese werden auch *Implementierungsmodelle* genannt und können direkt in Datenbanken abgebildet werden.

Es existieren verschiedene logische Modelle, von denen die meisten keine Anwendung mehr finden. Satzorientierte Modelle wie das *Netzwerkmodell* oder das *hierarchische Modell* wurden bereits vor einiger Zeit durch das einfache, mengenorientierte *relationale Modell* abgelöst. Eine derzeitige Alternative bieten sogenannte *NoSQL-Modelle*, die Sie in Abschnitt 2.4.3 kennenlernen werden.

2.4.2.1 Relationen

Relationale Datenbanken bestehen hauptsächlich aus einfachen, flachen Tabellen. Diese werden auch *Relationen* genannt. Ihnen zugrunde liegt ein mathematisches Modell, das die Werte einer Tabelle als Menge interpretiert. Die sogenannte *relationale Algebra* definiert mögliche Operationen auf diesen Mengen, mit denen Tabellen gefiltert, verknüpft oder verändert werden können. In *Mathematik für Informatik 1* sind Sie bereits in Kontakt mit Relationen und Funktionen gekommen. In diesem Modul *Programmierung 2* werden Sie den Umgang mit relationalen Datenbanken allerdings ohne die mathematische Herleitung erlernen.



KundenNr	Vorname	Nachname	E-Mail	Gewicht
1	Lea	Schuster	mail@lea-schuster.com	64.3
2	Max	Herman	mail@max-herman.de	76.7
3	Melanie	Bauer	contact@mel-bauer.de	72.0
…	…	…	…	…

Abbildung 6: Kundentabelle einer relationalen Datenbank. (©BSA/DHfPG)

Eine Zeile (engl. *row*) in einer Tabelle einer relationalen Datenbank repräsentiert ein gespeichertes Objekt und wird auch Datensatz genannt. Abbildung 6 zeigt die Entität *Kunde* als Tabelle in einem relationalen Modell. Die Kopfzeile der Tabelle gibt die Namen der Spalten (engl. *columns*) oder Attribute an. Möchten Sie einen neuen Kunden zu Ihrem System hinzufügen, so erstellen Sie eine neue Zeile in dieser Tabelle und tragen die Daten in die entsprechenden Felder dieser Zeile ein.

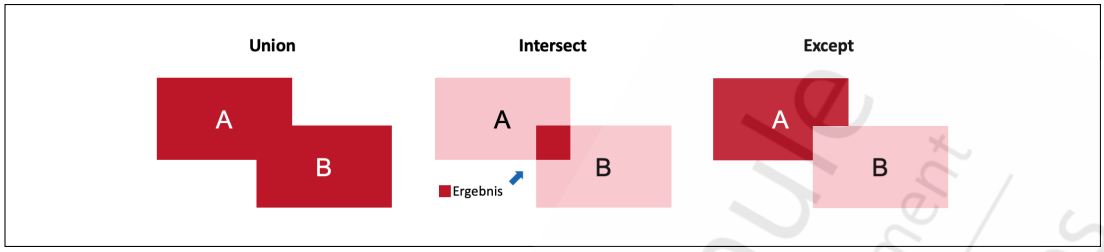


Abbildung 8: SQL-Mengenoperationen. (©BSA/DHfPG)

Mit dem Zusatz `ALL` können alle Mengenoperationen angewiesen werden, Duplikate nicht automatisch aus dem Ergebnis herauszufiltern. Führen Sie beispielsweise zwei Datensätze mit `UNION ALL` zusammen, und beide enthalten die gleiche Kundennummer, dann erhalten Sie einen Datensatz, in dem die Kundennummer zweimal vorkommt. Bei einem normalen `UNION` werden diese doppelten Einträge verhindert.

