

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

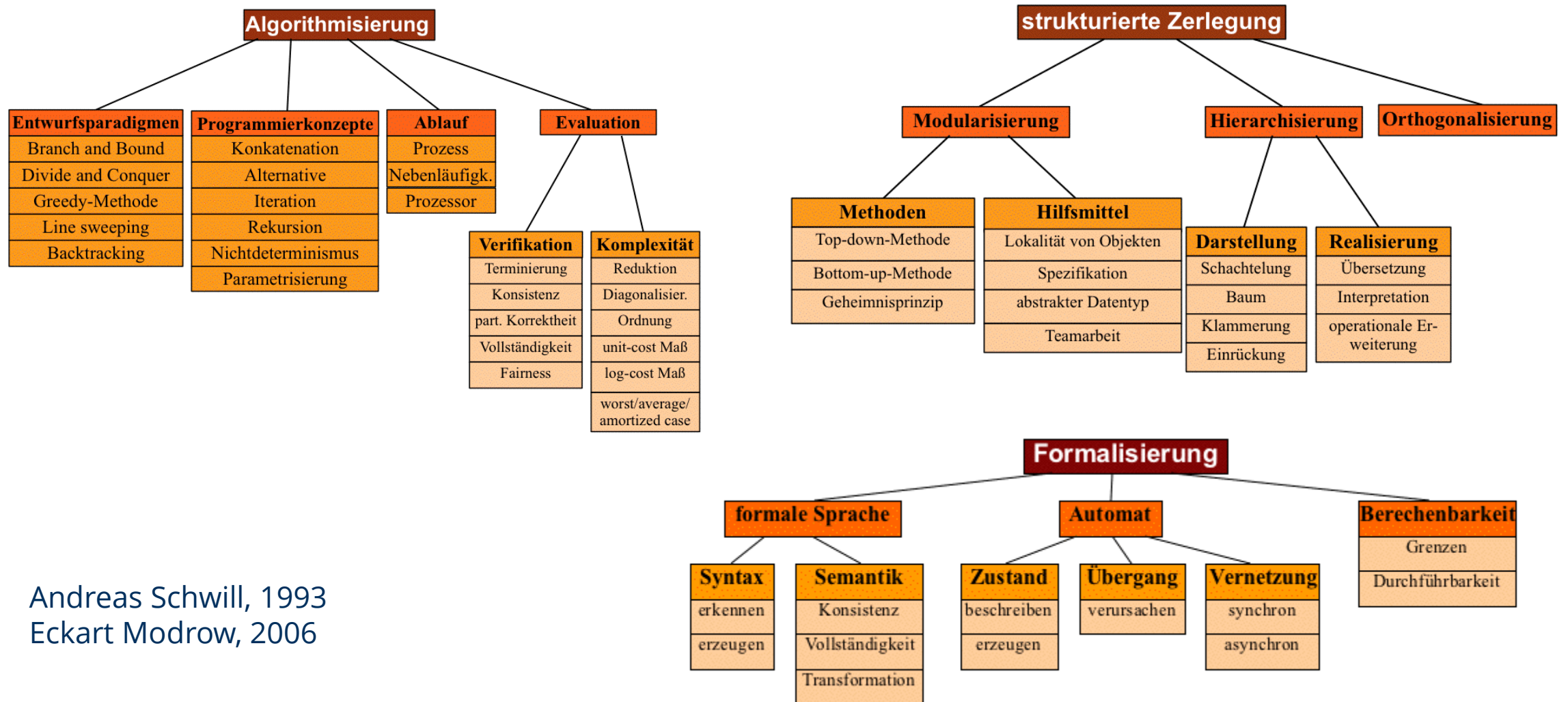
Programmierung

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

Unterstützung

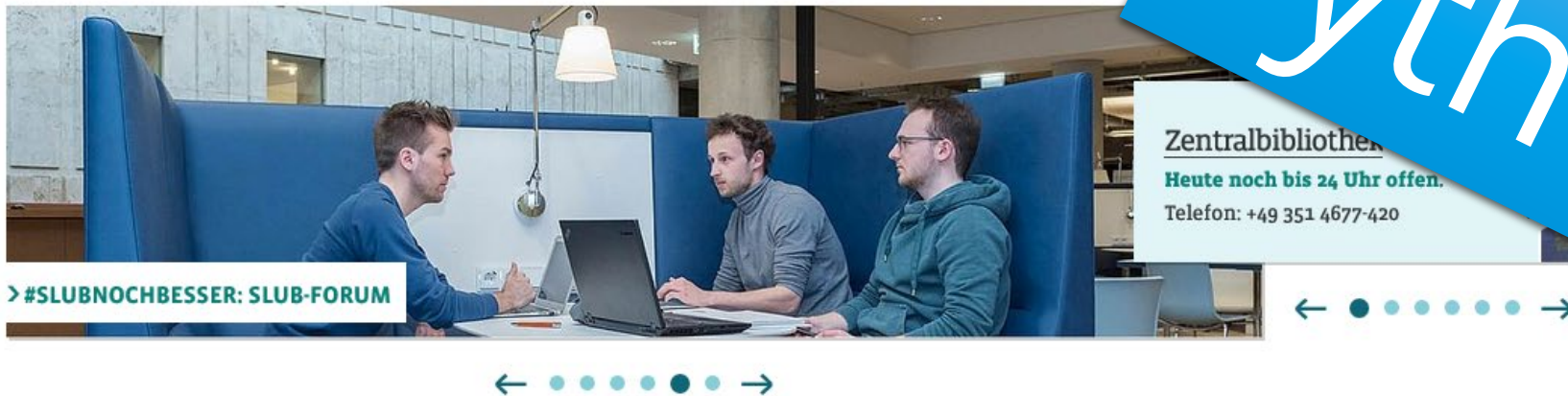
- vielfältige Betreuung und Hilfestellung für Ihr eigenständiges Lernen
- ausführliche Beratung in Ihren Übungen

Fundamentale Ideen der Informatik



Andreas Schwil, 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

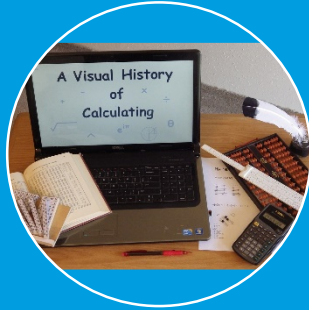
```
extract_number_and_incr(destination, source) int
*destination; unsigned char **source; { extract_number_and_incr(destination, *source); *source += 2; #ifndef EXTRACT_MACROS #undef EXTRACT_NUMBER_AND_INCR #define EXTRACT_NUMBER_AND_INCR(dest, src) \ extract_number_and_incr (&dest, &src) #endif /* not EXTRACT_MACROS */ #endif /* DEBUG */ // If DEBUG is defined, Regex prints many voluminous messages about what it is doing (if the variable 'debug' is nonzero). If linked with the main program in 'iregex.c', you can enter patterns and strings interactively. And if linked with the main program in 'main.c' and the other test files, you can run the already-written tests. */ #ifdef DEBUG /* We use standard I/O for debugging. */ #include <stdio.h> /* It is useful to test things that "must" be true when debugging. */ #include <assert.h> static int debug = 0; #define DEBUG_STATEMENT(e) e #define DEBUG_PRINT1(x) if (debug) printf (x) #define DEBUG_PRINT2(x1, x2) if (debug) printf (x1, x2) #define DEBUG_PRINT3(x1, x2, x3) if (debug) printf (x1, x2, x3) #define DEBUG_PRINT4(x1, x2, x3, x4) if (debug) printf (x1, x2, x3, x4) #define DEBUG_PRINT_COMPILED_PATTERN(p, s, e) \ if (debug) print_partial_compiled_pattern (s, e) #define DEBUG_PRINT_DOUBLE_STRING(w, s1, s2, sz1, s2, sz2) \ if (debug) print_double_string (w, s1, s2, sz1, s2, sz2) extern void printchar(); /* Print the fastmap in human-readable form. */ void print_fastmap (fastmap) char *fastmap; { unsigned was_a_range = 0; unsigned i = 0; while (i < (1 << BYTEWIDTH)) { if (fastmap[i++] { was_a_range = 0; printchar (i - 1); while (i < (1 << BYTEWIDTH) && fastmap[i] { was_a_range = 1; i++; } if (was_a_range) { printf ("-"); printchar (i - 1); } } } /* Print a compiled pattern string in human-readable form, starting at the START pointer into it and ending just before the pointer END. */ void print_partial_compiled_pattern (start, end) unsigned char *start; unsigned char *end; { int mcnt, mcnt2; unsigned char *p = start; unsigned char *pend = end; if (start == NULL) { printf ("(null)\n"); return; } /* Loop over pattern commands. */ while (p < pend) { switch ((re_opcode_t) *p++) { case no_op: printf ("no_op"); break; case exactn: mcnt = *p++; printf ("/exactn/%d", mcnt); do { printchar ("/"); printchar (*p++); } while (--mcnt); break; case start_memory: mcnt = *p++; printf ("/start_memory/%d/%d", mcnt, *p++); break; case stop_memory: mcnt = *p++; printf ("/stop_memory/%d/%d", mcnt, *p++); break; case duplicate: printf ("/duplicate/%d", *p++); break; case anychar: printf ("/anychar"); break; case charset: case charset_not: { register int c; printf ("/charset%$s", (re_opcode_t) *p - 1) == charset_not ? "_not" : ""; assert (p + *p < pend); for (c = 0; c < *p; c++) { unsigned bit; unsigned char map_byte = p[1 + c]; printchar ("/"); for (bit = 0; bit < BYTEWIDTH; bit++) if (map_byte & (1 << bit)) printchar (c * BYTEWIDTH + bit); } p += 1 + *p; break; } case begline: printf ("/begline"); break; case endlime: printf ("/endlime"); break; case on_failure_jump: extract_number_and_incr (&mcnt, &p); printf ("/on_failure_jump/0/%d", mcnt); break; case on_failure_keep_string_jump: extract_number_and_incr (&mcnt, &p); printf ("/on_failure_keep_string_jump/0/%d", mcnt); break; case dummy_failure_jump: extract_number_and_incr (&mcnt, &p); printf ("/dummy_failure_jump/0/%d", mcnt); break; case push_dummy_failure: printf ("/push_dummy_failure"); break; case maybe_pop_jump: extract_number_and_incr (&mcnt, &p); printf ("/maybe_pop_jump/0/%d", mcnt); break; case pop_failure_jump: extract_number_and_incr (&mcnt, &p); printf ("/pop_failure_jump/0/%d", mcnt); break; case jump_past_alt: extract_number_and_incr (&mcnt, &p); printf ("/
```

