Deutsche Hochschule
für Prävention und Gesundheitsmanagement
University of Applied Sciences

Programmierung 1

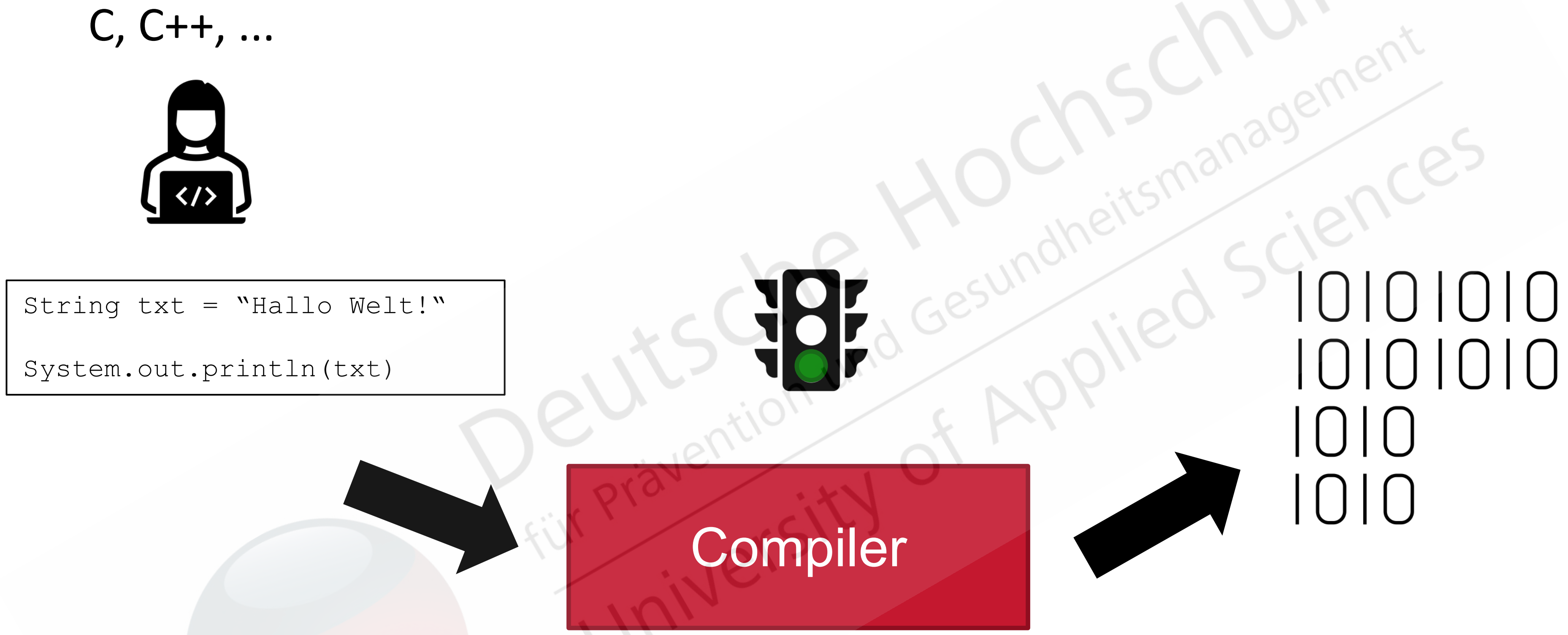
Speicherung und Interpretation von Informationen

- Computer bestehen aus **Hardware**
  - Prozessor (CPU)
  - Festplatte
  - Arbeitsspeicher
  - ...

- Ausgabegeräte
  - Bildschirm
  - Projektion
  - VR-Brille
  - ...

- Eingabegeräte
  - Maus und Tastatur
  - Joystick oder Gamepad
  - Gestensteuerung
  - Spracheingabe
  - ...

