

# Programmiergrundkurs in Python

- Grundlagen
  - Einleitung
  - Variablen und Listen
  - Schleifen
  - Bedingungen
  - Rechnungen
- Programmieren mit Python
  - Rechnen mit Python

# Grundlagen

# Einleitung

Diesen Einführungskurs gibt es auch als Jupyter Notebook, für die interaktive Bearbeitung der Codebeispiele:

- Einführungskurs: [Einführung.ipynb](#)
- Praktische Übungen: [Einführung II.ipynb](#)
- Auf dieser Seite können die Dateien geöffnet und bearbeitet werden: [jupyter.org](https://jupyter.org) (Hinweis: Alle Funktionen im Zusammenhang mit dem `input`-Befehl werden nicht funktionieren.)

**Besser ist die Installation des Jupyter Notebooks in VisualStudioCode oder Thonny. Hier stehen alle Funktionen zur Verfügung.**

## Aufbau eines Pythonprogramms

Ein Pythonprogramm beginnt üblicherweise mit dem Import von benötigten Funktionen. Z.B. wird in dieser Zeile die Bibliothek 'math' eingebunden, mit deren Hilfe man mit Zeit rechnen kann:

```
import math
```

## Definieren von Funktionen

Funktionen in Python sind definierte Codebereiche, die aber erst ausgeführt werden, wenn sie aufgerufen wurden.

```
# Hier stehen die Imports, wenn sie benötigt werden.  
import math # Wird allerdings für die grundlegenden Funktionen nicht benötigt.  
  
# Definition der Addition.  
def add(a,b):  
    return a+b  
  
ergebnis = add(1,2) # Das Ergebnis wird in einer Variablen gespeichert.  
print("1+2 ergeben: " + str(ergebnis)) # Das Ergebnis wird auf der Konsole ausgegeben.
```

# Variablen und Listen

## Variablen

Variablen sind Elemente in der Programmierung, die Daten enthalten können. Anders als andere Elemente der Programmierung kann der Inhalt während des Programmablaufs verändert werden.

Beispiele:

```
var x = 0 # Die Variable x wird mit dem Inhalt 0 befüllt.
```

```
x+=1 || x=x+1 # Die Variable x wird um den Wert 1 erhöht.
```

```
name = "Mr. Jacobs" # Die Variable name wird mit dem Inhalt "Mr. Jacobs" befüllt.
```

## Folgende Regeln gelten für die Benennung von Variablen:

- Eine Variable beginnt mit einem Buchstaben oder mit einem Unterstrich. Um Probleme zu vermeiden, sollen nur englische Buchstaben verwendet werden.
- Der Variablenname darf keine Sonderzeichen außer den Unterstrich enthalten.
- Variablennamen unterscheiden Groß-/Kleinschreibung. `fruit` und `Fruit` sind unterschiedliche Variablen.

## Sinnvolle Konventionen für Variablennamen

- Variablen beginnen mit einem Kleinbuchstaben.
- Konstanten (Variablen, die sich im Laufe des Programms nicht ändern) bestehen aus Großbuchstaben.
- Bei einer Verkettung von verschiedenen Wörtern beginnt jedes Wort mit einem Großbuchstaben oder man trennt die Wörter mit einem Unterstrich. Beispiel: `myVeryComplicatedVariable` oder auch: `my_very_complicated_variable`.

# Listen

Ein anderer Typ von Variable ist die Liste. In einer Liste können beliebig viele Elemente des gleichen Typs eingefügt und aus- oder eingelesen werden. Beispiel:

```
fruit = ["Apples", "Bananas", "Raspberries"]
```

```
myGrades = [3, 4, 2, 5]
```

```
for x in fruit:
```

```
    print (x)
```

```
sum =0
```

```
for x in myGrades:
```

```
    sum +=x
```

```
    print (x)
```

```
medianGrade = sum/len(myGrades)
```

```
print("Durchschnitt: ", medianGrade)
```

# Schleifen

## Schleifen

Schleifen wiederholen Code, der in ihnen enthalten ist. Dabei gibt es zwei grundsätzliche Möglichkeiten:

Schleifen prüfen mit jedem Durchlauf, ob eine Bedingung erfüllt/nicht erfüllt ist und brechen gegebenenfalls ab.

