

Grundlagen der Programmierung

Vorlesung und Übung

02 – Grundlegende Programmstrukturen

Prof. Dr. Andreas Biesdorf

Wirtschaft
Hauptcampus

H O C H
S C H U L E
T R I E R

WIEDERHOLUNG

Variablen

- **Name der Variablen**
 - Deskriptiv
 - Bedeutung tragend
 - Verständlich für Dritte
 - Kein Keyword
- **Wert der Variablen**
 - Zu speichernder Wert
 - Kann sich verändern

Zuweisung

- Berechnungen auf der rechten Seite werden zunächst ausgeführt, um den Wert zu erhalten => **Wert**
- Zuweisung auf die Variable der linken Seite => **Variable**

Beispiel

```
x = 2  
x = x*x  
y = x+1
```



WIEDERHOLUNG

Beispiel: Werte Tauschen

```
x = 2  
y = 3  
y = x  
x = y
```

```
x = 2  
y = 3  
tVar = y  
y = x  
x = tVar
```

Was ist der Wert von x?

Was ist der Wert von x?

- **Integer** - int
- **Float** - float
- **Boolean** - bool



Skalare Typen

- **String** - string

Nicht-Skalarer Typ

Beispiele / Erklärungen

- Buchstaben, Zeichen, Sonderzeichen, Leerzeichen
- **Syntax:** Einfache oder doppelte Hochkommata

```
hstDE = "Hochschule Trier"  
hstEN = 'Trier University of Applied Sciences'
```

- **Konkatenierung von Strings durch +**

```
text = hstDE + " schreibt man auf Englisch : " + hstEN
```

Beispiele / Erklärungen

- **Konkatenierung von Strings**

```
"H" + "ST"
```

- **Indexierung**

```
"HST"[1]  
"HST"[-1]
```

- **Substring**

```
"HST"[1:2]  
"HST":2  
"HST":1:  
"HST"::
```

- **Wiederholungen**

```
3* "Hallo! "
```

- **Ermittlung der Länge eines Strings**

```
len("HST")
```

Achtung: Zählung im String beginnt bei 0 und endet bei len(string)-1!

ÜBUNG 1

- 1) Was ist der Unterschied zwischen Variablen von skalarem Typ und nicht-skalaren Typ?
- 2) Benennen Sie das Ergebnis der folgenden Ausdrücke :
 - "H" + "ST"
 - 3 * "TUAS"
 - "3" * "Hello, world!"
 - "TUAS" [2]
 - "TUAS" [:]
 - "TUAS" [0:]
 - "TUAS" [1:]
 - "TUAS" [:1]

Beispiele / Erklärungen

- **Substring , jedoch nur jedes Xte Element**

```
hstDE = "Hochschule Trier"  
hstEN = "Trier University of Applied Sciences"  
hstDE[1:9:2]  
'ohcu'  
hstDE[::-2]  
'Hcshl re'
```

- **Prüfe auf Substring**

```
>>> "schule" in hstDE  
True  
>>> "schule" not in hstEN  
True
```

ÜBUNG 2

```
hstDE = "HST"  
delimiter = "-"  
hstEN = "TUAS"
```

1) Benennen Sie den Typ und das Ergebnis der folgenden Ausdrücke :

- a) hstDE
- b) hstDE[0]
- c) hstDE[1]
- d) hstDE[-1]
- e) len(hstDE)
- f) hstDE[len(hstDE)]
- g) hstDE + " " + delimiter + " " + hstEN
- h) hstDE * 2
- i) "HST" == hstDE
- j) "HST" == hstEN
- k) "HST" in hstDE
- l) "HST" in hstEN
- m) hstEN[2:3]
- n) hstEN[:3]

Output

- **Ausgabe von Text auf dem Terminal mit print**

```
hstDEGruendung = 1971
print(hstDEGruendung)
hstDEGruendungStr = str(hstDEGruendung)
hstDEGruendungStr
print(hstDEGruendungStr)
print("Gründung der HST:", hstDEGruendungStr, hstDEGruendung)
print("Gründung der HST: " + hstDEGruendungStr + " - " + str(hstDEGruendung))
```

Input

- **Einlesen von Input auf dem Terminal mit `input("")`**
- **Text in Anführungszeichen wird ausgegeben und Eingabe der Variable zugewiesen**

```
text = input("Wie gefällt es Ihnen an der HST? ")  
print("Es gefällt Ihnen: " + 5*text)
```

- **Achtung: Eingabetyp ist immer String**

```
schulNoteHST = int(input("Geben Sie der HST eine Schulnote von 1-6: "))  
print("Ihre Schulnote für die HST ist eine: " + str(schulNoteHST))
```

