

# Programmierung mit Python für Einsteiger

## Kapitel 2 - Zuweisungen und Fallunterscheidungen

Autor: Dr. Christian Heckler

# Eigenschaften von Python

- Interpretierte Skriptsprache
- Dynamisch getypt
- Verschiedene Programmierparadigmen (prozedural, funktional, OO, ...)
- Für alle gängigen Plattformen verfügbar (Windows, Unix, Mac)
- Insbesondere für Anfänger geeignet
- Umfangreiche Standardbibliothek
- Entwickelt von Guido van Rossum am CWI Amsterdam (Centrum Wiskunde & Informatica ) ab 1989
- Weite Verbreitung (s. diverse Programmiersprachenranglisten)

# Versionen von Python

- Es gibt verschiedene Versionen von Python. Aktuell in Gebrauch:
  - Python 2.x
  - Python 3.x
- Python 2 und Python 3 sind nicht (hundertprozentig) kompatibel
- Im Kurs wird Python 3 verwendet
- Der Interpreter für python 3 kann „python3“ heißen (z.B. unter Unix / Linux).
- Die Version erhält man über den Aufruf

```
python -V bzw. python3 -V
```

# Python-Interpreter

- Der Python-Interpreter kann ein Python-Programm ausführen, das in einer Textdatei (mit der Endung .py) abgespeichert ist. (Wie im Beispiel gesehen.)
- Es gibt aber auch einen interaktiven Modus, in dem Python-Anweisungen ausgeführt werden können, ohne ein Programm zu schreiben.

# Ausführung eines Python-Programms

- Es gibt mehrere Möglichkeiten zum Aufruf eines Python-Programms:
- Kommandozeile (cmd): `python progr_bsp.py`
- Windows:
  - Einer Datei kann über die Datei-Endung ein Programm zur Ausführung zugeordnet werden, z.B. „.doc“ → Word
  - Analog kann Dateien mit der Endung „.py“ der Python-Interpreter zugeordnet werden. Dann können Python-Programme auch per Doppelklick gestartet werden.
- Unix / Linux: Erste Zeile in der Programmdatei:

```
#!/usr/bin/python
```

- Aus einem Editor, der diese Funktionalität anbietet.
- Aus einer Entwicklungsumgebung (IDE).

# Python-Interpreter: Interaktiver Modus

- Möchte man Python-Anweisungen ausprobieren, ohne ein Programm in einer Textdatei zu erstellen, kann der Interpreter auch im „interaktiven“ Modus aufgerufen werden.
  - Kommandozeile: Eingabe von `python` (bzw. `python3` unter Unix)
  - Über das Windows-Startmenu
- Der Interpreter meldet sich mit dem „Prompt“: `>>>`
- Jetzt können Anweisungen eingegeben werden, die mit der Return-Taste abgeschlossen werden, z.B.:

```
>>> print("Hallo")  
Hallo
```

# Erstellung eines Python-Programms

- Ein Python-Programm ist eine Textdatei mit der Endung „.py“
- Zur Erstellung kann also ein beliebiger Texteditor verwendet werden, z.B. notepad(++).
- Der Quelltext darf keine Formatierungszeichen enthalten, wie sie beispielsweise von Textprogrammen (Word) erzeugt werden.
- Hilfreich sind Editoren, die eine Unterstützung für die Erstellung von Programmen bieten, z.B.
  - Syntaxhervorhebung
  - Start eines Programms aus dem Editor heraus
  - Start der interaktiven Shell aus dem Editor heraus



Je nach Editor werden also Teile des Programms hervorgehoben dargestellt. Die Textdatei enthält aber keine Formatierungszeichen.

# Der Editor Geany

- Allgemeiner Editor mit Python-Unterstützung
- Syntaxhervorhebung
- Start von (Python-) Programmen aus dem Editor
- [www.geany.org](http://www.geany.org)
- Einstellungen:
  - Zeichenkodierung UTF-8: Bearbeiten-Einstellungen-Dateien
  - Einrückung durch Leerzeichen: Bearbeiten-Einstellungen-Editor-Einrückung
  - Vor der Ausführung eines Programms aus Geany muss die Datei gespeichert sein (mit der Endung .py), sonst erkennt Geany nicht, dass es sich um ein Python-Prg. handelt.
  - Evtl. muss das Kommando zur Programmausführung angepasst werden:  
Erstellen – Kommandos zum Erstellen konfigurieren



# Thonny

- siehe [thonny.org](https://thonny.org)
- Thonny ist eine Python-Entwicklungsumgebung insbesondere für Anfänger.
- Eigenschaften
  - Syntaxhervorhebung
  - Ausführung von Python-Programmen aus dem Editor
  - Verwendung der interaktiven Shell aus dem Editor
  - Schrittweise Ausführung eines Python-Programms möglich
  - Anzeige von Variableninhalten
- Thonny bringt eine eigene Python-Implementierung mit. Es reicht also für die ersten Schritte, nur Thonny zu installieren.

