eils labs

Programming 101

Carl Herrmann
IPMB & DKFZ

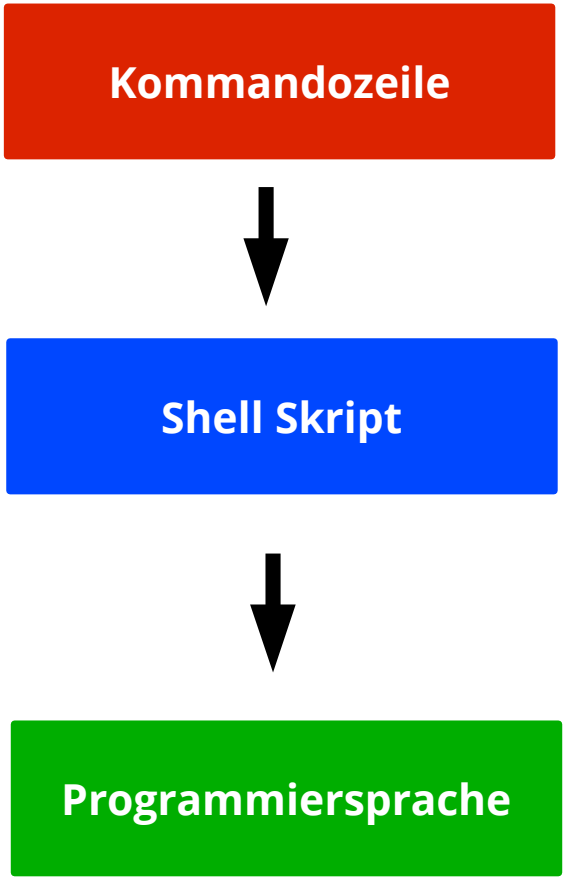
BioQuant
MODEL base of LIFE

IPMB

UNIVERSITÄT
HEIDELBERG
ZUKUNFT
SEIT 1386

dkfz. GERMAN
CANCER RESEARCH CENTER
IN THE HELMHOLTZ ASSOCIATION

Programmieren



Einfache Befehle

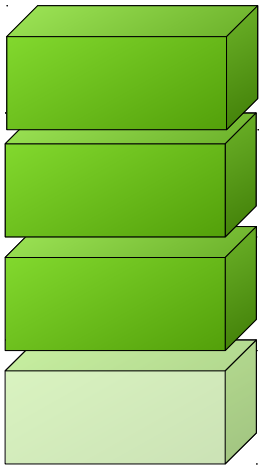
aneinanderketten von Befehlen
Schleifen

Komplexere Aufgaben

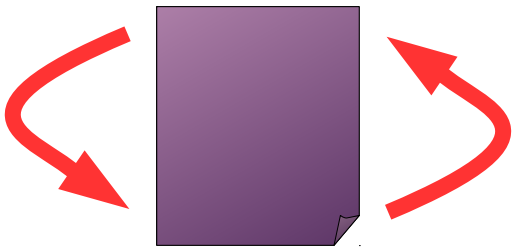
Gemeinsamkeiten

- Alle Programmiersprachen benutzen zum grossen Teil die gleichen **Syntaxelemente**

Variablen

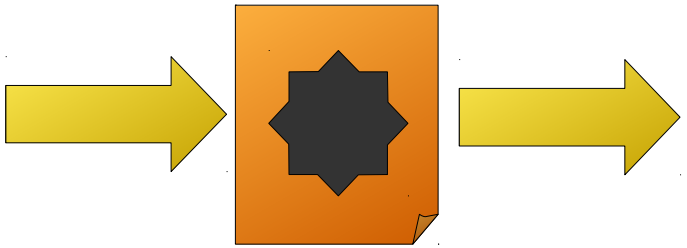


Schleifen

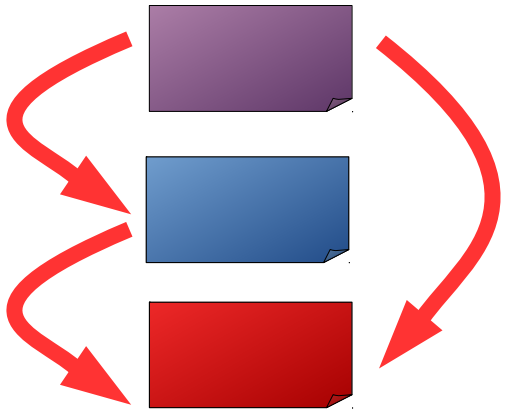


"wiederhole..."