ÜBUNG 3 – WIEDERHOLUNG VERZWEIGUNGEN

- 1) Benennen Sie das Ergebnis der folgenden Ausdrücke :

```
If 8 > 9:  
    print("HST")
```

```
If 8 > 9:  
    print("HST")  
else:  
    print("TUAS")
```

```
text = "HST"  
if text == "hst":  
    print("Hochschule Trier LowerCase")  
elif text == "HST":  
    print("Hochschule Trier UpperCase")  
else:  
    print("Hochschule Trier in anderem Format")
```

ÜBUNG 3 – WIEDERHOLUNG VERZWEIGUNGEN

- 1) Benennen Sie das Ergebnis der folgenden Ausdrücke (Fortsetzung):

```
temperatur = 20
if temperatur > 15:
    print("Frühling oder Herbst")
elif temperatur > 30:
    print("Sommer")
else:
    print("Winter")
```

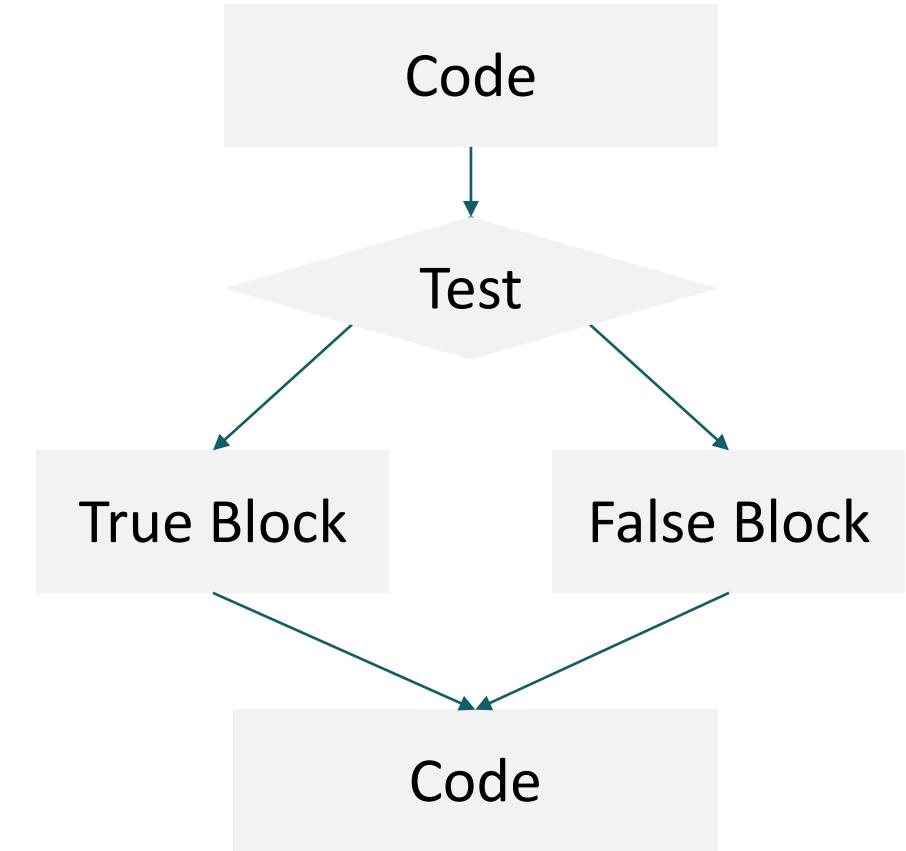
VERZWEIGUNGEN - WIEDERHOLUNG

Einfache Verzweigungen

- Einbindung von **Tests** im **Kontrollfluss**, um zu entscheiden **welcher Teil des Codes als nächstes ausgeführt wird**

Einfaches Beispiel:

```
x = int (input('Bitte Zahl eingeben: '))
if x%2 == 0:
    print('Gerade Zahl')
else:
    print('Ungerade Zahl')
```



VERGLEICHENDE OPERATOREN FÜR INTEGER UND FLOAT - WIEDERHOLUNG

Wirtschaft
Hauptcampus

H O C H
S C H U L E
T R I E R

Operation	Beschreibung	Resultierender Typ
$i > j$	Größer	Boolean: True oder False
$i \geq j$	Größer-gleich	
$i < j$	Kleiner	
$i \leq j$	Kleiner-gleich	
$i == j$	Gleich	
$i != j$	Ungleich	



LOGISCHE OPERATOREN AUF BOOL'SCHE WERTE – WIEDERHOLUNG

Operation	Beschreibung	Resultierender Typ
not a	True, wenn a False	Boolean: True oder False
	False, wenn a True	
a and b	True, wenn a und b True	
a or b	True, wenn a oder b True	



VERZWEIGUNGEN – MÖGLICHE VARIANTEN – WIEDERHOLUNG

if:

```
if <condition>:  
    <expression>  
    <expression>  
    ...
```

if / else:

