

Python & OOP

- Motivation
- Was wird benötigt?
- Einsatz in?
- Themen
- Wo stehen wir?
- Highlevel-Sprache
- C++ Compiler
- Python-Interpreter
- Struktur
- Variablen
- Vergleich Python mit C++
- REPL
- IDEs
- Imperative Programmierung
- Varianten rechnen?
- OOP, Klassen, Objekte
- OOP, Datencontainer
- Klasse „Kapital“
- Klasse, Objekte Erklärungen
- Klasse schrittweise 1 - 6
- kapitalClass.py
- Anhang IDEs
- Links

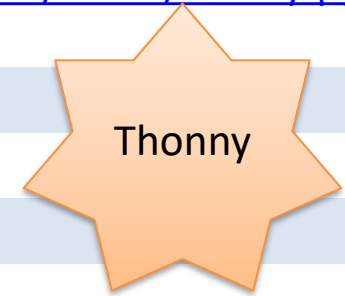
Motivation

- Man sollte etwas über Python machen.
- Was ist OOP?
- Gewitterwarner mit Pi-400 in Python.

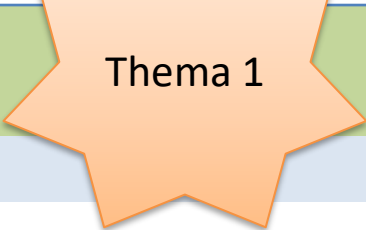
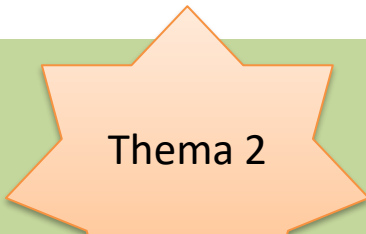


Was wird benötigt?



Online Python Editor	https://pynative.com/online-python-code-editor-to-execute-python-code/
Thonny portable 32-bit Windows	https://github.com/thonny/thonny/releases/download/v4.1.2/thonny-py38-4.1.2-windows-portable.zip
Referenz	https://quickref.me/python.html
Tutorials	https://docs.python.org/3/tutorial/index.html



Einsatz in?

Desktopanwendungen ...		C++	Python	
PC (Windows)		*.exe	*.py	
Raspberry Pi (Linux)			*.py	
GPIO-Anwendungen		C++	Python	
Raspberry Pi	Single-board Computer		*.py	
Arduino	Microcontroller	C++	Python	
		*.hex	(Firmata)	
ESP32/ESP8266	Microcontroller	C++	MicroPython	
		*.hex	*.py	
Raspberry Pi Pico	Microcontroller		*.py	

Themen

Python	<p>Einführung in Python.</p> <ul style="list-style-type: none">• Compiler vs. Interpreter.• Unterschiede C++ mit Python.• IDEs, integrierte Entwicklungsumgebungen.	 <p>Thema 1 Python & OOP</p>
OOP	<p>Objektorientiertes vs. imperatives Programmieren.</p> <ul style="list-style-type: none">• Beispiel ZinseszinsRechnung	
Python	<ul style="list-style-type: none">• Raspberry Pi & GPIO	
Python & Firmata	<ul style="list-style-type: none">• Arduino & Firmata	
MicroPython	<ul style="list-style-type: none">• PC/Raspberry Pi mit ESP8266/ESP32 (USB)• PC/Raspberry Pi mit Raspberry Pi Pico (USB)	 <p>Thema 2 MicroPython</p>
OOP & MicroPython	<p>Objektorientiertes Programmieren.</p> <ul style="list-style-type: none">• Ampelsteuerung	
Java-Hamster	<p>Das Java-Hamster Modell zur Einführung in OOP z.B. mit Python.</p>	


Wo stehen wir?

MS-DOS QBasic 1.1

```
DOSBox SVN-lfn (GNU GPL), CPU speed: 3000 cycles, Frameskip 0, LFN auto
Datei Bearbeiten Ansicht Suchen Ausführen Debug
ZINSEN.BAS
CLS
INPUT "Kapital="; A
INPUT "Zins="; Z
INPUT "Jahre="; N
E = A * (1 + Z / 100) ^ N
PRINT "Endkapital="; E
END
```

Python 3.11.4

```
1 strKapital = input('Kapital =')
2 strZinsen  = input('Zinsen  =')
3 strJahre   = input('Jahre   =')
4
5 kapital = float(strKapital)
6 zinsen  = int(strZinsen)
7 jahre   = int(strJahre)
8
9 endkapital = kapital * ( 1 + zinsen / 100 ) ** jahre
10
11 print('Kapital=', endkapital)
```



Einfacher
Einstieg

Wo stehen wir?