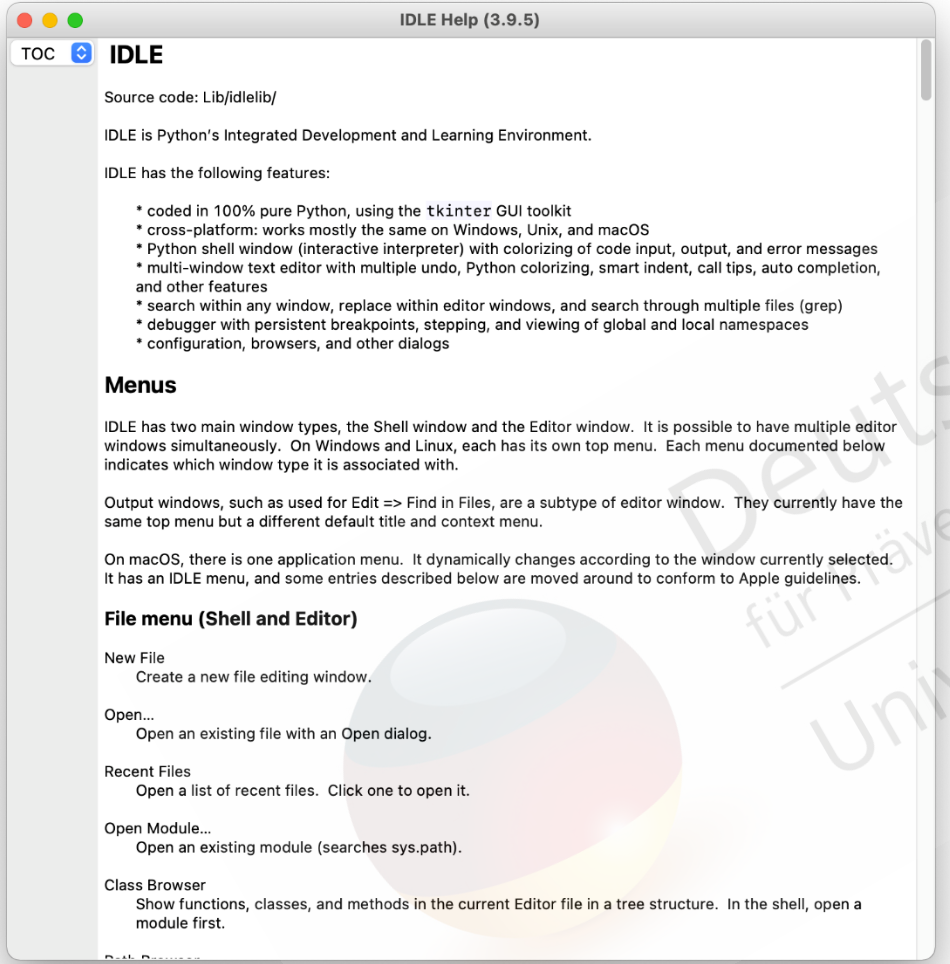
Compiler



```
String txt = "Hallo Welt!"

System.out.println(txt)
```

Hilfe-Funktionen in Python



IDLE Help (3.9.5)

TOC

IDLE

Source code: Lib/idlelib/

IDLE is Python's Integrated Development and Learning Environment.

IDLE has the following features:

* coded in 100% pure Python, using the `tkinter` GUI toolkit
* cross-platform: works mostly the same on Windows, Unix, and macOS
* Python shell window (interactive interpreter) with colorizing of code input, output, and error messages
* multi-window text editor with multiple undo, Python colorizing, smart indent, call tips, auto completion, and other features
* search within any window, replace within editor windows, and search through multiple files (grep)
* debugger with persistent breakpoints, stepping, and viewing of global and local namespaces
* configuration, browsers, and other dialogs

Menus

IDLE has two main window types, the Shell window and the Editor window. It is possible to have multiple editor windows simultaneously. On Windows and Linux, each has its own top menu. Each menu documented below indicates which window type it is associated with.

Output windows, such as used for Edit => Find in Files, are a subtype of editor window. They currently have the same top menu but a different default title and context menu.

On macOS, there is one application menu. It dynamically changes according to the window currently selected. It has an IDLE menu, and some entries described below are moved around to conform to Apple guidelines.

