Technische Universität Braunschweig

Programmieren 1 – Vorlesung #2

Arne Schmidt

Wiederholung

Python und Datentypen

Pythons Lexik
- Zeichen
- Schlüsselwörter
- Operatoren

Binärzahlen

$(110101)_2 = 53$

Boolean
- Wahrheitswert
- Unäre und Binäreoperatoren

True / False
1 / 0

Integer
- Beliebig groß
- Binäre Operatoren
- Bitweise Operatoren

3179, 5, 891, …

Float
- Beschränkt in Wertebreich **und** Präzision
- Vergleichsoperatoren

$(1.1001101101)_2 \cdot 2^5$

Strings
- Beliebig lang
- Binäre Operatoren + und *

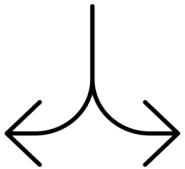
“Das ist ein String <(o.o<)”

Heute

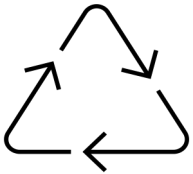
Wie kann ich nun Pythoncode schreiben?

Welche Strukturen stehen mir zur Verfügung?

Anweisungen



Verzweigungen



Schleifen

Kapitel 2.3 – Kontrollstrukturen

Kontrollstrukturen

Konstrollstrukturen geben vor, in welcher Reihenfolge Operationen durchgeführt werden.

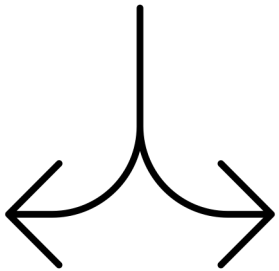
Diese können unterteilt werden in:
- Einfache Anweisungen
- Verzweigungen (bedingte Anweisungen)
- Schleifen (wiederholte Anweisungen)
- Sprünge und Methoden
- Rekursion

Für die imperative Programmierung greift man auf die ersten drei Strutkuren zurück.
Mit diesen lassen sich Sprünge und Methoden ersetzen.

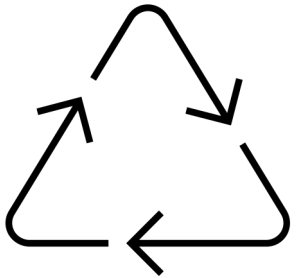
Rekursionen schauen wir uns erst später an.

Kapitel 2.3.1 – Anweisungen, Verzweigungen und Schleifen

Anweisungen

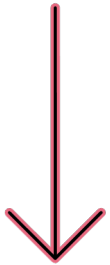


Verzweigungen

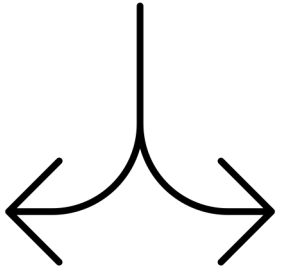


Schleifen

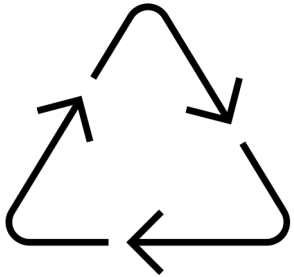
Kontrollstrukturen



Anweisungen



Verzweigungen



Schleifen

Anweisungen

Eine Anweisung besteht aus einer einfachen Operation. Darunter

Deklaration:
Eine Variable erhält in der Regel ein Namen und Typen.

Initialisierung:
Einer Variable wird zum ersten Mal ein Wert zugewiesen, welcher vorgegeben, oder durch Operationen bestimmt wird.

Ein- und Ausgabe:
Eingabe (z.B. input) bzw Ausgabe von Werten. In der Regel benötigt, um Parameter einer Methode zu übergeben, oder Lösungwerte zurückzugeben (return, siehe Kap. 2.3.2).
Eine Ausgabe kann auch auf Konsole (print), in Dateien oder grafisch erfolgen.

```
x = 9
y = 13 * x / 3
a = "x = " + str(x) + ", y = " + str(y)
```

Variablen **müssen** initialisiert werden!

```
x = input("Name: ")
print("x hat den Namen " + x)
return x
```

Anweisungen – Beispiel

Als Struktogramm:

Eingabe: $a \in \mathbb{R},\ b \in \mathbb{R}$
$x = a \cdot a$
$x = x + 2 \cdot a \cdot b$
$x = x + b \cdot b$
Gib x aus

