

AEuP

Schuljahr 2021/2022

Inhaltsverzeichnis

1.	EINFÜHRUNG.....	4
1.1.	BIG PICTURE DES FACHES ANWENDUNGSENTWICKLUNG UND PROGRAMMIERUNG	4
1.2.	ENTWICKLUNG DER PROGRAMMERSPRACHEN	5
1.3.	ÜBERBLICK DER HÖHEREN PROGRAMMERSPRACHEN	7
2.	GRUNDLAGEN.....	9
2.1.	BESTANDTEILE EINER PROGRAMMERSPRACHE	9
2.2.	PROGRAMMIERWERKZEUGE.....	10
2.3.	ÜBERSETZUNGSSARTEN	11
2.4.	BUGGING UND DEBUGGING	13
2.5.	ZUSAMMENFASSUNG	14
3.	GRUNDLAGEN PYTHON.....	15
3.1.	HELLO WORLD	15
3.2.	AUFBAU EINES PYTHON PROGRAMMS UND BLÖCKE.....	16
3.3.	KOMMENTARE	17
3.3.1.	ÜBUNG.....	17
3.4.	EINGABE UND AUSGABE FUNKTION	18
3.5.	ELEMENTARE SPRACHBESTANDTEILE	20
3.5.1.	VARIABLEN UND KONSTANTEN	20
3.5.3.	TYPKONVERTIERUNG	24
3.5.4.	OPERATOREN	25
3.6.	GEMISCHTE AUFGABE	27
4.	LISTEN, DICTIONARY, TUPEL.....	30
4.1.	EINDIMENSIONALE LISTEN	31
4.1.1.	BEARBEITEN VON LISTEN	33
4.1.2.	WEITERE METHODEN AUF LISTEN.....	34
4.1.3.	SLICEN	35
4.1.4.	ÜBUNGEN ZU LISTEN	36
4.2.	DICTIONARY.....	38
4.2.1.	ERSTELLEN EINES DICTIONARIES:	39
4.2.2.	BEARBEITEN VON DICTIONARIES:	40
4.3.	TUPEL	42
4.3.1.	ERSTELLEN EINES TUPELS	42
5.	KONTROLLSTRUKTUREN.....	43

5.1. STRUKTOGRAMME	44
5.1.1. ÜBUNGEN STRUKTOGRAMME.....	45
5.2. KONTROLLSTRUKTUREN – ALLGEMEIN	51
5.2.1. DIE FOR-SCHLEIFE	52
5.2.2. DIE IF-ANWEISUNG.....	54
5.2.3. DIE WHILE-SCHLEIFE	59
5.2.4. ÜBUNGEN ZU WHILE-SCHLEIFEN	60
5.2.5. GEMISCHTE ÜBUNGEN.....	61
5.3. ZUSAMMENFASSUNG KAPITEL 4 UND KAPITEL 5	62
 6. FUNKTIONEN	 64
 ARBEITSAUFRAG:	 68
6.1. FUNKTIONEN MIT VARIABLER PARAMETERZAHL	70
6.1.1. FUNKTIONENBEZEICHNER *ARGS & **KWARGS	71
➤ EXKURS: FORMATIERUNG VON DICTIONARIES	72
6.2. RÜCKGABEWERT BEI FUNKTIONEN.....	73
6.2.1. WARUM VARIABLE ÜBER RETURN ÜBERGEBEN.....	74
6.2.2. GELTUNGSBEREICH VON VARIABLEN	75
6.2.3. GLOBALE VARIABLEN	77
6.3. REKURSIVE FUNKTIONEN.....	78
6.4. MAIN METHODE.....	80
.....	80
6.5. ÜBUNGEN ZU FUNKTIONEN	81
 7. OBJEKTORIENTIERUNG.....	 82
 7.1. KLASSEN UND OBJEKTE.....	 84
7.1.1. KLASSE DEFINIEREN UND INITIALISIEREN	87
7.1.2. KONSTRUKTOREN EINER KLASSE.....	88
7.1.3. OBJEKTE (INSTANZ) EINER KLASSE DEFINIEREN.....	90

1. Einführung

1.1. Big Picture des Faches Anwendungsentwicklung und Programmierung



20 min.

1. Welche Erwartungen haben Sie an das Fach Anwendungsentwicklung?
<https://www.oncoo.de/59j0>
2. Überlegen Sie sich Stichpunkte, die Sie im Rahmen der Programmierung für wichtig erachten.

10. Klasse

Grundlagen der
Programmierung

- Grundbegriffe
- Programmierbasics

11. Klasse

Datenbanken
↳ SQL

12. Klasse

Programmierprojekt

1.2. Entwicklung der Programmiersprachen

Programmiersprachen haben sich im Verlauf der Zeit stetig weiterentwickelt. Allen Programmiersprachen ist jedoch gemein, dass diese letztlich in Maschinensprache übersetzt werden müssen, so dass der Computer, genauer betrachtet das Rechenwerk in Kooperation mit dem Steuerwerk, die gewünschten Befehle verstehen und ausführen können.

Arbeitsauftrag 1:



10 min.

Leider sind folgende Generationen von Programmiersprachen mit den jeweiligen Erklärungen durcheinandergeraten. Ihre Aufgabe ist es daher, die jeweilige Generation mit der jeweils passenden Erklärung durch eine Linie zu verbinden.

Unterscheidung der Generationen

Name	Beschreibung / Beispiel
1. Generation: Maschinensprache	Wurden dazu entwickelt, um Programme schreiben zu können, die menschliches Denken nachahmen.  Beispiel: LISP, PROLOG, Java, C#
2. Generation: Assemblersprache	Sprachen, die für ein bestimmtes Anwendungsgebiet (z. B. Datenbanksprache SQL) geschaffen wurden.  Beispiel: SQL (Select * FROM kunde)
3. 4. Generation: Höhere prozedurale Sprachen	Binärkode, maschinennahe Programmierung direkt auf dem Computer lauffähig, schwer verständlich.  Beispiel: 0001 1010 0011 0100 0001 0000 1101
5. Generation: Deskriptive (Beschreibende) Sprachen	Assemblersprache / leichter verständliche Symbole, nicht direkt auf dem Computer lauffähig.  Beispiel: mov ds / mov ax

6. Generation:
Wissens- und
objektorientierte
Programmiersprache

Prozeduraler Aufbau in Funktionen; der menschlichen Sprache angenähert; unabhängig vom
Computersystem



Beispiel: C++, C#, Python, Java

1.3. Überblick der höheren Programmiersprachen

Wir werden uns in AWP mit den „höheren Programmiersprachen“ beschäftigen, insbesondere mit Python. Andere Programmiersprachen von denen Sie vielleicht gehört haben, sind C,C++,Perl und Java.

Um einen kurzen Überblick über das Programmiersprachenuniversum zu erhalten, folgende Aufgabe

Arbeitsauftrag 1:



Erstellen Sie in Ihrer Gruppe mit PowerPoint (**2 Folien!**) eine Übersicht über die von Ihnen gewählte Programmiersprache.

Die Folien sollen einem Plakat entsprechen und einerseits mögliche Einsatzszenarien darstellen sowie Vor- und Nachteile nennen. Gerne können Sie auch ein kleines Codefragment oder einen Screenshot in Ihre Folien integrieren. Achten Sie hierbei darauf, die Daten grafisch anspruchsvoll, gerne auch aufgelockert, darzustellen. Bereiten Sie sich zudem darauf vor, Ihre Folien Ihren Mitschülern später erklären zu können.

Programmiersprache	Gruppe
Python	
Java	
C#	
C++	
JavaScript	
PHP	
ABAP	
SQL	
XML	
CSS	
HTML	

Arbeitsauftrag 2:



10 min.

Beantworten Sie nun schriftlich und in Stichpunkten die Frage, welche Sprache ihrer Meinung nach die beste Programmiersprache ist.



Platz für Ihre Lösung

- nicht pauschal festlegbar.
- abhängig vom Anwendungsfall & Vorlieben

Arbeitsauftrag 3:



10 min.

Schauen Sie zudem nach, ob Sie Ihre gewählte Sprache im *Tiobe Index* wiederfinden. Natürlich recherchieren Sie dabei, was der *Tiobe Index* überhaupt ist und notieren Ihre Erkenntnis im nachfolgenden Feld.
<https://www.tiobe.com/tiobe-index/>

Platz für Ihre Lösung

2. Grundlagen

2.1. Bestandteile einer Programmiersprache

Programmiersprachen sind den natürlichen Sprachen wie Deutsch oder Englisch sehr ähnlich. Sie werden allerding im Unterschied zu den natürlichen Sprachen künstlich entworfen. Der Aufbau einer Programmiersprache unterliegt strengen Regeln und lässt sich anhand verschiedener Informationen genau definieren. Zu Den Informationen zählen die Syntax, die Schlüsselwörter und der Zeichensatz.

Damit wir einen gemeinsamen Grundstock aufbauen, bearbeiten Sie die folgende Aufgabe.

Arbeitsauftrag:



10 min.

Beschreiben Sie Stichpunktartig die untenstehenden Begriffe. Sie können die Aufgabe gerne in Partnerarbeit lösen.

Bestandteile einer Programmiersprache	
Syntax	<ul style="list-style-type: none">- Grammatik einer Programmiersprache- Beschreibt sowohl zulässige Sprachelemente, als auch wie sie verwendet sind
Schlüsselwörter	<ul style="list-style-type: none">- "Vokabeln" der Programmiersprache- Reservierte Wörter, die einen bestimmten Zweck erfüllen und nicht als Variablennamen verwendet werden können
Zeichensatz	<ul style="list-style-type: none">- Vorrat aus festgelegten alphabetischen & nummerischen Zeichen, sowie Sonderzeichen

2.2. Programmierwerkzeuge

Um ein Programm zu schreiben, bedarf es nicht vieler Programmierwerkzeuge. Im einfachsten Fall reicht ein Editor, ein Compiler oder ein Interpreter.

Wenn die Programme umfangreicher werden, dann sollte über den Einsatz einer integrierten Entwicklungsumgebung (DIE) nachgedacht werden. In einer DIE sind neben einem Texteditor mit Syntaxhervorhebung, einem Compiler oder Interpreter auch Debugger, Profiler und Versionsverwaltung unter einer gemeinsamen Oberfläche zusammengefasst.

Um einen kurzen Überblick über die wichtigsten Werkzeuge zu erhalten, eine kleine Aufgabe.

Arbeitsauftrag:



15 min.

Beschreiben Sie Stichpunktartig die untenstehenden Begriffe. Sie können die Aufgabe gerne in Partnerarbeit lösen.

Wichtige Programmierwerkzeuge	
Texteditor	- Zur Erstellung und Änderung des Quelltextes von Programmen
GUI-Editor	Vereinfacht die Erstellung grafischer Oberflächen z.B. Drag & Drop
Compiler, Interpreter,	Übersetzt den Quellcode in für die Maschine lesbare Anweisungen
Debugger	Werkzeug zum diagnostizieren & auffinden von Fehlern
Versionsverwaltung	Erfassung von Änderungen an Dokumenten oder Dateien

2.3. Übersetzungsarten

Neben den oben erwähnten höheren Programmiersprachen gibt es auch niedere Programmiersprachen. Vereinfacht ausgedrückt, können Computer nur Programme ausführen, die in niederen Programmiersprachen geschrieben wurden. Programme, die in höheren Programmiersprachen geschrieben wurden, müssen deshalb zunächst verarbeitet werden, bevor sie ausgeführt werden können. Diese Verarbeitung benötigt ein bisschen Zeit, was ein kleiner Nachteil von höheren Programmiersprachen ist.

Natürlich haben höhere Programmiersprachen aber auch einige Vorteile:

- sie lassen sich schneller schreiben
- sie sind kürzer und einfacher zu lesen
- sie sind mit größerer Wahrscheinlichkeit richtig
- sie sind portierbar, d.h. Programme, die mit ihnen geschrieben wurden, können mir nur wenigen oder gar keinen Änderungen auf verschiedenen Arten von Computern ausgeführt werden

(mit niederen Programmiersprachen geschriebene Programme können nur auf einer Art von Computern ausgeführt werden und müssen für andere Computertypen neu geschrieben werden.)

Für die Verarbeitung von höheren in niedere Programmiersprachen sind zwei Arten von Programmen erforderlich: **Interpreter und Compiler**

Arbeitsauftrag:



20 min.



- 1) Schauen Sie aufmerksam das Video (<https://www.youtube.com/watch?v=F1GLYZ7fhvw&feature=youtu.be>)
- 2) Beschreiben Sie die Funktionen von Compiler und Interpreter
- 3) Recherchieren Sie für jede Übersetzungsart zwei Programmiersprachen
- 4) Stellen Sie die Funktionsweise von Interpreter und Compiler grafisch dar.