Funktionen



Kontrolstrukturen



"wenn... dann... sonst..."

Struktur eines Programms

Befehle die nacheinander ausgeführt werden

```
print "Mein erstes Program"

von = 1
bis = 20

if bis < von :

    print "Das Ende ist kleiner als der Anfang !"

else:

    for i in range(von,bis) :

        print(i)

    print "Fertig mit Zaehlen !!"
```

Struktur eines Programms

eine Zeichenkette anzeigen

einer Variablen einen Wert zuweisen

```
print "Mein erstes Program"

von = 1
bis = 20

if bis < von :

    print "Das Ende ist kleiner als der Anfang !"

else:

    for i in range(von,bis) :

        print(i)

    print "Fertig mit Zaehlen !!"
```

Struktur eines Programms


```
print "Mein erstes Program"

von = 1
bis = 20

if bis < von :
    print "Das Ende ist kleiner als der Anfang !"
else:
    for i in range(von,bis) :

        print(i)

    print "Fertig mit Zaehlen !!"
```

wenn ja

wenn nein

Struktur eines Programms


```
print "Mein erstes Program"

von = 1
bis = 20

if bis < von :

    print "Das Ende ist kleiner als der Anfang !"

else:

    for i in range(von,bis) :

        print(i)

    print "Fertig mit Zaehlen !!"
```

Struktur eines Programms

Befehle:

```
print "Mein Program"
```

Operationen mit Variablen

```
i=i+1
```

Kontrollbefehle

```
if bis < von : ... else : ...
```

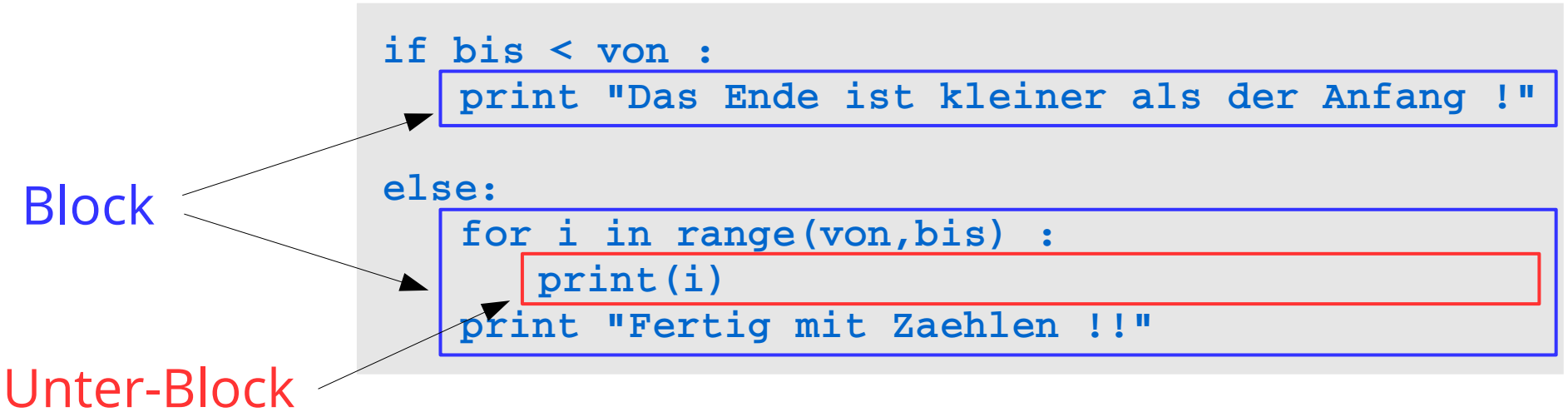
Schleifen

```
for I in range(von,bis) : ...
```

Befehlsblöcke

- Befehle, die zusammen ausgeführt werden

  bilden einen **Befehlsblock**
- In manchen Sprachen werden diese Blöcke durch {...} begrenzt
  → Perl
- in Python werden die Blöcke durch **Einrücken** markiert.



Ein Programm ausführen

- Entweder interaktiv in iPython ausführen
- oder mit dem Befehl

```
python mein_skript.py
```

Variablen

- Variablen sind **Speicherplatz** im Hauptspeicher des Computers, die verschiedene Werte enthalten können (Zahlen, Zeichenketten, logische Werte,...)