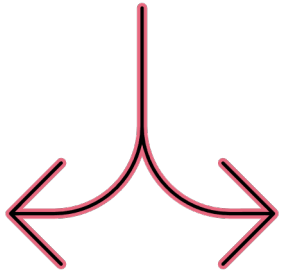
Als Python Programm:

```
a = int(input("Wert a: "))
b = int(input("Wert b: "))
x = a * a
x += 2 * a * b
x += b * b
print x
```

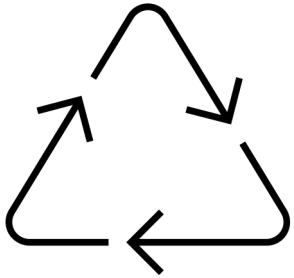
Welchen Wert berechnet dieses Programm?

Kontrollstrukturen

Anweisungen



Verzweigungen



Schleifen

Bedingte Anweisungen

Anweisungen werden nur unter bestimmten Bedingungen ausgeführt. Dazu unterscheiden wir:

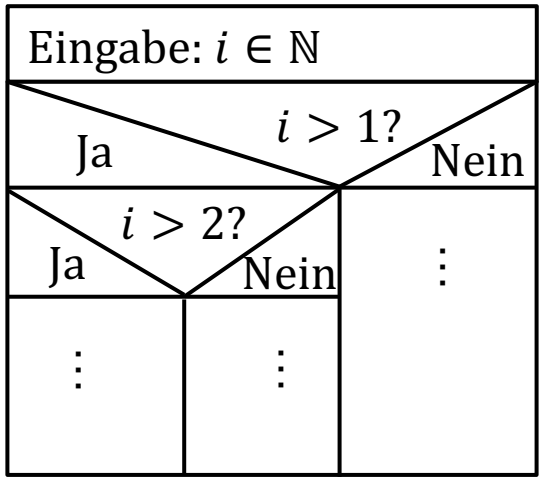
If-Verzweigung:
Verzweigung über einen Booleschen Ausdruck. Ist dieser wahr *(if)*, werden weitere Anweisungen ausgeführt. Scheitert dieser Test *(else)*, können andere Anweisungen durchgeführt werden.

Switch-Case:
Für eine Variable werden Anweisungen durchgeführt, welche zu einem bestimmten Wert passen. Ist ein Wert nicht vorgesehen, wird ein Default-Fall ausgelöst.

Sowohl If-Verzweigungen, als auch Switch-Case-Verzweigungen können verschachtelt werden.

Verzweigungen – Beispiel 1

Bedingte Verzweigung



Als Python Programm:

```
i = int(input("Wert i: "))
if i > 1:
    if i > 2:
        ...
    else:
        ...
else:
    ...
```

Alles, was durch den if- bzw. else-Block ausgeführt werden soll, wird eingerückt.

Python - Einrückung

Der Ablauf eines Pythonprogramms wird über Einrückung festgelegt.

Eine **Einrückung** wird durch einen **Anweisungskopf** (def, if, else, for, while, …) und durch einen **Doppelpunkt** (:) eingeleitet.

Alle Anweisung, die dazugehören, werden um 4 Leerzeichen oder 1 Tab eingerückt.
Die Art der Einrückung muss konsistent sein!

```
i = int(input("Wert i: "))
if i > 1:
    if i > 2:
        ...
    else:
        ...
else:
        ...
```

Wir legen uns auf 4 Leerzeichen fest.
Einige Texteditoren ersetzen automatisch Tab mit 4 Leerzeichen.

Scope von Variablen

Was wird im rechten Programm ausgegeben?

Für $i = 1$ ist print(x) nicht definiert!

Scope:
Eine Variable ist nur für Bereiche definiert, welche im aktuellen Block (inklusive weiterer Einrückungen) liegen.

