TECHNISCHE UNIVERSITÄT DRESDEN

DRESDEN concept

Thiemo Leonhardt
Professur für Didaktik der Informatik

Imperatives Programmierparadigma

1. Vorlesung Programmierung für das Lehramt

2019-04-09

Qualifikationsziele

Das Modul umfasst alle grundlegenden Teilbereiche der imperativen Programmierung:

- Grundlagen der Programmierung
- Strukturierte und dynamische Datentypen
- Effiziente Gestaltung des Problemlöseprozesses durch modulares Arbeiten mit Funktionen und Prozeduren
- Modellierung und Implementierung von Lösungen zu Problemstellungen bzw. deren Lösungen
- Grundalgorithmen in Problemlösungsstrategien
- Effizienzuntersuchungen von Algorithmen

Organisatorisches

Gliederung der Vorlesung:

- Vorlesung (V) (1 SWS)
- Übung (Ü) (2 SWS)
- Selbststudium

6 Leistungspunkte

- Vorlesung 15h
- Übung 30h
- Belegarbeit 60h
- Selbststudium 75h

Voraussetzungen:

- Grundkenntnisse des imperativen Programmierparadigmas
- Kenntnisse im Bereich Algorithmen und Datenstrukturen

Anforderungen:

Die Modulprüfung besteht aus einer Klausurarbeit von 90 Minuten Dauer und einer unbenoteten Belegarbeit im Umfang von 60 Stunden.

(imperatives) Programmieren anhand von Python

- … kann Ihnen niemand beibringen!
- … müssen Sie aktiv lernen!
- … erfordert Übung, Einsatz und Durchhaltevermögen
- Selbststudium + Belegarbeit 135h
  - 15 Wochen
  - 9h pro Woche

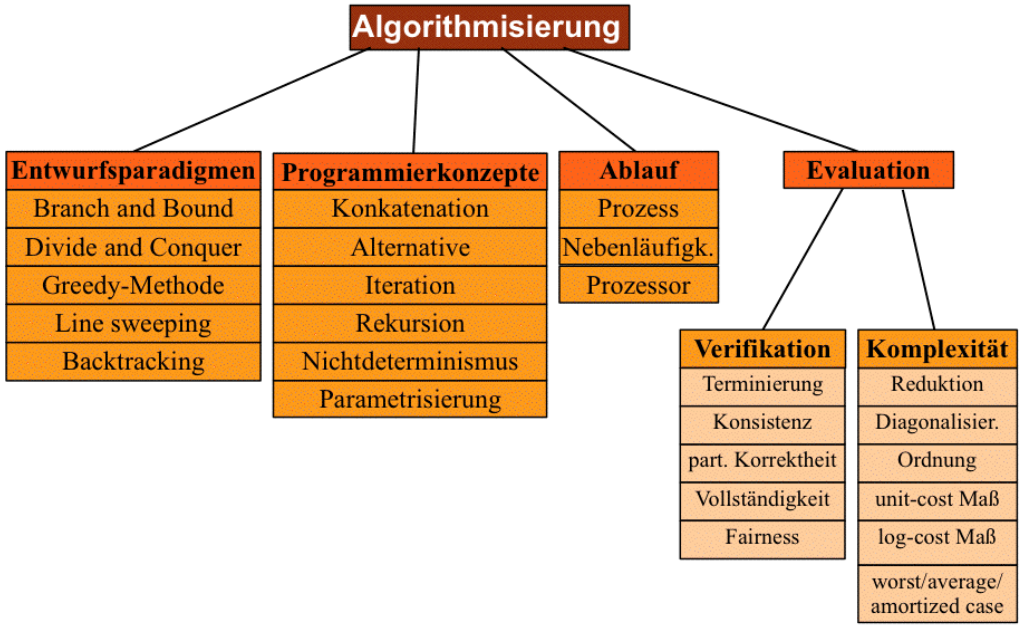
Programmierung

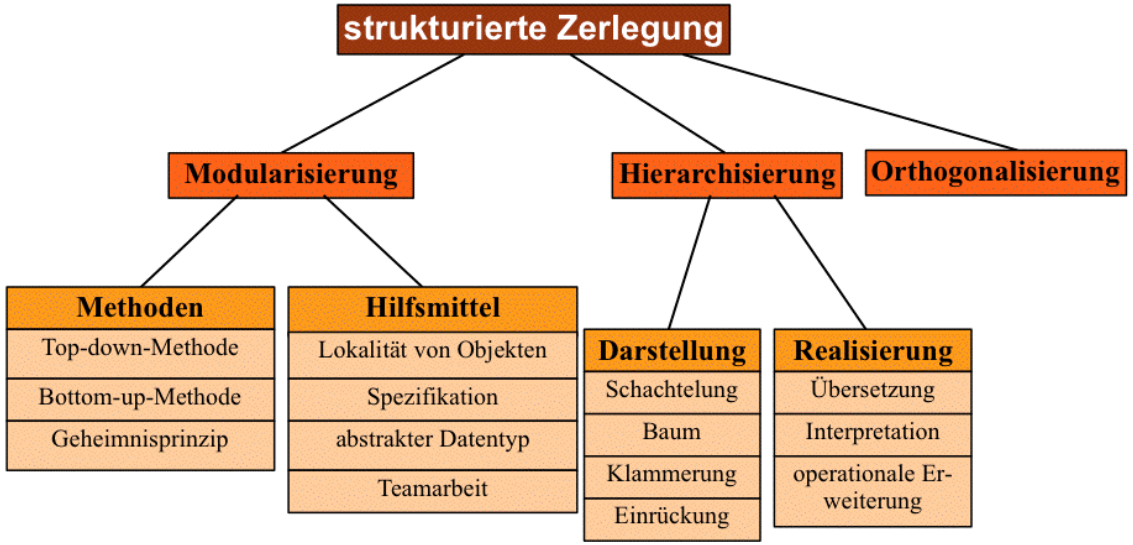
- Konzepte verstehen
- nicht nur eine Programmiersprache zu erlernen (Python ist exemplarisch!)
- ist wesentliche Denkweise und Methodik der Informatik

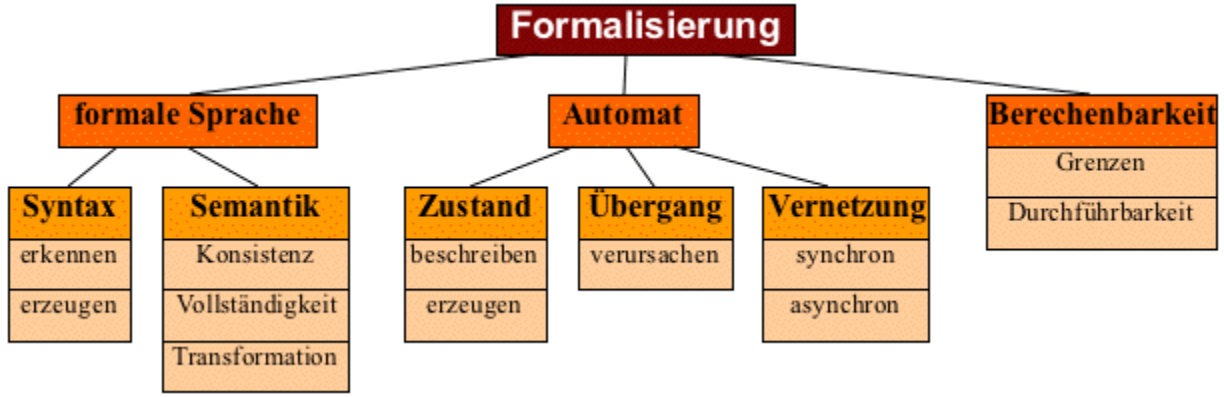
Unterstützung

- vielfältige Betreuung und Hilfestellung für Ihr eigenständiges Lernen
- ausführliche Beratung in Ihren Übungen

Fundamentale Ideen der Informatik

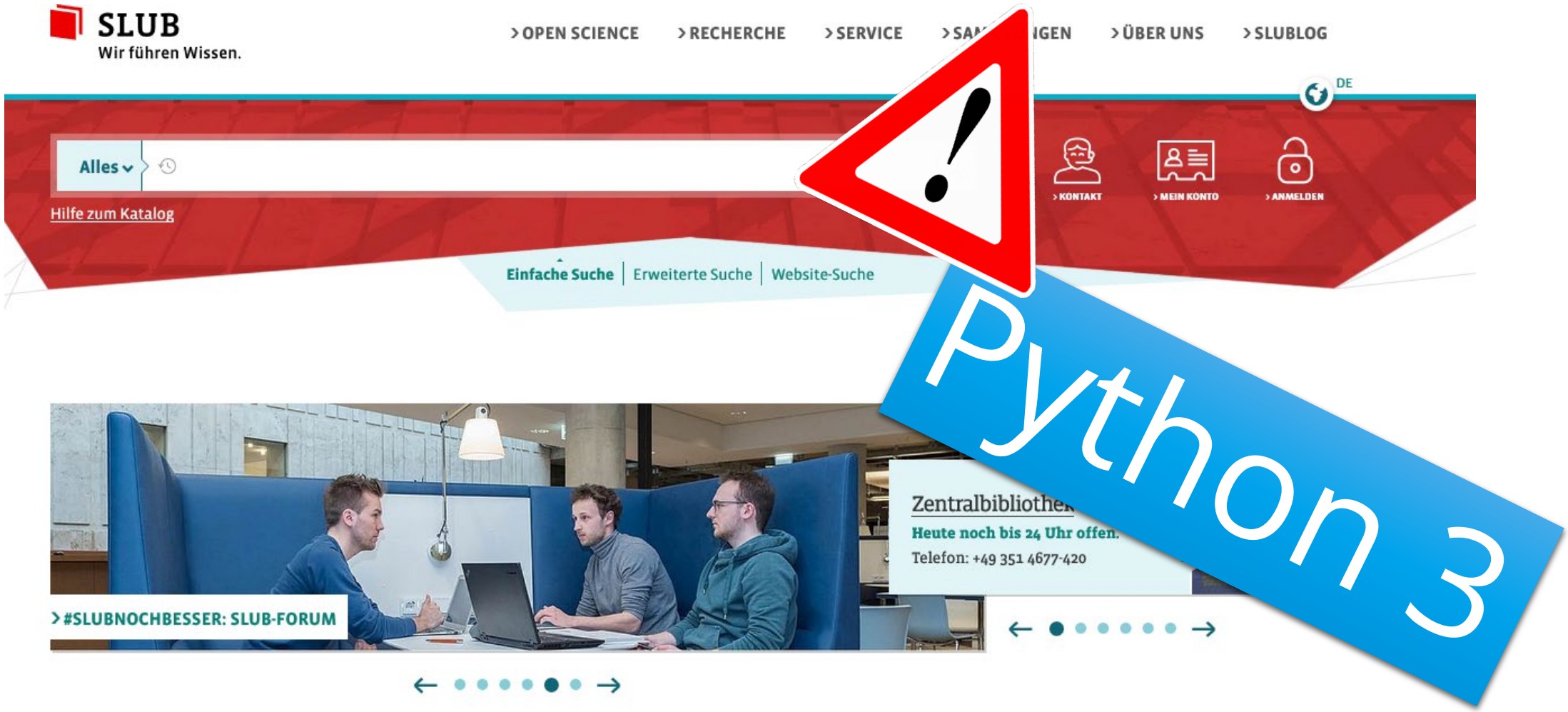






Andreas Schwill, 1993
Eckart Modrow, 2006

Vorlesung: Bücher



Übung: Online-Ressourcen

Tutorials:

https://docs.python.org/3/tutorial/

https://www.w3schools.com/python/

https://www.tutorialspoint.com/python/

https://www.python-kurs.eu/python3_kurs.php

Fragen und Antworten:

https://stackoverflow.com/

viele Online-Kurse: codecademy, udemy...

Selbstständiges Lernen

- Lectures are as effective as other methods, such as discussion or reading, for transmitting information. In general, however, students capture only 20-40 per cent of a lecture's main ideas in their notes (Kiewra, 2002) and without reviewing the lecture material, students remember less than 10 per cent after three weeks (Bligh, 1998).

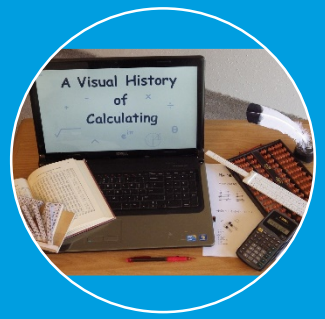
Aufbereitung

Allgemein gültige Inhalte zur *imperativen* Programmierung

Python-spezifische Inhalte

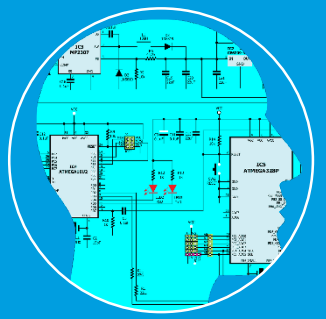
Was bedeutet *imperatives* Programmieren?

Definition Programm



Algorithmus
(Verhalten)

Datenstruktur
(Modell)



Bedienschnittstelle

Was ist ein *Algorithmus*?

Exakt definierte Handlungsvorschrift zur Bewältigung einer Aufgabe.

