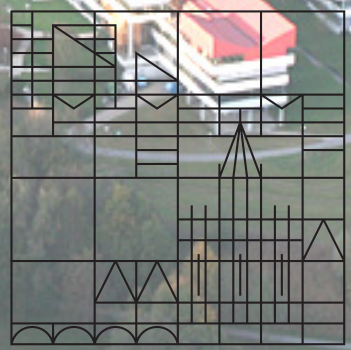
Computerphysik II

Python Einführung

S. Gerlach

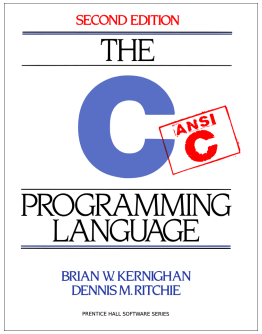
WiSe 2022/23

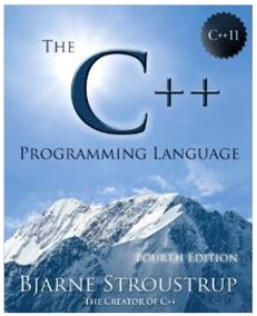
Universität
Konstanz

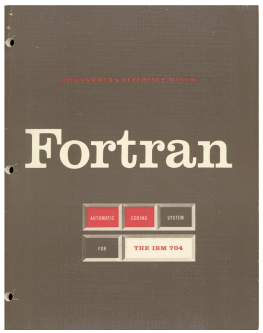


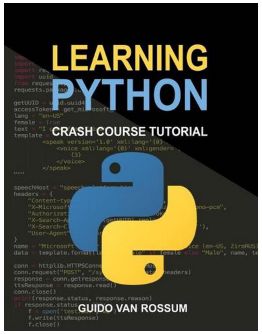
Programmiersprachen

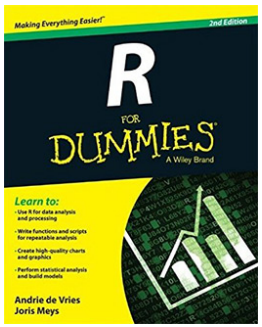
In den Naturwissenschaften:

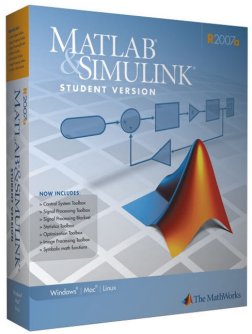


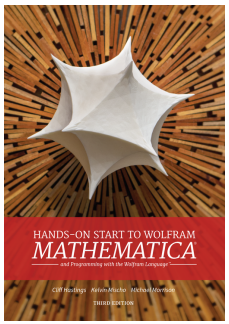


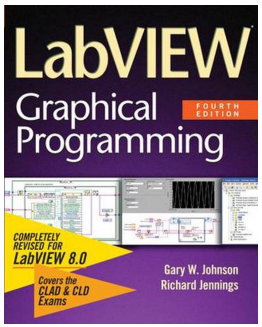




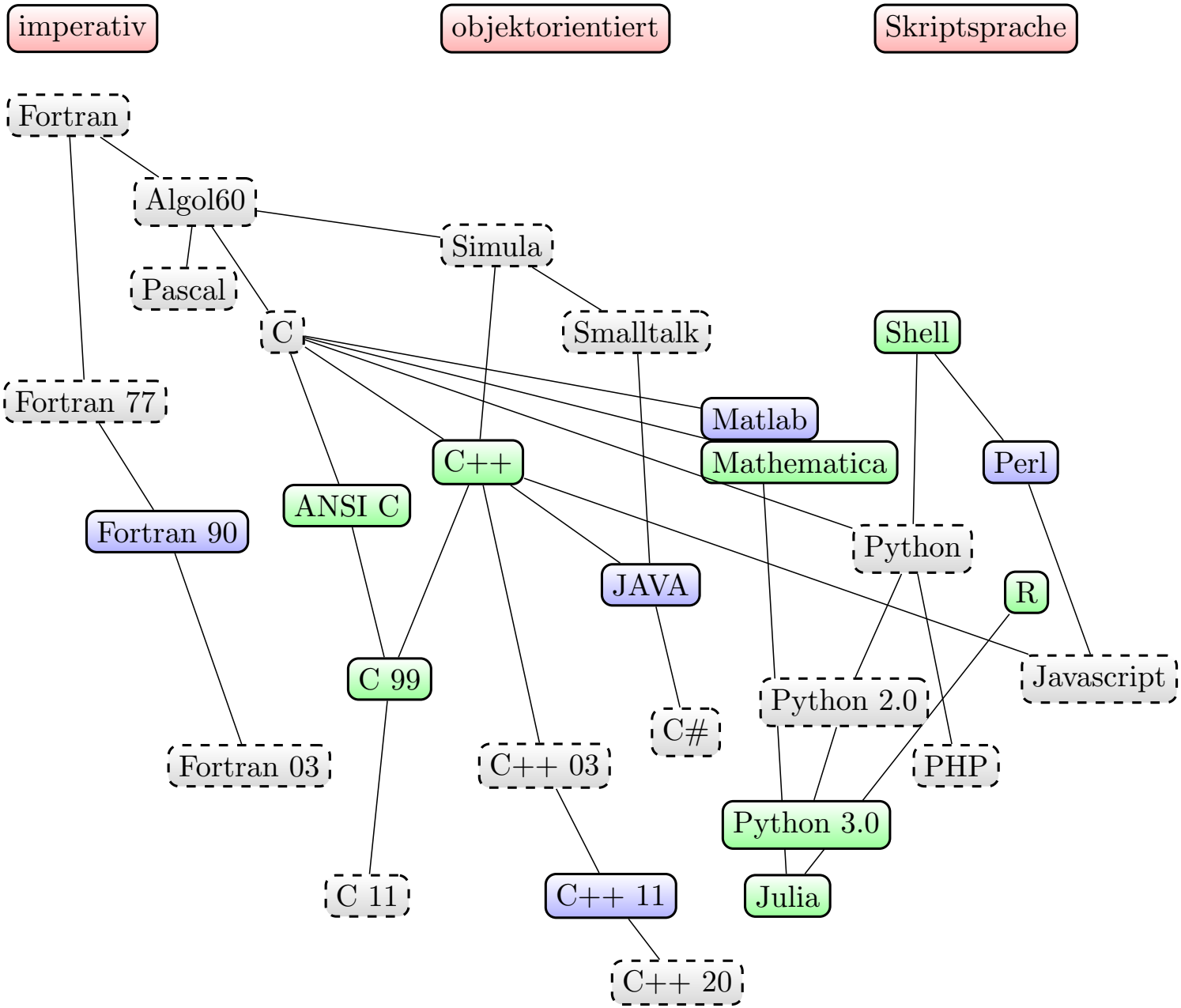








Programmiersprachen Entwicklung



Programmiersprachen - Einteilung

nach Eigenschaften:

- **Skriptsprachen**: Shell, Python, Perl, ...
- **Kompilierte Sprachen**: C/C++, Fortran, ...
- **Objektorientiert**: C++, Java, Python, ...
- **Anwendungssprachen**: R, Mathematica, Matlab, ...

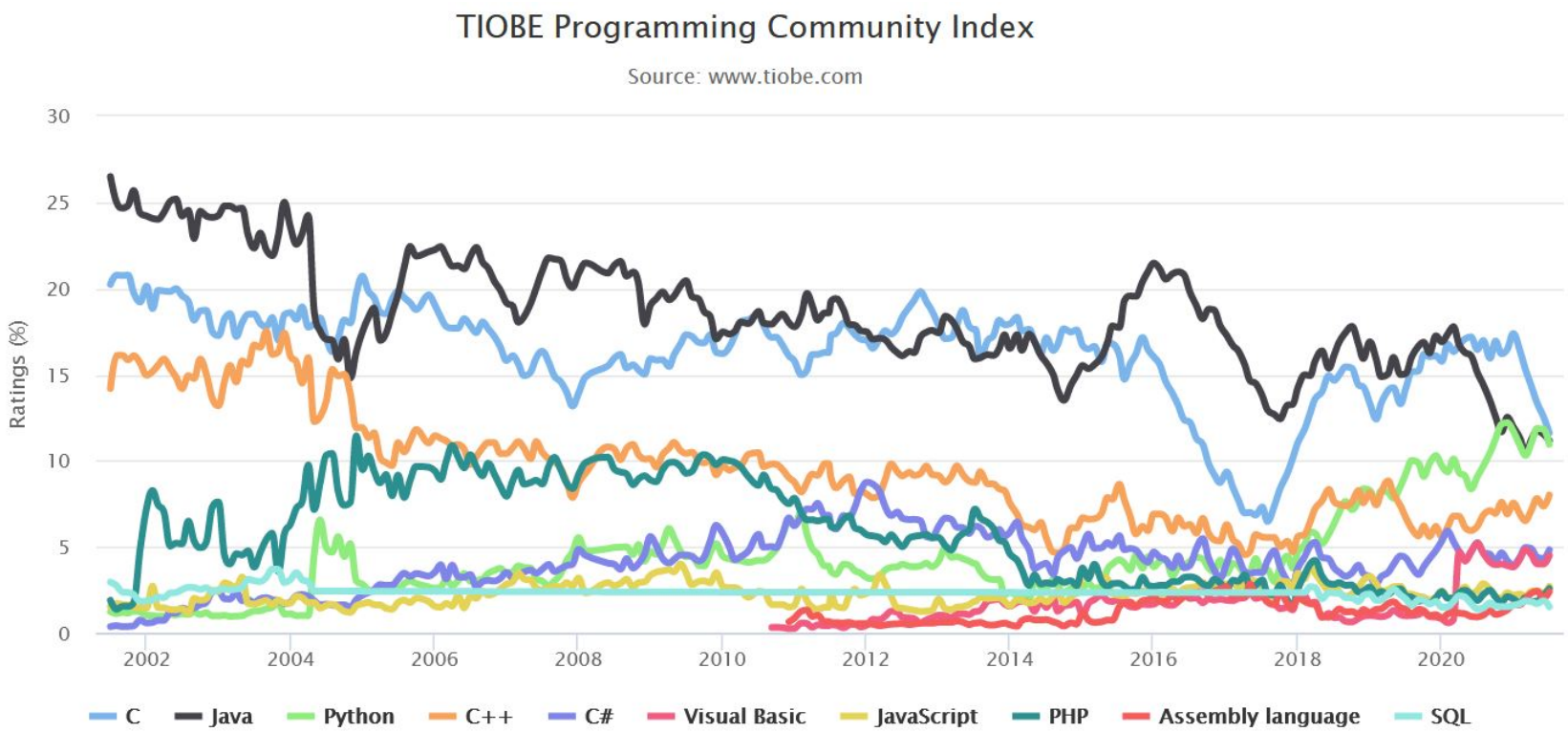
nach Anwendung:

- **Skripte, Automatisierung**: Shell, Python, Make, ...
- **Datenauswertung & Tests**: C/C++, Python, Matlab, ...
- **Analytische Rechnungen**: Mathematica, Maple, SAGE, Python (SymPy), ...
- **HPC + Scientific Computing**: C/C++, Fortran, Julia, Python (NumPy, SciPy), ...
- **Grafische Darstellung**: LabPlot, Python (matplotlib), Matlab, ...

Programmiersprachen - Verbreitung (TIOBE)

Programming Language	2021	2016	2011	2006	2001	1996	1991	1986
C	1	2	2	2	1	1	1	1
Python	2	5	6	8	25	25	-	-
Java	3	1	1	1	2	16	-	-
C++	4	3	3	3	3	2	2	6
C#	5	4	5	7	12	-	-	-
Visual Basic	6	13	-	-	-	-	-	-
JavaScript	7	7	10	9	9	20	-	-
PHP	8	6	4	4	10	-	-	-
Assembly language	9	11	-	-	-	-	-	-
SQL	10	-	-	-	37	-	-	-
Ada	29	28	17	16	18	7	5	2
Lisp	33	27	13	13	17	8	4	3
Pascal	268	75	15	17	15	5	3	7
(Visual) Basic	-	-	7	5	4	3	8	5

Programmiersprachen - Verbreitung (TIOBE)



Programmiersprache Python

- Hohe **Verbreitung** (gut bekannt, viele Bibliotheken)
- Freie Software
- interaktive Skriptsprache
- **performant** durch Verwendung von optimierten Bibliotheken bzw. Anbindung an C
- Einfach & flexibel (wenige Sprachfeatures, viele Bibliotheken)

Python 2.7.18 (End-of-Life 2020), aktuell: Python 3.8.15 - 3.10.8

`https://en.wikibooks.org/wiki/Python_Programming`
"Cheat-Sheets"