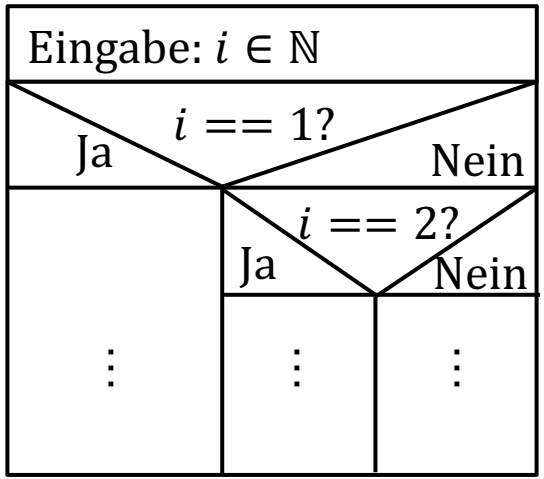
Sobald man aus einem Block herausgeht, in welchem eine Variable deklariert wurde, kann sie nicht weiter genutzt werden!

```
i = int(input("Wert i: "))
if i > 1:
    if i > 2:
        x = 2*i
    else:
        x = i
else:
    print("Nichts zu tun")
print(x)
```

```
Wert i:
1
Nichts zu tun
Traceback (most recent call last):
  File "main.py", line 12, in <module>
    print(x)
NameError: name 'x' is not defined
```

Verzweigungen – Beispiel 2

Bedingte Verzweigung

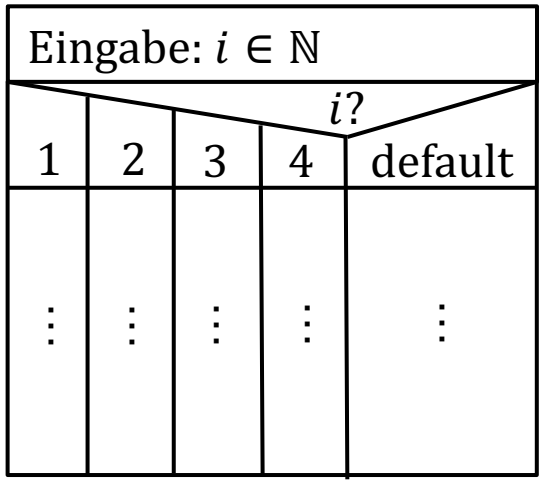


Als Python Programm:

```
i = int(input("Wert i: "))
if i == 1:
    ...
elif i == 2:
    ...
else:
    ...
```

Verzweigungen – Beispiel 3

Switch-Case-Verzweigung

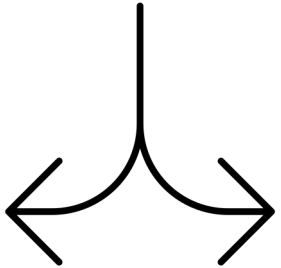


Als Python Programm:

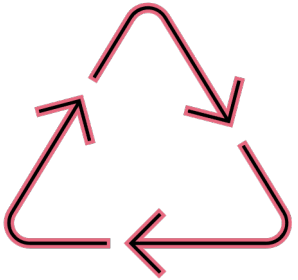
```
i = int(input("Wert i: "))
match i:
    case 1:
        ...
    case 2:
        ...
    case 3:
        ...
    case 4:
        ...
    case _:
        ...
```

Kontrollstrukturen

Anweisungen



Verzweigungen



Schleifen

Schleifen

Schleifen:
Eine Schleife wiederholt nach definiertem Schema oft einen Block von Kontrollstrukturen.

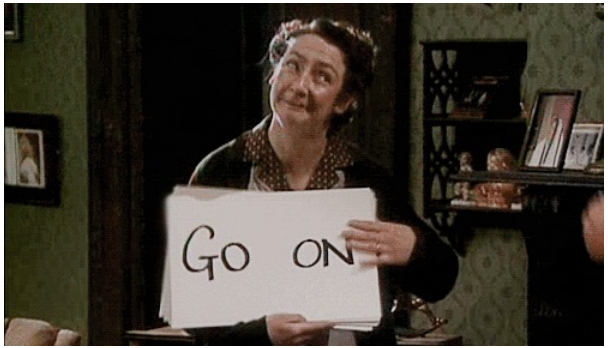
Typische Schleifen sind

For-Schleife:
Wiederhole Kontrollsturkturen *für jeden definierten Wert*.

While-Schleife
Wiederhole Kontrollsturkturen *solange eine Bedingung gilt*.

Repeat-Schleife
Wiederhole Kontrollsturkturen *bis eine Bedingung gilt*.

while True:



For-Schleife

Wir unterscheiden zwischen For- und ForEach-Schleifen.