## For-Schleife

Bei For-Schleifen ist beim ersten Durchgang schon klar, wie viele Iterationen es geben wird.

```
# Diese Schleife wird 10x ausgeführt. Gleichzeitig steht in jedem Durchlauf der Zähler i zur Verfügung, der  
jeweils um eins erhöht wird.  
for i in range(10):  
    pass  
    print("Die Schleife ist durchgelaufen.")  
  
fruit = ["Apples", "Bananas", "Raspberries"] # [1]  
for x in fruit:  
    print(x)  
  
## Iteration über einen String  
s = "Theodor-Heuss-Schule"  
for i in s:  
    print(i)
```

Eine Schleife kann nicht leer sein, da Python keine Klammern verwendet, um Bereiche zu kennzeichnen. Daher wird der `pass`-Befehl verwendet, der nichts tut und damit eine Zeile markiert.

## While-Schleife

Dieser Code wird 10x ausgeführt. Auch hier steht der Zähler i zur Verfügung.

```
i=0
while i < 10:
    i+=1
    print("Die Schleife ist durchgelaufen.")

# Dieser Code wird unendlich lange ausgeführt, da die Bedingung immer wahr ist.
while True:
    pass
    print("Ich werde nie angezeigt.")
```

[1] [Information zu Listen](#)

# Bedingungen

## Bedingungen

Ein weiteres wichtiges Element aller Programmiersprachen sind Bedingungen. Wenn die gegebene Bedingung erfüllt ist, dann wird der Code in dem Bereich ausgeführt.

Beispiel einfache Passwortabfrage

```
def passwordCorrect(passwd):  
    if passwd == "1234":  
        print("Das Passwort ist korrekt, auch wenn es nicht gut ist.")  
    else:  
        print("Du bekommst keinen Zugang.")
```

## Vergleichsoperatoren

```
==  
!=  
<  
>  
<=  
>=
```



# Rechnungen

## Rechnen mit dem Computer

### Modulo

Modulo ist eine praktische Art, um zu überprüfen, ob bei einer Division ein Rest bleibt.

Beispiel:

```
4%2 //Der Rest dieser Division ist 0.
```

# Programmieren mit Python

# Rechnen mit Python

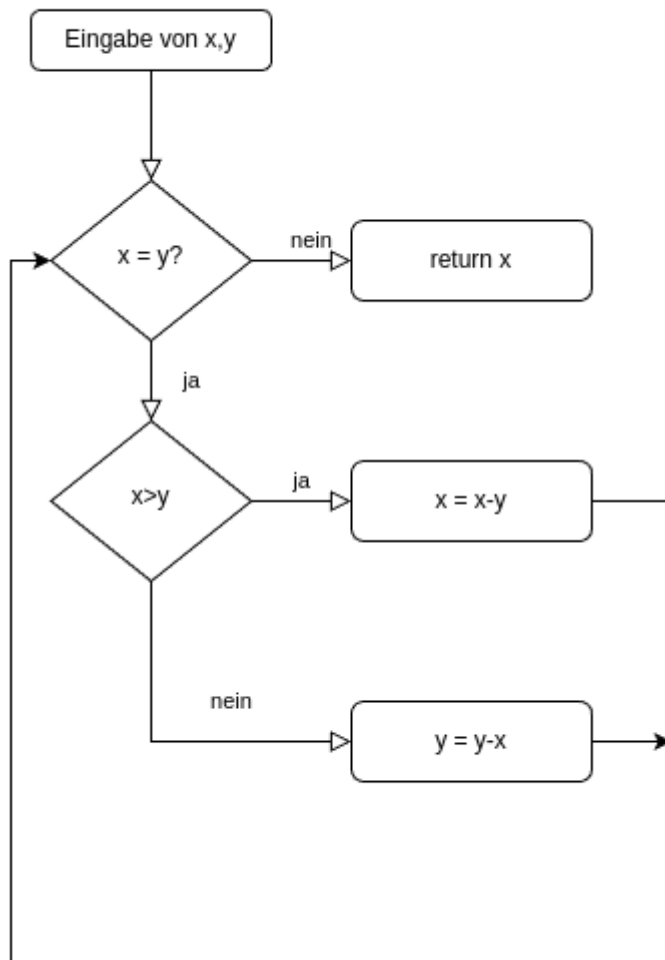
## Python Compiler

- [Online-Compiler](#)
- [Programmierungsumgebung Thonny](#)
- [Thonny ohne Installation](#)

## Kürzen eines Bruchs

Programmiere mit Python ein Programm, das einen Bruch (rationale Zahl) kürzt. Dazu benötigt man den ggT: Hier ist der Algorithmus für den ggT:

### Algorithmus für den ggT



Dann muss der Zähler und Nenner nur noch durch den ggT geteilt werden. Das benötigt ihr dazu: Die Informationen gibt es im Unterricht. Hier ist das Grundgerüst für dein Programm:

```
def ggt(x,y):  
    ""  
    Die Funktion berechnet den größten gemeinsamen Teiler aus den beiden  
    Parametern x und y.  
    ""  
    while x!=y:  
        #Dein Code  
    return x  
  
def kuerzen(x,y):  
    ""  
    Diese Funktion kürzt den Bruch x/y:  
    ""  
    g=ggt(x,y)
```

```
#Dein Code
```

```
ergebnis= str(int(x))+ '/' + str(int(y))
```

```
return ergebnis
```

```
print ("Der Bruch lautet gekürzt: ", kuerzen(65,135))
```

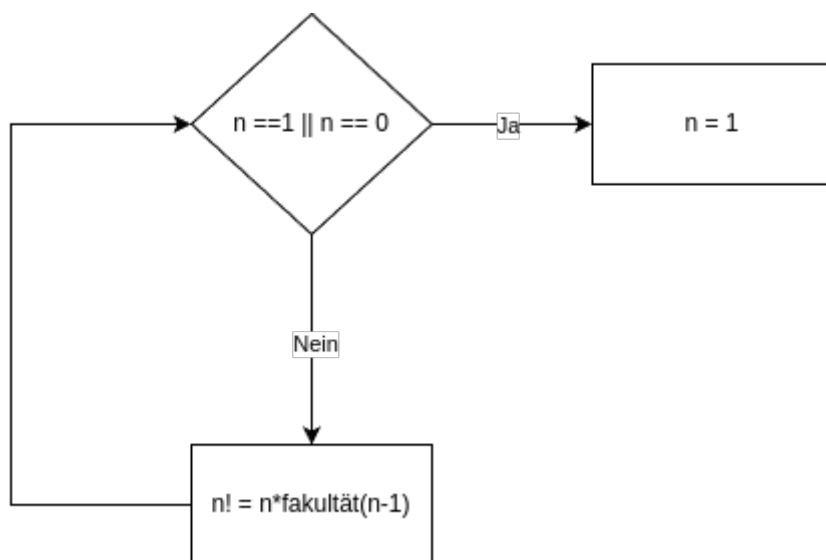
# Fakultät

Als nächstes berechnen wir die Fakultät. ACHTUNG: Zum Ausprobieren wählt keine großen Zahlen, da der Rechner schnell überfordert sein wird. Die Fakultät ist folgendermaßen definiert:

$$n! = n * fakultät(n - 1)!$$

$$n! = 1 \begin{cases} n = 0 \\ n = 1 \end{cases}$$

Um das zu berechnen, können wir einfach folgenden Algorithmus verwenden:



# Fibonacci-Folge

Für diejenigen, die schnell fertig sind, gibt es noch die Fibonaccifolge, die folgendermaßen definiert ist: 0,1,1,2,3,5,8,13... . Na, wie geht es wohl weiter?

# Ackermann-Funktion

$$A(m, n) = \begin{cases} n + 1 & \text{if } m = 0 \\ A(m - 1, 1) & \text{if } m \geq 1 \text{ and } n = 0 \\ A(m - 1, A(m, n - 1)) & \text{if } m \geq 1 \text{ and } n \geq 1 \end{cases}$$

Die Ackermann-Funktion gilt nur für Ganzzahlen  $\geq 0$ !

Folgendermaßen kann diese Funktion umgesetzt werden:

```
#### Die Ackermann-Funktion, umgesetzt in Python
import sys

sys.setrecursionlimit(10000) # Das Limit für Rekursion in Python ist 1000. Das ist schnell erreicht.

def ackermann(m,n):
    # Für den ersten Fall, dass m=0 ist, wird das Ergebnis n+1 zurückgegeben.
    # Schreibe hier deinen Code für diesen Fall.

    # Für den zweiten Fall, dass m>=1 und n=0 ist, wird als Ergebnis der erneute Aufruf der Funktion übergeben
    mit den beiden Parametern: ackermann(m-1,1)
    # Schreibe hier den Code für diesen Fall.

    # Für den dritten Fall gilt, dass m und n >= 1 sind. In diesem Fall ist das Ergebnis ackermann(m-1,
    ackermann(m,n-1))
    Schreibe hier deinen Code für den dritten Fall.
```

**Teste zunächst mit Zahlen  $\leq 4$ !!**