Python-spezifische Inhalte



Was bedeutet *imperatives* Programmieren?

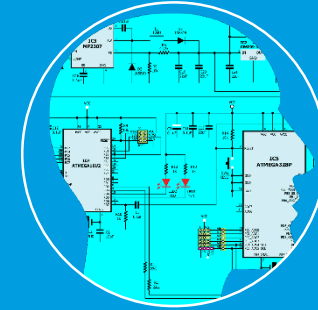
Definition Programm



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**

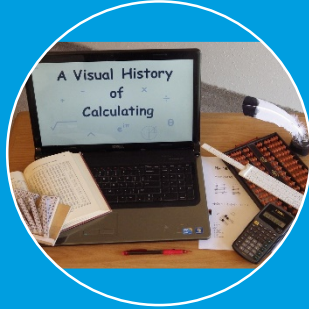
- Eindeutig beschrieben
- Terminierend
- Determiniert
- Korrekt
- Effizient
 - Speicherbedarf
 - Laufzeit
 - Technische Plattform
- Verständlich, nachvollziehbar
- Wiederverwendbar, übertragbar
- ...

Beispiele

- Wahlvorschriften
- Spielanleitungen
- Kochrezepte
- Systematisches Suchen, Sortieren, ...
- Rechenvorschriften:
 - Differenzieren
 - Lösen von Gleichungen
 - Bestimmen von Determinanten
- Ablauffähiger Algorithmus
 - Programm beschrieben in einer Programmiersprache



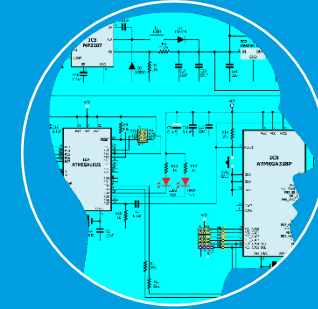
Definition Programm



Algorithmus
(Verhalten)



Datenstruktur
(Modell)



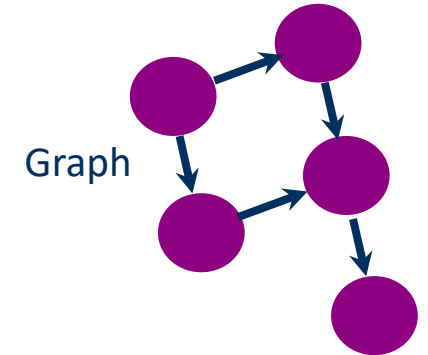
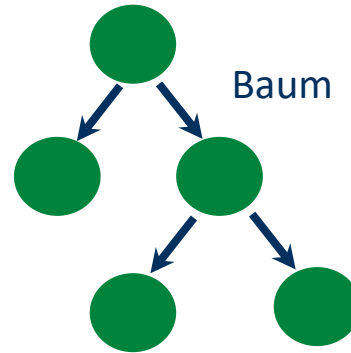
Bedienschnittstelle



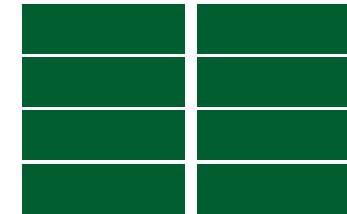
Was ist eine *Datenstruktur*?

Modell der relevanten Aspekte des Einsatzgebiets

- Art, Daten zu verwalten und miteinander zu verknüpfen
- unterschieden nach Zugriffsart, Einfüge- und Löschooperationen, ...



Tabelle



Liste



Stapel



Warteschlange





Daten finden - Wie machen Sie das?

53

880

76

44

79

4



Daten finden - Wie machen Sie das?





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

53



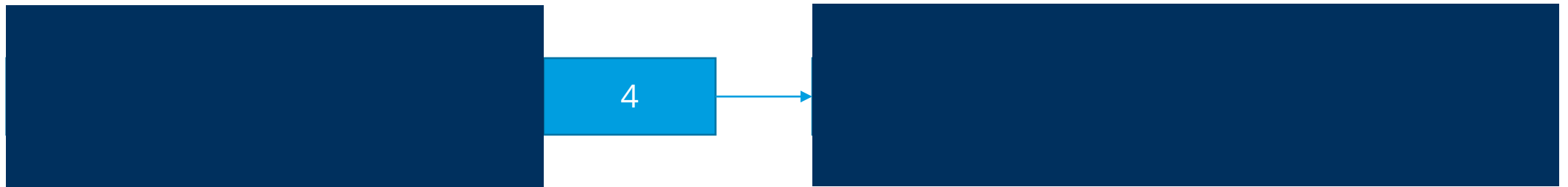


Daten finden





Daten finden



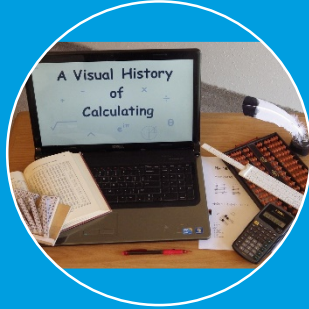


Daten finden





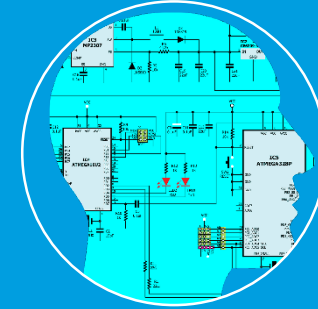
Definition Programm



Algorithmus
(Verhalten)



Datenstruktur
(Modell)