C++
Arduino Uno

```
1 void setup() {  
2     // initialize digital pin LED_BUILTIN as an output.  
3     pinMode(LED_BUILTIN, OUTPUT);  
4 }  
5  
6 // the loop function runs over and over again forever  
7 void loop() {  
8     digitalWrite(LED_BUILTIN, HIGH);    // turn the LED on (HIGH is the voltage level)  
9     delay(1000);                        // wait for a second  
10    digitalWrite(LED_BUILTIN, LOW);     // turn the LED off by making the voltage LOW  
11    delay(1000);                        // wait for a second  
12 }
```

MicroPython v1.20.0
ESP32



```
1 from machine import Pin           # module machine; class (type) Pin  
2 from time import sleep           # module time; function sleep  
3  
4 # label on board: Pin 2 => GPIO2 = 2  
5 LedPort = 2  
6 ledPin = Pin( LedPort, Pin.OUT)   # object ledPin derived from class Pin  
7  
8 while True:  
9     ledPin.value(not ledPin.value()) # calling method .value() from object ledPin  
10    sleep(0.5)                    # calling method sleep()  
11
```

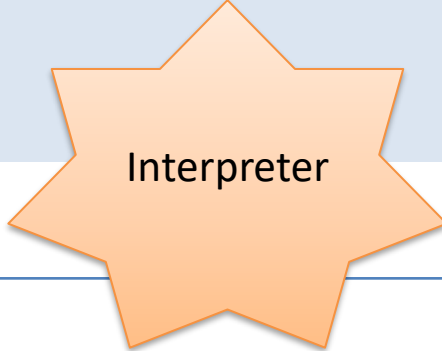
Highlevel-Sprache

- Python ist eine „höhere Programmiersprache“.
- Grobe Rangfolge:
1. Maschinensprache (Assembler)
 2. C/C++ (Lowlevel)
 3. Python (Highlevel)



Highlevel

Python vs. C++

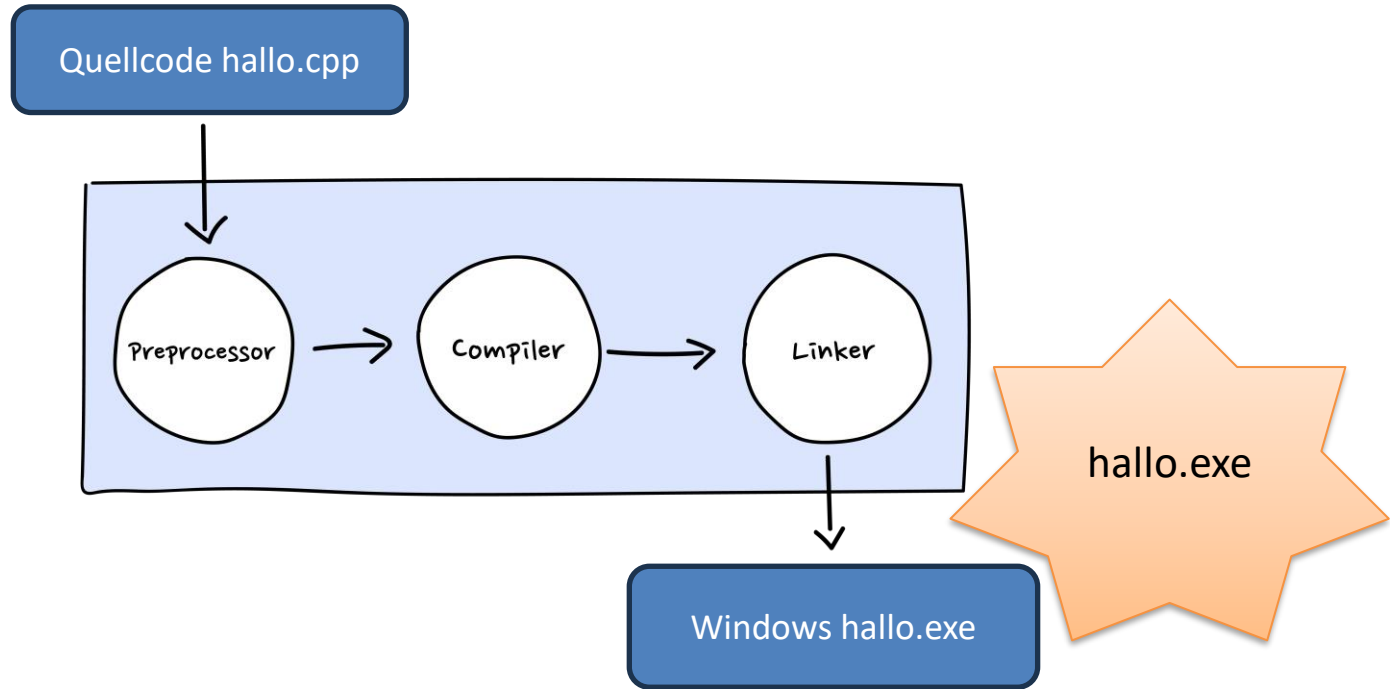


Interpreter

- Der Python-Interpreter selbst wurde in „C“ geschrieben.
- Im Unterschied zu C/C++ wird Python-Code nicht in ein ausführbares Programm übersetzt (kompiliert).
- Python benötigt einen Interpreter.
- Fehler werden daher erst zur Laufzeit (RuntimeError) erkannt.
- Python-Skripte sind Textdateien mit der Erweiterung (*.py).
- Der Interpreter erzeugt einen Bytecode mit der Erweiterung (*.pyc).

