

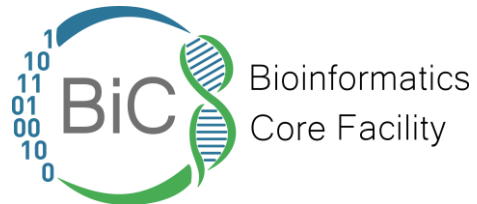
Programmieren für Einsteiger

- Arbeiten mit numerischen Daten in Python -

Tag 1

26.07.2022

Emanuel Barth, Daria Meyer





Organisatorisches & Kennenlernen

Organisatorisches



09:00 - 16:00, 12:00 - 13:00 Mittagspause
Bei Bedarf auch gerne Pausen zwischendurch



Aller Materialien werden hier zur Verfügung gestellt:
<https://git.uni-jena.de/du47nid/python-workshop-2022>



Literaturempfehlung:
"Objektorientiertes Programmieren mit Python 3" von Michael Weigend.

Wer sind wir?



Wer seid ihr?



Aufgabe: Stellt euch kurz vor.



Name



Erwartungen



Fachrichtung



Vorkenntnisse



Motivation



Besonderheit

Ausblick

Dienstag:

- Python: Überblick und Installation
- Datentypen und Variablen
- einfache Berechnungen

Mittwoch:

- Python-Skripte
- Kontrollstrukturen
- Funktionen

Donnerstag:

- Dateien lesen und schreiben
- NumPy

Freitag:

- Pandas
- Daten visualisieren
- Arbeiten an eigenem Projekt

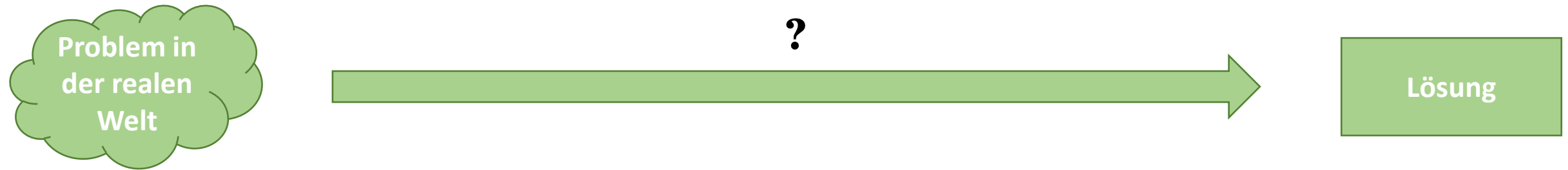
Lernziele für Heute

- Du kennst den Unterschied zwischen Programmieren und algorithmisch Denken.
- Du hast einen ersten Überblicks über die Programmiersprache Python (Historie, Eigenschaften, Vor- und Nachteile).
- Du kannst Python installieren und erste einfache Programme erstellen und diese über die Kommandozeile oder die IDLE ausführen.
- Du hast eine Vorstellung davon wie Variablen in Python funktionieren und kannst den verschiedenen Datentypen ihrem Nutzen zuordnen.
- Du kannst erklären warum Datentypumwandlung nötig sein kann und wie man diese in Python anwendet.

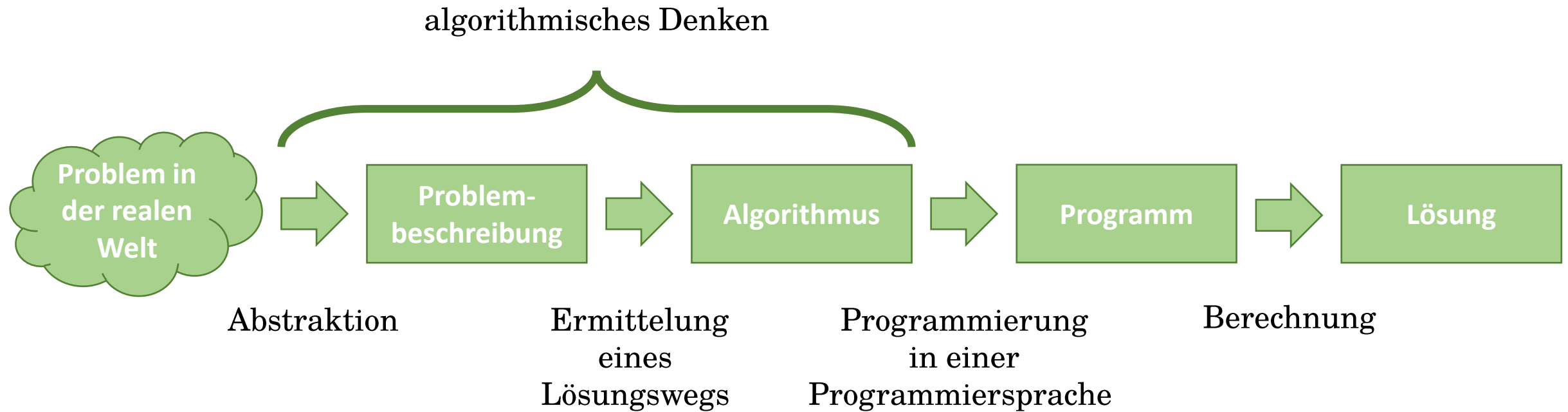
Algorithmus vs. Programmieren

- Was ist ein Algorithmus?
- Was ist Programmieren?

Was ist Programmieren?



Was ist Programmieren?



Was ist Programmieren?

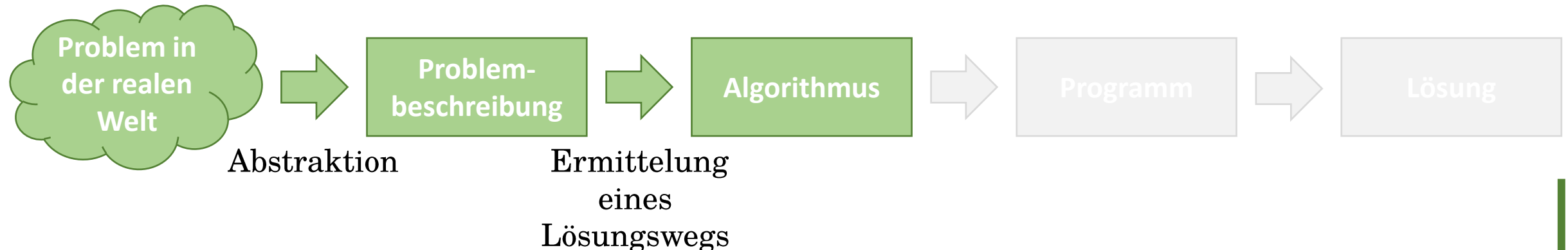
Problem: Du möchtest am Wochenende eine Party bei dir zuhause feiern und musst diese nun vorbereiten.

Problembeschreibung:

- Wohnung aufräumen
- Getränke und Essen einkaufen
- Party-Deko aufhängen
- Freunde einladen
- Nachbarn Bescheid sagen

Teilprobleme:

1. Schmutzige Wäsche einsammeln und in den Wäschekorb legen
2. Staubsaugen und Staubwischen
3. Bad putzen
4. Geschirr abwaschen
5. Geschirr in den Schrank räumen
6. Zum Supermarkt gehen um Wein, Cola, Bier, Chips, ... einzukaufen
7. ...



Was ist Programmieren?

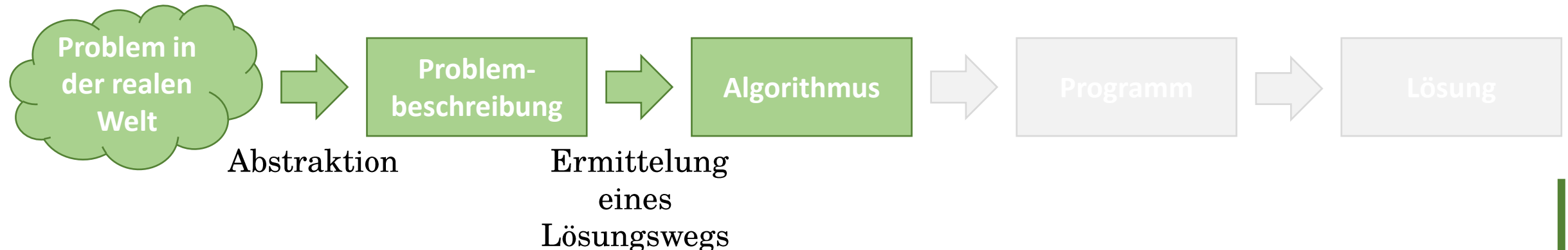
Problem: Deine Chefin möchte, dass du die Ergebnisse einer Studie A mit denen einer Studie B vergleichst.

Problembeschreibung:

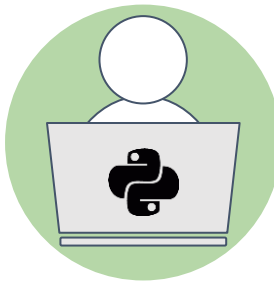
- Finden und herunterladen der Datensätze beider Studien
- Ordne die Daten so zueinander an, dass sie jeweils vergleichbar sind
- Verwerfe die restlichen Daten
- Wenn genügend Daten zum Vergleich übrig sind, dann vergleiche sie
- Teile deiner Chefin das Ergebnis deines Vergleichs mit

Teilprobleme:

1. Gehe auf scholar.google.de und tippe dort den Titel von Studie A ein.
2. Klicke auf "Suchen".
3. Wähle den passenden Link aus und folge ihm.
4. Suche auf der Studienseite nach dem "Datensatz Download"-Button.
5. Klicke diesen Button und speichere den Datensatz im Ordner "Downloads".
6. Wiederhole Schritt 1 bis 5 für Studie B.
7. ...



Was ist Programmieren?



Aufgabe: Unten sind zwei Problemstellungen gegeben. Zerlege diese Probleme in möglichst einfache und konkrete Teilprobleme.

Problem 1:

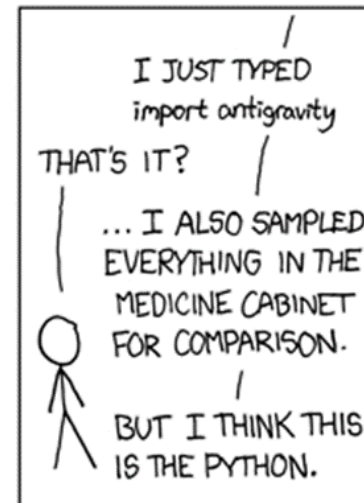
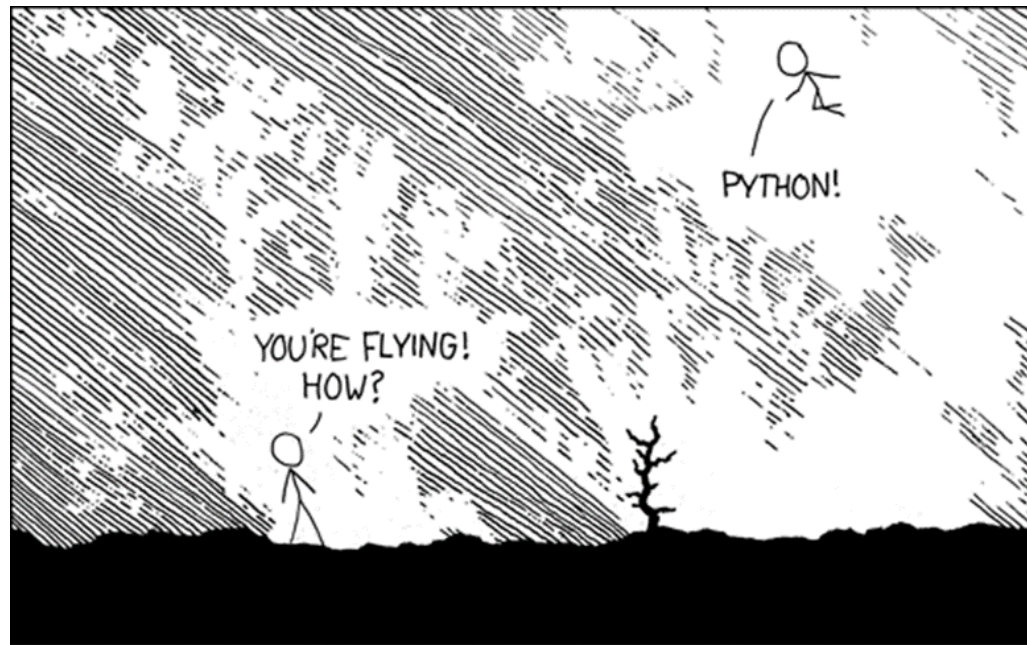
Gegeben eine Excel-Datei. Erstelle ein Häufigkeitsdiagramm aus allen numerischen Werten aus Spalte 3, wenn in der betreffenden Zeile in Spalte 1 der Begriff "Laser" auftaucht. Danach berechne Mittelwert und Standardabweichung über alle Werte aus Spalte 5.