2) Funktionen von Interpreter und Compiler.

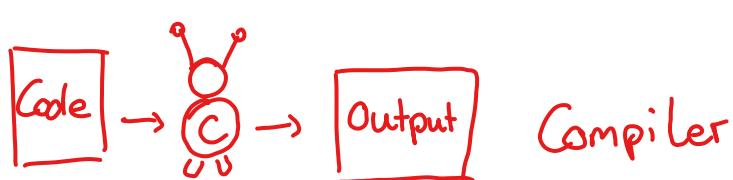
<u>Interpreter</u>	<u>Compiler</u>
<ul style="list-style-type: none"> - übersetzt den Code zeilenweise - langsamer als der Compiler - Ausbesserung von Fehlern möglich, da zeilenweise übersetzt wird. 	<ul style="list-style-type: none"> - übersetzt den Code in einem schnelleren Prozess - Fehler können nicht direkt ausgebessert werden.

3) Recherchieren Sie für jede Übersetzungsart zwei Programmiersprachen

-Interpreter : Python, Java, PHP, LISP

-Compiler: C#, C++

4) Stellen Sie die Funktionsweise von Interpreter und Compiler grafisch dar.



2.4. Bugging und Debugging

Beim Programmieren werden Sie bald feststellen, dass immer wieder Fehler – sogenannte Bugs – auftauchen. Das ist ganz normal und gehört zur täglichen Arbeit dazu. Der Vorgang diese Fehler aufzuspüren, wird als Debugging bezeichnet.

In einem Programm gibt es drei Arten von Fehlern: Syntaxfehler, Laufzeitfehler und semantische Fehler.

Um in Zukunft beim Programmieren diese Fehler leichter erkennen zu können, lohnt es sich diese Fehler unterscheiden zu können.

Arbeitsauftrag:



10 min.

1. Informieren Sie sich über die Fehlerarten: Syntaxfehler, semantischer Fehler und Laufzeitfehler
2. Beschreiben Sie die drei Fehlerarten
3. Finden Sie Beispiele für die Fehlerarten.

Syntaxfehler	<p><i>Der Code entspricht nicht der Grammatik der Programmiersprache</i></p> <p><i>Bsp: Rechtschreibfehler</i></p>
Semantischer Fehler	<p><i>Logische Fehler</i></p> <p><i>Bsp: Punkt-vor-Strich Rechnen</i></p>
Laufzeitfehler	<p><i>Tritt auf, wenn das Programm bereits läuft</i></p> <p><i>Bsp: Null-Pointer / Endlosschleife</i></p>

*Syntaktisch richtig
aber Fehler in der Logik*



1. Überlegen Sie sich nun in Partnerarbeit, welches Ihrer Meinung nach, der komplizierteste Fehler ist und auf welche Probleme wir dabei stoßen können.

Platz für Ihre Lösung

Semantischer Fehler

2.5. Zusammenfassung

Im Verlauf des Schuljahres werden wir am Ende eines jeden Kapitels die wichtigsten Punkte zusammenfassen und die Themen nochmals aufgreifen, bei denenklärungsbedarf besteht. Hierfür erhalten Sie folgende Aufgabe.

Arbeitsauftrag:



30 min.

1. Arbeiten Sie in Gruppen zusammen.
2. Fassen sie hier die für Sie wichtigsten Punkte stichpunktartig zusammen.
3. Notieren Sie zudem die Punkte, die Ihnen noch unklar erscheinen.

Platz für Ihre Ideen

3. Grundlagen Python

3.1. Hello World

Mit welchem Editor Sie programmieren wollen, bleibt Ihnen überlassen und ist Geschmackssache. Wir verwenden zum einheitlichen Lernen in der Schule den Editor Visual Studio Code.

Traditionell heißt das erste Programm, das Sie in einer neuen Sprache schreiben „Hallo Welt!“.



15 min.

1. Legen Sie sich auf Ihrem PC einen neuen Ordner für die Programmierung ab.
2. Erstellen Sie mittels VS-Code eine Datei namens „Hello World.py“
3. Recherchieren Sie sich einen Befehl, der Ihnen auf der Console „Hello World“ ausgibt.



10 min.

Sie haben nun die erste Funktion in Python kennengelernt. Notieren Sie diese und probieren Sie die Funktion weiter aus, indem Sie unterschiedliche Varianten eingeben, die Ihnen einfallen.



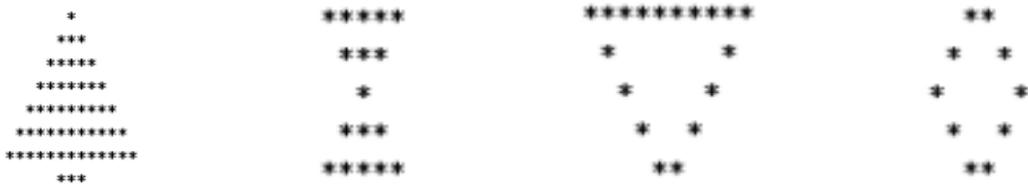
print(...)

Arbeitsauftrag: Übung macht den Meister.



15 min.

1. Erstellen Sie eine neue Datei „Symbole.py“
2. Lassen Sie sich mit der gelernten Funktion **zwei** der folgenden Symbole ausgeben.



3.2. Aufbau eines Python Programms und Blöcke

Grundsätzlich besteht ein Python-Programm aus einzelnen Anweisungen oder Anweisungsblöcken. Im einfachsten Fall, wie in „Hello World“, enthält das ganze Programm nur eine Anweisung, welche interpretiert und ausgeführt wird. Die Anweisung für „Hello World“ lautet:

```
print("Hello World") oder print('Hello World')
```

Allerdings besteht ein Programm in der Regel aus mehr als einer Anweisung also aus mehreren Anweisungsblöcken. Wie solche Blöcke bei Python aufgebaut sind, schauen wir uns nun genauer an. Die Strukturierung von Programmblöcken unterscheidet sich von anderen Sprachen. Während das Einrücken bei vielen Programmsprachen der besseren Lesbarkeit dient, ist dies bei Python unbedingt notwendig. Der Vorteil dabei, wir haben direkt sauber lesbaren Quellcode.

Die allgemeine Syntax eines Anweisungsblocks sieht wie folgt aus:

Beispiel

```
if a > b:
    a = a + 2
    print(a)
else:
    b = b - 3
    print(b)
```

The diagram illustrates the structure of the provided Python code. It shows two if-else blocks. Each block consists of an 'Anweisungskopf' (statement header) and an 'Anweisungskörper' (statement body). The 'Anweisungskopf' for the first block is 'if a > b:' and for the second is 'else:'. The 'Anweisungskörper' for the first block contains the statements 'a = a + 2' and 'print(a)'. The 'Anweisungskörper' for the second block contains the statement 'b = b - 3'.

Üblicherweise müssen sie bei Python keine Trennzeichen, wie etwa ein Semikolon verwenden. Um einzelne Blöcke voneinander abzutrennen wird der Unterblock eingerückt.

3.3. Kommentare

Bevor es los geht, wollen wir uns erst noch ein wichtiges Thema anschauen – die Kommentare. Kommentare dienen dazu, Notizen oder Bemerkungen direkt in den Quellcode mit aufzunehmen. Der Text in Kommentaren unterliegt keinen Einschränkungen und kann beliebig gestaltet werden. Alle Kommentare werden beim Ausführen des Programms ignoriert.

Arbeitsauftrag 3:



Überlegen Sie sich kurz, welchen Nutzen Kommentare bei der Entwicklung mit sich bringen

Beispiele für Kommentare:



Notieren Sie Beispiele für das Schreiben von Kommentaren. Unterscheiden Sie dabei:

- a) Einzeilig
- b) mehrzeilig

a) einzeilig: #

b) mehrzeilig: """

3.3.1. Übung



15 min.

Bearbeiten Sie die untenstehenden Übungsaufgaben.

1. Im nachfolgendem Quellcode sind sechs Syntaxfehler enthalten.
Finden Sie diese **ohne PyCharm**.

```
1 """print('Das ist mein erstes Programm')"""
#print('-----')
Print(Hello World);1 print("Das hat geklappt")
```

*↳ ein Semikolon ist zulässig um
mehrere Argumente in eine Zeile
zu schreiben*

3.4. Eingabe und Ausgabe Funktion

Arbeitsauftrag 1:

Sie machen Praktikum im Tierheim „WeCare“ in Kitzingen und sollen sich um die Digitalisierung kümmern. Ihr erster Auftrag ist es, eine Abfrage für die Neuanmeldung eines Tieres anzulegen.

Vorab: Der Fragebogen soll den Namen des Hundes und das Alter abfragen. Zusätzlich soll auch berechnet werden können, wie alt der Hund in Menschenjahren ist.

Skizzieren Sie einen Pseudocode in deutscher Sprache, der Ihren Code widerspiegeln soll. Am Ende des Fragebogens soll folgender Satz stehen: „Der Hund Maxi ist umgerechnet 70 Jahre alt.“



Platz für Ihre Ideen...

input (xy)
print (xy)

Hundalter in Menschenjahre = Hundalter . 7



Nachdem Sie sich den Pseudocode überlegt haben, setzen Sie ihn in VS-Code um.

Notieren Sie hier die Funktionen, die Sie für die Implementierung benötigen

- print
- input

Notieren Sie, welche Variablen Sie anlegen müssen:

name
alter \Rightarrow input
age_in_Human \rightarrow Rechnung

Notieren Sie, welche Herausforderungen bei der Umsetzung aufgetreten sind.

Datentypen beim print: String Trennzeichen \rightarrow + oder ,
Int Trennzeichen \rightarrow ,
Input age: Input muss in einen Integer umgewandelt werden.
Input ist standardmäßig ein String

Schon fertig? Dann gibt es eine weitere Aufgabe:

Addition zweier Zahlen: Es soll ein Programm entwickelt werden, in das der Benutzer zwei ganze Zahlen eingeben kann und welches danach die Summe dieser Zahlen berechnet und ausgibt. Die Ausgabe im Konsolenfenster soll wie folgt aussehen:

Überschrift: Berechnung der Summe von zwei ganzen Zahlen
Ausgabe im Fenster:
Geben Sie die erste Zahl ein: 23
Geben Sie die zweite Zahl ein 15
Die Summe der beiden Zahlen ist 38.

3.5. Elementare Sprachbestandteile

3.5.1. Variablen und Konstanten

Variablen

Wie in unserem Programm zu sehen ist, müssen Werte, welche in ein Programm eingegeben wurden, erst einmal gespeichert werden, damit mit diesen gearbeitet werden kann. Dafür werden **Variablen** verwendet. Generell lässt sich sagen, das Variablen zum Speichern von Informationen dienen. In vielen anderen Programmiersprachen werden Variablen mit einem Datentyp und einem frei wählbaren Namen versehen. Python löst dies etwas anders. In Python haben Variablen zunächst keinen bestimmten Datentyp und müssen somit nicht extra deklariert werden. Der Name ist frei wählbar, allerdings muss ein gültiger Bezeichner gewählt werden. Das heißt, ein Bezeichner darf nur alphanumerische Zeichen und Unterstriche enthalten. Leerzeichen und Sonderzeichen sind nicht erlaubt.

Variablen können an beliebiger Stelle innerhalb einer Klasse oder Methode angelegt und verwendet werden. Um die Lesbarkeit des Quelltextes zu erhöhen, wird aber empfohlen, dies jeweils am Anfang einer Klasse oder Methode zu tun.

Konstanten

Konstanten, bei denen die Werte nach der Zuweisung nicht mehr geändert werden können, gibt es in Python nicht. Hier gilt nur die Regel, dass man normale Variablen verwendet und diese nach der Initialisierung nicht mehr verändert. Das hängt aber stark von der Disziplin des Programmierers ab und wird vom Interpreter nicht überprüft. Kenntlichmachen kann man solche „speziellen“ Variablen, indem man sie z.B. nur mit großen Buchstaben schreibt.

Arbeitsauftrag



10 min.

2. Erläutern Sie kurz den Unterschied zwischen einer Variable und einer Konstante.
- Notieren Sie Ihr Ergebnis.
 - Überlegen Sie sich je ein Beispiel.