Verwendung - interaktiv

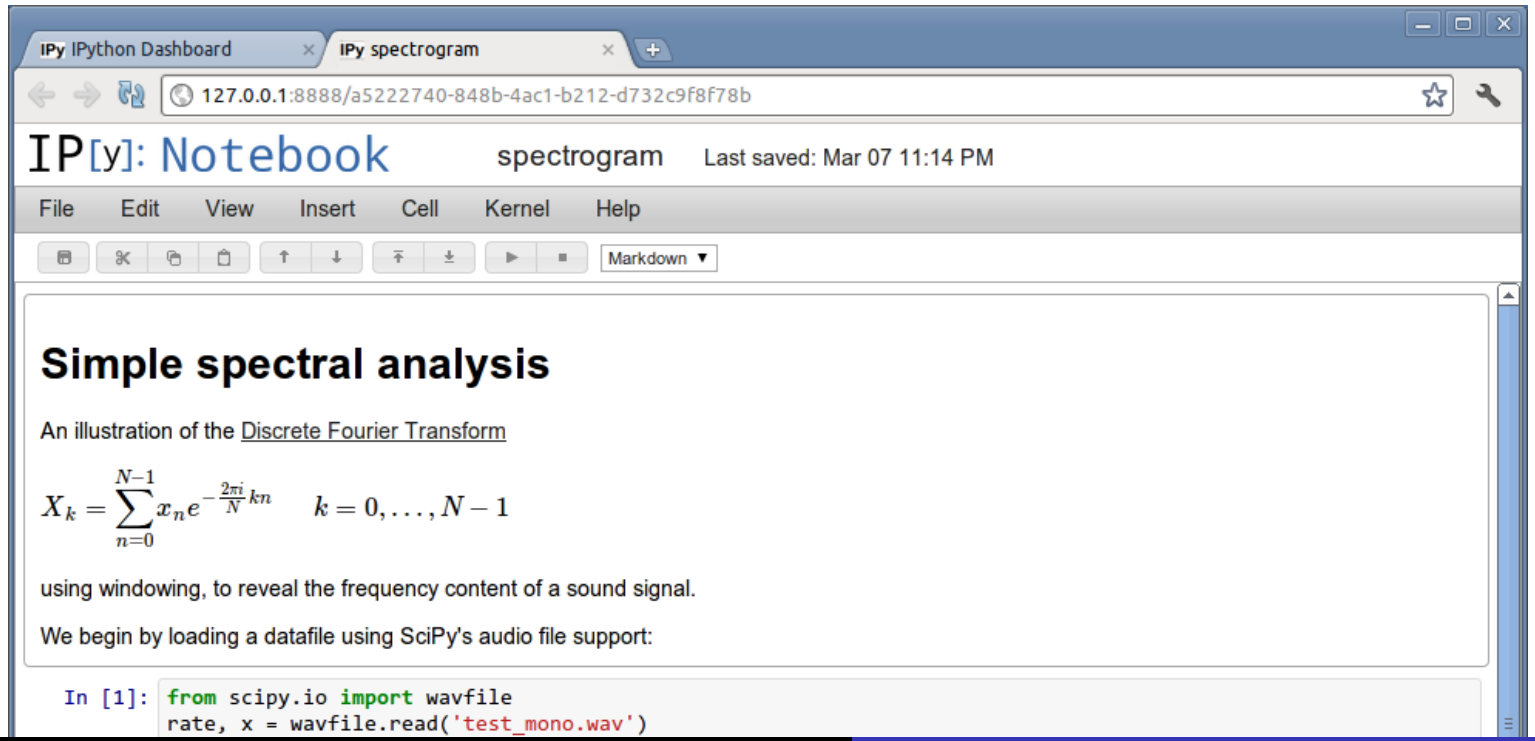
1. **Terminal**: `python`, `python3`

   ```
   >>> 1+1
   ```

2. **IPython** - opt. Terminal: `ipython3`, `ipython3 qtconsole`

   ```
   In[1]: %hist
   In[2]: hist?
   In[2]: help(abs)
   ```

3. **Im Browser** - IPython-Notebook: `ipython3 notebook`

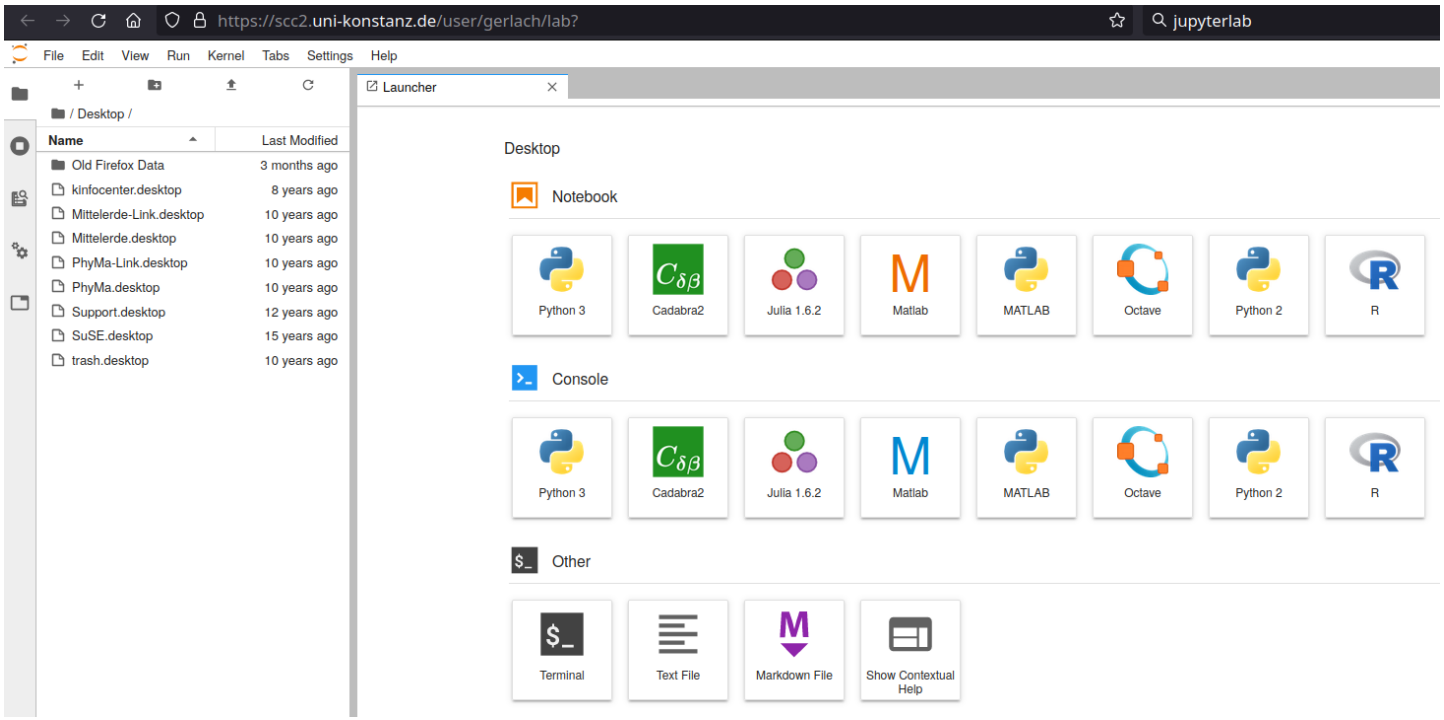


Verwendung - interaktiv

4 **IDE**: Spyder, Eric, PyCharm, ...