# ❓ Übung - Ein erstes Python-Programm

- Die erste Übung beim Erlernen einer neuen Programmiersprache besteht im allgemeinen darin, ein Programm zu schreiben, das `Hallo Welt` ausgibt.
- Erstellen Sie eine Textdatei mit dem Namen `hallo_welt.py`, die genau eine Zeile enthält:

```
print("Hallo Welt")
```

- Rufen Sie das Programm auf:
  - aus dem Editor und
  - auf der Kommandozeile („Eingabeaufforderung“):
    - Wechseln Sie in das Verzeichnis, in dem die Datei liegt (Kommando `cd`)
    - Aufruf `python hallo_welt.py`

# Hinweis zum Aufruf des Interpreters

- Falls der Interpreter (python) nicht im Windows-Pfad enthalten ist, muss man beim Aufruf den vollständigen Pfad angeben, z.B:

```
C:\Python36\python programm.py
```

- Python zum Pfad hinzufügen:
  - Systemeinstellung, System und Sicherheit, System
  - Erweiterte Systemeinstellungen, Umgebungsvariablen
  - Systemvariablen
  - Bearbeitung der Variablen Path:
    - am Ende einfügen: Semikolon und Pfad zum Interpreter, z.B. %SystemRoot%  
...System32;C:\Python36\python programm.py
- Wenn man der Dateiendung „.py“ den Python-Interpreter zuordnet, kann man Python-Programme auch per Doppelklick starten.

# ❓ Übung: Interaktive Shell

- Starten Sie die Python-Konsole (über das Windows-Menü oder die Kommandozeile)
- Geben Sie `Hallo Welt` aus über die Eingabe von

```
print("Hallo Welt")
```

# Arbeiten mit Zahlen

- Ein Programm verarbeitet also Daten (EVA-Prinzip, EDV = Elektronische Datenverarbeitung)
- Welche Art von Daten?
- Zum Beispiel Zahlen:
  - Ganze Zahlen („Integer“, „int“):
    - Bsp.: Anzahl der Abteilungen (3)
  - Gleitkommazahlen („Floating Point“, „float“):
    - Werden in Python mit einem Punkt geschrieben: 0.8
    - Bsp: Rabatt-Faktor
- In der Shell kann man mit Zahlen rechnen wie mit einem Taschenrechner ⓘ

# Operationen auf Zahlen

- Grundrechenarten: `+`, `-`, `*`, `/`
- Ganzzahlige Division: `//`
- Rest bei der ganzzahligen Division: `%`
- Potenzieren: `**`



Die „normale“ Division mit `/` führt auch bei ganzen Zahlen zu einer Gleitkommazahl: `8 / 4` ergibt also die Gleitkommazahl `2.0`

# Ausdrücke

- Die Verknüpfung von Werten mittels Operatoren nennt man Ausdruck.
- Bsp:  $3 * (7+8)$
- Ein Ausdruck wird zu einem Wert ausgewertet.
- Im obigen Beispiel ist der Wert des Ausdrucks die ganze Zahl 45.

# Variablen

- Um in einem Programm Werte wieder verwenden zu können, kann man Werte Variablen zuweisen:

```
anzahl_abteilungen = 3  
rabatt_faktor = 0.8
```

- Eine Variable ist also eine Art „Behälter“ für einen Wert.
  - `anzahl_abteilungen` beinhaltet als Wert die ganze Zahl `3`
  - `rabatt_faktor` die Gleitkommazahl `0.8`



Das Gleichheitszeichen `=` wird also für die **Zuweisung** eines Wertes zu einer Variablen verwendet. Es bedeutet nicht *mathematische Gleichheit*.