Variable	Konstante
<p>- sind veränderbar</p> <p>- dient zum Speichern von Werten</p>	<p>- Wert verändert sich nicht</p> <p>↳ Python unterscheidet hier nicht</p>

Beispiel:

$x = 100$

=> Angabe eines Datentyps ist nicht notwendig

- Großschreibung kennzeichnet Konstanten

Bsp:

$MAX_VALUE = 120$



Übung: Entscheiden Sie ob die Variablen gültig sind oder nicht.

Variable	gültig	ungültig
<code>some_value = "abc"</code>	✗	
<code>1number = 12.3</code>		✗
<code>some\$signs% = "&^%"</code>		✗
<code>go_2_home = "ok"</code>	✗	
<code>go_2_Home = "ok"</code>	✗	
<code>else = "yes"</code>		✗
<code>so_gehts = 7</code>		✗
<code>Aber so_gehts = 7</code>		✗



Regel für Variablen

- ✓ keine Schlüsselwörter als Variablenname
- ✓ keine Sonder-/Leerzeichen
- ✓ alphanumerische Zeichen nicht zu Beginn der Variable
- ✓ Datentypen werden automatisch zugewiesen (Python)



Achtung: nicht verwendbare Bezeichner für Variablen

Schlüsselwörter				
and	as	assert	break	class
continue	def	del	elif	else
except	False	finally	for	from
global	if	import	in	is
lambda	None	nonlocal	not	or
pass	raise	return	try	True
while	with	yield		

3.5.2. Datentypen

Wie zuvor erwähnt, müssen bei der Variablen Deklaration in Python keine Datentypen angeben werden (anders als z.B. in C # oder Java). Python kennt diese Datentypen intern aber trotzdem. So ist es u.a. manchmal notwendig, Variablen in einen anderen Datentyp zu wandeln. Deshalb ist es wichtig, die einzelnen Datentypen von Python zu kennen. Es gibt die folgenden drei elementaren Datentypen: Zahlen (numerische Datentypen), Zeichen und Boolesche Datentypen.



10 min.

1. Beschreiben Sie die untenstehenden Datentypen.
2. Nennen Sie jeweils ein Beispiel.

Numerische Datentypen

In Python werden die numerischen Datentypen in Ganzzahl-, Gleitkommazahl- und komplexe Datentypen unterteilt.

Datentyp	Beschreibung / Art	Beispiel
Int	Ganzzahliger Datentyp (auch Minus)	number = 12
Float	- Zahlen mit Nachkoma (Gleitkommazahl) - als Trennzeichen wird ein Punkt verwendet	price = 12.99
Complex	Datentyp für komplexe Zahlen Der Datentyp erhält einen realen & imaginären Teil	2+4j

Boolesche Datentypen

- Gibt entweder True oder False zurück

Bsp: Loginname_ok = True

Zeichentypen (Strings)

Zeichenketten & Zeichen werden als Strings gespeichert

Bsp: lastname = "Müller"

3.5.3. Typkonvertierung

Manchmal ist es notwendig, den Typ einer Variablen oder eines Literals in einen anderen Typ umzuwandeln. Es soll z.B. ein Gleitkomma-Datentyp in einen Ganzzahl-Datentyp umgewandelt werden. Dieses kann mithilfe der Typkonvertierung geschehen.

Dazu stellt Python entsprechende Funktionen zur Verfügung:



Arbeitsauftrag: Beschreiben Sie die Typkonvertierungen.

Typkonvertierung	Beschreibung
int(...) a = int(2.345) → a = 2	Wandelt einen Float in einen Int um. int(float("55.0"))
float(...) a = float(2) → a = 2.0	Wandelt einen Integer in einen Float um.
str(...) a = str(2) → a = „2“	Wandelt einen Integer / Float in einen String um



10 min.

Entscheiden Sie, ob die folgende Typkonvertierung möglich ist und wenn ja, welchen Wert die Variable annimmt.

Typkonvertierung	gültig	Ausgabe?
a = int(2.3456)	x	a = 2
a = int(„55“)	x	a = 55
a = int(„55.45“)		Fehler
a = int(„Hallo“)		Fehler
a = float(2)	x	a = 20
a = float(„55“)	x	a = 55.00
a = float(„55.45“)	x	a = 55.45
a = float(„Hello World“)		Fehler
a = str(2)	x	a = "2"
a = str(5.44)	x	a = "5.44"
a = str(„Hello World“)		Fehler

3.5.4. Operatoren

Damit Variablen miteinander agieren können, benötigen sie sog. Operatoren. Diese führen bestimmte Aktionen auf den sog. Operanden aus. Dies können Variablen oder sonstige Ausdrücke sein. Ein Ausdruck ist beinahe alles, was einen Wert hat. Variablen, Ziffern, Text sind Beispiele für Ausdrücke. In den nachfolgenden Abschnitten schauen wir uns die wichtigsten Typen von Operatoren an.



10 min.

Erläutern Sie die einzelnen Operatoren

Rechen-, String- und Vergleichsoperatoren

Operator	Funktion
+ -	Addition, Subtraktion / einfache Vorzeichen
+ - * /	Division / Multiplikation
//	ganzzahlige Division ohne Nachkommastellen
%	Modulo → Ist der Rest der Division
**	Gibt Potenzen an ($a^{**}b = a^b$)
+= -= *= /=	Zuweisung mit Addition, Subtraktion, usw. $a += 3 \rightarrow a = a + 3$
!= <= >= ==	- ungleich - kleiner/größer gleich - gleich ⇒ Vergleich zweier Werte

Bit-Operatoren

Operator	Funktion
&	Binäres und
	Binäres oder

Ergebnistabelle

x	y	x & y	x y
1	1	1	1
1	0	0	1
0	1	0	1
0	0	0	0

Boolsche Operatoren (logische Operatoren)

Operator	Funktion
Not	logisches nicht
And	logisches und
Or	logisches oder

Ergebnistabelle

x	y	x and y	x or y
True	True	True	True
True	False	False	True
False	True	False	True
False	False	False	False

3.6. Gemischte Aufgabe

1. Legen Sie für alle einzelnen Daten in der Tabelle eine Variable an. Wählen Sie sinnvolle Variablennamen. Bestimmen sie den Datentyp

Mitarbeiterdaten	
Laufende Nummer	12 int
Name des Mitarbeiters	Heidi Sommer string
Alter des Mitarbeiters	28 int
Adresse	Mühlenweg 1a, 01069 Dresden string
Telefonnummer	0177-123456789 string
Beruf	Fachinformatikerin string
Familienstand	Ledig string
Kinder	0 int
Gehalt	2356.99 float

2. Welchen Wert hat x nach jeder Anweisung. Gegeben ist x = 12.

a) $x+=12$	24
b) $x*=12$	144
c) $x-=12$	0
d) $x/=12$	1
e) $x\%=12$	0
f) $x=((x+x)-(x*x))/4$	0

3. Geben Sie an, ob das Ergebnis „true“ oder „false“ ist.

Gegeben: x1= true x2 = false

a) Ergebnis = x1 and x2	false
b) Ergebnis = not x1 or x2	false
c) Ergebnis = not x1 and not x2	false
d) Ergebnis = (x1 or x2) and (not x1 and x2)	false

4. Folgender Text soll ausgegeben werden:

„Dies ist eine Testausgabe.

Zum Beenden des Programms drücken Sie eine beliebige Taste“

5. Schreiben Sie ein Programm, welches die folgende Ausgabe erzeugt:

Tabelle: Preisliste

Artikel	Preis
Butter	1,99
Hose	133,55
Tagescreme	11,89

Durch \n kann man einen Zeilenumbruch
in einem String erschaffen

6. Einkaufskalkulation:

Der Benutzer soll zu Beginn den Listeneinkaufspreis und die Bezugskosten eingeben. Mit Hilfe von Python soll dann der Bezugspreis berechnet werden. Der Rabatt beträgt 10%, der Skonto 2%.

Bitte geben Sie den Listeneinkaufspreis an: 1000

Bitte geben Sie die Bezugskosten an: 20

Einkaufskalkulation

=====

Listeneinkaufspreis 1000.00

- Lieferrabatt 100.00

=====

Zieleinkaufspreis 900.00

- Skonto 18.00

=====

Bareinkaufspreis 882.00

+ Bezugskosten 20.00

=====

Bezugspreis 902.00

Schon fertig? Dann erweitern Sie die Kalkulation bitte noch um eine Eingabe für Skonto und Rabatt sowie um die Umsatzsteuer.

Bitte geben Sie den Listeneinkaufspreis an: 10000

Bitte geben Sie die Bezugskosten an: 250

Bitte geben Sie den Rabatt in Prozent an: 20

Bitte geben Sie den Skonto in Prozent an: 1

Einkaufskalkulation

=====

Listeneinkaufspreis 10000.00

- Lieferrabatt 2000.00

Zieleinkaufspreis 8000.00

- Skonto 80.00

Bareinkaufspreis 7920.00

+ Bezugskosten 250.00

Bezugspreis 8170.00

+ USt 1552.30

Bruttopreis 9722.30

4. Listen, Dictionary, Tupel



Zurück zum Tierheim.

Das Tierheim ist mittlerweile schon recht groß und der Überblick geht verloren.

Eine Liste, die alle Hunde enthält, wäre sicherlich hilfreich.

Um dem Tierheim helfen zu können, müssen wir uns mit dem Thema Listen beschäftigen. Was Sie in Python nun unter der Bezeichnung Liste kennenlernen, würde in den meisten anderen Programmiersprachen als array bezeichnet werden. Eine Liste ist wohl der wichtigste Datentyp zur Speicherung einer variablen Anzahl an Elementen. Die Ordnung der Elemente bleibt erhalten, es können jederzeit und an jeder beliebigen Stelle Elemente hinzugefügt oder gelöscht werden.

4.1. Eindimensionale Listen



Folgende Hunde befinden aktuell im Tierheim:



Sparky, Bello, Stella, Bruno, Lilly, Dory

Setzen Sie die untenstehenden Aufgaben mittels Python um und notieren Sie die Befehle, die Sie hierfür verwendet haben.

	Aufgabe	Befehl:
Liste erstellen	Erstellen Sie eine Liste mit allen Hunden.	<code>dogs = [] / dogs = ["sparky", "Belo", ...]</code>
Liste ausgeben	Lassen Sie sich Ihre Liste ausgeben	<code>print(dogs)</code>

Index von Listen.

Eine Liste kann man sich als eine Beziehung zwischen Indizes und Elementen vorstellen. Diese Beziehung nennt man Mapping. Jedem Index ist eines der Elemente zugeordnet.



Visualisieren Sie den Zusammenhang zwischen den Elementen einer Liste und einem Index.

Listenindex
 dogs → 0 = Sparky
 1 = Bell
 2 = Stella



Notieren Sie die Besonderheit des Indexes.

Listenindex beginnt immer bei 0!

Weitere Möglichkeiten, mit Listen zu arbeiten

Ebenfalls hilfreich kann es sein herauszufinden, ob wir ein bestimmtes Element in unserer Liste haben oder wie viele Hunde sich gerade im Tierheim befinden.
Bearbeiten Sie hierfür die untenstehenden Aufgaben.

	Aufgabe	Befehl:
Ein bestimmtes Listenelement abfragen.	Wie lautet der 2. Hund auf der Liste?	<code>print(dogs[1]) => Bello</code>
Liste ausgeben	Lassen Sie sich Ihre Liste ausgeben	<code>print(dogs)</code>
True or false?	Prüfen Sie, ob der Hund Caesar im Tierheim ist?	<code>print("caesar" in dogs) → False</code>
True or False	Prüfen Sie, ob Bruno im Tierheim ist?	<code>print("Bruno" in dogs) → True</code>
Listenlänge	Lassen Sie sich die Listenlänge ausgeben.	<code>print(len(dogs))</code> ↓ length

4.1.1. Bearbeiten von Listen

Eindimensionale Listen sind nicht festgeschrieben. Das bedeutet, wir können Werte hinzufügen, löschen usw. Um ein Gefühl dafür zu bekommen, bearbeiten Sie bitte die folgenden Aufgaben.

	Aufgabe	Befehl:
Elemente überschreiben	Der dritte Name auf der Liste heißt nun Milo.	<code>dogs[2] = "Milo"</code>
Liste sortieren	Sortieren Sie die Liste und lassen Sie sich diese ausgeben.	<code>dogs.sort()</code>
Element hinzufügen	Der Hund Peppa ist neu ins Tierheim dazu gekommen. Ergänzen Sie die Liste.	<code>dogs.append("Peppa")</code> <code>dogs.insert(7, "Peppa")</code>
Elemente löschen und in andere Variable speichern	Der Hund Peppa wurde vermittelt. Löschen Sie diesen und nehmen Sie ihn in eine neue Variable „vermittelterHund“ auf.	<code>vermittelterHund.append(dogs.pop(6))</code> Beim Löschen des letzten Elements kann der Index weggelassen werden
Element löschen	Löschen Sie Bello aus der Liste	<code>dogs.remove("Bello")</code> <code>del dogs[0]</code>

4.1.2. Weitere Methoden auf Listen

Sortieren von Listen / aufsteigend & absteigend.