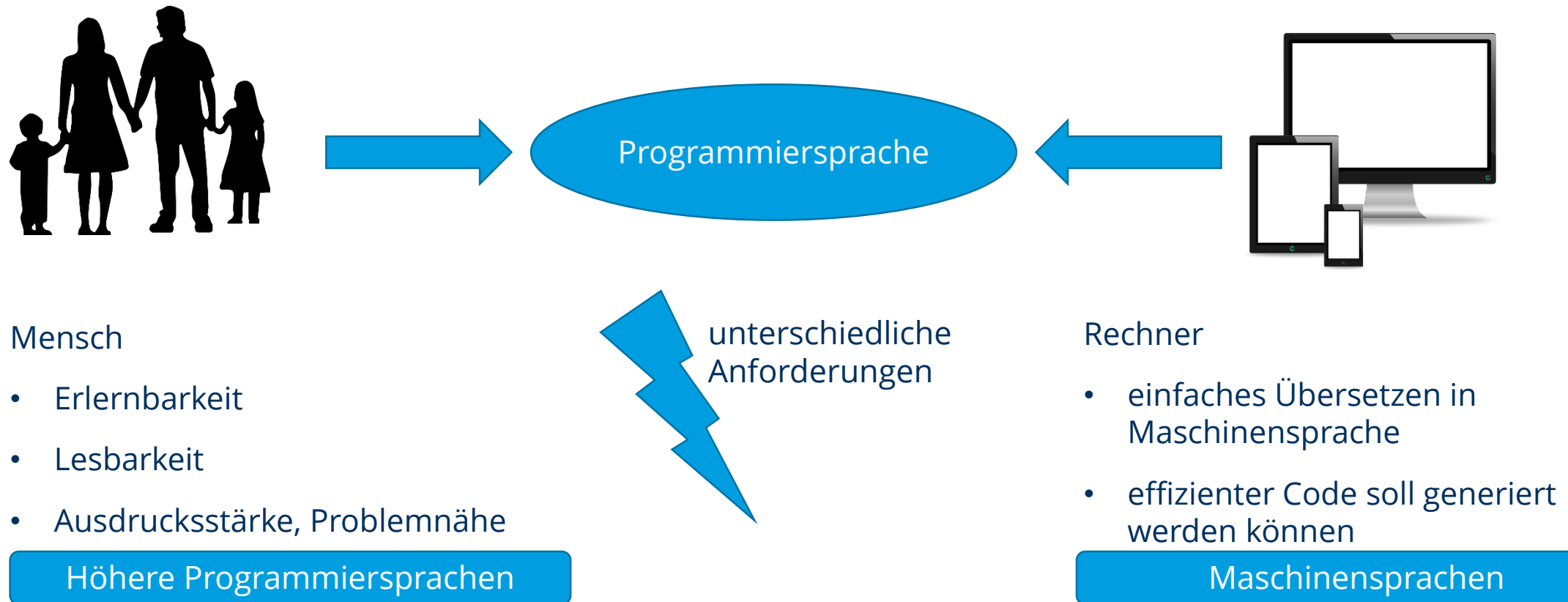


Bedienschnittstelle



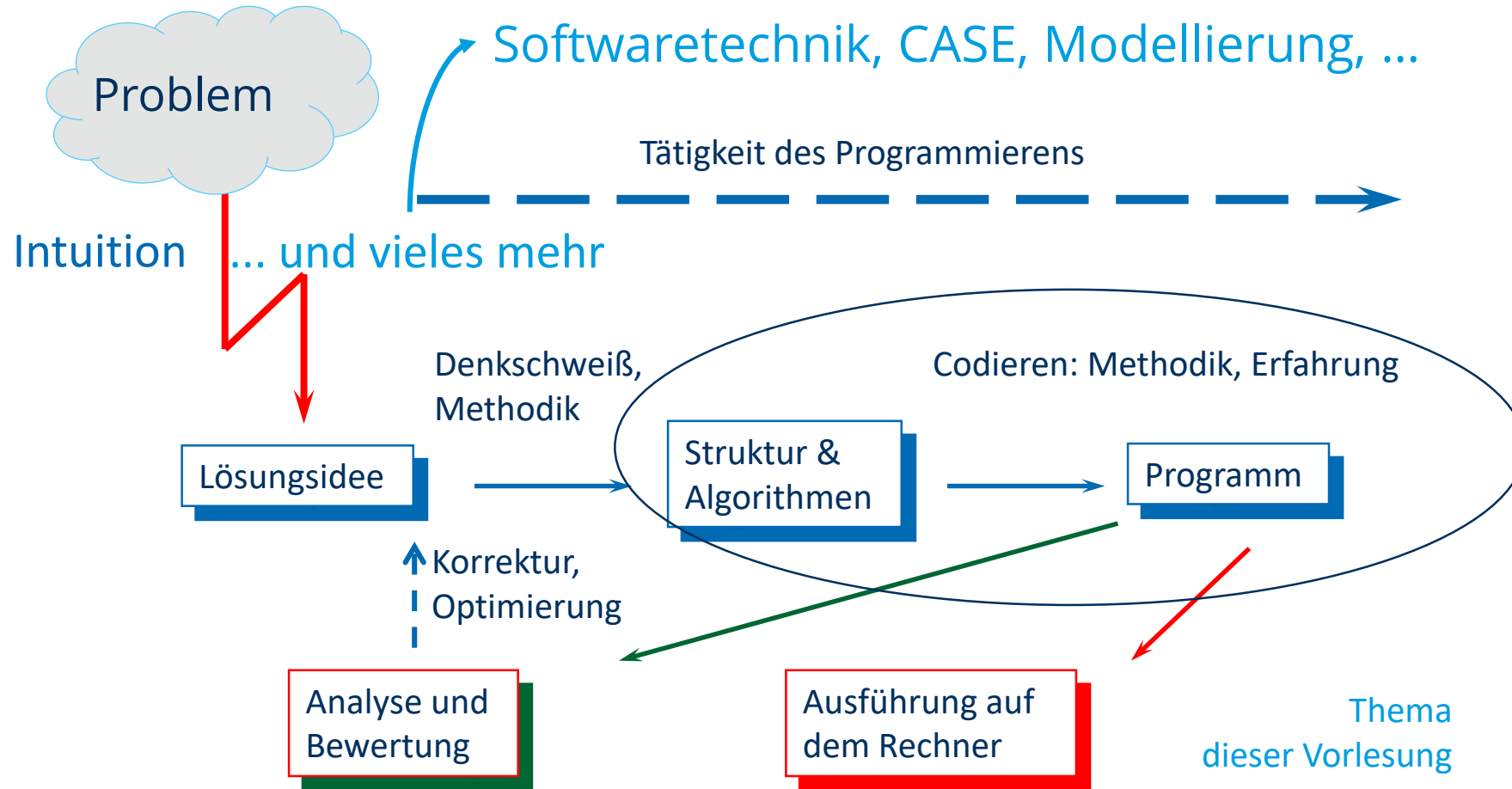
Was bedeutet *Schnittstelle*?

Die Programmiersprache bildet die Schnittstelle zwischen Mensch und Rechner.





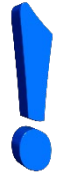
Vom **Problem** zum **Programm** und wieder zurück



Programmieren im Kleinen



Imperatives Programmierkonzept



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



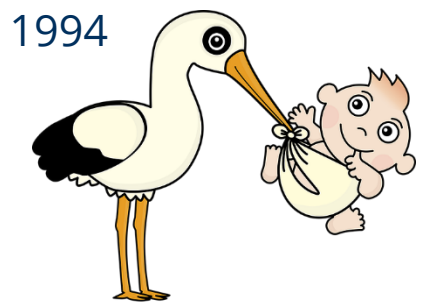
Weitere Programmierparadigmen:

- Deklarative Programmierparadigma
- Objektorientierte Programmierparadigma
- ...

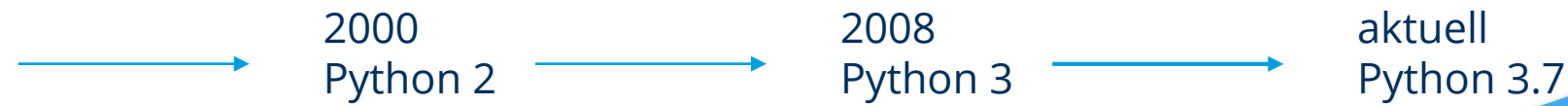
Keine scharfe Trennung viele Sprachen enthalten Aspekte aus mehreren Programmierparadigmen.

Programmiersprache *Python*

Geschichte



Guido van Rossum



damit arbeiten wir



freie Lizenz



plattformunabhängig



Find, install and publish Python packages with the Python Package Index

Search projects

Or browse projects

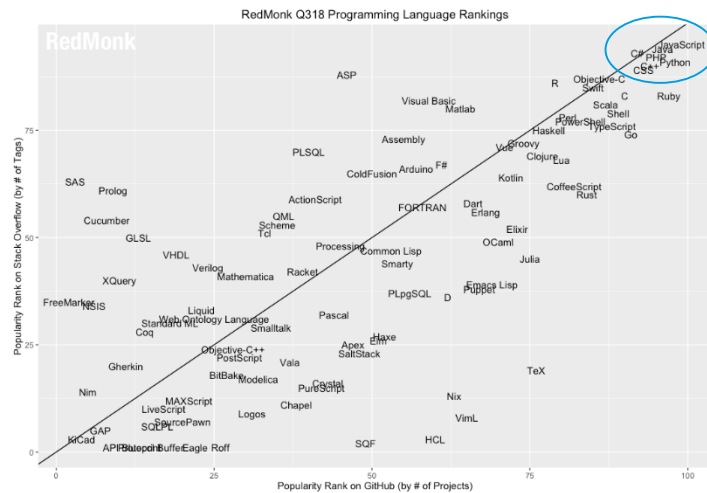
172,478 projects 1,253,087 releases 1,780,508 files 312,454 users

große Community
viele Packages

Popolarität 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/>

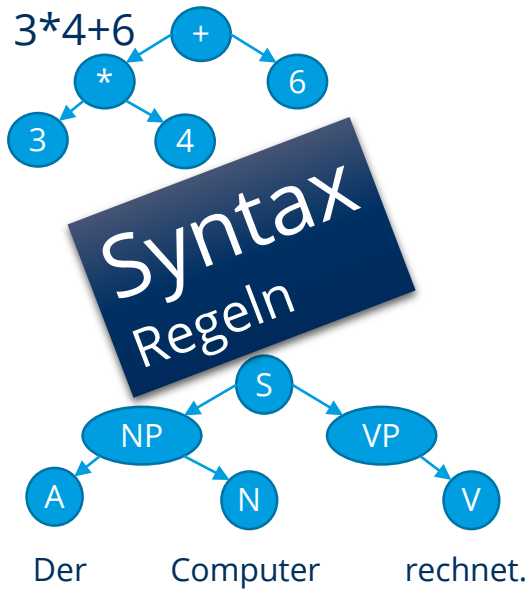
Worldwide, Feb 2019 compared to a year ago:

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



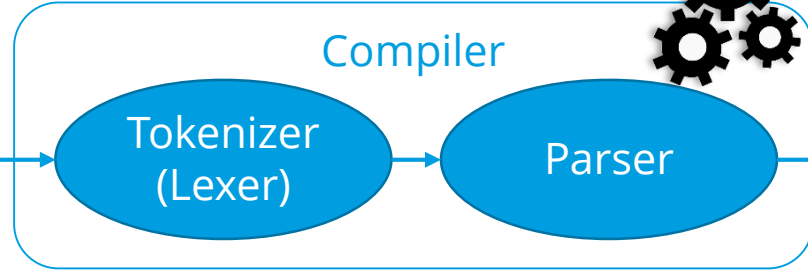
Pragmatik

- 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 x
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
7
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



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...