Datensätze sortieren Mit dem Zusatz `ORDER BY`, können Sie die Datensätze einer SQL-Abfage sortieren. Dabei können Sie eine Spalte angeben, nach der sortiert werden soll. Außerdem können Sie eine Sortierrichtung festlegen. Mit dem Schlüsselwort `ASC` (engl. ascending) können Sie die Einträge aufsteigend sortiert ausgeben lassen. Mit dem Schlüsselwort `DESC` (engl. descending) erhalten Sie eine absteigende Sortierung. Die folgende Abfrage gibt alle Ihre Kunden, aufsteigend sortiert nach Ihren Nachnamen, aus.

```
1  # Alle Kunden nach Nachnamen sortiert
2  SELECT * FROM Kunde
3    ORDER BY nachname ASC;
4
5  # Bei gleichen Nachnamen, nach Vorname sortieren
6  SELECT * FROM Kunde
7    ORDER BY nachname, vorname ASC;
```

Geben Sie mehrere Spaltennamen durch Kommata getrennt an, können Sie außerdem festlegen, wie sortiert wird, wenn zwei Einträge denselben Wert besitzen. Gibt es zwei Kunden mit dem gleichen Nachnamen, wird deren Reihenfolge anhand ihrer Vornamen festgelegt.

Tabellen verbinden mit JOIN Bei der Abfrage von Daten aus Datenbanken ist es meist notwendig Daten aus verschiedenen Tabellen miteinander zu verbinden. Wenn Sie sich eine Liste aller Trainings ausgeben lassen, möchten Sie vielleicht auch wissen, wie der Kunde hieß, der diese Trainingseinheit absolviert hat. Im Normalfall erhalten Sie ja lediglich die in der Trainingstabelle abgespeicherte Kundennummer. In diesem Fall können Sie diese beiden Tabellen bei der Abfrage miteinander verbinden. Dabei erhalten Sie eine neue Tabelle, in der die Spalten beider Tabellen aufgeführt werden. Darunter beispielsweise die Trainingsdetails, aber auch der Vor- und Nachname des Kunden. Dieser Schritt wird auch *Join* genannt und kann auf verschiedenen Wegen erfolgen.

tation[2] da. Immer wenn Sie nicht weiterkommen oder das Gefühl haben, dass es einen einfacheren Weg gibt, etwas zu tun, werfen Sie einen Blick in die Dokumentation und sehen nach, ob es nicht schon eine Routine gibt, die genau das tut, was Sie brauchen.

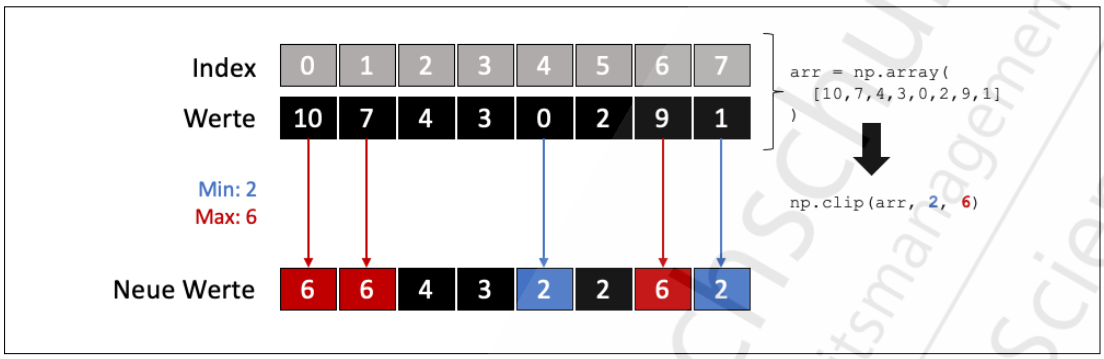


Abbildung 11: Beispiel für np.clip() mit Minimum 2 und Maximum 6. (©BSA/DHfPG)

Hier brauchen Sie eine Funktion, die ein Array als Parameter erhält und sicherstellt, dass die Werte ein bestimmtes Minimum oder Maximum nicht überschreiten - `np.clip()` tut genau das (siehe Abbildung 11). Für das zweite Argument von `np.clip()` übergeben Sie die Punkte und stellen sicher, dass jede neu ‚gebogene' Note nicht niedriger als die ursprüngliche Note ist. Als drittes Argument übergeben Sie jedoch einen einzigen Wert: 100. NumPy nimmt diesen Wert und überträgt ihn auf jedes Element in `neue_punkte`, um sicherzustellen, dass keine der neuen Noten eine perfekte Punktzahl von 100 überschreitet.

5.3 Vectors, Array Shapes und Achsen

Nachdem Sie nun einen ersten Eindruck darüber haben, was NumPy alles kann, ist es an der Zeit, dieses Fundament mit einigen wichtigen Theorien zu festigen. Es gibt ein paar Konzepte, die man im Hinterkopf behalten sollte, besonders wenn man mit Arrays in höheren Dimensionen arbeitet.