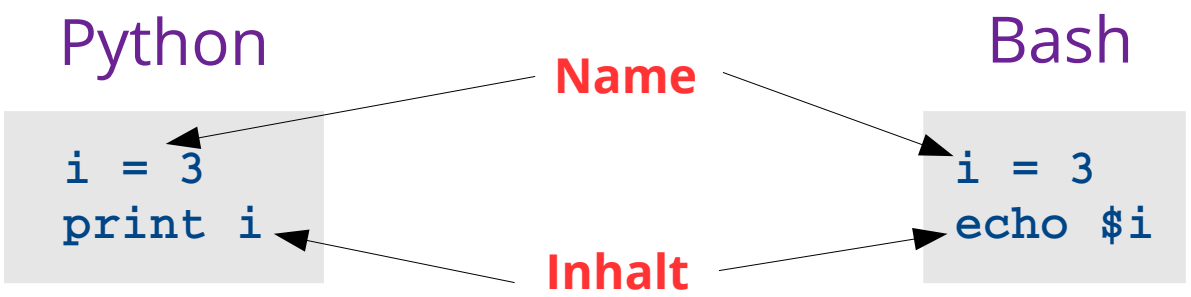
- Eine Variable hat
  - einen Namen
  - einen Typ (numerisch, Zeichenkette, logisch,...)
  - einen Inhalt

Typ = boolean

Typ = string

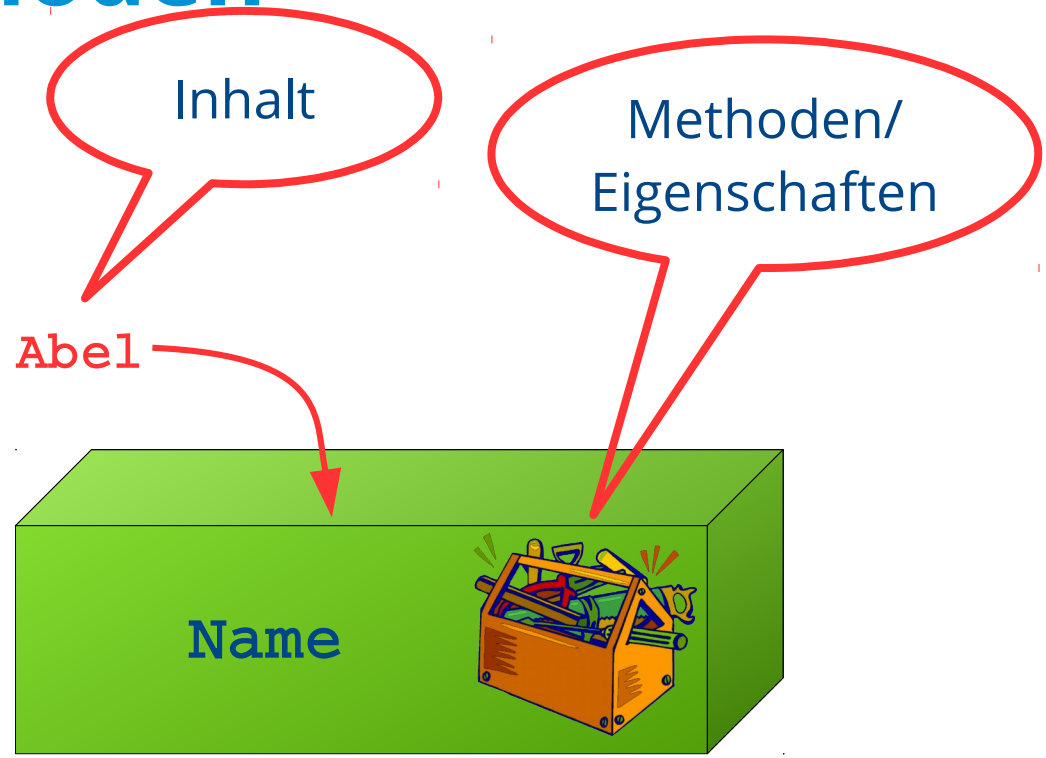
Variablen

- In Python unterscheidet man nicht zwischen **Namen** der Variablen und **Inhalt** ; in Bash wird unterschieden



