

Computerphysik II

Python Einführung

S. Gerlach

WiSe 2021/22

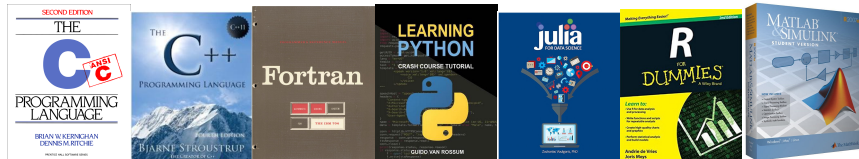
Universität
Konstanz



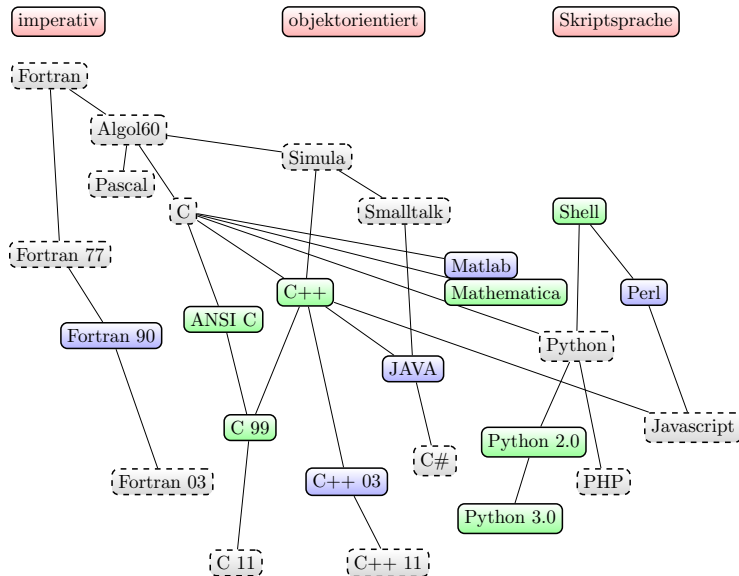
Programmiersprachen



In den Naturwissenschaften:



Programmiersprachen in der Wissenschaft



Programmiersprachen - Einteilung

Eigenschaften:

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

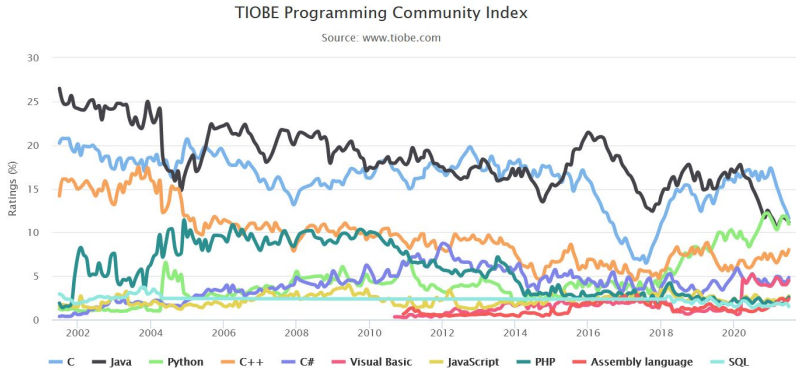
Anwendungen:

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

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)

aktuell: Python 2.7.18 (End-of-Life 2020), Python 3.6.15 - 3.10.0



https://en.wikibooks.org/wiki/Python_Programming

[http://www.scipy-lectures.org/_downloads/
ScipyLectures-simple.pdf](http://www.scipy-lectures.org/_downloads/ScipyLectures-simple.pdf)

Verwendung - interaktiv

1 **Terminal:** python, python3

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

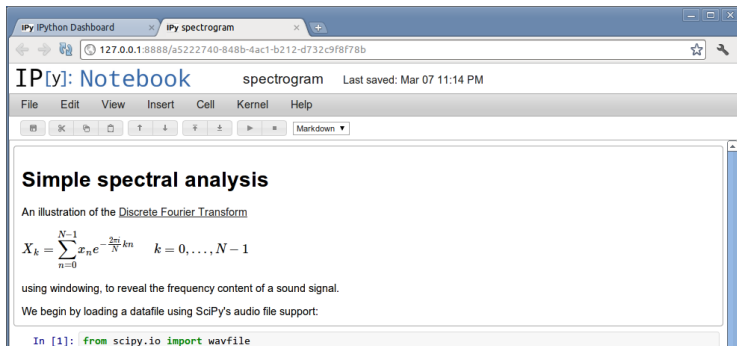
2 **IPython** - opt. Terminal: ipython3, ipython3 qtconsole

```
In[1]: %hist
```

```
In[2]: hist?
```

```
In[2]: help(abs)
```