File menu (Shell and Editor)

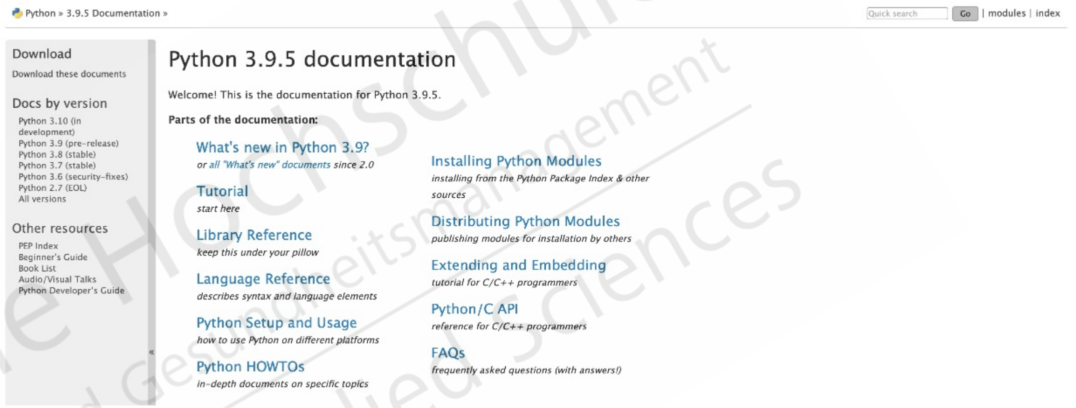
New File
    Create a new file editing window.

Open...
    Open an existing file with an Open dialog.

Recent Files
    Open a list of recent files. Click one to open it.

Open Module...
    Open an existing module (searches sys.path).

Class Browser
    Show functions, classes, and methods in the current Editor file in a tree structure. In the shell, open a module first.



Python » 3.9.5 Documentation » | Quick search | Go | modules | index

Download
Download these documents

Docs by version
Python 3.10 (in development)
Python 3.9 (pre-release)
Python 3.8 (stable)
Python 3.7 (stable)
Python 3.6 (security-fixes)
Python 2.7 (EOL)
All versions

Other resources
PEP Index
Beginner's Guide
Book List
Audio/Visual Talks
Python Developer's Guide

Python 3.9.5 documentation

Welcome! This is the documentation for Python 3.9.5.

Parts of the documentation:

What's new in Python 3.9?
or all "What's new" documents since 2.0

Tutorial
start here

Library Reference
keep this under your pillow

Language Reference
describes syntax and language elements

Python Setup and Usage
how to use Python on different platforms

Python HOWTOs
in-depth documents on specific topics

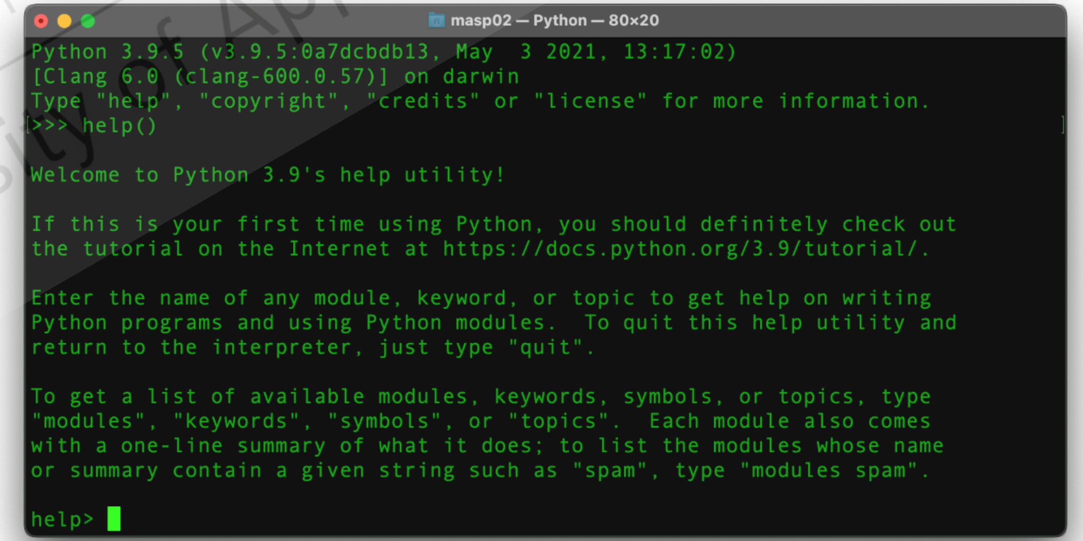
Installing Python Modules
installing from the Python Package Index & other sources

