FRIEDRICH-SCHILLER-
UNIVERSITÄT
JENA

Einführung in Python

1. Vorlesung

Allgemeines und Fundamentale Datentypen

- Vorlesung normal 90 min und Übung flexibel
- Übungsaufgaben werden nicht benotet, aber eine regelmäßige Teilnahme an den Übungen ist Prüfungsvorrausetzung
- Keine Klausur, aber ein kleines Abschlussprojekt

Vorlesungsinhalte

1. Allgemeines und Fundamentale Datentypen
2. Kontrollstrukturen und Funktionen
3. Spezielle Funktionen und Datei Ein-/Ausgabe
4. Dateiformate, Stringoperationen und reguläre Ausdrücke
5. Objektorientiere Programmierung: Eigene Klassen und Module
6. OOP Modellierung und GUI-Programmierung
7. Threads und Multiprocessing
8. NumPy, SciPy und Matplotlib
9. Webprogrammierung
10. Ipython, juPyter und Sonstiges
11. Biopython
12. Testen und Tuning
13. Besprechung des Abschlussprojekts

- Mögliche weitere Themen:
  - Jpython, PyPy
  - Datenbanken
  - …
- Guter Programmierstil
  - Kommentare
  - Bezeichnungen
  - Struktur
  - Häufige Fehler
- Comics
  - xkcd.com
  - …

FRIEDRICH-SCHILLER-
UNIVERSITÄT
JENA

Allgemeines zu Python

www.python.org

Allgemeines

- Skriptsprache (Interpreter statt Compiler)
- Offen entwickelt (Python Software Foundation)
- Implementiert in C
- Verschiedene andere Implementierungen (z. B. Jython, IronPython, PyPy, …)

Geschichte

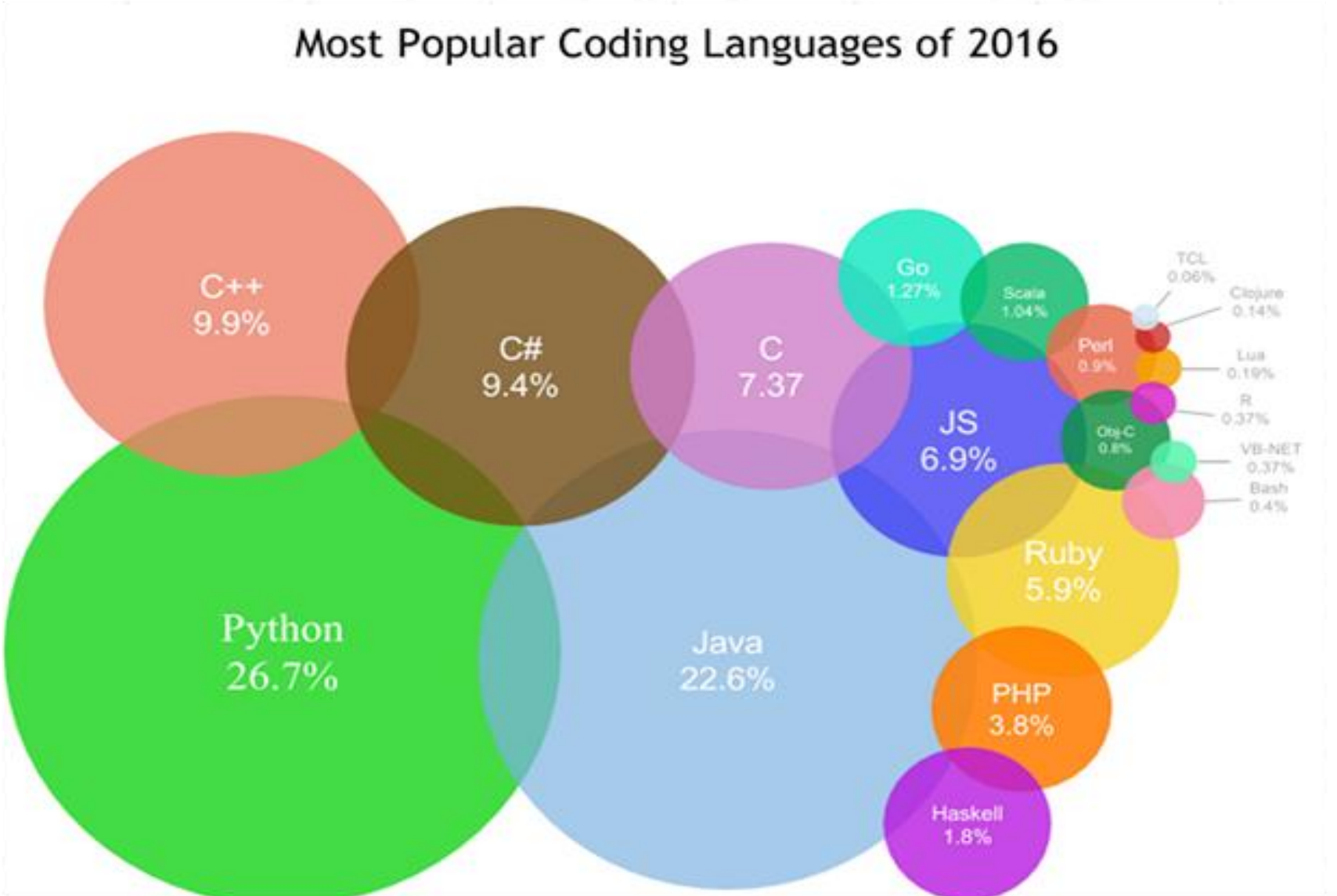
- Entwickelt Anfang der 1990er von Guido van Rossum als Nachfolger der Sprache ABC
- Name angelehnt an Monty Python
- 1994 erste funktionierende Vollversion
- Ab 2008 zwei verschiedene Versionen die parallel weiterentwickelt werden
- Aktueller Stand: Python 2.7.15

  Python 3.7.1

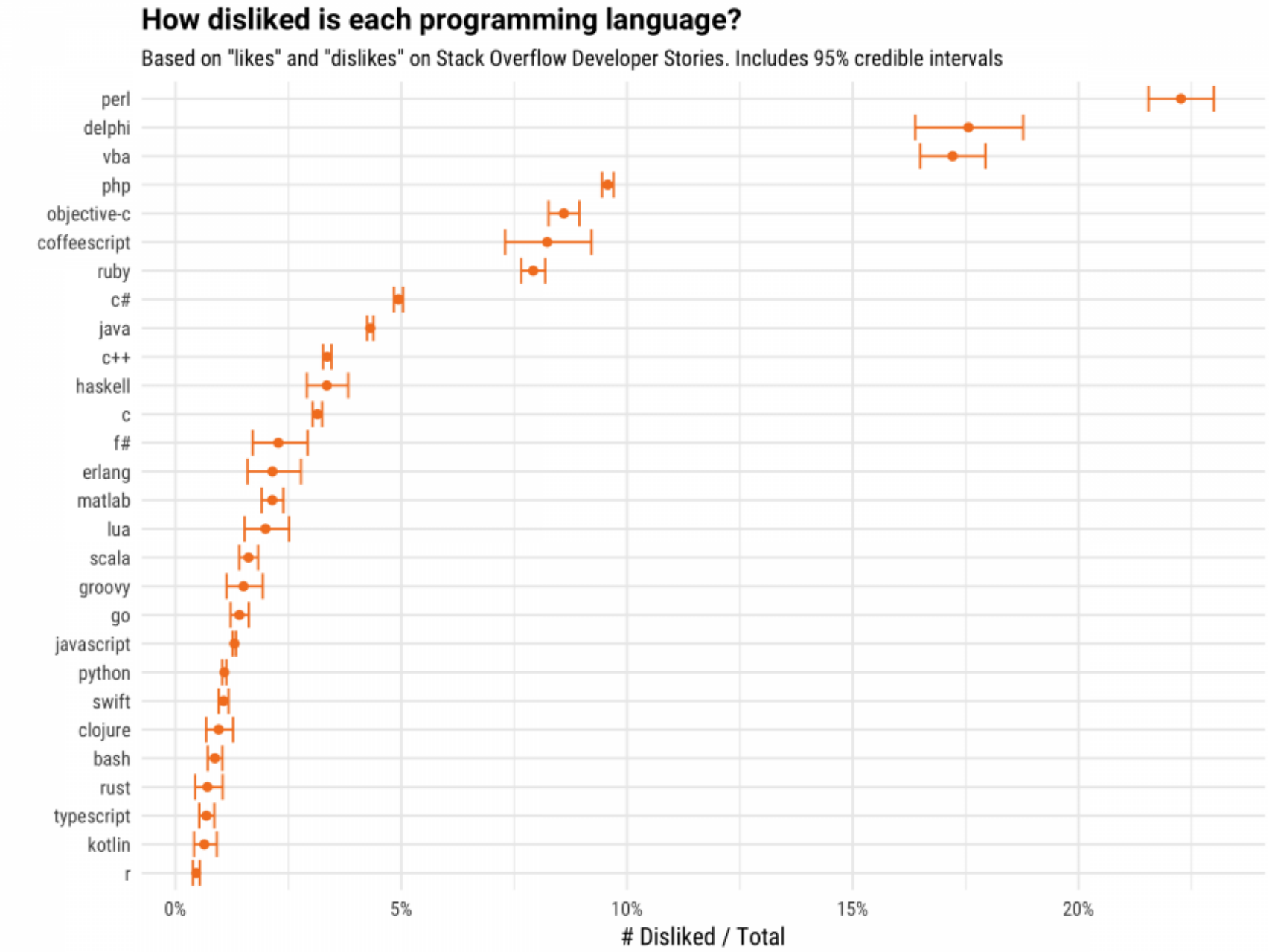
Warum Python?