C++ Compiler

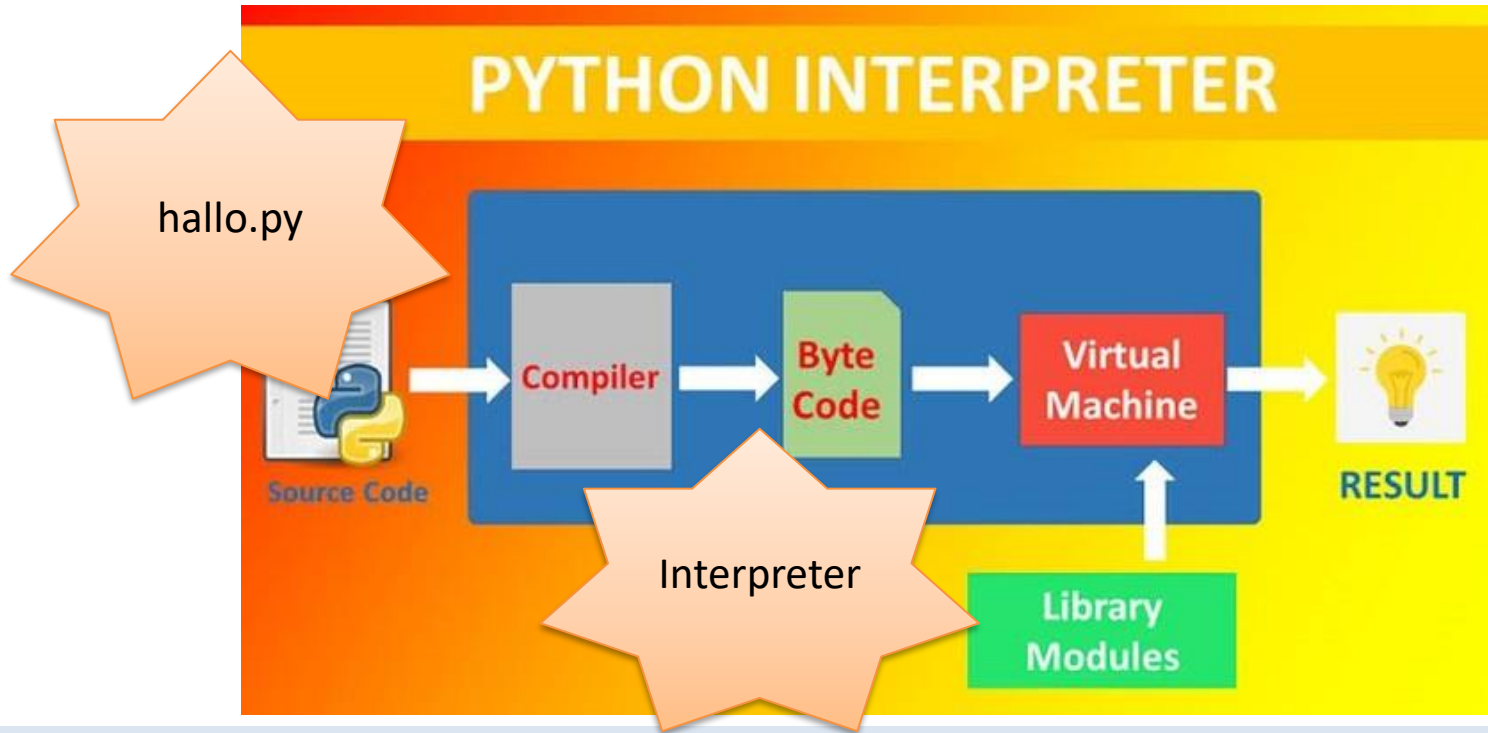
Der Quellcode in C++ wird über Kompilieren und Linken zur ausführbaren Datei.



Quelle: <https://www.codecademy.com/article/cpp-compile-execute-locally>

Python-Interpreter

Python benötigt einen Interpreter, der das Python-Script zur Laufzeit auswertet und anwendet.



Quelle: <https://www.datasciencecentral.com/how-python-interpreter-works-1/>

Struktur

Lesbarkeit

- Python wurde für die Lesbarkeit von Programmen optimiert.