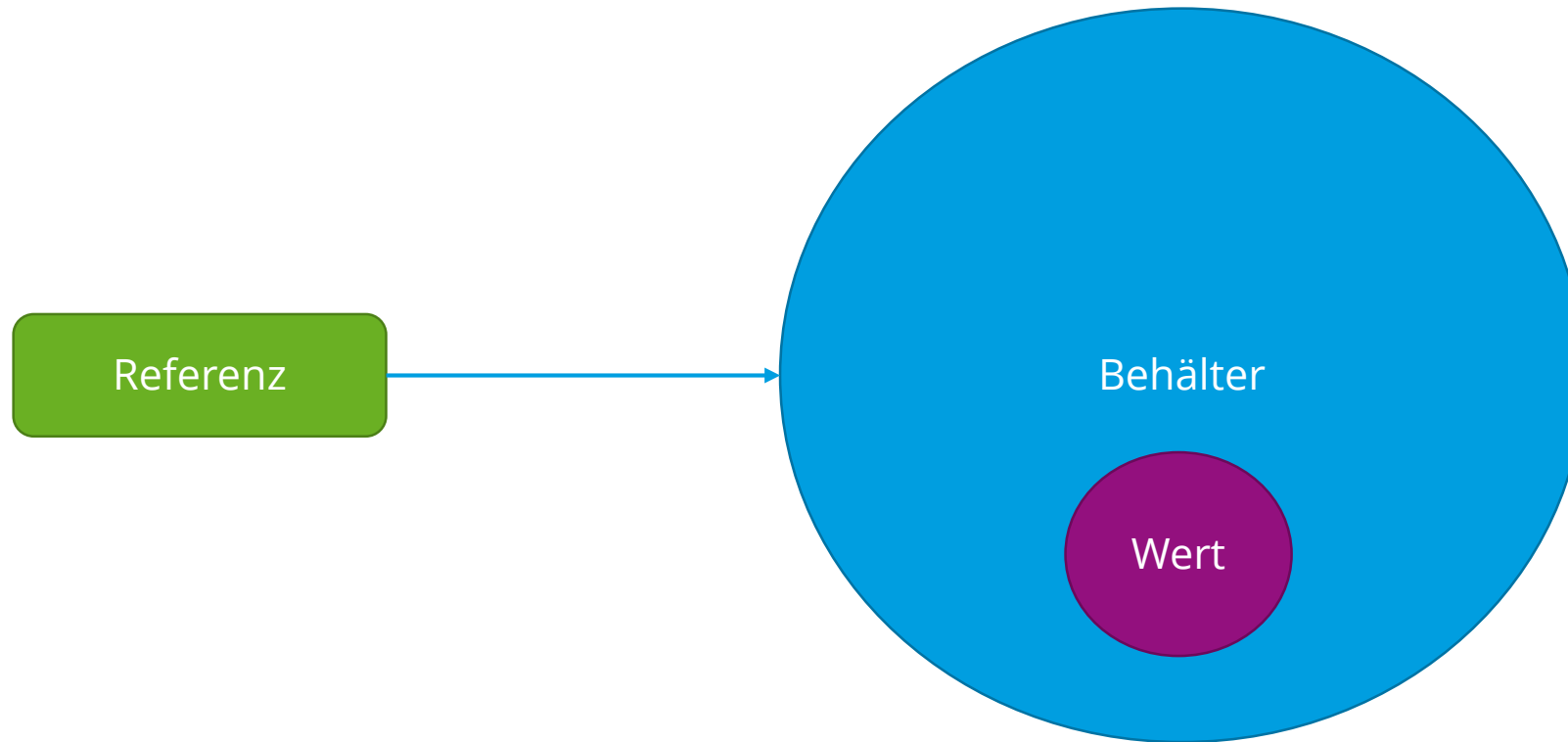
...die wiederum **Anweisungen**
enthalten.

“Puh!”