Vektoren, also eindimensionale Zahlenreihen, sind am einfachsten zu handhaben. Das mathematische Konzept von Vektoren und Matrizen ist im Detail im Studienbrief *Mathematik für Informatik 1* beschrieben. Zwei Dimensionen eines Arrays oder Vektors sind noch recht einfach zu handhaben, denn sie sind ähnlich wie Tabellenkalkulationen. Aber bei drei Dimensionen, wie zum Beispiel in der 3D-Computergrafik, wird es schon schwieriger, und wenn dann noch beispielsweise bei Animationen die Zeit als vierte Dimension dazukommt, wird es schon äußerst komplex und schwierig dies zu visualisieren. NumPy bietet hier aber Werkzeuge, um zumindest die Verarbeitung von mehrdimensionalen Datenstrukturen zu vereinfachen. Mit der Visualisierung dieser Daten in Form von Diagrammen werden Sie sich dann in einem späteren Kapitel beschäftigen (siehe Kapite 7).

[2] `https://numpy.org/doc/stable/reference/routines.html`

zontale Achse (Achse 1). Viele Funktionen und Befehle in NumPy ändern ihr Verhalten, je nachdem, welche Achse Sie ihnen zur Verarbeitung vorgeben.

Standardmäßig gibt `.max()` den größten Wert im gesamten Array zurück, unabhängig davon, wie viele Dimensionen es hat. Sobald Sie jedoch eine Achse angeben, wird die Berechnung für jeden Satz von Werten entlang dieser Achse durchgeführt. Mit dem Argument `axis=0` wählt `.max()` beispielsweise den Maximalwert in jedem der vier vertikalen Wertesätze des Arrays aus und gibt ein Array zurück, das zu einem eindimensionalen Array (oder Vektor) reduziert oder aggregiert wurde.

Tatsächlich verhalten sich viele der NumPy-Funktionen auf diese Weise: Wenn keine Achse angegeben ist, führen sie eine Operation am gesamten Datensatz durch. Andernfalls führen sie die Operation achsenweise durch.

5.3.3 Numpy-Arrays dynamisch erstellen

Der wichtigste Typ in Numy ist ein Array-Typ namens `ndarray` . NumPy bietet eine Vielzahl von Routinen zur Erstellung von Arrays für verschiedene Situationen. `np.arange()` ist beispielsweise eine solche Funktion, die auf numerischen Bereichen basiert.

Die Argumente von `np.arange()` , die die im Array enthaltenen Werte definieren, entsprechen den numerischen Parametern `start` , `stop` und `step` . Sie müssen mindestens einen von ihnen übergeben. `start` ist die Zahl (ganzzahlig oder mit Nachkommastellen), die den ersten Wert im Array definiert. `stop` ist die Zahl, die das Ende des Arrays definiert und *nicht* in das Array aufgenommen wird. `step` gibt den Abstand (oder Differenz) zwischen zwei aufeinanderfolgenden Werten im Array an. Standardmäßig ist dies 1 und nicht 0. Andernfalls erhalten Sie einen `ZeroDivisionError` .