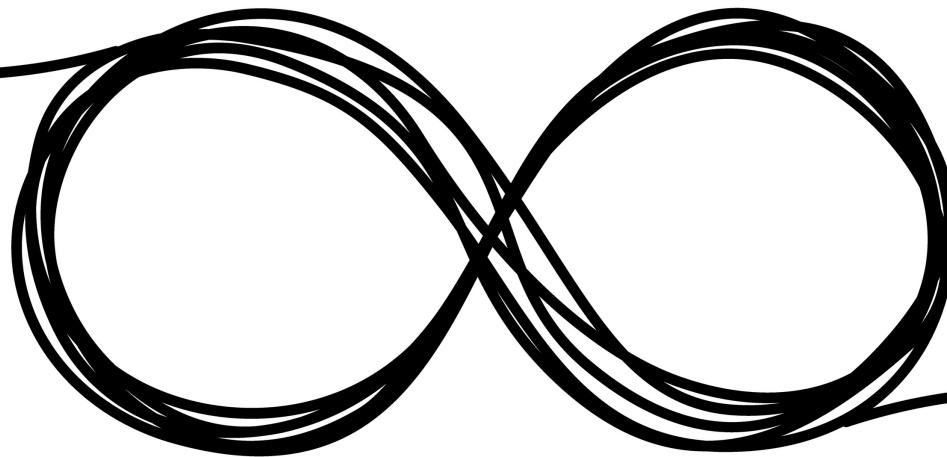
```
if <condition>:  
    <expression>  
    ...  
else:  
    <expression>  
    ...
```

if / elif / else:

```
if <condition>:  
    <expression>  
    <expression>  
    ...  
elif <condition>:  
    <expression>  
    <expression>  
    ...  
else:  
    <expression>  
    <expression>  
    ...
```

LIMITIERUNGEN VON VERZWEIGUNGEN

Einfache
Verzweigungen
können zwar
Entscheidungen
darstellen, aber **der**
Code bleibt
weiterhin linear



Es gibt Probleme,
bei denen der **Code**
beliebig häufig
durchlaufen
werden muss

EINFACHES BEISPIEL – KINDER FRAGEN ELTERN



```
<Antwort> = Frage Mama
if <Antwort> == "nein":
    <Antwort> = Frage Papa
    if <Antwort> == "nein":
        <Antwort> = Frage Mama
        if <Antwort> == "nein":
            <Antwort> = Frage Papa
            ...
else:
    print("gewonnen!")
else:
    print("gewonnen!")
else:
    print("gewonnen!")
```

In Realität läuft der Vorgang
unendlich und der Code
müsste entsprechend lang
werden

EINFACHES BEISPIEL – KINDER FRAGEN ELTERN



```
antwort = input("Frage Mama: ")
while antwort == "nein":
    antwort = input("Frage Papa: ")
    if antwort == "nein":
        antwort = input("Frage Mama: ")
print ("gewonnen!")
```

**Annahme: irgendwann
gibt eines der
Elternteile auf**

while

```
while <condition>:  
    <expression>  
    <expression>  
    ...
```

- Solange <condition> den Wert True hat, durchlaufe die Ausdrücke in der Schleife
- Nach Durchlaufen der Schleife:
 - prüfe erneut, ob <condition> den Wert True hat
 - Wenn <condition> True, durchlaufe erneut
 - Wenn <condition> False, setze den Code nach der Schleife fort
- Achtung: <condition> kann während Durchlaufen der Schleife den Wert ändern, jedoch zählt nur die Prüfung vor erneutem Eintritt in die Schleife!

WHILE-LOOPS UND FOR-LOOPS

while

```
n = 0
while n < 5:
    print("HST - " + str(n))
    n +=1
```

- Recht umständliche Formulierung – eleganter mit einer For-Schleife:

for

```
for n in range(5):
    print("HST - " + str(n))
```

for

```
for <variable> in range(...):  
    <expression>  
    <expression>  
    ...
```

- Für jeden Durchlauf der Schleife, `<variable>` nimmt iterativ die Werte in `range` an

range

```
range(5)  
range(2,5)  
range(3,10,2)  
range(5)
```



Break Statement in einer while- oder for-Schleife

```
while <condition A>:  
    while <condition B>:  
        <expression 1>  
        break  
        <expression 2>  
<expression 3>
```

```
for <variable A> in range(...):  
    for <variable B> in range(...):  
        <expression 1>  
        break  
        <expression 2>  
<expression 3>
```

- Break bricht die (innerste) Schleife ab, unabhängig von der Iteration
- Der übrige Code wird nicht ausgeführt
- Es wird immer nur die aktuelle Schleife abgebrochen, d.h. bei Schachtelung läuft die äußere Schleife weiter

Break Statement in einer while- oder for-Schleife

```
summeSiebenerReiheAbFuenf = 0
for i in range(5, 200, 7):
    summeSiebenerReiheAbFuenf += i
    if summeSiebenerReiheAbFuenf == 5:
        break
print(summeSiebenerReiheAbFuenf)
```

- Was ist der Wert von `summeSiebenerReiheAbFuenf`?