Benötige man die Listeneinträge sortiert, kann die über die Funktion `sorted()` durchgeführt werden. Um Listen bspw. Absteigen zu sortieren hilft uns die Method: `reverse = True`

Die Anweisung könnte folgendermaßen aussehen:

```
dogs.sort(reverse=True)      True → Absteigend  
                           False → Aufsteigend
```

Werte an bestimmte Stelle einfügen

Möchten wir Werte an eine bestimmte, hilft uns der Befehl `insert()`. Hierfür wird in der Klammer der Index angegeben, an der das Element eingefügt werden soll. Anschließend folgt noch das Listenelement.

```
dogs.insert(2, "Noxi")
```



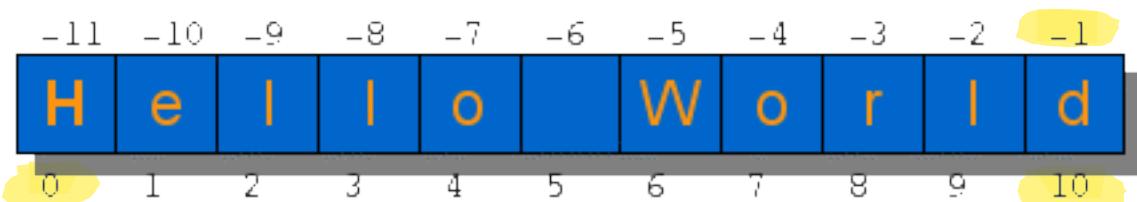
Good to know:

Du möchtest dir alle Methoden zu Listen anzeigen lassen? Ganz einfach. Nutze den Befehl: `print(dir(list))`

4.1.3. Slicen

In vielen Programmiersprachen kann es ziemlich schwierig sein, einen Teil eines Strings zu schneiden, und noch schwieriger. Python macht es mit seinem Slice-Operator sehr einfach. Slicing wird häufig in anderen Sprachen als Funktion mit möglichen Namen wie "Teilstring", "gstr" oder "substr" implementiert.

Jedes Mal, wenn Sie einen Teil eines Strings oder einer Liste in Python extrahieren möchten, sollten Sie den Slice-Operator verwenden. Die Syntax ist einfach. Eigentlich sieht es ein bisschen so aus, als würde man auf ein einzelnes Element mit einem Index zugreifen, aber statt nur einer Zahl haben wir mehr, getrennt durch einen Doppelpunkt ":". Wir haben einen Start- und einen Endindex, einer oder beide fehlen möglicherweise. Es ist am besten, die Funktionsweise von Slice anhand von Beispielen zu untersuchen:



Aufgabe:

1. Erstellen Sie eine String-Variable mit dem Wert „Python ist großartig“
2. Versuchen Sie unterschiedliche Arten des slicing und notieren Sie diese hier.
Verwenden Sie für die Übung den String: `slogan = „Python ist großartig“`

Die ersten fünf Buchstaben	<code>firstFive = slogan[0:5]</code> ↗ nicht mehr im Bereich
Ab dem fünften Buchstaben starten	<code>slogan[4:]</code>
Nur die ersten fünf Buchstaben	<code>slogan[:5]</code>
Die letzten beiden abschneiden	<code>slogan[:-2]</code>

1. Wenden Sie dies nun für die folgende Liste an: `cities = ["Wien", "London", "Paris", "Berlin", "Zürich", "Hamburg"]`
 - a. Lassen Sie sich nun die ersten drei Städte ausgeben.
 - b. Alles außer der die letzten beiden Städte.

- c. Jetzt extrahieren wir alle Städte außer den letzten beiden, "Zürich" und "Hamburg":

4.1.4. Übungen zu Listen



Bearbeiten Sie die Übungen zu Listen.

1. Best Friends

- Erstellen Sie eine Liste, die Ihre drei besten Klassenfreunde speichert und diese anschließend wieder ausgibt.
- Ein Freund ist in Ungnade gefallen, löschen Sie diesen aus der Liste. Geben Sie den Inhalt der Liste anschließend wieder aus.
- Sie haben eine neue Freundin kennen gelernt, sie heißt Uschi. Speichern Sie auch diese in Ihrer Liste. Geben Sie den Inhalt der Liste anschließend wieder aus.

2. Stellen Sie sich vor, Sie seien der Python-Interpreter und müssten den Code abarbeiten, um herauszufinden wie die Ausgabe lautet.

```
eighties = [", 'duran duran', 'B-52s', 'muse']
newwave = ['flock of seagulls', 'postal service']
remember = eighties[1]
eighties[1] = 'culture club'#786
band = newwave[0]
eighties[3] = band
eighties[0] = eighties[2]
eighties[2] = remember
print(eighties)
```

3. Sie wollen herausfinden, wie viele Kilometer Sie im Durchschnitt mit einer Tankfüllung fahren können. Hierzu haben Sie sich folgende Werte notiert: (1020, 923,780, 890).

- a) Berechnen Sie die durchschnittliche Kilometerreichweite für Ihr Fahrzeug.
- b) Ermitteln Sie den durchschnittlichen Kraftstoffverbrauch je 100 km unter der Annahme, dass Sie einen 70 Liter Tank haben.

4. Erstellen Sie ein Array mit den Zahlen von 0 bis 9. Erstellen Sie anschließend ein zweites Array und weisen Sie diesen den Inhalt des ersten Arrays in umgekehrter Reihenfolge zu. Folgendes Beispiel soll dies simulieren. [0,1,2,3,4,5,6,7,8,9]→[9,8,7,6,5,4,3,2,1,0]

5. Komplimente-Generator:

Um einen Komplimente Generator zu erstellen, müssen sie sich zunächst das random Modul importieren (import random).

Jetzt möchten wir uns einen Komplimente-Generator entwickeln. Dieses Programm soll uns loben, was das Zeug hält. Wer gerne schimpft, darf die Sache auch gerne umdrehen 😊. Erstellen Sie nun zwei Listen. Eine Liste enthält Nomen, die zweite die Adjektive. Ihr dürft eure Liste beliebig erweitern.

Liste 1: beste, liebenswürdigste, schönste, coolste

Liste 2: Mensch, Hecht, Kumpel, Freund

6. Lottozahlen:

Erstellen Sie ein Programm, das die Ziehung der Lottozahlen simuliert. Füllen Sie zunächst ein Array mit sechs Zahlen, die zufällig über folgenden Befehl erzeugt werden. Der Spieler kann anschließend sechs Zahlen zwischen 1 und 49 eingeben, die in einem weiteren Array gespeichert werden. Das Programm zeigt anschließend beide Zahlenreihen untereinander an. Der Abgleich der richtig erratenen Zahlen wird zunächst dem Nutzer überlassen.

Erweiterung für Schnelle: Sortieren Sie die Ausgabe.

7. Erklären Sie den Unterschied zwischen der sample() und der choice() Methode. Erläutern Sie den Unterschied anhand der Lottozahlen.

Choice dopplet sample nicht

8. Berechnen Sie aus den untenstehenden Notenwerten folgende Informationen:

1. Durchschnitt
2. Beste/schlechteste Note
- 3.

Beispiel: 1,3,4,3,2,1,2,3,4,5,2,1,2,3,3,2,1,4,4

4.2. Dictionary

Der Datentyp Dictionary (auf Deutsch „Wörterbuch“ bzw. assoziative Liste) ermöglicht ein Aufbau ähnlich einer Liste aber als Indizes anstelle von Zahlen dann „Text“. Der Begriff Dictionary ist wirklich mit dem dafür deutschen Wort „Wörterbuch“ am besten verständlich. Über den **Datentyp Dictionary** können wir also in Python **eine Zuordnungstabelle** (in anderen Sprachen „assoziatives Array“) aufbauen und nutzen.

4.2.1. Erstellen eines Dictionaries:

Schauen wir uns den Aufbau anhand einer deutsch-englisch Wörterbüches an. Wir wollen Zahlen in beiden Sprachen erhalten:

Key
null = zero *Value*

eins = one

zwei = two

drei = three

Ablauf zum Erstellen eines Dictionaries:

1. Als Dictionary wird in Python im ersten Schritt ein **leeres „Wörterbuch“** erstellt – dies ist an den **geschweiften Klammern zu sehen**. Wir nennen es „germanEnglish“.
2. Werte füllen: entweder Paar für Paar → **germanEnglish ['eins'] = 'one'** oder alles zusammen → **germanEnglish = {'eins': 'one', ...}**

Key *Value*

→ **Beachte:** beim Erstellen eines Dictionaries gibt es immer einen „Key“ und einen Value.



Entscheiden Sie, welche Werte die Keys und welche Werte die Values sind.

```
germanEnglish['null'] = 'zero'  
germanEnglish['eins'] = 'one'
```

Key *Value*

4.2.2. Bearbeiten von Dictionaries:



Erstelle dir ein Dictionary und teste die unterschiedlichen Methoden.

Dictionary löschen	<code>del dictionaryName[index]</code> Löscht das Dictionary oder den angegebenen Index
Key ausgeben lassen	<code>Dictionary.keys()</code> → alle Keys
Value ausgeben lassen	<code>Dictionary.values()</code> → alle Values
Alle Werte ausgeben ↳ key & value	<code>Dictionary.items()</code>
Inhalte aus Dictionary löschen	<code>Dictionary.clear()</code> → Löscht nur Inhalte
Dictionary kopieren	<code>DictionaryTwo = dict(Dictionary)</code> <code>Dictionary.copy()</code>



Good to know:

Du möchtest weitere Methoden testen? `Dir(dict)` gibt die eine Liste aller Methoden für Dictionaries

4.3. Tupel

Wir haben bereits Variablen und Listen kennengelernt. Man fragt sich nun natürlich nach dem Unterschied zwischen Variablen und Listen im Vergleich zu Tupeln in Python.

Unterschied Variablen und Listen:

- Variable speichert einen Inhalt
- Liste speichert VIELE Inhalte (bzw. kann das)

Und was ist ein Tupel? Eigentlich eine Liste aber der wichtige Unterschied ist, dass der Inhalt eines Tupels NICHT änderbar ist. Daher sind Tupel besonders geeignet, um Konstanten zu verkörpern.

Um unsere Auflistung von oben zu komplementieren:

- **Variable** speichert **einen Inhalt**
- **Liste** speichert **VIELE Inhalte** (bzw. kann das)
- **Tupel** speichern **VIELE Inhalte UNVERÄNDERLICH**

4.3.1. Erstellen eines Tupels

Ein Tupel lässt sich ähnlich wie eine Liste erstellen. Anstelle der eckigen Klammern werden hier nun runde Klammern verwenden.

Beispiel:

```
tupelErstellung = ('Wert1', 'Wert2', 'Wert3', ...)
```

Zugriff auf Tupel: Auf einen Wert eines Tupel wird mit Hilfe der Print-Anweisung und des Index zugegriffen.

```
print(tupelErstellung[0])
```

5. Kontrollstrukturen

Kontrollstrukturen sind ein zentraler Bestandteil fast jeder Programmiersprache. Erst diese ermöglichen einen flexiblen, unterschiedlichen Fluss des Programmablaufes in Abhängigkeit der Auswertung von Variablen. Im Jahr 1972 kam Isaac Nassi und Ben Shneiderman auf die Idee, Sachverhalte der strukturierten Programmierung in Form von Diagrammen, genannt Struktogramm abzubilden. Ziel war es, weniger sogenannten Spagetticode zu produzieren. Daher und auch weil die IHK das Thema Struktogramme immer noch spannend bei der Prüfung findet, machen wir einen kleinen Exkurs zum Thema Struktogramme.

5.1. Struktogramme

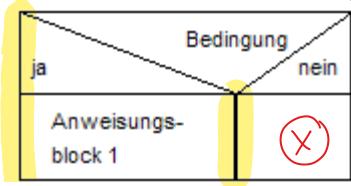
Linearer Ablauf (Sequenz)



Jede Anweisung wird in einen rechteckigen Strukturblock geschrieben. Die Strukturböcke werden nacheinander von oben nach unten durchlaufen. Leere Strukturböcke sind nur in Verzweigungen zulässig.