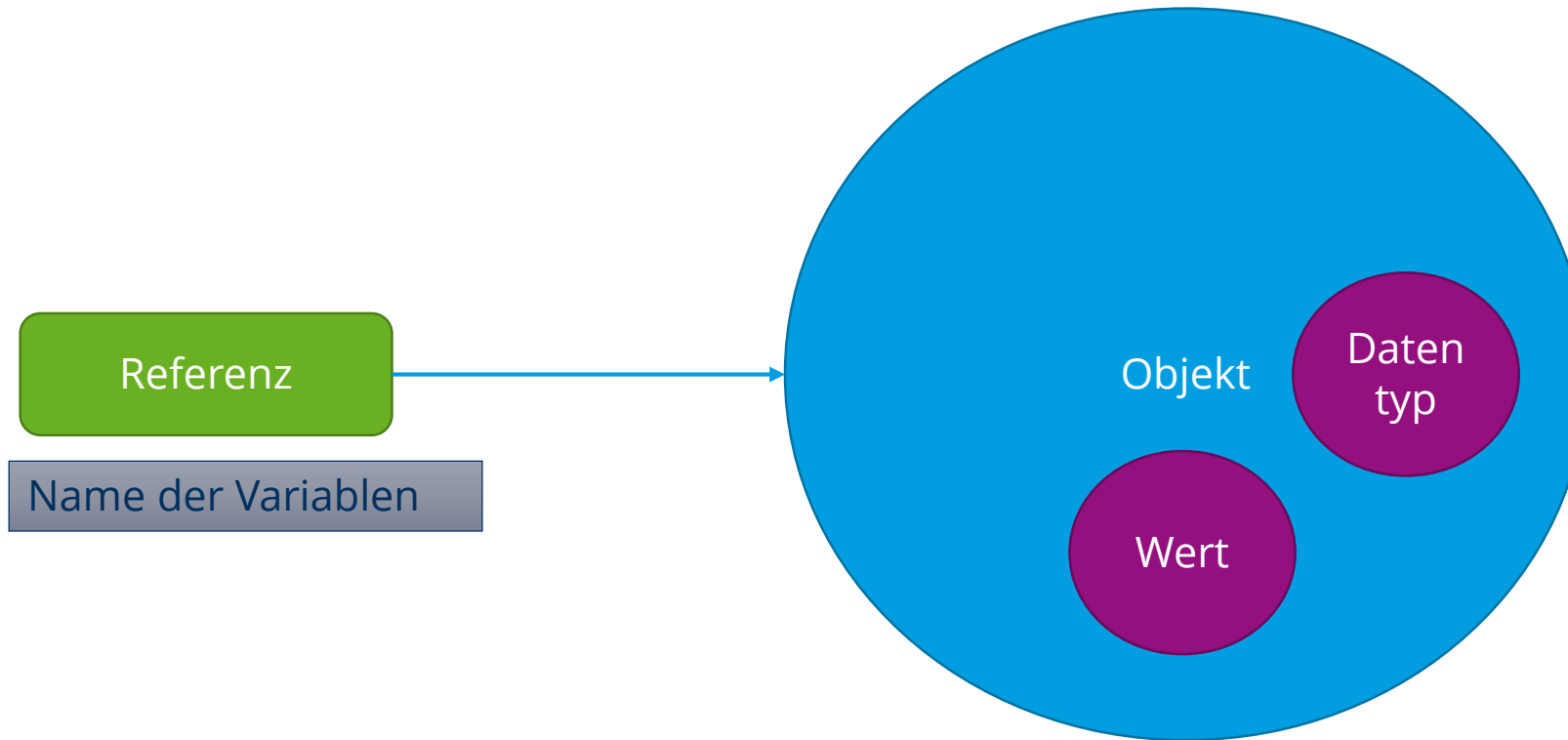




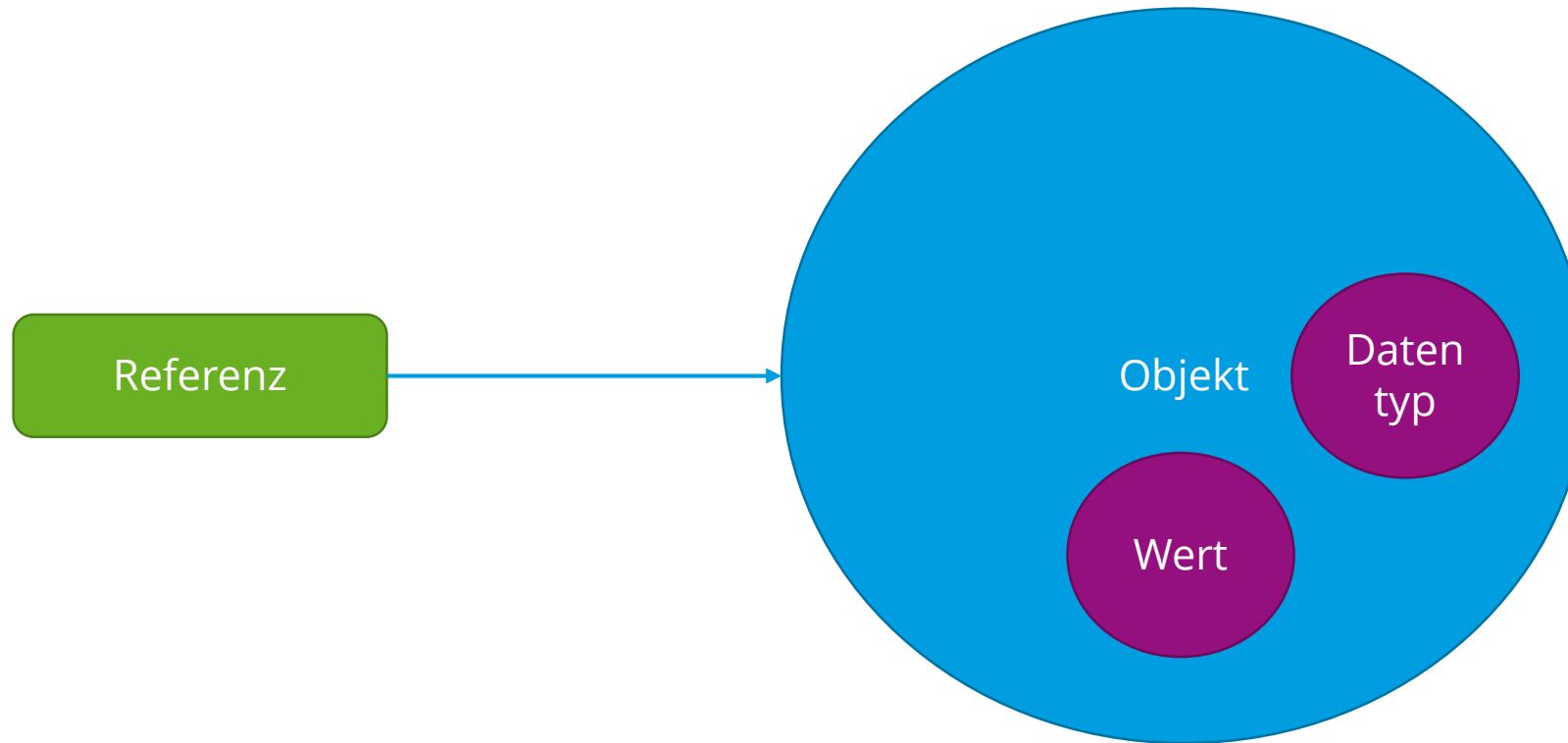
Variablen in imperativen Programmiersprachen



Variablen in Python

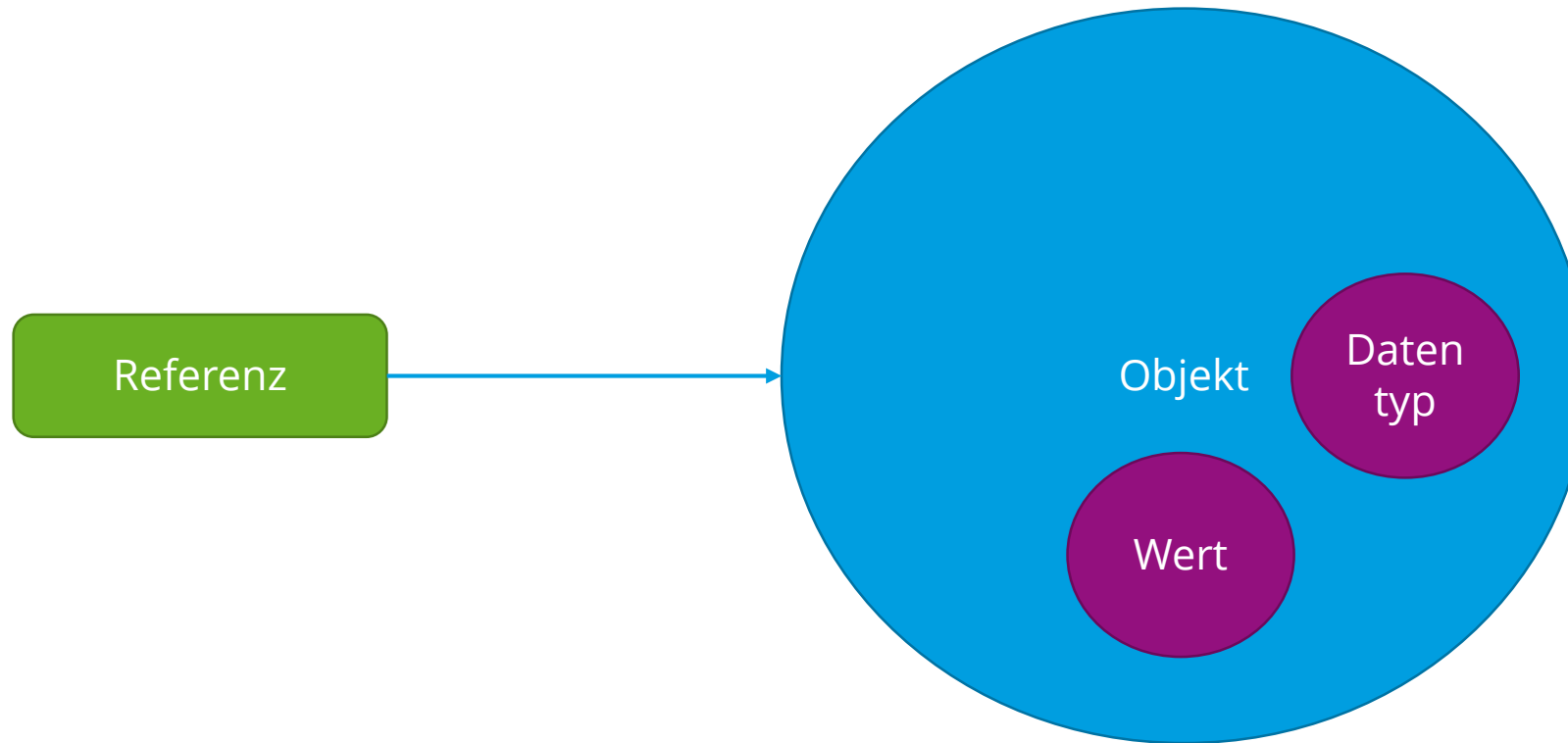


Variablen in Python



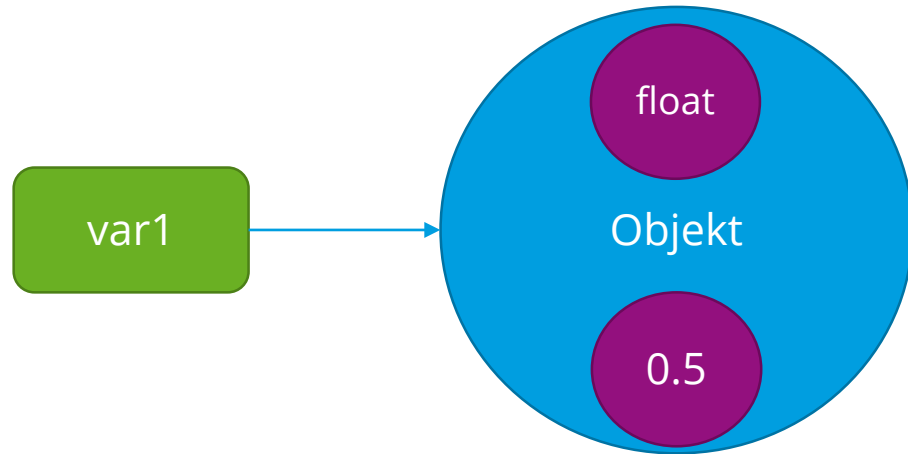
```
ausgabe_text = "Ich bin eine Ausgabe"
```