Problem 2:

Gegeben eine große Textdatei. Speichere alle Jahresangaben sortiert nach der Häufigkeit ihres Vorkommens im Text in eine separate Datei.

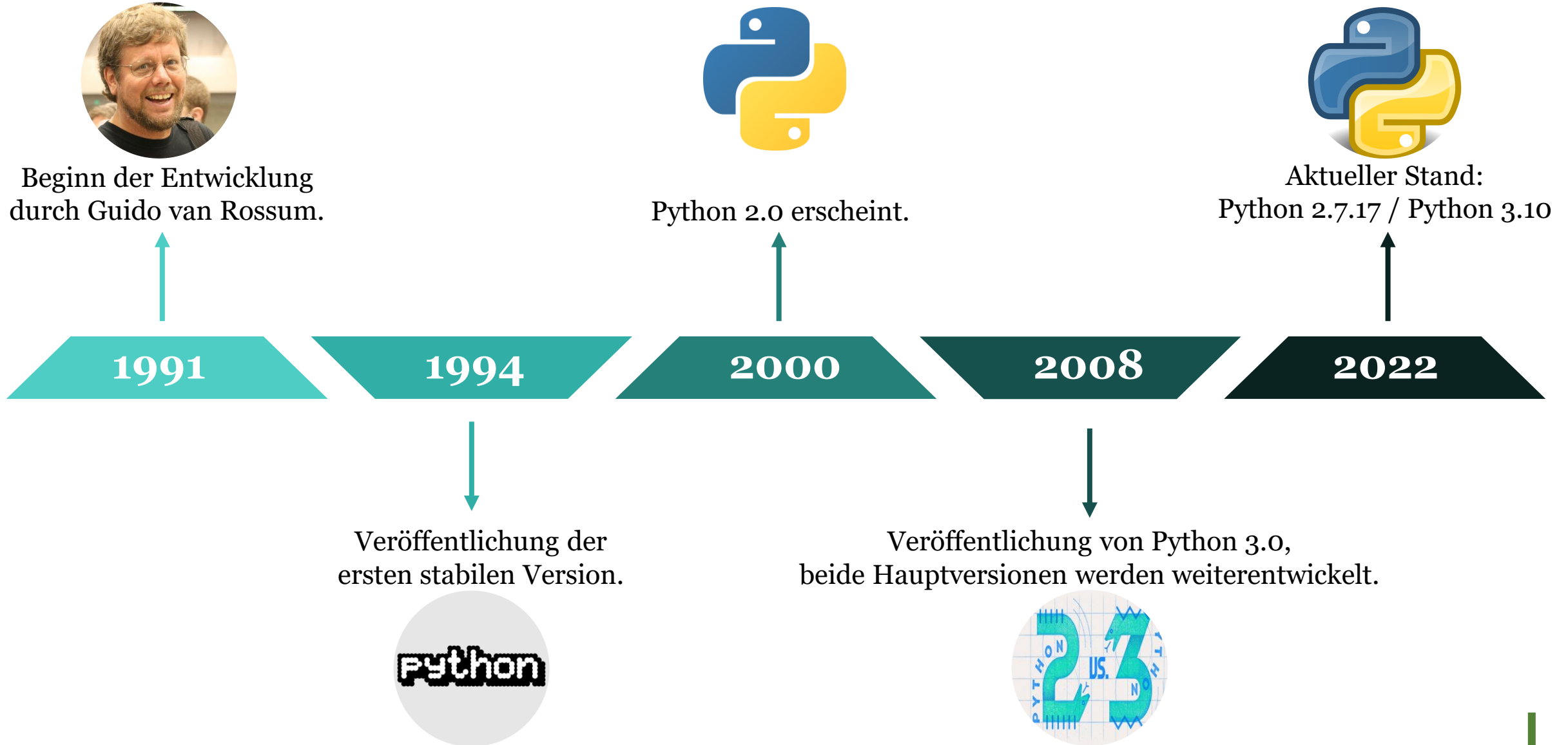
	A	B	C	D	E
1	# Start der Messungen von Experiment X				
2					
3	Messverfahren	Datum	Wert X	Name	Wert Y
4	Laser	19.09.2021	5	Ada	18,753
5	Radiometer	19.09.2021	19	Ada	12,996
6	Laser	19.09.2021	8	Ada	11,154
7	Radiometer	20.09.2021	36	Bernd	16,549
8	Radiometer	21.09.2021	119	Clara	16,734
9	Laser	21.09.2021	12	Clara	13,005
10	Nicht-Laser	21.09.2021	8	Daniel	17,991
11	laser	21.09.2021	19	Daniel	15,555

Gertrude Blanch wurde 1938 als technische Direktorin des Mathematical Tables Project eingestellt.[14][15] Ihre Arbeit bildete den Grundstein für den Übergang von handgesteuerten Rechenmaschinen zur modernen Computerära. Vier Jahre später (1942) wurden Hedy Lamarr und George Antheil mit ihrer gemeinsamen Erfindung, welche dem heutigen Bereich der „wireless communication“ zugeordnet wird, bekannt. Marlyn Meltzer begann im Jahre 1945 ihre Laufbahn in der Informatik. Aufgrund ihrer Kompetenz in Mathematik und der Bedienung von Rechenmaschinen wurde sie in das erste Team der ENIAC-Programmiererinnen aufgenommen. Obwohl 6 Frauen an der Programmierung des ENIAC maßgeblich beteiligt waren, wurden im Februar 1946 bei dessen Präsentation nur die beteiligten Männer gewürdigt. Kathy Kleiman machte die Öffentlichkeit darauf aufmerksam und erreichte so, dass im Jahre 1997 die 6 beteiligten Programmiererinnen bei einem Festakt in Silicon Valley nach über 50 Jahren ausgezeichnet wurden.[12] Diese Programmiererinnen – Kathleen McNulty Mauchly Antonelli, Jean Bartik, Frances Elizabeth „Betty“ Holberton, Marlyn Meltzer, Frances Spence und Ruth Teitelbaum waren ab diesen Zeitpunkt als ENIAC-Frauen[16] bekannt und wurden 1997 in die Women in Technology International (WITI) Hall of Fame aufgenommen. Kathleen Booth entwickelte 1948 die ARC Assembler Sprache für Computersysteme am Birkbeck College der University of London.[17]



Allgemeines über Python

Entwicklungsgeschichte



Was wünscht ihr euch von einer Programmiersprache?

- Welche Eigenschaften sollte sie haben?
- Was sollte man mit ihr umsetzen können?

Python ist...



eine Skriptsprache (d. h. interpretiert statt kompiliert).



offen entwickelt und verwaltet durch die Python Software Foundation.



Ursprünglich implementiert in C, aber es existieren mittlerweile weitere Implementierungen (Jython, IronPython, PyPy, ...).



einfach zu erlernen und sehr flexibel.

Warum Python?

Minimalistisch

Es existieren nur relative wenige Schlüsselwörter mit einer auf Übersichtlichkeit fokussierten Syntax.

Multi- paradigmen- sprache

Verschiedene Stile werden unterstützt, z. B. funktionale, aspekt- oder objektorientierte Programmierung.

Interaktiv

Dank der Pythonshell und anderen interaktiven Terminals eignet sich die Sprache zum schnellen Testen von Code.

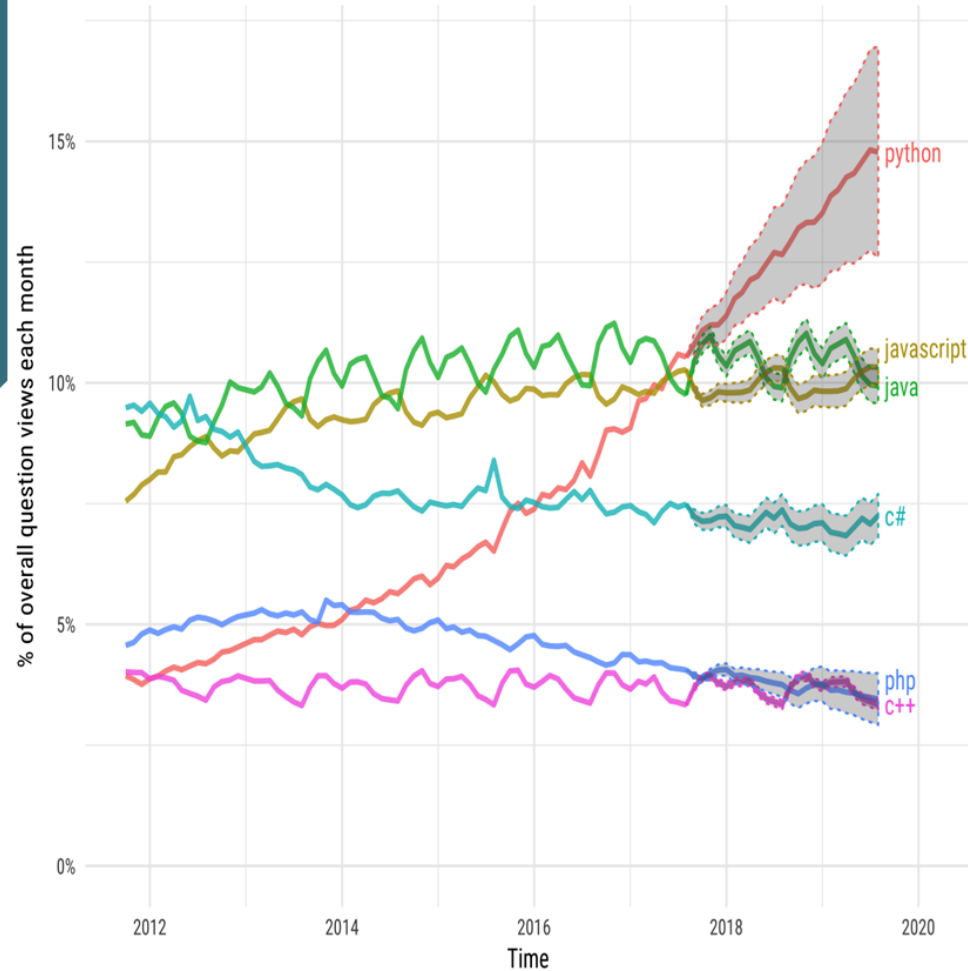
Aktive Community

Viele begeisterte und hilfreiche Programmierer und eine große Auswahl an verschiedenen Modulen.