Zeilenschaltung

- Die Zeilenschaltung (CR) schließt eine Anweisung ab, kein Semikolon (;) wie in C++.

Einrückungen

- Python verwendet Einrückungen (whitespaces), um einen Block abzugrenzen und keine ({ ... }) wie in C++.

Einrückungen

4 Leerzeichen

```
num1 = 50  
num2 = 100
```

```
if num1 > num2:
```

```
    print(num1, 'greater than', num2)
```

```
elif num2 > num1:
```

```
    print(num2, 'greater than', num1)
```

```
else:
```

```
    print('Both numbers are equal')
```

Der Doppelpunkt geht
einem Block voraus


Code Block 1

Code Block 2

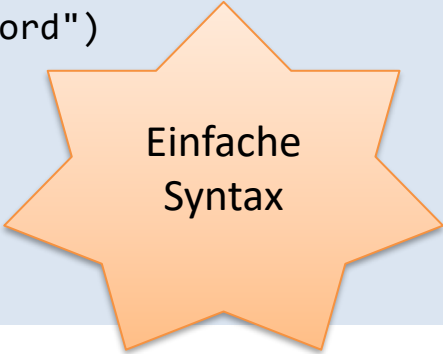
Code Block 3

Quelle: <https://pynative.com/get-started-with-python/#h-using-blank-lines-in-code>

Variablen

Variablen	<ul style="list-style-type: none">• Python erfordert keine Variablen-Deklaration.• Variablen sind keinem Datentyp fest zugeordnet, d.h. einer Variablen kann abwechselnd eine Zahl als auch eine Zeichenkette zugeordnet werden (Mutierbare Datentypen).	
Eine Variable wird bei der ersten Wertzuweisung deklariert.	<pre>x = 5 Y = "EBW"</pre>	
Variablen werden ohne Typangabe deklariert.	<pre>x = 4 # ist vom Typ Ganzzahl (int) y = 3.2 # ist vom Typ Dezimalzahl (float) Z = "EBW" # ist vom Typ Zeichenkette (str)</pre>	
Casting: Datentypen umwandeln.	<pre>strZins = input("Jahre=") # „input“ liefert immer eine Zeichenkette zins = int(strZins) # Zeichenkette in Ganzzahl umwandeln</pre>	
Mutable: Datentypen ändern.	<pre>zins = input("Jahre=") # „zins“ enthält eine Zeichenkette zins = int(zins) # „zins“ enthält eine Ganzzahl</pre>	

Vergleich Python mit C++

Python	C++
<pre>hello.py print("Hello Word")</pre>  <p>Einfache Syntax</p>	<pre>hello.cpp #include <iostream> using namespace std; int main() { cout << "Hello World!"; return 0; }</pre>
Online-Editor: https://pynative.com/online-python-code-editor-to-execute-python-code/	Online-Editor: https://www.w3schools.com/cpp/trycpp.asp?filename=demo_helloworld

Vergleich Python mit C++

Python

ifStatement.py

```
a = 33  
b = 200
```

```
if b > a:  
    print("b is greater than a")
```

Block durch
einrücken

<https://pynative.com/online-python-code-editor-to-execute-python-code/>

C++

demo_if2.cpp

```
#include <iostream>  
using namespace std;
```

```
int main()  
{  
    int x = 20;  
    int y = 18;  
    if (x > y) {  
        cout << "x is greater than y";  
    }  
    return 0;  
}
```

Block durch
{ ... }

https://www.w3schools.com/cpp/trycpp.asp?filename=demo_if2

REPL

REPL

Shell: Read-Eval-Print Loop

- Python kommt mit einer Python-Kommandozeile: REPL genannt.
- Statt der Befehle im Script können hier alle Befehle interaktiv getestet werden.

Thonny IDE



REPL

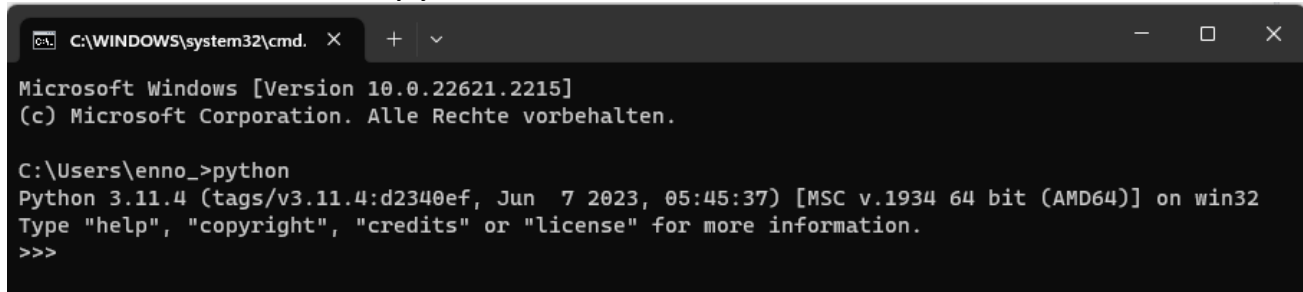
REPL Shell

Shell x

```
Python 3.10.11 (C:\Users\Public\thonny-4.1.1-windows-portable\python.exe)  
>>>
```