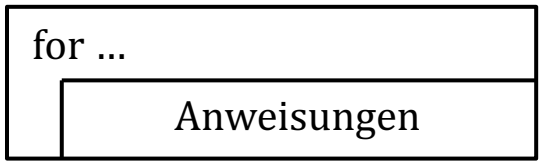
For-Schleife:
for (Init; Test; Inkrement)

Init: Beschreibt, was für die erste Iteration gilt.
Test: Ein Boolescher Ausdruck, ob die nächste Iteration durchgeführt wird.
Inkrement: In der Regel ein Ausdruck, um wie viel eine Variable erhöht werden soll

ForEach-Schleife:
for var in Bereich

var: Eine Variable, die jeden Wert aus Bereich annimmt.
Bereich: Eine Menge an Werten, bspw. als Liste, *Array,* oder sogar Strings

Als Struktogramm



In Python wird die ForEach-Schleife genutzt.

```
for var in [1, 2, 3, 4, 5]:
    print(var)
```

Gibt die Zahlen 1 bis 5 aus.

Arrays

Ein Array ist eine sehr einfache Datenstruktur. Es besteht aus mehreren Feldern, welche Werte enthalten, und kann einer Variable zugewiesen werden.

6	9	1	8	5	2	7	0	3

In Python:
A = [6, 9, 1, 8, 5, 2, 7, 0, 3]

Auf einzelne Felder kann in der Regel mit A[i] zugegriffen werden.
Achtung: Die Nummerierung startet mit 0. Das erste Elemente steht also in A[0].

Python bietet gute Operationen für Arrays

len(A)	Gibt die Anzahl an Elementen
A[-1]	Das letzte Element
A[i:j]	Gibt das Teilarray von i bis j-1 zurück

Schleifen – Beispiel 1

For-Schleife

Eingabe: $n \in \mathbb{N}$
$x = n$
for $(i = 1; i < n; i{+}{+})$
$\quad x = x + i$
Gib x aus

Als Python Programm:

```
n = int(input("Wert n: "))
x = n
for i in range(1, n):
    x += i
return x
```

range(m, n) gibt ein Array mit den Werten m,…, n-1 zurück.

Wird m nicht angegeben, gilt m = 0.

Schleifen – Beispiel 2

For-Schleife

Eingabe: $n \in \mathbb{N}$
$x = n$
for $i \in \{1, \dots, n-1\}$
$\quad x = x + i$
Gib x aus

Als Python Programm:

```
n = int(input("Wert n: "))
x = n
for i in range(1, n):
    x += i
return x
```

range(m, n) gibt ein Array mit den Werten m,…, n-1 zurück.

Wird m nicht angegeben, gilt m = 0.

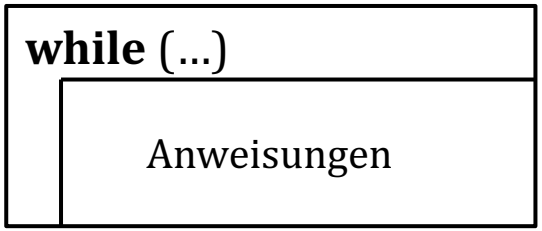
While und Repeat-Schleifen

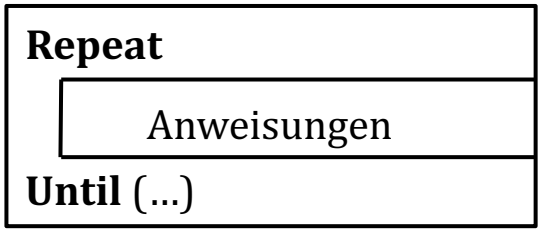
While-Schleife
while (Bedingung)

Bedingung: Ein Test, der ***vor*** jeder Iteration überprüft wird. Nur, wenn der Test ***wahr*** zurückgibt, wird die nächste Iteration durchgeführt.

Repeat-Schleife
repeat … until (Bedingung)

Bedingung: Ein Test, der ***nach*** jeder Iteration überprüft wird. Nur, wenn der Test ***false*** zurückgibt, wird die nächste Iteration durchgeführt.