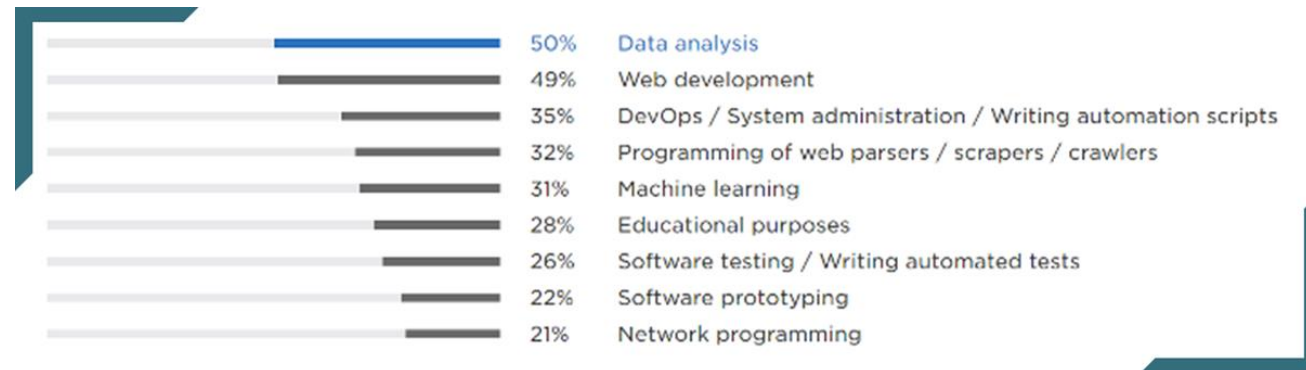
Warum Python?

Projections of future traffic for major programming languages

Future traffic is predicted with an STL model, along with an 80% prediction interval.



Data from stackoverflow.com



Data from python.org

The Zen of Python

- Beautiful is better than ugly.
- Explicit is better than implicit.
- Simple is better than complex.
- Complex is better than complicated.
- Flat is better than nested.
- Sparse is better than dense.
- Readability counts.
- Special cases aren't special enough to break the rules.
- Although practicality beats purity.
- Errors should never pass silently.
- Unless explicitly silenced.
- In the face of ambiguity, refuse the temptation to guess.
- There should be one-- and preferably only one --obvious way to do it.
- Although that way may not be obvious at first unless you're Dutch.
- Now is better than never.
- Although never is often better than **right** now.
- If the implementation is hard to explain, it's a bad idea.
- If the implementation is easy to explain, it may be a good idea.
- Namespaces are one honking great idea -- let's do more of those!



Python Code Beispiel

```
import os
import sys

#This Scripts reads a Fasta file and parses it into a dictionary.

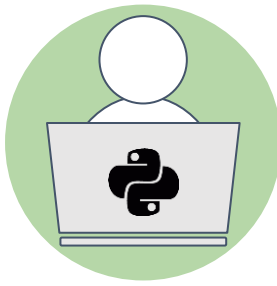
fastaDict = {}
fastaFile = open('/home/emanuel/test.fna')
header = ''
sequence = ''

for line in fastaFile:                                #Start reading file
    if line[0] == '>':                                  #Search for header
        if len(header) > 0:
            fastaDict[header] = sequence                #Create new dictionary entry
            sequence = ''
            header = line
        else:
            sequence += line                            #No header, extend sequence

fastaDict[header] = sequence                            #Final dictionary entry
fastaFile.close()                                     #Close file handle
```

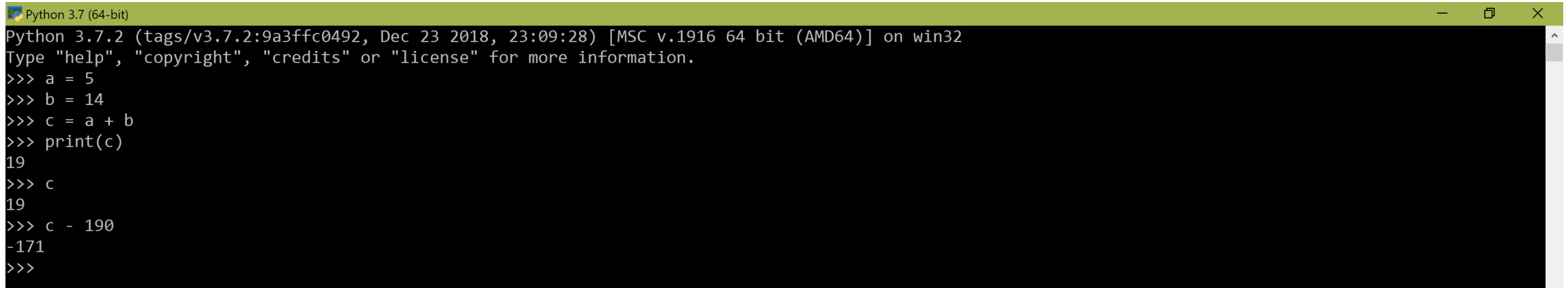

Python installieren

- Offizielle Seite: <https://www.python.org/downloads/>
- Wir arbeiten mit **Python 3.6** oder höher.
- Folgende optionale Features sollten mitinstalliert werden:
 - IDLE
 - pip
 - tcl\tk
- Alternativ kann man auch miniconda installieren:
<https://docs.conda.io/en/latest/miniconda.html>



Aufgabe: Installiert die neuste Version von Python auf eurem PC und startet Python im Anschluss.
Was seht ihr?

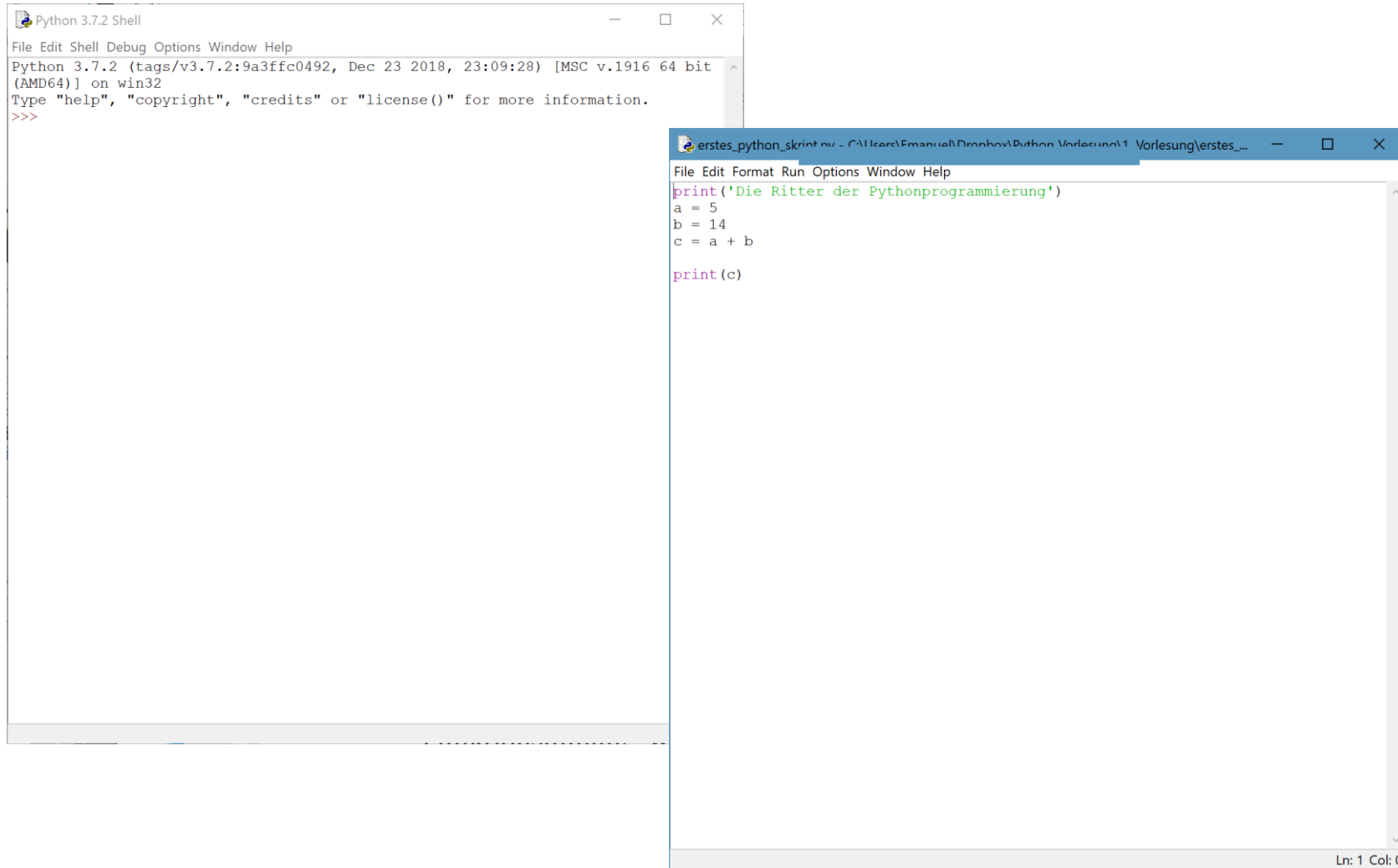
Die Pythonshell



```
Python 3.7 (64-bit)
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 23:09:28) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> a = 5
>>> b = 14
>>> c = a + b
>>> print(c)
19
>>> c
19
>>> c - 190
-171
>>>
```

- Durch die Pythonshell kann man interaktiv mit dem Pythoninterpreter arbeiten.
- Die Pythonshell eignet sich um Codeabschnitte oder einzelne Funktionen ausgiebig zu testen.
- Neben der Standardshell gibt es auch Varianten mit einem größerem Funktionsumfang, wie z. B. IPython (<http://ipython.org/>) → Dazu später mehr.

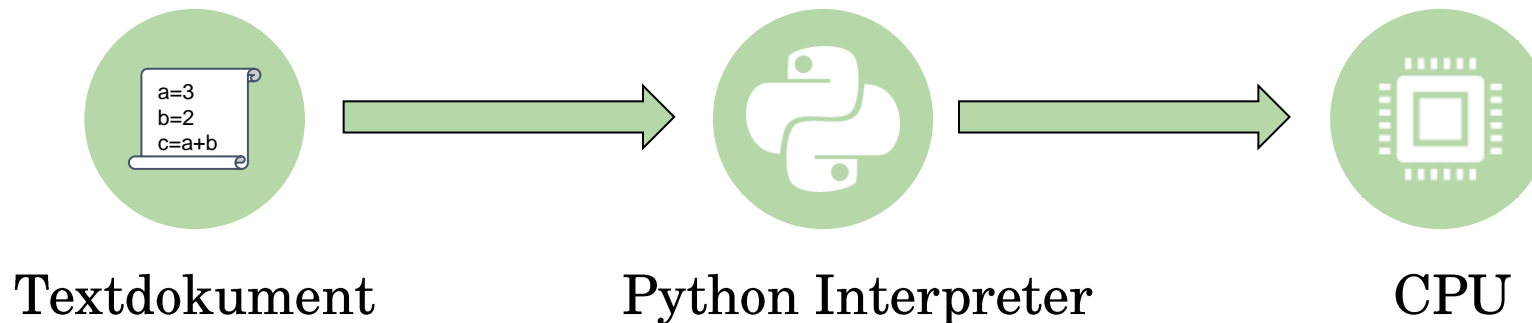
Die IDLE



- Die IDLE ist der mitgelieferte Standardeditor von Python mit integrierter Pythonshell.
- Über den Editor lassen sich Pythonskripte erstellen, modifizieren und direkt ausführen.

Pythonskripte

- Ein Pythonskript ist letztendlich nur eine einfache Textdatei, welche (hoffentlich) ausführbaren Pythoncode enthält.
- Neben Python Code können auch Kommentare, markiert durch ein #-Symbol, im Skript stehen.
- Pythonskripte können mit einem beliebigen, einfachen Texteditor erstellt und verändert werden und sind üblicherweise mit der Dateiendung **.py** gekennzeichnet.
- Übergibt man einem Pythoninterpreter eine Datei, so versucht er diese auszuführen, in dem er den darin enthaltenen Code (Text) interpretiert.