REPL in Windows (Python installiert)

Windows + R > cmd > python



```
C:\WINDOWS\system32\cmd. x + v  
Microsoft Windows [Version 10.0.22621.2215]  
(c) Microsoft Corporation. Alle Rechte vorbehalten.  
  
C:\Users\enno_>python  
Python 3.11.4 (tags/v3.11.4:d2340ef, Jun 7 2023, 05:45:37) [MSC v.1934 64 bit (AMD64)] on win32  
Type "help", "copyright", "credits" or "license" for more information.  
>>>
```

IDEs

IDE	Integrated Development Environment.
(Online Python Editor)	https://pynative.com/online-python-code-editor-to-execute-python-code/ https://www.programiz.com/python-programming/online-compiler/
Thonny	https://thonny.org/ <ul style="list-style-type: none">• Kommt mit „Python 3.10.11“ vorinstalliert.• Portable Version herunterladen und entpacken.• In den Editor-Optionen alle „Automatic & Request-Optionen“ aktivieren, um Code-Vervollständigung zu erhalten.• Thonny kann für Python & MicroPython genutzt werden!• Ist auch auf Raspberry Pi installiert.• Empfehlenswert!
Python installieren	https://www.python.org/ftp/python/3.11.4/python-3.11.4-amd64.exe <ul style="list-style-type: none">• Unbedingt Python zur Umgebungsvariable „path“ hinzufügen!
Python IDE	IDLE Shell: Ein einfacher Editor für die ersten Schritte. <ul style="list-style-type: none">• Windows Suche (neben Start) > IDLE eingeben & starten.



Imperative Programmierung

lt. Wikipedia

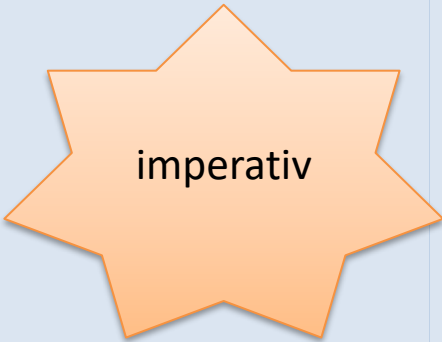
- Ist ein Programmierparadigma, nach dem ein Programm aus einer Folge von Anweisungen besteht, die vorgeben, in welcher Reihenfolge was vom Computer getan werden soll.
- Früher die klassische Art des Programmierens, zum Beispiel in: Fortran, Basic, Pascal, und C.

Prinzip

- Der Code wird von Zeile zu Zeile ausgeführt.
„First do this and next do that“.

Python

kapital.py



imperativ

```
# Imperative Programmierung anhand einer ZinseszinsRechnung
kapital = 5000
zinsen  = 5
jahre   = 10

endkapital = kapital * (1 + zinsen / 100) ** jahre
print ('Kapital =', endkapital, 'Euro')

# weitere alternative Rechnungen möglich durch
# deklarieren von weiteren Variablen...
```

Varianten rechnen?

Kapital variieren	<code>kapital_1 = 5000</code> <code>kapital_2 = 4500</code>
Zinsen variieren	<code>zinsen_1 = 5</code> <code>zinsen_2 = 2.5</code>
Jahre variieren	<code>jahre_1 = 10</code> <code>jahre_2 = 5</code>
Formel variieren	<code>endkapital_1 = kapital_1 * (1 + zinsen_1 / 100) ** jahre_1</code> <code>endkapital_2 = kapital_2 * (1 + zinsen_2 / 100) ** jahre_2</code>
Fehler oder nicht?	<code>endkapital_3 = kapital_1 * (1 + zinsen_1 / 100) ** jahre_2</code>
Nachteile	<ul style="list-style-type: none">Die zu einer Rechnung zugehörigen Variablen sind nicht aneinander gebunden, können daher frei kombiniert werden.Vielzahl von Variablendeklarationen.