While – Beispiel

While-Schleife

Eingabe: $n \in \mathbb{N}$
$x = n$
$f = 1$
while $x > 0$
$\quad f = f \cdot x$
$\quad x = x - 1$
Gib f aus

Als Python Programm:

```
n = int(input("Wert n: "))
x = n
f = 1
while x > 0:
    f *= x
    x -= 1
return f
```

Repeat – Beispiel

Repeat-Schleife

Eingabe: $n \in \mathbb{N}$
$x = n$
$f = 0$
repeat
$f = f + x$
$x = x - 1$
until $x = 0$
Gib f aus

Als Python Programm:

```
n = int(input("Wert n: "))
x = n
f = 1
while 1:
    f += x
    x -= 1
    if x == 0:
        break
return f
```

Break und Continue

Break und Continue sind besondere Befehle, um Schleifen zu steuern.

Break:
Bricht die aktuelle Schleife sofort ab.

```
x = 0
A = [1, 5, 3, 7, 9, 2, 11]
for i in A:
   if i % 2 == 0:
      break
   x += 1
print x
```

Position der ersten geraden Zahl in A.

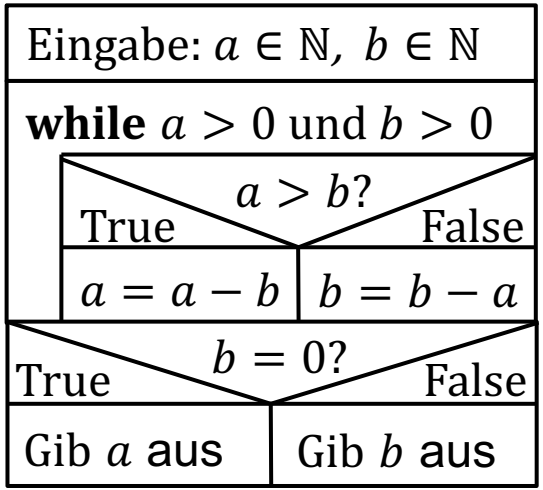
Continue:
Bricht die aktuelle Iteration ab.

```
x = 0
A = [1, 5, 3, 7, 9, 2, 11]
for i in A:
   if i % 2 == 0:
      continue
   x += i
print x
```

Summe der ungeraden Zahlen in A.

Gesamtbeispiel

Als Struktogramm:



Als Python Programm:

```
a = int(input("Wert a: "))
b = int(input("Wert b: "))
while a > 0 and b > 0:
    if a > b:
        a -= b
    else:
        b -= a
if b == 0:
    return a
else:
    return b
```

Kapitel 2.3.2 – Methoden

Methoden

Um Code zu strukturieren kann man Teile, die im Code immer wieder das gleiche berechnen, in eine Subroutine (Funktion, Methode) auslagern.

Eine **Methode** wird allgemein definiert durch *Rückgabewert Name(Inputparameter).* Innerhalb einer Methode, kann ein Wert mit **return** zurückgegeben werden. Ein return beendet die Methode.

Ein Wert muss nicht immer zurückgegeben werden. Auch über *Seiteneffekte* können Werte verändert werden.

Da Python automatisiert Typisiert, brauchen nur Namen und Parameter angegeben werden.

Beispiel in Python

```
def abs(n):
    if n < 0:
        return –n
    return n
```

Ein paar Standard-Methoden

str(zahl):
- Gibt die Zahl zahl als String zurück.
- Beispiel: str(4.01) liefert ‘4.01‘
- Wird benötigt, um Zahlen zu Strings hinzuzufügen.

print(text):
- Gibt den String text auf der Konsole aus.
- Beispiel: print(“Ich bin schon “ + str(32) + “ Jahre alt.“)
- Beispiel: print(42)

len(array):
- Gibt die Anzahl an Elementen in array zurück.
- Beispiel:
  ```
  array = [9, 5, 6, 2, 8, 1]
  len(array) #liefert Wert 6
  ```

Seiteneffekte Beispiel 1

```
def arraychange(A):
    A[0] = 42

def numberchange(a):
    a = 42


A = [1, 2, 3]
a = 1
arraychange(A)
numberchange(a)
print(A)
print(a)
```

Was gibt das Programm heraus?

1.) A = [1, 2, 3] und a = 1

2.) A = [1, 2, 3] und a = 42

3.) A = [42, 2, 3] und a = 1

4.) A = [42, 2, 3] und a = 42

Seiteneffekte Beispiel 2

```
def arraychange(A):
    A = [9, 9, 9]

A = [1, 2, 3]
arraychange(A)
print(A)
```

Was gibt das Programm heraus?

