SAE

Abschlussprüfung Teil 2

- HTML
- Struktogramme
- Datenbanken
- Pseudocode
- Python
- Web Requests
- Sortieralgorithmen
- Polymorphismus & Überladen
- Programmierparadigmen
- CSS - Cascading Style Sheet
- Entity-Relationship-Modell

HTML

Einführung:

Bei HTML handelt es sich NICHT um eine Programmiersprache sondern um eine Beschreibungssprache. HTML steht für "Hypertext Markup Language". Diese zeichnet sich dadurch aus, dass optimalerweise jeder Bereich in sogenannte Tags gefasst wird.

Jedes HTML Dokument besteht aus folgendem Grundgerüst:

```
<!doctype html>
<html lang="de">
<head>
<!-- Hier kommen alle Metadaten hin -->
  <meta charset="utf-8"/>
<meta name="viewport" content="width=device-width, initial-scale=1.0"/>
<title>Hierhin gehört der Seitentitel</title>
</head>
<body>
<!-- Hier kommt der gesamte Inhalt rein -->
<h1> Überschrift </h1>

</body>
</html>
```

Ein HTML Dokument dient zwar offiziell nur für den Inhalt und das Grundgerüst einer Website. Dennoch ist es in einem gewissen Rahmen auch möglich, das Aussehen einer Website zu bearbeiten. Die zugehörigen Tags können folgender Tabelle entnommen werden:

Textdesign

Überschriften	Überschrift kann von 1 bis 6 verwendet werden
Paragraph	Textparagraph
Fett	Fett
Kursiv	Kursiv

Markiert	<mark>Markiert</mark>
Klein	Klein
Unterstrichen	<ins>Unterstrichen</ins>
Durchgestrichen	~~Durchgestrichen~~
Hochgestellt	^{Hochgestellt}
Tiefgestellt	_{Tiefgestellt}
Abkürzungen	<abbr title="World Health Organization">WHO</abbr>
Anführungszeichen	<q>Quotes</q>

Es gibt in HTML zwei verschiedene Arten von Elementen. Es gibt zum einen die Block Elemente, die immer in einer neuen Zeile beginnen, zum anderen gibt es die Inline Elemente, die im Fliesstext stattfinden.

Style Attribute?

Tabellen und Listen

```
<table>
 <tr>□□□□□□□<!-- Neue Zeile -->
   <th colspan="2">Name</th> □<!-- Tabellenüberschrift, die sich über 2 Spalten streckt -->
   <th>Age</th>□□□□<!-- Zelleninhalt -->
 </tr>
 <tr>
   <td>Jill</td>
   <td>Smith</td>
   <td>43</td>
 </tr>
 <tr>
   <th rowspan="2">Phone</th>□<!-- Zeilenbeschriftung, die sich über 2 Zeilen streckt -->
   <td>555-1234</td>
   <td>xyz</td>
 </tr>
 <tr>
   <td>Eve</td>
```

```
    <td>Jackson</td>
  </tr>
</table>
```

Listen

```
<!-- Geordnete Listen -->
<ol>
  <li>Coffee</li>
  <li>Tea
    <ol type="a">
      <li>Black tea</li>
      <li>Green tea</li>
    </ol>
  </li>
    <li>Energy
    <ol type="A">
      <li>Red Bull</li>
      <li>Monster</li>
    </ol>
  </li>
  <li>Milk</li>
</ol>

<!-- Ungeordnete Listen -->

<ul>
  <li>Coffee</li>
  <li>Tea
    <ul>
      <li>Black tea</li>
      <li>Green tea</li>
    </ul>
  </li>
  <li>Milk</li>
</ul>

<!-- Sonstige  Listen -->

<dl>
```

```
  <dt>Coffee</dt>
  <dd>- black hot drink</dd>
  <dt>Milk</dt>
  <dd>- white cold drink</dd>
</dl>
```

Klassen

Auch in HTML gibt es Klassen. Allerdings definieren diese hier nur, dass alles innerhalb eines Bereichs zu einer Klasse gehört. Auf diese Klassen können dann mit CSS oder JavaScript Veränderungen angewendet werden. Die Zuweisung geschieht wie in folgendem Code gezeigt:

```
<div class="city">
  <h2>London</h2>
  <p>London is the capital of England.</p>
</div>
```

Des weiteren kann jedem Element ein einzigartiger Identifyer zugewiesen werden. Dies geschieht mittels dem id Element.

```
<h1 id="myHeader">My Header</h1>
```

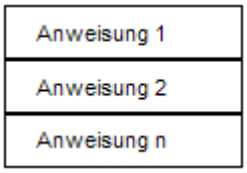
Links

Links auf andere Websites können in HTML wie folgt eingebunden werden. Dabei ist zu beachten, dass der Linktext nicht zwingend gleich dem eigentlichen Link sein muss.

```
 <a href="https://doku.stnd.io">This is a link</a>
```

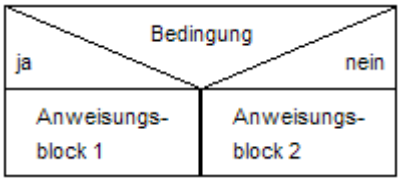
Struktogramme

- auch als Nassi-Shneiderman-Diagramme bekannt
- Ziel: Darstellung eines Programms unabhängig von der Programmiersprache
- Symbole
  - mit Hilfe dieser Symbole lässt sich der Ablauf eines Programms beschreiben
  - fast alle Symbole lassen sich beliebig ineinander verschachteln
  - Prozess Symbol / Process Symbol: Anweisungen werden nacheinander von oben nach unten durchlaufen



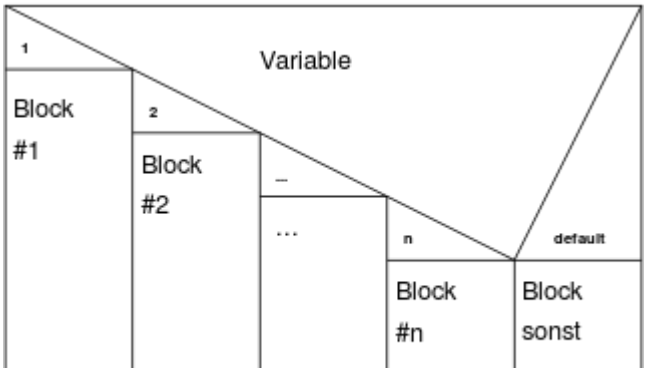

```
# Anweisungen in Python
a = input("Zahl eingaben")  # Anweisung 1
b = 5 * a                   # Anweisung 2
print(b)                    # Anweisung 3
```

  - Verzweigung / Decision Symbol: Bedingung wird geprüft, wenn Sie zu trifft wird "ja" ausgeführt, andernfalls "nein". Kann verschachtelt sein.




```
# Verzweigung in Python
if (a == 5):        # Prüfung der Variablen a -> Ergebnis True oder False
    print("a ist fünf")    # Wenn Prüfung True, wird dieser Block ausgeführt
else:
    print("a ist nicht fünf")  # Wenn Prüfung False, wird dieser Block ausgeführt
```

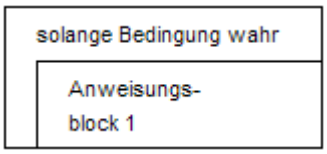
  - Sonderfall: Case-Statement: Inhalt der Variablen wird geprüft und entsprechender Fall wird ausgeführt. Manche Programmiersprachen haben "Switch" ansonsten mit "if - else if - else" auflösbar.



```
// Mit Switch in C aufgelöst
case (a) {□□□□□□□□□□// Angabe welche Variable geprüft werden soll
  □1: printf("a hat den Wert eins");□□// Auswahl entsprechend der Prüfung
    2: printf("a hat den Wert zwei");
    3: printf("a hat den Wert drei");
    default: print("a ist größer drei");□// Sollte keine Prüfung zutreffen
wird dieser Fall ausgeführt
};

// Mit if - else if - else aufgelöst
if (a == 1) {□□□□□□□□□// Prüfung der Variable a -> Ergebnis True / False
□printf("a hat den Wert eins");□□□// Block der True als Ergebnis hat wird
ausgeführt
} else if (a == 2) {
□printf("a hat den Wert zwei");
} else if (a == 3) {
□printf("a hat den Wert drei");
} else {
□printf("a ist größer drei");□□□// Wird ausgeführt sollte kein Block True sein
};
```

- Schleifen oder Wiederholungsstruktur: Struktur die solange durchlaufen wird, bis Endbedingung erfüllt ist
  - Kopfgesteuerte - Schleifen (z.B. while und for): Bedingung wird vor ersten durchlauf geprüft und nur dann betretten, wenn Bedingung zutrifft. Kann also auch nicht durchlaufen werden, wenn Endbedingung direkt zutrifft.