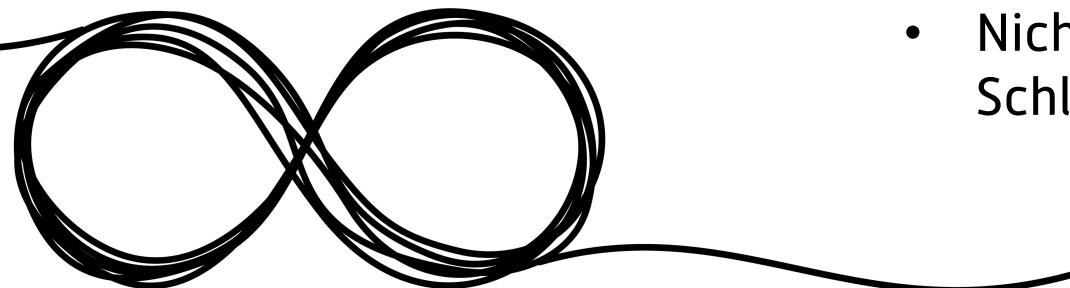
FOR VS. WHILE SCHLEIFEN

for Schleife

- Maximale Anzahl Iterationen bekannt
- Kann jederzeit durch break gestoppt werden
- Nutzt einen internen Zähler
- for-Schleifen können in while-Schleifen umgeschrieben werden

while Schleife

- Maximale Anzahl Iterationen zu Beginn unbekannt
- Kann jederzeit durch break gestoppt werden
- Kann einen Zähler verwenden. Dieser muss jedoch explizit zuvor initialisiert und in der Schleife erhöht werden
- Nicht jede while-Schleife kann in eine for-Schleife umgeschrieben werden



ÜBUNG 4

1) Benennen Sie das Ergebnis der folgenden Ausdrücke:

```
zaehler = 0
while zaehler <= 5:
    print(zaehler)
    zaehler += 1
print("Ausserhalb der Schleife")
print(zaehler)
```

```
anzahlSchleifen = 0
zaehler = 2
while anzahlSchleifen < 10:
    zaehler *= 2
    zaehler += anzahlSchleifen
    anzahlSchleifen -= 1
print("Zaehler: " + str(zaehler))
```

ÜBUNG 4

1) Benennen Sie das Ergebnis der folgenden Ausdrücke (Fortsetzung):

```
zaehler = 10
while zaehler > 3:
    zaehler -= 1
    print(zaehler)
```

```
zaehler = 10
while True:
    if zaehler < 7:
        print('Schleife wird verlassen')
        break
    print(zaehler)
    zaehler -= 1
print('Ausserhalb der Schleife')
```

ÜBUNG 4

- 1) Benennen Sie das Ergebnis der folgenden Ausdrücke (Fortsetzung):

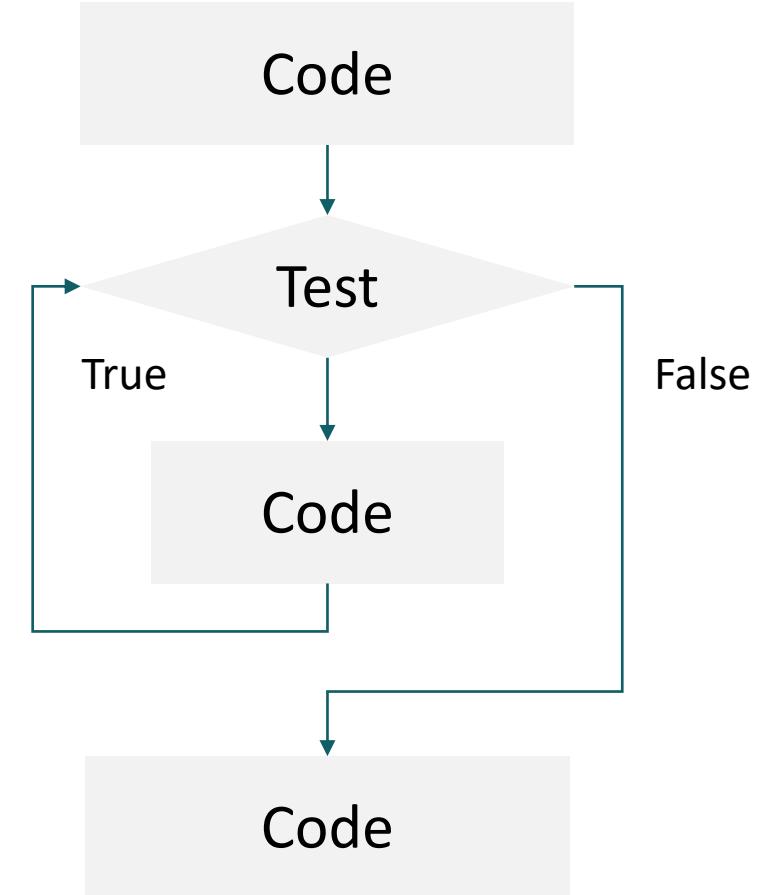
```
zaehler = 100
while not False:
    if zaehler < 0:
        break
    print('zaehler: ' + str(zaehler))
```