- Minimalistische Sprache
- Umfassend Objektorientiert
- Interaktiver Modus (Python-Shell)
- Viele (wissenschaftliche) Module
- Nichtkommerziell
- Aktive große Community

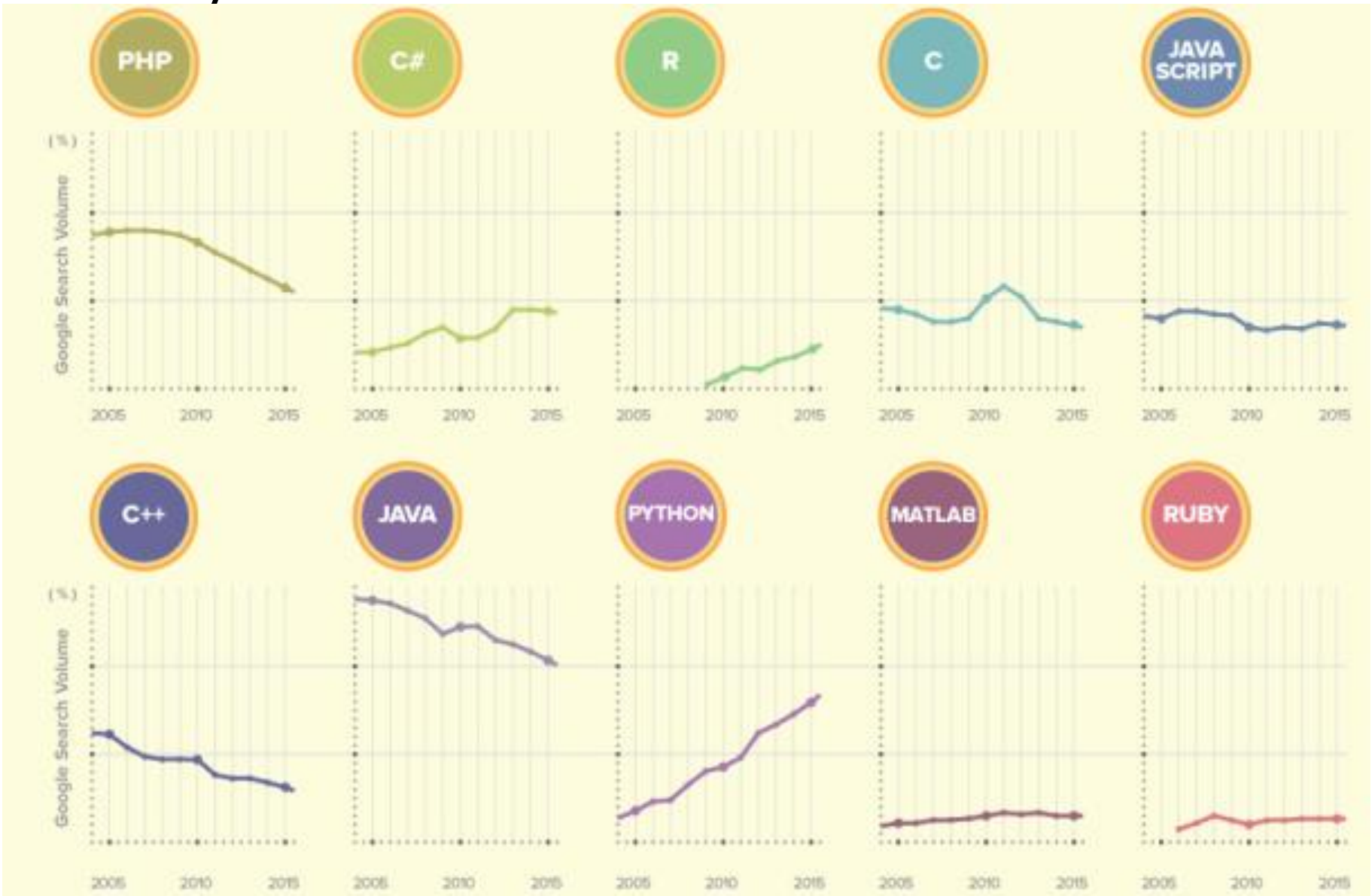
Warum Python?



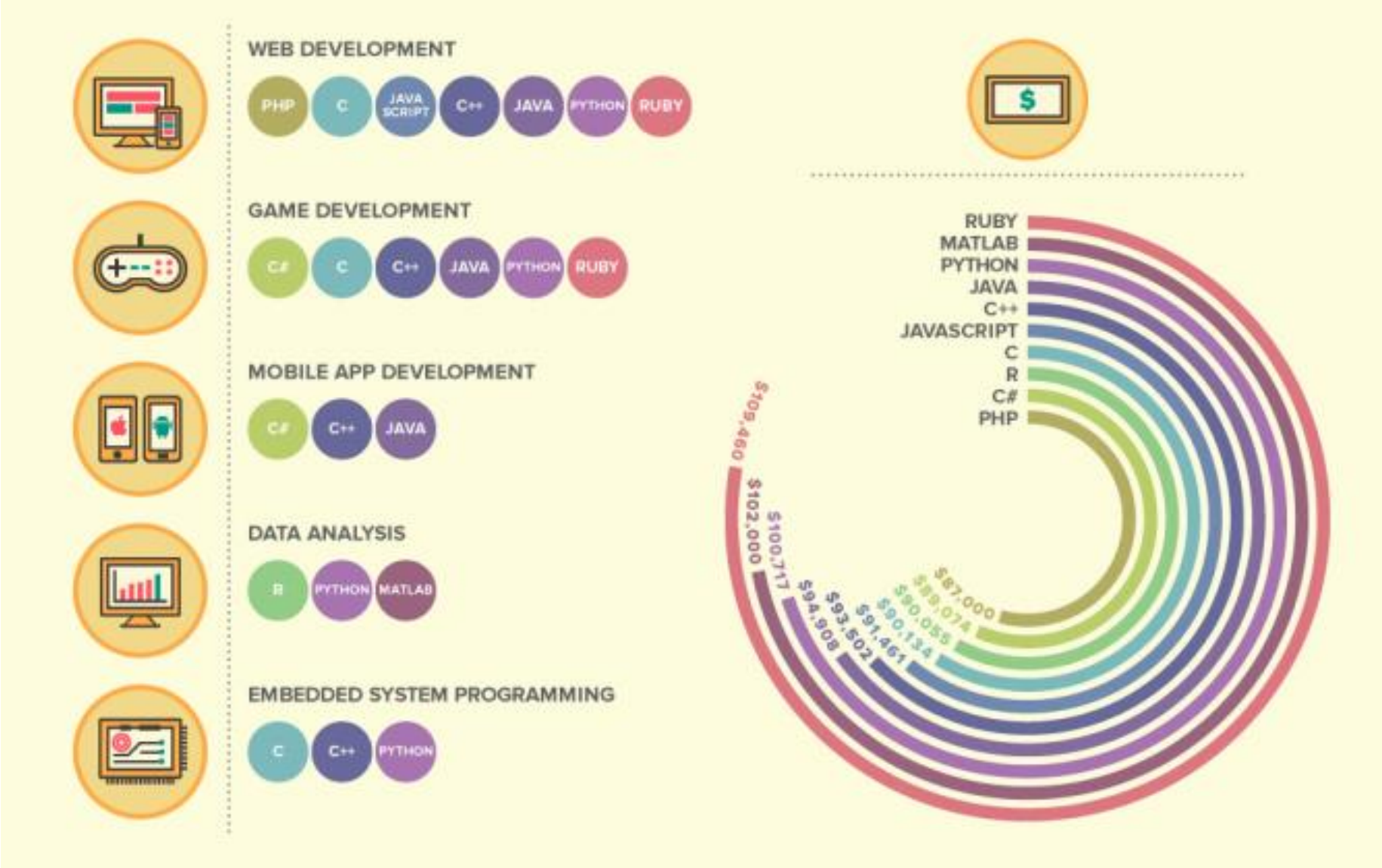
Warum Python?


How disliked is each programming language?
Based on "likes" and "dislikes" on Stack Overflow Developer Stories. Includes 95% credible intervals
perl
delphi
vba
php
objective-c
coffeescript
ruby
c#
java
c++
haskell
c
f#
erlang
matlab
lua
scala
groovy
go
javascript
python
swift
clojure
bash
rust
typescript
kotlin
r
0%
5%
10%
15%
20%
Disliked / Total

Warum Python?



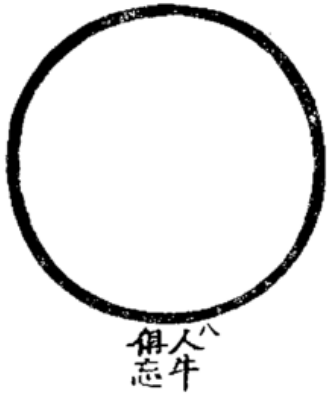
Warum Python?



Eigenschaften von Python

- Whitespace sensitiv
- Objektorientiert (alles ist ein Objekt)
- Dynamisch getypt und interpretiert (alles ist dynamisch)
- Funktional
- Aspektorientiert
- Interaktiv (ad-hoc Skripte)
- Automatische Speicherverwaltung
- Sehr gut lesbar

"There should be one, and preferably only one, obvious way to do it."