OOP, Klassen, Objekte

① Wunsch: Karteikasten der Computersammlung



② Musterkarteikarte

PC-Sammlung

Typ _____
Zeit _____
Hersteller _____
Bezeichnung _____

Bleistift &
Radiergummi

③ Musterkarteikarte anwenden & Daten eingeben

Karte A

Typ Taschenrechner
Zeit 70iger
Hersteller Sharp
Bezeichnung TX20

Bleistift &
Radiergummi

Karte B

Typ Homecomputer
Zeit 80iger
Hersteller Apple
Bezeichnung Ilc

Bleistift &
Radiergummi

OOP-Sprache



Klasse

Attribute (ohne Werte)

Methoden

Ein Objekt der Klasse

Attribute mit Werten


Methoden

Ein Objekt der Klasse

Attribute mit Werten

Methoden

OOP, Datencontainer

Ansatz	Datenkapselung: Variablen in einem „Container“ abgrenzen.	
Container „Rechnung“	Variablen „kapital, zinsen, jahre“ einer Rechnungsvariante zuordnen.	
Rechnungsvariante:	Variablen:	Kombinieren zu:
endkapital_1	kapital zinsen jahre	endkapital_1.kapital = 5000 endkapital_1.zinsen = 5 endkapital_1.jahre = 10
		
endkapital_2	kapital zinsen jahre	endkapital_2.kapital = 4500 endkapital_2.zinsen = 5 endkapital_2.jahre = 10
Vorteil	Wir kommen mit einem Satz von Variablen aus. Der Container wird durch den vorangestellten Bezeichner gebildet.	
Allgemeine Zuweisung	bezeichner.variable = wert	
In OOP-Sprache	object.attribut = wert Einführung des Punktoperators (.) als Zugriffsoperator.	

Klasse „Kapital“

Reale Welt:

Aufgabe

Variablen

Formel

In einer Zinseszinsrechnung sollen die Laufzeiten variiert werden.

kapital, zinsen, jahre

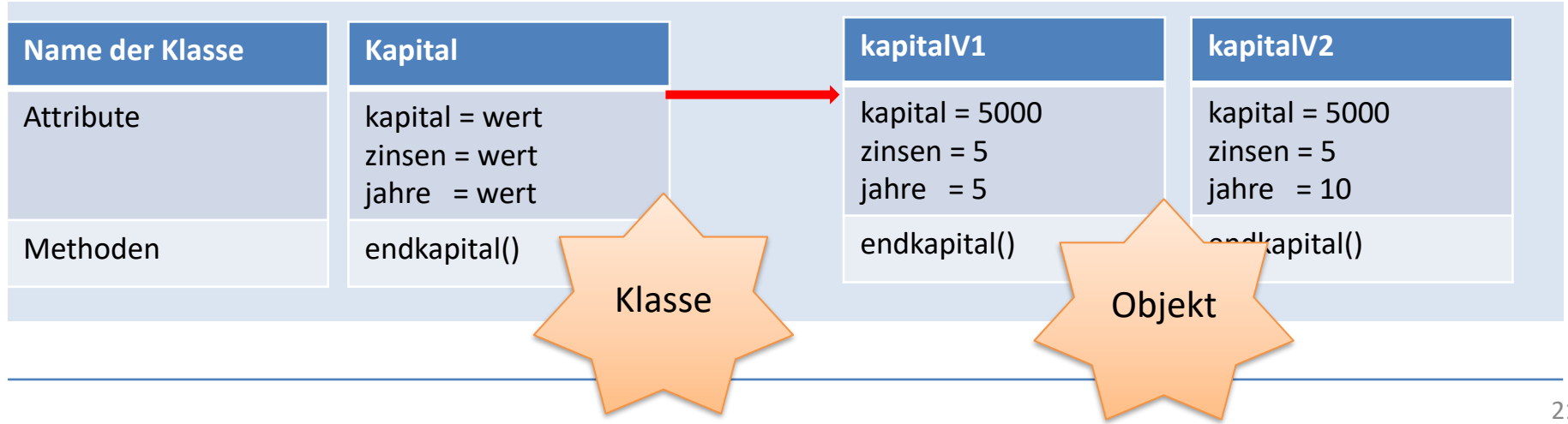
$\text{endkapital} = \text{kapital} * (1 + \text{zinsen} / 100) ** \text{jahre}$

Erster Schritt

Reale Welt in Python als Datenstruktur innerhalb einer „Klasse“ abbilden.

Klassendiagramm „ist Bauplan für“

→ Objekte der Klasse „Kapital“



Klasse, Objekte Erklärungen

Name der Klasse	Kapital
Attribute	kapital = wert zinsen = wert jahre = wert
Methoden	endkapital()

Die Klasse wird als Blaupause für Objekte verstanden.

Die Klasse enthält keine Werte!

Innerhalb der Klasse werden die Variablen definiert.

Die Methoden dienen zur Bearbeitung der Daten.
Hinter „endkapital()“ steht hier die Zinseszinsformel.

kapitalV1	kapitalV2
kapital = 5000 zinsen = 5 jahre = 5	kapital = 5000 zinsen = 5 jahre = 10
endkapital()	endkapital()

Alternative Sprechweisen:

Ein Objekt von der Klasse ableiten.

Ein Exemplar der Klasse erzeugen.

Eine Instanz der Klasse bilden.

Klasse Schritt 1.