Beispiele des täglichen Lebens

- Kochrezept,
  - wenn alle Angaben genau beschrieben sind
  - es für alle Teilaufgaben, wie Braten, Rühren, etc., ebenfalls Algorithmen gibt.
- Installationsanweisungen, Bauvorschriften, …
- Reparatur- und Bedienungsanleitungen
- Hilfen zum Ausfüllen von Formularen
- Mathematische Lösungsvorschriften
- Programme

Bsp.: Euklidischer *Algorithmus*

Mathematische Rechenanweisung:

- 300 v. Chr.: Ermittlung des größten gemeinsamen Teilers (ggT) zweier natürlicher Zahlen a und b:

    1. Ziehe die kleinere von der größeren Zahl ab
    2. Wenn a und b ungleich sind, dann fahre fort mit Schritt 1, wenn sie gleich sind, ist dies der größte gemeinsame Teiler

```
def ggt(x, y):
    while x!=y:
        if x > y:
            x = x - y
        else:
            y = y - x
    return x
```

Algorithmus und Programm

Eigenschaften von Algorithmen

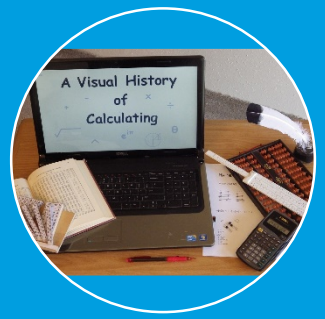
- Eindeutig beschrieben
- Terminierend
- Determiniert
- Korrekt
- Effizient
  - Speicherbedarf
  - Laufzeit
  - Technische Plattform
- Verständlich, nachvollziehbar
- Wiederverwendbar, übertragbar
- …

Beispiele

- Wahlvorschriften
- Spielanleitungen
- Kochrezepte
- Systematisches Suchen, Sortieren, …
- Rechenvorschriften:
  - Differenzieren
  - Lösen von Gleichungen
  - Bestimmen von Determinanten

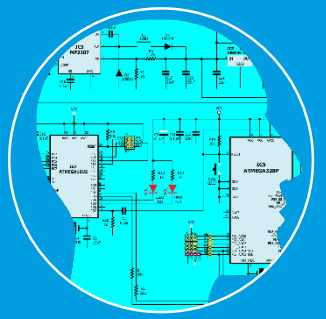
- Ablauffähiger Algorithmus
  - Programm beschrieben in einer Programmiersprache

Definition Programm



Algorithmus
(Verhalten)

Datenstruktur
(Modell)

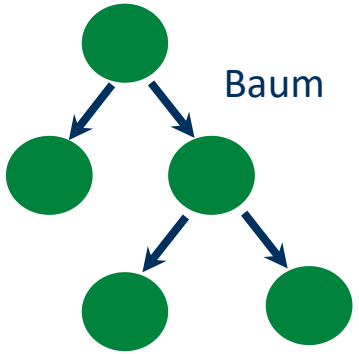


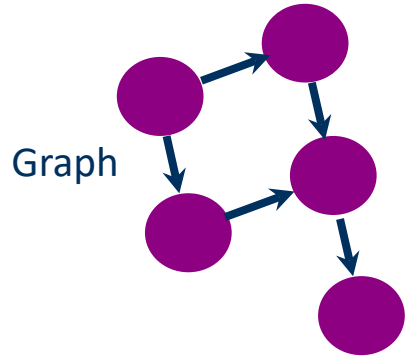
Bedienschnittstelle

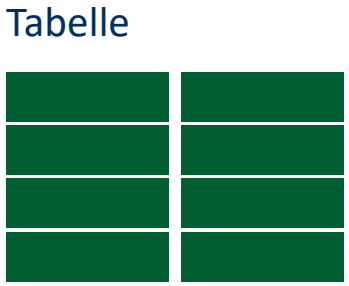
Was ist eine *Datenstruktur*?

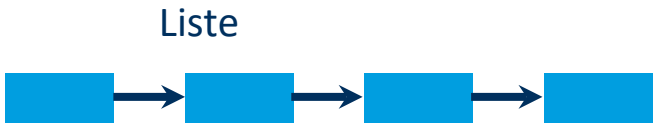
Modell der relevanten Aspekte des Einsatzgebiets

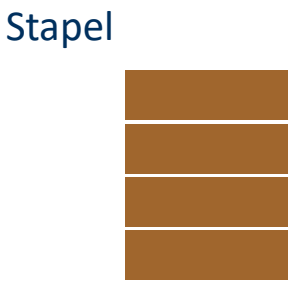
- Art, Daten zu verwalten und miteinander zu verknüpfen
- unterschieden nach Zugriffsart, Einfüge- und Löschoperationen, …

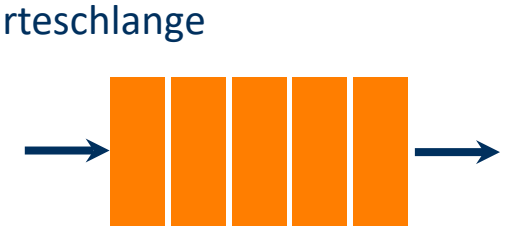




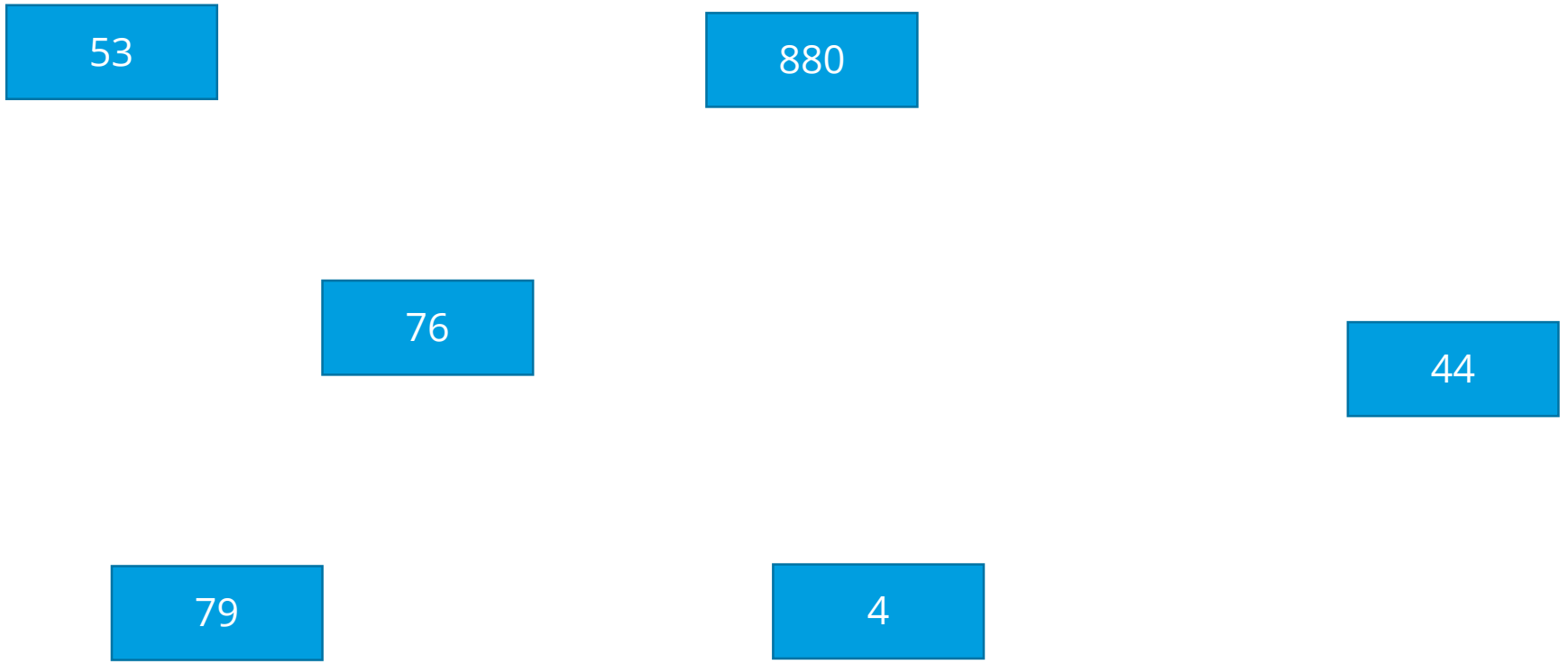




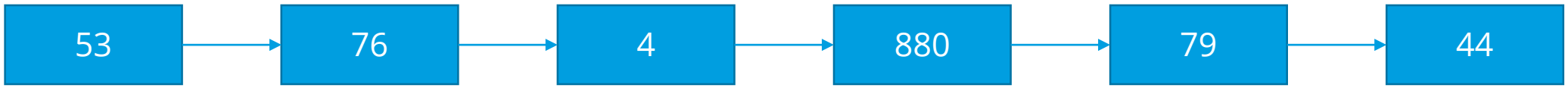




Daten finden – Wie machen Sie das?



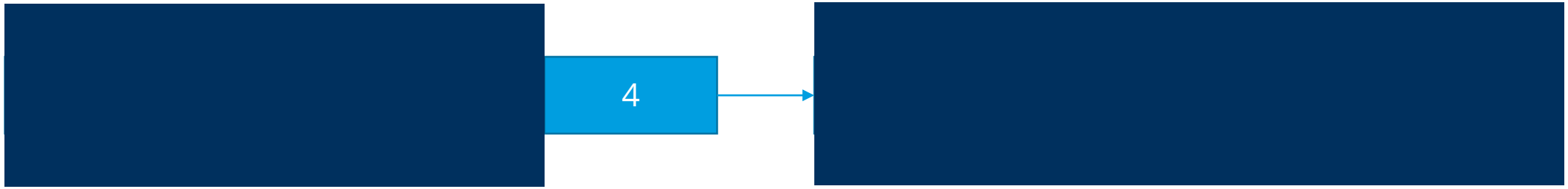
Daten finden – Wie machen Sie das?



Daten finden – Was sieht der Computer/Rechner/Algorithmus?

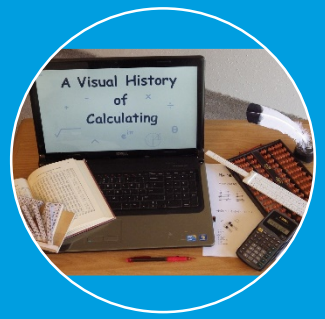
Daten finden

Daten finden



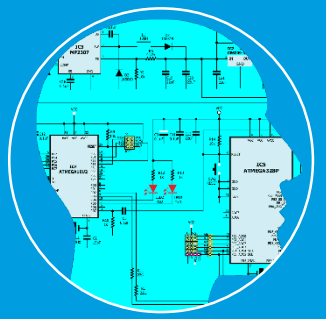
Daten finden

Definition Programm



Algorithmus
(Verhalten)

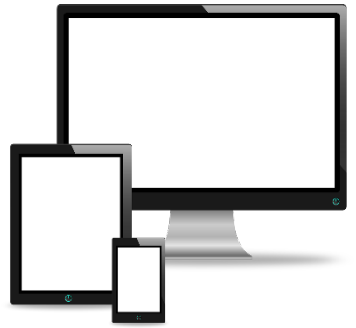
Datenstruktur
(Modell)



Bedienschnittstelle

Was bedeutet *Schnittstelle*?

Die Programmiersprache bildet die Schnittstelle zwischen Mensch und Rechner.



unterschiedliche Anforderungen

Mensch

- Erlernbarkeit
- Lesbarkeit
- Ausdrucksstärke, Problemnähe

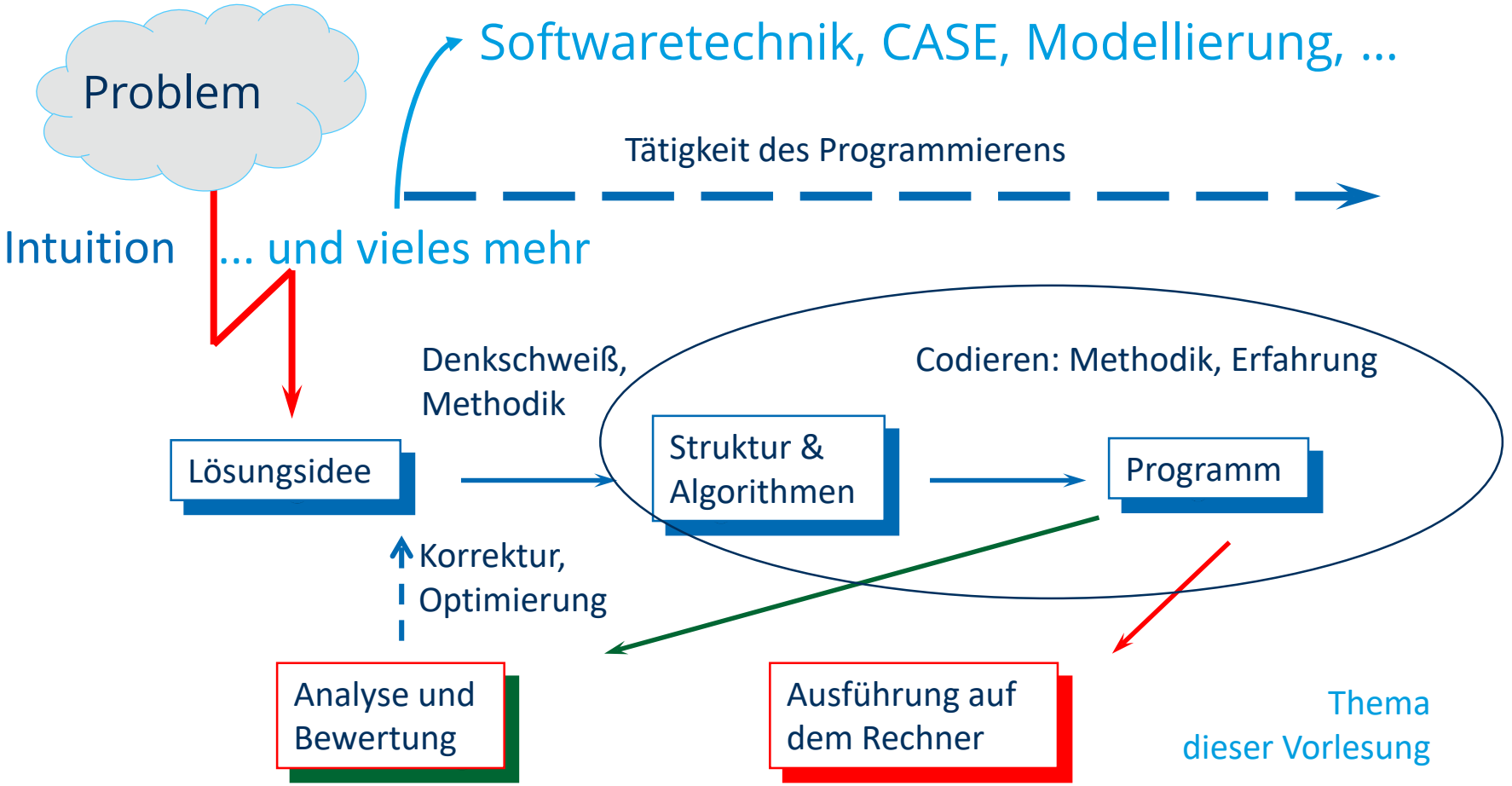
Höhere Programmiersprachen

Rechner

- einfaches Übersetzen in Maschinensprache
- effizienter Code soll generiert werden können

Maschinensprachen

Vom *Problem* zum *Programm* und wieder zurück



Programmieren im Kleinen

Imperatives Programmierkonzept

Die Beschreibung eines Algorithmus in elementaren Verarbeitungsschritten, im besonderen der Zuweisung von Werten an Variablen

Weitere Programmierparadigmen:

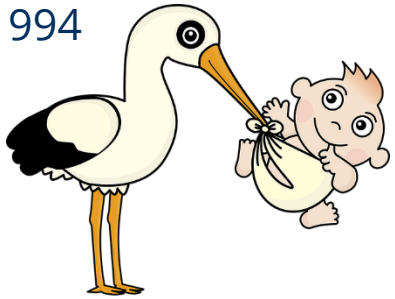
- Deklarative Programmierparadigma
- Objektorientierte Programmierparadigma
- ...