FRIEDRICH-SCHILLER-
UNIVERSITÄT
JENA

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-Shell & IDLE

- Interaktiver Modus in jeder Konsole startbar
- Ermöglicht einfaches testen und experimentieren mit Python Code
- IDLE (**I**ntegrated **D**eve**L**opment **E**nvironment) ist eine einfache Entwicklungsumgebung
- Syntax-highlighting, integrierter Debugger
- Es existieren weitere Python-Shells (z. B. iPython)

FRIEDRICH-SCHILLER-
UNIVERSITÄT
JENA

Good Python Practice

FRIEDRICH-SCHILLER-
UNIVERSITÄT
JENA

Good ~~Python~~ Programming Practice

FRIEDRICH-SCHILLER-
UNIVERSITÄT
JENA

Good ~~Python~~ Programming Practice: Lektion I

- Kommentiert die Funktion eurer Skripte
- Kommentiert eure Klassen
- Kommentiert eure Funktionen
- Kommentiert eure Variablen
- Kommentiert euren Code
- Kommentiert eure Kommentare
- Schreibt ausführliche Kommentare

```
#Einzeilige Kommentare starten mit ‘#‘


“““
Blockkommentare starten und enden mit
drei Anführungszeichen und können über
mehrere Zeilen gehen.
“““



#!/usr/bin/env python
# -*- coding: utf8 -*-
```

Python Code Beispiel

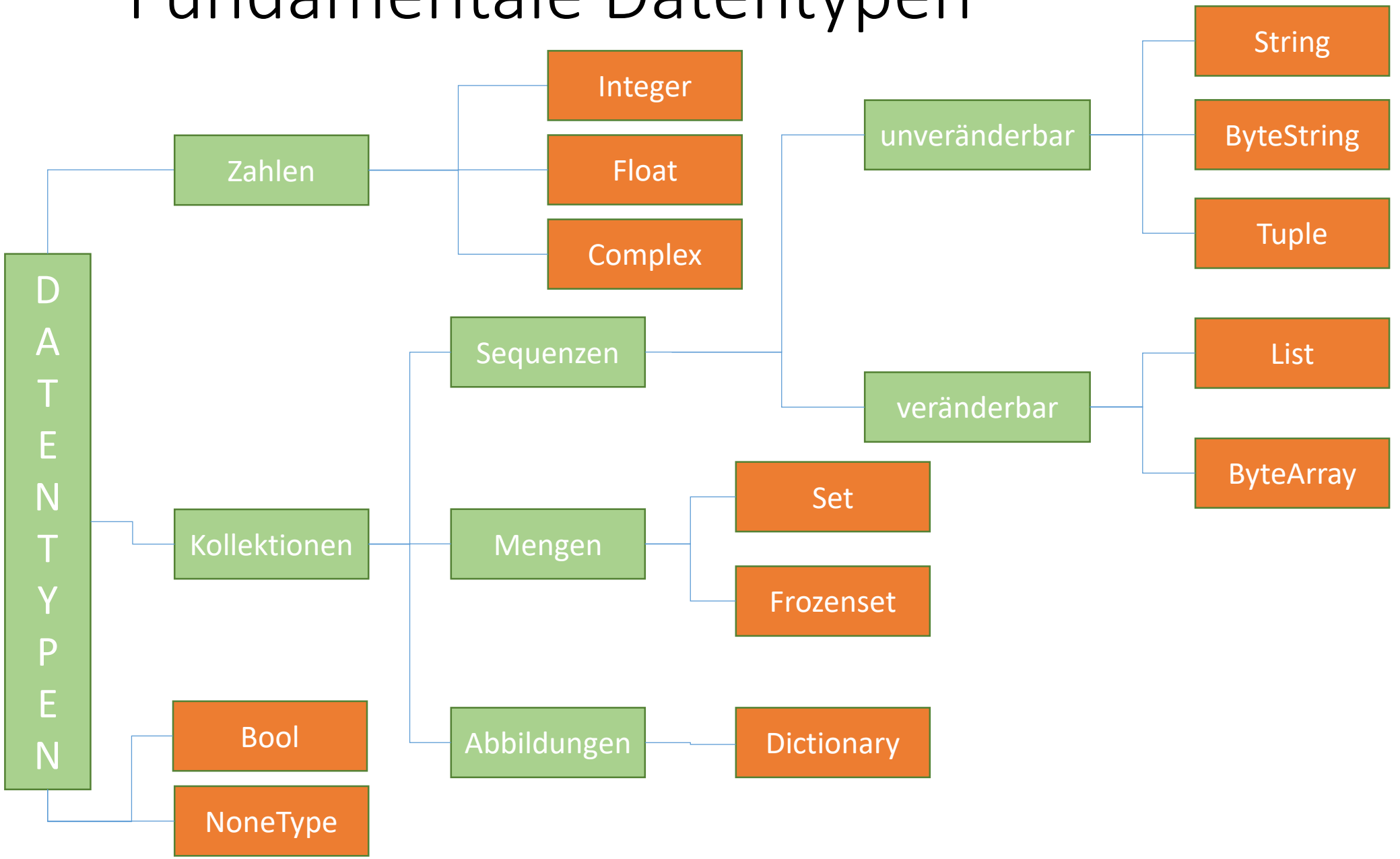
FRIEDRICH-SCHILLER-
UNIVERSITÄT
JENA

```
import os
import sys

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

fastaDict = dict()
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 dict entry
            sequence = ''
            header = line
      else:
            sequence += line                  #No header, extend sequence

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

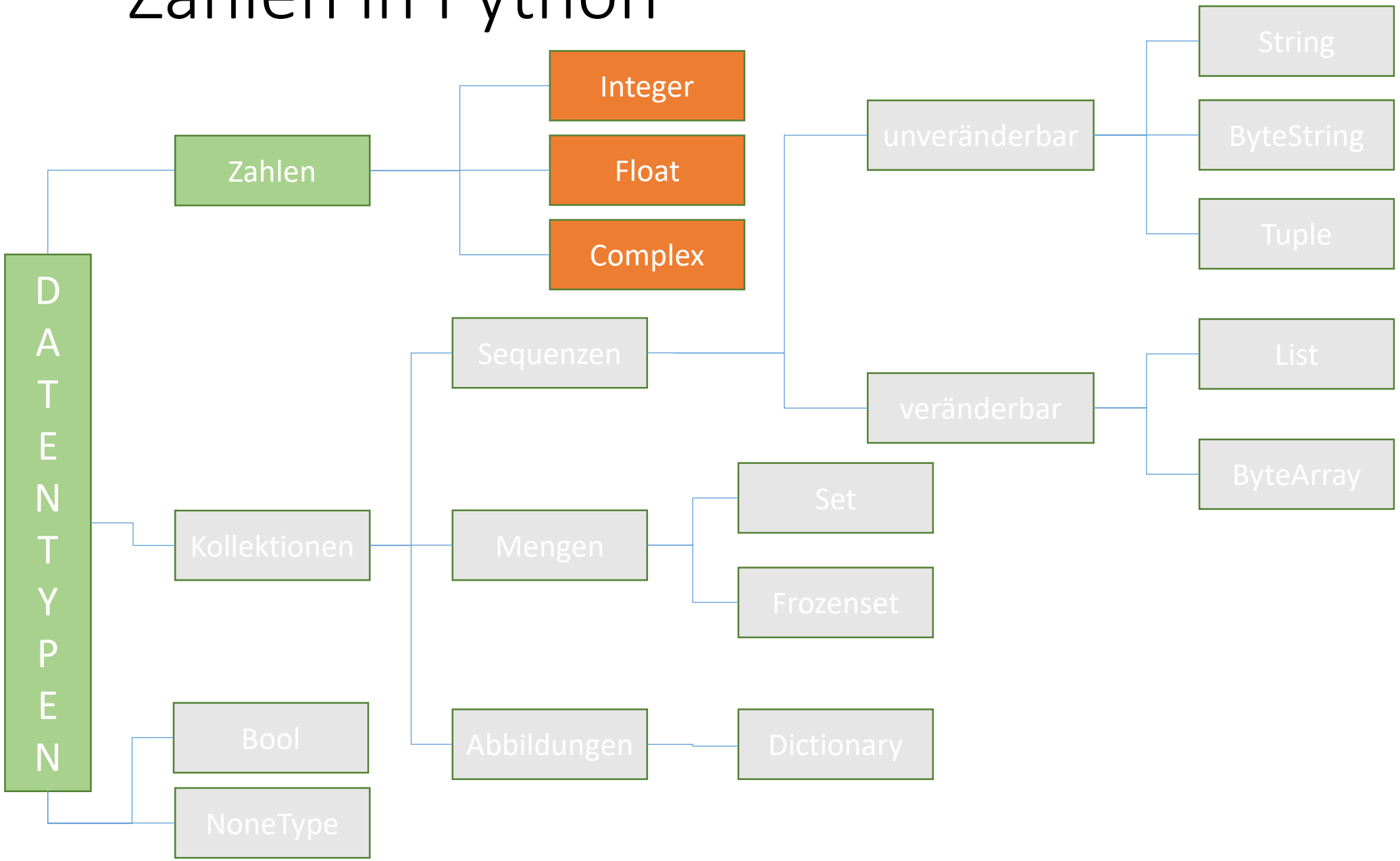
FRIEDRICH-SCHILLER-
UNIVERSITÄT
JENA

Fundamentale Datentypen

Fundamentale Datentypen



Zahlen in Python



Zahlen (Integers, Floats, Complex)

- Ganze Zahlen (Integer) können dezimal, oktal, hexadezimal, binär dargestellt werden
- Zahlen können in Python beliebig groß werden

```
>>>  19                          #Dezimalzahl
19
>>>  0o10                        #Oktalzahl
8
>>>  0x10                        #Hexadezimalzahl
16
>>> 0b10                        #Binärzahl
2

>>>  019                         #Dezimalzahl darf nicht mit führender Null beginnen
SyntaxError: invalid token
```

Zahlen (Integers, Floats, Complex)

- Gleitkommazahlen können als Dezimalbruch oder in Exponentialschreibweise dargestellt werden

```
>>>  f = 19.36                #In Python gibt es nur Floats als Gleitkommadarstellung