Distributing Python Modules
publishing modules for installation by others

Extending and Embedding
tutorial for C/C++ programmers

Python/C API
reference for C/C++ programmers

FAQs
frequently asked questions (with answers!)



masp02 — Python — 80×20

```
Python 3.9.5 (v3.9.5:0a7dcbdb13, May  3 2021, 13:17:02)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> help()

Welcome to Python 3.9's help utility!

If this is your first time using Python, you should definitely check out
the tutorial on the Internet at https://docs.python.org/3.9/tutorial/.

Enter the name of any module, keyword, or topic to get help on writing
Python programs and using Python modules.  To quit this help utility and
return to the interpreter, just type "quit".

To get a list of available modules, keywords, symbols, or topics, type
"modules", "keywords", "symbols", or "topics".  Each module also comes
with a one-line summary of what it does; to list the modules whose name
or summary contain a given string such as "spam", type "modules spam".

help> 
```

Arbeiten mit Strings

- Strings können unter Benutzung von folgenden Anführungszeichen angegeben werden:
  - einzelne Anführungszeichen (’)

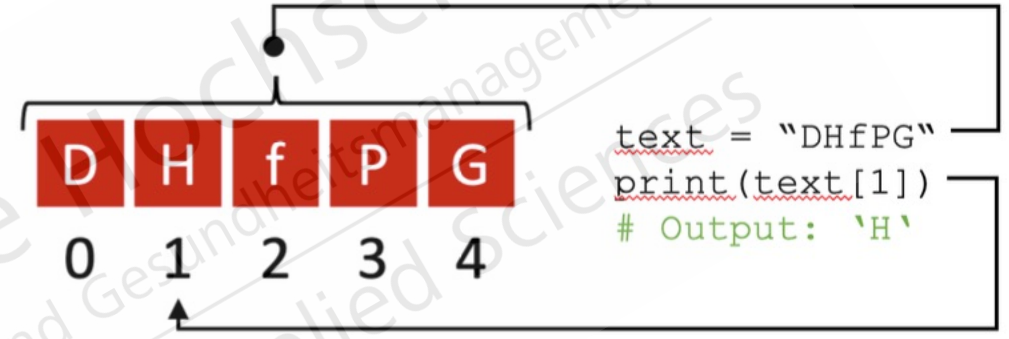
    ```
    ‘Dies ist ein String mit einfachen Hochkommata‘
    ```

  - doppelte Anführungszeichen (”)

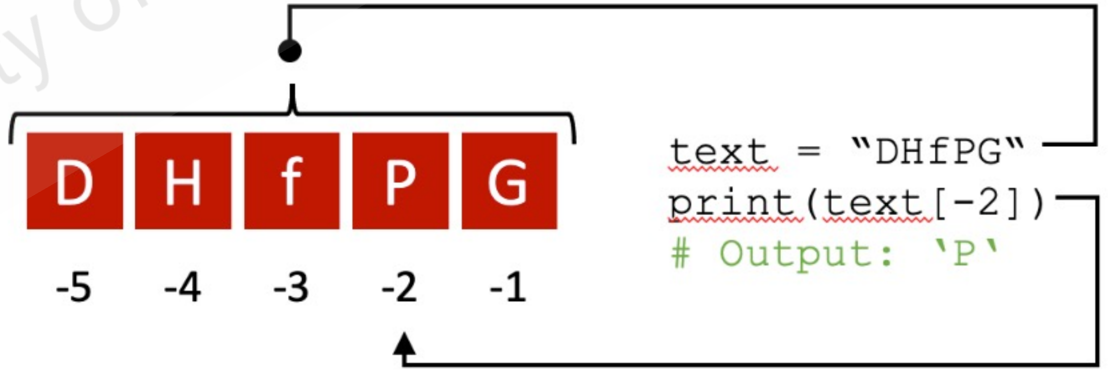
    ```
    ‘‘Mayers‘ Dackel heißt Waldi‘‘
    ```