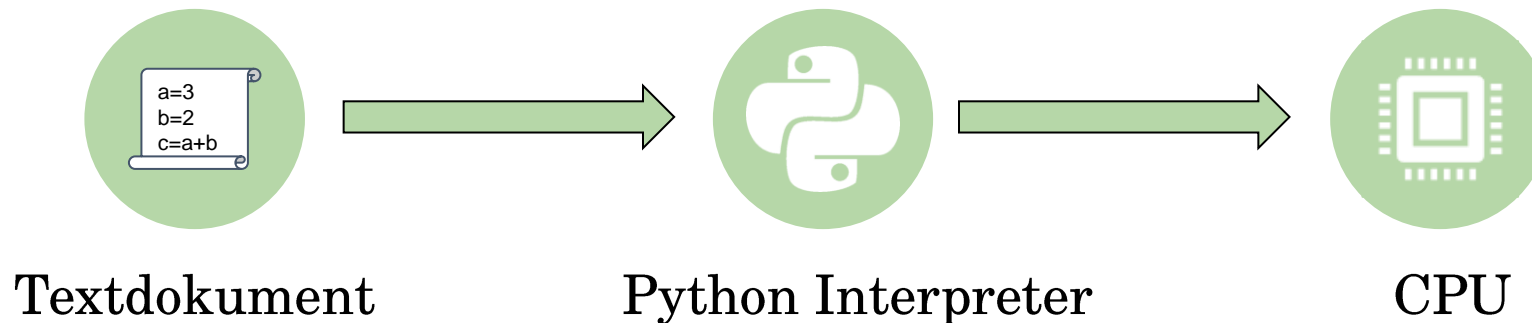


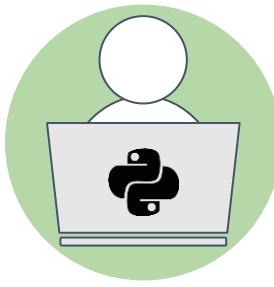
Aufgabe:

1. Ladet euch die drei Dateien mit den Nummern 04, 05 und 06 im Übungsmaterialordner herunter und speichert sie in eurem Downloadsordner.
2. Starte die Kommandozeile und navigiert mit folgendem Befehl in den Downloadsordner:

`cd C:\Users\USERNAME\Downloads\`

3. Versucht jeder der drei Dateien mit Hilfe des Pythoninterpreters auszuführen.





Aufgabe:

Führt die untenstehende einfache Code-Zeile auf die folgenden drei verschiedenen Arten aus:

```
print('Hallo Welt')
```

1. Über die interaktive Pythonshell.
2. Über ein Pythonskript, welches ihr mit Hilfe der IDLE erstellt habt.
3. In dem ihr selbst eine Pythonskriptdatei erstellt und über die Kommandozeile direkt an den Pythoninterpreter übergebt.

```
[1]> 2+"2"
=> "4"

[2]> "2"+[1]
=> "[2]"

[3]> (2/0)
=> NaN

[4]> (2/0)+2
=> NaN

[5]> ""+" "
=> " + "

[6]> [1,2,3]+2
=> FALSE

[7]> [1,2,3]+4
=> TRUE

[8]> 2/(2-(3/2+1/2))
=> NaN.00000000000000013

[9]> RANGE(" ")
=> (" ", " ", " ", " ", " ", " ")

[10]> +2
=> 12

[11]> 2+2
=> DONE

[14]> RANGE(1,5)
=> (1,4,3,4,5)

[13]> FLOOR(10.5)
=> |
=> |
=> |
=> |__10.5__
```

Variablen & fundamentale Datentypen

Variablen in Python

- Intuitiv kann man sich Variablen wie im mathematischen Kontext vorstellen:

```
x = 5  
print(x)  
5
```

- In der Programmierung kann eine Variable aber auch nicht-nummerische Werte haben:

```
x = 'Hallo, ich bin eine string variable.'  
print(x)  
'Hallo, ich bin eine string variable.'
```

- Man kann nicht nur einfache Werte, sondern auch Berechnungen (bzw. deren Ergebnisse) einer Variablen zuweisen:

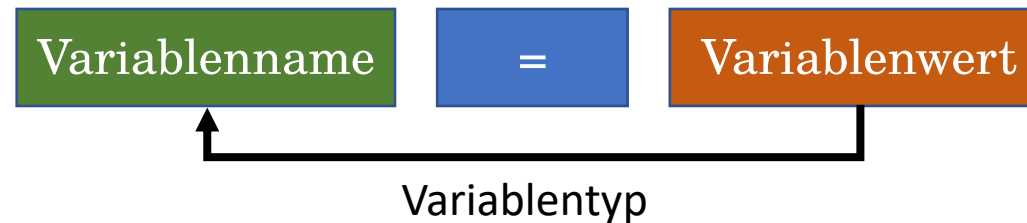
```
x = 5 + 14  
print(x)  
19
```

Variablen in Python

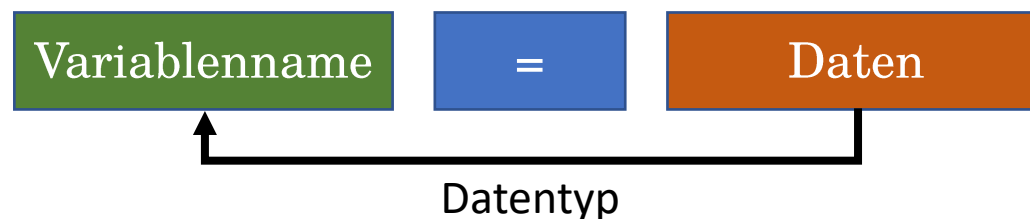
- Grundslegend haben wir einfache einen (Variablen)Namen, welchem wir einen Wert (oder Daten) zuweisen:



- Jeder Variablenwert besitzt auch einen bestimmten Variablentyp, z. B.:
 - 5 ist eine Ganzzahl (*Integer*)
 - 'Python' ist ein Text (*String*)
- Daher legt der Wert einer Variable auch deren Typ in Python fest:

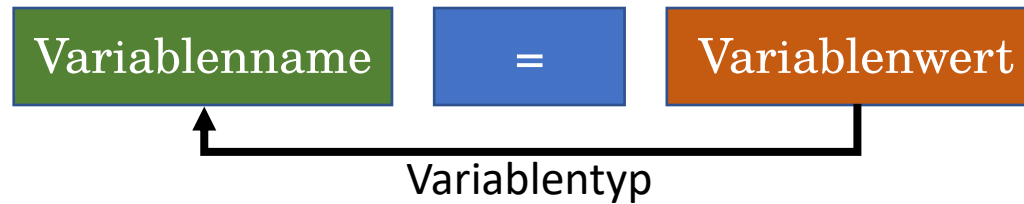


- Man kann auch sagen:

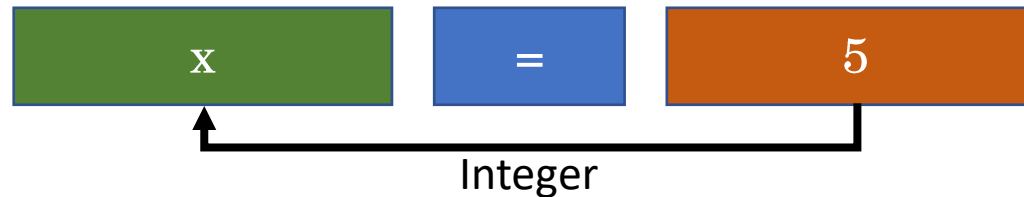


Variablen in Python

- Daher legt der Wert einer Variable auch deren Typ in Python fest:



```
x = 5  
type(x)  
int
```



- Variablen nennt man deshalb in Python auch *dynamisch Typisiert*, da sich der Typ einer Variable immer automatisch ändert, wenn ihr ein neuer Wert zugewiesen wird:

```
x = 5  
type(x)  
int  
x = 'Python'  
type(x)  
str
```

Variablenzuweisungen

- Aber was genau passiert eigentlich wenn man eine neue Variable in Python anlegt, oder ihr einen neuen Wert (und damit eventuell einen neuen Typ) zuweist?

```
x = 5  
y = 19
```



```
x = 'Python'
```



- Speicheradressen werden automatisch gelöscht, wenn kein Variablenname mehr auf sie zeigt.

Variablenzuweisungen

- Wir wissen schon das nicht nur einfache Werte, sondern auch direkt das Ergebnis einer Berechnung einer Variable zugewiesen werden kann:

```
x = 5 + 14
```

- Das funktioniert auch wenn sich Variablen innerhalb der Berechnung befinden:

```
x = x + 1  
print(x)  
20
```

- Variableninhalte lassen sich auch auf andere Variablen übertragen:

```
y = x  
print(y)  
20  
x = x + 5  
print(x)  
25  
print(y)  
20
```

```
x + 500  
print(x)  
25
```

#Achtung! Variableninhalte ändern sich nur NACH einer Zuweisung!

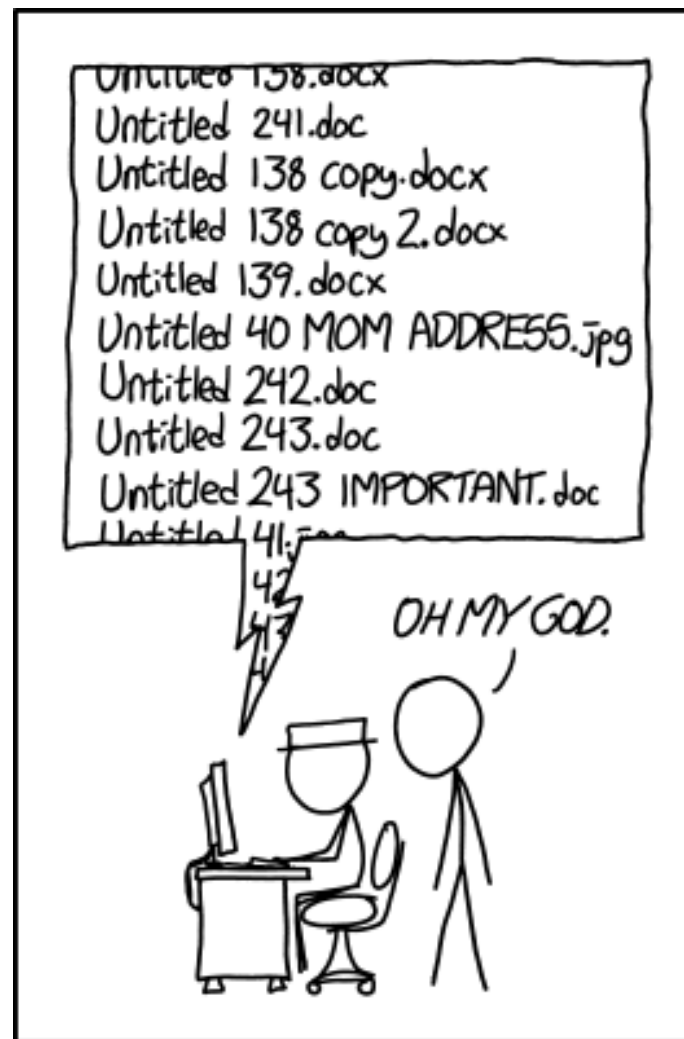
Variablen in Python

- Es lässt sich auch der gleiche Wert mehreren Variablen auf einmal zuweisen:

```
x = y = 19  
print(x)  
19  
print(y)  
19
```

- Oder mehrere Werte auf eben so viele Variablen auf einmal zuweisen:

```
x, y = 19, 'Python'  
print(x)  
19  
print(y)  
Python
```



PROTIP: NEVER LOOK IN SOMEONE
ELSE'S DOCUMENTS FOLDER.