# Variablen 2

- Eine Variable kann in einem Ausdruck statt eines Wertes verwendet werden. Es wird dann zur Berechnung der zugewiesene Wert genommen.

```
beitrag = (60 + anzahl_abteilungen * 20) * rabatt_faktor
```

- Der Wert auf der rechten Seite wird berechnet („der Ausdruck wird ausgewertet“) und dann der Variablen auf der linken Seite zugewiesen.

# Variablen 3

Einer Variablen kann auch ein neuer Wert zugewiesen werden. Dieser wird dann ab dem Zuweisungszeitpunkt genommen, z.B.:

```
grundbeitrag = 60  
rabatt_faktor = 0.8  
beitrag = grundbeitrag * rabatt_faktor
```

Die Variable `beitrag` hat nun den Wert 48.

```
rabatt_faktor = 0.5  
beitrag = 60 * rabatt_faktor
```

Die Variable `rabatt_faktor` hat nun den Wert 0.5 und die Variable `beitrag` den Wert 30.

# Variablen 4

- Eine Variable ist im Programm bekannt (und kann somit verwendet werden), sobald ihr zum ersten mal ein Wert zugewiesen wurde (und existiert nur während der Programmausführung!).
- Beispiel:

```
beitrag = 60 * faktor
```

liefert einen Fehler, da die Variable `faktor` noch nicht bekannt ist.

- Das folgende funktioniert:

```
faktor = 0.8  
beitrag = 60 * faktor
```

Zuerst wird eine Variable `faktor` eingeführt, die dann zur Berechnung von `beitrag` benutzt wird.

# Variablen 5

- Eine Variable kann auch auf der linken und rechten Seite einer Zuweisung vorkommen.
- Beispiel: Der Wert der Variablen `summe` soll um den Wert der Variablen `x` erhöht werden:

```
x = 5  
summe = 1  
summe = summe + x
```

Die Variable `summe` hat nun den Wert `6`.

- Das kommt sehr oft vor. Daher gibt es eine abkürzende Schreibweise:

```
summe += x
```

# Variablen 6

- Gültige Variablennamen („Bezeichner“) bestehen aus:
  - Buchstaben (groß und klein)
  - Unterstrich `_`
  - Ziffern ( `0 - 9` ) (aber nicht am Anfang)
  - Unicode-Zeichen
- beliebige Länge
- Groß- und Kleinschreibung wird unterschieden
  - `meine_zahl` und `meine_Zahl` bezeichnen also verschiedene Variablen
- Python-Schlüsselwörter dürfen nicht verwendet werden.
- Konvention: Kleinschreibung und `_`, z.B.
  - `meine_variable`, `anzahl_abteilungen`, `alter_in_hundejahren`

# Funktionen

- Beispiel:

```
x = -5  
x = abs(5)
```

- Aufruf einer Funktion über den Funktionsnamen ( `abs` ) und Argumenten in runden Klammern.
- Der Aufruf gibt einen Wert zurück.
- Der Aufruf einer Funktion kann an jeder Stelle in einem Ausdruck erfolgen, an der auch ein Wert oder eine Variable stehen kann.
- Beispiel:

```
y = 3 * abs(x) + 7
```

# Bibliotheken


- Funktionen können in sog. Bibliotheken „gesammelt werden“.
- Eine Bibliothek muss vor ihrer Verwendung importiert werden.
- Bei der Benutzung wird der Bibliotheksname vorangestellt. Dann folgt der Funktionsname mit einem Punkt getrennt.
- Bsp.:

```
import math  
x = math.sqrt(5)
```

# Ein- und Ausgabe

- Erinnerung: EVA-Prinzip
- Ausgabe: Z.B. auf dem Bildschirm:

```
x = 5  
print("Der Wert ist: ", x)
```

- Ausgabe von beliebig vielen Werten durch Komma getrennt
- `print` ist eine Funktion. Der Rückgabewert interessiert uns nicht.
- Eingabe: Z.B. mit der Tastatur: `input`-Funktion
  - Ganze Zahl: `x = int(input("Geben Sie eine Zahl ein: "))`
  - Gleitkommazahl: `x = float(input("Geben Sie eine Zahl ein: "))`
- Der Text zwischen `" "` wird genau wie angegeben auf dem Bildschirm ausgegeben (mehr dazu später, Stichwort „Zeichenkette“ („String“)).
- Beispiel: `ein_ausgabe.py` 