  - dreifache Anführungszeichen (“‘)

    ```
    ‘‘‘
    String in dreifachen Anführungszeichen können
    auch über mehrere Zeilen gehen und ’einfache’
    und ”doppelte” Anführungszeichen enthalten.
    ‘‘‘
    ```



```
text = “DHfPG“
print(text[1])
# Output: ‘H‘
```



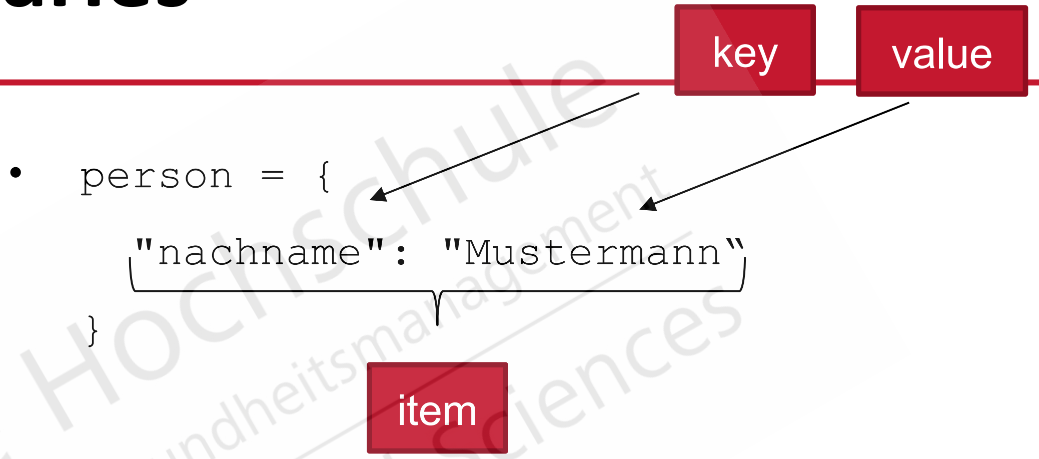
```
text = “DHfPG“
print(text[-2])
# Output: ‘P‘
```

Dictionaries

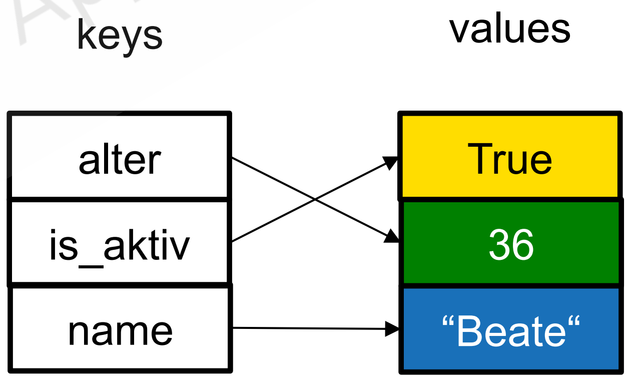
- Schlüssel:Wert-Paare (oder key:value-Paare)
- mit ‚{ }' umschlossen
- Schlüssel/Key immer in doppelten Anführungszeichen

  “key“ : value

- geordnet (seit Python 3.7) => Reihenfolge
- veränderbar
- keine Duplikate (Schlüssel sind eindeutig)



-
  ```
  person = {
      "nachname": "Mustermann"
  }
  ```