Keine scharfe Trennung viele Sprachen enthalten Aspekte aus mehreren Programmierparadigmen.

Programmiersprache *Python*

Geschichte

1994



Guido van Rossum

2000
Python 2

2008
Python 3

aktuell
Python 3.7

damit arbeiten wir

python™

freie Lizenz

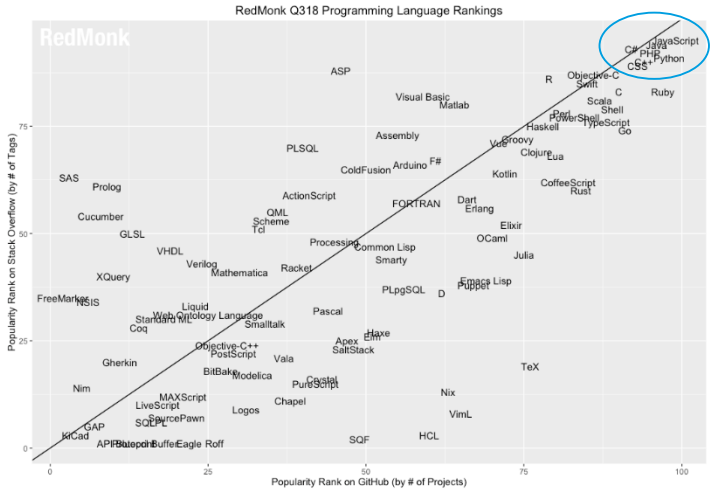
plattformunabhängig

große Community
viele Packages

Popularität 2018

https://www.tiobe.com

Mar 2019	Mar 2018	Change	Programming Language	Ratings	Change
1	1		Java	14.880%	-0.06%
2	2		C	13.305%	+0.55%
3	4	↑	Python	8.262%	+2.39%
4	3	↓	C++	8.126%	+1.67%
5	6	↑	Visual Basic .NET	6.429%	+2.34%



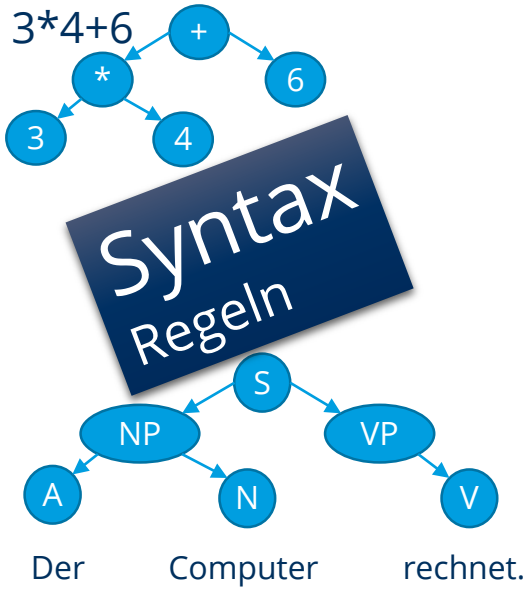
https://redmonk.com/

Worldwide, Feb 2019 compared to a year ago:

Rank	Change	Language	Share	Trend
1	↑	Python	26.42 %	+5.2 %
2	↓	Java	21.2 %	-1.3 %
3	↑	Javascript	8.21 %	-0.3 %
4	↑	C#	7.57 %	-0.5 %
5	↓↓	PHP	7.34 %	-1.2 %

http://pypl.github.io/

Wie kommt es, dass wir uns verstehen?



Python ist eine (Programmier-)Sprache

Syntax

- Vokabular und Regeln zur Anordnung stark eingeschränkt
- im Gegensatz zur Umgangssprache muss man sich strikt an die Regeln halten , damit eine "dumme" Maschine dies auch versteht

Semantik

- legt die Bedeutung aller Konstrukte eindeutig fest
- es darf niemals Missverständnisse geben

Pragmatik

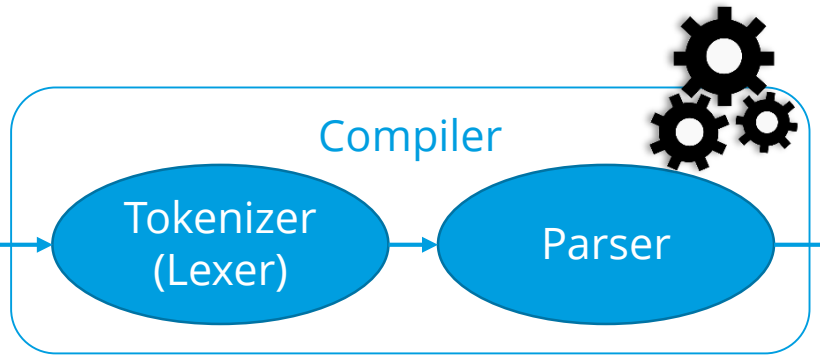
- verlangt allerdings intensive Beschäftigung für mehrere Monate
- wie drücke ich etwas mit den wenigen Sprachmitteln sinnvoll aus?
- welche vordefinierten Formulierungen gibt es, die ich nur verwenden muss?

Vom *Quellcode* zum ausführbaren *Programm*

ggt.py

```
1  def ggt(x, y):
2      while x!=y:
3          if x > y:
4              x = x - y
5          else:
6              y = y - x
7      return x
8
9  print(ggt(10, 4))
```

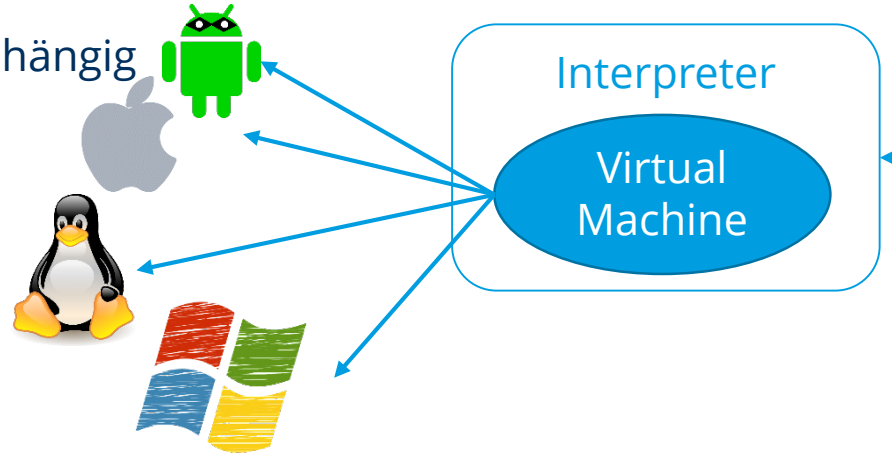
Python Source code




```
4           0 SETUP_LOOP              38 (to 40)
      >>    2 LOAD_FAST                0 (x)
            4 LOAD_FAST                1 (y)
            6 COMPARE_OP               3 (!=)
            8 POP_JUMP_IF_FALSE       38

5          10 LOAD_FAST                0 (x)
           12 LOAD_FAST                1 (y)
           14 COMPARE_OP               4 (>)
           16 POP_JUMP_IF_FALSE       28

6          18 LOAD_FAST                0 (x)
           20 LOAD_FAST                1 (y)
           22 BINARY_SUBTRACT
           24 STORE_FAST               0 (x)
           26 JUMP_ABSOLUTE            2

8     >>   28 LOAD_FAST                1 (y)
           30 LOAD_FAST                0 (x)
           32 BINARY_SUBTRACT
           34 STORE_FAST               1 (y)
           36 JUMP_ABSOLUTE            2
      >>   38 POP_BLOCK

9     >>   40 LOAD_FAST                0 (x)
           42 RETURN_VALUE
```

Bytecode

plattformunabhängig



Programmkonstrukte *Python*

Python-Programme: Das Grundmuster

Das alles schreiben Sie
als **Text** in der Sprache Python!

Sie brauchen eine **Quelldatei**...

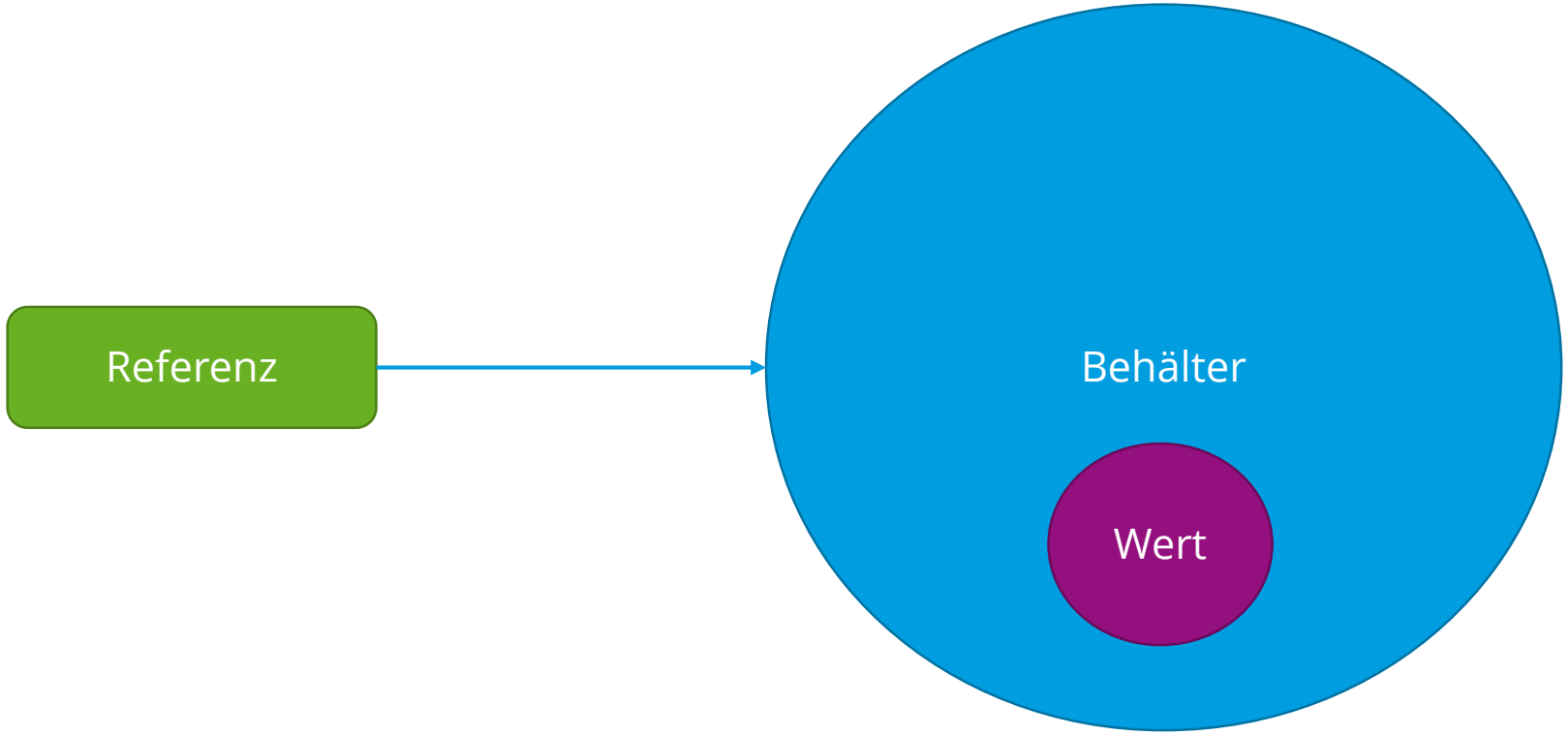
...in der Sie Anweisungen definieren...

...in der Sie **Anweisungsköpfe** definieren...