1. Klasse definieren
2. „pass“: Platzhalter

```
# Beginn der Klasse
class Kapital:
    pass
# Ende der Klasse
# Beginn Python Script
kapitalV1 = Kapital()
```

3. Objekt „kapitalV1“ der Klasse „Kapital“ erzeugen.

4. Methode „welcome()“ hinzufügen

```
# Beginn der Klasse
class Kapital:
    def welcome(self):
        print('Hallo EBW')
# Ende der Klasse
# Beginn Python Script
kapitalV1 = Kapital()
kapitalV1.welcome()
```

5. Objekt „kapitalV1“ ruft Methode „welcome()“ auf.

Klasse Schritt 2.

6. Objekt „kapitalV2“ der Klasse „Kapital“ hinzufügen.

```
# Beginn der Klasse
class Kapital:
    def welcome(self):
        print('Hallo EBW')
# Ende der Klasse
# Beginn Python Script
kapitalV1 = Kapital()
kapitalV2 = Kapital()
```

7. Objekt „kapitalV2“ ruft Methode „welcome()“ auf.

```
# Beginn der Klasse
class Kapital:
    def welcome(self):
        print('Hallo EBW')
# Ende der Klasse
# Beginn Python Script
kapitalV1 = Kapital()
kapitalV2 = Kapital()
kapitalV1.welcome()
kapitalV2.welcome()
```


Klasse Schritt 3.

8. Methode “__init__()” hinzufügen. Sie definiert Attribute und initialisiert diese.

```
# Beginn der Klasse
class Kapital:
    def welcome(self):
        print('Hallo EBW')
    def __init__(self, version):
        self.version = version
        self.kapital = 0
        self.zinsen = 0
        self.jahre = 0
# Ende der Klasse
```

9. Objekte an “__init__()” anpassen.

10. Dem Objekt „kapitalV1“ über dem Punktoperator das Attribut „kapital“ zuordnen und Wert zuweisen.

11. Dem Objekt „kapitalV2“ ebenfalls Werte zuweisen.

```
# Beginn Python Script
kapitalV1 = Kapital('Rechnung V1')
kapitalV2 = Kapital('Rechnung V2')
# ...
kapitalV1.kapital = 5000
kapitalV1.zinsen = 5
kapitalV1.jahre = 10

kapitalV2.kapital = 5000
kapitalV2.zinsen = 5
kapitalV2.jahre = 5
```

Klasse Schritt 4.

12. Methode „zinsRechnung()“ hinzufügen. Sie gibt Ergebnis der Zinseszinsrechnung zurück.

```
# Beginn der Klasse
class Kapital:
    def welcome(self):
        print('Hallo EBW')
    def __init__(self):
        self.kapital = 0
        self.zinsen = 0
        self.jahre = 0

    def zinsRechnung(self):
        return self.kapital * ( 1 + self.zinsen / 100 ) ** self.jahre
# Ende der Klasse
```

13. Objekt „kapitalV1“ ruft Methode „zinsRechnung()“ auf.
14. Objekt „kapitalV2“ ebenfalls.

```
# Beginn Python Script
# ...
kapitalV1.kapital = 5000
# ...
kapitalV2.kapital = 5000
# ...

print('Kapital V1=', kapitalV1.zinsRechnung())
print('Kapital V2=', kapitalV2.zinsRechnung())
```

Klasse Schritt 5.

15.	Set-Methode „setKapital()“ zum zu Weisen eines Wertes an das Attribut „self.kapital“.	# Beginn der Klasse class Kapital: # ... def zinsRechnung(self): return self.kapital * (1 + self.zinsen / 100) ** self.jahre def setKapital(self, kapital): self.kapital = kapital def setZinsen(self, zinsen): self.zinsen = zinsen def setJahre(self, jahre): self.jahre = jahre # Ende der Klasse
16.	Set-Methode „setZinsen()“.	
17.	Set-Methode „setJahre()“.	
18.	Auskommentieren der Zuweisungen im Script.	# kapitalV1.kapital = 5000 # kapitalV1.zinsen = 5 # kapitalV1.jahre = 10 # # kapitalV2.kapital = 5000 # kapitalV2.zinsen = 5 # kapitalV2.jahre = 5

Klasse Schritt 6.

		<pre># Beginn Python Script # ... # kapitalV2.jahre = 5</pre>
19.	Hinzufügen der Methodenaufrufe zum Bestimmen der Werte für das Objekt „kapitalV1“.	<pre>kapitalV1.setKapital(5000) kapitalV1.setZinsen(5) kapitalV1.setJahre(10)</pre>
20.	Hinzufügen der Methodenaufrufe zum Bestimmen der Werte für das Objekt „kapitalV2“.	<pre>kapitalV2.setKapital(1000) kapitalV2.setZinsen(5) kapitalV2.setJahre(5)</pre>
21.	Objekt „kapitalV1“ ruft Methode „zinsRechnung()“ auf.	<pre>print('Kapital V1=', kapitalV1.zinsRechnung())</pre>
22.	Objekt „kapitalV2“ ruft Methode „zinsRechnung()“ auf.	<pre>print('Kapital V2=', kapitalV2.zinsRechnung())</pre>

Klasse Schritt 7.