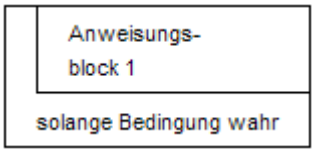



```
// Kopfgesteuerte while-Schleife
while (i <= 10) {□□□□□// Die Variable i wird geprüft, sollte sie bereits größer
10 sein, wird Schleife nicht ausgeführt
□printf("%i\n", i);□□□□// Wird ausgeführt sollte i kleiner oder gleich 10 sein
    i++;
}

// Kopfgesteuerte for-Schleife
for (i = 0; i <= 10; i++) {□□□// Sonderfall i läuft von 0 bis 10
□printf("%i\n", i);
```

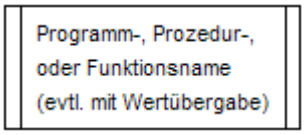
```
}
```

- Fußgesteuerte - Schleife (z.B. do-while): Endbedingung wird nach dem ersten Durchlauf geprüft und wenn diese Zutrifft, wird die Schleife beendet. Sollte sie nicht zu treffen, wird die Schleife solange durchlaufen bis die Bedingung zutrifft.




```
// Fussgesteuerte-Schleife
do {□□□□□□□□// Schleife wird betretten
□printf("%i\n", a);□□□□// Schleife wird ausgeführt
    a++;
} while (a <= 10);□□□□// Prüfung und sollte a größer 10 sein, wird Schleife
beendet. Ansonsten wird Schleife erneut ausgeführt
```

- Funktion und Funktionsaufruf: Um nicht mehrfach den gleichen Code schreiben zu müssen, kann man Funktionen schreiben die man immer wieder aufrufen kann. Dazu muss man zwei Dinge beachten.
  Zum Einen muss die Funktion aufgerufen werden ggf. Parameter übergeben werden.
  Zum Anderen wird die Funktion als eigenes Struktogram geschrieben und am Anfang die möglichen Übergabeparamet genannt.
  Sollte die Funktion Rückgabeparameter haben sind diese mit **Rückgabe** zu markieren (nicht Ausgabe).
  - Funktionsaufruf

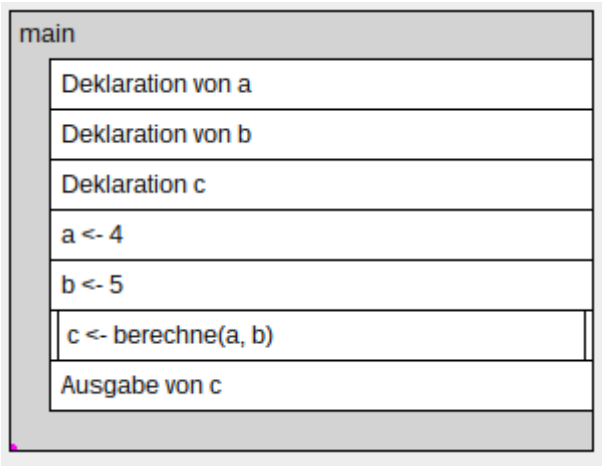



```
// Möglicher Funktionsaufruf in C

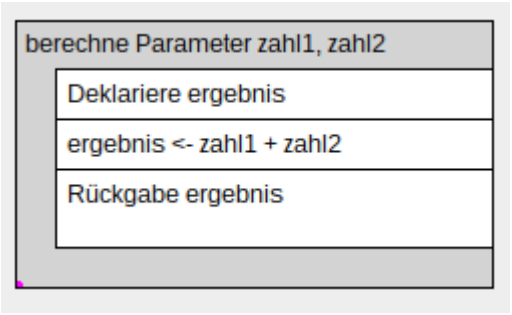
// Funktionsdefinition
int berechne(int zahl1, int zahl2) {□// Funktion bekommt zwei Integer
übergeben und gibt einen Integer zurück
  int ergebnis;□□□□□□□// lokale Variable
  ergebnis = zahl1 + zahl2;□□□□// Berechnung von ergebnis
  return ergebnis;□□□□□□// Rückgabewert
}

//Hauptfunktion
void main(void) {
  int a;□□□□□□□□// Deklaration von Variablen
```

```
  int b;
  int c;
  a = 4;□□□□□□□□// Zuweisung von Wert an Variable
  b = 5;
  c = ergebnis(a, b);□□□□□// Aufruf von Funktion "berechne" Übergabeparameter
sind Werte von a, b. Rückgabe von Funktion wird zugewiesen
  printf("%i\n", c);□□□□□// Ausgabe von c
}
```

- 

llt. Hauptfunktion (main) und



Datenbanken

Normalisierung von Datenbanken

Unter Normalisierung einer relationalen Datenbank versteht man die Aufteilung von Attributen in mehrere Relationen (Tabellen) mit Hilfe der Normalisierungsregeln. Insgesamt gibt es fünf Normalformen, aber im Alltag reicht eine Normalisierung bis zur dritten Normalform vollkommen aus. Die vierte und fünfte Normalform sind für Spezialfälle wie Atomkraftwerke usw. notwendig.

Die einzelnen Normalformen bauen aufeinander auf, d.h. die dritte Normalform ist nur erfüllt wenn auch die zweite Normalform erfüllt ist und die zweite Normalform ist nur erfüllt, wenn auch die erste Normalform erfüllt ist.

Ziel der Normalisierung:

- Beseitigung von Redundanzen
- Vermeidung von Anomalien
- Erstellen eines klar strukturierten Datenbankmodells

Erste Normalform (1NF)

Die **erste Normalform** ist dann erfüllt, wenn alle Informatione einer Tabelle **atomar** vorliegen. Atomar bedeutet, dass jede Information eine eigene Spalte bekommt.

Beispiel:

- nicht normalisierte Form

R-Nr	Datum	Name	Straße	Ort	Artikel	Anzahl	Preis
187	01.01.2022	Max Mustermann	Musterweg. 1	12345 Musterstadt	Bleistift	5	1.00€

- erste Normalform

R-Nr	Datum	Name	Vorname	Straße	Hnr	PLZ	Ort	Artikel	Preis	Währung
187	01.01.2022	Mustermann	Max	Musterstr.	1	12345	Musterstadt	5	1.00	Euro

Zweite Normalform (2NF)

Die **zweite Normalform** ist dann erfüllt, wenn jedes nicht-Schlüsselattribut **voll funktional** vom Schlüssel abhängig ist.

Beispiel:

- zweite Normalform

Rechnung		
R-Nr	Datum	Knr
187	01.01.2022	007

Kunde						
Knr	Name	Vorname	Straße	Hnr	PLZ	Ort
007	Mustermann	Max	Musterstr.	1	12345	Musterstadt

Rechnungsposition			
R-P-NR	R-Nr	Art-Nr	Anzahl
1	187	69	5

Artikel		
Art-Nr	Artikel	Preis
69	Bleistift	1.00

Dritte Normalform (3NF)

Die **dritte Normalform** ist dann erfüllt, wenn kein Nichtschlüsselattribut **transitiv** von einem Kandidatenschlüssel abhängt. Einfacher gesagt, es gibt immer nur einen möglichen Primärschlüssel und kein Attribut die auch Primärschlüssel sein können werden zusätzliche Tabellen ausgelagert und als Fremdschlüssel eingebunden.