Verzweigung (Alternativen)

Einfache Auswahl



If-Anweisung

Alternativ: bedingte Verarbeitung, Selektion, einfache Selektion (if)

Nur wenn die Bedingung zutreffend (wahr) ist, wird der Anweisungsblock 1 durchlaufen. Ein Anweisungsblock kann aus einer oder mehreren Anweisungen bestehen. Trifft die Bedingung nicht zu (falsch), wird der Durchlauf ohne eine weitere Anweisung fortgeführt (Austritt unten).

Zweifache Auswahl

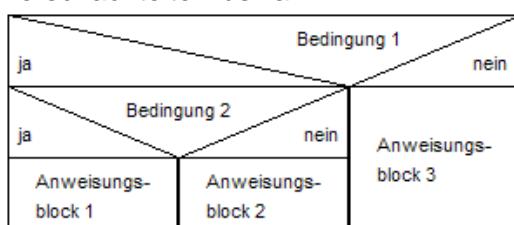


If-Anweisung

Alternativ: alternative Verarbeitung, alternative Verzweigung (if then else)

Wenn die Bedingung zutreffend (wahr) ist, wird der Anweisungsblock 1 durchlaufen. Trifft die Bedingung nicht zu (falsch), wird der Anweisungsblock 2 durchlaufen. Ein Anweisungsblock kann aus einer oder mehreren Anweisungen bestehen. Austritt unten nach Abarbeitung des jeweiligen Anweisungsblocks.

Verschachtelte Auswahl



Es folgt eine weitere Bedingung. Die Verschachtelung ist ebenso im Nein-Fall (noch) möglich.

Kopf- und zählergesteuerte Schleifen

While-/For-Schleife



Es wird eine While-Schleife und For-Schleife dargestellt. Die Zählervariable wird wie eine Art „L“ dargestellt und so lange geführt bis die Schleife zu Ende ist.

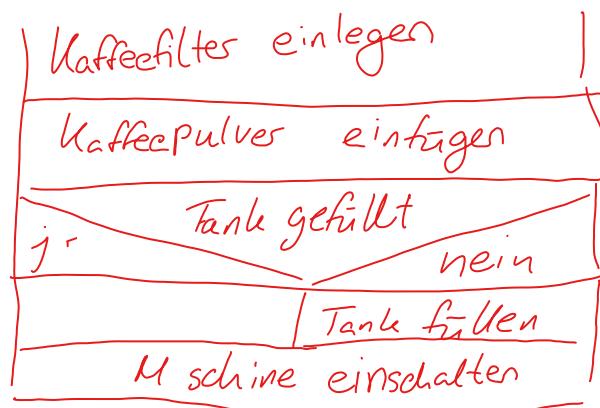
5.1.1. Übungen Struktogramme

Übungsaufgaben Kontrollstrukturen

Aufgabe 1

Beim Kaffeekochen in einer herkömmlichen Maschine fallen folgende Tätigkeiten an:

- Kaffeefilter einlegen
- Kaffeepulver einfüllen
- Prüfen, ob noch Wasser im Tank ist. Wenn nein, dann Tank auffüllen
- Maschine einschalten



Aufgabe 2

Ein Betrieb zahlt seinen Außendienstmitarbeitern neben einem Grundgehalt (Fixum) eine umsatzabhängige Provision:

- bei einem Umsatz bis 100.000,- € 2%
- bei einem Umsatz bis 500.000,- € 3%
- bei einem Umsatz von mehr als 500.000,- € 5%.

Ein Programm ermittelt den Betrag, der einem Außendienstmitarbeiter in diesem Monat überwiesen werden muss.

Aufgabe 3:

In der Bundesrepublik ist das Kindergeld folgendermaßen gestaffelt:

- Für das erste und das zweite Kind gibt es jeweils 204,- €.
- Für das dritte Kind gibt es 210,- €.
- Für jedes weitere Kind gibt es 235,- €.

Das Programm soll, abhängig von der eingegebenen Kinderzahl einer Familie, das Kindergeld berechnen. (Stand 08/2020)

Aufgabe 4:

Entwickeln Sie ein Struktogramm, das den BMI (Body-Maß-Index) des Benutzers berechnet und „einordnet“. Der BMI ist ein Maß für das Gewicht in Relation zur Körpergröße, er wird wie folgt berechnet (Gewicht in kg und Körpergröße in cm): $bmi = \text{gewicht} * 10000 / \text{groesse}^2$

BMI-Index	Frauen	Männer
Untergewicht	< 19	< 20
Normalgewicht	19 – 24	20 – 25
Übergewicht	25 – 30	26 – 30
Behandlungsbedürftiges Übergewicht	> 30	

Aufgabe 5: Schreibe ein Programm, welches von 1 – 100 zählt und die Zahlen ausgibt.

Aufgabe 6:

Weil die astronomische Dauer eines Jahres (wenn die Erde die Sonne einmal umrundet hat) etwas länger ist als 365 Tage, wurden Schaltjahre zum Ausgleich eingefügt.

Ein Schaltjahr ist ein Jahr,

- welches eine Jahreszahl hat, die durch 4 teilbar ist.
- Jahreszahlen, die durch 100 teilbar sind, sind allerdings keine Schaltjahre.
- Es sei denn, die Jahreszahl ist durch 400 teilbar.

Erstellen Sie ein Struktogramm für ein Programm, welches prüft, ob eine eingegebene Jahresziffer ein Schaltjahr ist oder nicht und anschließende eine entsprechende Antwort ausgibt.

Aufgabe 7:

1. Der Spieler hat 7-mal die Chance die vom Computer durch Zufall festgelegte Zahl zu erraten. Als Rückmeldung bekommt er nur:
 - deine geratene Zahl ist zu groß
 - deine geratene Zahl ist zu klein
 - Gewonnen! Die geheime Zahl ist nicht mehr geheim

Wenn der Spieler gewonnen hat, kommt eine entsprechende Meldung. Hat der Spieler die Zahl nicht erraten, kommt als Nachricht: „Schade – verloren. Einfach nochmals probieren“.

5.2. Kontrollstrukturen – allgemein

Wie wir im vorangehenden Kapitel gelernt haben, wird ein Programm zeilenweise von oben nach unten abgearbeitet. Nun gibt es Fälle, wo wir einen Programmblock nur unter einer Bedingung ausführen oder einige Anweisung mehrmals wiederholen wollen.

Es ist oft notwendig, einen ganzen Programmblock für jedes Element einer Liste zu wiederholen. Dies ist eine Form der Iteration. Wir sprechen von einer *Iteration über den Elementen einer Liste*. In Python wird dies mit einer `for` Schleife gemacht. Wir werden in diesem Zusammenhang auch lernen, wie in Python ein Programmblöck gekennzeichnet wird.

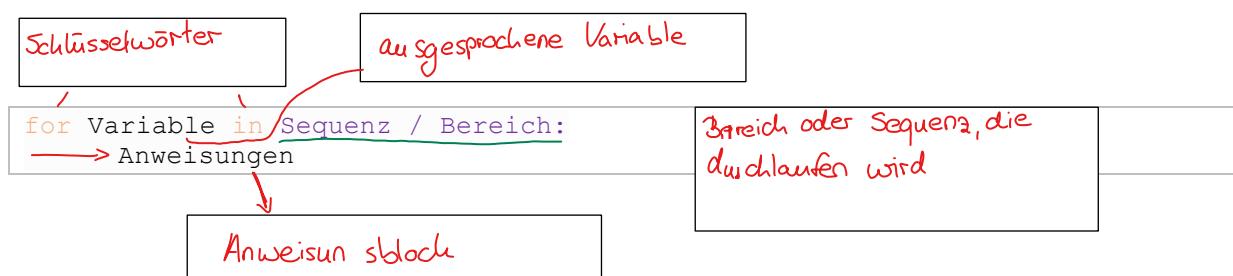
In anderen Fällen ist es notwendig, zu entscheiden, ob ein Programmblöck ausgeführt werden soll oder nicht. Hier handelt es sich um eine Selektion. Dafür müssen wir zuerst lernen, wie in Python Bedingungen formuliert werden. Anschliessend betrachten wir die `if` Verzweigung, welche uns das bedingte Ausführen eines Programmblöcks ermöglicht.

Teilweise reicht diese sehr spezielle Form der Wiederholung nicht. Es kann auch notwendig sein, einen Block zu wiederholen, bis eine Bedingung nicht mehr wahr ist. Dies nennen wir eine *bedingte Iteration*.

5.2.1. Die For-Schleife

Wie schon erwähnt, ist die `for` Schleife ein Spezialfall der Iteration, nämlich eine Iteration über einer Liste. Man kann alle Probleme, welche mit `for` gelöst werden können, auch mit der später besprochenen, allgemeineren `while` Schleife lösen. Das Iterieren über einer Liste ist aber eine so häufige Form der Iteration, dass `for` häufiger vorkommt als `while`.

Aufbau einer for-Schleife



Runden, Sequenzen und Bereiche

Mit Hilfe einer For-Schleife und den Bereich, den du angibst, sagst du Python, wie viele Runden die Schleife drehen soll. Hierbei hilft dir der Befehl `range()`.

Ein Bereich ist dabei eine **Zahlenfolge wie von 0 – 5**. Das heißt, die Schleife befolgt die Anweisung 5-mal.

Eine Sequenz ist ähnlich zum Bereich. Allerdings sprechen wir hier nicht von Zahlen, sondern **Listen**.



Übersetze die Sätze in Python Code oder in Maschinensprache.

Wie Python es versteht	Wie Menschen es verstehen
<code>for participant in namelist: sendInvitation</code>	Für alle Teilnehmer, die auf der Namensliste stehen, soll eine Einladung verschickt werden.
<code>For balloon in balloonsWithoutGasInBox: fillBalloon</code>	Füll jeden Ballon in der Box, der nicht mit Gas gefüllt ist, mit Gas.
<code>for song in songlist: playSong</code>	Für jedes Lied in der Musikliste, spiele das Lied ab.



Übungsaufgaben zu Schleifen

1. Teste die range() Funktion mit einer For-Schleife.
 - a) Es sollen die Zahlen 1-5 ausgegeben werden.
 - b) Es sollen die Zahlen 3-5 ausgegeben werden
 - c) Es sollen die Zahlen 0-100 ausgegeben werden, aber in 2er Schritten. (alle geraden Zahlen).

2. Gegeben ist der folgende Anfang eines Programms:

```
animals = ["tiger", "mouse", "bird", "python", "elephant", "monkey"]  
# ...
```

Ergänze das Programm so, dass für jedes Tier aus der Liste `animals` der Satz „... ist ein Tier“ in der Konsole ausgegeben wird. Benutze dafür die `print()` Funktion.

3. Mit `append()` kannst du einer Liste ein Element hinten anfügen. Schreibe ein Programm, welches den Benutzer mit `input()` fünf Mal nach einem Wort fragt und alle diese Worte in einer Liste abspeichert. Anschliessend werden alle Worte aus der Liste mit `print()` zurückgegeben.

Schon fertig? Frage den Benutzer zuerst, wie viele Worte er eingeben will und frage dann genau so viel mal nach einem Wort.

5.2.2. Die If-Anweisung

Wie bereits erklärt, kommt es vor, dass wir einen Programmblöck nur unter einer Bedingung ausführen wollen. Falls die Bedingung nicht erfüllt ist, führen wir entweder nichts oder einen anderen Block aus.

Beispiel:

Bedingungen abfragen

```
wert = 3
If wert < 5:
    print(„Wert ist kleiner als 5“)
else:
    print(„Wert ist größer als 5“)
```

Anweisung, wenn Bedingung nicht zutrifft

Mit dem Wort **if** wird eine Bedingung abgefragt. In diesem Fall ist dies die Frage, ob die als Antwort gespeicherte Zeichenkette in der Liste enthalten ist oder nicht. Das „enthalten sein“ wird mit dem Wort „in“ geprüft.

Merke:

- Jede eingerückte Zeile unter dem If-Teil gehört zur If-Anweisung
 - Nach jedem if und der Bedingung benötigen wir einen Doppelpunkt
 - Nach jedem else benötigen wir einen Doppelpunkt

Mehrere Bedingungen: **elif**

Das bisherige Programm ist nicht aussagekräftig. Es gibt die Aussage „Der Wert ist größer als 5“ oder „Wert ist kleiner als 5“. Schicker wären die folgenden drei Ergebnisse:

- Wert ist **kleiner als 5**
- Wert ist **exakt 5**
- Wert ist **größer als 5**



Kombinieren Sie die elif Bedingung mit der If-Anweisung

```
if wert<5:  
    print("Der Wert ist kleiner als 5")  
elif wert>5:  
    print("Der Wert ist größer als 5")  
else:  
    print("Der Wert ist 5")
```



Es können beliebig viele elif verwendet werden.