1.) A = [1, 2, 3]
2.) A = [9, 9, 9]

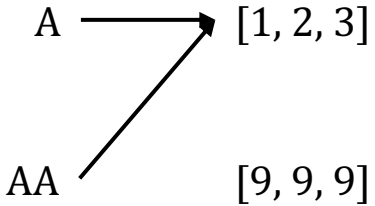
Warum verhält sich das anders als

```
def arraychange(A):
    A[0] = 42

A = [1, 2, 3]
arraychange(A)
print(A)
```

Pass-By-Value

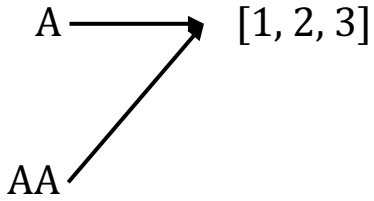
Beim Funktionsaufruf wird der ***Wert*** der Variablen **kopiert** und übergeben.
Für Python und Java gilt:
Sowohl die Variable, die als Adresse im Speicher übergeben wird, als auch die Variable in der Methode **zeigen** auf das gleiche ***Objekt***. Beide Variablen sind aber unterschiedlich!

Ändert man Teile des Objektes, wirken sich die Änderungen auf das ursprüngliche Objekt aus.
Weist man dem Methodenparameter eine neue ***Instanz*** des Objektes zu, bleibt die ursprüngliche Variable davon unberührt!




```
def arraychange(AA):
    AA = [9, 9, 9]

A = [1, 2, 3]
arraychange(A)
print(A)
```

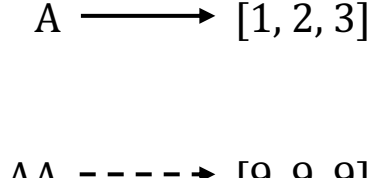
```
def arraychange(AA):
    AA[0] = 42

A = [1, 2, 3]
arraychange(A)
print(A)
```



Pass-By-Value

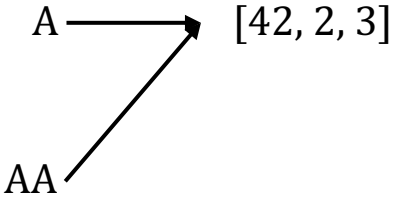
Beim Funktionsaufruf wird der ***Wert*** der Variablen **kopiert** und übergeben.
Für Python und Java gilt:
Sowohl die Variable, die als Adresse im Speicher übergeben wird, als auch die Variable in der Methode **zeigen** auf das gleiche ***Objekt***. Beide Variablen sind aber unterschiedlich!

Ändert man Teile des Objektes, wirken sich die Änderungen auf das ursprüngliche Objekt aus.
Weist man dem Methodenparameter eine neue ***Instanz*** des Objektes zu, bleibt die ursprüngliche Variable davon unberührt!



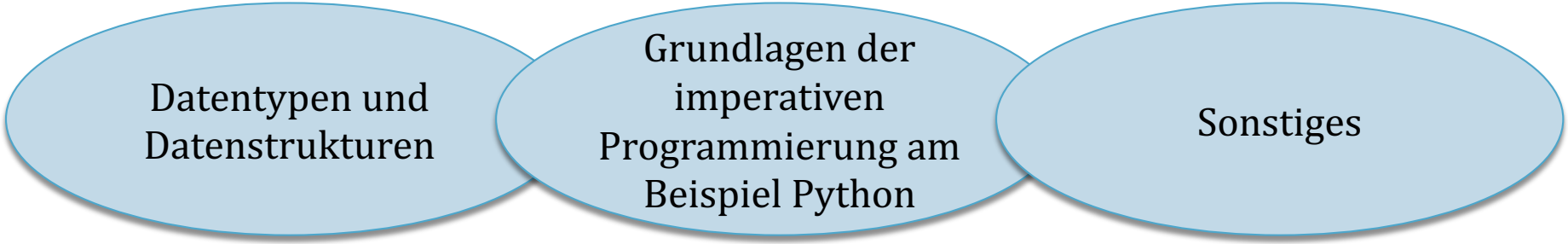

```
def arraychange(AA):
    AA = [9, 9, 9]

A = [1, 2, 3]
arraychange(A)
print(A)
```

```
def arraychange(AA):
    AA[0] = 42

A = [1, 2, 3]
arraychange(A)
print(A)
```



Rückblick

Was haben wir bisher gelernt?