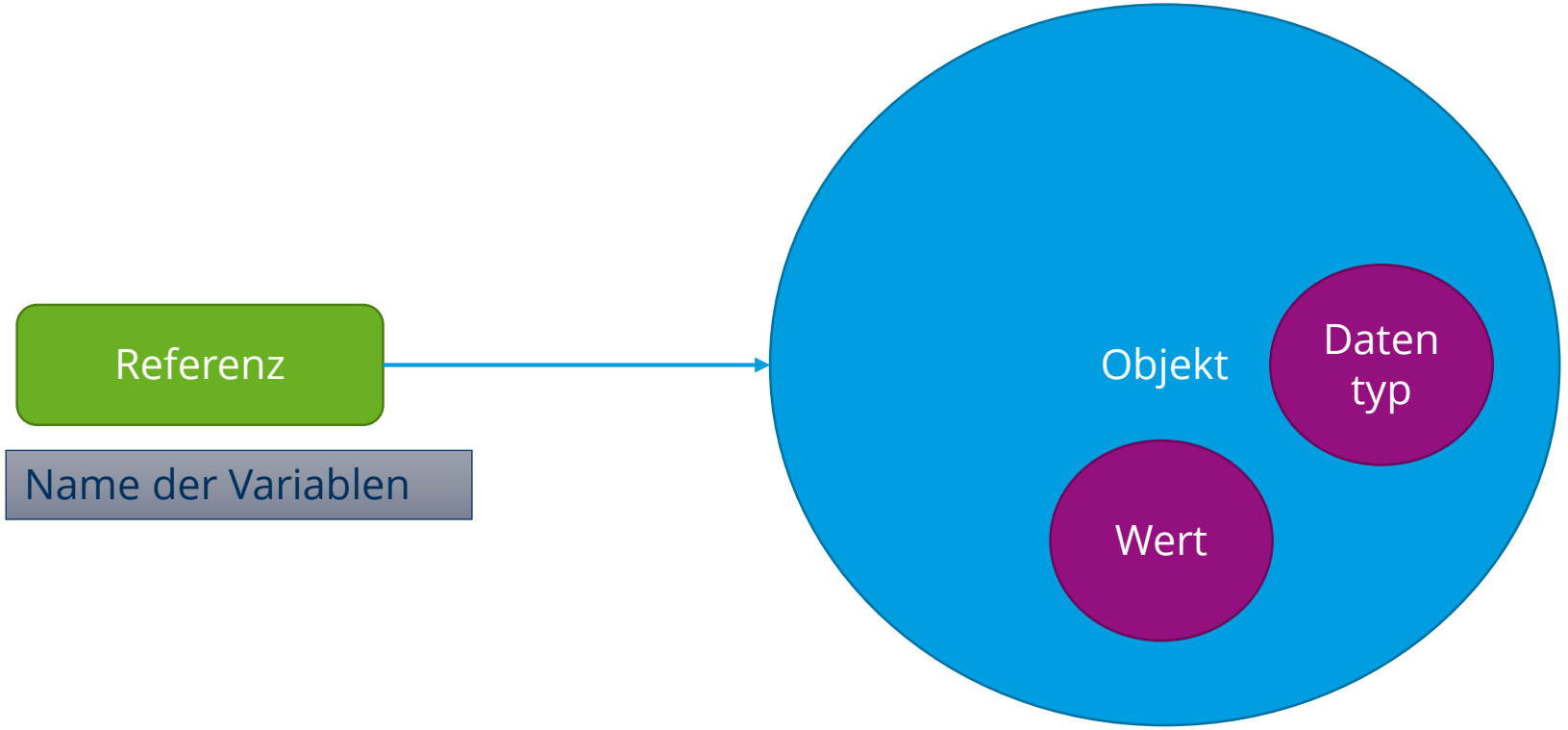
...die wiederum **Anweisungen** enthalten.

"Puh!"

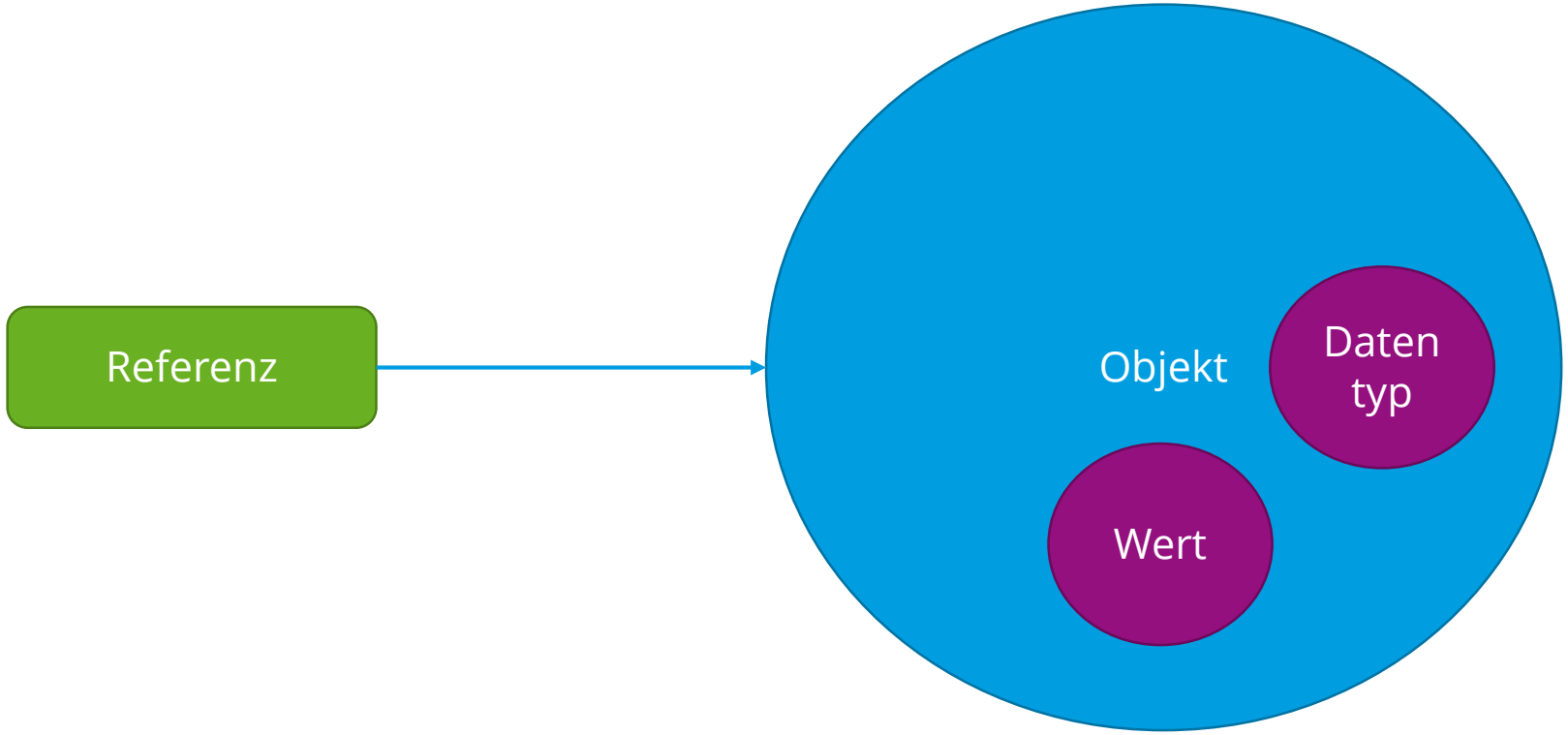
Variablen in imperativen Programmiersprachen



Variablen in Python

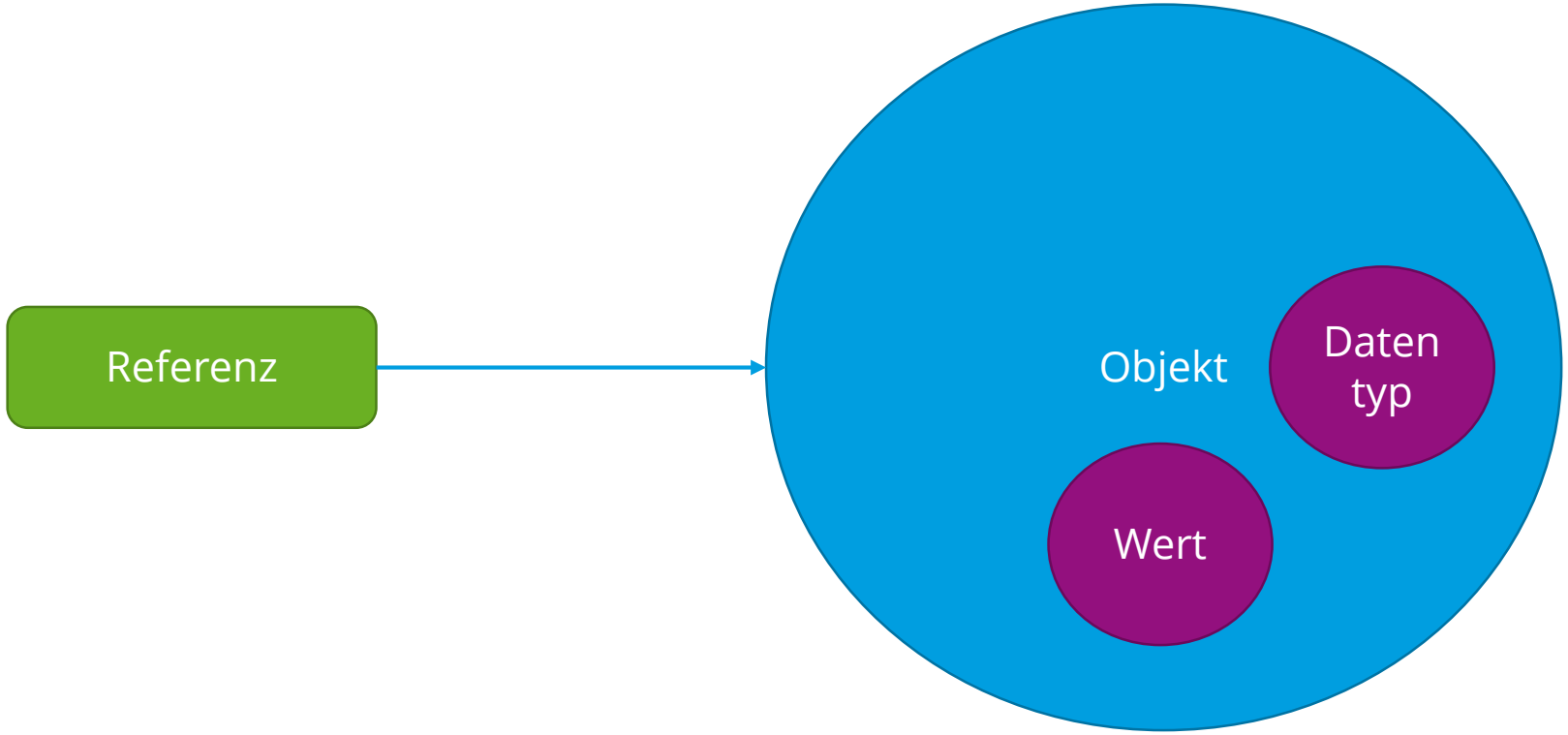


Variablen in Python



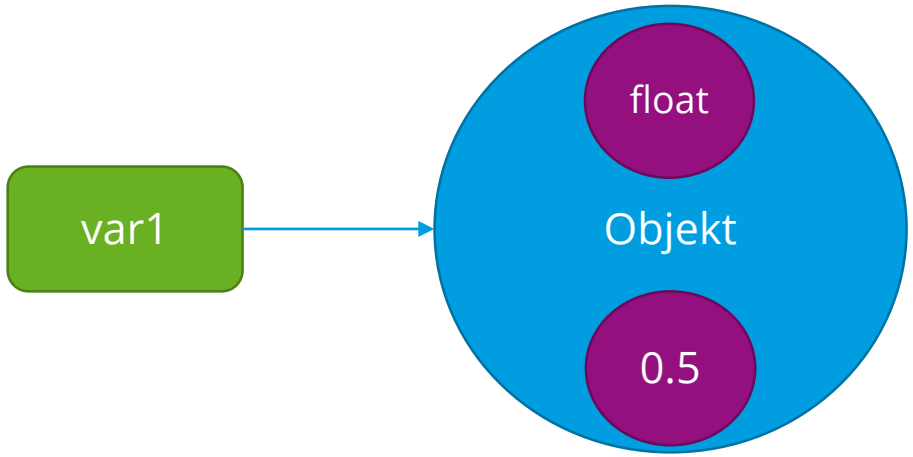

```
ausgabe_text = “Ich bin eine Ausgabe“
```