>>>  5.0e-7                   #entspricht 0.0000005

>>>  149e+0.69                #Ungültig, da Exponent keine ganze Zahl

>>>  3.0/2
1.5

>>> 3/2
1.5

>>> 2/2
1.0                  #Das Ergebnis einer Python-Divison ist immer eine Float (Python3!)
```

- In Python gibt es nur den Float-Typ für Gleitkommazahlen anstelle von typischerweise Float und Double

Zahlen (Integers, Floats, Complex)

- Komplexe Zahlen folgen in Python folgendem Schema: x + y*j*

```
>>>  c = 5+19j                    # j bezeichnet den imaginären Teil der komplexen Zahl

>>>  (1+1j)*2                     # Ist unter den Operanden mind. eine komplexe Zahl ist
(2+2j)                              auch das Ergebnis eine komplexe Zahl
```

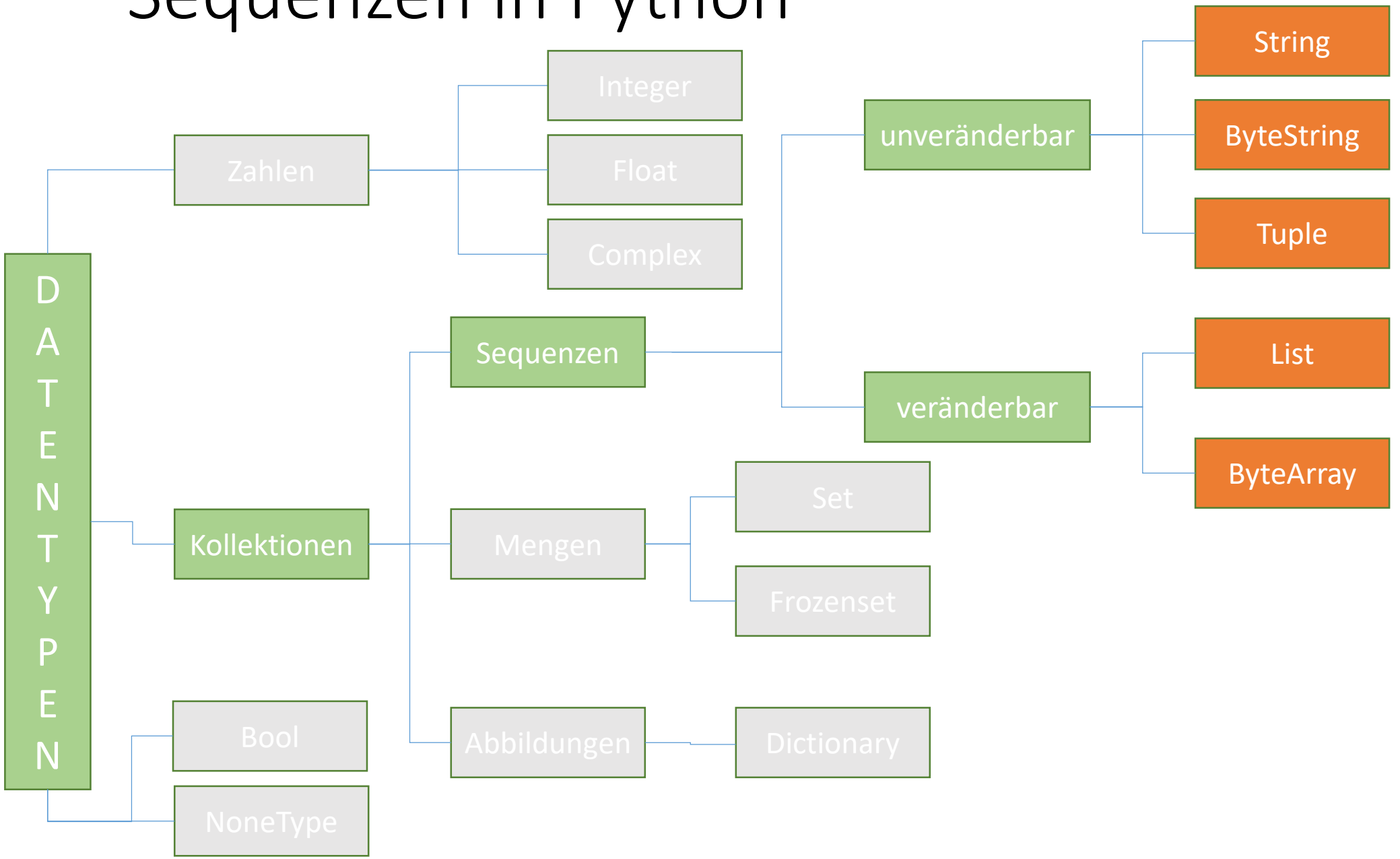
- Divison durch 0 löst einen *ZeroDivisionError* aus

Zahlen - Grundoperationen

FRIEDRICH-SCHILLER-
UNIVERSITÄT
JENA

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 (ganzzahliger Divisionsrest)	5 % 3	2
**	Potenz	2 ** 3	8

Sequenzen in Python



Sequenzen (Strings, Tupel, Listen)

FRIEDRICH-SCHILLER-
UNIVERSITÄT
JENA

- Iterierbar
- Geordnet, d.h. jedes Element ist durch einen Index gekennzeichnet

```
>>>  s = ‘Dies ist ein String.‘   #Strings können entweder in ‘ oder “ eingeschlossen sein
>>>  t = (‘Tupel‘, 5, s, (‘artus‘, ‘lancelot‘, ‘bedevere‘))
>>>  person = (‘Max‘, ‘Muster‘, ‘Jena‘ , 07745, ‘Musterstr. 19‘)

>>>  l = [90, 91, ‘Liste‘, t, [1, ‘robin‘, ‘galahad‘]]

>>>  s[0]                                  #Elemente sind mit 0 beginnend indiziert
‘D‘

>>>  t[2]
‘Dies ist ein String.‘

>>> l[-1]                          #Von hinten beginnend startet die Indizierung mit -1
[1, ‘robin‘, ‘galahad‘]
```

Sequenzen - Grundoperationen

Operation	Ergebnis
x in *s*	Gibt 1 wieder wenn ein Element mit dem Wert *x* in der Sequenz *s* enthalten ist, sonst 0.
x not in *s*	Gibt 0 wieder wenn ein Element mit dem Wert *x* in der Sequenz *s* enthalten ist, sonst 1.
s + *t*	Konkatenation der beiden Sequenzen s und t.
s * *n*	n Kopien der Sequenz s werden hintereinandergehängt.
s[i]	Wiedergeben des i-ten Elementes von *s*.
s[i:j]	Wiedergabe des Ausschnitts (*slice*) von *s*, der vom i-ten bis zum j-ten Element (nicht einschließlich) geht.
len(*s*)	Gibt die Länge der Sequenz *s* wieder.
min(*s*)	Gibt das kleinste Element der Sequenz *s* wieder.
max(*s*)	Gibt das größte Element der Sequenz *s* wieder.

Sequenzen – Grundoperationen I

```
>>>  sequenzName[ ]                          #Elementzugriff auf bestimmten Index durch [ ]

>>>  [5, 19, 36, 69, 119] + [149]           #Konkatenation funktioniert nur für Sequenzen gleichen Typs
[5, 19, 36, 69, 119, 149]

>>>  schlangen = (‘Python‘ , ‘Cobra‘)
>>>  nichtSchlangen = (‘Perl‘ , ‘Ruby‘)
>>>  scriptTiere = schlangen + nichtSchlangen
(‘Python‘ , ‘Cobra‘, ‘Perl‘ , ‘Ruby‘)
>>>  ‘Python ist toller als ‘ +[‘Perl‘]  #Fehler bei Konkatenierung von Sequenzen ungleichen Typs
TypeError: can only concatenate string (not “list“) to string

>>>  [1, 2, 3] * 4                          #Alle Sequenzen lassen sich mit * Vervielfältigen
[1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3]

>>>  liste = [ [ 1, 2, 3, 4] ]
>>>  len(liste)                             #Bestimmen der Länge einer Sequenz mit der len Funktion
1
>>>  len(liste[0])
4
```

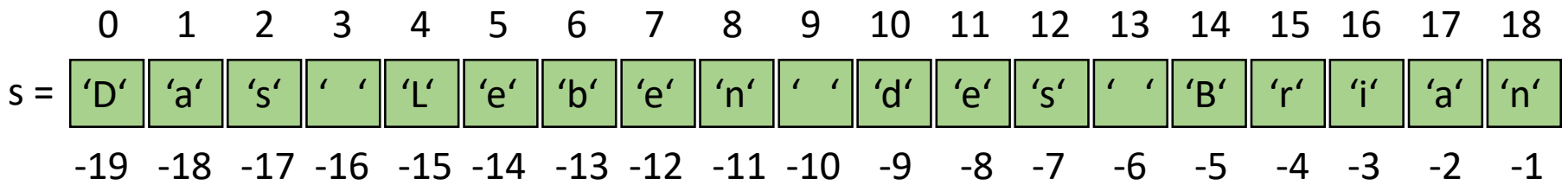
Sequenzen – Grundoperationen II

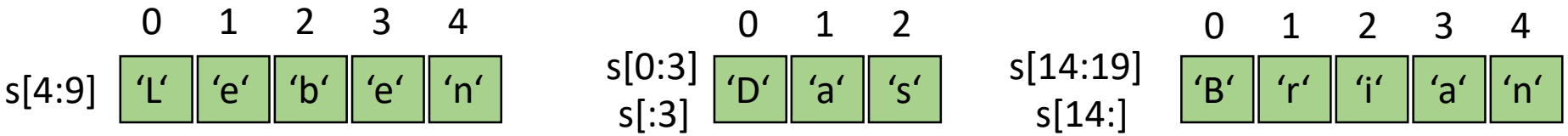
FRIEDRICH-SCHILLER-
UNIVERSITÄT
JENA

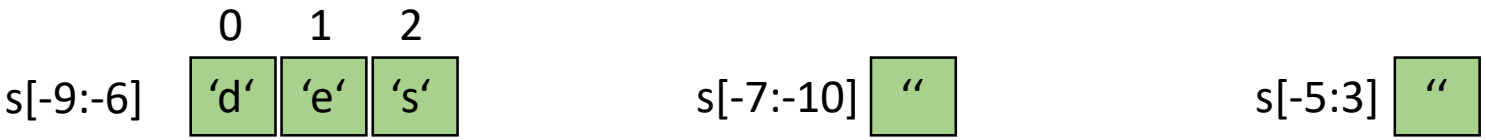
```
>>>  cooleSkriptsprachen = [‘Python‘ , ‘Ruby‘ , ‘Javascript‘ , ‘julia‘]
>>>  ‘Perl‘  in  cooleSkriptsprachen
False
>>>  ‘Perl‘  not in  cooleSkriptsprachen
True