# Beispiel Ein- und Ausgabe

*# Eine Zeile, die mit # anfaengt, wird vom Interpreter ignoriert und dient als Kommentar*

*# Eingabe einer ganzen Zahl*

```
ganze_zahl = int(input("Geben Sie eine ganze Zahl ein: "))  
print("Der eingegebene Wert ist: ", ganze_zahl)
```

*# Eingabe einer Gleitkommazahl*

```
gleitkomma_zahl = float(input("Geben Sie eine Gleitkommazahl ein: "))  
print("Der eingegebene Wert ist: ", gleitkomma_zahl)
```

*# Ausgabe von mehreren Werten:*

```
x1 = 5  
x2 = 3  
x3 = 8  
print("x1=", x1, "x2=", x2, "x3=", x3)
```

# Programm

- Ein Programm ist eine Sequenz von Anweisungen (und Ausdrücken), die nacheinander ausgeführt werden.
- Anweisungen: Bisher kennengelernt:
  - Zuweisung ( `variable = Ausdruck` )
- Also bisher:
  - Eingabe via `Input` -Funktion und Zuweisung an eine Variable
  - Zuweisungen (von Werten, die mittels Ausdrücke berechnet werden)
  - Ausgabe via `print` -Funktion
- Bsp.: Programm, das drei Zahlen einliest und den Durchschnitt berechnet:
  - `durchschnitt.py` !

# Beispiel Programm zur Durchschnittsberechnung

```
# Zeilen, die mit # anfangen, werden vom Interpreter ignoriert und können für  
Kommentare verwendet werden  
#
```

```
# Eingabe
```

```
zahl1 = int(input("Geben Sie die erste Zahl ein: "))  
zahl2 = int(input("Geben Sie die zweite Zahl ein: "))  
zahl3 = int(input("Geben Sie die dritte Zahl ein: "))
```

```
# Berechnung
```

```
durchschnitt = (zahl1 + zahl2 + zahl3) / 3
```

```
# Ausgabe
```

```
print("Der Durchschnitt der drei Zahlen ist: ", durchschnitt)
```

# Kommentare

- Text, der in einer Zeile hinter dem `#`-Zeichen stehen, werden vom Python-Interpreter ignoriert.
- Dies kann zur Kommentierung des Programmtextes verwendet werden.
- Kommentare sollten mit Bedacht verwendet werden.

# ? Übung 01 - Berechnung Stunden und Minuten

Schreiben Sie ein Programm, das eine ganze Zahl einliest („Anzahl Minuten“) und diese Minutenanzahl in Stunden und Minuten umrechnet und ausgibt.

Beispiel:

- Eingabe: 136
- Ausgabe: 2 Stunden und 16 Minuten

siehe Ordner `uebungen/01_minuten`

# Fallunterscheidungen

- Je nach dem, ob eine Bedingung erfüllt ist, werden andere Anweisungen ausgeführt.
- Umgangssprachliches Beispiel [\[Kle\]](#):

Wenn es morgen regnet, werde ich den Keller entrümpeln. Dann werde ich die Schränke aufräumen und Fotos sortieren.

Werden die Schränke nur im Regenfall aufgeräumt?

- Wie sieht es jetzt aus?

Wenn es morgen regnet, werde ich den Keller entrümpeln. Dann werde ich die Schränke aufräumen und Fotos sortieren. Ansonsten werde ich schwimmen gehen. Abends gehe ich mit meiner Frau ins Kino.

# Blöcke

- Es gibt also drei „Blöcke“:
  - Tätigkeiten, die nur ausgeführt werden, wenn eine Bedingung erfüllt ist.
  - Tätigkeiten, die nur ausgeführt werden, wenn die Bedingung nicht erfüllt ist.
  - Tätigkeiten, die danach auf jeden Fall ausgeführt werden.
- Das kann man durch Einrückungen veranschaulichen [\[Kle\]](#):

Wenn es morgen regnet:

    Werde ich den Keller entrümpeln.

    Dann werde ich die Schränke aufräumen  
    und Fotos sortieren.