Beispiel:

- dritte Normalform

Kunde					
Knr	Name	Vorname	Straße	Hnr	PLZ
007	Mustermann	Max	Musterstr.	1	12345

Postleitzahl	
PLZ	Ort
12345	Musterstadt

Manipulation von Datenbanken

Eine Datenbank ist eine organisierte Sammlung von strukturierten Informationen oder Daten, die elektronisch in einem Computer gespeichert werden. Eine Datenbank wird in der Regel von einem Datenbankmanagementsystem (DBMS) gesteuert. Die Daten und das DBMS werden zusammen mit den damit verbundenen Anwendungen als Datenbanksystem bezeichnet, oft auch nur als Datenbank.

Die Daten in den heute gebräuchlichsten Arten von Datenbanken werden in der Regel in Zeilen und Spalten in einer Reihe von Tabellen dargestellt, um eine effiziente Verarbeitung und Datenabfrage zu ermöglichen. Die Daten können dann leicht abgerufen, verwaltet, geändert, aktualisiert, kontrolliert und organisiert werden. Die meisten Datenbanken verwenden eine strukturierte Abfragesprache (SQL) zum Schreiben und Abfragen von Daten.

Die `INSERT INTO` Anweisung wird verwendet, um einen neuen Datensatz (Zeile) in eine Tabelle einzufügen.
 	
Es gibt zwei Varianten:
• In Spalten der Reihe nach einfügen
• Einfügen in Spalten nach Namen
	`-- In Spalten der Reihe nach einfügen:`
	`INSERT INTO table_name`
	`VALUES (value1, value2);`
	
	`-- Einfügen in Spalten nach Namen:`
	`INSERT INTO table_name (column1, column2)`
	`VALUES (value1, value2);`

Die `DELETE` Anweisung wird verwendet, um Datensätze (Zeilen) in einer Tabelle zu löschen. Die `WHERE` Anweisung gibt an, welcher Datensatz oder welche Datensätze gelöscht werden sollen. Wenn `WHERE` weggelassen wird, werden alle Datensätze gelöscht.	`DELETE FROM table_name` `WHERE some_column = some_value;`

Die `UPDATE` Anweisung wird verwendet, um Datensätze (Zeilen) in einer Tabelle zu bearbeiten. Sie enthält eine `SET` Anweisung, die die zu bearbeitende Spalte angibt, und eine `WHERE` Anweisung, die den oder die Datensätze spezifiziert.	`UPDATE table_name`
	`SET column1 = value1, column2 = value2`
	`WHERE some_column = some_value;`

Datenbank Abfragen

Mit dem `AND` Operator können mehrere Bedingungen kombiniert werden. Die Datensätze müssen beiden Bedingungen entsprechen, die mit `AND` verknüpft sind, um in die Ergebnisse aufgenommen zu werden. Die angegebene Abfrage zeigt alle Autos an, die blau sind und nach 2014 hergestellt wurden.	`SELECT model`
	`FROM cars`
	`WHERE color = 'blue'`
	`  AND year > 2014;`

Mit dem `OR` Operator können mehrere Bedingungen kombiniert werden. Datensätze, die einer der beiden Bedingungen entsprechen, die durch das `OR` verbunden sind, werden in die Ergebnisse aufgenommen. Die angegebene Abfrage findet Kunden, deren Staat entweder „CA“ oder „NY“ ist.	`SELECT name`
	`FROM customers`
	`WHERE state = 'CA'`
	`OR state = 'NY';`

`WHERE` wird verwendet, um Datensätze (Zeilen) zu filtern, die eine bestimmte Bedingung erfüllen. Die angegebene Abfrage wählt alle Datensätze aus, bei denen das pub_year gleich 2017 ist.	`SELECT title`
	`FROM library`
	`WHERE pub_year = 2017;`

Mit `LIKE` kann innerhalb einer `WHERE` ein bestimmtes Muster abgefragt werden. Die angegebene Abfrage findet jeden Film, dessen Titel mit „Star“ beginnt.	`SELECT name`
	`FROM movies`
	`WHERE name LIKE 'Star%';`

Der Platzhalter `%` kann mit `LIKE` verwendet werden, um keine oder mehrere nicht spezifizierte Zeichen zu finden. Die angegebene Abfrage findet jeden Film, der mit „The“ beginnt, gefolgt von keinem oder mehreren beliebigen Zeichen.	`SELECT name`
	`FROM movies`
	`WHERE name LIKE 'The%';`

Der Platzhalter `_` kann mit `LIKE` verwendet werden, um ein beliebiges einzelnes, nicht spezifiziertes Zeichen zu finden. Die angegebene Abfrage findet jeden Film, der mit einem einzelnen Zeichen beginnt, gefolgt von „ove“.	`SELECT name`
	`FROM movies`
	`WHERE name LIKE '_ove';`

Die `SELECT *` Anweisung gibt alle Spalten aus der angegebenen Tabelle in der Trefferliste zurück. Die angegebene Abfrage holt alle Spalten und Datensätze (Zeilen) aus der Tabelle „Filme“.	`SELECT *` `FROM movies;`

`ORDER BY` kann verwendet werden, um die Ergebnisse einer Spalte alphabetisch oder numerisch zu sortieren. Es kann auf zwei Arten sortiert werden:
• Mit `DESC` werden die Ergebnisse in absteigender Reihenfolge sortiert.
• Mit `ASC` werden die Ergebnisse in aufsteigender Reihenfolge sortiert (Standard).
	`SELECT *`
	`FROM contacts`
	`ORDER BY birth_date DESC;`

Der Operator `BETWEEN` kann zum Filtern nach einem Wertebereich verwendet werden. Bei dem Wertebereich kann es sich um Text, Zahlen oder Datumsdaten handeln. Die angegebene Abfrage findet alle Filme, die zwischen den Jahren 1980 und 1990 gedreht wurden.	`SELECT *`
	`FROM movies`
	`WHERE year BETWEEN 1980 AND 1990;`

Die Bedingung `LIMIT` wird verwendet, um die Ergebnisse auf eine bestimmte Anzahl von Zeilen zu begrenzen. Die angegebene Abfrage begrenzt die Ergebnismenge auf 5 Zeilen.	`SELECT *`
	`FROM movies`
	`LIMIT 5;`

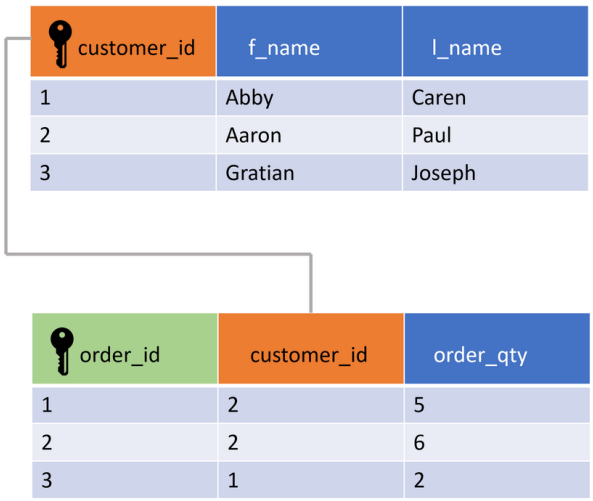
Spalteninhalte können `NULL` sein oder keinen Wert haben. Diese Datensätze können mit den Operatoren `IS NULL` und `IS NOT NULL` in Kombination mit `WHERE` abgeglichen werden. Die angegebene Abfrage wird alle Adressen anzeigen, bei denen die Adresse einen Wert hat oder nicht `NULL` ist.	`SELECT address`
	`FROM records`
	`WHERE address IS NOT NULL;`