>>>  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

>>>  liste[1:]
[2, 3, 4, 5, 6, 7]
>>> liste[:3]
[1, 2, 3]

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

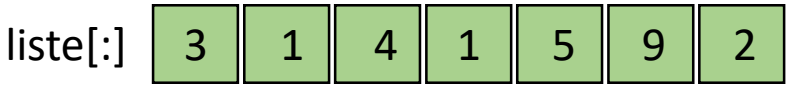
Sequenzen – Slicing

```
>>>  s = ‘Das Leben des Brian‘
>>>  liste = [3, 1, 4, 1, 5, 9, 2]
```









Listen - Operationen

Operation	Ergebnis
s[i] = x	Das Element mit Index i wird durch x ersetzt.
s.append(x)	An die Liste s wird ein neues Element x angehängt.
s.extend(t)	Die Liste s wird um die Element der Sequenz t verlängert.
s.count(x)	Gibt die Anzahl der Listenelemente mit dem Wert x zurück.
del s[i]	Das Element mit Index i wird aus der Liste s entfernt, damit veringert sich auch die Länge der Liste um 1.
s.index(x)	Zurückgegeben wird der kleinste Index von s an dem ein Element gleich x ist.
s.insert(i, x)	Falls i >= 0 wird x vor dem Index i in die Liste s eingefügt.
s.pop()	Das letzte Element von s wird aus der Liste entfernt und wiedergegeben.
s.remove(x)	Das erste Element gleich x wird aus der Liste s entfernt.
s.sort()	Die Elemente der Liste werden aufsteigend sortiert.

Listen - Operationen

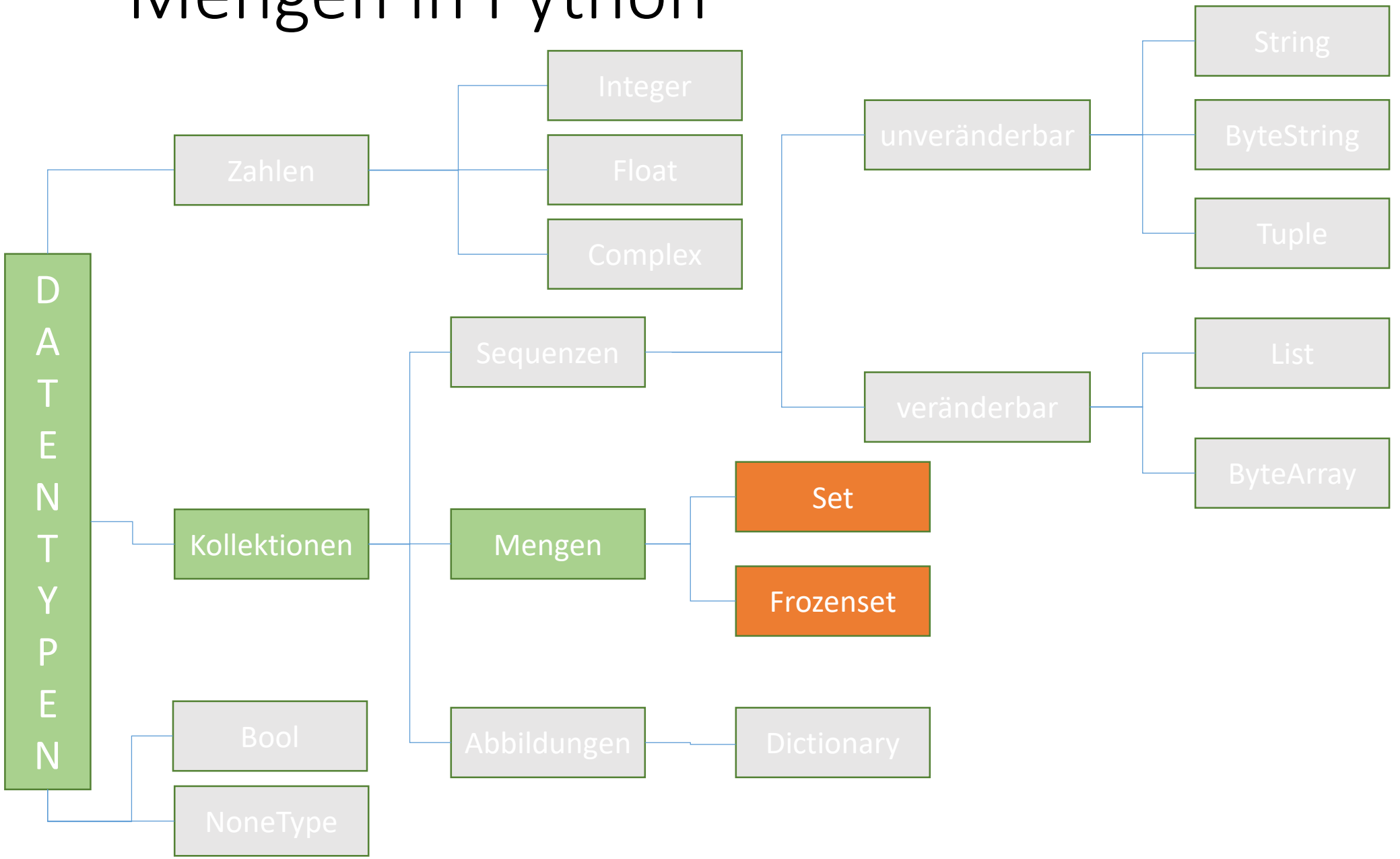
```
>>> l = [1, 2, 3, 4]

>>> 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]
```

Listen erweitern und sortieren

```
>>> 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]
```

FRIEDRICH-SCHILLER-
UNIVERSITÄT
JENA

Mengen in Python



Mengen (Sets, Frozensets)

- Iterierbar
- Ungeordnet, d. h. die Elemente haben keine feste Reihenfolge (keine Indizes)
- Ohne Duplikate, d. h. jedes Element kommt nur genau einmal vor

```
>>> {1,2,3}
{1,2,3}

>>> set([19,19,5,90,36])
{90, 19, 36, 5}

>>> set('Mississippi')
{'s', 'p', 'i', 'M'}
```