Variablen und Methoden

- In Python sind Variablen **Objekte**
  - Inhalt
  - Eigenschaften
  - Methoden
- Jeder **Typ** von Variablen hat eigene Methoden




```
dna = "ACCGATT"

print(dna)          # ACCGATT

dna.islower()       # false

dna.lower()         # accgatt
```

Inhalt

Eigenschaft

Methode

Einfache Variablen

eine Variable kann numerische Werte oder Zeichenketten enthalten

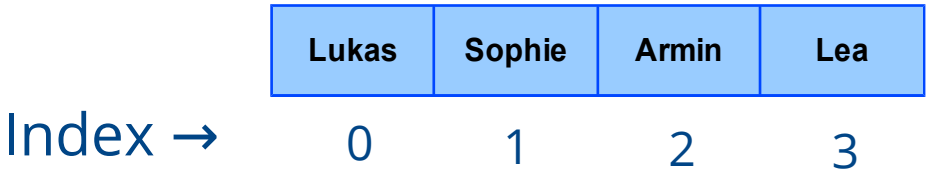
```
i = 3                       # Typ : integer
i = 3.2                     # Typ : float
i = 'Hello World !'         # Typ : string (einfache oder
                            # doppelte Ausführungszeichen
```

das "=" Zeichen dient dazu, einer Variablen einen Wert zuzuordnen

```
i = 2
i = j      # Variable i nimmt den Wert der Variablen j an
```

Tabellen ("arrays", "lists")

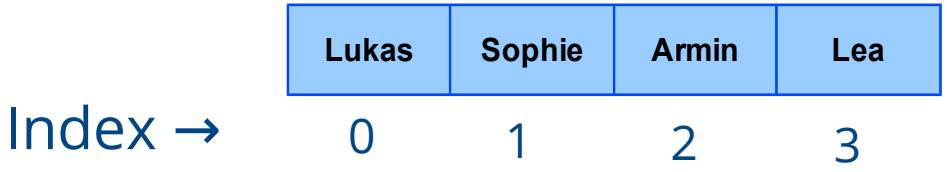
- Tabellen sind Variablen, die mehrere Werte enthalten können
  - Perl : **array** → `@studenten = ('Lukas','Sophie','Armin','Lea')`
  - Python : **list** → `studenten = ['Lukas','Sophie','Armin','Lea']`



- Elemente der Tabelle werden durch die Indizes bezeichnet (Achtung : erstes Element hat Index 0!)
  - `studenten[0]` → `'Lukas'`
  - `studenten[1:3]` → `'Sophie', 'Armin'`

Achtung : der "bis" Index ist exclusiv, d.h. das Element mit diesem Index wird nicht angezeigt !

Tabellen ("arrays", "lists")



- Man kann Elemente auch "überspringen" :

```
studenten[0:3:2] → 'Lukas','Armin'
```

von...

bis...

jedes 2. Element...
("Inkrementationswert")

- Ist der Inkrementationswert negativ → wird von hinten gezählt.

```
studenten[::-1] → 'Lea','Armin','Sophie','Lukas'
```

Tabellen ("arrays", "lists")

- einige Methoden ...
  - `extend` : addiert neue Elemente ans Ende einer Liste

```
In [25]: studenten
Out[25]: ['Lukas', 'Sophie', 'Armin', 'Lea']

In [26]: neue_studenten = ['Rosa','Nick']

In [27]: studenten.extend(neue_studenten)

In [28]: studenten
Out[28]: ['Lukas', 'Sophie', 'Armin', 'Lea', 'Rosa', 'Nick']
```

  - `count(pattern)` : zählt wie oft pattern in der Liste vorkommt

```
In [33]: studenten.count('Armin')
Out[33]: 1

In [34]: studenten.count('toto')
Out[34]: 0
```

Tabellen ("arrays", "lists")

- einige Methoden ...
  - **`index`**: wo in der Liste befindet sich ein Wert ?

```
In [38]: studenten.index('Armin')
Out[38]: 3

In [39]: studenten.index('armin')
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-39-f57e8236a338> in <module>()
----> 1 studenten.index('armin')

ValueError: 'armin' is not in list
```

  - Man kann überprüfen, ob ein Wert in der Liste vorhanden ist...

```
In [40]: 'armin' in studenten
Out[40]: False
```

Alle Methoden : help(list)

besser als Listen :

Dictionnaries ...

```
noten = [1,3,2,1.3,1.7,2.0,3]
```

die Referenz auf Elemente einer Liste erfolgt über den Index
noten[0], noten[3],...

praktischer wäre es, eine Note mit dem Namen des Studenten zu verbinden...
→ ***dictionnaries (hash in anderen Sprachen)***

```
noten = {'Armin':1.0,'Sophie':1.3,'Lukas':2.1,'Annabel':1.7}

noten['Annabel']    # 1.7

noten['Toto']       # Fehler …

noten.get('toto',"Nicht vorhanden")  # "Nicht vorhanden"
```

besser als Listen :

Dictionnaries ...