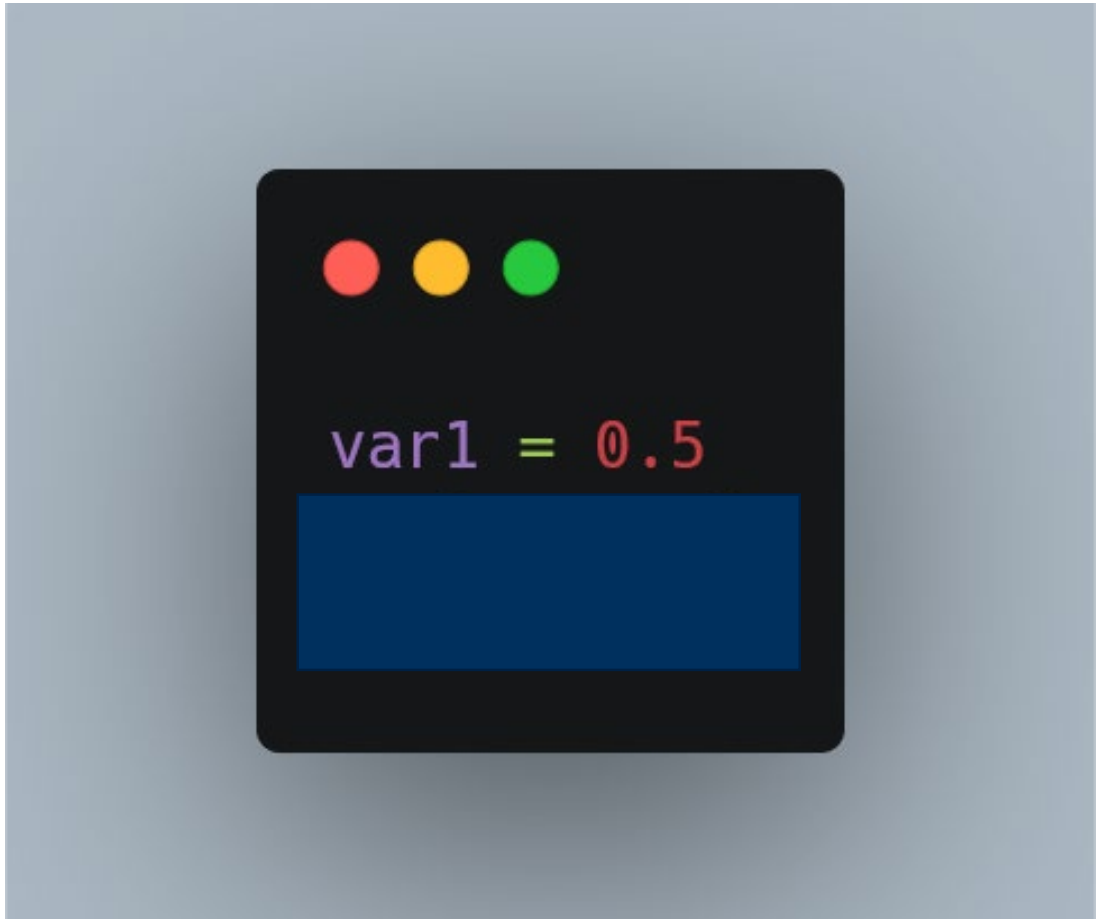
Variablen in Python



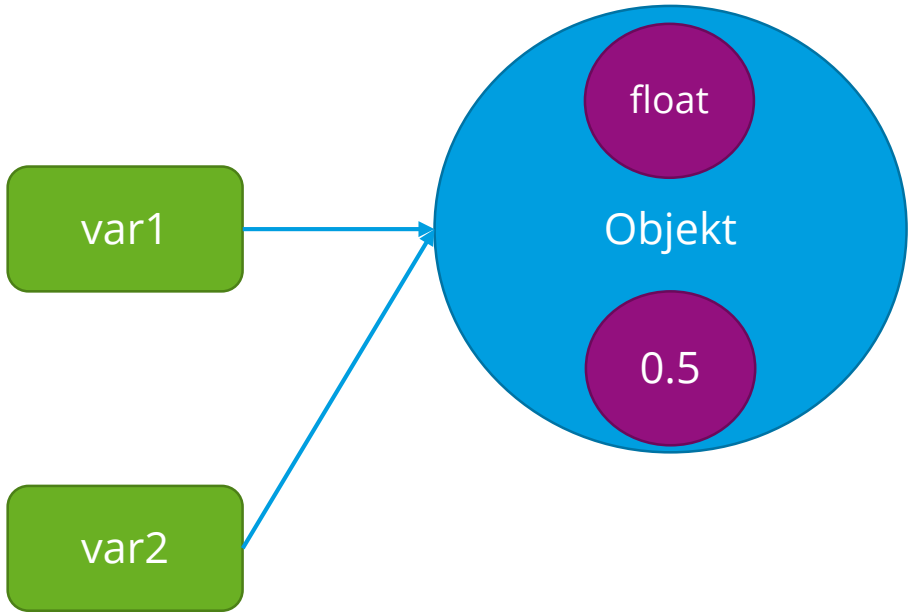

```
eingabe = 4.2
```

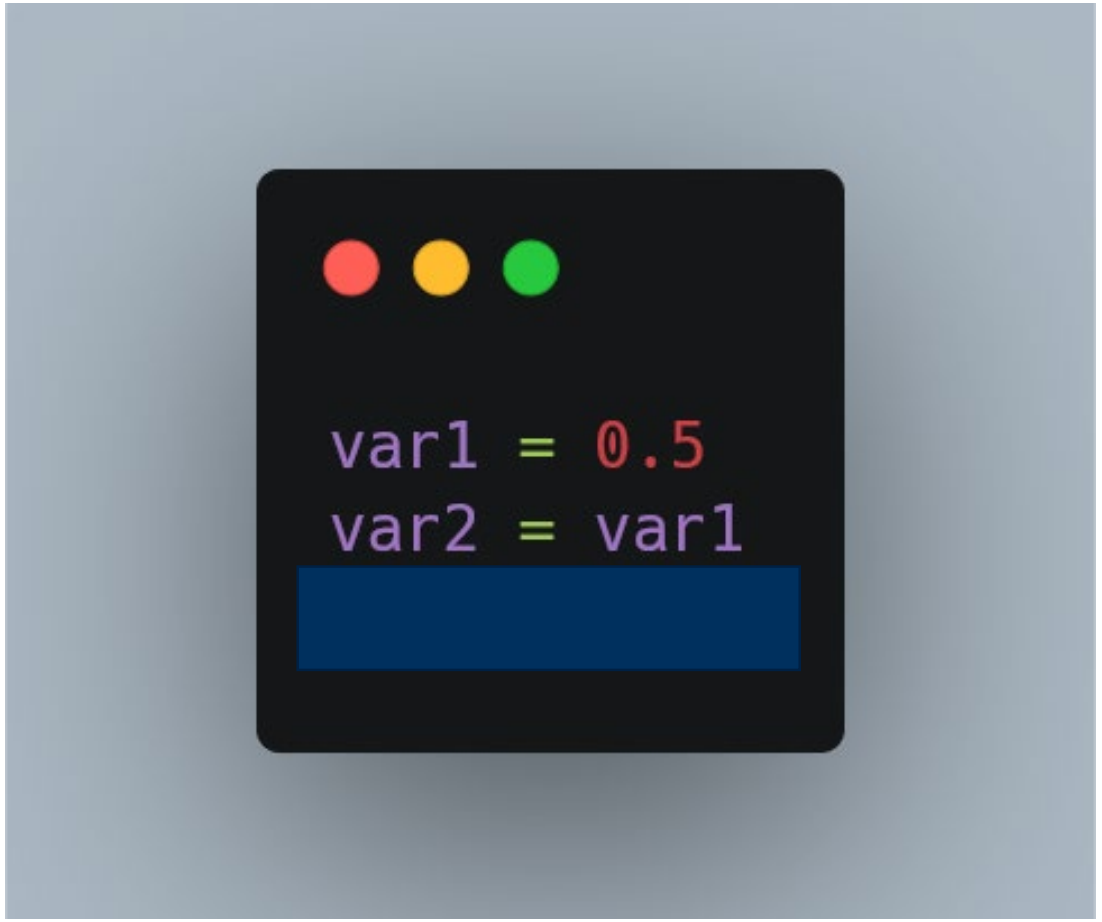
Variablen in Python



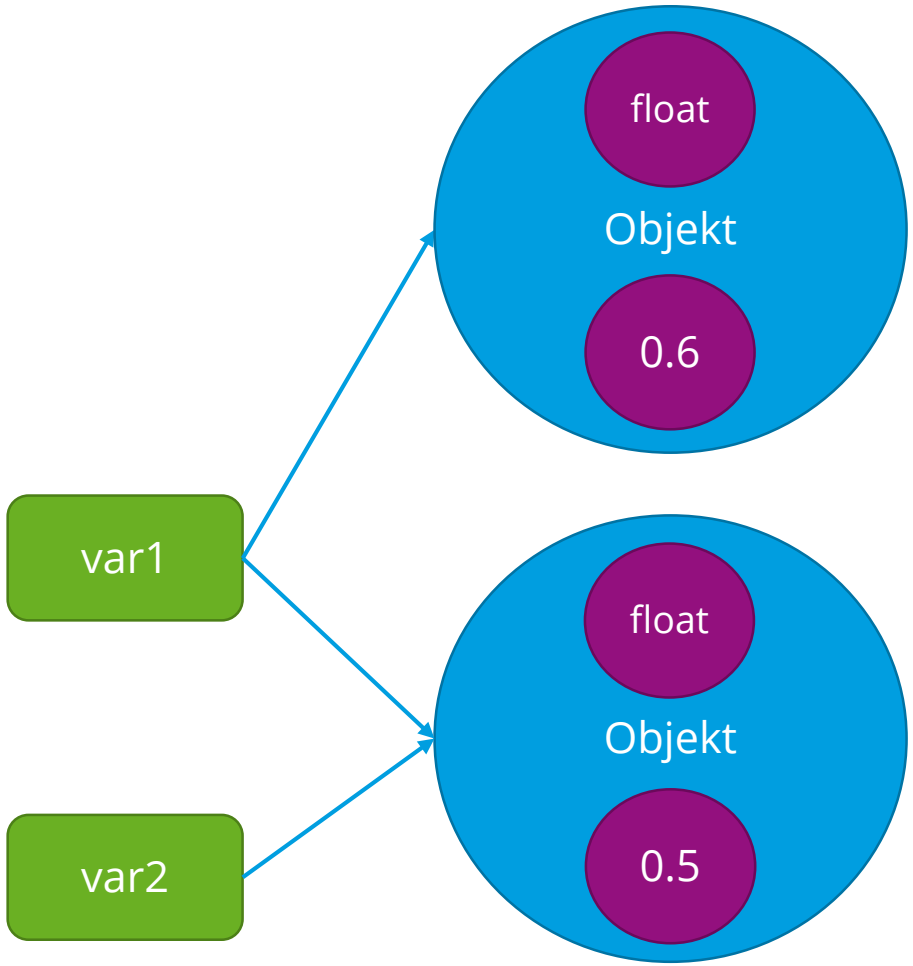


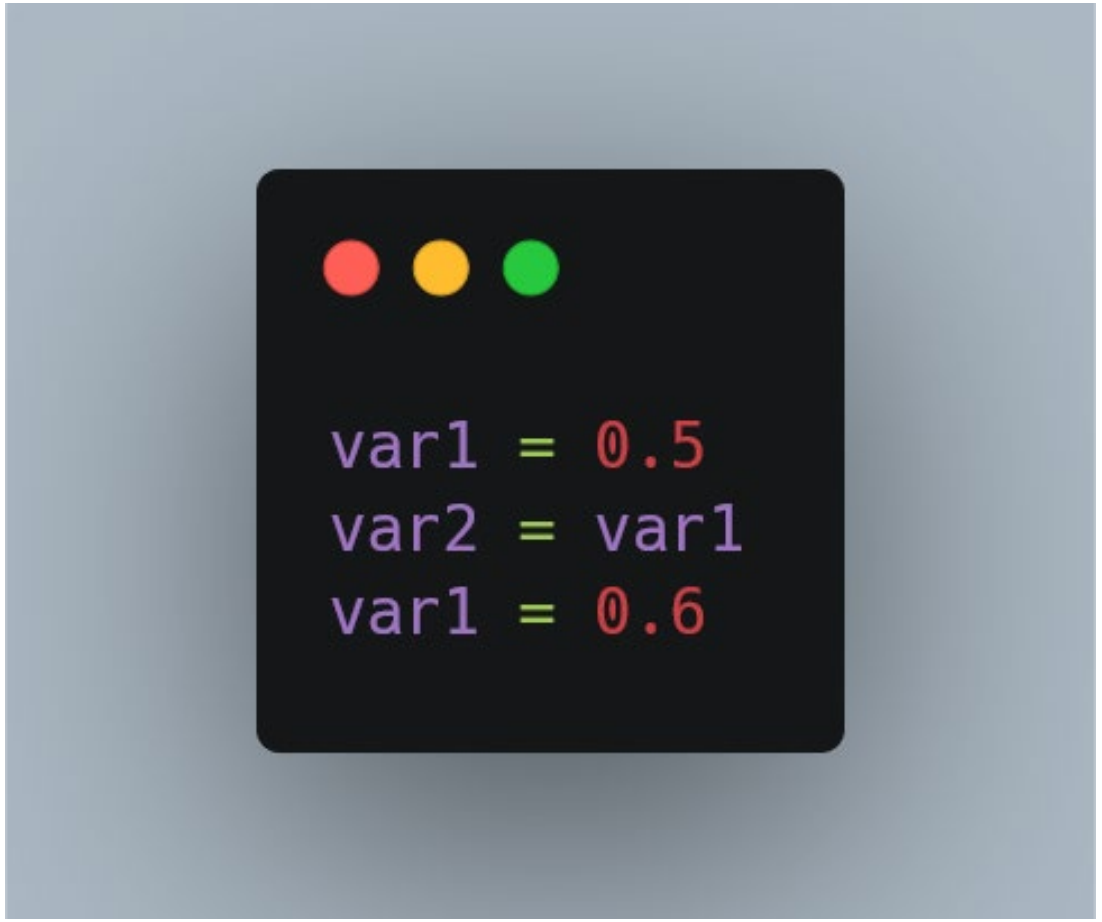
Variablen in Python





Variablen in Python



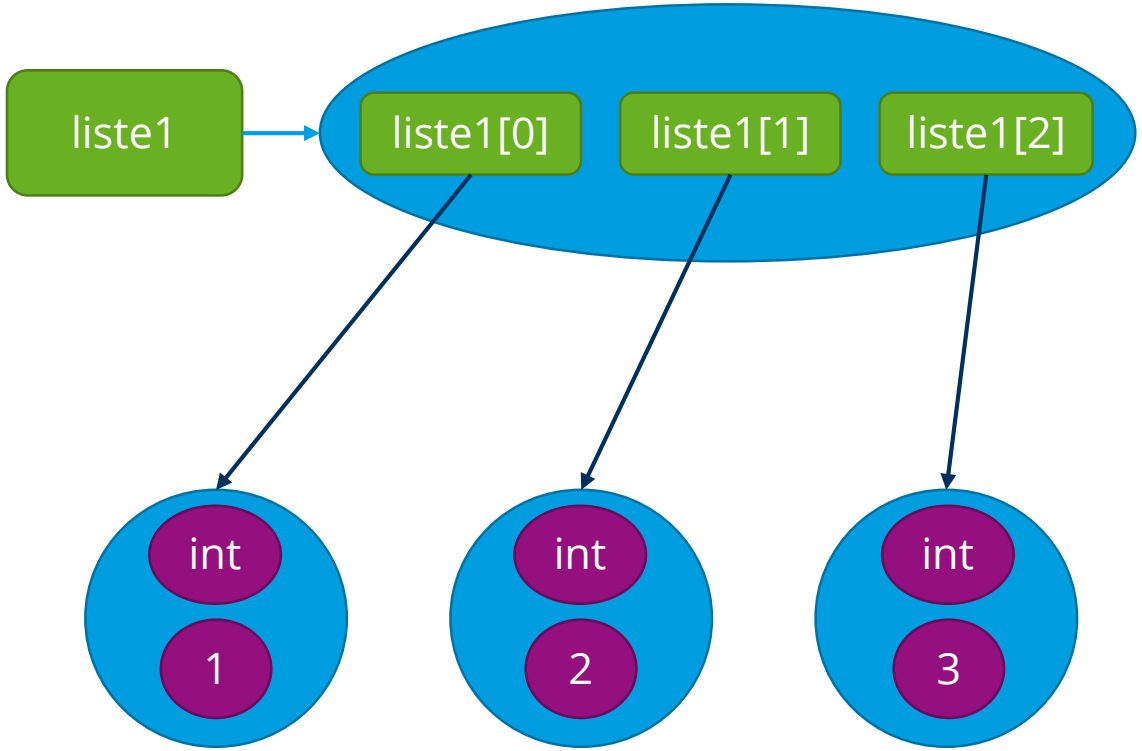


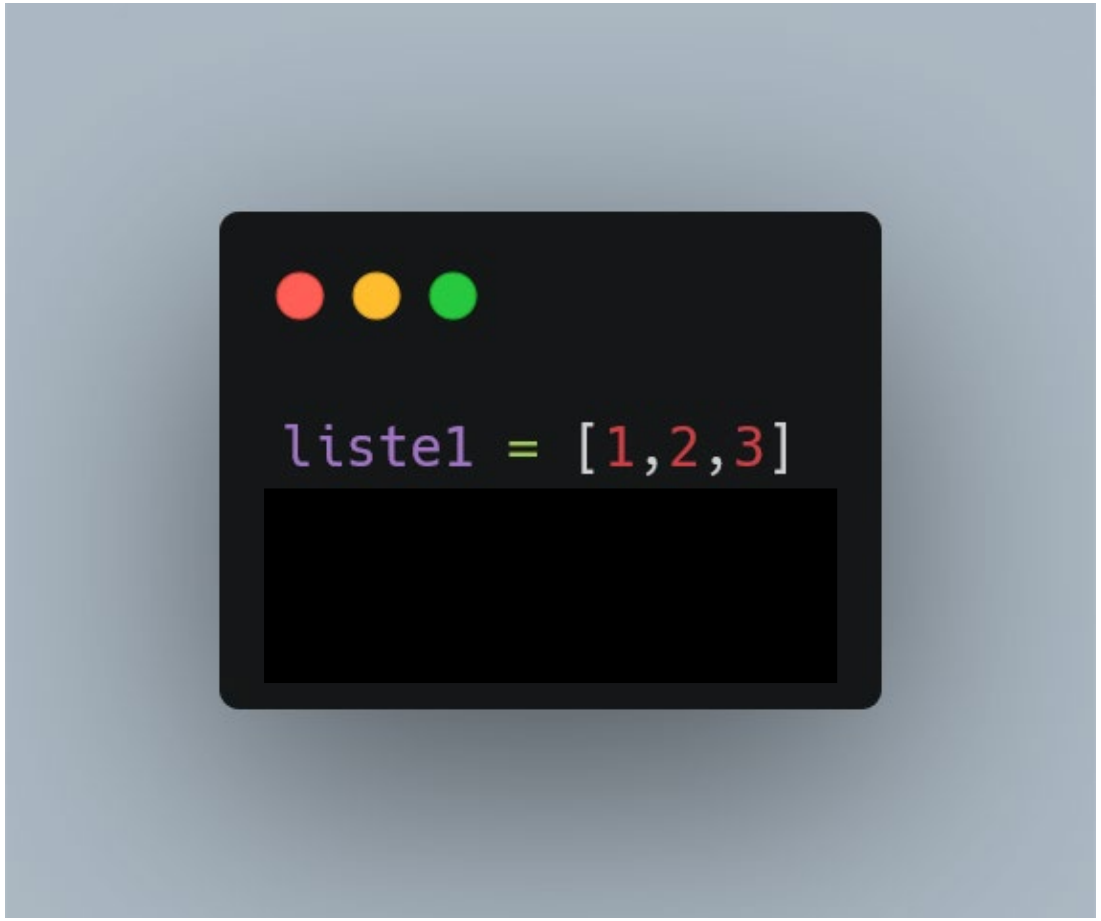