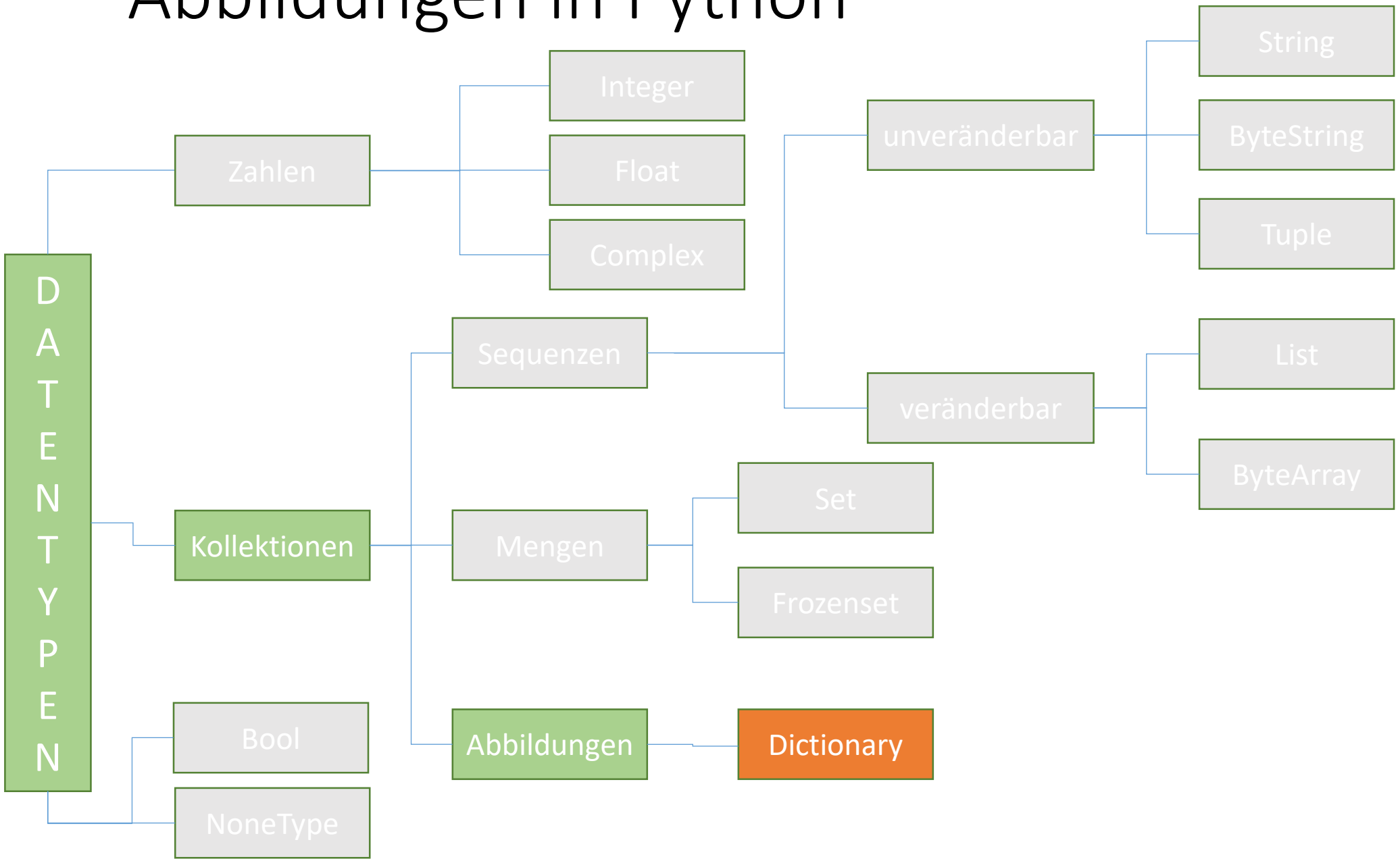
Mengen - Grundoperationen

FRIEDRICH-SCHILLER-
UNIVERSITÄT
JENA

Operation	Kurzform	Bedeutung
s1.union(s2)	s1 | s2	Alle Elemente von s1 und s2.
s1.intersection(s2)	s1 & s2	Alle gemeinsamen Elemente von s1 und s2.
s1.difference(s2)	s 1 – s2	Alle Elemente von s1 die nicht auch in s2 sind.
s1.symmetric_difference(s2)	s1 ^ s2	Alle Elemente in s1 oder s2, aber nicht in beiden.
s1.issubset(s2)	s1 <= s2	Liefert *TRUE* wenn s1 Teilmenge von s2 ist.
s1.isupperset(s2)	s1 >= s2	Liefert *TRUE* wenn s1 Obermenge von s2 ist.
s.copy()		Liefert eine flache Kopie von s,
x in s x not in s		Liefert *TRUE*, falls *x* Element von s ist. Liefert *TRUE*, falls *x* nicht Element von s ist.

```
>>> set1 = {1,2,3,4,5}
>>> set2 = {4,5,6,7,8}

>>> set1.union(set2)                          #Vereinigung
{1,2,3,4,5,6,7,8}

>>> set1.intersection(set2)                   #Schnitt
{4,5}

>>> set1.symmetric_difference(s2)             #Symmetrische Differenz
{1,2,3,6,7,8}

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

>>> 1 in s1                                   #Element enthalten in set
True
```

Abbildungen in Python



Abbildungen (Dictionary)

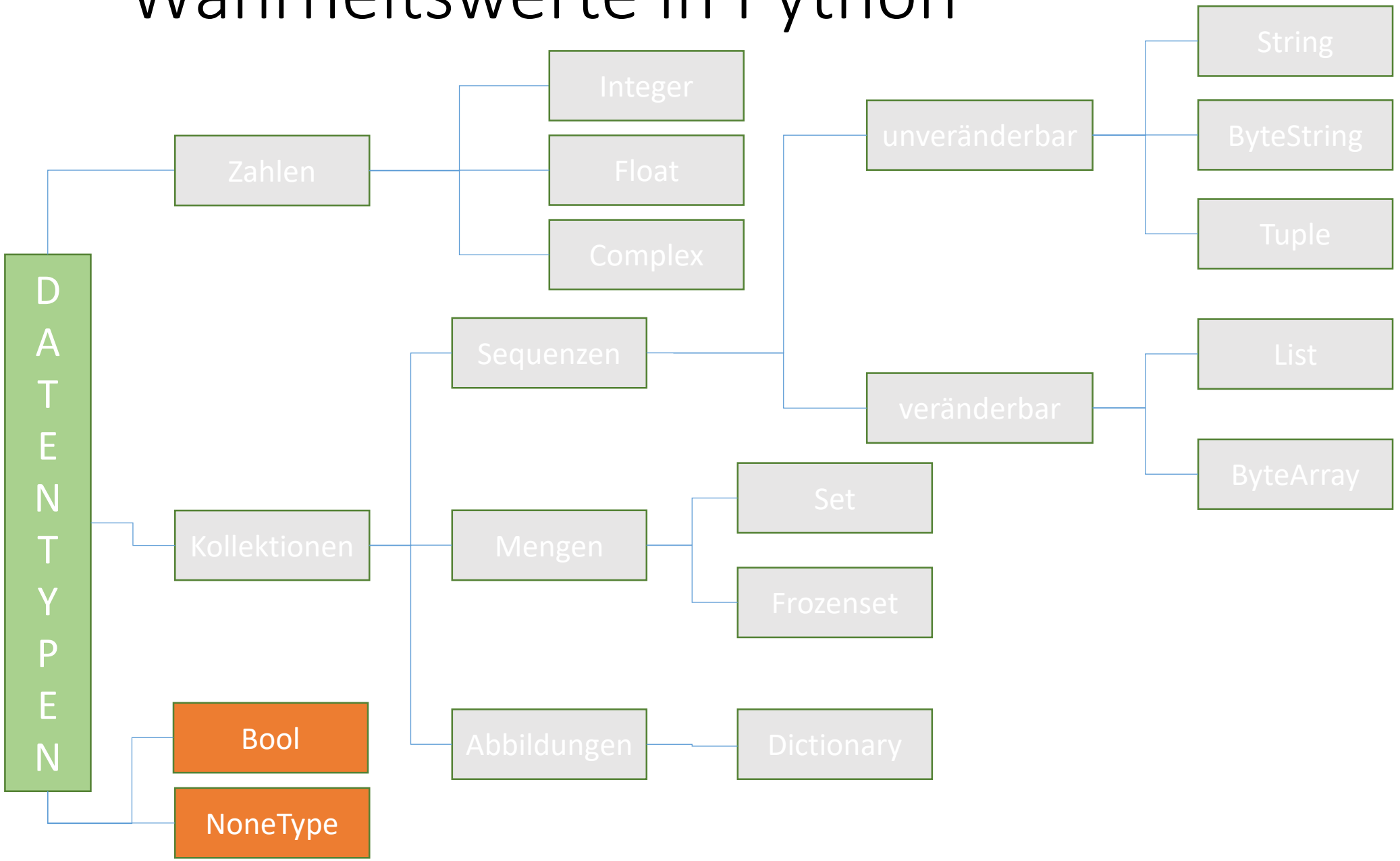
- Iterierbar
- Ungeordnet
- Indexierung kann durch willkürliche Werte vorgenommen werden
- Besteht aus Paaren der Form: *schlüssel : wert*

```
>>> 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
{'Drei': 3, 'Hello': 'World', 'Liste': [1, 2, 3], 'Vier': 4, 'Zwei': 2,
'Eins': 1}
```

Wichtige Dictionary Operationen

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

Wahrheitswerte in Python



NoneType und Bool

- NoneType hat genau ein Literal: *None*
  - Besitzt keinen Wert und unterstützt fast keinerlei Funktionen

```
>>> n = None
>>> n
>>>
>>> print(n)
None
```

- Bool hat genau zwei Literale: *True, False*
  - Einfacher Wahrheitswert (Haben auch numerische Werte 1, 0)
  - *Achtung:* Alle anderen Fundamentalen Datentypen haben auch einen Wahrheitswert

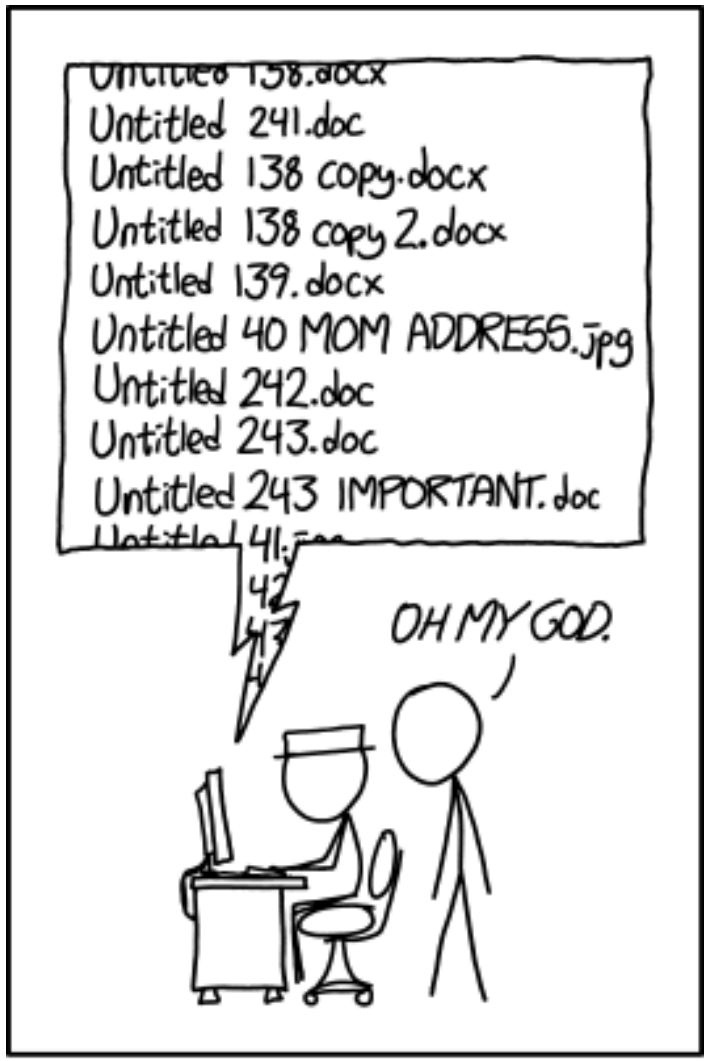
```
>>> b = True
>>> if(19 > 5  or  1  and  b):
...         print('Der Ausdruck ist wahr!')
Der Ausdruck ist wahr!
```