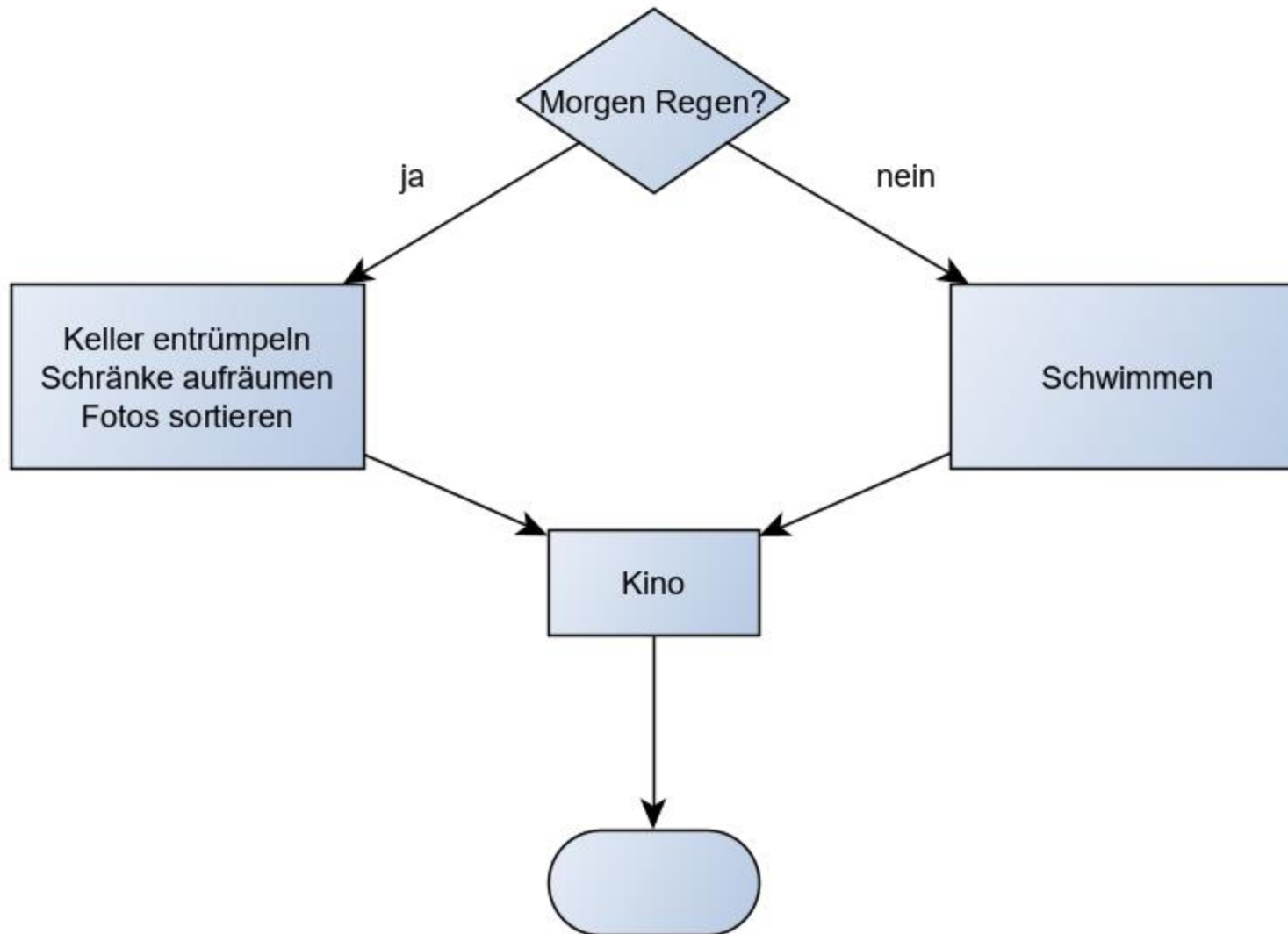
Andernfalls:

    Werde ich schwimmen gehen.