Listen von Objekten in Python

- Eine Liste kann eine Folge beliebiger Objekte gespeichert werden
  - Zum Beispiel Strings, Integers, Float-Zahlen aber auch Listen und Tupel selbst.
- Listen Notation: Eckige Klammern [] und mit Kommata getrennt.

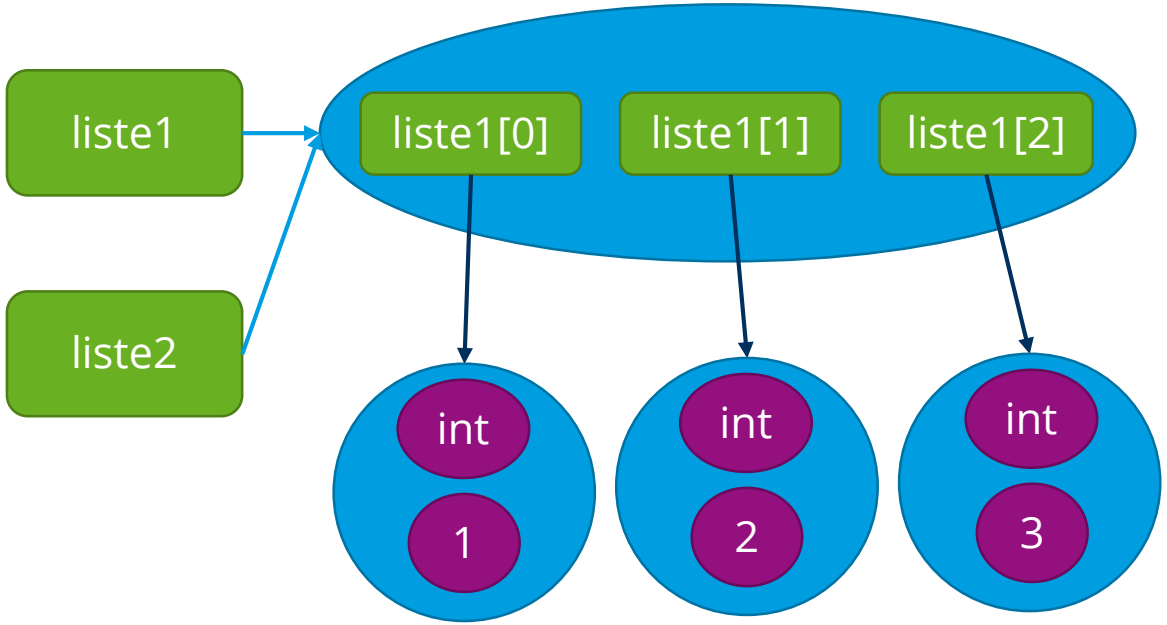
```
zahlen_liste = [1,2,3]
text_liste = ["Hallo", "wie", "gehts?"]
mix_liste = [1, "Eins", 1.0]

print(mix_liste[2])
text_liste[0] = 5
```