Gute Programmierpraxis

- Variablennamen -

Gute Programmierpraxis – Variablennamen

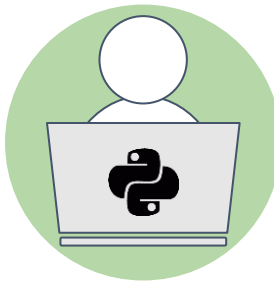
- Ein Programmcode sollte, wie etwa auch eine Gebrauchsanleitung, möglichst verständlich geschrieben werden.
- Dies hilft anderen (aber auch einem selbst!) das eigene Programm zu verstehen ohne sich großartig wieder "hineindenken" zu müssen.
- Idealerweise ist Code selbstdokumentierend, d. h. theoretisch werden für das Verständnis keine zusätzlichen Kommentare als Erläuterung gebraucht.
- Dies kann, unter Anderem, durch geschickte Wahl von Variablennamen erreicht werden, die den Inhalt und Sinn einer Variable selbst erklären.
- Namenskonventionen:

```
counter = 0

telefonBuch = {'Ida': '01234/5678910'}

wichtige_koordinaten_messung = (5.3698, 19.4234)

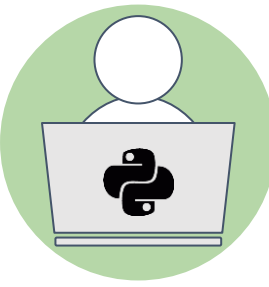
KREISZAHL_PI = 3.141592653
```



Aufgabe:

Tragt in der folgenden Tabelle die Werte der Variablen ein, die sie nach Ausführung der Anweisung in der ersten Spalte haben.

Anweisung	x	y	z
<code>x = y = 1</code>	1	1	-
<code>x = 2</code>			
<code>z = x</code>			
<code>z = z * 3</code>			
<code>x, y = y, 6</code>			
<code>y = y/2</code>			
<code>z, x = y, z</code>			
<code>x, y, z = x, x, x</code>			
<code>x = 'y'</code>			

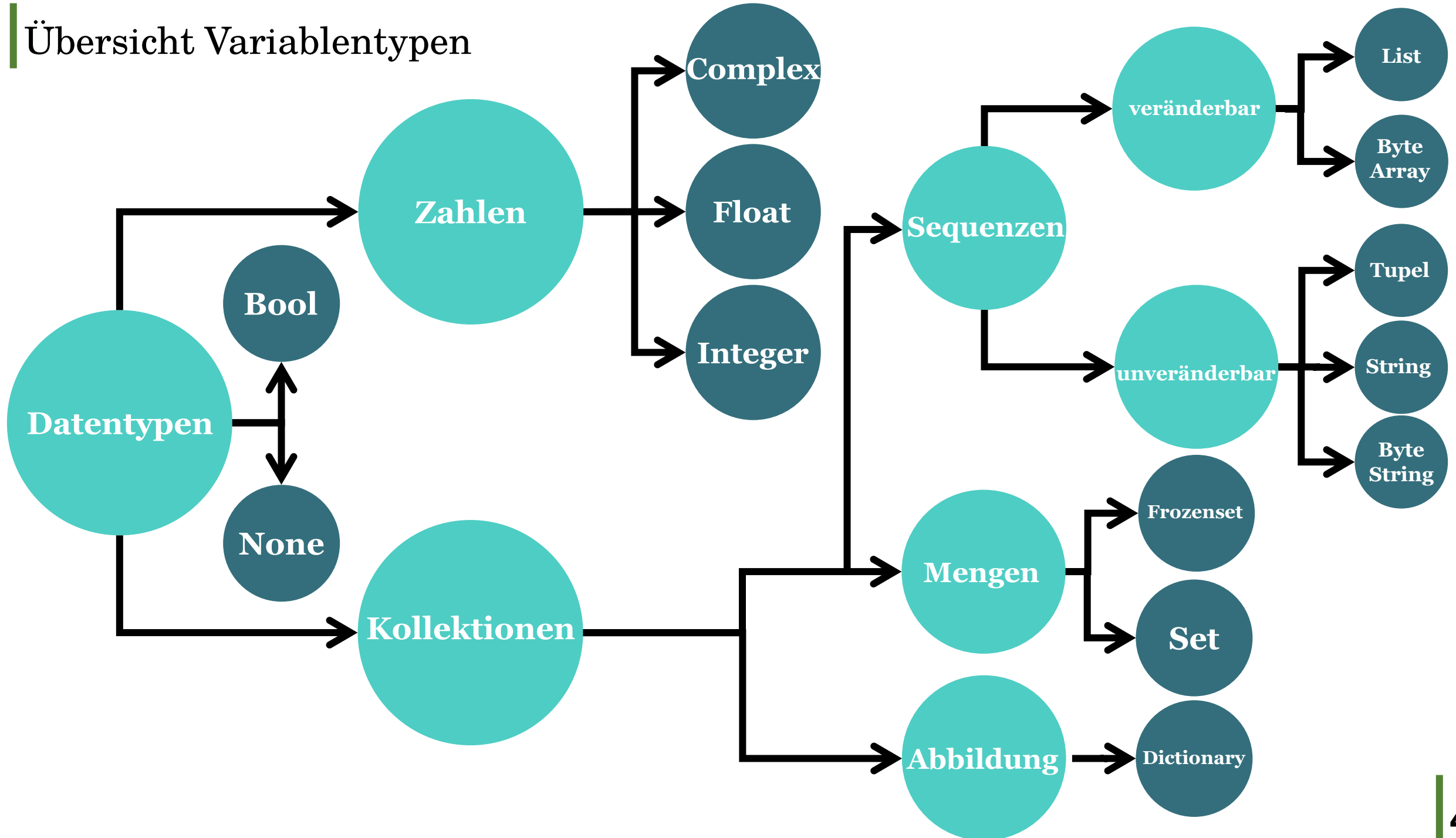


Aufgabe:

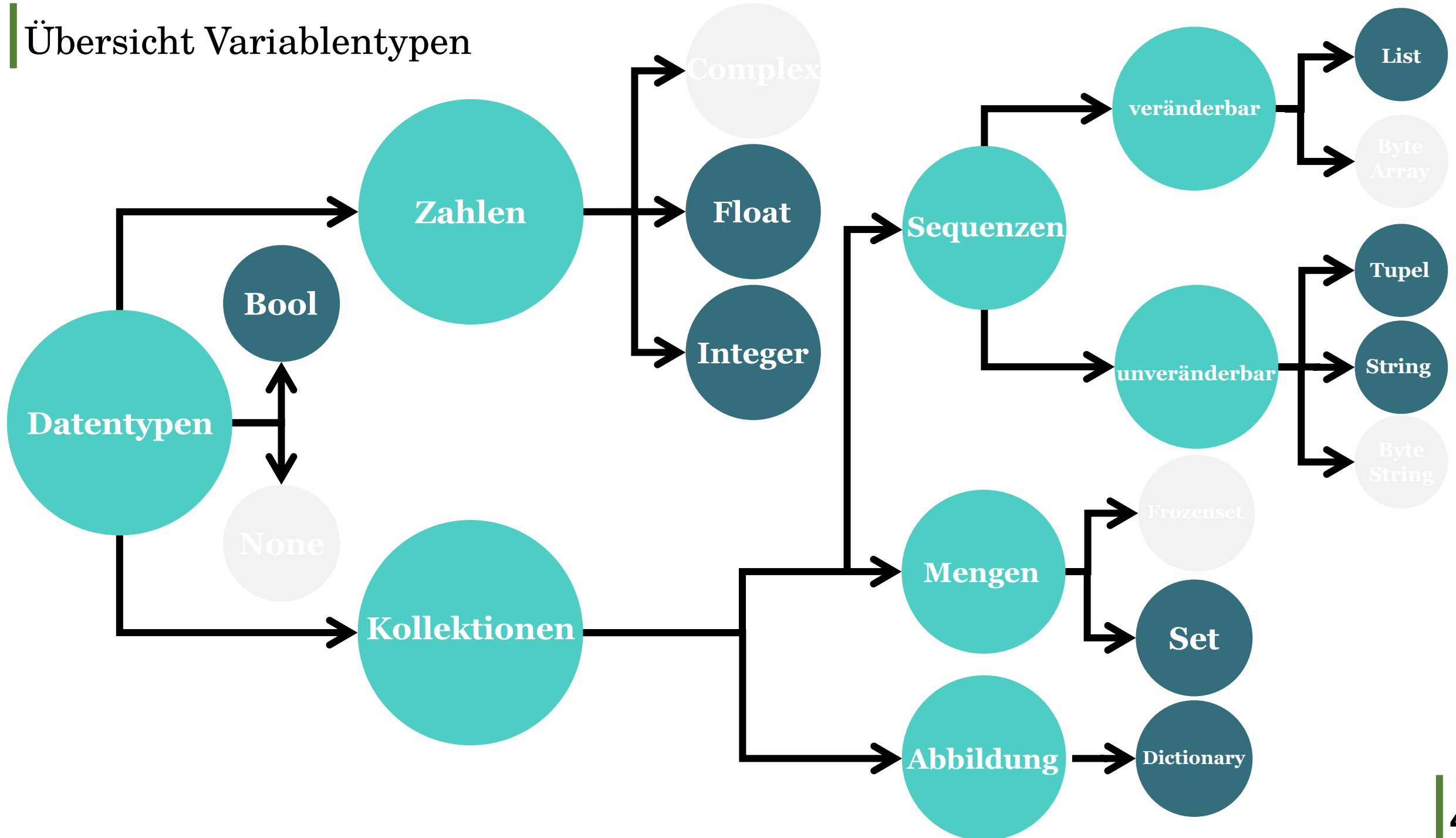
Überlegt euch, was es für grundsätzliche Variablentypen geben könnte oder eurer Meinung nach geben sollte.

Wofür werden diese Variablentypen gebraucht?

Übersicht Variablentypen



Übersicht Variablentypen



Integer

- Stellen ganze Zahlen dar
- Werden üblicherweise als Dezimalzahl geschrieben
- Ihr Datentyp lautet `int`

```
19          #Dezimalzahl, wie gewohnt

019         #führende Null nicht erlaubt
SyntaxError: invalid token

type(19)
<class 'int'>
```

Float

- Stellen Dezimalbrüche dar
- Können als Dezimalbruch oder in Exponentialschreibweise geschrieben werden
- Ihr Datentyp lautet `float`

```
19.36       #Dezimalbruch

5.0e-7      #Exponentialschreibweise

type(19.36)
<class 'float'>
```

Grundoperationen die alle Zahlentypen gemeinsam haben:

Operator	Bedeutung	Beispiel	Ergebnis
+	Positives Vorzeichen (unär)	+5	5
-	Negatives Vorzeichen (unär)	-19	-19
+	Addition	1 + 2	3
-	Subtraktion	1 - 2	-1
*	Multiplikation	2 * 2.595	5.19
/	Division	3 / 2	1.5
//	Ganzzahlige Division	3 // 2	1
%	Modulo	5 % 3	2
**	Potenz	2 ** 3	8

Kleines Beispiel für einfache Berechnungen

- Das folgende Programm zerlegt einen gegebenen Geldbetrag in Scheine und Münzen:

```
geld = input("Geldbetrag in Euro: ")
geld = int(geld)

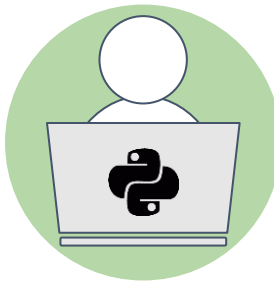
zwanziger = geld // 20
geld = geld % 20

zehner = geld // 10
geld = geld % 10

fuenfer = geld // 5
geld = geld % 5

zweier = geld // 2
einer = geld % 2

print("Betrag setzt sich wie folgt zusammen:")
print(zwanziger, "mal 20 Euro")
print(zehner, "mal 10 Euro")
print(fuenfer, "mal 5 Euro")
print(zweier, "mal 2 Euro")
print(einer, "mal 1 Euro")
```



Aufgabe:

Ladet euch das Programm zur Zerlegung eines Geldbetrags aus dem Übungsmaterialordner (Datei mit der Nummer 11) und passt es so an, dass ein gegebener Geldbetrag zusätzlich auch in 100er und 50er Euroscheine, aber nicht mehr in 20er und 5er Euroscheine zerlegt werden kann.