Abends gehe ich mit meiner Frau ins Kino

# Fallunterscheidung im Flussdiagramm

Nach [\[Kle\]](#)





# Blockbildung in Python

- Wie im Beispiel findet die Blockbildung durch Einrückung statt.
- Anweisungen, die gleich weit eingerückt sind, gehören zusammen und bilden einen Block.
- Achtung: Einrückung muss einheitlich durch Leerzeichen oder Tabulatoren erfolgen. Empfehlung: Leerzeichen.
- Beispiel [\[Kle\]](#):

```
if Regen:
    keller_entrümpeln()
    schraenke_aufraeumen()
    fotos_sortieren()
else:
    schwimmen_gehen()
kino_gehen(ehefrau)
```

# Einfache if-Anweisungen

- Allgemeine Form

```
if bedingung:  
    anweisungen  
anweisungen
```

- Beispiel [\[Kle\]](#):

```
alter = int(input("Dein Alter"))  
if alter < 12:  
    print("Zu jung")  
    print("Waehle einen anderen Film")  
print("Programmende")
```

# If-Anweisung mit Else-Teil

- Allgemeine Form

```
if bedingung:  
    anweisungen  
else:  
    anweisungen  
anweisungen
```

- Beispiel [\[Kle\]](#):

```
alter = int(input("Dein Alter"))  
if alter < 12:  
    print("Zu jung")  
    print("Waehle einen anderen Film")  
else:  
    print("Viel Spass")  
print("Programmende")
```

# „Verschachtelte“ Blöcke

- In einem Anwendungsblock, der in einer If-Anweisung vorkommt, kann auch eine weitere If-Anweisung vorkommen.
- Das nennt man dann verschachtelte If-Anweisung.
- Bsp [\[Kle\]](#):

```
alter = int(input("Dein Alter"))
if alter < 4:
    print("Der Film ist zu kompliziert")
else:
    if alter < 12:
        print("Viel Spass")
    else:
        if alter < 16:
            print("Bist Du Dir sicher?")
        else:
            print("Wollen Sie sich das antun?")
print("Programmende")
```

# Elif

- Das ist im Zweifelsfall schwer zu lesen. Daher gibt es mit „elif“ eine Abkürzung [\[Kle\]](#)

```
alter = int(input("Dein Alter"))
if alter < 4:
    print("Der Film ist zu kompliziert")
elif alter < 12:
    print("Viel Spass")
elif alter < 16:
    print("Bist Du Dir sicher?")
else:
    print("Wollen Sie sich das antun?")
print("Programmende")
```

# Bedingungen

- Was genau steht hinter dem „if“?
- Ein „Ausdruck“, dessen Wert „wahr“ oder „falsch“ ist.
- Vergleichsoperatoren ⓘ:

Operator	Bedeutung	Beispiel
==	gleich	42 == 42
!=	ungleich	42 != 43
<	kleiner	42 < 43
<=	kleiner gleich	42 <= 42
>	größer	43 > 42
>=	größer gleich	42 >= 42

# Logische Operatoren

- Wahrheitswerte lassen sich mit logischen Operatoren verknüpfen
- Diese sind: `and`, `or`, `not`
- Beispiel: Prüfung, ob eine Zahl `a` zwischen 20 und 100 liegt:

```
if 20 <= a and a <= 100:
```

- Prüfung des Gegenteils: **i**
  - `a` liegt nicht zwischen 20 und 100:

```
if not (20 <= a and a <= 100):
```

- `a` ist kleiner als 20 oder größer als 100

```
if a < 20 or a > 100:
```

- Wahrheitstabellen **i**

# Erläuterung



# Der Datentyp `bool`

- Bisher: Rechnen mit Zahlen
  - ganze Zahlen
  - Gleitkommazahlen
- Wahrheitswerte stellen einen weiteren Datentyp dar.
- Es gibt genau zwei Elemente diesen Datentyps:
  - `True`
  - `False`
- Auch Werte diesen Datentyps können einer Variablen zugewiesen werden.
- Es können Ausdrücke gebildet werden mit den Operatoren `and`, `or` und `not`.
- Beispiel: `datentyp_bool.py` !

# Beispiel Datentyp boolean

```
w1 = True
w2 = False
```

```
print("w1 and w2: ", w1 and w2)
print("w1 or w2: ", w1 or w2)
print("not w1: ", not w1)
```

```
alter = int(input("Dein Alter: "))
ist_jugendlich = (alter <= 18)
ist_senior = (alter >= 65)
print("ist_jugendlich: ", ist_jugendlich)
print("ist_senior: ", ist_senior)
```

```
if ist_jugendlich:
    print("Du bekommst einen Junioren-Rabatt!")
elif ist_senior:
    print("Sie bekommen den Seniorenrabatt!")
else:
    print("Sie muessen noch ", 65 - alter , " Jahre warten, um den Senioren-
Rabatt zu bekommen.")
print("Programmende")
```