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

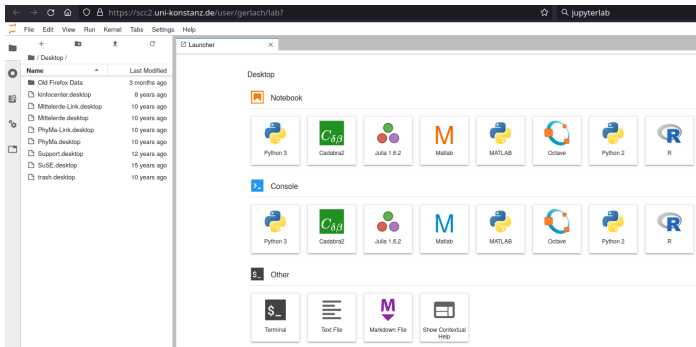


The screenshot shows a web browser window with two tabs: 'IPython Dashboard' and 'IPython spectrogram'. The active tab is 'IPython spectrogram', showing a URL: '127.0.0.1:8888/a5222740-848b-4ac1-b212-d732c9f8f78b'. The notebook interface has a title bar 'IP[y]: Notebook spectrogram' with a 'Last saved: Mar 07 11:14 PM' timestamp. Below the title bar is a menu bar with 'File', 'Edit', 'View', 'Insert', 'Cell', 'Kernel', and 'Help'. A toolbar contains icons for file operations, cell navigation, and execution, along with a 'Markdown' dropdown menu. The main content area displays a section titled 'Simple spectral analysis' with the text 'An illustration of the [Discrete Fourier Transform](#)'. It includes a mathematical formula for the Discrete Fourier Transform:
$$X_k = \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi i}{N} kn} \quad k = 0, \dots, N-1$$
 followed by the text 'using windowing, to reveal the frequency content of a sound signal.' and 'We begin by loading a datafile using SciPy's audio file support:'. At the bottom, a code cell shows the command:

```
In [1]: from scipy.io import wavfile
```


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!
```

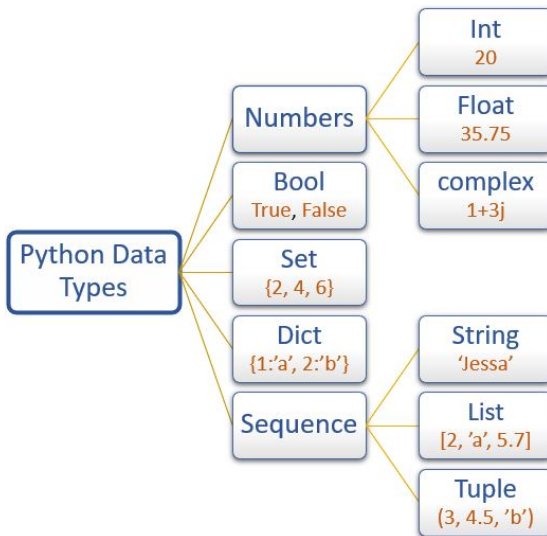
Eingebaute Hilfe:

```
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
5 print('%e' % a)
6 # e - Exponent, f - Floating, g - Automatic
7 # (wie in C)
8
9 print("Wert: %s" % v)
10 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 # Index beginnt bei 0 (wie C)
4 >>> a[0]
5 42
6 >>> a[-1]
7 'blau'
8 >>> a[1:]
9 [1+2j, 'blau']
```

Slicing: `a[start:stop:step]` ($start \leq i < stop$)

`len(a)`

Achtung: `b=a` (gleiches Objekt!), `a[:]=3,4,5` (in place)

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()
```

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      ...
```

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)
```

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
```

```
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]
```

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()
```

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

verschiedene Möglichkeiten:

```
1 import bib
2 import bib as alias
3 from bib import *      # schlecht
4 from bib import function
```

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])
```


- ▶ **sys**: argv, version, path, ..
- ▶ **os**: getcwd(), listdir(), mkdir(), remove(), ..
- ▶ **math**: sin(), cos(), log(), exp(), pi, e, ..
- ▶ **cmath**: für komplexe Zahlen
- ▶ **random**: Zufallszahlen
- ▶ **tkinter**: Tk (GUI)
- ▶ **numpy**: NumPy (Numerische Methoden)
- ▶ **scipy**: SciPy (wissenschaftliche 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 - GUI

```
1 from tkinter import *
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()
```



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)
```

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.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..
```

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)
```

```
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}
```

- ▶ **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** $(\sin(x+y))$
 $\sin(x)*\cos(y) + \sin(y)*\cos(x)$

- ▶ `limit(sin(x)/x, x, 0)`
1
- ▶ `series(sp.cos(x))`
 $1 + x^2/2 + \dots$
- ▶ `sp.sin(x).series(x,0,10)`
 $x - x^3/6 + \dots$
- ▶ `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 int add(int i, int 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 import example  
2 print(example.add(23, 42))
```

Python in C/C++

```
1 #define PY_SSIZE_T_CLEAN
2 #include <Python.h>
3
4 int main(int argc, char *argv[]) {
5     wchar_t *program = Py_DecodeLocale(argv[0], NULL);
6     if (program == NULL) {
7         fprintf(stderr, "Fatal error: cannot decode argv[0]\n");
8         exit(1);
9     }
10    Py_SetProgramName(program); /* optional but recommended */
11    Py_Initialize();
12    PyRun_SimpleString("from time import time, ctime\n"
13                       "print('Today is ', ctime(time()))\n");
14    if (Py_FinalizeEx() < 0) {
15        exit(120);
16    }
17    PyMem_RawFree(program);
18    return 0;
19 }
```

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

Python Multithreading

```
1 import logging
2 import time
3 import concurrent.futures
4
5 def function(name):
6     logging.info("Thread_%s:_starting", name)
7     time.sleep(2)
8     logging.info("Thread_%s:_finishing", name)
9
10 logging.basicConfig(format="%(asctime)s:_%(message)s", level=logging.INFO,
11                     datefmt="%H:%M:%S")
12 with concurrent.futures.ThreadPoolExecutor(max_workers=3) as executor:
13     executor.map(function, range(4))
```

```
1 import multiprocessing
2 import time
3
4 def square(number):
5     return sum(i * i for i in range(number))
6
7 numbers = [5_000_000 + x for x in range(20)]
8
9 start_time = time.time()
10
11 with multiprocessing.Pool() as pool:
12     pool.map(square, numbers)
13
14 duration = time.time() - start_time
15 print(f"Duration_{duration}_seconds")
```