Variablen in Python

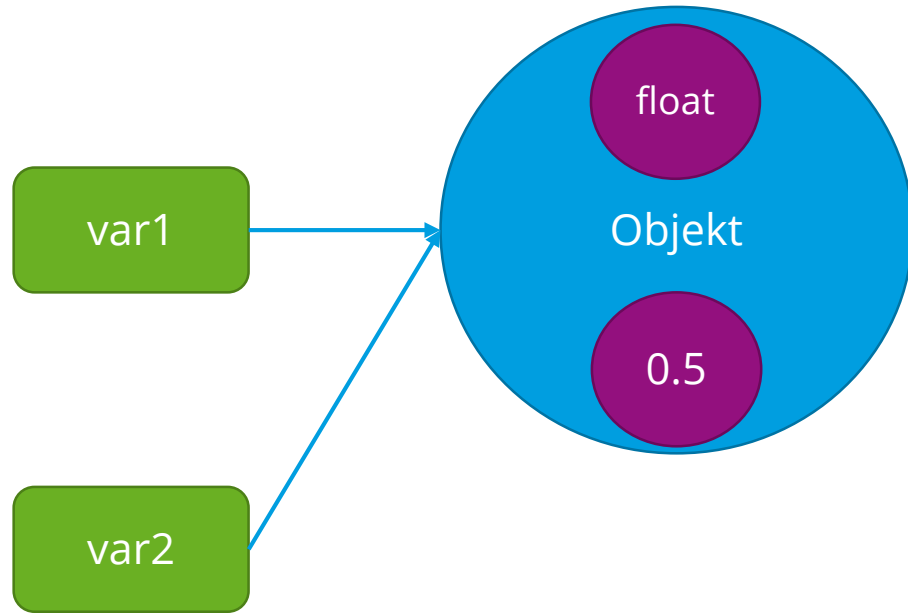


`eingabe = 4.2`

Variablen in Python



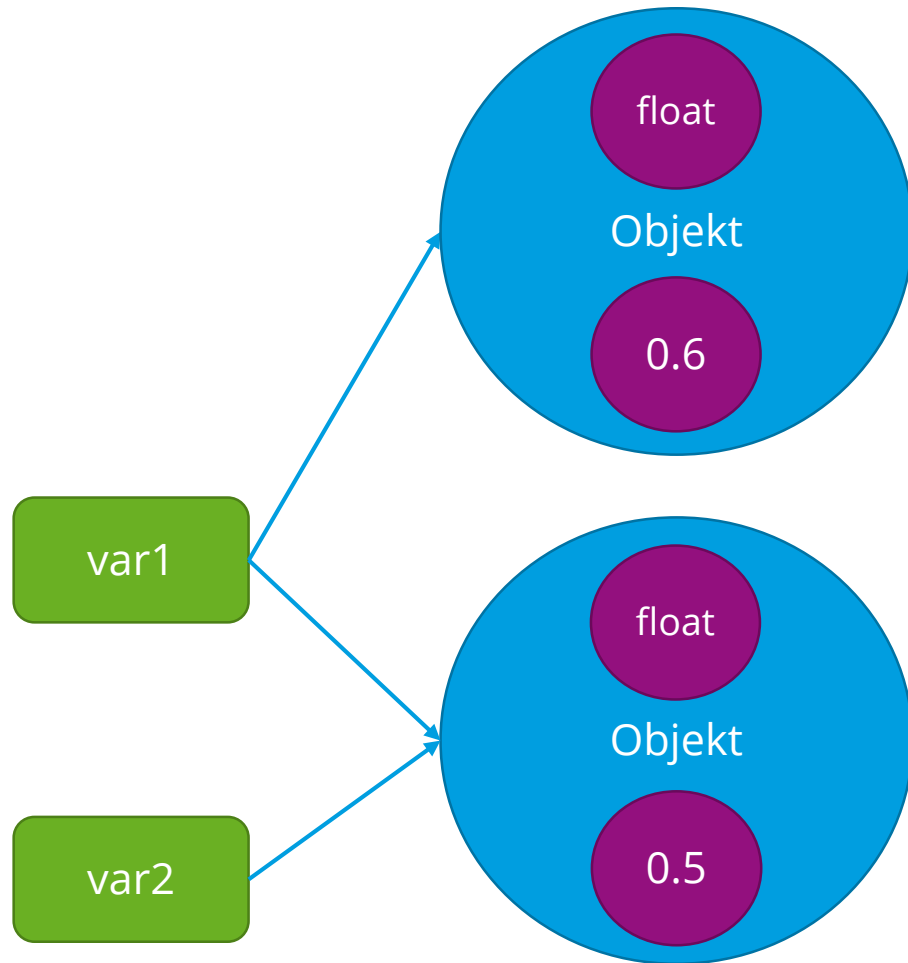
Variablen in Python



```
var1 = 0.5  
var2 = var1
```

A terminal window with a dark background and three colored window control buttons (red, yellow, green) at the top. It displays two lines of Python code: 'var1 = 0.5' and 'var2 = var1'. Below the code is a solid blue horizontal bar.

Variablen in Python



```
var1 = 0.5
var2 = var1
var1 = 0.6
```



Listen von Objekten in Python

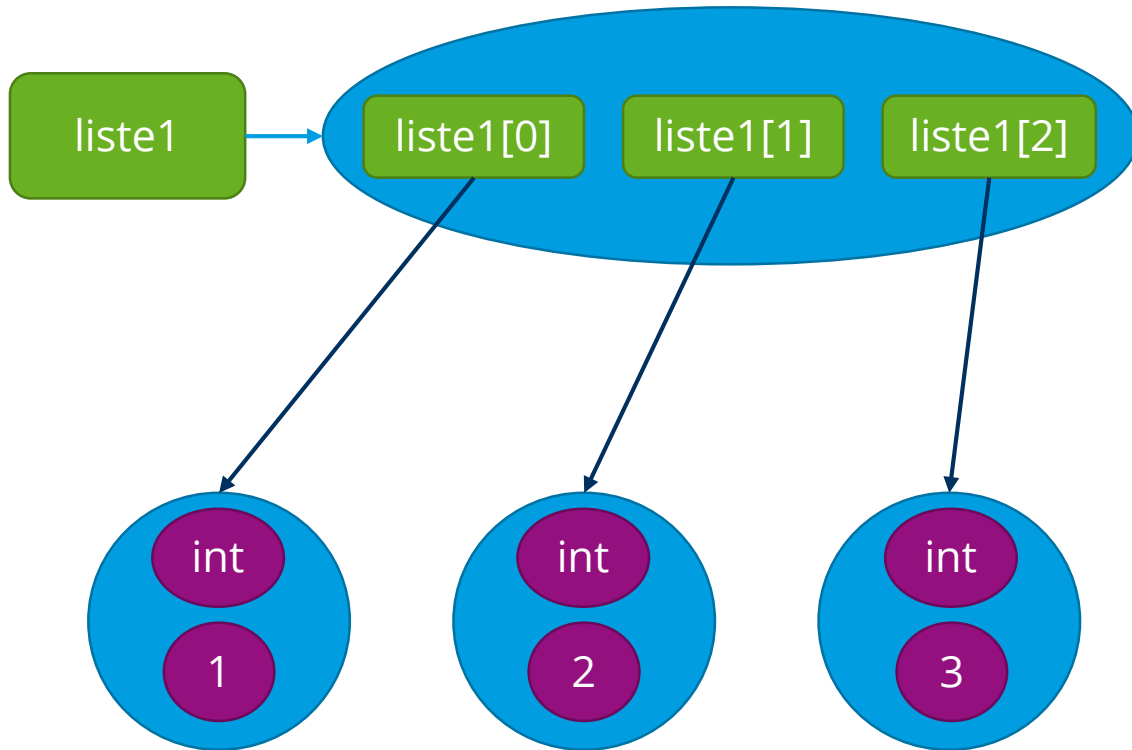
- Eine Liste kann eine Folge beliebiger Objekte gespeichert werden
 - Zum Beispiel Strings, Integers, Float-Zahlen aber auch Listen und Tupel selbst.
- Listen Notation: Eckige Klammern [] und mit Kommata getrennt.