Testet im Anschluss euer angepasstes Programm.



Bool

- Wahrheitswerte kommen aus dem Bereich der Aussagenlogik und können nur entweder `True` oder `False` sein
- Sie spielen bei bedingten Anweisungen eine wichtige Rolle (dazu morgen mehr)
- Ihr Datentyp lautet `bool`

```
x = True  
type(x)  
<class 'bool'>
```

```
19 < 15           #Vergleiche können wahr oder falsch sein  
False
```

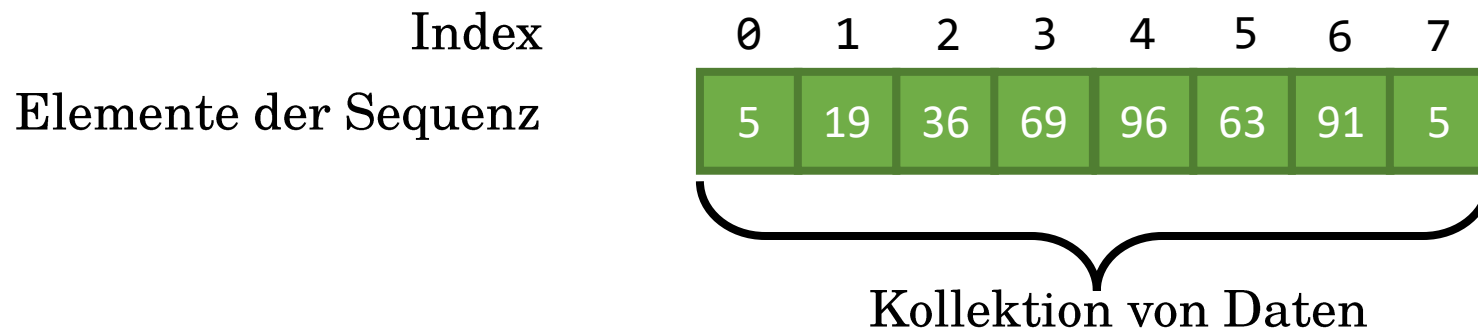
Sequenzen in Python

String

Tupel

List

- Alle Sequenztypen (String, List, Tuple) stellen Kollektionen von Daten dar.
- Jeder dieser Datentypen eignet sich für die Speicherung und Verarbeitung ganz bestimmter Daten.
- Sie bestehen aus einzelnen Elementen, welche beginnend bei 0 durchnummeriert sind (man sagt auch indexiert).
- Über den Index kann man auf ein einzelnes Element einer Sequenz zugreifen.



- Strings sind Folgen von Zeichen aus einem Alphabet und werden entweder durch einfache oder doppelte Anführungszeichen markiert.

```
'Das Leben des Brian'
"Die Ritter der Kokosnuss"

"Flyin' Circus"           #Gültiger String
'Flyin' Circus'           #Ungültiger String
SyntaxError: invalid syntax
```

- Strings dienen zur Speicherung und Verarbeitung von Texten.
- Strings stellen einen besonders wichtigen Datentyp dar, da die meisten Daten und Dateiformat in Form von Text auf Computern gespeichert werden (dazu am Mittwoch mehr).

	Index									
	0	1	2	3	4	5	6	7	8	9
Elemente des Strings 'Python 3.9'	'P'	'y'	't'	'h'	'o'	'n'	' '	'3'	'.'	'9'

- In Python kann ein String alle möglichen Zeichen enthalten, allerdings ist der Backslash (\) eine Ausnahme → Man verwendet ihn zur Markierung von Sonderzeichen z. B. einem Zeilenumbruch.

Escape-Sequenz	Erklärung	Beispiel
\n	Zeilenumbruch	"eins\nzwei" eins zwei
\t	Tabstopp	"eins\tzwei" eins zwei
\\	Backslash in einem String	"eins\\zwei" eins\zwei

- In einem Tupel sind mehrere Daten mit eventuell unterschiedlichem Datentypen zusammengefasst.
- Tupel werden zur Darstellung bzw. Speicherung komplexer Einzelobjekte benutzt, wie etwa:
 - Name und Geburtsdatum einer Person
 - Beschreibung eines Punktes in einem 3-dimensionalen Raum mittels X-, Y- und Z-Koordinaten
 - Adresse als Tupel mit fünf Elementen (Name, Straße, Hausnummer, Postleitzahl, Stadt)
 - Beschreibung einer experimentellen Messung (Messgerät, Datum, Probe, Messwert)
- In Python besteht ein Tupel aus einer Folge von Daten die mit Komma getrennt und von runden Klammern eingeschlossen sind:

```
(1, 2, 3)
```

```
person = ('Maximilia', 'Musterfrau', '01.01.1990')
```

```
adresse = ('Berlin', 12109, 'Musterstr.', 1, person)
```

```
print(adresse)
```

```
('Berlin', 12109, 'Musterstr.', 1, ('Maximilia', 'Musterfrau', '01.01.1990'))
```

```
person = ('Maximilia', 'Musterfrau', '01.01.1990')  
  
adresse = ('Berlin', 12109, 'Musterstr.', 1, person)  
print(adresse)  
('Berlin', 12109, 'Musterstr.', 1, ('Maximilia', 'Musterfrau', '01.01.1990'))
```

Elemente des Tupels *adresse*

Index	0	1	2	3	4
	'Berlin'	12109	'Musterstr.'	1	('Maximilia', 'Musterfrau', '01.01.1990')

- Auch in einer Liste können Daten unterschiedlichen Typs gespeichert werden.
- Listen werden zur Speicherung und Verarbeitung von vielen Einzelobjekten benutzt, wobei sich der Inhalt der Liste ständig ändern kann, wie z. B.:
 - Bestsellerliste eines Buchladens
 - Abonnentenliste eines Youtube-Kanals
 - Liste mit Messungen eines laufenden Experiments
- In Python besteht eine Liste aus einer Folge von Daten die mit Komma getrennt und von eckigen Klammern eingeschlossen sind:

```
[1, 2, 3, 4, 5, 'Python', ('A', 'B')]  
  
farben = ['grün', 'gelb', 'blau', 'rot', 'orange', 'mauve']  
  
messung1 = ('Laser', '06.07.22', 13.587)  
messung2 = ('Laser', '07.07.22', 109.433)  
messwerte = [messung1, messung2]  
print(messwerte)  
[('Laser', '06.07.22', 13.587), ('Laser', '07.07.22', 109.433)]
```

- Eine Liste kann auch andere Listen als Elemente enthalten, man spricht dann von verschachtelten Listen:

```
schachtel = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

Elemente der Liste <i>schachtel</i>			
Index	0	1	2
	[1, 2, 3]	[4, 5, 6]	[7, 8, 9]

Zugriff auf Elemente einer Sequenz

- Um auf ein einzelnes Element einer Sequenz (String, Tupel, List) zugreifen zu können, schreibt man in eckigen Klammern hinter den Namen der Sequenzvariable den Index des Elements:

```
name = 'Maximilia Musterfrau'
koordinaten = (4.5, 7.3, 9.1)
farben = ['grün', 'gelb', 'blau', 'rot', 'orange', 'mauve']

print(name[10])
M

print(koordinaten[2])
9.1

print(farben[0])
grün
```

- Handelt es sich bei dem gewählten Element einer Sequenz selbst wieder um eine Sequenz, so kann man mit der selben Weise auf dessen Elemente zugreifen:

```
schachtel = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
print(schachtel[1][1])
5
```

Sequenzen Grundoperationen

- Alle Sequenzen (String, Tupel, List) haben eine Reihe von Grundoperationen gemeinsam, z. B.:

Operation	Ergebnis
<code>s[i]</code>	Wiedergeben des i-ten Elementes von s.
<code>s[i:j]</code>	Wiedergabe des Ausschnitts (slice) von s, der vom i-ten bis zum j-ten Element (nicht einschließlich) geht.
<code>s + t</code>	Konkatenation der beiden Sequenzen s und t.
<code>s * n</code>	n Kopien der Sequenz s werden hintereinander gehängt.
<code>x in s</code>	Gibt True wieder wenn ein Element mit dem Wert x in der Sequenz s enthalten ist, sonst False .
<code>x not in s</code>	Gibt True wieder wenn <u>kein</u> Element mit dem Wert x in der Sequenz s enthalten ist, sonst False .
<code>len(s)</code>	Gibt die Länge der Sequenz s wieder, d. h. die Anzahl der enthaltenen Elemente.
<code>min(s)</code>	Gibt das kleinste Element der Sequenz s wieder.
<code>max(s)</code>	Gibt das größte Element der Sequenz s wieder.

Sequenzen Grundoperationen – Beispiele

#Konkatenation funktioniert nur für Sequenzen gleichen Typs

```
[5, 19, 36, 69, 119] + [149]
```

```
[5, 19, 36, 69, 119, 149]
```

```
schlangen = ('Python' , 'Cobra')
```

```
nichtSchlangen = ('Perl' , 'Ruby')
```

```
schlangen + nichtSchlangen
```

```
('Python' , 'Cobra', 'Perl' , 'Ruby')
```

#Fehler bei Konkatenation von Sequenzen ungleichen Typs

```
'Perl ist toller als ' + ['Python']
```

```
TypeError: can only concatenate string (not "list") to string
```

#Alle Sequenzen lassen sich mit * vervielfältigen

```
[1, 2, 3] * 4
```

```
[1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3]
```

#Bestimmen der Länge einer Sequenz mit der len Funktion

```
liste = [[ 1, 2, 3, 4]]
```

```
len(liste)
```

```
1
```

```
len(liste[0])
```

```
4
```

Sequenzen Grundoperationen – Beispiele

```
cooleSkriptsprachen = ['Python' , 'Ruby' , 'Javascript' , 'julia']
'Perl' in cooleSkriptsprachen
False
'Perl' not in cooleSkriptsprachen
True

liste = [1, 2, 3, 4, 5, 6, 7]
liste[3:6]           #Slicing [x:y] bedeutet Teilsequenz von einschließlich)
[4,5,6]              #x-ten Element bis (nicht einschließlich) y-ten Element

min(liste)           #Kleinstes Element der Sequenz
1
max(liste)           #Größtes Element der Sequenz
7
```


Sequenzen Grundoperationen – Slicing

```
s = 'Das Leben des Brian'
```

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
s =	'D'	'a'	's'	' '	'L'	'e'	'b'	'e'	'n'	' '	'd'	'e'	's'	' '	'B'	'r'	'i'	'a'	'n'
	-19	-18	-17	-16	-15	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

	0	1	2	3	4
s[4:9]	'L'	'e'	'b'	'e'	'n'

	0	1	2
s[0:3] s[:3]	'D'	'a'	's'

	0	1	2	3	4
s[14:19] s[14:]	'B'	'r'	'i'	'a'	'n'

	0	1	2
s[-9:-6]	'd'	'e'	's'

	0
s[-7:-10]	' '

	0
s[-5:3]	' '



Aufgabe:

1. Ermittelt mit Hilfe der interaktiven Pythonshell, welche der folgenden Zeichenfolgen gültige Pythonwerte sind und korrigiert die ungültigen Zeichenfolgen.
2. Nutzt die `type()` Funktion um den Datentyp der Zeichenfolgen zu bestimmen.

- 3
- 3.0
- 3.5
- 0,3
- .4
- 1.000e-2
- Python
- 'Python'
- "Python"
- (1; 2; 3)
- 10%7
- "2" + "3"
- (1, 2, 3)[3]
- [5, abc, 7]
- [1, 2] + [(3, 4)]
- (1, 2) + 3
- 2 – 1.0
- 6/3
- 5//2
- 4 ** (1.0//2)
- 3 * 'Hoch! '
- "10" / "5"
- ((1, 2, 3), 4, 5))
- 1aText = 'Toller Text'
- 5 – 7 + 3 = ergebnis
- [19, 5, 36, 149][:]

Listen versus Tupel in Python

- Aber was ist der Unterschied zwischen Listen und Tupeln?
→ Listen kann man verändern und Tupel nicht.