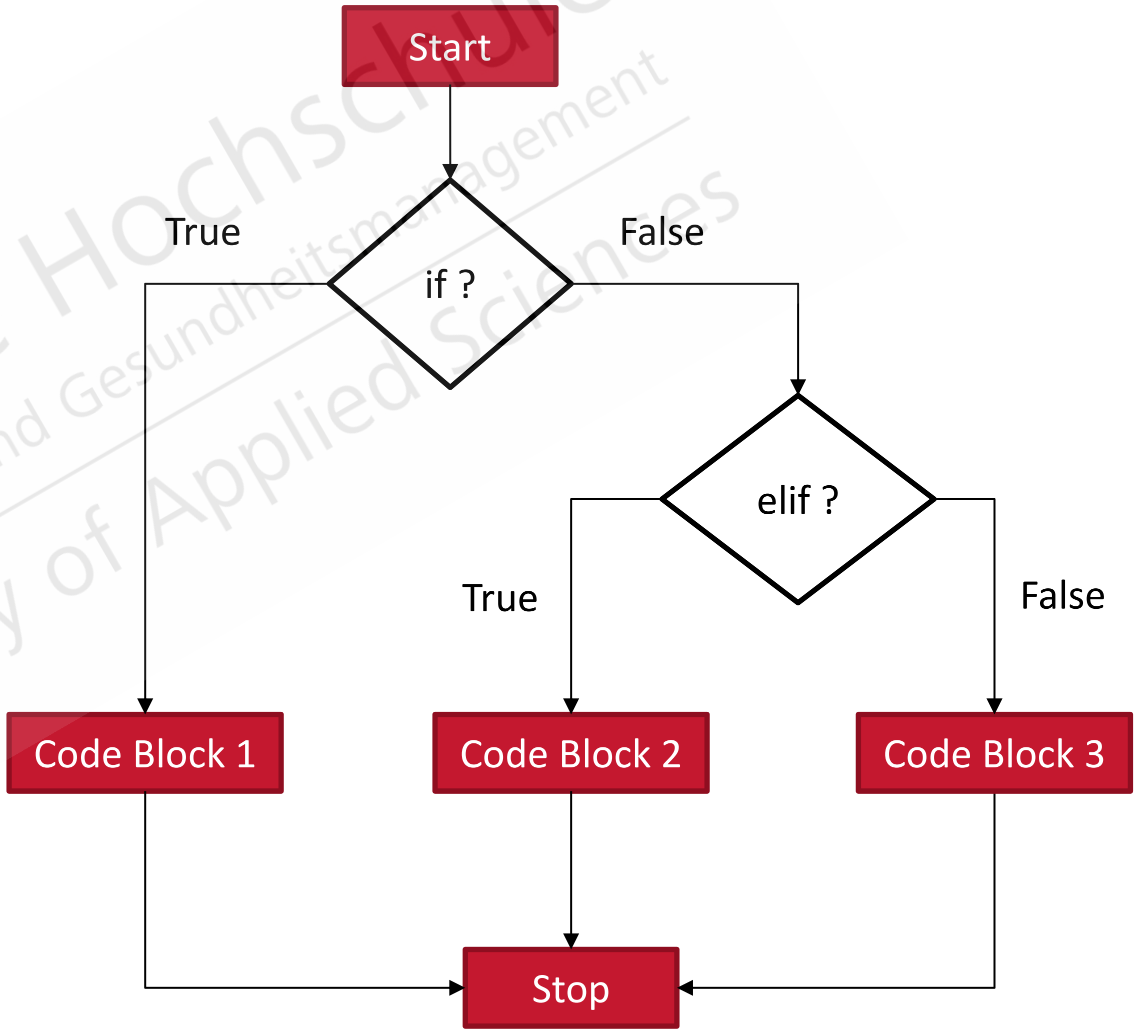


Fallunterscheidungen (if/elif/else)

-
  ```
  if Logical_Expression_1 :
      Code Block 1
  elif Logical_Expression_2 :
      Code Block 2
  else :
      Code Block 3
  ```