- 2) Schreiben Sie zwei kleine Programme, welche folgende Ausgabe zunächst mit einer while- und dann mit einer for-Schleife erzeugt:

```
2
4
6
8
10
Auf Wiedersehen!
```

While-Schleifen für komplexe Programmstrukturen

- Mithilfe von Schleifen kann ein Programm beliebig häufig ausgeführt werden
- Start mit einem **Test**
- Solange der Test `True` ist, führe den Code in der Schleife aus
- Sobald der Test `False` ist, verlasse die Schleife und setze den Code danach fort



BEISPIEL 1/1

Berechnung des Quadrats einer Zahl

```
n = 3
nQuadrat = 0
verbleibendeAdditionen = n
while (verbleibendeAdditionen != 0):
    nQuadrat = nQuadrat + n
    verbleibendeAdditionen = verbleibendeAdditionen - 1
print(str(n) + '*' + str(n) + ' = ' + str(nQuadrat))
```

$$n^2 = \underbrace{n + n + n + \dots + n}_{n \text{ mal}}$$

- n wird durch iterative Addition quadriert

BEISPIEL 2/2

Berechnung des Quadrats einer Zahl

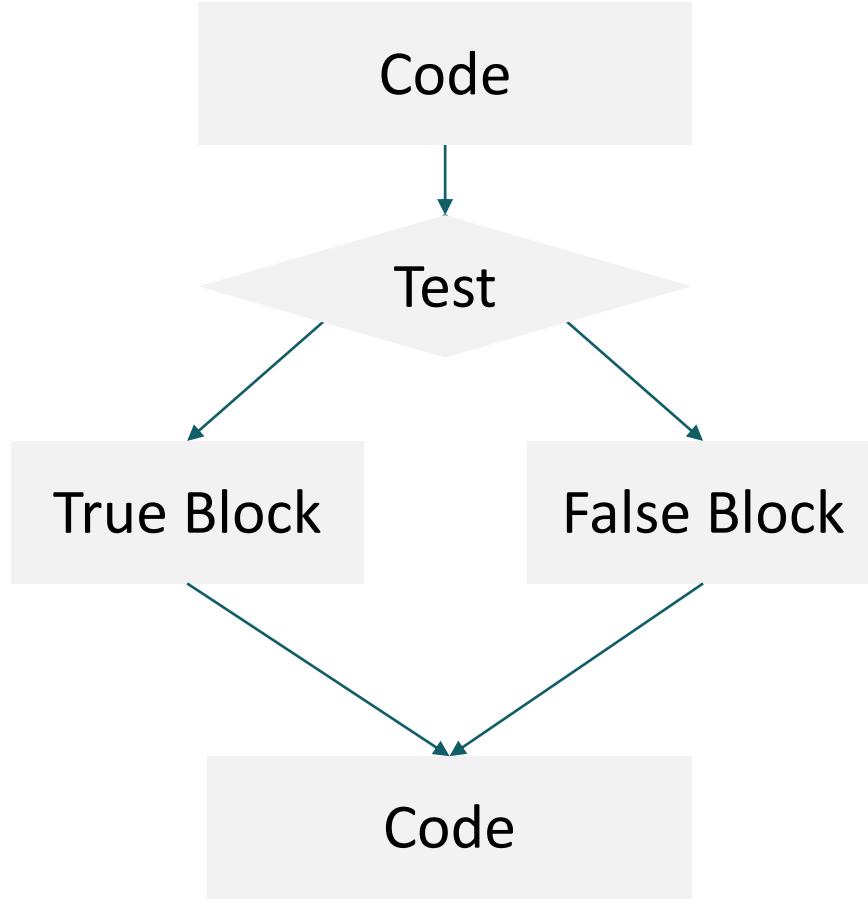
```
n = 3
nQuadrat = 0
verbleibendeAdditionen = n
while (verbleibendeAdditionen != 0):
    nQuadrat = nQuadrat + n
    verbleibendeAdditionen = verbleibendeAdditionen - 1
print(str(n) + '*' + str(n) + ' = ' + str(nQuadrat))
```

n	nQuadrat	verbleibendeAdditionen
3	0	3
	3	2
	6	1
	9	0

- *Variable muss außerhalb der Schleife initialisiert werden*
- *Wert wird innerhalb der Schleife verändert*
- *Jede Iteration muss ein Test durchgeführt werden, ob die Schleife fortgesetzt werden muss*