```
zahlen = [1, 2, 3, 9]
```

```
zahlen[3] = 4
```

```
print(zahlen)
```

```
[1, 2, 3, 4]
```

```
koordinaten = (1, 2, 3, 9)
```

```
koordinaten[3] = 4
```

```
TypeError: 'tuple' object does not support item assignment
```

```
zahlen.append(5)
```

```
print(zahlen)
```

```
[1, 2, 3, 4, 5]
```

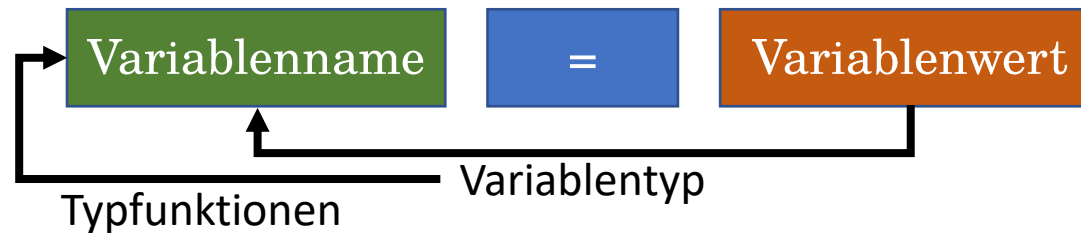
```
koordinaten.append(5)
```

```
AttributeError: 'tuple' object has no attribute 'append'
```

Typfunktionen

```
zahlen = [1, 2, 3, 4]
zahlen.append(5)
print(zahlen)
[1, 2, 3, 4, 5]
```

- Wir haben schon gelernt, dass der Wert einer Variable auch den Datentyp der Variable bestimmt.
- Zusätzlich verleitet dieser Datentyp einer Variable auch bestimmte Funktionen um die gespeicherten Variablenwerte zu verarbeiten oder zu verändern.
- Diese Funktionen werden auch *Typfunktionen* genannt.



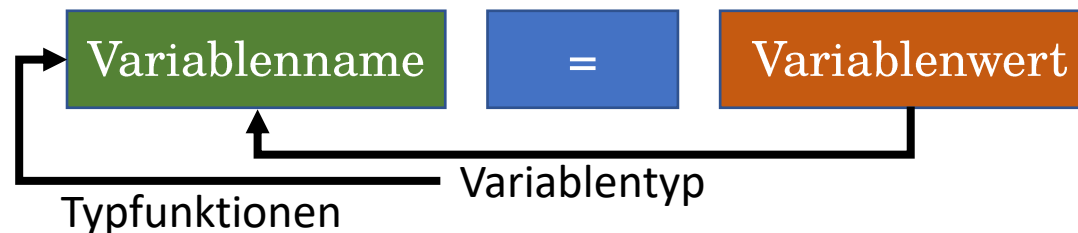
Typfunktionen

- Jeder Variablentyp hat eigene Typfunktionen, welche zur Verarbeitung seiner Variablenwerte nützlich sind, für andere Variablentypen aber nicht sinnvoll wären.
- Ändert sich der Typ einer existierenden Variable, weil sich ihr Variablenwert verändert, dann ändern sich auch automatisch die Typfunktionen, welche die Variable kennt.

```
zahlen = [1, 2, 3, 4]
zahlen = zahlen - 5
TypeError: unsupported operand type(s) for -: 'list' and 'int'

zahlen = 19
zahlen = zahlen - 5

zahlen.append(5)
AttributeError: 'int' object has no attribute 'append'
```



Typfunktionen von Listen

- Hier eine Auswahl von häufig genutzten Typfunktionen von Listen:

Operation	Ergebnis
<code>s.append(x)</code>	An die Liste <code>s</code> wird ein neues Element <code>x</code> angehängt.
<code>s.extend(t)</code>	Die Liste <code>s</code> wird um die Element der Sequenz <code>t</code> verlängert.
<code>s.count(x)</code>	Gibt die Anzahl der Listenelemente mit dem Wert <code>x</code> zurück.
<code>del s[i]</code>	Das Element mit Index <code>i</code> wird aus der Liste <code>s</code> entfernt, damit verringert sich auch die Länge der Liste um 1.
<code>s.index(x)</code>	Zurückgegeben wird der kleinste Index von <code>s</code> an dem ein Element gleich <code>x</code> ist.
<code>s.insert(i, x)</code>	Ein neues Element <code>x</code> wird vor dem Index <code>i</code> in die Liste <code>s</code> eingefügt.
<code>s.pop()</code>	Das letzte Element von <code>s</code> wird aus der Liste entfernt und wiedergegeben.
<code>s.remove(x)</code>	Das erste Element gleich <code>x</code> wird aus der Liste <code>s</code> entfernt.
<code>s.sort()</code>	Die Elemente der Liste werden aufsteigend sortiert.
<code>s.reverse()</code>	Die Elemente der Liste werden absteigend sortiert.

Typfunktionen von Listen – Beispiele

```
l = [1, 2, 3, 4]

l[2] = 'neu!'
#[1, 2, 'neu!', 4]

l.insert(1,100)
#[1, 100, 2, 'neu!', 4]

del l[1:3]
#[1, 'neu!', 4]

l.append('neu!')
#[1, 'neu!', 4, 'neu!']

l.remove(1)
#[1, 4, 'neu!']

l.pop()
neu!
#[1, 4]
```

Typfunktionen von Listen – Beispiele

```
l = [1]

l = l + [2, 3]
#[1, 2, 3]

l.extend((1,2))
#[1, 2, 3, 1, 2]

l.extend('123')
#[1, 2, 3, 1, 2, '1', '2', '3']

l.append('123')
#[1, 2, 3, 1, 2, '1', '2', '3', '123']

l = [1, 2, 3, 1, 2]
l.sort()
#[1, 1, 2, 2, 3]

l.reverse()
#[3, 2, 2, 1, 1]
```




Aufgabe:

Schreibt zu den folgenden umgangssprachlichen Beschreibungen passende Python-Anweisungen auf:

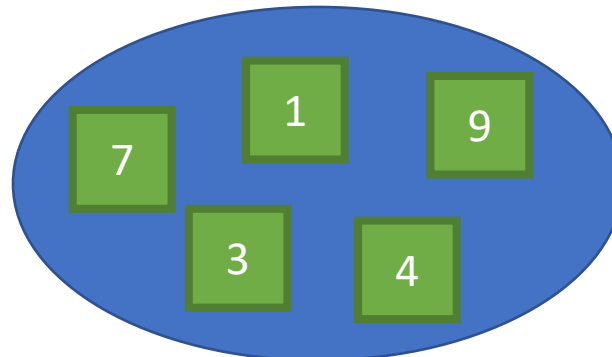
Beschreibung	Python Anweisung
Der String "Sigrid" erhält den Namen <code>person</code> .	<code>person = 'Sigrid'</code>
Der Variable <code>zahl</code> wird der Wert 10 zugewiesen.	
Der Inhalt der Variable <code>zahl</code> wird um 5 erhöht.	
Der Inhalt der Variable <code>zahl</code> wird auf dem Bildschirm ausgegeben.	
Der Inhalt der Variable <code>x</code> wird mit dem Inhalt der Variable <code>y</code> multipliziert und das Ergebnis der Variable <code>produkt</code> zugewiesen.	
Den Werten <code>["rot", "gelb", "grün"]</code> werden der Name <code>farben</code> zugewiesen. Dann wird der Variable <code>farben</code> gesagt, es soll die Reihenfolge seiner Listenelemente umkehren. Anschließend sollen die geänderten Werte auf dem Bildschirm ausgegeben werden.	

Set

- Mengen stellen auch Kollektionen von Daten dar, allerdings im Vergleich zu Listen oder Tupeln mit besonderen Eigenschaften:
 - Jedes Element in einer Menge kann nur genau **einmal** auftauchen.
 - Die Elemente einer Menge haben keine feste Reihenfolge und damit auch keinen Index um auf ein bestimmtes Element der Menge zugreifen zu können.
- In Python besteht eine Menge aus einer Folge von Daten die mit Komma getrennt und von geschweiften Klammern eingeschlossen sind:

```
menge = {4, 7, 3, 9, 1}  
print(menge)  
{1, 3, 4, 7, 9}
```

Elemente der Menge





Set

- Mit Hilfe der `set()` Funktion lassen sich Strings, Tupel oder Listen in Mengen umwandeln:

```
set([19,19,5,90,36])  
{5, 19, 36, 90}  
  
set('Mississippi')  
{ 'M', 'i', 'p', 's' }
```

- Die Typfunktionen von Mengen orientieren sich an denen in der Mathematik üblichen Mengenoperationen:
 - Vereinigung
 - Durchschnitt
 - Differenz
 - symmetrische Differenz
 - Teilmenge

Typfunktionen von Mengen

- Für zwei Mengen (hier: $s1$ und $s2$) existieren folgende Typfunktionen:

Operation	Kurzform	Bedeutung
<code>s1.union(s2)</code>	$s1 \mid s2$	Alle Elemente von $s1$ und $s2$.
<code>s1.intersection(s2)</code>	$s1 \& s2$	Alle gemeinsamen Elemente von $s1$ und $s2$.
<code>s1.difference(s2)</code>	$s1 - s2$	Alle Elemente von $s1$ die nicht auch in $s2$ sind.
<code>s1.symmetric_difference(s2)</code>	$s1 \wedge s2$	Alle Elemente in $s1$ oder $s2$, aber nicht in beiden.
<code>s1.issubset(s2)</code>	$s1 \leq s2$	Liefert TRUE wenn $s1$ Teilmenge von $s2$ ist.
<code>s1.isuperset(s2)</code>	$s1 \geq s2$	Liefert TRUE wenn $s1$ Obermenge von $s2$ ist.
<code>s1.copy()</code>		Liefert eine Kopie von $s1$
<code>x in s1</code>		Liefert TRUE, falls das Element x in $s1$ vorkommt.
<code>x not in s1</code>		Liefert TRUE, falls das Element x nicht in $s1$ vorkommt.

Typfunktionen von Mengen – Beispiele

```
s1 = {1,2,3,4,5}
s2 = {4,5,6,7,8}
```

```
s1.union(s2)           #Vereinigung
{1,2,3,4,5,6,7,8}
```

```
s1.intersection(s2)    #Schnitt
{4,5}
```

```
s1.symmetric_difference(s2) #Symmetrische Differenz
{1,2,3,6,7,8}
```

```
{1,2,3}.issubset(s1)    #Untermenge
True
```

```
1 in s1                 #Element enthalten in Menge
True
```

```
'1' in s1
False
```

```
'1' not in s1
True
```



- Ein Dictionary stellt eine Kollektionen von Daten dar, deren einzelne Elemente so genannte **Schlüssel-Wert-Paare** sind.
- Am einfachsten kann man sich diesen Datentyp als Wörterbuch vorstellen (daher der Name), denn zu jedem Schlüssel (jeder Vokabel) gibt es genau einen Wert (eine Übersetzung).
- In Python besteht ein Dictionary aus einer Folge von Datenpaaren die mit Komma getrennt und von geschweiften Klammern eingeschlossen sind, wobei die Datenpaare mit Doppelpunkt getrennt sind:

```
d = {'Eins': 1, 'Zwei': 2, 'Drei': 3, 'Liste': [1,2,3], 'Monty': 'Python'}
```

- Auf die Elemente eines Dictionary greift man nun **nicht** per Index, sondern mit Hilfe des Schlüssels zu.