Übungsaufgaben zu If-Anweisungen

1. Für Vergleiche, ob etwas grösser oder kleiner ist, können die in der Mathematik üblichen Zeichen `<` und `>` benutzt werden. Schreibe ein Programm, welches die folgenden Schritte ausführt:

- Es wird vom Benutzer per `input()` eine Zahl eingelesen.
- Es wird geprüft, ob die Zahl grösser oder kleiner als 10 ist.
- Mit dem Befehl `print()` wird entsprechend entweder „Die Zahl ist grösser als 10“ oder „Die Zahl ist kleiner als 10“ ausgegeben.

2. Schreibe ein Programm, welches vom Benutzer 10 Zahlen einliest und diese in einer Liste speichert. Anschliessend soll das Minimum und das Maximum der Zahlen aus der liste bestimmt und ausgegeben werden.

3. Ob ein Jahr im Gregorianischen Kalender ein Schaltjahr ist, kann nach den folgenden Regeln entschieden werden:

1. Jahre sind Schaltjahre, falls ihre Jahrzahl durch 400 teilbar ist.
2. Jahre sind Schaltjahre, falls ihre Jahrzahl durch 4 teilbar ist, ausser die Jahrzahl ist durch 100 teilbar.

Schreibe ein Programm, welches den Benutzer nach einer Jahreszahl fragt und anschliessend prüft, ob es sich um ein Schaltjahr handelt.

4. Professor Ungerechtmann der Kantonsschule Unfairdorf braucht ein Programm für die Notenvergabe der Abschlussprüfung. Die Abschlussnote hängt von den folgenden Parametern ab:

- Prüfungsnote (von 1 bis 6 mit Halbpunkten);
- Augenfarbe (z.B. dunkel=1, hell=0);
- Frisur (z.B. kurze Haare=1, lange Haare=0);
- Wetter (z.B. schön=1, nicht schön=0).
- Schüler bringt Sushi mit. (ja = 1, nein = 0)

Es gilt folgendes:

- Hat der Prüfling dunkle Augen und...
 - kurze Haare, so wird die Abschlussnote um 10% erhöht (d.h. Abschlussnote = Prüfungsnote + 10% Prüfungsnote).
 - lange Haare, so wird die Abschlussnote um 10% reduziert.
- Hat der Prüfling helle Augen und...
 - kurze Haare, so wird die Abschlussnote um 10% reduziert.
 - lange Haare, so wird die Abschlussnote um 10% erhöht.
- Ist das Wetter schön, so wird die Abschlussnote um eine Einheit verbessert.
- Bringt der Schüler Sushi mit, wird die Note um zwei Einheit verbessert.
- Die Abschlussnoten müssen auf halbe Noten gerundet werden.

Hinweis: Wie kann man auf halbe Noten runden? Die Funktion `round()` rundet auf ganze Noten, z.B. `round(5.4) = 5` aber `round(5.4*2) = 11 ... ;)`

Zusatzaufgabe: Erfinde und implementiere einige neue Bedingungen, von denen die Abschlussnote abhängt.

Schon fertig??

5. Mit der Blutalkoholkonzentration (BAK) beschreibt man den Anteil des Alkohols im Blut. Dieser Anteil wird üblicherweise in Promille (als Gewichtsanteil g/kg) angegeben. Die Blutalkoholkonzentration spielt u.a. eine große Rolle bei der Beurteilung der Fahrtüchtigkeit, wenn man Alkohol zu sich genommen hat. Ab 0,5 Promille Blutalkoholkonzentration darf man nach der derzeitigen Rechtslage kein Kraftfahrzeug führen. Aber auch schon bei einer geringeren Blutalkoholkonzentration ist die Konzentrations- und Reaktionsfähigkeit eingeschränkt.

Die Blutalkoholkonzentration hängt von zahlreichen Faktoren ab. Sicherlich spielt die Menge an Alkohol, die man getrunken hat, eine entscheidende Rolle. Zu berücksichtigen ist dabei, wieviel (reiner) Alkohol tatsächlich im Getränk vorliegt. Zu berücksichtigen ist auch, auf wieviel Blut sich der Alkohol im Körper verteilt. Letzteres hängt u.a. vom Gewicht der Person ab, die den Alkohol getrunken hat.

Einen groben Schätzwert für die Blutalkoholkonzentration liefert die Widmark-Formel:

$$\text{Blutalkoholkonzentration (in Promille)} = \frac{\text{Masse des aufgenommenen Alkohols (in g)}}{\text{Masse der Person (in kg)} \cdot \text{Reduktionsfaktor}}$$

Dabei berechnet man die Masse des aufgenommenen Alkohols mit der folgenden Formel:

$$\text{Masse-Alkohol (in g)} = 10 \cdot \text{Volumen des Getränks (in l)} \cdot \text{Alkoholvolumenanteil (in \%)} \cdot 0.8$$

Der Reduktionsfaktor beträgt bei jungen Frauen etwa 0.6, bei jungen Männern etwa 0.7.

3. Setzen Sie die Struktogramme in Python um.

5.2.3. Die While-Schleife

Es gibt Fälle, wo eine `for` Schleife nicht ausreicht, weil wir keine Liste von Elementen haben und auch nicht zu Beginn wissen, wie oft etwas ausgeführt werden soll.

Die `while` Schleife wird solange ausgeführt wie eine Bedingung wahr ist. Sie hat die folgende Struktur:

while Bedingung:

Befehl 1

Befehl 2

Befehl 3

Dabei werden die Befehle 1 und 2 solange ausgeführt, wie die Bedingung wahr (also `True`) ist. Sobald die Bedingung falsche (`False`) wird, springt Python zur nächsten nicht eingerückten Zeile, also Befehl 3. Zur Formulierung der Bedingung können dieselben Formen wie bei der `if` Abfrage benutzt werden.

Man könnte sich zum Beispiel vorstellen, dass wir in einem Programm dem Benutzer eine Frage gestellt haben, welche mit „Ja“ oder „Nein“ zu beantworten ist. Wenn der Benutzer nicht entweder `J` für Ja oder `N` für Nein eingibt, ist die Eingabe ungültig und wir müssen nochmals nachfragen. Dies würde in Python wie folgt umgesetzt:

```
antwort = input('Beantworte die Frage mit Ja oder Nein (J/N): ')  
  
while antwort not in ['j', 'J', 'n', 'N']:  
    print('Eingabe ungültig!')  
    antwort = input('Beantworte die Frage mit Ja oder Nein (J/N): ')  
  
print('Vielen Dank!')
```

Der markierte Codeblock wird so lange wiederholt, wie die Bedingung `antwort not in ['j', 'J', 'n', 'N']` wahr ist. Das heisst, so lange die Antwort nicht in der Liste mit gültigen Antworten enthalten ist.

5.2.4. Übungen zu While-Schleifen

1. Schreibe ein Programm, welches von 1 – 100 zählt und die Zahlen ausgibt.
2. Schreibe ein Programm, welches Ihnen nach Eingabe eines Anlagenbetrages und eines Zinssatzes ausgibt, wie lange (in Jahren) Sie das Geld anlegen müssen, um über eine Millionen Euro zu verfügen.
Kontrolle:
Anlagebetrag: 700.000 *Zinssatz: 10%*
3. Erstellen Sie ein Programm, dass die lineare Abschreibung berechnet. Der Nutzer soll den Anschaffungspreis und die Nutzungsdauer angeben.

Beispieleingabe::

20000 (Fr. Einkaufspreis), 5 (Jahre)

Beispielausgabe::

Jahr 1: 16000.00 Jahr 2: 12000.00 Jahr 3: 8000.00 Jahr 4: 4000.00

Jahr 5: 0.00

4. Es soll solange gewürfelt werden, bis Sie eine 6 würfeln. Achten Sie darauf, dass die aktuell geworfene Zahl ausgegeben wird. Geben Sie zudem die Anzahl der Versuche ein.

5.2.5. Gemischte Übungen

2. Ein Würfel wird so lange geworfen, bis die Summe aller Würfe 100 oder mehr beträgt. Zum simulieren eines Würfels kannst du die Funktion `randrange()` aus dem Modul `random` benutzen.
 - a. Bestimme mit einem Programm, wie viele Würfe dazu nötig sind.
 - b. Schreibe ein Programm, welches mit Hilfe einer Liste zählt, wie oft welche Augenzahl geworfen wurde. Am Ende soll die Liste mit `print()` ausgegeben werden.
3. Der Spieler hat 7-mal die Chance die vom Computer durch Zufall festgelegte Zahl zu erraten. Als Rückmeldung bekommt er nur:
 - deine geratene Zahl ist zu groß
 - deine geratene Zahl ist zu klein
 - Gewonnen! Die geheime Zahl ist nicht mehr geheim

Wenn der Spieler gewonnen hat, kommt eine entsprechende Meldung. Hat der Spieler die Zahl nicht erraten, kommt als Nachricht: „Schade – verloren. Einfach nochmals probieren“.

4. Lotto die zweite
Erstellen Sie ein Lottozahlenspiel.
Der Nutzer soll 6 einzelne Zahlen nacheinander eingeben können. Bitte prüfen Sie, ob die Zahlen im Bereich von 1-49 liegen. Füllen Sie alle gültigen Zahlen in eine Liste.
Das Programm soll nun sechs Zufallszahlen ziehen und die beiden Listen miteinander vergleichen. Als Ausgabe erhalten Sie die Meldung, ob Sie gewonnen haben oder nicht.
5. Schreibe ein Programm, welches alle Primzahlen zwischen 1 und einer vom Benutzer gewählten oberen Grenze ausgibt.

5.3. Zusammenfassung Kapitel 4 und Kapitel 5

Arbeitsauftrag: Zusammenfassung Kapitel 4



Erstellen Sie eine Zusammenfassung zu Kapitel 4.

- a) Beschreiben Sie, welche Inhalte Sie gelernt haben.
- b) Beschreiben Sie die Unterschiede der unterschiedlichen Listen.
- c) Gibt es Funktionen, die Sie für besonders wichtig erachten?
- d) Pain Points vorhanden? Notieren Sie sie.

Arbeitsauftrag: Zusammenfassung Kapitel 5



Erstellen Sie eine Zusammenfassung zu Kapitel 5.

- e) Beschreiben Sie, welche Inhalte Sie gelernt haben.
- f) Beschreiben Sie die Unterschiede der Kontrollstrukturen.
- g) Erläutern Sie die wichtigsten Punkte zu Struktogrammen.
- h) Pain Points vorhanden? Notieren Sie sie

6. Funktionen

Das Konzept einer Funktion ist eines der wichtigsten in der Mathematik. Eine übliche Verwendung von Funktionen in Computersprachen ist die Implementierung **mathematischer Funktionen**. Eine solche Funktion berechnet ein oder mehrere Ergebnisse, die vollständig durch die an sie übergebenen Parameter bestimmt werden.

Das ist Mathematik, aber wir sprechen über Programmierung und Python.

Was ist also eine **Funktion in der Programmierung?**

Im allgemeisten Sinne ist eine Funktion ein **Strukturierungselement** in Programmiersprachen, um eine Reihe von **Anweisungen zu gruppieren**, damit sie in einem Programm mehr als einmal verwendet werden können. Die einzige Möglichkeit, dies ohne Funktionen zu erreichen, besteht darin, Code durch Kopieren und Anpassen an verschiedene Kontexte wiederzuverwenden, was eine schlechte Idee wäre. Redundanter Code - in diesem Fall sich **wiederholender Code** - sollte vermieden werden! Die Verwendung von Funktionen verbessert normalerweise die Verständlichkeit und Qualität eines Programms. Dies senkt auch die Kosten für die Entwicklung und Wartung der Software.

Funktionen sind unter verschiedenen Namen in Programmiersprachen bekannt, z. als Unterprogramme, Routinen, Prozeduren, Methoden oder Unterprogramme.

Arbeitsauftrag:



Beschreiben Sie stichpunktartig den Vorteil / Nutzen von Funktionen in Python

- Kontrolle von Codeblöcken
- Wiederverwendbarkeit
- Übersichtlichkeit / verbessert die Lesbarkeit

Beispiel:

```
print("Das Programm startet")
print("Hallo Peter")
print("Schön dich zu sehen!")
print("Viel Spaß mit dem Programm!")
# hier stelle man sich irgendwelchen Code vor:
egal = "über dies nicht nachdenken"
ist nicht wichtig = "normalerweise schon"
print("Hallo Dora")
print("Schön dich zu sehen!")
print("Viel Spaß mit dem Programm!")
# irgendein Code:
wie auch immer = "mir auch "
x = "steht stellvertretend"
y = "stellvertretend für wichtigen Code"
print("Hallo Kevin")
print("Schön dich zu sehen!")
print("Viel Spaß mit dem Programm!")
```

Ausgabe:

Das Programm startet
Hallo Peter
Schön dich zu sehen!
Viel Spaß mit dem Programm!
Hallo Dora
Schön dich zu sehen!
Viel Spaß mit dem Programm!
Hallo Kevin
Schön dich zu sehen!
Viel Spaß mit dem Programm!