- Neues Element hinzufügen :

```
noten['Tobias'] = 1.3  # Neues Element

noten['Armin'] = 1.3   # Existierendes Element updaten
```

- Alle keys anzeigen :

```
noten.keys()                    # alle Schlüssel

noten.has_key('Thorsten')       # False
```

Operationen auf Variablen

Man kann Variablen miteinander vergleichen

→ **True** oder **False**

Numerische Variablen :

Gleichheit: ==

Ungleichheit !=

grösser, kleiner : <,>,<=,>=

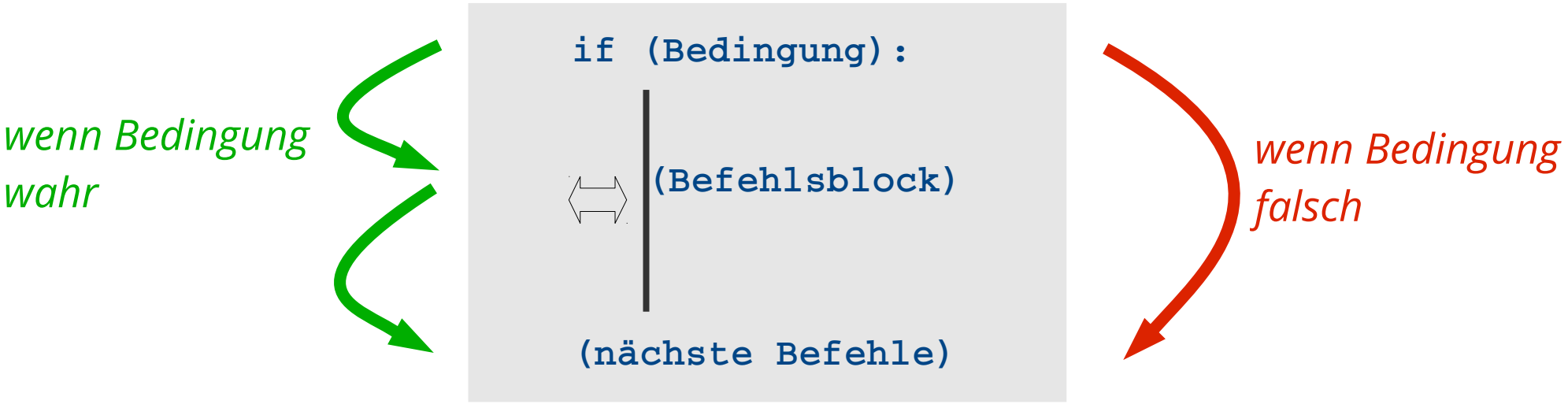
Beispiel:

```
3 != 5          # True
a <= b          # True oder False, je nachdem
```

Achtung : Zuweisung (=) und Vergleich (==) sind verschieden !

Kontrollstrukturen

In manchen Fällen muss man zwischen verschiedenen Möglichkeiten unterscheiden, um eine Aktion durchzuführen