- Beispiel:

```
if n == 2:
    print(„Die Zahl ist 2.“)
elif n == 4:
    print(„Die Zahl ist 4.“)
else:
    print(„Die Zahl ist weder 2 noch 4.“)
print(„Ende.“)
```



for-Schleifen

„Für jede Frucht aus Früchte gilt: Gib die Frucht auf der Konsole aus!".

$$\downarrow$$

$$\forall\ \textbf{frucht} \in \textbf{fruechte} : print(\textbf{frucht}).$$

$$\downarrow$$

```
fruechte = ["Apfel", "Banane", "Kiwi"]

for frucht in fruechte:
    print(frucht)

# Output:
# 'Apfel'
# 'Banane'
# 'Kiwi'
```

```
for Iterator in Sequence:
    Codeblock
```

Rekursion

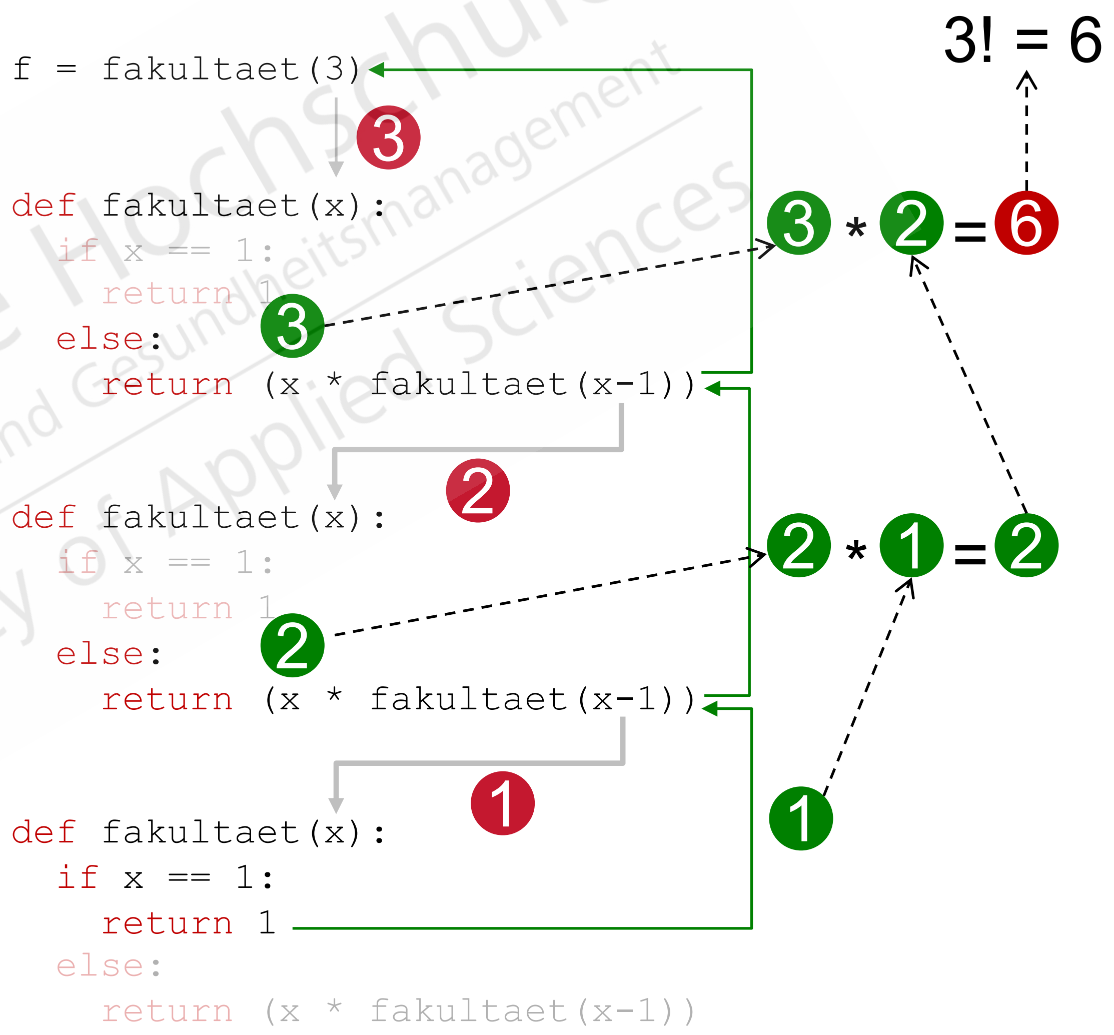
$$f(n) = n! = n \cdot (n-1) \cdot \ldots \cdot 2 \cdot 1$$