Erstellen von Tabellen

Nach `CREATE TABLE` folgt der Tabellenname. Innerhalb der Klammern werden die Spalten und deren Datentyp aufgelistet. Der `PRIMARY KEY` gibt dabei an, in welcher Spalte die eindeutige Identifikationsnummer steht. Getrennt werden die einzelen Einträge mit Komma. Gibt es einen Verweis auf eine andere Tabelle, dann erfolgt dies durch den sogenannten `FOREIGN KEY`. Dieser gibt an in welcher Spalte die Referenz steht und auf welche Spalte einer anderen Tabelle referenziert wird.	`CREATE TABLE Personen {`
	`PersonenID INTEGER PRIMARY KEY,`
	`Nachname varchar(255),`
	`Vorname varchar(255),`
	`GeburtsOrtID INTEGER,`
	`FOREIGN KEY (GeburtsOrtID) REFERENCES Ort(OrtsID)`
	`}`

Foreign Key

Ein Fremdschlüssel ist ein Verweis von Datensätzen einer Tabelle auf den Primärschlüssel einer anderen Tabelle. Um mehrere Datensätze für eine bestimmte Zeile zu erhalten, spielt die Verwendung von Fremdschlüsseln eine wichtige Rolle. Um z. B. alle Bestellungen eines bestimmten Kunden zu verfolgen, kann die Tabelle Bestellung (unten im Bild) einen Fremdschlüssel enthalten.



customer_id	f_name	l_name
1	Abby	Caren
2	Aaron	Paul
3	Gratian	Joseph

order_id	customer_id	order_qty
1	2	5
2	2	6
3	1	2

Primary Key

Ein Primärschlüssel in einer SQL-Tabelle wird verwendet, um jeden Datensatz in dieser Tabelle eindeutig zu identifizieren. Ein Primärschlüssel kann nicht `NULL` sein. In diesem Beispiel ist `customer_id` der Primärschlüssel. Derselbe Wert kann in einer Spalte nicht noch einmal vorkommen. Primärschlüssel werden oft in `JOIN` Operationen verwendet.

customer_id	f_name	l_name
1	Abby	Caren
2	Aaron	Paul
3	Gratian	Joseph

Mehrere Tabellen verbinden

Bei einem Outer Join werden Zeilen aus verschiedenen Tabellen kombiniert, auch wenn die Join-Bedingung nicht erfüllt ist. Bei einem `LEFT JOIN` wird jede Zeile der linken Tabelle in die Ergebnismenge zurückgegeben. Wenn die Join-Bedingung nicht erfüllt ist, wird `NULL` verwendet, um die Spalten der rechten Tabelle aufzufüllen.

```
SELECT column_name(s)
FROM table1
LEFT JOIN table2
  ON table1.column_name =
table2.column_name;
```

`JOIN` ermöglicht die Rückgabe von Ergebnissen aus mehr als einer Tabelle, indem sie diese mit anderen Ergebnissen auf der Grundlage gemeinsamer Spaltenwerte verbindet, die mit `ON` angegeben werden. `INNER JOIN` ist der Standard `JOIN` und gibt nur Ergebnisse zurück, die der durch `ON` angegebenen Bedingung entsprechen.	``` SELECT * FROM books JOIN authors ON books.author_id = authors.id; ```

Pseudocode

> Pseudocode ist eine detaillierte und dennoch lesbare Beschreibung dessen, was ein Computerprogramm oder ein Algorithmus machen soll. Pseudocode wird in einer formal gestalteten, natürlichen Sprache und nicht in einer Programmiersprache ausgedrückt.

Fallunterscheidungen

- `if ... then ... else ... end if/exit`
- `wenn ... dann ... sonst ... wenn_ende`
- `falls ... dann ... falls_nicht ... falls_ende`

Schleifen

- `wiederhole ... solange/bis ... wiederhole_ende`
- `while ... do ...`
- `repeat ... until ...`
- `for ... to ... step Schrittweite ... next`

Kommentare

- `// kommentar`
- `# kommentar`
- `/* kommentar */`

Beispiel

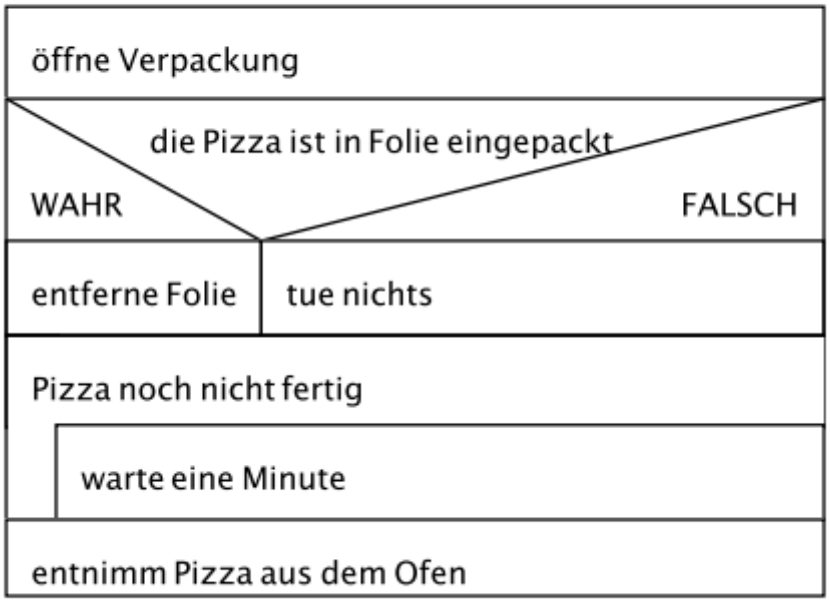
```
WENN die Pizza in Folie eingepackt ist
□Entferne Folie

schalte Ofen ein
gib Pizza auf Blech in Ofen

SOLANGE Pizza noch nicht fertig
□warte eine Minute

entnimm Pizza aus dem Ofen
```

Tiefkühlpizza backen



Python

Dateiverarbeitung

Lesen	`with open("file.txt", "r") as datei:` `  text = datei.read()`
Schreiben	`with open("file.txt", "w") as datei:` `  datei.write(text + "Hallo\n")`
Leerzeichen entfernen	`text.strip("\n")`

Nutzerinteraktion

input()
input("str")
	Unterbricht den Programmablauf und wartet auf Eingabe des Users. Programm wird danach fortgesetzt. Der Funktion kann ein String übergeben werde, der auf dem Bildschirm angezeigt wird. Die Eingabe des Users ist der Rückgabewert dieser Funktion. Rückgabewert ist immer String.	`# Eingabeaufforderung`
		`# Eingabe des Users wird in abfrage geschrieben`
		`abfrage = input("Hier könnte Ihre Werbung stehen: ")`
		
		`# Bildschirmausgabe`
		`Hier könnte Ihre Werbung stehen:`
		`# Usereingabe`
		`Nein`
		
		`print(abfrage)`
		`# Ausgabe`
		`Nein`

Verzweigungen

```
if bmi < 19:
  print("Untergewicht")
elif bmi <= 24:
  print("Normalgewicht")
else:
  print("Uebergewicht")
```

Schleifen

while	`i = 0`
	`while i < 10:`
	`  print(i)`
	`  i += 1`

for (eigentlich foreach)	`for zahl in [1, 2, 3]:` `  print(zahl)`

Operatoren

in	Prüft ob Variable in angegebener Sequenz vorhanden ist
	vorhanden gibt "True" zurück
	nicht vorhanden gibt "False" zurück
		`#true`
		`print(2 in [2, 3, 4])`
		
		`#false`
		`erg = "bla" in ["bli", "blub"]`
		`print(erg)`

range	Funktion gibt eine Sequenz von Zahlen zurück
	`range(start, stop, step)`
	
	start = inklusiver Anfang der Sequenz, optional, default ist 0
	stop = exklusves Ende der Sequenz, benötigt
	step = Inkrement, optional, default 1
		`# 0, 1, 2, 3, 4`
		`range(5)`
		