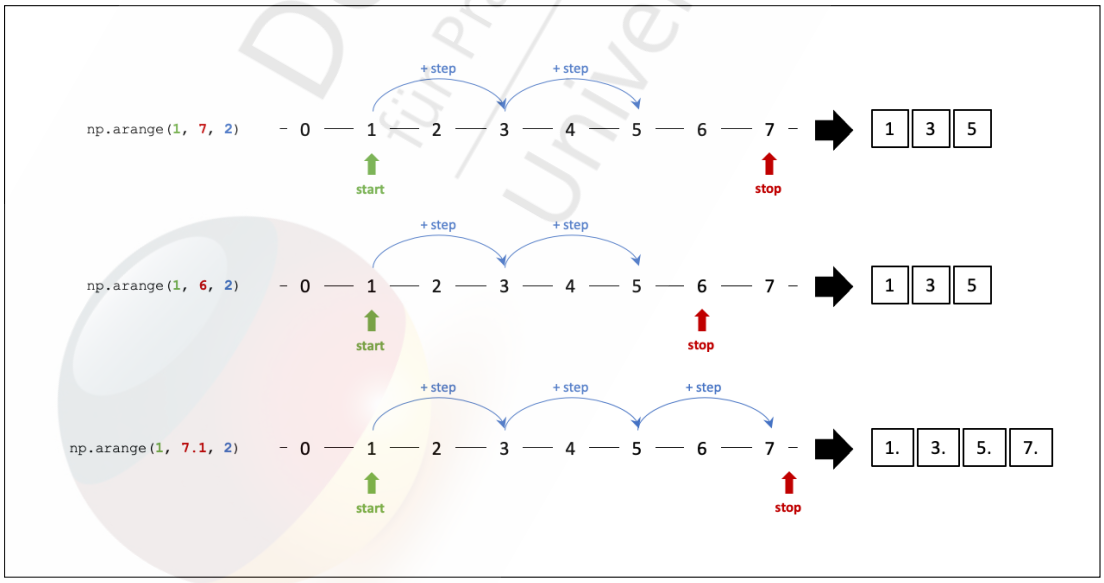


Abbildung 13: Beispiele für `.arange()` mit verschiedenen `stop` -Werten bei gleichen Startpunkten und Schrittgrößen. (©BSA/DHfPG)

7.2 Diagramme erstellen mit .subplots()

Jetzt sind Sie bereit, die bisherige Theorie in die Praxis umzusetzen und einige Diagramme zu erstellen. Im weiteren Verlauf des Kapitels wird der zustandslose (objektorientierte) Ansatz verwendet, der besser anpassbar ist und sich als nützlich erweist, wenn Diagramme komplexer werden.

Dieser Ansatz beinhaltet den Aufruf von `plt.subplots()` und ist leider nicht allzu intuitiv. Dies ist auch das einzige Mal, dass der objektorientierte Ansatz `pyplot` als Modul von `matplotlib` verwendet, um eine Abbildung und Achsen zu erstellen:

```
1    fig, ax = plt.subplots()
```

Hier werden die Vorteile des *Unpacking* (oder Entpacken) von Iterables genutzt, um jedem der beiden Ergebnisse von `plt.subplots()` eine eigene Variable zuzuweisen. Mehr zu dem Thema Unpacking finden Sie im Studienbrief *Programmierung 1*. Beachten Sie, dass hier keine Argumente an `.subplots()` übergeben wurden, sodass die Parameter die Standardwerte erhalten (hier `nrows=1` , `ncols=1`). Diese beiden Parameter geben die Anzahl der Zeilen und Spalten in der Abbildung an. Folglich ist `ax` in diesem Beispiel ein einzelnes `AxesSubplot` -Objekt. Der Plot oder das Diagramm lassen sich dann manipulieren, indem die zugehörigen Methoden der Instanz aufgerufen werden. Ein Beispieldiagramm dazu sehen Sie in Abbildung 21, in der ein fiktives Schuldenwachstum über die Zeit als gestapeltes Flächendiagramm dargestellt wird.