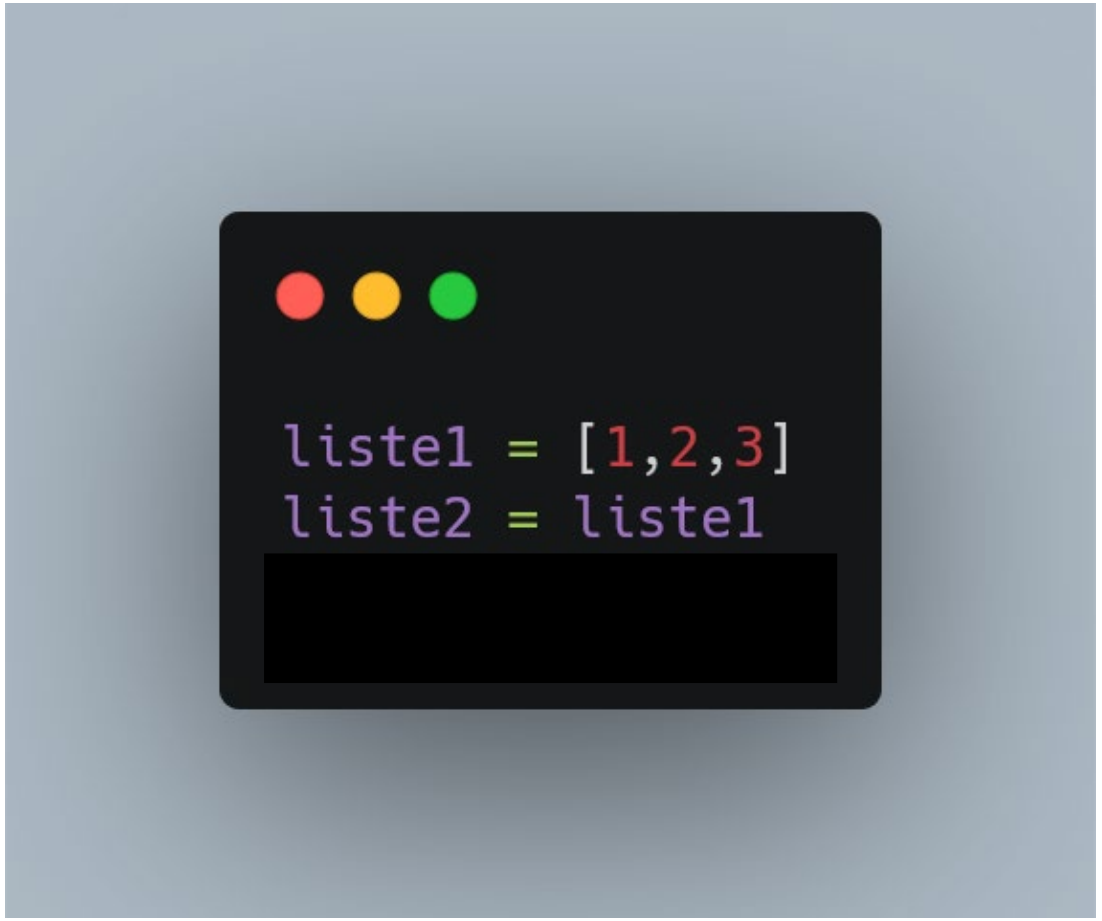
Variablen und Listen in Python



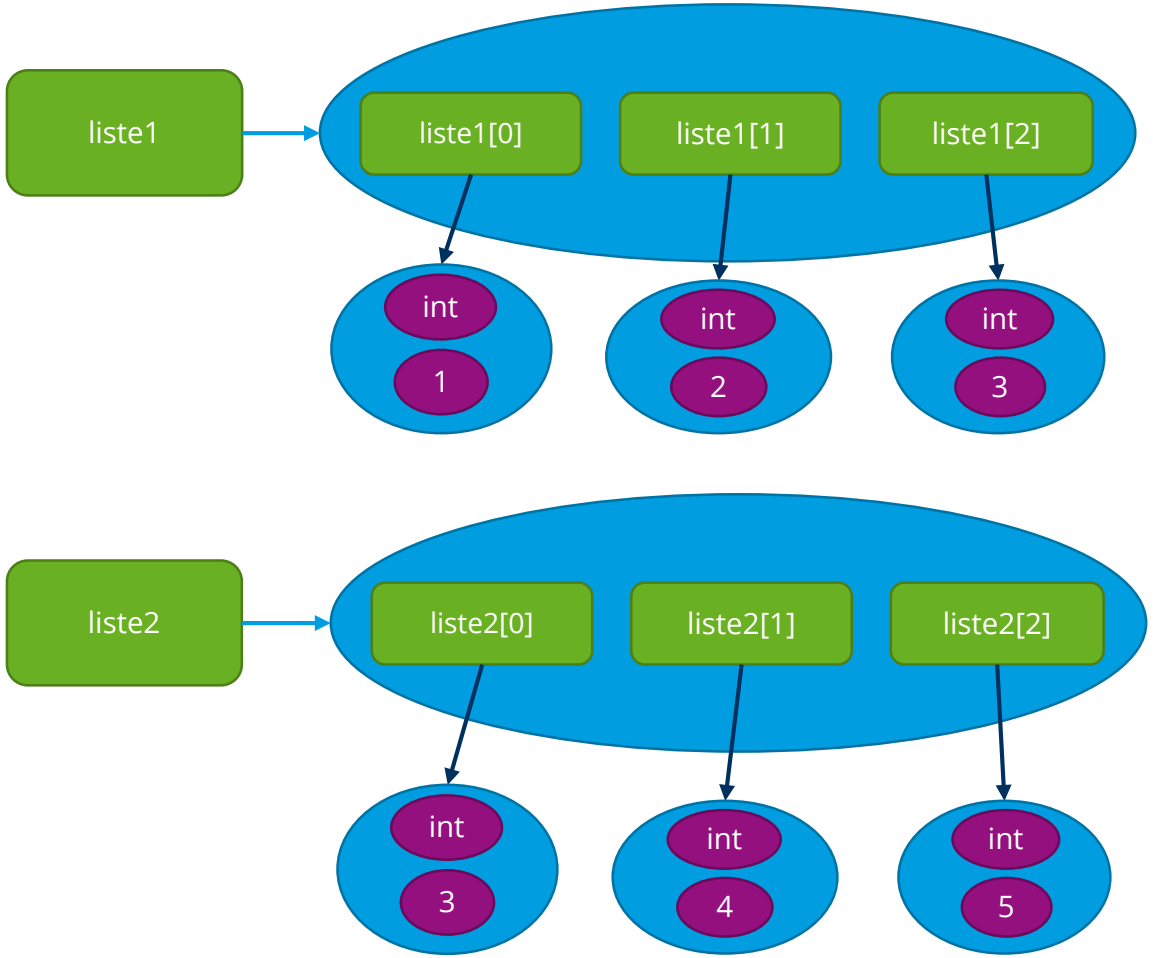


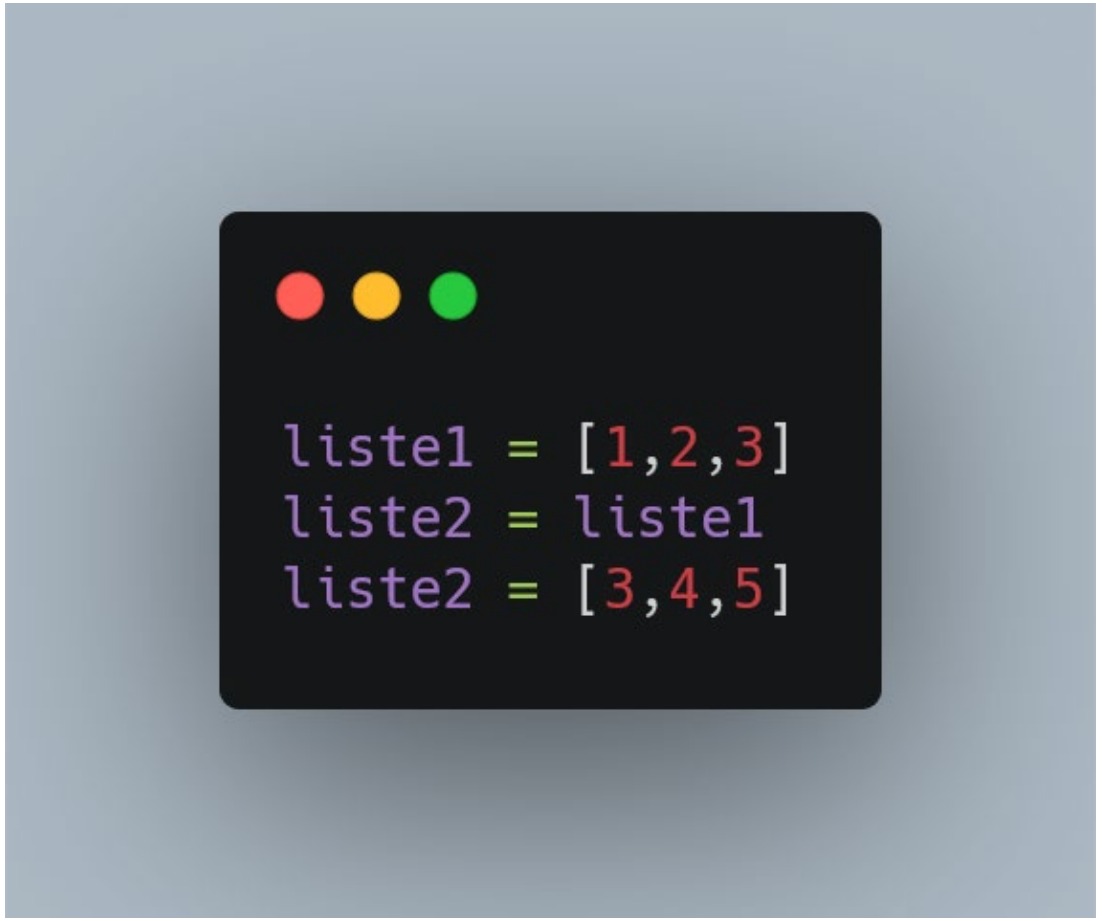
Variablen und Listen in Python



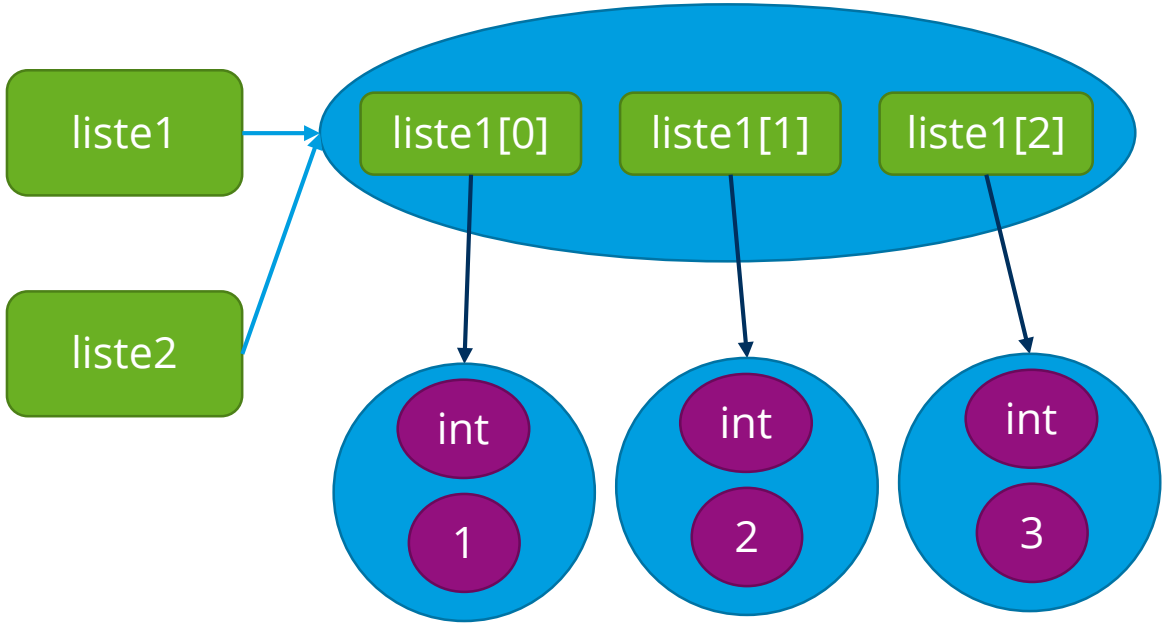


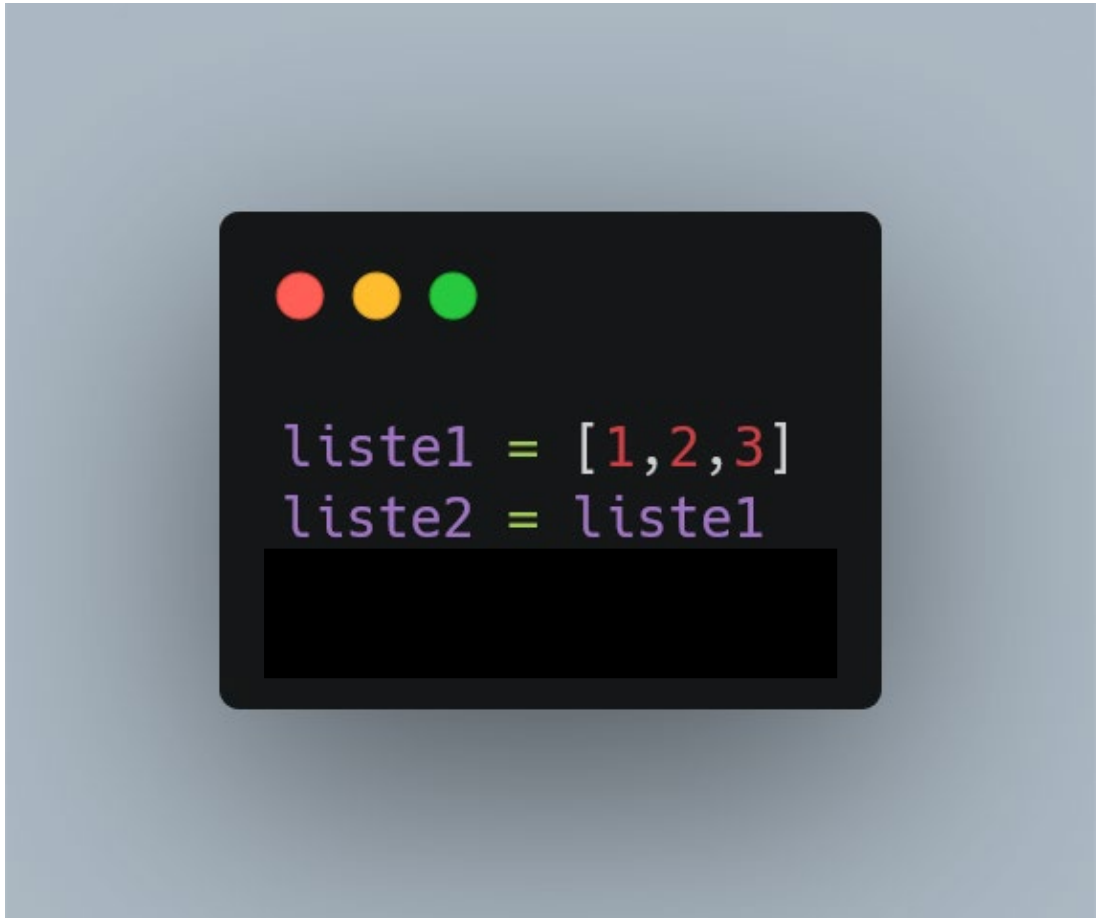
Variablen und Listen in Python



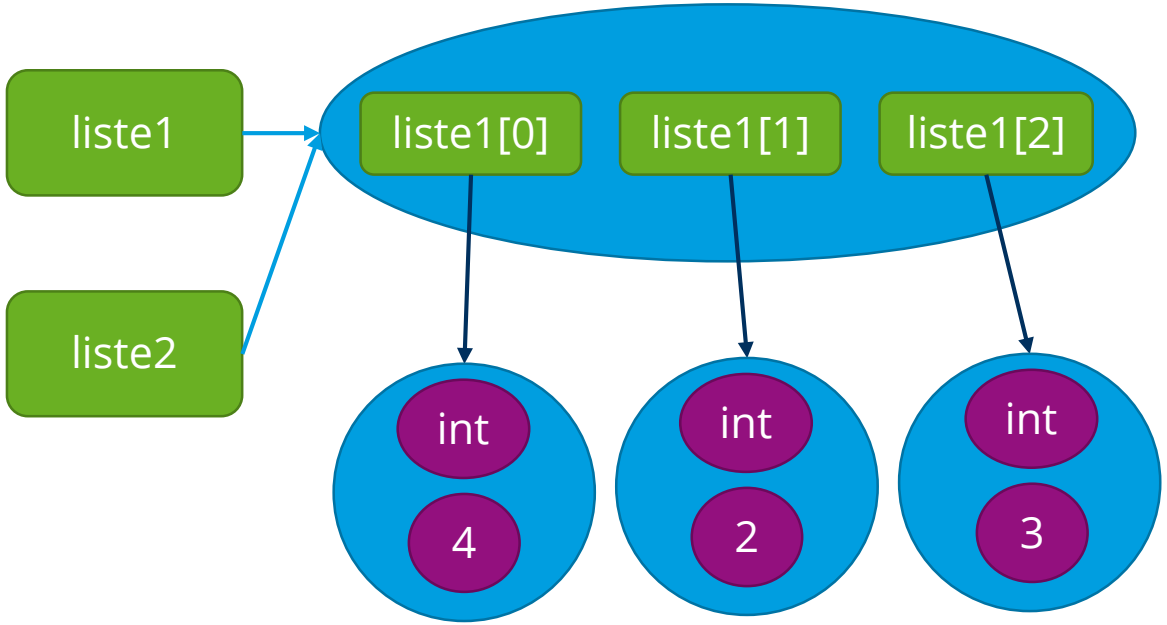


Variablen und Listen in Python



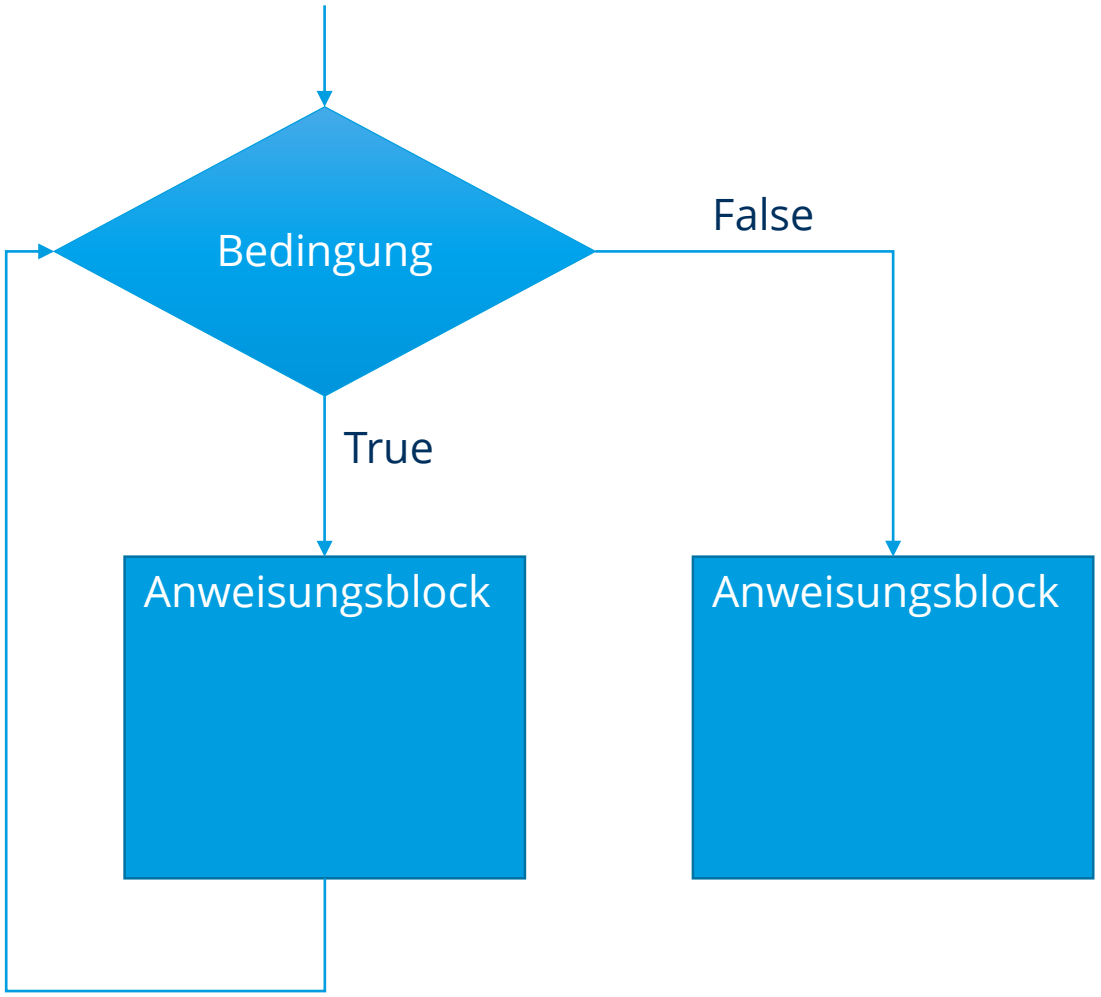


Variablen und Listen in Python




```
liste1 = [1,2,3]
liste2 = liste1
liste2[0] = 4
```