ITERATIVER CODE VS. VERZWEIGUNGEN

Verzweigungen

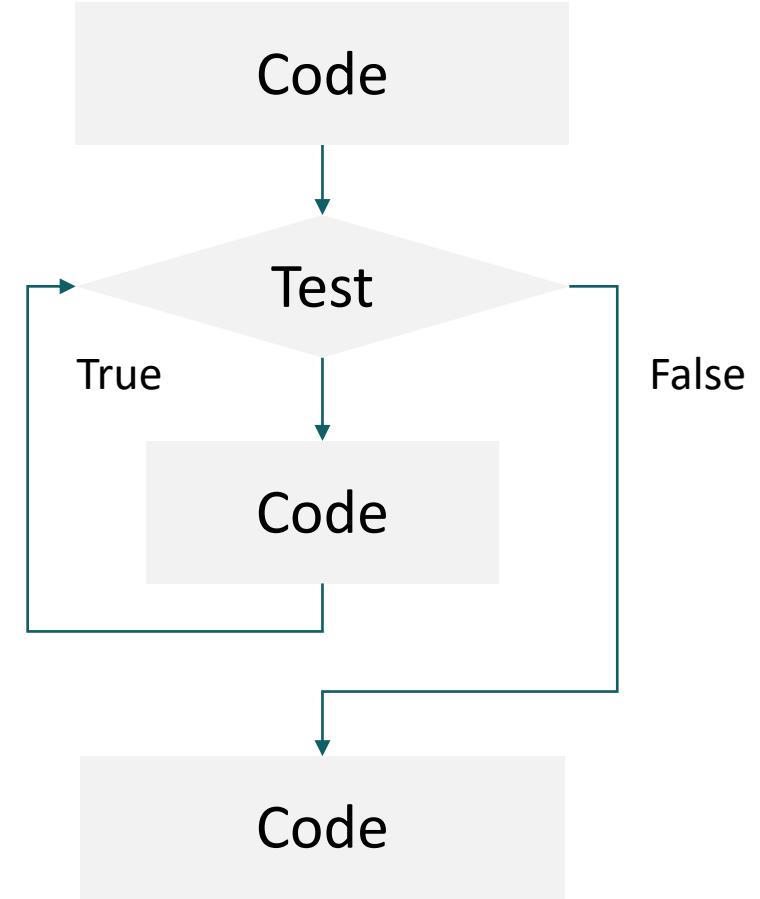


- Verzweigungen ermöglichen das Ausführen **unterschiedlicher Codeteile basierend auf einem Test**

- Iterativer Code ermöglichen das Ausführen von Code **bis eine Bedingung erfüllt ist**

Die Laufzeit des Programms kann je nach Situation sehr unterschiedlich sein!

Iterativer Code



1) Benennen Sie die Ausgabe der folgenden Programme:

```
zaehler = 10
for zaehler in range(5):
    print(zaehler)
print(zaehler)
```

```
divisor = 2
for zaehler in range(0, 10, 2):
    print(zaehler/divisor)
```

```
for variable in range(20):
    if variable % 4 == 0:
        print(variable)
    if variable % 16 == 0:
        print('Foo!')
```

ÜBUNG 5

- 1) Benennen Sie die Ausgabe der folgenden Programme (Fortsetzung):

```
for zeichen in 'hallo':  
    print(zeichen)
```

```
zaehler = 0  
for zeichen in 'Herbst :-)':  
    print('Zeichen #' + str(zaehler) + ' ist ' + str(zeichen))  
    zaehler += 1  
    break  
print(zaehler)
```

ÜBUNG 5

```
meinString = '9.00z'

for char in meinString:
    print(char)

print('fertig')
```

- 1) Wie häufig wird die Zahl 9 ausgegeben?
- 2) Wie häufig wird das Zeichen '.' ausgegeben?
- 3) Wie häufig wird die Zahl 0 ausgegeben?
- 4) Wie häufig wird z ausgegeben?
- 5) Wie häufig wird fertig ausgegeben?

ÜBUNG 5

```
begruessung = 'Hallo!'  
zaehler = 0  
for zeichen in begruessung:  
    zaehler += 1  
    if zaehler % 2 == 0:  
        print(zeichen)  
        print(zeichen)  
print('fertig')
```

- 1) Wie häufig wird H ausgegeben?
- 2) Wie häufig wird a ausgegeben?
- 3) Wie häufig wird l ausgegeben?
- 4) Wie häufig wird o ausgegeben?
- 5) Wie häufig wird ! ausgegeben?
- 6) Wie häufig wird fertig ausgegeben?

ÜBUNG 5

```
einrichtung = 'Hochschule Trier'  
anzVokale = 0  
anzKonsonanten = 0  
  
for char in einrichtung:  
    if char == 'a' or char == 'e' or char == 'i' or char == 'o' or char == 'u':  
        anzVokale += 1  
    elif char == 'o' or char == 'T':  
        print(char)  
    else:  
        anzKonsonanten -= 1  
  
print('Anzahl Vokale: ' + str(anzVokale))  
print('Anzahl Konsonanten: ' + str(anzKonsonanten))
```

- 1) Wie häufig wird o ausgegeben?
- 2) Wie häufig wird T ausgegeben?
- 3) Was ist der Wert von anzVokale und anzKonsonanten?