Variablendeklaration Übersicht

```
>>> st = ‘Ritter der Kokosnuss‘
>>> st = “Flying Circus“
>>> st = str()

>>> i = 5
>>> i = int()

>>> f = 19.0
>>> f = float()

>>> se = {36,69,119}
>>> se = set()

>>> l = [‘Python‘, 3.5, 2.7]
>>> l = list()

>>> t = (x,y,z)
>>> t = tupel()
```

FRIEDRICH-SCHILLER-
UNIVERSITÄT
JENA

Good Programming Practice: Lektion II

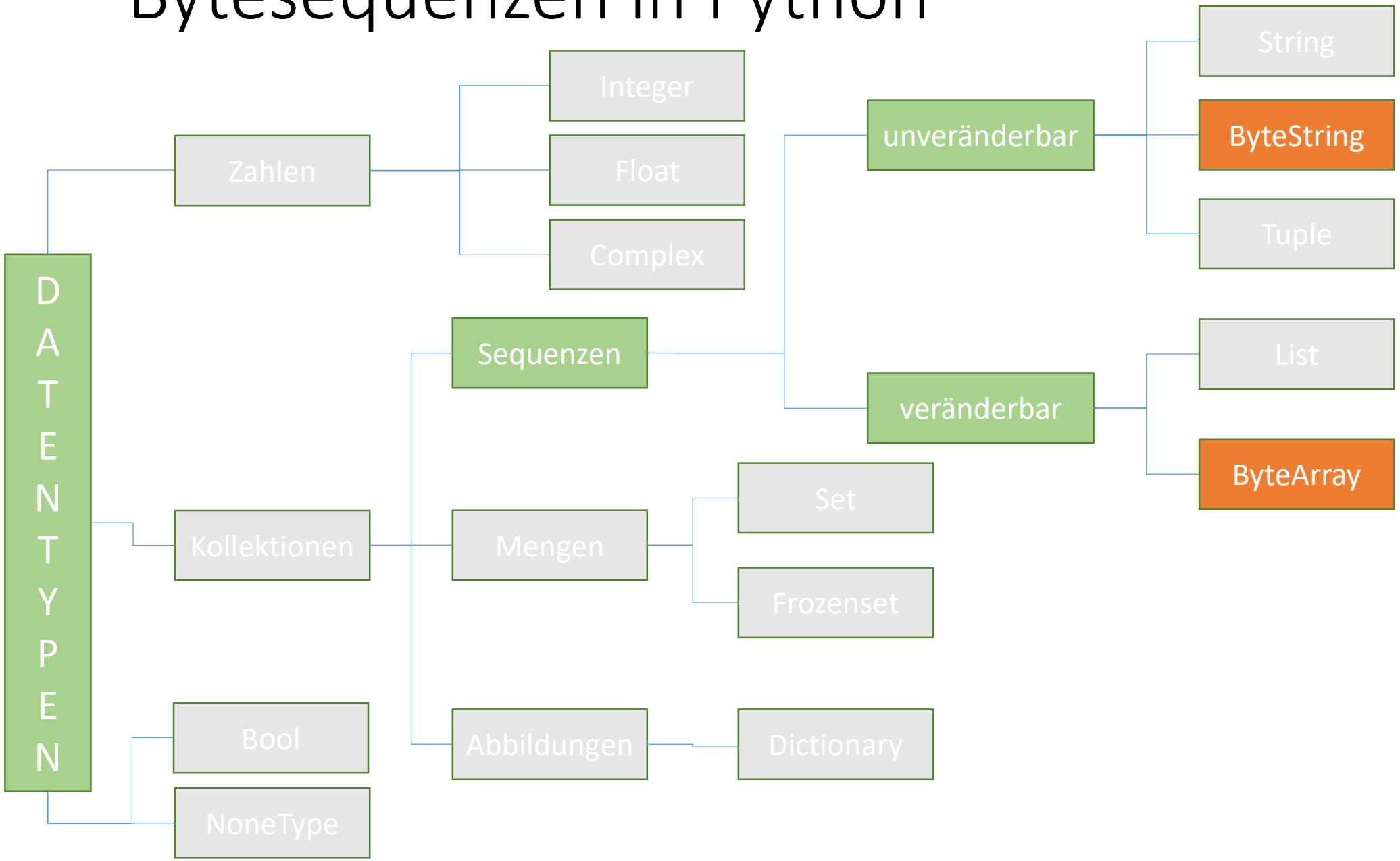


PROTIP: NEVER LOOK IN SOMEONE ELSE'S DOCUMENTS FOLDER.

Good Programming Practice: Lektion II

- Benutzt **bezeichnende** Variablen- , Funktions-, Klassennamen
- Nur Klassennamen beginnen mit einem großem Buchstaben und werden im ***CamelCase*** Stil geschrieben: *MeinEigeneKlasse*
- Variablennamen aus mehreren Wörten werden im ***mixedCase*** Stil geschrieben: *langerVariablenname*
- Funktionsnamen aus mehreren Wörtern werden mit **Unterstrich** geschrieben: *dies_ist_eine_funktion*
- Konstanten werden **komplett groß** und mit **Unterstrich** geschrieben: EINE_KONSTANTE

Bytesequenzen in Python



ByteString und ByteArrays

- ByteStrings sind Folgen von Zahlen zwischen 0 und 255 und werden im ByteString-Literal möglichst als ASCII-Zeichen dargestellt.

- ByteArrays sind Listen aus Zahlen zwischen 0 und 255 und sind damit im Gegensatz zu ByteStrings veränderbar.

- Wichtig wenn man Symbolkodierung zwischen verschiedenen Systemen beachten muss.

```
>>>  b = bytes([80, 121, 116, 104, 111, 110])
>>>  b
b‘Python‘
```

FRIEDRICH-SCHILLER-
UNIVERSITÄT
JENA

ASCII Table

Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char
0	0	0		32	20	40	[space]	64	40	100	@	96	60	140	`
1	1	1		33	21	41	!	65	41	101	A	97	61	141	a
2	2	2		34	22	42	"	66	42	102	B	98	62	142	b
3	3	3		35	23	43	#	67	43	103	C	99	63	143	c
4	4	4		36	24	44	$	68	44	104	D	100	64	144	d
5	5	5		37	25	45	%	69	45	105	E	101	65	145	e
6	6	6		38	26	46	&	70	46	106	F	102	66	146	f
7	7	7		39	27	47	'	71	47	107	G	103	67	147	g
8	8	10		40	28	50	(	72	48	110	H	104	68	150	h
9	9	11		41	29	51	)	73	49	111	I	105	69	151	i
10	A	12		42	2A	52	*	74	4A	112	J	106	6A	152	j
11	B	13		43	2B	53	+	75	4B	113	K	107	6B	153	k
12	C	14		44	2C	54	,	76	4C	114	L	108	6C	154	l
13	D	15		45	2D	55	-	77	4D	115	M	109	6D	155	m
14	E	16		46	2E	56	.	78	4E	116	N	110	6E	156	n
15	F	17		47	2F	57	/	79	4F	117	O	111	6F	157	o
16	10	20		48	30	60	0	80	50	120	P	112	70	160	p
17	11	21		49	31	61	1	81	51	121	Q	113	71	161	q
18	12	22		50	32	62	2	82	52	122	R	114	72	162	r
19	13	23		51	33	63	3	83	53	123	S	115	73	163	s
20	14	24		52	34	64	4	84	54	124	T	116	74	164	t
21	15	25		53	35	65	5	85	55	125	U	117	75	165	u
22	16	26		54	36	66	6	86	56	126	V	118	76	166	v
23	17	27		55	37	67	7	87	57	127	W	119	77	167	w
24	18	30		56	38	70	8	88	58	130	X	120	78	170	x
25	19	31		57	39	71	9	89	59	131	Y	121	79	171	y
26	1A	32		58	3A	72	:	90	5A	132	Z	122	7A	172	z
27	1B	33		59	3B	73	;	91	5B	133	[	123	7B	173	{
28	1C	34		60	3C	74	<	92	5C	134	\	124	7C	174	|
29	1D	35		61	3D	75	=	93	5D	135	]	125	7D	175	}
30	1E	36		62	3E	76	>	94	5E	136	^	126	7E	176	~
31	1F	37		63	3F	77	?	95	5F	137	_	127	7F	177	