5 **Online (REPL)**: `https://www.python.org/shell/`, `https://repl.it/languages/python3`

6 **Jupyterlab/Jupyterhub**: `jupyter lab`, Online (SCCKN)



7 Von anderen Sprachen (CYTHON, ...)

"Hello, World" in Python

A) Interaktiv:

```
>>> print('Hello, World!')
Hello, World!
```

B) Python-Skript:

```
1 print('Hello, World!')
```

```
$ python3 skript.py
```

C) Shell-Skript:

```
1 #!/usr/bin/env python3
2
3 # Kommentar
4 print('Hello, World!')
```

```
$ chmod +x skript.py
$ ./skript.py
Hello, World!
```

Hilfe in Python

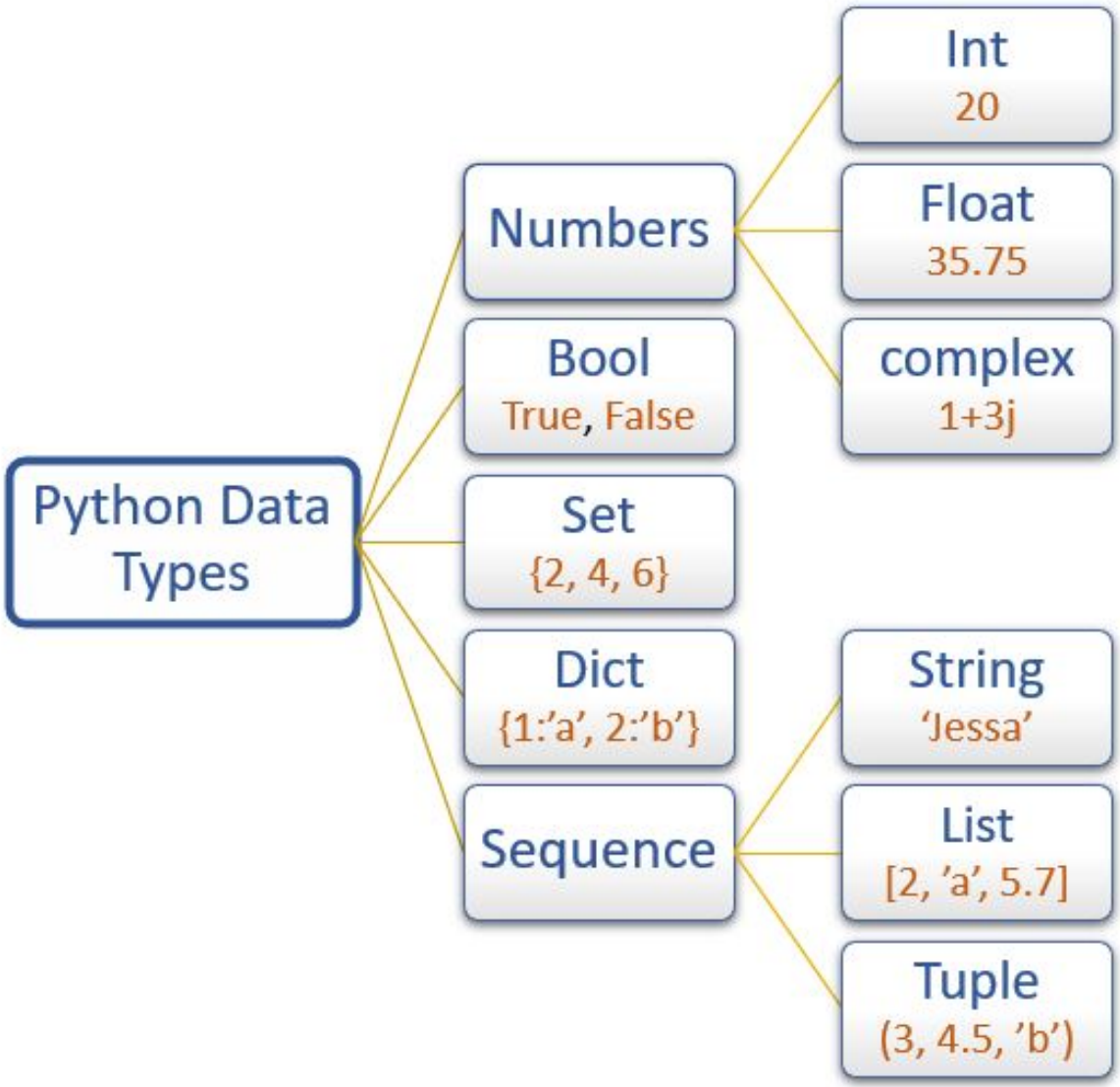
Eingebaute Hilfe:

```
1 >>> help()
2 help>
3 help> topics
4 ..
5 help> float # Thema, Funktion oder Modul
6 ..
7 >>> help(float)
```

oder

`https://www.python.org/doc/`

Datentypen in Python



Basistypen in Python

- **Integer** (ganzzahlig): `int`

  `>>> 1+2`

```
1 >>> 3/2
2 1.5 (Python 2)
3 1.5 (Python 3)
```

- **Fließkommazahl**: `float`

  `1.5, 2.`

- **Komplexe Zahl**: `complex`

  `1.5+0.4j`

- **Boolean**: `bool`

  `False, True`

Variablen in Python

```
1 a = 4
2 b = 7.
3 c = 1.+2.j
4 d = True
```

→ Datentyp wird automatisch zugewiesen. Deklaration unnötig!
Groß/Kleinschreibung, Spaces and Tabs

```
1 >>> type(c)
2 <class 'complex'>
```

Mehrere Variablen:

```
1 a,b = 1,2
2 a = b = 2.
3 a +=1; a -= 1; a *= 2; a /= 2
4 a,b = b,a
```

Variablenausgabe/-eingabe

```
1  print(a)
2  print(a, b)
3  print('Ergebnis:', a)
4  print(f'Ergebnis: {a}')
5
6  print('%e' % a)
7  # e - Exponent, f - Floating, g - Automatic
8  # (wie in C)
9
10 print("Wert: %s " % v)
11 print("Werte: %s and %s " % (v1, v2))
```