Vorteilhaft?!

Fällt hier etwas auf?

```

print("Das Programm startet")
print("Hallo Peter")
print("Schön dich zu sehen!")
print("Viel Spaß mit dem Programm!")

# hier stelle man sich irgendwelchen Code vor:
egal = "über dies nicht nachdenken"
ist_nicht_wichtig = "normalerweise schon"
print("Hallo Dora")
print("Schön dich zu sehen!")
print("Viel Spaß mit dem Programm!")

# irgendein Code:
wie_auch_immer = "mir auch "
x = "steht stellvertretend"
y = "stellvertretend für wichtigen Code"
print("Hallo Kevin")
print("Schön dich zu sehen!")
print("Viel Spaß mit dem Programm!")

```


Was passiert hier?

- der Code wird unnötigerweise immer wiederholt
- Lediglich der Name wird geändert

Verbesserung?
Name 1
Name 2
Name 3


```

print("Hallo ")
print("Schön dich zu sehen!")
print("Viel Spaß mit dem Programm!")

```



Überlegen Sie sich, wie der Code verbessert werden könnte.
Notieren Sie Vorschläge.

```
def Begrüße(vorname, nachname):  
    print('Hallo' + vorname + nachname)  
    print('Schön dich zu sehen!')  
    print('Viel Spaß mit dem Programm!')
```



Wir wollen die Personen aber nun mit Vorname und Nachname begrüßen.
Ideen?

Funktionen allgemein:



Beschreiben Sie den allgemeinen Aufbau einer Funktion anhand des Beispiels zu Beginn.

```
def  funktionName(variablen):  
    hier folgt code
```

Arbeitsauftrag:



Beantworten Sie die untenstehenden Fragen.

1. Definieren Sie eine Funktion, die dem Bereich der For-Schleife alle Werte übergibt.

def foo(anfang, ende, schrittweite):
 for x in range(anfang, ende, schrittweite):
 print(x)

foo(4, 20, 2)

2. Die Schrittweite des Ranges soll durch die Funktion auf den Standardwert = 1 gesetzt werden. Beschreiben Sie ein mögliches Vorgehen.

def foo(anfang, ende, schrittweite=1):
 for x in range(anfang, ende, schrittweite):
 print(x)

foo(4, 20) → Schritte standardmäßig auf 1
foo(4, 20, 2) → Standardwert wird überschrieben

3. Ihr Kollege wollte den Anfangswert des Ranges auf 1 setzen. Allerdings bekommt er eine Fehlermeldung. Erläutern Sie, warum hier ein Fehler auftritt.

```
def ausgabe(anfangswert=1, endwert, schrittweite=1):
    for x in range(anfangswert, endwert, schrittweite):
        print(x)
    print("Funktion ausgabe durchlaufen")

ausgabe(9)
```

Lösung:

- Bei Python müssen Variablen mit einem Standardwert am Ende stehen
- Problem: beim Aufruf von ausgabe(9) setzt Python den Anfangswert auf 9.
Somit wird für den Endwert kein Wert übergeben.

def foo(endwert, anfang=1, schritte=1):
 usw.

6.1. Funktionen mit variabler Parameterzahl

Bisher sind wir bei der Übergabe der Parameter immer davon ausgegangen, dass wir die Anzahl der Parameter kennen.

```
def ausgabe(wert1, wert2):
    print("Ausgabe von Text aus einer Funktion")
    print(wert1)
    print(wert2)

ausgabe(5,3)
```

Was passiert aber, wenn wir die Anzahl der Parameter nicht kennen? Wenn also eine variable Parameteranzahl vorliegt? Auch für diesen Fall hat und Python eine Lösung. Wir kennzeichnen unseren Parameternamen mit einem Stern am Anfang.



```
def ausgabe(*mehrereParameter):
    for einzelwert in mehrereParameter:
        print(einzelwert)

ausgabe('Hello','World','guten','Morgen')
```

6.1.1. Funktionenbezeichner *args & **kwargs

- **Beliebe Anzahl Argumente (*args):** Ist die Anzahl der Argumente unbekannt verwenden wir * als Argument.



Erstellen Sie eine Funktion, die es erlaubt eine beliebige Anzahl an Kindern zu übergeben.

```
def myFunc(*kids):
    print('The youngest child is' + kids[2])
myFunc('Paul', 'Noah', 'Lea', 'Marie')
```

- **Beliebe Anzahl Keyword Arguments (**kwargs):** Ist die Anzahl der Keywords unbekannt werden ** verwendet. In der Regel verwenden wir hier die Bezeichnung **kwargs



Erstellen Sie eine Funktion, die es erlaubt ein Dictionary mit einer beliebigen Anzahl Keyvalues zu übergeben.

```
def func2(**kwargs):
    for key, value in kwargs.items():
        print(...)
func2(vorname='Alex')
```

➤ Exkurs: Formatierung von Dictionaries.

Dictionaries werden in der Regel nicht sehr sauber ausgegeben. Daher haben wir die Möglichkeit mit der Funktion `.format()`, die Ausgabe zu formatieren.

```
def uebergeben3(**kwargs):
    for key, value in kwargs.items():
        print("The capital of {} is {}.".format(key, value))

uebergeben3(england = "london", deutschland = "berlin", oesterreich = "wien")
```

- Die `format()` Methode formatiert bestimmte Werte und fügt diese in einen Platzhalter ein.
- Der Platzhalter wird mit `{}` definiert.
- Es gibt drei Varianten des Platzhalters:
 - benannter Index: „My name is {fname}, i'm {age}“.`.format(fname = Anne, age = 17)`
 - nummerierter Index: „My name is {0}, i'm {1}“.`.format(fname = Anne, age = 17)`
 - leerer Platzhalter: „My name is { }, i'm { }“.`.format(fname = Anne, age = 17)`

6.2. Rückgabewert bei Funktionen

Funktionen sind praktisch um immer wieder verwendeten Code nutzen zu können. Bisher haben wir bei unseren Funktionen in Python immer fleißig Daten in die Funktion reingegeben. In diesem Kapitel lassen wir uns Ergebnisse aus einer Funktion herausgeben.

Mit den herausgegebenen Ergebnissen in Form von Variablen können wir dann im weiteren Programmcode nach Belieben weiteres anstellen.



Erstellen Sie eine Funktion, die einen Eingabewert übergeben bekommt und anschließend mit 2 multipliziert und das Ergebnis ausgibt.

6.2.1. Warum Variable über return übergeben.

Warum müssen wir überhaupt die Variable über die `return`-Funktion zurückgeben? Eigentlich geben wir nicht die Variable, sondern den Wert der Variable zurück.



Führen Sie den untenstehenden Code aus und versuchen Sie diesen nachzuvollziehen. Was fällt auf?

```
def bspfunktionfuerueckgabe(eingabewert):
    rueckgabewert = eingabewert * 2
    return rueckgabewert

ergebnisausfunktion = bspfunktionfuerueckgabe (5)
print(ergebnisausfunktion)
print(rueckgabewert)
```

6.2.2. Geltungsbereich von Variablen

Das Verständnis der Unterschiede zwischen globalen und lokalen Variablen ist extrem wichtig bei der Verwendung von Variablen innerhalb und außerhalb von Funktionen.

Bauen wir für das Verständnis ein kleines Python-Programm auf, dass nur für die Nutzung der Variablen da ist. Dazu haben wir eine Funktion und diese Funktion bekommt Funktionen integriert.



Durchdenken Sie den untenstehenden Code, ohne ihn auszuführen. Welche Ergebnisse würden Sie erhalten? Notieren Sie die Ergebnisse.

```
variablenWert = "außerhalb der Funktion"
print("Variablenwert vor Funktion:", variablenWert)

def bspfunktion():
    print("Variablenwert in Funktion 1:", variablenWert)
    variablenWert = "IN der Funktion"
    print("Variablenwert in Funktion 2:", variablenWert)

bspfunktion()
print("Variablenwert nach Funktion:", variablenWert)
```

○



Vergleichen Sie die beiden Codeausschnitte. Worin liegt der Unterschied?

```
variablenWert = "außerhalb der Funktion"
print("Variablenwert vor Funktion:", variablenWert)

def bspfunktion():
    variablenWert = "IN der Funktion"
    print("Variablenwert in Funktion:", variablenWert)

bspfunktion()
print("Variablenwert nach Funktion:", variablenWert)
```

```
variablenWert = "außerhalb der Funktion"
print("Variablenwert vor Funktion:", variablenWert)

def bspfunktion():
    print("Variablenwert in Funktion 1:", variablenWert)
    variablenWert = "IN der Funktion"
    print("Variablenwert in Funktion 2:", variablenWert)

    bspfunktion()
    print("Variablenwert nach Funktion:", variablenWert)
```

6.2.3. Globale Variablen



Führen Sie den untenstehenden Code aus und versuchen Sie diesen nachzuvollziehen. Was fällt auf?

```
variablenWert = "außerhalb der Funktion"
print("Variablenwert vor Funktion:", variablenWert)
def bspfunktion():
    global variablenWert
    variablenWert = "IN der Funktion"
    print("Variablenwert in Funktion:", variablenWert)

bspfunktion()
print("Variablenwert nach Funktion:", variablenWert)
```

- Sobald wir in der Funktion die Variable auf global setzen, wird der Variablenwert auch außerhalb der Funktion gesetzt.
- ohne global wird der Wert außerhalb der Funktion ausgegeben
- Achtung: globale Werte können nicht in einer Zeile definiert werden.

6.3. Rekursive Funktionen

Wie bereits erwähnt, darf eine Funktion eine andere aufrufen. Es ist sogar erlaubt, dass sich eine Funktion selbst wieder aufruft. Dies kann für einige Dinge nützlich sein, wie zum Beispiel hier:

```
def countdown(n):
    if n <=0:
        print("Bumm!")
    else:
        print(n)
        countdown(n-1)
```

Merke:

Eine Funktion, die sich selbst aufruft, nennt man rekursiv und den Vorgang nennt man Rekursion

Wenn n gleich 0 oder negativ ist, wird das Wort „Bumm“ ausgegeben. Ansonsten wird n ausgegeben, eine Funktion mit dem Namen countdown wird aufgerufen (**dieselbe Funktion!**) und n-1 wird als Argument übergeben.

Beispiel: countdown(2)

Da n größer ist als 0, gibt die Funktion 2 aus und ruft sich selbst auf....
Die Ausführung von countdown beginnt mit n=1. Da n größer ist als 0, gibt die Funktion 1 aus und ruft sich selbst auf....

Die Ausführung beginnt mit n=0. Da n nicht größer als 0 ist, gibt die Funktion „Bumm!“ aus und kehrt zurück.

Die Ausgabe sieht also so aus:

```
3
2
1
Bumm!
```

Rekursive Algorithmen eignen sich besonders gut, um hierarchische Datenstrukturen zu verarbeiten, also z.B. um alle Verzeichnisse der Festplatte oder Baumstrukturen zu durchsuchen.

Schauen wir uns doch einmal ein etwas kompliziertes Beispiel einer Rekursion an: Die Fibonacci - Zahlen:

Dabei ist diese Fibonacci-Folge simpel: Der Beginn ist bei null und eins, danach ist jede Zahl die Summe der beiden unmittelbar vorangegangenen Zahlen. Also: $0+1=1$; $1+1=2$; $1+2=3$; $2+3=5$; $3+5=8$ und so weiter.

Ganz allgemein können wir festhalten: **$\text{fibonacci}(n) = \text{fibonacci}(n-1) + \text{fibonacci}(n-2)$**



Implementieren Sie eine rekursive Funktion, die die Fibonacci Folge wiedergibt

Kontrolle:

n	f _n						
1	1	11	89	21	10 946	31	1 346 269
2	1	12	144	22	17 711	32	2 178 309
3	2	13	233	23	28 657	33	3 524 578
4	3	14	377	24	46 368	34	5 702 887
5	5	15	610	25	75 025	35	9 227 465
6	8	16	987	26	121 393	36	14 930 352
7	13	17	1 597	27	196 418	37	24 157 817
8	21	18	2 584	28	317 811	38	39 088 169
9	34	19	4 181	29	514 229	39	63 245 986
10	55	20	6 765	30	832 040	40	102 334 155
							50 12 586 269 025

6.4. Main Methode

Damit wir nicht immer jede einzelne Funktion einzeln aufrufen möchten, implementieren wir uns in Zukunft ein Main() Methode. In dieser starten wir alle Funktionen und können auf einem Blick einzelne Funktionen auskommentieren.