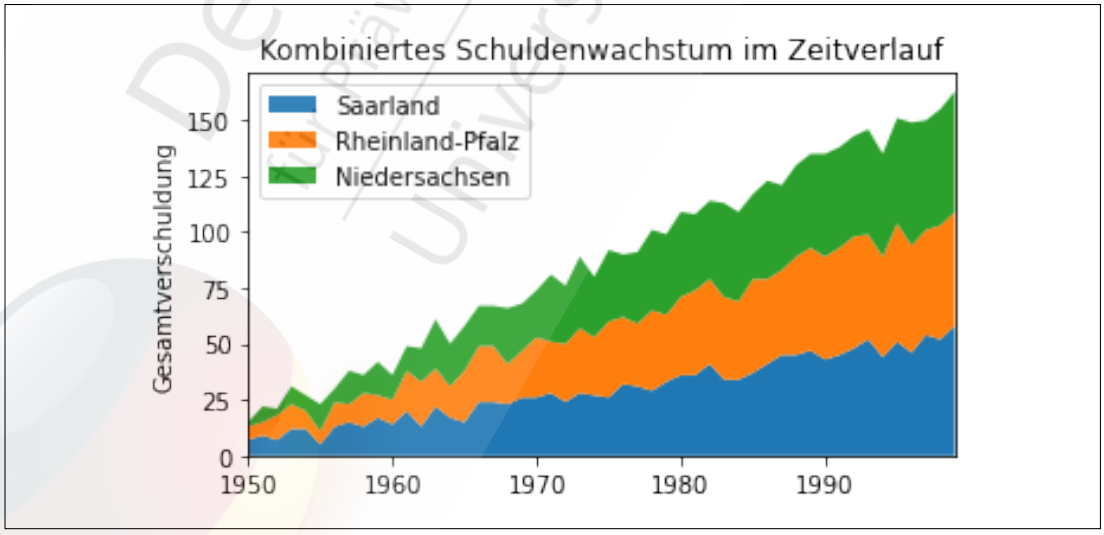


Abbildung 21: Schuldenwachstum im Zeitverlauf. (©BSA/DHfPG)

Denken Sie daran, dass mehrere Diagramme (Axes) in eine bestimmte Abbildung eingeschlossen werden können oder zu ihr *gehören*. Im diesem Fall erhält man mit `fig.axes`

Merke

Das HTTP-Protokoll ist eine Vereinbarung darüber, wie Webseiten im Internet übertragen werden. Alle Kommunikationspartner, das heißt zum Beispiel Browser (Client) und Webserver (Server), müssen sich an diese Vereinbarung halten. So wissen alle Partner jederzeit, wie sie eine Ressource anfordern können und auf welche Weise sie diese erhalten werden.

8.1.4 REST-Schnittstellen

REpresentational State Transfer (REST) ist ein Paradigma für die Architektur von verteilten Softwaresystemen. Es wird eingesetzt, um die einheitliche Kommunikation zwischen zwei Computern zu ermöglichen und wird oft im World Wide Web verwendet. Möchten Sie Daten im World Wide Web abrufen oder anbieten, funktioniert dies häufig mit sogenannten *REST-Schnittstellen*. Diese halten sich an das REST-Paradigma. Vorteil für Sie ist, dass der Aufbau von REST-Schnittstellen immer denselben Prinzipien folgt. Obwohl in der urpsprünglichen Definition von Roy Fielding keine Technologie zur Implementierung festgelegt wurde, verwenden REST-Schnittstellen im World Wide Web fast immer das zuvor erläuterte HTTP-Protokoll und dessen Anfragemethoden und Zustandslosigkeit (siehe Abbildung 30).

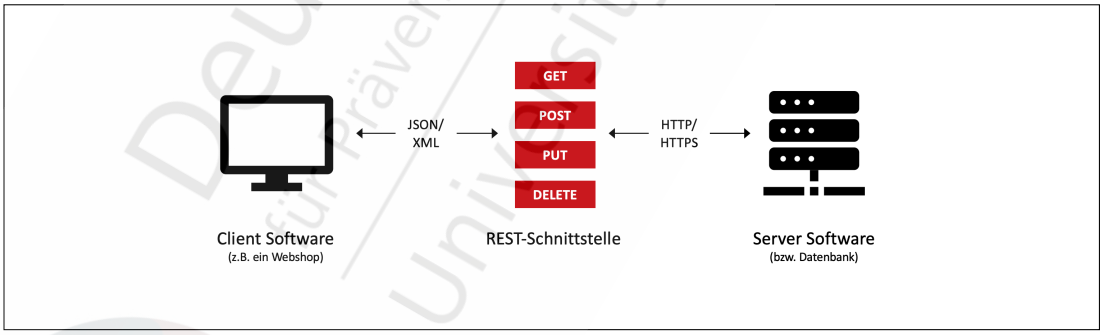


Abbildung 30: Beispiel einer REST-Schnittstelle. (©BSA/DHfPG)