„Guess-and-Check“ Algorithmen

- Iterative Algorithmen erlauben uns schwierigere Aufgaben zu lösen, als nur mit einfacher Arithmetik bzw. einfachen Verzweigungen
- Auf Basis eines Tests können Schritte mehrfach wiederholt werden
- Beispiel: Klasse der „Guess-and-Check“ Algorithmen, Idee:
 - Start mit einer **groben Abschätzung** des Ergebnis
 - **Prüfung der Abschätzung**
 - falls **nah genug am Ergebnis** → Ende
 - falls **zu hoch / zu niedrig** → Anpassung der Abschätzung

Was ist die (ganzzahlige) Kubikwurzel aus x ?

Guess-and-Check Ansatz:

- Schleife mit allen Werten, die folgende Prüfung durchführt:

$$g: 0 \rightarrow 0^3 > x?$$

$$g: 1 \rightarrow 1^3 > x?$$

$$g: 2 \rightarrow 2^3 > x?$$

...

- Erzeuge beliebig viele Abschätzungen (Guesses), um das Ergebnis zu finden

- Sobald $g^3 > x$ wird die Schleife beendet
- Achtung: limitiert auf positive ganzzahlige Werte, dafür jedoch abzählbar viele Iterationen (aber nicht besonders effizient)

Kubikwurzel – Guess-and-Check Ansatz

```
x = int(input('Bitte einen ganzzahligen Wert eingeben: '))
ans = 0
while ans**3 < x:
    ans = ans + 1
if ans**3 != x:
    print(str(x) + ' hat keine ganzzahlige Kubikwurzel!')
else:
    print('Kubikwurzel von ' + str(x) + ' ist ' + str(ans))
```

- Deklarativer Ansatz, um die Wurzel zu finden
- Leider jedoch limitiert auf positive ganzzahlige Werte
- Einfach zu verstehen

Frage:

Was müsste ich am Code ändern, um auch Kubikwurzeln aus negativen Zahlen zu finden?

Kubikwurzel – Guess-and-Check Ansatz – nun auch für negative Werte

```
x = int(input('Bitte einen ganzzahligen Wert eingeben: '))
ans = 0
while ans**3 < abs(x):
    ans = ans + 1
if ans**3 != abs(x):
    print(str(x) + ' hat keine ganzzahlige Kubikwurzel!')
else:
    if x < 0:
        ans = -ans
    print('Kubikwurzel von ' + str(x) + ' ist ' + str(ans))
```

Schleifenvariable

- Initialisiert außerhalb der Schleife
- Veränderung innerhalb der Schleife
- Test für Terminierung der Schleife hängt an der Schleifenvariable

Allgemeinere Idee: Dekrementierende Funktionen

- Abbildung von Programmvariablen auf einen **ganzzahligen Wert**
- Beim **Start** der Schleife sollte der Wert **positiv** sein
- Sobald der Wert **0** erreicht, wird die **Schleife beendet**
- Der Wert sollte **bei jedem Schleifendurchlauf kleiner werden**
- Mögliche Dekrementierende Funktion in diesem Beispiel:
 $\text{abs}(x) - \text{ans}^{**3}$

Typische Fehler bei Schleifen – Achtung!

- Initialisierung der Schleifenvariable außerhalb der Schleife vergessen
 - NameError, außer die Schleifenvariable wurde schon woanders verwendet
- Keine Veränderung der Schleifenvariable innerhalb der Schleife
 - Endlosschleife

Idee

- Grobe Abschätzung des Ergebnisses
- Prüfung der Korrektheit des Ergebnisses
- Verbesserung der Abschätzung, bis
 - der richtige Wert gefunden, oder
 - alle Werte geprüft wurden

In unserem Beispiel:

„**Lineare Suche**“, „**Brute-Force**“ oder „**Exhaustionsmethode**“,
da potentiell alle Möglichkeiten getestet werden

Lineare Suche

Kubikwurzel – Guess-and-Check Ansatz – etwas eleganter

```
wert = int(input("Bitte einen ganzzahligen Wert eingeben: "))  
for schaetzung in range(wert+1):  
    if schaetzung**3 == wert:  
        print("Kubikwurzel von ", wert, " ist ", schaetzung)
```

Fehler:

- Berechnung klappt nur, wenn eine korrekte Anfrage gestellt wird
- Berechnung klappt wieder nur für positive Kubikwurzeln