Eingabe:

```
1  input()     # warte auf ENTER
2  # Lese String
3  answer = input("please enter something")
4  # Wert einlesen
5  eval(input("please enter number"))
6
7  a = int(s)        # string -> int
8  s = str(a)        # numeric -> string
```

Erweiterte Datentypen: Container

Listen: Sammlung verschiedener Typen

```
1  a = [42, 1+2j, 'blau']
2
3  >>> len(a)
4  3
5  >>> a[0]    # Index beginnt bei 0 (wie C)
6  42
7  >>> a[-1]
8  'blau'
9  >>> a[1:]
10 [1+2j, 'blau']
11 >>> a[::-1]
12 ['blau', (1+2j), 42]
13 >>> a + a
14 [42, (1+2j), 'blau', 42, (1+2j), 'blau']
15 >>> b = [*a, *a]    # unpacking
```

Slicing: `a[start:stop:step]` ($start <= i < stop$)
`a[:]=3,4,5` (in place)
Achtung: `b=a` (gleiches Objekt!)

Erweiterte Datentypen: Container

String: unveränderbare Liste

```
1 s1 = "Hello," + "World"
2 s2 = "Hello " * 5
3 print(s1)
4 len(s2)
5 # Sonderzeichen: \:, \", \\
```

Tuples: unveränderbare Listen `a = (1,2,3)`
Mengen (set): ungeordnete, unique Listen

```
1 a = set('a','b','c')
```

Wörterbuch (dict): Key-Value-Liste

```
1 tel = {'Stefan':3825, 'Hans': 3815}
2 tel.keys()
3 tel.values()
4 tel.get('Stefan')
```

Kontrollstrukturen

if: Vergleich (==, !=, <, <=, > , >=)
(Einrückung!)

```
1  if i < 0:
2      i = 0
3  elif i == 0:
4      i = 1
5  else:
6      print(i)
```

Varianten:

```
1  if 0 < x <= 1:
2      ...
3  if e in a:
4      ...
```

Kontrollstrukturen

for: Schleife mit fester Anzahl an Durchläufen

```
1 for e in a:
2     print(e)
```

(Einrückung!)

```
1 for e in range(10):
2     print(e)
3
4 # range(10) : [0,..,9]
5 # range(6,15,3) : [6,9,12]
```

```
1 for index, item in enumerate(a):
2     print(index, item)
```

Kontrollstrukturen

Funktionale Programmierung:

```
1 zahlen = [1, 2, 3, 4, 5]
2 quadrate = [n**2 for n in zahlen]
3 print(quadrate)
```

while: Schleife mit unbekannter Anzahl an Durchläufen

```
1 b = 0
2 while b < 10:
3     print(b)
4     b += 2
```

Funktionen

```
1 def q(x):
2     return x**2
3
4 # q(2) : 4
5 # default parameter: f(a, b=0)
```

Funktionale Programmierung:

```
1 a = range(10)
2
3 map(q, a)
4 # [0,1,4,9,...,81]
```

Arbeiten mit Dateien

Lesen:

```
1 # open(FILENAME, MODE)
2 f = open('example.tmp', 'r')
3 for line in f:
4     print(line)
5 f.close()
```

Modes: r - readonly, w - write, r+ - read and write, a - append only

Schreiben:

```
1 f = open('example.txt', 'w')
2 f.write('hello, world')
3 f.close()
```

Zusammenfassung

Was ist Python?

- Open-Source, all platforms, portable
- High-level (further away from machine language, and closer to natural languages)
- Interactive
- Interpreted (Scripting)
- General purpose
- Multiparadigm (OO, Functional, ..)
- Minimalistic constructs
- Structured code (indentation)
- Dynamically typed
- Built-in advanced types (string, dictionary, list, ..)
- Powerful standard library and many extensions
- Widely used, excellent support

Standardbibliotheken

Import (versch. Möglichkeiten):

```
1 import bib
2 import bib as alias
3 from bib import *     # problematisch
4 from bib import function1, function2
```

Beispiele:

```
1 import numpy
2 numpy.array([1,2,3])
```

```
1 import numpy as np
2 np.array([1,2,3])
```

```
1 from numpy import array
2 array([1,2,3])
```

```
1 from script import fuction
2 function()  # function from script.py
```

Standardbibliotheken

- **sys**: argv, version, path, ..
- **os**: getcwd(), listdir(), mkdir(), remove(), ..
- **math**: pi, e, sin(), cos(), log(), exp(), ..
- **cmath**: für komplexe Zahlen
- **random**: Zufallszahlen und -verteilungen
- **tkinter**, **PyQt**: GUI (Tk, Qt)
- **numpy**: NumPy (Numerische Daten und Methoden)
- **scipy**: SciPy (Wissenschaftliche Methoden und Anwendungen)
- **sympy**: SymPy (Symbolisches Rechnen)
- **matplotlib**: Matplotlib (Grafische Darstellung)

```
1 In [1]: import math
2 In [2]: help(math)
3 In [3]: print(dir(math))
```

Standardbibliotheken - math

```
pi, e, sin(), cos(), log(), exp(), ..
x = 1_000_000
print(f'{x:_}')
```