Schleifen - While



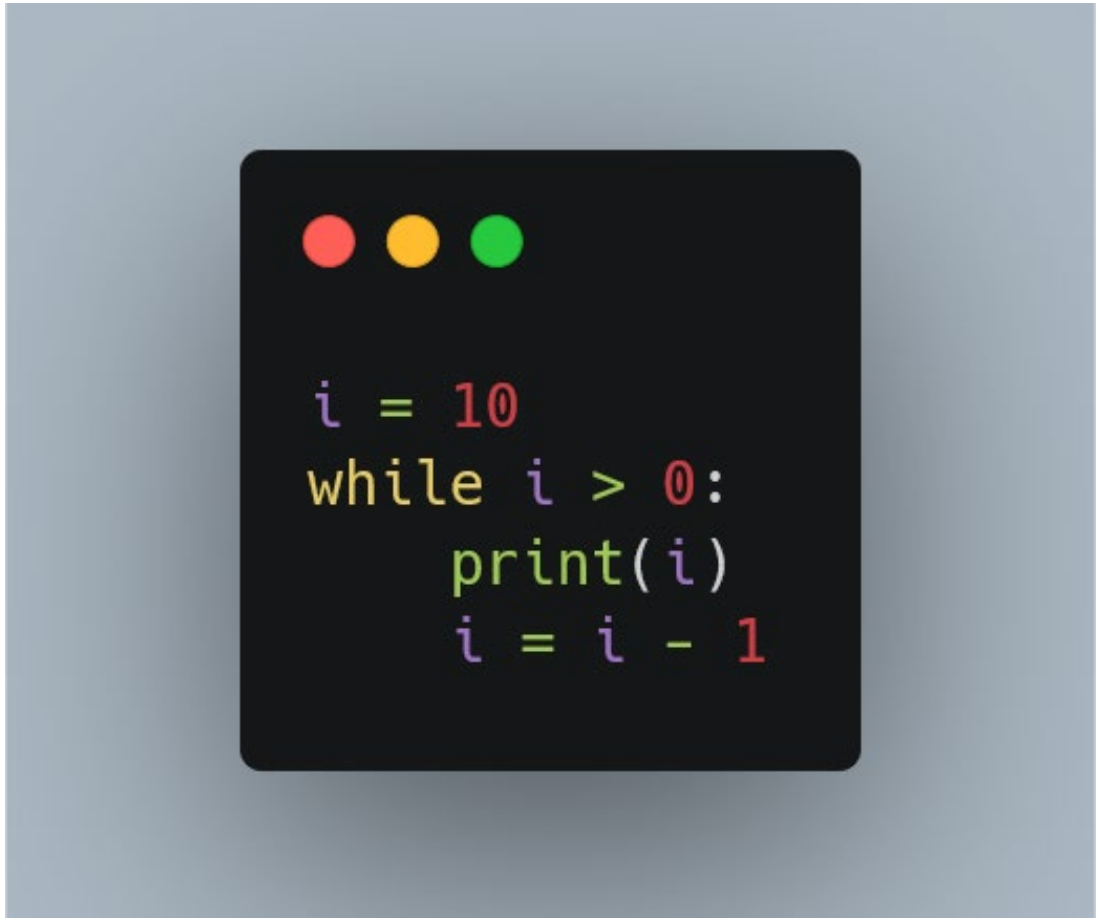
Notation:

while *Bedingung*:
    Anweisung
    ...
    Anweisung

Schleifen - While

Notation:

while *Bedingung*:
    Anweisung
    ...
    Anweisung



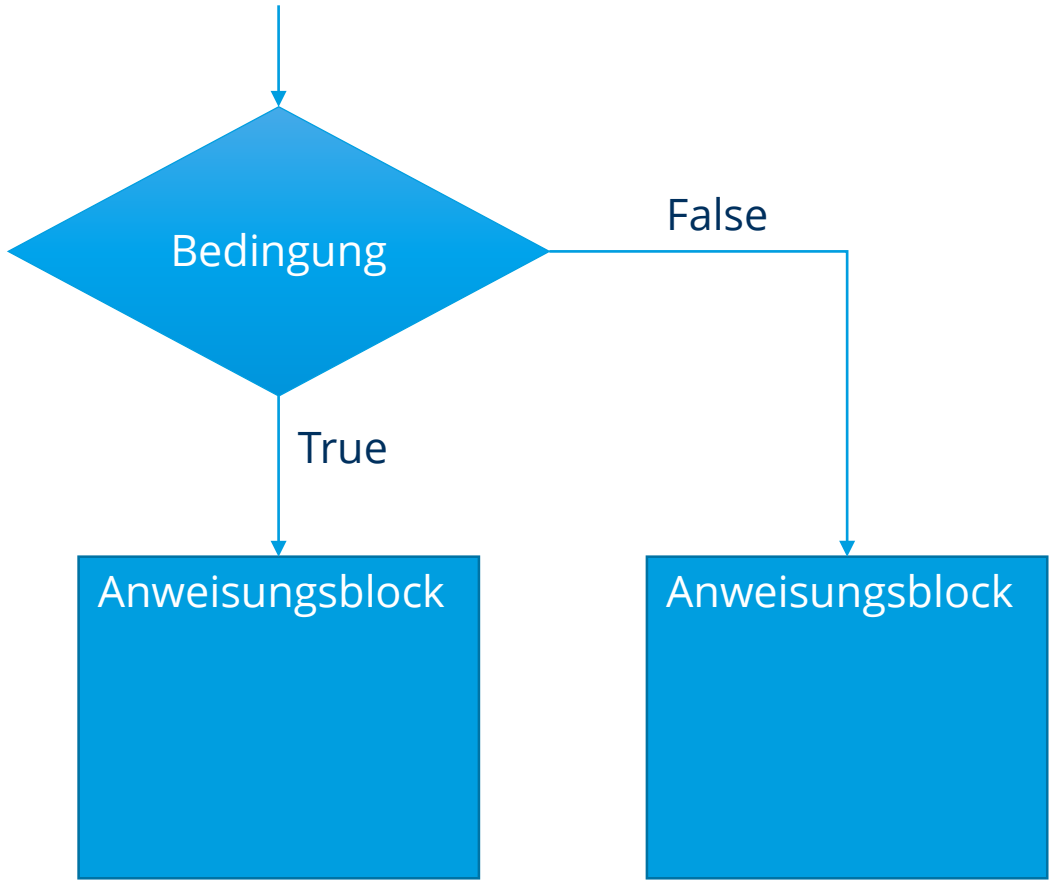
While-berechenbar

- While-berechenbar und Turing-berechenbar sind äquivalent
- Konsequenz:
  - → Jede berechenbare Funktion ist auch while-berechenbar
  - → jedes **beliebige** berechenbare Problem, lässt sich durch ein **WHILE-Programm ausdrücken**
  - → alle weiteren Konstrukte unterstützen die Lesbarkeit, Wartbarkeit und den Entwurf von Software aber beeinflussen **nicht** die **Ausdrucksstärke**

Definition WHILE-Programm

$$
\begin{aligned}
P ::= \quad & x_i = x_j + c \\
\mid \; & x_i = x_j - c \\
\mid \; & P; P \\
\mid \; & \text{WHILE } x_i \neq 0 \text{ DO } P \text{ END}
\end{aligned}
$$

Alternative: If-Bedingung



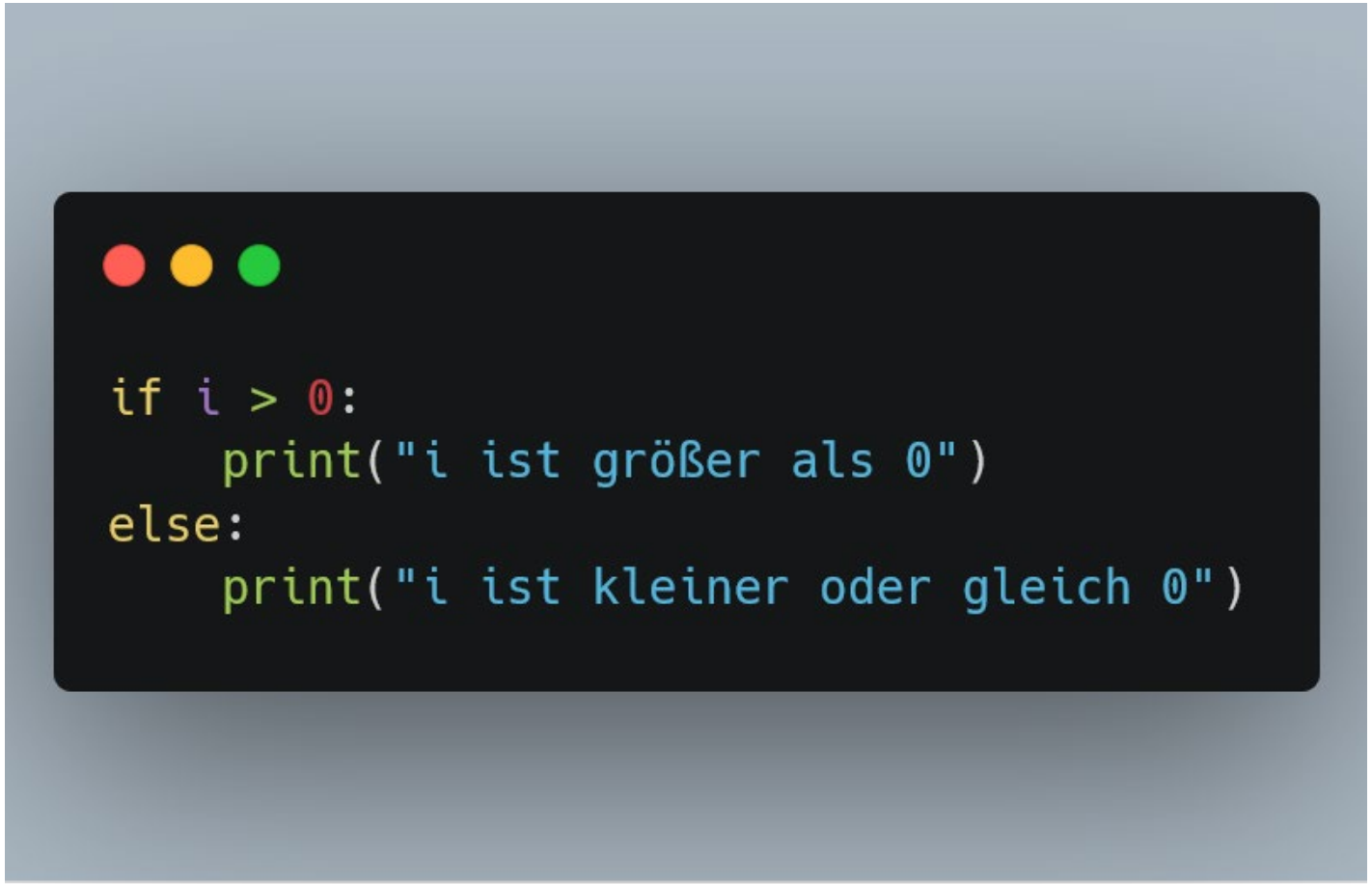
Notation:

if *Bedingung*:
    Anweisung
    ...
    Anweisung
else:
    Anweisung
    ...
    Anweisung

Alternative: If-Bedingung

Notation:

if *Bedingung*:
    Anweisung
    ...
    Anweisung
else:
    Anweisung
    ...
    Anweisung



```
if i > 0:
    print("i ist größer als 0")
else:
    print("i ist kleiner oder gleich 0")
```

Was Sie behalten und wiederholen sollten

- Imperatives Programmierkonzept
- Was ist ein Programm?
  - Algorithmus?, Datenstruktur?, Bedienschnittstelle?
- Mensch-Maschine-Kommunikation
  - Syntax, Semantik, Pragmatik
- Vom Quellcode zum ausführbaren Programm *in Python*
  - Compiler, Interpreter, Virtuelle Maschine
- Variablenkonzept *in Python*