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

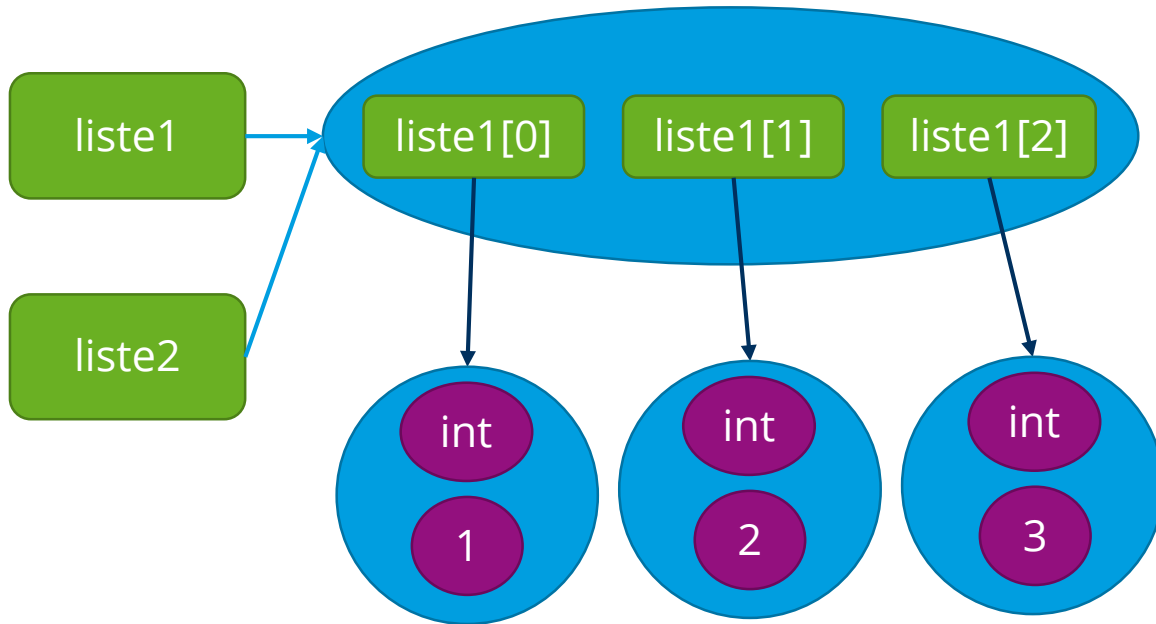
print(mix_liste[2])
text_liste[0] = 5
```

Variablen und Listen in Python



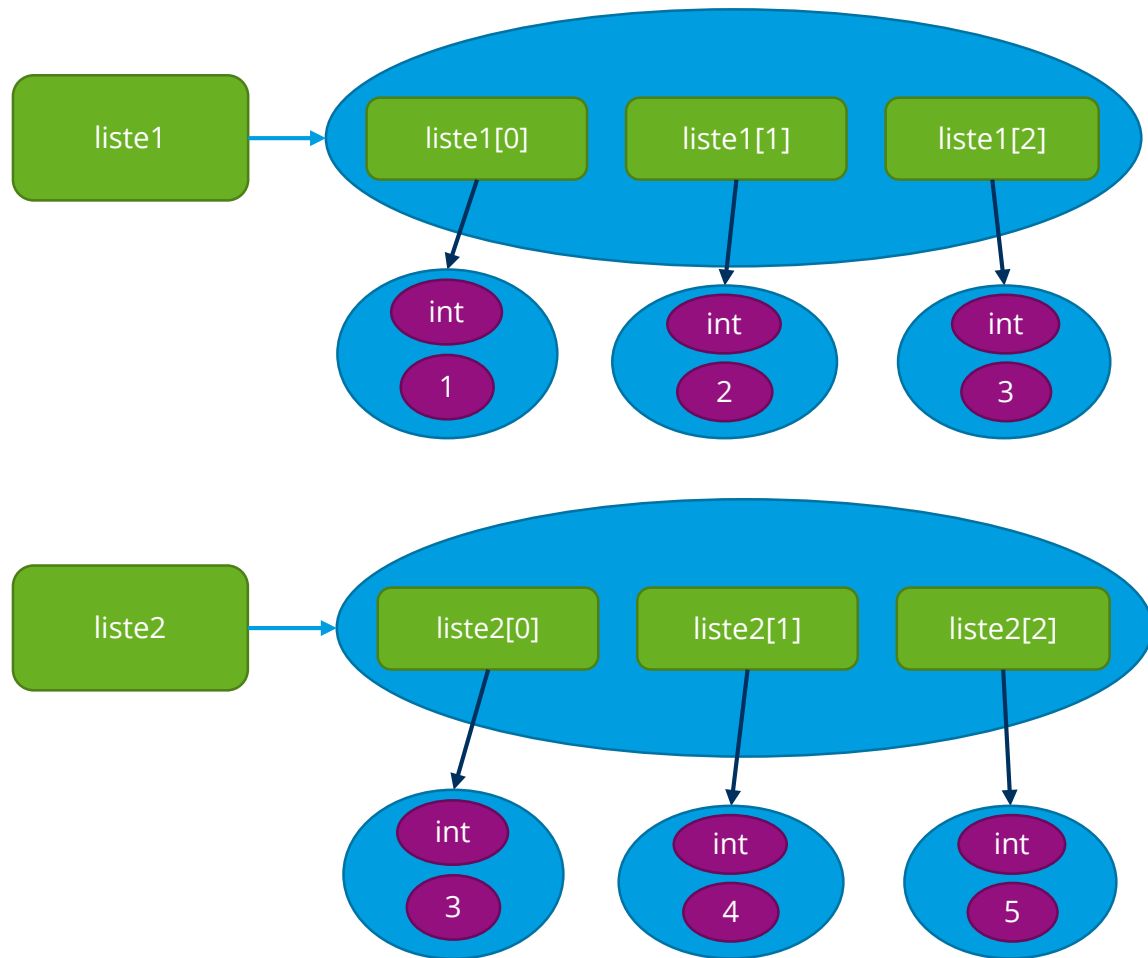
```
liste1 = [1,2,3]
```

Variablen und Listen in Python



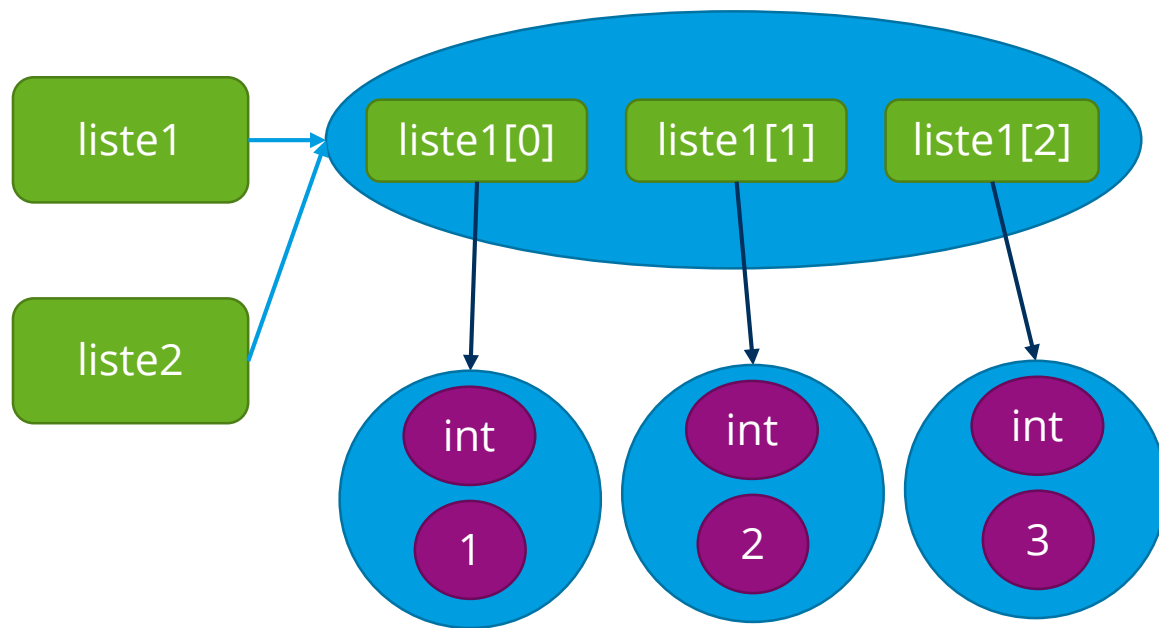
```
liste1 = [1,2,3]
liste2 = liste1
```

Variablen und Listen in Python



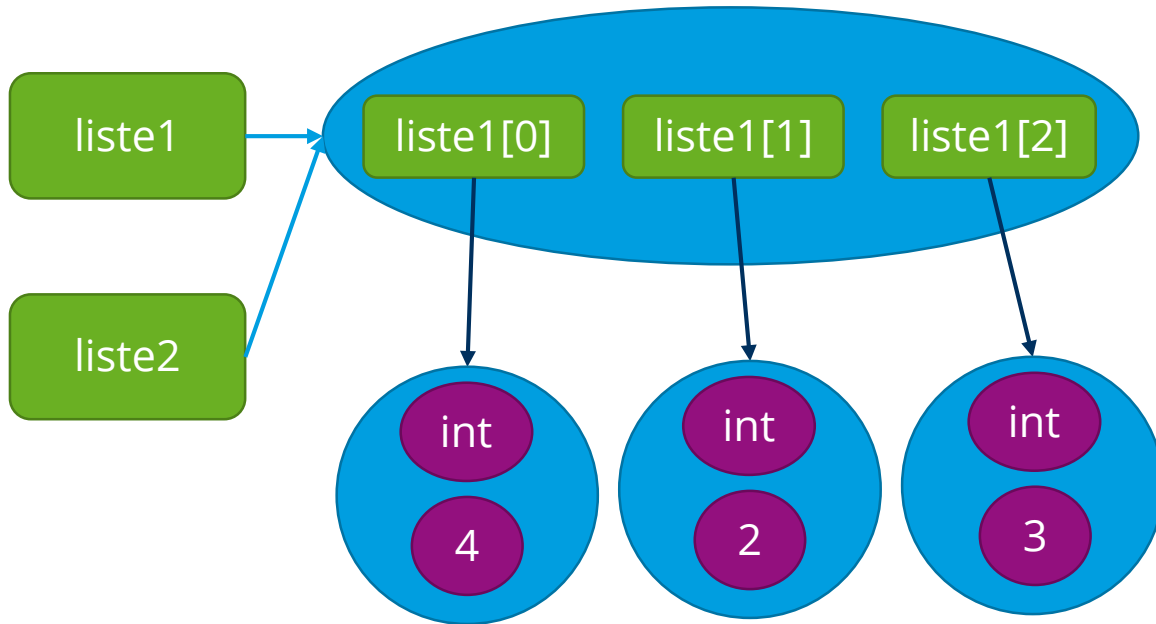
```
liste1 = [1, 2, 3]
liste2 = liste1
liste2 = [3, 4, 5]
```

Variablen und Listen in Python



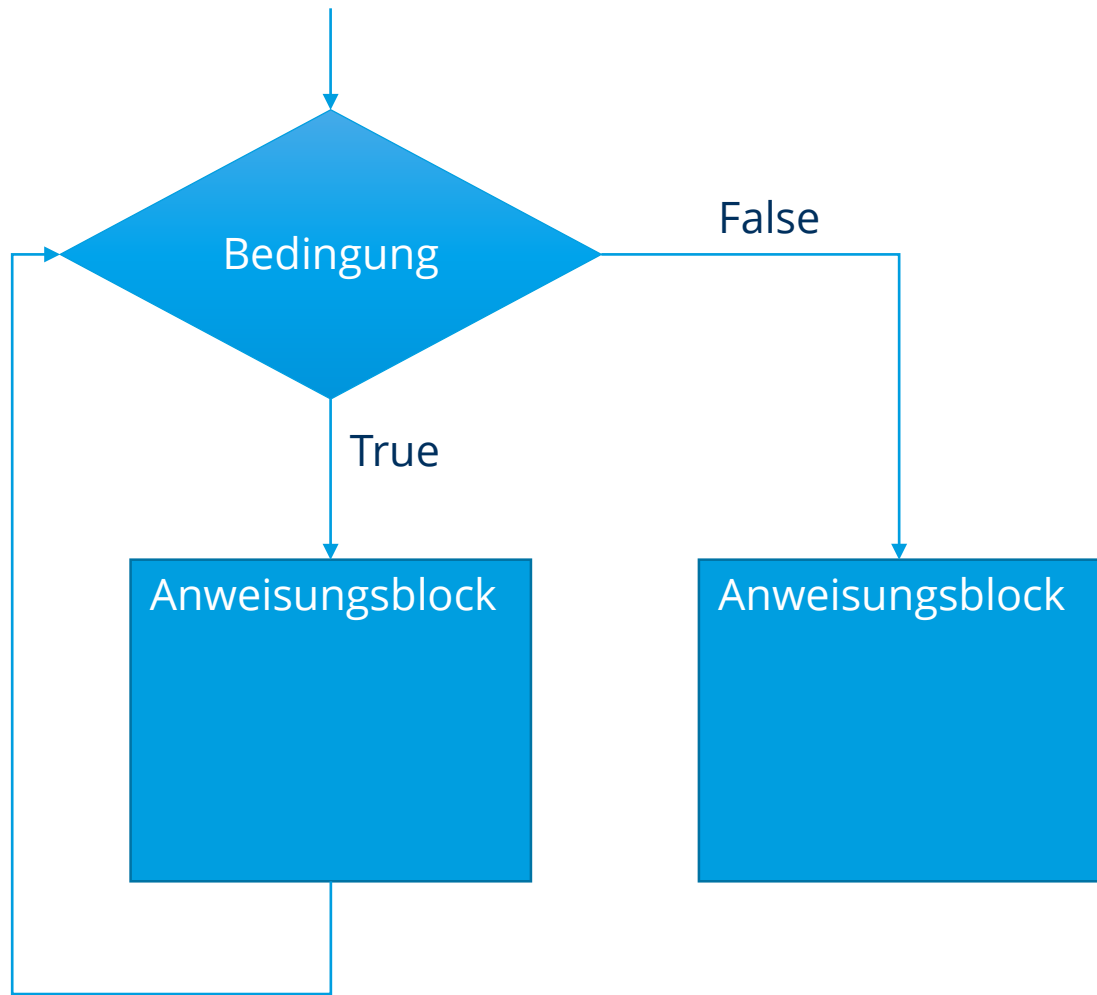
```
liste1 = [1,2,3]
liste2 = liste1
```

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
```

```
i = 10  
while i > 0:  
    print(i)  
    i = i - 1
```