Betrachten Sie das folgende konkrete Beispiel. Sie haben eine Datenbank mit Fitnessübungen aufgebaut und möchten die darin enthaltenen Daten für andere zugänglich machen. Sie könnten hierfür zum Beispiel eine Webseite erstellen, auf der die Daten angezeigt werden. Diese würde jedoch hauptsächlich von echten Menschen genutzt werden. Möchten Sie es anderen Entwicklern ermöglichen, die Daten programmatisch abzufragen, müssen Sie einen anderen Weg finden, die Daten im Internet zur Verfügung zu stellen. Damit könnten andere Entwickler eigene Webseiten oder sogar mobile Apps bauen, auf oder in denen

```
  margin: 5%;

  /* 10px Abstand auf allen Seiten */
  margin: 10px;

  /* 1.6em Abstand für oben/unten, 20px für rechts/links */
  margin: 1.6em 20px;

  /* oben 10px, links/rechts 3%, unten 1em */
  margin: 10px 3% 1em;

  /* oben 10px, rechts 3px, unten 30px, links 5px */
  margin: 10px 3px 30px 5px;

  /* 1em Abstand oben/unten; Box wird horizontal zentriert */
  margin: 1em auto;

  /* Box wird horizontal zentriert, kein Abstand oben/unten */
  margin: auto;
```

Einige CSS-Eigenschaften lassen sich demmnach für jede Seite einzeln angeben. Statt einen Außenabstand in alle Richtungen mit `margin: 10px;` anzugeben, kann ein Abstand nur nach oben direkt mit `margin-top: 10px` festgelegt werden. Eine Umrandungslinie nur für die Unterseite gibt es mit `border-bottom: 1px solid black` .

Gleiches gilt auch für das Padding von Elementen. Der Unterschied zwischen Margin und Padding wird beispielsweise dann relevant, wenn Sie einem Element eine Hintergrundfarbe (mit `background-color`) geben. Erhöhen Sie anschließend das Padding, wächst der farbige Hintergrund mit. Verwenden Sie ein größeres Margin, wird nur der Abstand zum nächsten Element größer. Abbildung 33 veranschaulicht den Unterschied zwischen Inhalt, Padding, Border und Margin.

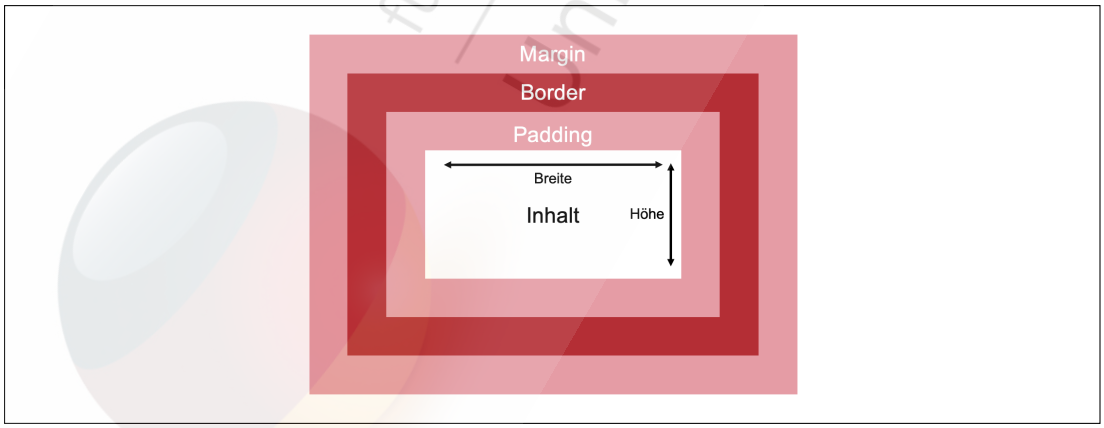


Abbildung 33: Abbildung eines CSS-Boxmodells. (©BSA/DHfPG)

8.4 JavaScript

In den vergangenen Abschnitten haben Sie mit HTML und CSS bereits zwei der drei essenziellen Sprachen zur Erstellung von Webseiten kennengelernt. Mit JavaScript lernen Sie in diesem Kapitel die populärste Skriptsprache kennen, mit der Sie eine Webseite interaktiv gestalten können. Mittlerweile wird JavaScript nicht mehr nur im *Frontend* eingesetzt, also um eine Webseite nach außen für den Nutzer zu gestalten, sondern auch in anderen Umgebungen wie beispielsweise *Backend-Anwendungen* mit *Node.js*.

JavaScript wurde 1995 von Netscape als Werkzeug entwickelt, um statischen HTML-Seiten mehr Dynamik zu verleihen, um damit die Lernkurve zum Einstieg in eine Programmiersprache oder Skriptsprache so niedrig wie möglich gehalten werden. Aus diesem Grund ist die Sprache unter anderem nicht streng typisiert wie Python. Das bedeutet, dass sich Datentypen von Variablen während der Laufzeit unerwartet ändern können - oder auch dürfen.