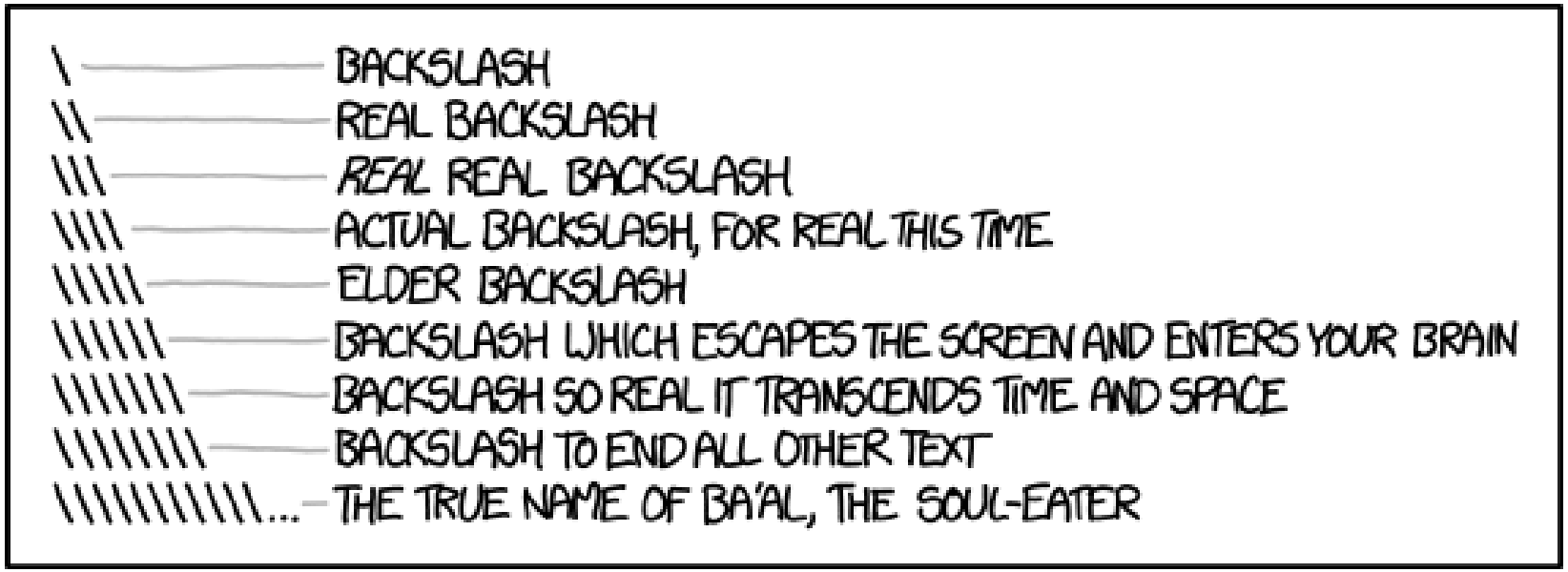
ByteString und ByteArrays

FRIEDRICH-SCHILLER-
UNIVERSITÄT
JENA

```
>>>  s = bytes(‘ß ä ö ü‘, ‘ascii‘)
UnicodeEncodeError: ‘ascii‘ codec can‘t encode character ‘\xdf‘ in position 0

>>>  s = bytes(‘ß ä ö ü‘, ‘utf-8‘)
>>>  s
b‘\xc3\x9f \xc3\xa4 \xc3\xb6 \xc3\xbc‘



>>>  s = ‘Dies ist eine‘
>>>  s = s + ‘ teure‘                    #Da Strings unveränderbar sind wird hier
>>>  s = s + ‘ Operation‘                #in jedem Schritt ein neues Objekt angelegt


>>>  s = bytearray(‘Dies ist eine‘, ‘ascii‘)
>>>  s = s + b‘ effiziente‘
>>>  s = s + b‘ Operation‘
```

Escape-Sequenzen

```
>>>  print(‘Um ‘Anführungszeichen‘ in Strings darzustellen braucht man ein \‘)
SyntaxError: invalid syntax

>>>  print(‘Um \‘Anführungszeichen\‘ in Strings darzustellen braucht man ein \\‘)
Um 'Anführungszeichen' in Strings darzustellen braucht man ein \

>>> print(‘Wir sind die Ritter die\t \‘Ni\‘ sagen:\nNiiiii‘)
Wir sind die Ritter die       ‘Ni‘ sagen:
Niiiii
```



Typumwandlung

Typumwandlung

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


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

Typumwandlung

- Variablen sind grundsätzlich dynamisch getypt
- Ausdrücke sind **stark** getypt → "Explicit is better than implicit."

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

- Umwandlung durch einfache Typ-/Castfunktionen:

int(), float(), complex(), bool(), str(), dict(), list(), tuple()

```
>>> 5 + int('14')
19

>>> str(5) + '14'
'514'
```

Typumwandlung

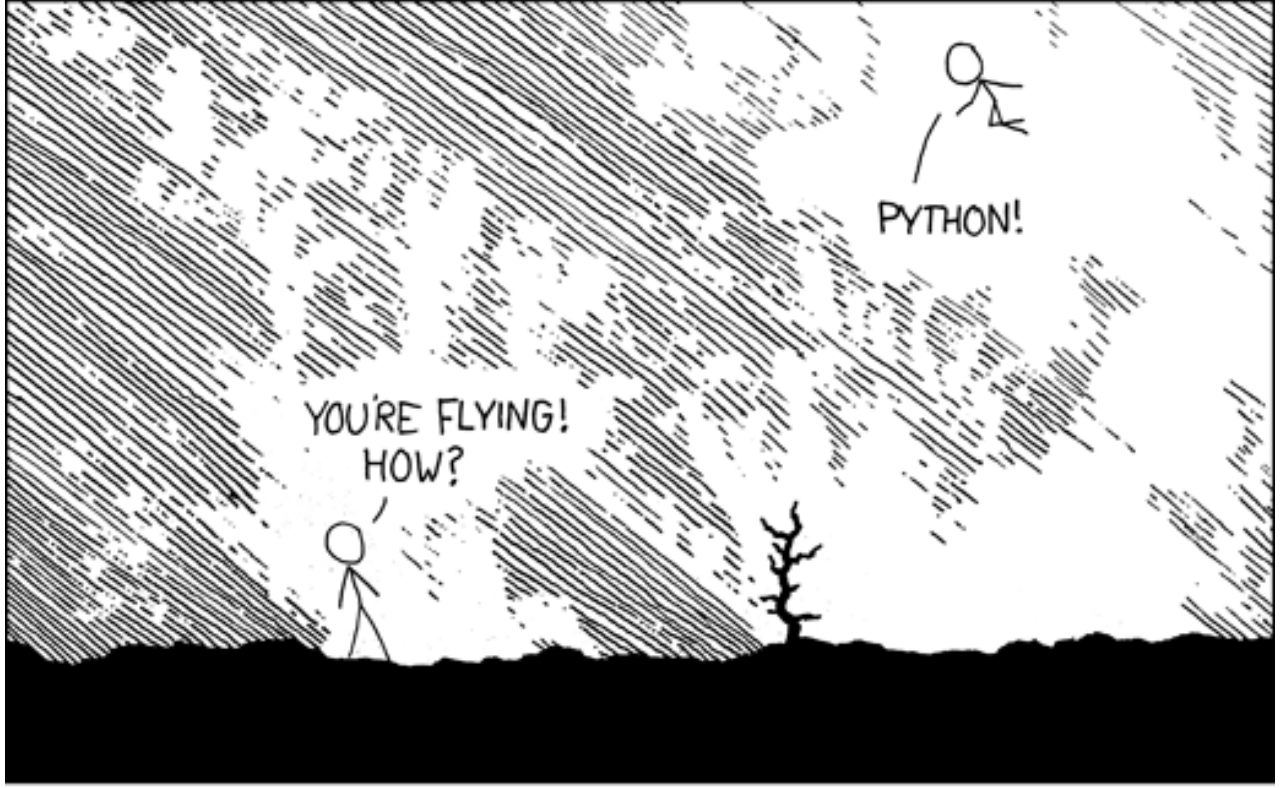
```
>>>  int(1.7)                  #Gleitkommazahlen werden immer abgerundet
1

>>>  int('5193669119149')
5193669119149

>>>  int('3.1415')             #str zu int castings dürfen nur aus Zahlen bestehen
ValueError: invalid literal for int() with base 10: '3.1415'


>>>  float('3.1414')
3.1415

>>>  float(12)
12.0

>>>  float('0.00001')
1e-5

>>>  float('1E-5')
1e-5
```

Typumwandlung

```
>>>  bool()                          #Wahrheitswert von None
False

>>>  bool(1)
True

>>>  bool('ABC')
True


>>>  str(3.1414)                     #Umwandlung einer Zahl in einen String
'3.1415'


>>>  type('12')                      #type() gibt den Objekttyp einer Eingabe wieder
<class 'str'>

>>>  x = 1e-5
>>>  type(x)
<class 'float'>
```

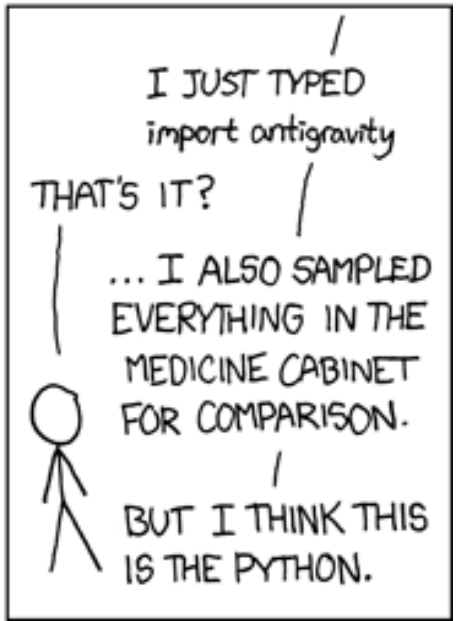
Typumwandlung

```
x = input(‘x: ‘)
y = input(‘y: ‘)
summe = float(x) + float(y)
print(‘Die Summe von ‘ +x +‘ und ‘ +y +‘ ist ‘ +str(summe))


X: 2
y: 3
Die Summe von 2 und 3 ist 5
```


PYTHON!
YOU'RE FLYING!
HOW?


I LEARNED IT LAST NIGHT! EVERYTHING IS SO SIMPLE!
HELLO WORLD IS JUST print "Hello, world!"


I DUNNO... DYNAMIC TYPING? WHITESPACE?
COME JOIN US! PROGRAMMING IS FUN AGAIN! IT'S A WHOLE NEW WORLD UP HERE!
BUT HOW ARE YOU FLYING?


I JUST TYPED import antigravity
THAT'S IT?
... I ALSO SAMPLED EVERYTHING IN THE MEDICINE CABINET FOR COMPARISON.
BUT I THINK THIS IS THE PYTHON.