$$f(n) = \begin{cases} 1 & ,n = 1 \\ n * f(n-1) & ,n > 1 \end{cases}$$

```
def fakultaet(x):

  if x == 1:
    return 1
  else:
    return (x * fakultaet(x-1))
```

```
# 1. Aufruf mit dem Wert 3
fakultaet(3)
# 2. Aufruf mit dem Wert 3-1=2
3 * fakultaet(2)
# 3. Aufruf mit dem Wert 2-1=1
3 * (2 * fakultaet(1))
# Rückgabe des 3. Aufrufs, da 1 als Wert
3 * 2 * 1
# Rückgabe des 2. Aufrufs, da 2*1=2
3 * 2
# Rückgabe des 1. Aufrufs, da 3*2=6
6
```



```
f = fakultaet(3)

def fakultaet(x):
  if x == 1:
    return 1
  else:
    return (x * fakultaet(x-1))

def fakultaet(x):
  if x == 1:
    return 1
  else:
    return (x * fakultaet(x-1))

def fakultaet(x):
  if x == 1:
    return 1
  else:
    return (x * fakultaet(x-1))
```

Objektorientierte Programmierung (OOP)

Raum

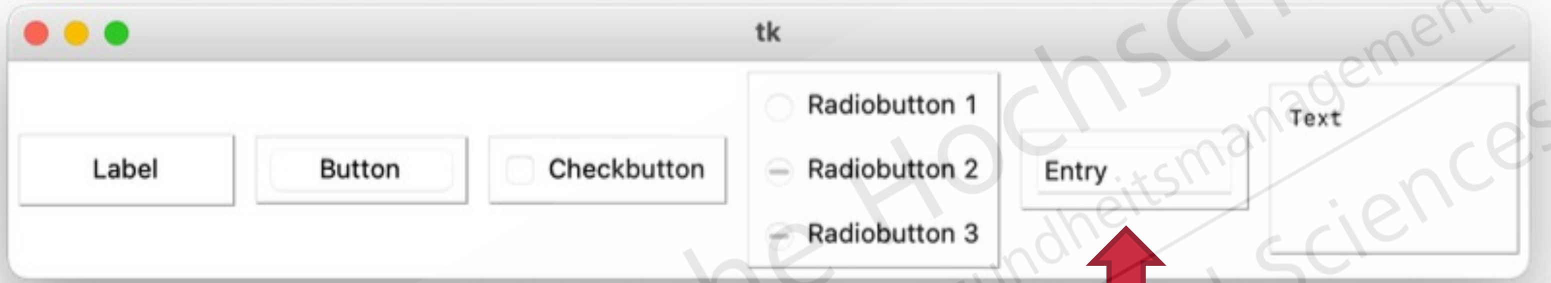
IP-Adressen

Musterstraße 12
12345 Musterstadt

123.255.21.0
192.168.0.100 (lokal)

+49 681 6855 150

Entry-Widget



```
import tkinter as tk
window = tk.Tk()
label = tk.Label(text="Name")
label.pack()
entry = tk.Entry()
entry.pack()
```

```
# löscht 1. Zeichen
entry.delete(0)
# löscht 4 Zeichen ab dem 1. Zeichen
entry.delete(0,4)
# löscht alles zwischen 1. und letztem Zeichen
entry.delete(0, tk.END)
```

```
entry.insert(0, "Welt")

entry.insert(0, "Hallo ")
```

"Welt"

"Hallo Welt"

```
>>> name = entry.get()
>>> name
```

→ „Hallo Welt!"