# ❓ Übungen If-Anweisung

- Übung 2: Erweiterung des Minutenbeispiels um Abfrage auf negative Zahlen  
siehe Ordner `Uebungen\02_minuten_mit_if`
- Übung 3 [\[Kle\]](#): Umrechnung Hundejahre in Menschenjahre  
siehe Ordner `Uebungen\03_hunde_jahre`

# Zeichenketten (Strings)

- Bisher: Zahlen und Wahrheitswerte
- Oft werden auch Zeichenketten benötigt, z.B. Namen von Vereinsmitgliedern
- Definition mit einfachen oder doppelten Hochkommata:

```
name = "Christian"  
name = 'Christian'
```

- Ein- / Ausgabe:

```
name = input("Geben Sie Ihren Namen ein.")  
print("Hallo", name)
```

- Eine Zeichenkette ist eine Sequenz (Folge) von Zeichen.
- Zeichen: Beliebiges (Unicode-) Zeichen (also z.B. auch chinesische Zeichen)

# Umwandlung von Datentypen



Beachte Unterschied zwischen der Zahl 42 und der Zeichenkette bestehend aus den Zeichen '4' und '2'.

```
zahl = 42  
zeichenkette = "42"
```

- Mit Zahlen kann man rechnen, mit Zeichenketten nicht.
- Das folgende liefert einen Fehler: `x = zahl + zeichenkette`
- Eine Zeichenkette kann in eine Zahl umgewandelt werden (wo es sinnvoll ist), und umgekehrt:

```
ganze_zahl = int("42")  
gleitkomma_zahl = float("3.14")  
zeichenkette = str(42)
```

# Input-Funktion

- Jetzt erschließt sich auch die Eingabe von Zahlen via der Input-Funktion:

Von der Tastatur werden immer Zeichenketten gelesen. Diese müssen gegebenenfalls in eine Zahl umgewandelt werden.

- Die Umwandlung geschieht mit den Funktionen `int()` und `float()`:

```
x = int(input("Geben Sie eine Zahl ein: "))  
x = float(input("Geben Sie eine Zahl ein: "))
```

# Länge und Indizierung

- Eine Zeichenkette hat eine Länge (Anzahl der Zeichen):

```
s = "Python"  
print(len(s))
```

- Länge kann auch 0 sein. Man spricht dann vom „leeren“ String

```
leerer_string = ""  
print(len(leerer_string)) ❶
```

❶ Ergibt 0.

- Zugriff auf das i-te Zeichen via „Indizierung“: `s[i]`
- Dabei kann man auch von hinten zählen: `s[-1]`

# Operationen auf Zeichenketten

- Zeichenketten können verglichen werden: `==`, `<`, `<=`, `>`, `>=`
- `+` : Konkatenation („Aneinanderhängen“)
- `*` : Wiederholung
- Ausschneiden („Slicing“): `"Python"[2:4]`
  - Der letzte Index gehört nicht dazu
  - obiges Beispiel liefert also die Zeichenkette: `"th"`



# Zugriff auf Zeichen

- Zugriff auf ein Zeichen eines Strings wie gesehen via Indizierung

```
s = "Python"  
print(s[1])
```

- Es ist aber nicht möglich, durch Zuweisung ein Zeichen zu ändern:

```
s[0] = "J" # Fehler
```

- Ein String-Wert ist unveränderlich!
- Daher nennt man den Datentyp String einen **unveränderlichen Datentyp**.
- Möchte man ein Zeichen ändern, muss man eine neue Zeichenkette erzeugen (die man wieder der Variablen s zuweisen kann):

```
s = "J" + s[1:len(s)]
```

# ? Übung 04 - Zeichenketten

- Eingabe:
  - Eine Zeichenkette
  - Ein Index  $i$  (ganze Zahl)
  - Ein Faktor  $f$  (ganze Zahl)
- Das Programm soll  $f$  mal das  $i$ -te Zeichen ausgeben.
- Beispiel:
  - Eingabe "Hallo", 2, 10
  - Ausgabe llllllllll
- siehe Ordner Uebungen\04\_zeichenkette
- Zusatzaufgabe: Was passiert, wenn Sie einen „falschen“ Index angeben. Verhindern Sie dies durch eine Abfrage (if-Anweisung)

# Referenzen

- [Kle] Bernd Klein, Einführung in Python 3, Hanser-Verlag