Da Sie in *Programmierung 1* bereits die Programmiersprache Python kennengelernt haben, sollte Ihnen der Einstieg in JavaScript nicht sonderlich schwerfallen. Die beiden Sprachen sind sich in einigen Aspekten sehr ähnlich. Beide Sprachen bieten zum Beispiel ähnliche Datentypen und Konstrukte zum Kontrollfluss an. Ein kurzes Codesegment in JavaScript werden Sie wahrscheinlich ohne Weiteres bereits jetzt lesen und verstehen können. Im nächsten Abschnitt werden Sie schließlich die größten Unterschiede zwischen Python und JavaScript kennenlernen.

8.4.1 Vergleich zu Python

In den folgenden beiden Codebeispielen sehen Sie den gleichen Code in Python...

```
kunden = ["Lea", "Peter", "Jamil"]

if len(kunden) > 0:
  for kunde in kunden:
    print("Hallo " + kunde + "!")
else:
  print("Keine Kunden.")
```

... und in JavaScript:

```
let kunden = ["Lea", "Peter", "Jamil"];

if (kunden.length > 0) {
  for (let kunde of kunden) {
    console.log("Hallo " + kunde + "!");
  }
} else{
  console.log("Keine Kunden.");
}
```

Exkurs

HTML-Formulare werden eingesetzt, um Nutzerdaten zu sammeln und anschließend meist an einen Server zur Weiterverarbeitung zu schicken. Mit dem Attribut `action` kann eine URL definiert werden, an die eine HTTP-Anfrage mit den Formularwerten gesendet werden soll. Mit dem Attribut `method` kann festgelegt werden, ob die Daten als GET- oder POST-Parameter übertragen werden. POST wird verwendet, wenn sensiblen Informationen nicht in der URL sichtbar sein sollen.

Ein HTML-Formular kann aus mehreren Eingabefeldern bestehen. Jedes Feld wird durch ein `<input>` -Element hinzugefügt. Zusammengehörige Eingabefelder können in einem `<fieldset>` gruppiert werden. Zu jedem Eingabefeld kann mit dem Attribut `type` angegeben werden, welche Daten darin erwartet werden. Mit dem Typ `text` erhalten Sie ein Textfeld. Der Typ `checkbox` hingegen wird als *Checkbox* dargestellt, damit Nutzer die booleschen Werte `True` oder `False` angeben können. Mit dem Attribut `name` müssen Sie festlegen, wie der GET- oder POST-Parameter heißt, in dem der eingegebene Wert abgeschickt wird. Mit diesem Namen können Sie den Wert auf einem Server über die HTTP-Anfrage auslesen.

Zu jedem Eingabeelement sollten Sie zudem stets eine zugehörige Beschriftung mit dem Element `<label>` definieren. Welche Beschriftung zu welchem Eingabeelement gehört, können Sie mit dem Attribut `for` festlegen. Dort tragen Sie bei einem Label die `id` des zugehörigen Eingabefeldes ein.

Schließlich benötigen Sie noch einen Bestätigungs-Button, mit dem das Formular abgeschickt werden kann. Dieser wird als `<input>` mit `type="submit"` erstellt. Das `value` -Attribut legt Beschriftung des Buttons fest.

Eine detaillierte Beschreibung aller Attribute und Verhaltensweisen von HTML-Formularen finden Sie in der Webdokumentation von Mozilla[a].

[a] `https://developer.mozilla.org/en-US/docs/Web/HTML/Element/form`

Erstellen Sie nun eine Django-View, die das zuvor definierte Template beim Aufruf der Seite `http://127.0.0.1:8000/kundenformular` als HTTP-Antwort zurückgibt. Dazu benötigen Sie den *Django Template Loader*, den Sie am Anfang der `views.py` -Datei importieren können. Mit der Funktion `get_template` laden Sie das zuvor geschriebene HTML aus der Template-Datei. Anschließend wird das Template *gerendert* und als Antwort zurückgegeben. Warum Laden und Rendern zwei verschiedene Schritte sind und welche unterschiedlichen Zwecke, die beiden erfüllen, erfahren Sie in Kürze.

```
1  from django.template import loader
2
3  def kundenformular(request):
4    # Template laden
```