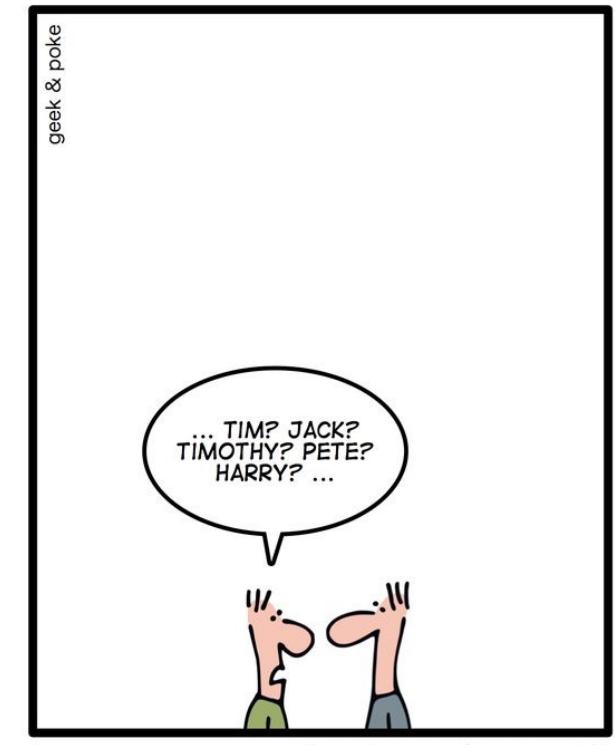
Kubikwurzel – Guess-and-Check Ansatz – etwas eleganter

```
wert = int(input("Bitte einen ganzzahligen Wert eingeben: "))
schaetzung = 0
for schaetzung in range(abs(wert)+1):
    if schaetzung**3 >= abs(wert):
        break
if schaetzung**3 != abs(wert):
    print(str(wert) + ' hat keine ganzzahlige Kubikwurzel! ')
else:
    if wert < 0:
        schaetzung = -schaetzung
print("Kubikwurzel von ", wert, " ist ", schaetzung)
```

- **Einfacher Ansatz** für **viele Probleme**, die mit einem Test einer **finiten Anzahl Möglichkeiten** gelöst werden können
- Ansatz, um Abschätzungen in einem **strukturierten Ansatz** zu erzeugen
- Funktioniert auch für **nicht sortierte Werte!**
- Relativ einfach zu verstehen, zu implementieren und zu testen

Lineare Suche

SIMPLY EXPLAINED:
BRUTE FORCE ATTACK



ÜBUNG 6

1) Welche der alternativen Formulierungen ergeben den gleichen Output?

```
iteration = 0
zaehler = 0
while iteration < 5:
    for zeichen in "hallo, welt":
        zaehler += 1
    print("Iteration " + str(iteration) + "; Zaehler ist: " + str(zaehler))
    iteration += 1
```

1

```
for iteration in range(5):
    zaehler = 0
    while True:
        for zeichen in "hallo, welt":
            zaehler += 1
        print("Iteration " + str(iteration) + "; Zaehler ist: " + str(zaehler))
```

2

```
for iteration in range(5):
    zaehler = 0
    while True:
        for zeichen in "hallo, welt":
            zaehler += 1
        print("Iteration " + str(iteration) + "; Zaehler ist: " + str(zaehler))
        break
```

1) Welche der alternativen Formulierungen ergeben den gleichen Output?

```
iteration = 0
zaehler = 0
while iteration < 5:
    for zeichen in "hallo, welt":
        zaehler += 1
    print("Iteration " + str(iteration) + "; Zaehler ist: " + str(zaehler))
    iteration += 1
```

3

```
zaehler = 0
text = "hallo, welt"
for iteration in range(5):
    index = 0
    while index < len(text):
        zaehler += 1
        index += 1
    print("Iteration " + str(iteration) + "; Zaehler ist: " + str(zaehler))
```

4

```
zaehler = 0
text = "hallo, welt"
for iteration in range(5):
    while True:
        zaehler += len(text)
        break
    print("Iteration " + str(iteration) + "; Zaehler ist: " + str(zaehler))
```

ÜBUNG 6

1) Welche der alternativen Formulierungen ergeben den gleichen Output?

```
iteration = 0
zaehler = 0
while iteration < 5:
    for zeichen in "hallo, welt":
        zaehler += 1
    print("Iteration " + str(iteration) + "; Zaehler ist: " + str(zaehler))
    iteration += 1
```

5

```
zaehler = 0
text = "hallo, welt"
for iteration in range(5):
    zaehler += len(text)
    print("Iteration " + str(iteration) + "; Zaehler ist: " + str(zaehler))
```

ÜBUNG 6

- 2) Schreiben Sie ein Programm, welches bei einem String s die Anzahl Vokale bestimmt.

Beispielausgabe:

```
s = 'hstmeanstuasmeanshst'  
Anzahl Vokale: 6
```

- 3) Schreiben Sie ein Programm, welches ausgibt, wie häufig der String hst in einem beliebigen String s auftritt

Beispielausgabe:

```
s = 'hstmeanstuasmeanshst'  
Anzahl von hst in String: 2
```

ÜBUNG 6

- 4) Schreiben Sie ein Programm, welches den längsten Substring in s findet, bei dem die Buchstaben in alphabetischer Reihenfolge vorkommen.

Beispielausgabe:

$s = 'hstmeanstuasmeanshst'$

Länger Substring in alphabetischer Reihenfolge: anstu