```
d = {'Eins': 1, 'Zwei': 2, 'Drei': 3, 'Liste': [1,2,3], 'Monty': 'Python'}
d['Eins']
1

d['Liste']
[1,2,3]

d['Vier']
KeyError: 'Vier'

d['Vier'] = 4
print(d)
{'Drei': 3, 'Monty': 'Python', 'Liste': [1, 2, 3], 'Vier': 4, 'Zwei': 2, 'Eins': 1}

d['Vier'] = 19
print(d)
{'Drei': 3, 'Monty': 'Python', 'Liste': [1, 2, 3], 'Vier': 19, 'Zwei': 2, 'Eins': 1}
```

Typfunktionen von Dictionarys

- Auch für Dictionarys existieren eine ganze Reihe von Typfunktionen wie:

Operation	Ergebnis
<code>d[k]</code>	Zurückgeben des Wertes mit dem Schlüssel <code>k</code> .
<code>d[k] = x</code>	Dem Schlüssel <code>k</code> wird der Wert <code>x</code> zugewiesen.
<code>d.clear()</code>	Alle Elemente werden aus <code>d</code> entfernt. Zurück bleibt ein leeres Dictionary <code>{}</code> .
<code>d.copy()</code>	Zurückgegeben wird eine Kopie von <code>d</code> .
<code>del d[k]</code>	Das Element mit Schlüssel <code>k</code> wird gelöscht.
<code>d.get(k, x)</code>	Zurückgegeben wird <code>d[k]</code> , falls <code>k</code> in <code>d</code> , sonst <code>x</code> .
<code>k in d</code>	Liefert <code>TRUE</code> , falls <code>d</code> einen Schlüssel <code>k</code> enthält, sonst <code>FALSE</code> .
<code>k not in d</code>	Liefert <code>FALSE</code> , falls <code>d</code> einen Schlüssel <code>k</code> enthält, sonst <code>True</code> .
<code>d.keys()</code>	Liefert ein Objekt das eine Liste der Schlüssel von <code>d</code> enthält.
<code>d.values()</code>	Liefert ein Objekt das eine Liste der Werte von <code>d</code> enthält.
<code>d1.update(d2)</code>	Für alle Schlüssel <code>k</code> im Dictionary <code>d2</code> wird im Dictionary <code>d1</code> ein neues Element <code>d1[k]</code> eingefügt.

Typfunktionen von Dictionarys – Beispiele

```
d = {'Eins': 1, 'Zwei': 2, 'Drei': 3, 'Liste': [1,2,3], 'Monty': 'Python'}
```

```
'Monty' in d  
True
```

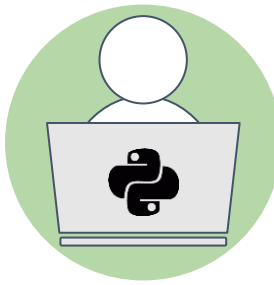
```
del d['Monty']  
print(d)  
d = {'Eins': 1, 'Zwei': 2, 'Drei': 3, 'Liste': [1,2,3]}
```

```
'Monty' in d  
False  
'Monty' not in d  
True
```

```
liste = d['Liste']  
print(liste)  
[1,2,3]
```

```
liste2 = d['Liste2']  
KeyError: 'Liste2'
```

```
liste2 = d.get('Liste2', [])  
print(liste2)  
[]
```

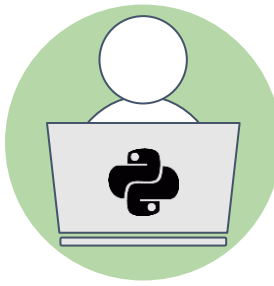


Aufgabe:

In einem Programm müssen konkrete oder gedankliche Objekte aus der Wirklichkeit durch Variablen eines geeigneten Typs abgebildet werden. Gebt in der folgenden Tabelle zu jedem Objekt der Wirklichkeit einen passenden Python-Datentyp an.

Manchmal sind auch mehrere Datentypen geeignet.

Wirklichkeit	Datentyp	Beispiel
Durchmesser von Atomen	Gleitkommazahl (Float)	0.000000000000123
Pflanzennamen		
Namen der Teilnehmer eines Wettlaufs		
Name, Vorname und Alter einer Person		
Name, Vorname und Alter der Personen eines Wettlaufs		
Spielstand in einem Fußballspiel z. B. (1:0)		
Tabelle, in denen chemische Elementsymbole (z. B. H) sowie deren englischen und deutschen Namen vermerkt sind		



Aufgabe:

Erstellt ein kleines Programm, welches vom Benutzer zwei Zahlen zwischen 0 und 100 verlangt, aus diesen die Summe bildet und das Ergebnis auf dem Bildschirm zurückgibt. Verwendet für die Benutzereingabe die `input()` Funktion. Entspricht das Ergebnis euren Erwartungen?

Hinweis:

Bevor ihr anfangt Code zu schreiben: Zerlegt die Aufgabe zuerst möglichst kleinschrittig in die einzelnen Teilaufgaben, die vom Programm erledigt werden müssen.

```
[1]> 2+"2"  
=> "4"  
[2]> "2"+[1]  
=> "[2]"  
[3]> (2/0)  
=> NaN  
[4]> (2/0)+2  
=> NAP  
[5]> ""+" "  
=> " + "  
[6]> [1,2,3]+2  
=> FALSE  
[7]> [1,2,3]+4  
=> TRUE  
[8]> 2/(2-(3/2+1/2))  
=> NaN.00000000000000013  
[9]> RANGE(" ")  
=> (" ", " ", " ", " ", " ", " ")  
[10]> + 2  
=> 12  
[11]> 2+2  
=> DONE  
[14]> RANGE(1, 5)  
=> (1, 4, 3, 4, 5)  
[13]> FLOOR(10.5)  
=> |  
=> |  
=> |  
=> |__10.5__
```

Typumwandlung

Code

```
#input() liefert eine Zeichenkette die vom Benutzer eingegeben wurde  
x = input('Bitte x eingeben: ')  
y = input('Bitte y eingeben: ')  
summe = x + y  
print('Summe: ', summe)
```

Console

```
Bitte x eingeben: 2  
Bitte y eingeben: 3  
Summe: '23'
```

Typumwandlung

- Da Variablen **dynamisch** Typisiert sind, werden Ausdrücke **stark** Typisiert, d. h. bei Uneindeutigkeiten gibt der Pythoninterpreter einen Fehler aus, weil er nicht wissen kann, welchen Datentyp ihr als Ergebnis erwartet.
→ *"Explicit is better than implicit."*

```
5 + '14'  
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

- Durch Typumwandlungsfunktionen (auch Casting Funktionen genannt) lassen sich Variablentypen ineinander umwandeln:
`int()`, `float()`, `bool()`, `set()`, `tuple()`, `list()`

```
5 + int('14')  
19  
  
str(5) + '14'  
514
```

Typumwandlung – Beispiel

```
int(1.7)           #Gleitkommazahlen werden immer abgerundet
1

int('5193669119149')
5193669119149

                        #String zu Integer Umwandlungen dürfen nur aus Ziffern bestehen
int('3.1415')
ValueError: invalid literal for int() with base 10: '3.1415'

str(3.1415)         #Umwandlung einer Zahl in einen String
'3.1415'

float('3.1415')
3.1415
```

Typumwandlung – Beispiel

```
float(12)  
12.0
```

```
bool(1)  
True
```

```
bool('ABC')  
True
```

```
bool([])  
False
```

```
tuple([1, 2, 3])  
(1, 2, 3)
```

```
d = {'a': 1, 'b': 2, 'c': 3}  
list(d)  
['a', 'b', 'c']
```

```
dict([('e', 19), ('f', 36)])  
{'e': 19, 'f': 36}
```


Typumwandlung – Beispiel

- Aber nicht jeder Datentyp lässt sich in jeden beliebigen anderen Datentyp umwandeln:

```
int({'a': 1, 'b': 2})  
TypeError: int() argument must be a string, a bytes-like object or a number, not 'dict'  
  
float({1})  
TypeError: float() argument must be a string or a number, not 'set'
```

- Typumwandlungsfunktionen lassen sich auch ineinander schachteln:

```
str(int('4')+1)  
5
```

- Jeder Datentyp lässt sich in einen String umwandeln. Man erhält entweder eine Stringrepräsentation der Daten, die Speicheradresse der Variable:

```
def f():  
    pass  
  
str(f)  
<function f at 0x7fb3cd836700>
```

Code

```
#input() liefert eine Zeichenkette die vom Benutzer eingegeben wurde
x = input('Bitte x eingeben: ')
y = input('Bitte y eingeben: ')
summe = int(x) + int(y)          #explizite Typumwandlung durch die int()-Funktionen
print('Die Summe von ', x, ' und ', y, ' ist ', summe)
```

Console

```
Bitte x eingeben: 2
Bitte y eingeben: 3
Die Summe von 2 und 3 ist 5
```

- Die `print` Funktion wandelt jede Eingabe die kein String ist, automatisch in einen String um.

- Du kennst den Unterschied zwischen Programmieren und algorithmisch Denken.
- Du hast einen ersten Überblicks über die Programmiersprache Python (Historie, Eigenschaften, Vor- und Nachteile).
- Du kannst Python installieren und erste einfache Programme erstellen und diese über die Kommandozeile oder die IDLE ausführen.
- Du hast eine Vorstellung davon wie Variablen in Python funktionieren und kannst den verschiedenen Datentypen ihrem Nutzen zuordnen.
- Du kannst erklären warum Datentypumwandlung nötig sein kann und wie man diese in Python anwendet.



Aufgabe:

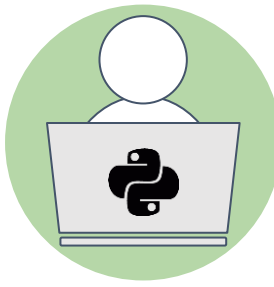
Schreibt ein Programm, das die Kosten für eine Urlaubsreise für eine Reisegruppe mit Bus berechnet. Vom Benutzer werden folgende Angaben erfragt:

- Anzahl der Personen
- Hotelkosten pro Person
- Gesamtkosten für den Reisebus
- sonstige Gesamtkosten.

Das Programm gibt eine kleine Übersicht der Gesamturlaubskosten aus sowie den Betrag den jeder einzelne Reisende zahlen muss.

Hinweis:

Bevor ihr anfangt Code zu schreiben: Zerlegt die Aufgabe zuerst möglichst kleinschrittig in die einzelnen Teilaufgaben die vom Programm erledigt werden müssen.



Aufgabe:

Eine ISBN (International Standard Book Number) besteht aus zehn Ziffern:

$z_1 z_2 z_3 z_4 z_5 z_6 z_7 z_8 z_9 z_{10}$

Die letzte Ziffer z_{10} ist eine Prüfziffer, die sich wie folgt berechnet:

$$s = 1*z_1 + 2*z_2 + 3*z_3 + 4*z_4 + 5*z_5 + 6*z_6 + 7*z_7 + 8*z_8 + 9*z_9$$

Die Prüfziffer z_{10} ist der Rest der ganzzahligen Division von s durch 11.

Schreibt ein Programm, welches die Prüfziffer einer ISBN berechnet. Einggegeben wird eine natürliche neunstellige Zahl und ausgegeben wird die Prüfziffer als Zahl zwischen 0 und 10.

Hinweis:

Bevor ihr anfangt Code zu schreiben: Zerlegt die Aufgabe zuerst möglichst kleinschrittig in die einzelnen Teilaufgaben die vom Programm erledigt werden müssen.