		`# 4, 5, 6, 7`
		`range(4, 8)`
		
		`# 2, 4, 6, 8`
		`range(2, 9, 2)`

Listen

append()	hängt Objekt an das Ende der Liste	`# hängt den Wert 5 an Liste an` `liste.append(5)`

index()	gibt den ersten Index zu angegebenem Wert zurück	`liste = [2, 3, 4, 5, 6, 3, 4, 5]`
		`print(liste.index(4))`
		`# Ausgabe ist 2`

insert()	fügt Wert vor Index ein `liste.insert(index, value)`	`liste = [2, 3, 4, 5]`
		`liste.insert(1, 6)`
		`print(liste)`
		`# Ausgabe [2, 6, 3, 4, 5]`

remove()	entfernt das erste Vorkommen des Werts	`liste = [2, 3, 4, 5]`
		`liste.remove(4)`
		`print(liste)`
		`# Ausgabe [2, 3, 5]`

reverse()	Umdrehen der Liste in-place	`liste = [2, 3, 4, 5]`
		`liste.reverse()`
		`print(liste)`
		`# Ausgabe [5, 4, 3, 2]`

sort()	Sortiert die Liste in-place
	default = aufsteigend
	"reverse=True" = absteigend
		`# Sortieren in aufsteigender Reihenfolge`
		`liste = [3, 2, 5, 4]`
		`liste.sort()`
		`print(liste)`
		`# Ausgabe [2, 3, 4, 5]`
		
		`# Sortieren in absteigender Reihenfolge`
		`liste = [3, 2, 5, 4]`
		`liste.sort(reverse=True)`
		`print(liste)`
		`# Ausgabe [5, 4, 3, 2]`

Ausgabe Formatierung	Die Ausgabe von Listen kann nach folgender Notation formatiert werde `listenname[start:stop:step]`
	
	start = inklusiver Startwert, default 0
	stop = exklusiver Stopwert, default Länge
	step = Inkrement, default 1
		`liste = [2, 3, 4, 5, 6, 7, 8, 9]`
		
		`print(liste[0:4])`
		`# Ausgabe [2, 3, 4]`
		
		`print(liste[1:6:2])`
		`# Ausgabe [3, 5, 7]`
		
		`print(liste[2:6:2])`
		`# Ausgabe [4, 6]`

Umwandlung von Datentypen

> Wichtig: Alle Funktionen sind **KEINE** in-place Ersetzung

type()	Gibt den Datentyp einer Variablen zurück	`type(5)`
		`# Ausgabe <class 'int'>`
		
		`type(5.0)`
		`# Ausgabe <class 'float'>`
		
		`type(5)`
		`# Ausgabe <class 'str'>`

str()	Rückgabewert dieser Funktion ist ein String	`str(5)` `# Ausgabe '5'`
int()	Rückgabewert dieser Funktion ist ein Integer	`int("5")` `# Ausgabe 5`
float()	Rückgabewert dieser Funktion ist eine Fließkommazahl	`float(5)` `# Ausgabe 5.0`

Web Requests

Allgemeines

Sogenannte Web Requests sind eigentlich Anfragemethoden des HTTP Protokolls. Sie dienen dazu, mit Webservern zu kommunizieren und die Anfrage zu Klassifizieren.

GET

Mithilfe des GET Requests werden Inhalte vom Server angefordert. Dieser ist so ziemlich der häufigste Vorgang. Die Anfrage wird in der URL mitgeschickt.

> Mittels GET können auch Daten an den Server übertragen werden, allerdings ist hier die Menge an zu übertragenden Daten begrenzt (URL Begrenzung) daher sollten hier maximal 255 Zeichen verwendet werden.

POST

Dient zu Übertragung von Daten an den Server (zum Beispiel Formulardaten). Hiermit können unbegrenzt große Daten übertragen und auch validiert werden. Anfrage wird im "request Body" übermittelt.

HEAD

Mittels eines HEAD Requests wird der Server dazu angewiesen, nur den Header der Daten zu übertragen und nicht wie bei GET mit dem Gesamten Body.

PUT

Dient dazu Daten auf einem Server Upzudaten oder Abzulegen. Findet meist bei APIs Anwendung ist aber bei den normalen Webservern zumeist aus Sicherheitsgründen deaktiviert.

DELETE

Hiermit können Daten auf dem Webserver gelöscht werden. Allerdings ist DELETE wie auch PUT meist nicht Implementiert.

TRACE

Testet das Clientverhalten, in dem es so tut als ob der Webserver die Daten nie erhalten hat. Wird meist zum Debuggen verwendet.

Vergleich GET und POST

	GET	**POST**
BACK button/Reload	kein Problem	Daten werden erneut übermittelt, Browser sollte Meldung ausgeben, dass Daten erneut übermittelt werden
Bookmarked (Lesezeichen dieser erstellen)	geht als Bookmark	geht **nicht** als Bookmark
Cached	kann gecached werden	kann **nicht** gecachet werden
History	Parameter sind in Browser History	Parameter werden **nicht** in der Browser History gespeichert
Datenlänge	Länge der URL ist begrenzt	keine Begrenzung
Datentyp	nur ASCII Zeichen sind erlaubt	keine Begrenzung
Sicherheit	weniger sicher wie POST, da Daten in URL übertragen werden Kritisch bei Passwörtern und sensiblen Informationen	etwas sicherer als GET, da Parameter nicht in der Browser History und den Webserver Logs gespeichert werden (verschlüsselt Übertragung aber nicht)
Sichtbarkeit	Daten sind in URL sichtbar	Daten werden **nicht** in der URL gezeigt

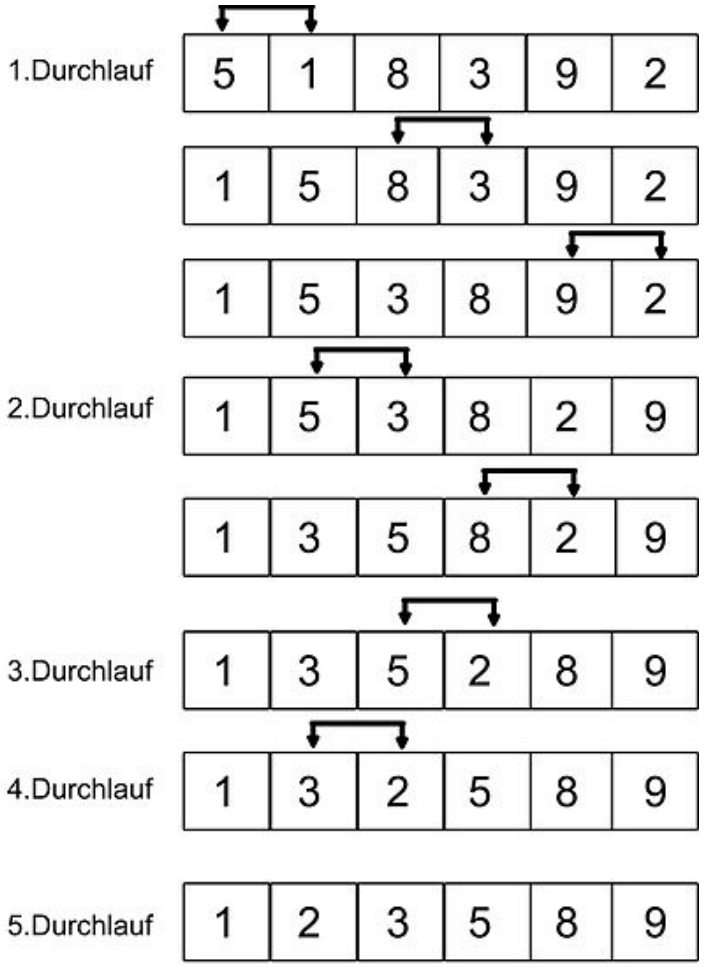
Sortieralgorithmen

Bubblesort

Video - Bubblesort

https://www.youtube-nocookie.com/embed/ARZLBeagiJ4

Beim Bubblesort Algorithmus wird ein Array – also eine Eingabe-Liste – immer paarweise von links nach rechts in einer sogenannten Bubble-Phase durchlaufen. Man startet also mit der ersten Zahl und vergleicht diese dann mit ihrem direkten Nachbarn nach dem Sortierkriterium. Sollten beide Elemente nicht in der richtigen Reihenfolge sein, werden sie ganz einfach miteinander vertauscht. Danach wird direkt das nächste Paar miteinander verglichen, bis die gesamte Liste einmal durchlaufen wurde. Die Phase wird so oft wiederholt, bis der gesamte Array vollständig sortiert ist.