23. Auskommentieren der bisherigen Version.
24. Erweiterte Version der Methode „setKapital()“, nunmehr mit Eingabeprüfung.

```
# Beginn der Klasse
class Kapital:
    # ...
    def zinsRechnung(self):
        return self.kapital * ( 1 + self.zinsen / 100 ) ** self.jahre

    # def setKapital(self, kapital):
    #     self.kapital = kapital

    def setKapital(self, kapital):
        if kapital < 1:
            print(self.version, ' ', 'Kapital? ')

    def setZinsen(self, zinsen):
        self.zinsen = zinsen

    def setJahre(self, jahre):
        self.jahre = jahre
# Ende der Klasse
```

Klasse Schritt 8.

25. Fehlerhaften Wert für das Kapital von Objekt „kapitalV2()“ testen.

```
# Beginn Python Script
# ...
kapitalV2.setKapital(0)
kapitalV2.setZinsen(5)
kapitalV2.setJahre(5)

print('Kapital V1=', kapitalV1.zinsRechnung())

print('Kapital V2=', kapitalV2.zinsRechnung())
```

26. Shell (REPL) Ausgabe der Fehlermeldung.

```
>>> %Run -c $EDITOR_CONTENT

MPY: soft reboot
Hallo EBW
Hallo EBW
Rechnung V2    Kapital?
Kapital V1= 8144.469
Kapital V2= 0.0
>>>
```

kapitalClass.py

Python

kapitalClass.py; Besonderheit: Klasse und Script in einer Datei.

```
# ----- Beginn Klassendefinition -----
class Kapital:
    def __init__(self):
        self.kapital = 0
        self.zinsen = 0
        self.jahre = 0

    def setKapital(self, kapital):
        self.kapital = kapital

    def setZinsen(self, zinsen):
        self.zinsen = zinsen

    def setJahre(self, jahre):
        self.jahre = jahre

    def zinsRechnung(self):
        return (self.kapital * (1 + self.zinsen / 100)
                ** self.jahre)

# ----- Ende Klassendefinition -----
```

```
# ----- Beginn des Python Scripts -----
kapitalV1 = Kapital()
kapitalV2 = Kapital()

kapitalV1.setKapital(5000)
kapitalV1.setZinsen(5.0)
kapitalV1.setJahre(5)

print ('Kapital V1 =', kapitalV1.zinsRechnung(),
      'Euro')

kapitalV2.setKapital(5000)
kapitalV2.setZinsen(5)
kapitalV2.setJahre(10)

print ('Kapital V2 =', kapitalV2.zinsRechnung(),
      'Euro')
# ----- Ende des Python Scripts -----
```

Anhang IDEs

PyScripter	https://sourceforge.net/projects/pyscripter/files/PyScripter-v4.2/PyScripter-4.2.5-x64.zip/download <ul style="list-style-type: none">• zip-Archiv herunterladen & entpacken• IDE mit Code-Vervollständigung. Für Einsteiger geeignet.
Visual Studio Code	https://code.visualstudio.com/download <ul style="list-style-type: none">• zip-Archiv herunterladen & entpacken• Erweiterung (Extension) Python installieren• Professionelle IDE mit Code-Vervollständigung. Für Experten.
Pymakr	Visual Studio Code mit Erweiterung für MicroPython „Pymakr“ habe ich nicht zum Laufen gebracht?
uPyCraft	https://randomnerdtutorials.com/install-upycraft-ide-windows-pc-instructions/ <ul style="list-style-type: none">• nur für MicroPython mit ESP8266 & ESP32
Arduino lab for MicroPython 0.8.0	https://labs.arduino.cc/en/labs/micropython <ul style="list-style-type: none">• nur für MicroPython mit ESP8266• Für Einsteiger gut geeignet.

Links

<https://www.w3schools.com/python/default.asp>

<https://pynative.com/get-started-with-python/>

<https://docs.python.org/3/tutorial/index.html>

<https://www.tutorialspoint.com/python/index.htm>

<https://realpython.com/python-first-steps/>

<https://www.python-lernen.de/python-grundlagen.htm>

<https://runestone.academy/ns/books/published/pythonds/index.html?mode=browsing>

<https://bildungsserver.berlin-brandenburg.de/inf-sek1-python-bb>

<https://www.python-lernen.de/python-turtle.htm>

<https://realpython.com/python-repl/>

<https://quickref.me/python.html>

<http://python4kids.net/how2think/index.htm>

https://www.python-kurs.eu/python3_entstehung_python.php