Erstellen Sie sich eine main() Funktion mit dem Name main.

def main():
 Funktionen hier
 einfügen

main()

6.5. Übungen zu Funktionen

1. Berechnung Tankkosten: Erstellen Sie ein Programmcode, der mittels einer Funktion die Tankkosten berechnet.
 - Folgende Angaben sind zu berücksichtigen:
 - Menge = float, input
 - Preis = float, input
 - Kosten = menge * preis
 - Ausgabe: kosten

a) Die Funktion soll nun unabhängig von der Ein- und Ausgabe Funktion sein. Dafür benötigen wir eine separate Berechnungsfunktion. Setzen Sie diese um.

2. Konfektionsgrößen

Die Konfektionsgrößen für Frauen und Männer können wir anhand des Brustumfangs und der Körpergröße nach folgender Formel berechnen:

$$\text{Konfektionsgröße} = \frac{\text{Brustumfang}}{2}$$

Für Frauen müssen wir zusätzlich noch folgende Besonderheiten berücksichtigen:

- Die Konfektionsgröße für Frauen wird um den Wert 6 verringert, um andere Größenangaben als bei Männern zu erreichen
- Die Normalgröße (32-44) gilt für Frauen, die zwischen 164 und 170 cm groß sind.
- Die Langgröße (64-88) gilt für Frauen, die größer als 170 cm sind. Sie ergibt sich aus der verdoppelten Normalgröße.
- Die Kurzgröße (16-22) gilt für Frauen, die kleiner als 164 cm sind. Sie ergibt sich aus der halben Normalgröße.

Geben Sie eine Funktion an, die die Konfektionsgröße für Frauen und Männer nach den oben definierten Kriterien berechnet.

3. Die Funktion f soll die Fakultät einer ganzen Zahl errechnen. Die Fakultät von n ist das Produkt aller Zahlen zwischen 1 und n. Demzufolge gibt das Programm für 5!(! steht für Fakultät) $1*2*3*4*5 = 120$ aus. Schreiben Sie das Programm und nutzen Sie hierfür Rekursion.

7. Objektorientierung

Die objektorientierte Programmierung (OOP) ist aus der modernen SW-Entwicklung nicht mehr wegzudenken und bietet einige Vorteile:

- Nähe zur natürlichen Umgebung:
Alle Eigenschaften ähnlicher Objekte können in einer Klasse beschrieben werden. Objekte, die im realen Leben verbunden sind, sollten auch im Programm verbunden sein.
- Wiederverwendbarkeit:
Erstmals erzeugte und überprüfte Implementierungen einer Klasse können in vielen anderen Anwendungen wieder eingesetzt werden und müssen nicht neu implementiert werden.
- Kontrollfunktion:
Während der Übersetzung und der Laufzeit erfolgt eine strengere Überprüfung.

OOP ganz einfach: Sprechen wir mal von Dingen (um die Materie greifbar zu machen)

- Dinge haben Eigenschaften wie z.B. Größe, Farbe und Alter
- Dinge können Aktionen machen/bzw. mit diesen Dingen gemacht werden wie z.B. knurren, schmusen und schlafen

Bisher hatten wir in Python entweder Daten (was wir bei unseren Dingen mit Eigenschaften beschrieben haben) oder wir haben irgendwelche Aufgaben erledigt mit Funktionen und Methoden (in die wir zeitweise verschiedene Daten reingekippt haben).

Was aber passiert, wenn wir Daten und Methoden miteinander verknüpfen? Dann haben wir schon objektorientierte Programmierung (OOP) bzw. den Kerngedanken begriffen.

Wir trennen uns von den unspezifischen Datenstrukturen wie Variablen, Listen und Tupeln und gehen hin zu Datenstrukturen, die ein Objekt (sprich ein Ding) beschreiben.

Schauen wir uns einmal ganz konkret (m)eine Katze an. Die ist schwarz, dick und frisst nur Milch, falls sie nicht schläft und heißt ChiChi. Überleg einmal, welche Eigenschaften von Katzen einem einfallen und was Katzen so machen.



Überlegen Sie sich Eigenschaften für die Klasse Katze und notieren Sie diese.

Wir bauen uns also ein allgemeines Bild von einer Katze – einen Bauplan. Wir spielen mit Python Gott und schaffen einen allgemeinen Katzen-Zusammenbau-Plan. Das ist unsere Katzen-Klasse.

Und nun können wir virtuelle Katzen in beliebiger Anzahl erschaffen – sprich ganz viele Objekte, die grundlegend Gleich nach dem Bauplan aufgebaut sind, aber sich in Ihren Eigenschaften (Farbe, Alter, Name) unterscheiden und in der Ausprägung der Methoden.

7.1. Klassen und Objekte



Füllen Sie zunächst in der untenstehenden Tabelle Informationen zu Objekten und Klassen.

Klasse	Objekt
allgemeiner Bauplan: Klasse Katze	Konkretes Tier: Objekt katzeChiChi
	
Eigenschaften:	Objekt:
Methoden:	Methoden:

--	--



Beschreiben Sie den Unterschied zwischen einem Objekt und einer Klasse.

➤ **Klassen:**

--

➤ **Objekte:**

--

Übung: Verbinden Sie mit einer Linie konkrete Objekte mit einer Klasse. Notieren Sie jeweils unter dem Bild, ob es sich um ein Objekt oder um eine Klasse handelt.

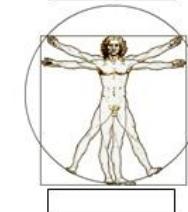
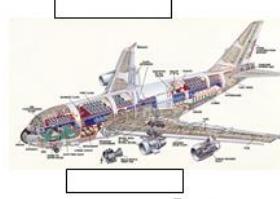


365 TAGE
URLAUB

365 TAGE
URLAUB

365 TAGE
URLAUB

365 TAGE
URLAUB



7.1.1. Klasse definieren und initialisieren

Nun starten wir mit Python und klassenorientierter Programmierung.

Wir definieren eine Klasse. Die Definition von Klassen muss vor dem Hauptprogramm im Code stehen.



1. Erstellen Sie sich nun die Klasse: class Cat
2. Versehen Sie die Klasse mit dem Schlüsselwort pass.

Hinweis: In der OOP werden Klassennamen mit relativ kurzen Namen versehen. Zum Festlegen einer Klassen verwenden wir das Schlüsselwort class und darauf folgt der Name der Klasse.

Schlüsselwort Pass:

Klasse Cat:

7.1.2. Konstruktoren einer Klasse

Ein Konstruktor ist eine Methode einer Klasse, die beim Erzeugen eines Objektes als erstes aufgerufen wird. Sie werden meist dafür verwendet, um die Eigenschaften der Klasse zu definieren und das Objekt zu initialisieren. Das bedeutet, den Eigenschaften Anfangswerte zuzuweisen oder späteren Speicherplatz zu reservieren.

Konstruktor



Aufbau des Konstruktors: Der Aufbau des Konstruktors erfolgt immer nach dem gleichen Schema.

Jetzt wollen wir unsere Eigenschaften einführen. Dazu wird ein neuer eingerückter Block erstellt, der immer den gleichen Aufruf hat:

`def __init__(self, ...):`

Merke:

-
-

Das erste Argument `self` bei `__init__()` ist eine Referenz auf das Objekt. Auf diese Weise ist z.B. die Zuordnung in der Klasse `Cat`

```
self.name = name
```

unmissverständlich. Das heißt `self.name` steht für die Instanzvariable des Objekts, welches erstellt wird und `name` steht für das Argument welches der Funktion `__init__()` übergeben wird. Natürlich kann man die Argumente auch anders benennen. Jedoch sollte klar ersichtlich sein, welches Argument zu welcher Instanzvariable gehört.

➤ **Standardwerte im init festlegen**

Beim Anlegen einer Klasse wäre es praktisch, auch bereits Standardwerte festlegen zu können. So könnte man sich ja vorstellen, dass man die Katzen immer „frisch geschlüpft“ bekommt. Somit hätten die als Alter einfach 0.

Genau das wollen wir zum Testen für das Alter als Standardwert hinterlegen.

Hier kommt wieder unsere `__init__()`-Methode zum Tragen. Wir können Eigenschaften mit Vorgabewert erweitern. Das kann man, muss aber nicht. Im Beispiel machen wir es nur für die Eigenschaft „alter“:

So könnte beim Anlegen der Instanz auf die Angabe des Alters verzichtet werden.



1. Sehen Sie sich nun nochmals die definierten Eigenschaften der Katze an.
2. Initialisieren Sie die Klasse Cat.

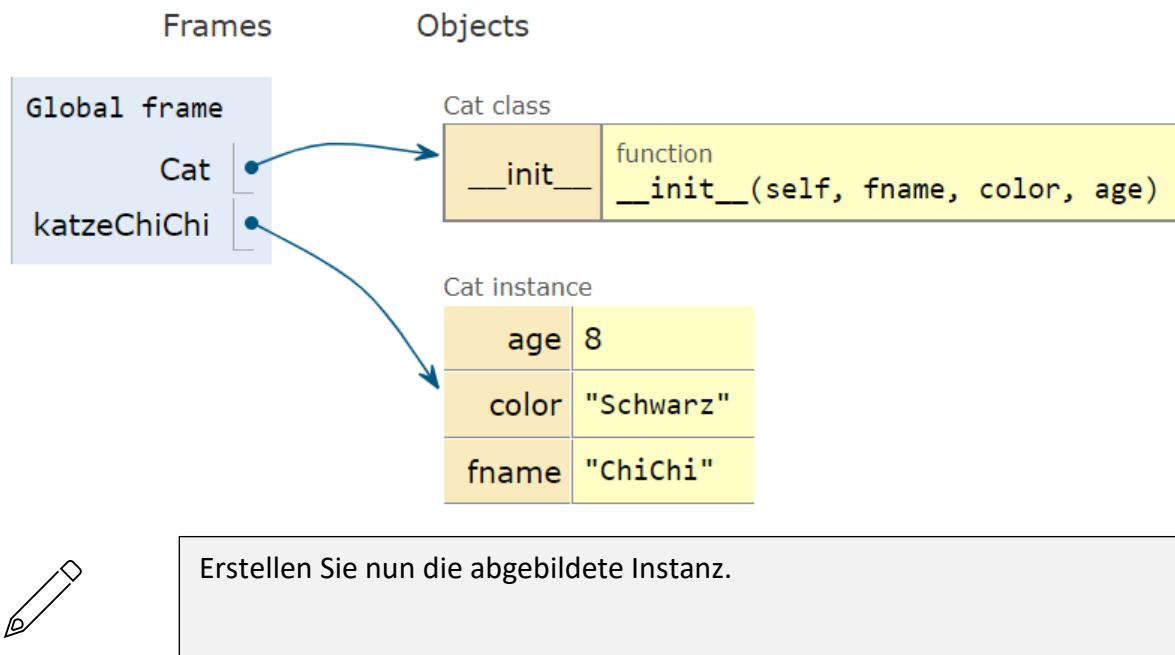
7.1.3. Objekte (Instanz) einer Klasse definieren

Wir haben nun die Klasse mit einem Konstruktor initialisiert. Nun können wir die ersten Objekte anlegen. Das Erstellen eines Objektes erfolgt immer im selben Schema:

Anlegen von Objekten

Bezeichner = Klasse(Parameterliste)

Die untenstehende Abbildung verdeutlicht nochmals den Unterschied zwischen einer Klasse und einem Objekt.



Schauen wir uns die Unterschiede mit dem Python Tutor an.

<https://pythontutor.com/live.html#mode=edit>

Übung: Erstellen Sie eine Klasse „Cars“ und initialisieren Sie diese. Bearbeiten Sie dafür untenstehende Aufgaben

- a) Welche Eigenschaften besitzen Autos? Notieren Sie diese
Farbe, Baujahr, KM-Stand, Marke, Sitzplätze
- b) Erzeugen Sie nun ein Objekt Ihrer Wahl
`Bmw = cars(„white“, 2020, 23000, „bmw“, 4)`
- c) Übergeben Sie nun Standardwerte:
KM: 2000
Sitze: 5