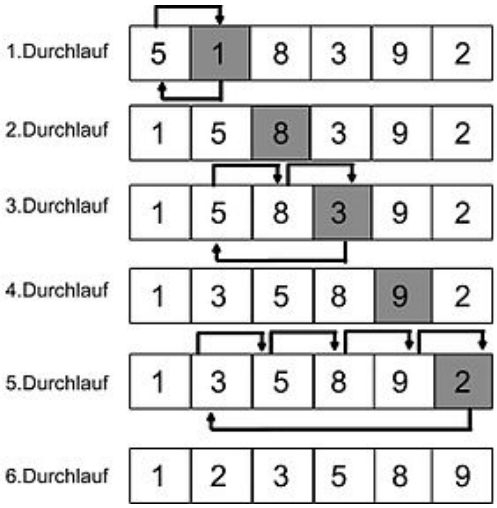
Abbruch, da keine Vertauschungen mehr auftreten.

Insertionsort

Video - Insertionsort

https://www.youtube-nocookie.com/embed/JMRU2nh6bfs

Der Insertion Sort gehört in der Informatik zu den stabilen Sortieralgorithmen und kann als Sortieren durch Einfügen beschrieben werden, deswegen auch Einfügesortierenmethode genannt. Das Ganze lässt sich natürlich einfach durch die englischen Wörter insertion = Einfügen und sort = sortieren ableiten, weswegen der Sortieralgorithmus auch manchmal als Insertsort bezeichnet wird. Allgemein kann auch noch gesagt werden, dass der Sortieralgorithmus einfach zu implementieren ist und dabei bei kleiner oder schon teilweise vorsortierten Eingabemengen sehr effizient arbeitet. Da das Sortierverfahren keinen zusätzlichen Speicherplatz benötigt, arbeitet der Algorithmus in-place, was natürlich für seine Speicherplatzkomplexität spricht.



Quicksort

Video - Quicksort

https://www.youtube-nocookie.com/embed/UoJJ78K-uc0

Zunächst wird die zu sortierende Liste in zwei Teillisten („linke“ und „rechte“ Teilliste) getrennt. Dazu wählt Quicksort ein sogenanntes Pivotelement aus der Liste aus. Alle Elemente, die kleiner als das Pivotelement sind, kommen in die linke Teilliste, und alle, die größer sind, in die rechte Teilliste. Die Elemente, die gleich dem Pivotelement sind, können sich beliebig auf die Teillisten verteilen. Nach der Aufteilung sind die Elemente der linken Liste kleiner oder gleich den Elementen der rechten Liste.

Anschließend muss man also noch jede Teilliste in sich sortieren, um die Sortierung zu vollenden. Dazu wird der Quicksort-Algorithmus jeweils auf der linken und auf der rechten Teilliste ausgeführt. Jede Teilliste wird dann wieder in zwei Teillisten aufgeteilt und auf diese jeweils wieder der Quicksort-Algorithmus angewandt, und so weiter. Diese Selbstaufrufe werden als Rekursion bezeichnet. Wenn eine Teilliste der Länge eins oder null auftritt, so ist diese bereits sortiert und es erfolgt der Abbruch der Rekursion.

Die Positionen der Elemente, die gleich dem Pivotelement sind, hängen vom verwendeten Teilungsalgorithmus ab. Sie können sich beliebig auf die Teillisten verteilen. Da sich die Reihenfolge von gleichwertigen Elementen zueinander ändern kann, ist Quicksort im Allgemeinen nicht stabil.

Das Verfahren muss sicherstellen, dass jede der Teillisten mindestens um eins kürzer ist als die Gesamtliste. Dann endet die Rekursion garantiert nach endlich vielen Schritten. Das kann z. B. dadurch erreicht werden, dass das ursprünglich als Pivot gewählte Element auf einen Platz

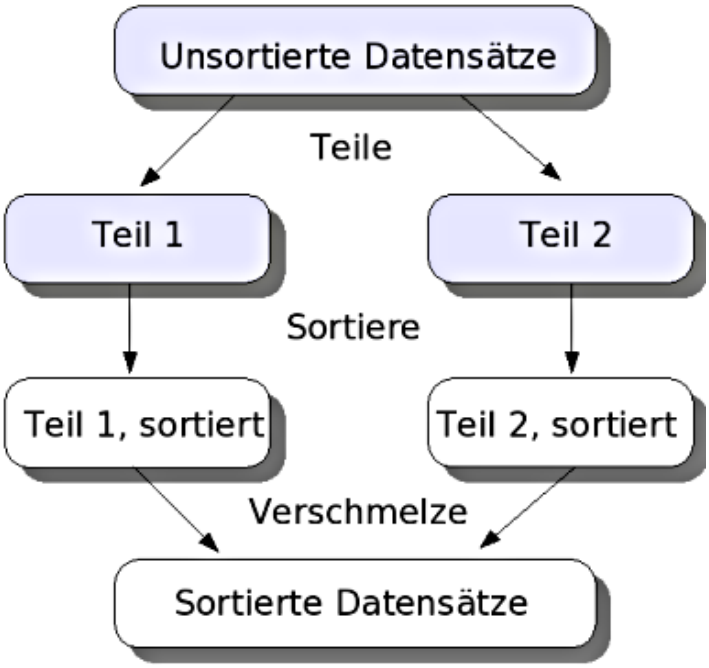
zwischen den Teillisten gesetzt wird und somit zu keiner Teilliste gehört.

Mergesort

Video - Mergesort

https://www.youtube-nocookie.com/embed/yKgzwtqWvFU

Mergesort betrachtet die zu sortierenden Daten als Liste und zerlegt sie in kleinere Listen, die jede für sich sortiert werden. Die kleinen sortierten Listen werden dann im Reißverschlussverfahren zu größeren sortierten Listen zusammengefügt, bis eine sortierte Gesamtliste erreicht ist. Das Verfahren arbeitet bei Arrays in der Regel nicht in-place, es sind dafür aber Implementierungen bekannt, in welchen die Teil-Arrays üblicherweise rekursiv zusammengeführt werden. Verkettete Listen sind besonders geeignet zur Implementierung von Mergesort, dabei ergibt sich die in-place-Sortierung fast von selbst.



Polymorphismus & Überladen

Polymorphismus

Bei Polymorphismus gibt es eine Basisklasse, von der mehrere Klassen erben. Innerhalb der Basisklasse ist eine Methode deklariert, welche aufgrund der Vererbung in den Unterklassen implementiert werden muss. Wird nun ein Objekt einer der Unterklassen erstellt und mit diesem Objekt die Methode aus der Basisklasse aufgerufen, kann das Programm anhand der Klasse, von welcher das Objekt erstellt wurde, entscheiden, welche der Implementierungen aufgerufen wird.

Überladen

Beim Überladen wird dem gleichen Operator, Literal, usw. eine unterschiedliche Bedeutung zugeordnet. Die Bedeutung geht aus dem Kontext hervor.

```
a = 1 + 2
b = "abc" + "def"
```

Bei diesem Beispiel wird der Operator "+" überladen. Je nach Kontext handelt es sich dabei um eine Stringconcatination oder um eine Addition von zwei Integern.

Programmierparadigmen

Imperative Programmierung

> Die imperative Programmierung (von lateinisch imperare = befehlen) ist das älteste Programmierparadigma. Gemäß diesem Paradigma besteht ein Programm aus einer klar definierten Abfolge von Handlungsanweisungen an einen Computer.