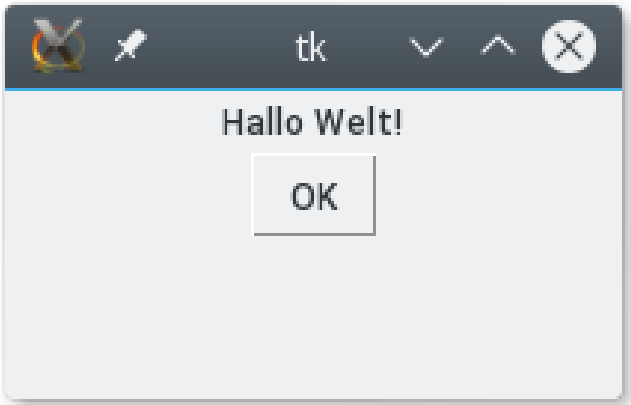
```
from math import fsum

l = [0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4]
print(sum(l))
print(fsum(l))
print(prod(l))  # Python 3.8
```

```
3.9999999999999996
4.0
0.00010485760000000006
```

```
fabs(x), ceil(x), floor(x), round(x, [-]n)
fmod(x, y), modf(x), frexp(x), ldexp(m, e)
isqrt()
copysign(x, y)
isclose(x, y, [rel_tol=..., abs_tol=...])
isinf(x), isnan(x), isfinite(x)
gcd(x, y)  # Python 3.9: gcd(x, y, ...), lcm(x, y, ..)
```

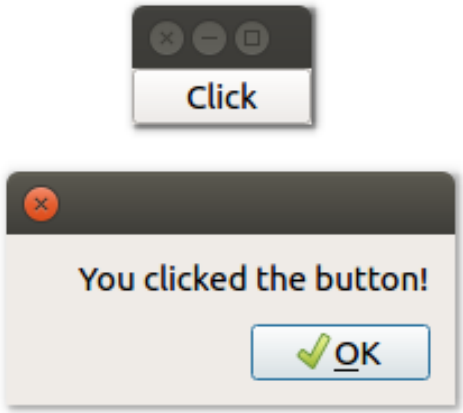
Standardbibliotheken - GUI (Tk)

```
1  from tkinter import Tk, Label, Button
2  fenster = Tk()
3  fenster.geometry("200x100")
4  label = Label(fenster, text="Hallo, Welt!")
5  label.pack()
6  def befehl():
7      fenster.destroy()
8  button = Button(fenster, text="OK", command=befehl)
9  button.pack()
10 fenster.mainloop()
```



Standardbibliotheken - GUI (Qt)

```
1  from PyQt5.QtWidgets import *
2
3  app = QApplication([])
4  button = QPushButton('Click')
5
6  def on_button_clicked():
7      alert = QMessageBox()
8      alert.setText('You clicked the button!')
9      alert.exec()
10
11 button.clicked.connect(on_button_clicked)
12 button.show()
13 app.exec()
```



NumPy - NumPy-Arrays

NumPy-Arrays: optimiert für Zahlen (int/float)

```
1  from numpy import array
2  v = array([1,2,3,4])
3  M = array([[1.,2.],[3.,4.]])
4
5  type(v)      # <class 'numpy.ndarray'>
6  v.shape      # (4,1)
7  M.shape      # (2,2)
8  v.size       # 4
9  M.size       # 4
10 v.dtype      # int64
11 M.dtype      # float64
12
13 v.reshape(2,2)     # array([[1, 2], [3, 4]])
14 v = array([1,2,3,4], dtype=complex)
```

→ speed.py

NumPy - Arrays erzeugen

```
1  from numpy import arange, linspace, diag
2
3  arange(0,1,0.1)      # [0,0.1,...,0.9]
4  linspace(0,1,6)      # [0,0.2,0.4,0.6,0.8,1.0]
5  diag([1,2])          # [[1,0],[0,2]]
6  zeros([2,3], dtype=int) # [[0 0 0], [0 0 0]]
7  ones([2,3], dtype=int)  # [[1 1 1], [1 1 1]]
8  eye(3, dtype=int)  # [[1 0 0], [0 1 0], [0 0 1]]
9  array[array > 0]  # boolean indexing
10 sarray = array.sort()    # Sortieren
```

```
1  from numpy import sin, cos
2
3  v=array([1,2,3])
4  sin(v)     # [ 0.84147098,  0.90929743,  0.14112001]
5  cos(v)     # ...
6
7  v+1, v*v, 2*x     # elementweise
```

Zugriff, Slicing: wie bei Python-Listen

NumPy - Matrix, etc.

```
1 from numpy import dot, matrix
2
3 dot(M, v)          # M.v
4 x=matrix(v).T      # transpose
5 x.T*x              # Skalarprodukt
6 dot(M, M)          # Matrixprodukt
```

- Mathematische Funktionen
- Statistik
- **numpy.polynomial**: Polynome
- **numpy.linalg**: Lineare Algebra
- **numpy.fft**: Fourier-Transformation
- **numpy.random**: Zufallszahlen/-verteilungen
- ...