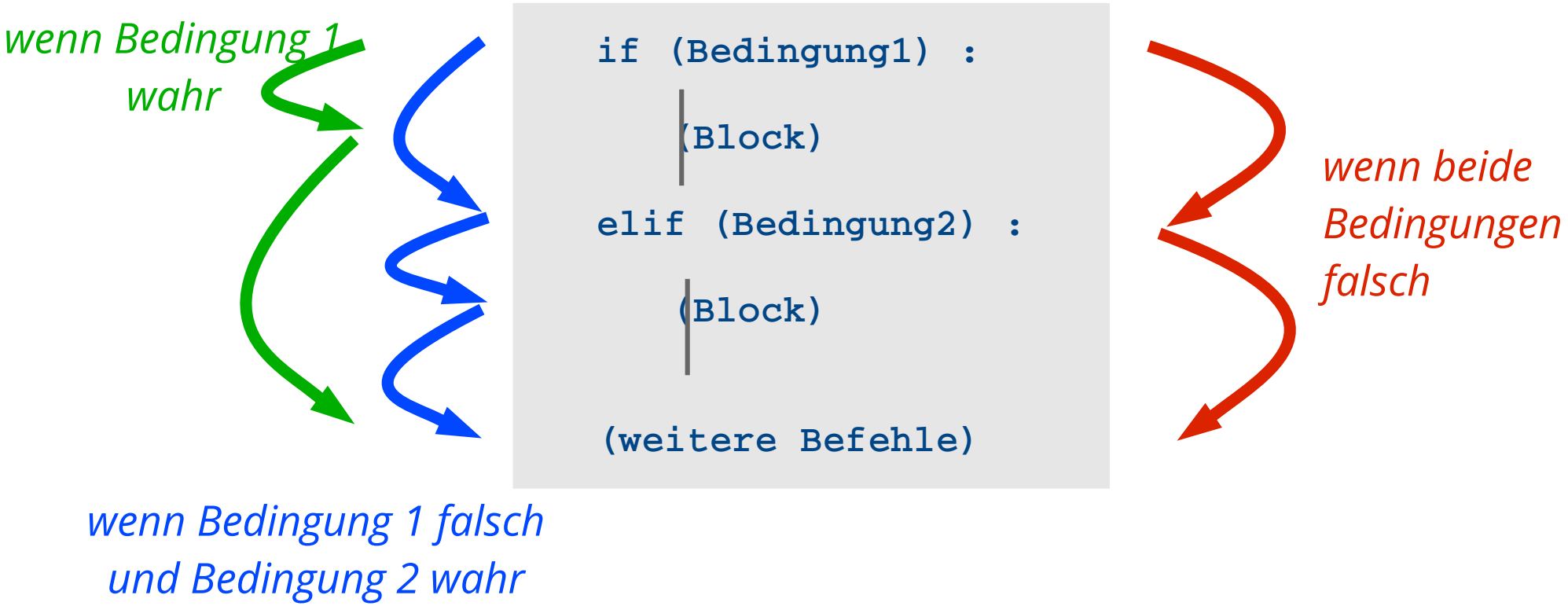
Die Bedingung, die getestet wird ist ein **logischer** Ausdruck :

z.B

- `if (a<b) : …`
- `if (a!= "Tom") : ...`

Kontrollstrukturen

mann kann auch mehrere Bedingungen untersuchen
wenn beide wahr sind, wird nur die erste ausgeführt !



Schleifen

Werde benutzt, um eine Operation mehrere Male durchzuführen

Verschiedene Möglichkeiten :

wiederholen eine bestimmte Anzahl von Malen

```
for i in range(1,11) :
    ...
```

Beispiele:

```
for i in range(1,11) :
    print(i)
```

Dieser Block wird bei jeder Iteration ausgeführt.

```
for i in range(1,11):
    print("Iteration :" + str(i))
    print(" Noch " + str(10-i) + " zu laufen...")
```

Schleifen

Man kann auch über den Inhalt einer Liste iterieren

```
for i in <Liste> :
        ...
```

Beispiele:

```
for student in studenten :
    print(student + " ist in der Gruppe")
```

Bei jeder Iteration wird
student ersetzt durch den nächsten
Wert in der Liste → **Platzhalter**

Schleifen

Alternative : Iterationen laufen lassen, solange eine bestimmmte Bedingung erfüllt ist :

```
while <Bedingung> :
        ...
```

Wenn diese Bedingung immer erfüllt ist läuft die Schleife unendlich weiter ...

Beispiel:

```
count = 0
while (count < 9):
   print 'The count is:', count
   count = count + 1

print "Good bye!"
```

Programme schreiben

- Die ipython Konsole kann man benutzen, um einzelne Befehle auszuprobieren
- Wenn längere Abfolgen von Befehlen ausgeführt werden müssen sie in einer Datei geschrieben werden, die als **Python Skript gespeichert wird** (Achtung ! Endung .py)
- Dieser Skript kann dann **aus der bash Konsole** ausgeführt werden *(Achtung, nicht aus der ipython Konsole!!)*

```
$ python /pfad/zu/meiner/Datei/script.py
```