Der Quellcode imperativer Sprachen reiht also Befehle aneinander, die festlegen, was wann vom Computer zu tun ist, um ein gewünschtes Ergebnis zu erzielen. In Variablen eingesetzte Werte werden dabei zur Laufzeit des Programms verändert. Um die Befehle zu steuern, werden Kontrollstrukturen wie Schleifen oder Verzweigungen in den Code integriert.

Strukturierte Programmierung

Beim strukturierten Programmierungsansatz handelt es sich um eine **vereinfachte Form** der imperativen Programmierung. Die entscheidende Änderung zum Grundprinzip: Anstelle der absoluten Sprungbefehle (Anweisungen, die dazu führen, dass die Verarbeitung nicht mit dem nachfolgenden Befehl, sondern an anderer Stelle weitergeführt wird) sieht dieses Paradigma für die Software-Programmierung den **Einsatz von Kontrollschleifen bzw. -strukturen vor.**

Ein Beispiel hierfür ist die Nutzung von „**do...while**“, um eine Anweisung automatisch so lange auszuführen, wie eine bestimmte Bedingung wahr ist (mindestens ein Mal).

> Populäre Vertreter sind **C** und **C++, C#, COBOL, Fortran, Java, Pascal, Python** oder auch **Visual Basic**.

Prozedurale Programmierung

Das prozedurale Programmierparadigma erweitert den imperativen Ansatz um die Möglichkeit, **Algorithmen in überschaubare Teile aufzugliedern**. Diese werden als Prozeduren oder – je nach Programmiersprache – auch als Unterprogramme, Routinen oder Funktionen bezeichnet.

Sinn und Zweck dieser Aufteilung ist es, den **Programmcode übersichtlicher** zu machen und unnötige **Code-Wiederholungen zu vermeiden.** Durch die Abstraktion der Algorithmen stellt das prozedurale Software-Paradigma einen entscheidenden Schritt von einfachen Assemblersprachen hin zu komplexeren Hochsprachen dar.

> Beispiele für typische prozedurale Programmiersprachen sind **C und Pascal**.

Objektorientierte Programmierung

Objektorientierte Programmierung (OOP) ist ein Modell der Computerprogrammierung, bei dem das Softwaredesign auf **Daten oder Objekten basiert** und nicht auf Funktionen und Logik. Ein Objekt kann als ein **Datenfeld definiert** werden, das eindeutige Attribute und Verhaltensweisen aufweist.

Die Struktur beziehungsweise die Bausteine der objektorientierten Programmierung umfassen:

- **Klassen** sind benutzerdefinierte Datentypen, die als Blaupause für individuelle Objekte, Attribute und Methoden dienen.
- **Objekte** sind Instanzen einer Klasse, die mit speziell definierten Daten erstellt werden. Objekte können realen Objekten oder einem abstrakten Gebilde entsprechen. Bei der anfänglichen Definition einer Klasse ist die Beschreibung das einzige Objekt, das definiert wird.
- **Methoden** sind Funktionen, die innerhalb einer Klasse definiert sind und das Verhalten eines Objekts beschreiben.
- **Attribute** werden in der Klassenvorlage definiert und stellen den Zustand eines Objekts dar. Objekte haben Daten, die im Attributfeld gespeichert werden. Klassenattribute gehören zur Klasse selbst.

> „ Zu den Programmiersprachen, die hauptsächlich für die objektorientierter Programmierung entwickelt wurden, gehören **Java, Python und C++**

Deklarative Programmierung

> Kennzeichnend für die deklarativen Programmiersprachen ist, dass sie immer ein gewünschtes Endergebnis beschreiben, statt alle Arbeitsschritte aufzuzeigen. Um das Ziel zu erreichen, wird bei der deklarativen Programmierung der Lösungsweg automatisch ermittelt. Dies funktioniert so lange gut, wie die Spezifikationen des Endzustands klar definiert sind und ein passendes Ausführungsverfahren existiert. Trifft beides zu, ist die deklarative Programmierung sehr effizient.

Da die deklarative Programmierung das „Wie“ nicht festschreibt, sondern auf einem sehr hohen Abstraktionsniveau arbeitet, lässt das Programmierparadigma zudem Raum für Optimierung. Wird ein besseres Ausführungsverfahren entwickelt, lässt sich dies über den integrierten Algorithmus auffinden und verwenden. Auf diese Weise ist das Paradigma sehr zukunftssicher: Beim Schreiben des Codes muss das Verfahren, wie das Ergebnis zu erreichen ist, nicht feststehen.

Logische Programmierung

Das logische Software-Paradigma, das auch als prädikative Programmierung bezeichnet wird, **beruht auf der mathematischen Logik**. Anstelle einer Folge von Anweisungen enthält eine Software, die nach diesem Prinzip programmiert wird, eine Menge von Grundsätzen, die sich als Sammlung von Fakten und Annahmen verstehen lässt. Jegliche Anfragen an das Programm werden verarbeitet, indem der Interpreter auf diese Grundsätze zurückgreift und zuvor definierte Regeln auf diese anwendet, um zum gewünschten Ergebnis zu kommen.

> Einer der wichtigsten Vertreter der logischen Programmierung ist die Programmiersprache **Prolog**.

Funktionale Programmierung

Im Mittelpunkt der funktionalen Herangehensweise beim Programmieren stehen **Funktionen**. Bei einem funktionalen Programm können alle Elemente als Funktion aufgefasst und der Code kann durch **aneinandergereihte Funktionsaufrufe** ausgeführt werden. Im Umkehrschluss gibt es keine eigenständigen Zuweisungen von Werten.

> Zu den wichtigsten Programmiersprachen, die auf dem funktionalen Ansatz basieren, zählen **LISP, Haskell, F#** oder **Erlang**

CSS - Cascading Style Sheet

Definition von CSS

CSS (Cascading Style Sheet) ist eine Möglichkeit das Aussehen von HTML-Elementen zu beeinflussen. Also die Schriftart, -größe, -farbe, die Umrandung, sowie den Abstand zwischen den einzelnen Elementen und zur Umrandung.

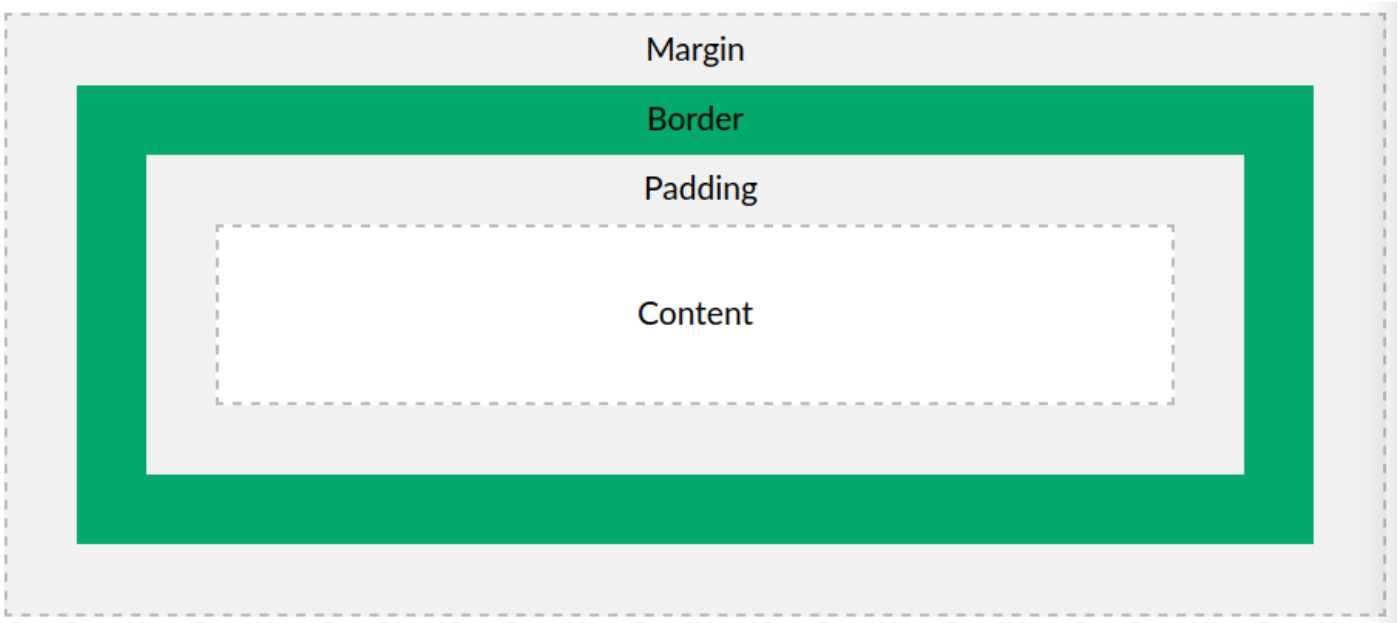
Es gibt drei Möglichkeiten in denen man speichern kann wie die einzelnen Elemente dargestellt werden sollen:

- externe CSS-Datei
- im "head" eines HTML-Files
- direkt im Attribut

Dabei gibt es zwei Grundsätze zu beachten:

- möglichst viele über die externe CSS-Datei anpassen, dass erleichtert spätere Veränderungen
- je dichter am Element eine Anpassung ist, desto höher die Priorität bei der Darstellung

Box-Modell



- Content: Inhalt der Box, hier erscheinen Text und Bilder (height, width)

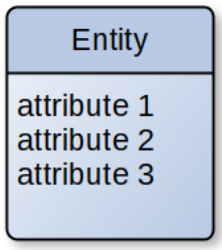
- Padding: Klärt ein Gebiet um den "content" herum und erscheint transparent (top, right, bottom, left)
- Border: Rahmen um Content und Padding, ist sichtbar und kann verändert werden (size)
- Margin: Klärt ein Gebiet um die "border" herum und erscheint transparent (top, right, bottom, left)

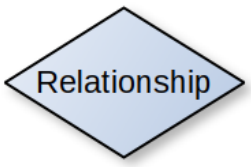
> Bei zwei benachbarten Elementen gilt die größere Margin. Sollte das eine Element eine margin von 20px und das andere Element eine margin von 30px haben, dann haben die beiden border einen Abstand von 30px (nicht 50px).

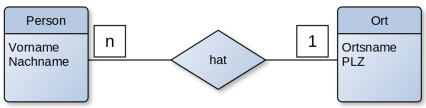
Entity-Relationship-Modell

Entity-Relationship-Modell (kurz: ER-Modell oder ERM)

- Modell zur Darstellung von Dingen, Gegenständen und Objekten, sowie deren Beziehung und Zusammenhänge
- Begriffe:
  - Entiät (Entity): individuell identifizierbares Objekt der Wirklichkeit (Gegenstand, Ding, Objekt)
  - Beziehung (Relationship): Verknüpfung / Zusammenhang zwischen zwei oder mehr Entitäten
  - Eigenschaften (Attribut): Was über eine Entität im Kontext wichtig ist
- es gibt unterschiedliche Notationen (=Darstellungsformen) für ein ER-Modell. In der Berufsschule wurden die Chen-Notation und die Krähenfuß-Notation verwendet
- verwendete Symbole:





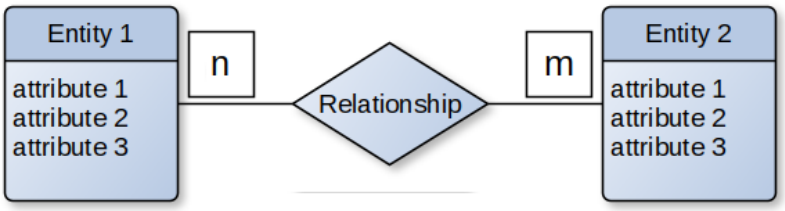


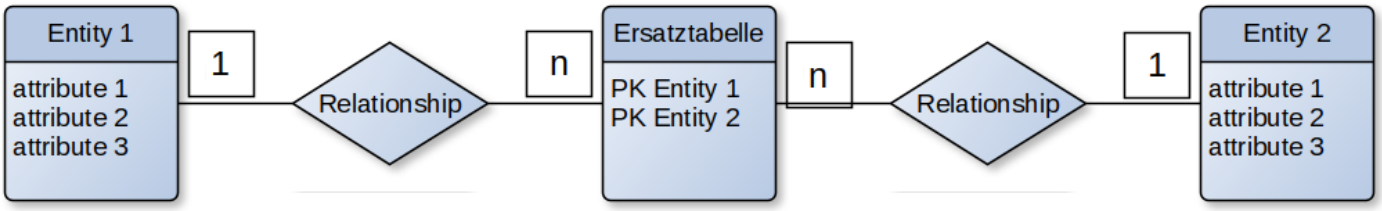
Chen-Notation

1:1	Jede Entität der **ersten** Tabelle steht mit **genau einer** Entiät der **zweiten** Tabelle in Beziehung. Jede Entität der **zweiten** Tabelle steht mit **genau einer** Entität der **ersten** Tabelle in Beziehung.
1:n	Jede Entiät der **ersten** Tabelle steht mit **mindestens einer** Entität der **zweiten** Tabelle in Beziehung. Jede Entität der **zweiten** Tabelle steht mit **genau einer** Entität der **ersten** Tabelle in Beziehung.

n:m	Jede Entität der **ersten** Tabelle steht mit **mindestens einer** Entität der **zweiten** Tabelle in Beziehung. Jede Entität der **zweiten** Tabelle steht mit **mindestens einer** Entität der **ersten** Tabelle in Beziehung. > Eine solche Beziehung ist nur im ER-Modell darstellbar und muss bei der Umwandlung für die Datenbank aufgelöst werden. Dazu wird eine dritte Tabelle geschaffen, die die Primary Keys der beiden ursprünglichen Tabellen enthält. Die neue Tabelle steht mit den alten Tabellen in einer 1:n Beziehung.

Auflösen einer n:m Beziehung



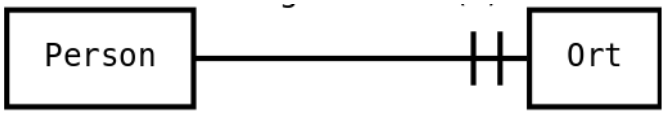
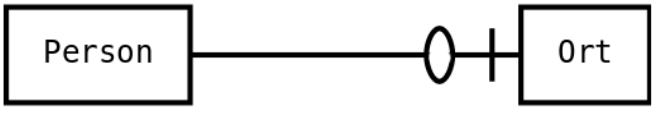


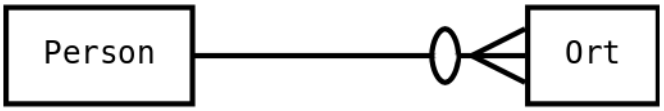
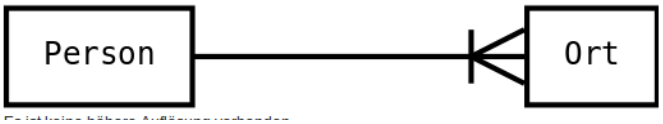
Erweiterte Chen-Notation

- in der erweiterten Chen-Notation wird der Buchstabe "c" eingeführt. Dieser kann für "0" oder "1" stehen
- Kombinationen sind mit allen oben genannten Beziehungen möglich

Krähenfuß-Notation / Martin-Notation

- in dieser Notation werden vier unterschiedliche Symbole verwendet um die Beziehung von einer Tabelle zur anderen darzustellen

Symbolik	**Bedeutung**
	Person hat **genau eine** Beziehung zu Ort
	Person hat **maximal eine** Beziehung zu Ort

	Person hat **beliebig viele** Beziehung zu Ort
	Person hat **mindestens eine** Beziehung zur Ort