SciPy - wissenschaftl. Funktionen

- **scipy.constants**: (physik.) Konstanten
- **scipy.special**: Spezielle Funktionen
- **scipy.stats**: Statistik
- **scipy.optimize**: Optimierung
- **scipy.interpolate**: Interpolation
- **scipy.linalg**: Lineare Algebra
- **scipy.integrate**: Numerische Integration
- **scipy.fft**: Fourier-Transformation
- **scipy.signal**: Signalanalyse
- ...

```
1 from scipy.special import binom, j0
2
3 print(binom(4,2))          # 6
4 print(j0(1))               # 0.765..
```

Matplotlib - Darstellung

Im Skript:

```
1 # in "python3":
2 #import matplotlib as mpl
3 #mpl.use("Qt5Agg")
4 import pylab as pl
5
6 x = pl.arange(0,7,0.01)
7 pl.plot(x,x**2)
8 pl.show()
```

Im Notebook:

```
$ ipython3 notebook
In[1]: %pylab inline
In[2]: x=arange(0,10)
In[3]: plot(x**2)
```

Matplotlib - Darstellung

```
1 xlabel("x-Achse")
2 title("$\sin \pi x$")
3 xlim(0,5)
4 plot(x,y,'--r')
```

Legende:

```
1 plot(x,y1,label="sin")
2 plot(x,y2,label="cos")
3 legend(loc='best')
```

SymPy - Symbolisches Rechnen

```
1 import sympy as sp
2
3 e = sp.sqrt(8)      # 2*sqrt(2)
4 e.evalf()          # 2.1828...
5 e.evalf(50)        # 50 Stellen
6 x,y = sp.symbols('x y')
7 e = x**2-2*y
8 e + x**2           # 2*x**2 - 2*y
9 e.subs(x, 1.5)     # Einsetzen
```

```
1 >>> sp.init_printing()    # nette Ausgabe
2       ___
3 2*\/ 2
4 >>> sp.latex(sp.sqrt(x**2))
5 2 \\sqrt{2}
```

SymPy - Algebra

- **expand**`((x+1)**2)`
  `x**2 + 2*x + 1`
- **factor**`(x**3 - x**2 + x - 1)`
  `(x-1)(x**2+1)`
- **collect**`(x*y + x*x + x, x)`
  `x**2 + x(y+1)`
- **cancel**`((x**2+2*x+1)/(x**2+x))`
  `(x+1)/x`
- **apart**`((x**2+2*x)/(x+1))`
  `x+1 - 1/(x+1)`
- **trigsimp**`((sin(x))**2 + (cos(x))**2)`
  `1`
- **expand_trig**`(sin(x+y))`
  `sin(x)*cos(y) + sin(y)*cos(x)`

SymPy - Calculus

- **limit**`(sin(x)/x, x, 0)`
  `1`
- **series**`(sp.cos(x))`
  `1 + x**2/2 + ...`
- `sp.sin(x).series(x,0,10)`
  `x - x**3/6 + ...`
- **diff**`(x**2,x)`
  `2*x`
- **solve**`(x**2-2, x)`
  `(-sqrt(2), sqrt(2))`
- **summation**`(2*i+1,(i,1,n))`
  `n**2 + 2*n`
- **integrate**`(x**2,x)`
  `x**3/3`
- **integrate**`(sp.sin(x),(x,0,10)).evalf()`
  `1.8390715...`

Anbindung an C/C++

CYTHON: Python mit C Datentypen → autom. Kompilieren als C-Code möglich!

```
1  import pyximport; pyximport.install()
2  # compile and execute helloworld.pyx (print("Hello, World!"))
3  import helloworld
```

pybind11: exposes C++ types in Python and vice versa
create Python bindings for existing C++ code

```
1  #include <pybind11/pybind11.h>
2
3  double add(double i, double j) {
4      return i + j;
5  }
6
7  PYBIND11_MODULE(example, m) {
8      m.doc() = "pybind11 example plugin"; // optional
9      m.def("add", &add, "A function which adds two numbers");
10 }
```

```
1  from example import add
2  print(add(23, 42))
```

Python in C/C++

```c
#define PY_SSIZE_T_CLEAN
#include <Python.h>

int main(int argc, char *argv[]) {
    wchar_t *program = Py_DecodeLocale(argv[0], NULL);
    if (program == NULL) {
        fprintf(stderr, "Error: cannot decode argv[0]\n");
        exit(1);
    }
    Py_SetProgramName(program);  // optional but recommended
    Py_Initialize();
    PyRun_SimpleString("from time import time,ctime\n"
                       "print('Today is', ctime(time()))\n");
    if (Py_FinalizeEx() < 0) {
        exit(120);
    }
    PyMem_RawFree(program);
    return 0;
}
```

```
gcc `python3-config --includes` -o test test.c `python3-config --libs`
```

Python Multithreading

```python
import logging
import time
import concurrent.futures

def function(name):
    logging.info("Thread %s: starting", name)
    time.sleep(2)
    logging.info("Thread %s: finishing", name)

logging.basicConfig(format="%(asctime)s: %(message)s", level=logging.INFO,
                    datefmt="%H:%M:%S")
with concurrent.futures.ThreadPoolExecutor(max_workers=3) as executor:
    executor.map(function, range(4))
```

```python
import multiprocessing
import time

def square(number):
    return sum(i * i for i in range(number))

numbers = [5_000_000 + x for x in range(20)]

start_time = time.time()

with multiprocessing.Pool() as pool:
    pool.map(square, numbers)

duration = time.time() - start_time
print(f"Duration {duration} seconds")
```