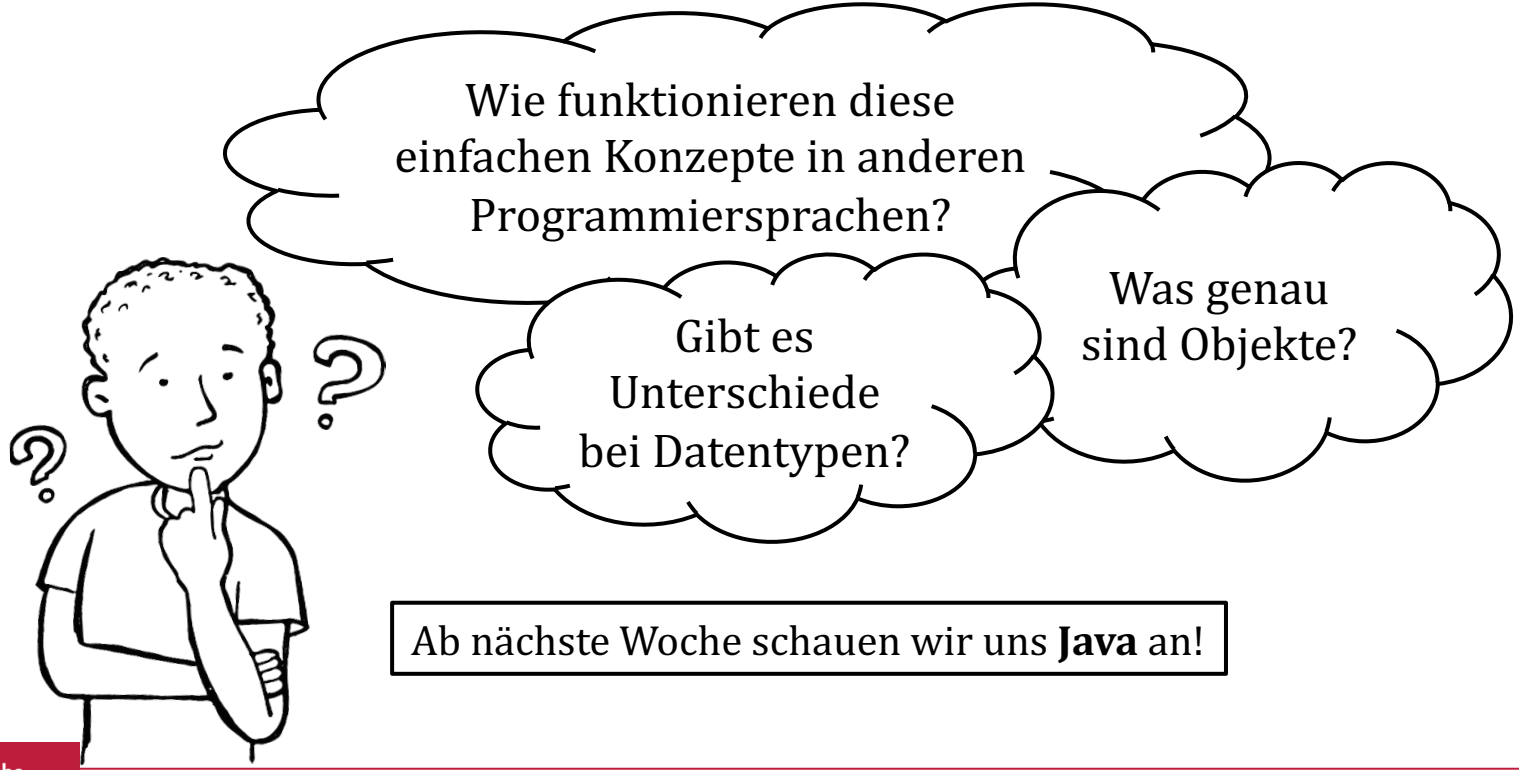


Integer
Float
Boolean
Strings
Arrays

Kontrollstrukturen
- Anweisung
- Verzweigung
- Schleifen
- Methoden

Unäre / binäre Operatoren
Binärzahlen
Scope
Pass-by-Value

Ausblick



Ab nächste Woche schauen wir uns **Java** an!

Nächste Woche

Einführung in Java

- Lexik
- Syntax
- Datentypen