While-berechenbar

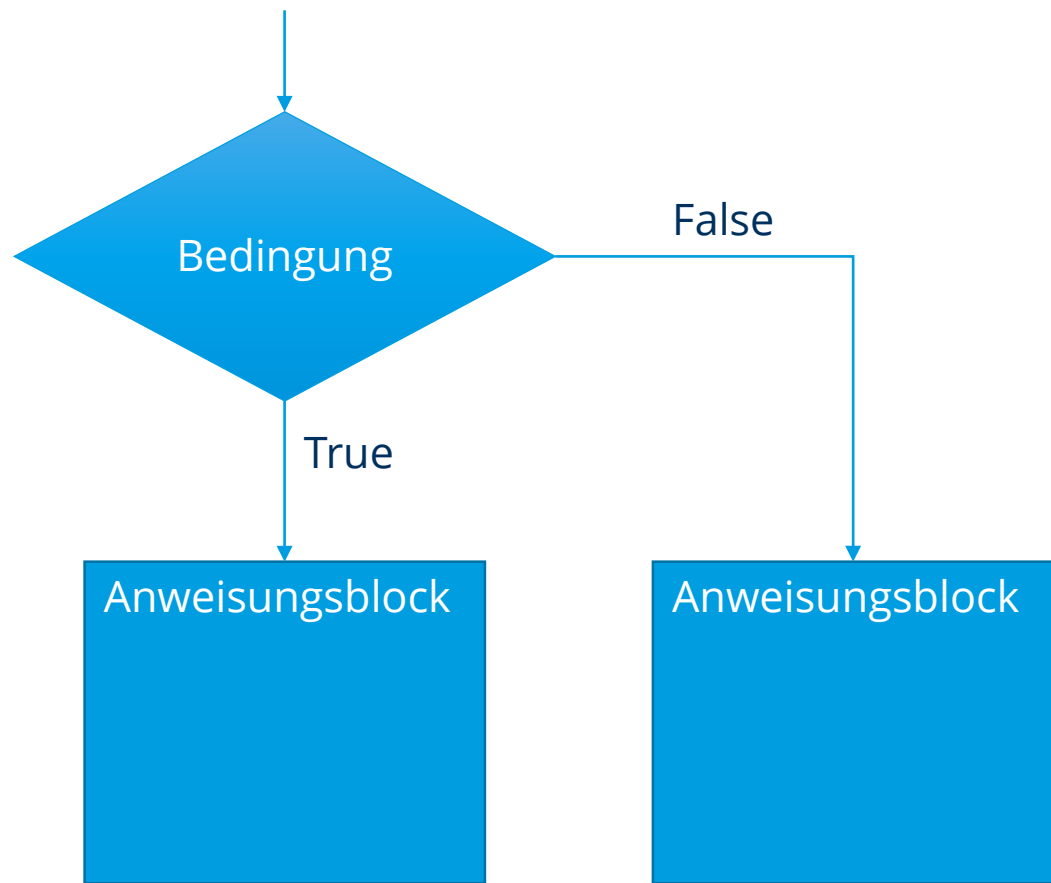
- While-berechenbar und Turing-berechenbar sind äquivalent
- Konsequenz:
 - Jede berechenbare Funktion ist auch while-berechenbar
 - jedes **beliebige** berechenbare Problem, lässt sich durch ein **WHILE-Programm ausdrücken**
 - alle weiteren Konstrukte unterstützen die Lesbarkeit, Wartbarkeit und den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

Definition WHILE-Programm

```
P ::=      Xi = Xj + C
          | Xi = Xj - C
          | P; P
          | WHILE Xi ≠ 0 DO P END
```



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*