# Python

## Programmaufbau

```
#Bibliotheken einbinden
import turtle
import random

#Globale Variablen
name = "Manfred"

#Funktionsblock
def nameSetzen(nameNeu):
    global name
    name = nameNeu

#Hauptblock
nameSetzen("Gustav")
turtle.forward(random.randint(1, 10))
```

## Operatoren

### Vergleichsoperatoren

| Operator | Bedeutung | Beispiel |
|---|---|---|
| == | ist gleich | a == b |
| != | ist ungleich | a != b |
| > | größer | a > b |
| < | kleiner | a < b |
| >= | größer gleich | a >= b |
| ⇐ | kleiner gleich | a ⇐ b |

### Logische Operatoren

| Operator | Bedeutung | Beispiel |
|---|---|---|
| and | und | (a == b) and (z > y) |
| or | oder | (a == b) or (z > y) |
| not | nicht | not (a == b) |

### Rechenoperatoren

Zur Durchführung von Berechnungen stehen verschiedene Rechenoperatoren zur Verfügung:

| Operator | Zeichen | Beispiel | Ergebnis |
|---|---|---|---|
| Addition | + | 3 + 4 | 7 |
| Subtraktion | - | 5 - 11 | -6 |
| Multiplikation | * | 3 * 6 | 18 |
| Division | / | 9 / 2 | 4.5 |
| Potenz | ** | 3 ** 4 | 81 |

```
zahl_a = zahl_x + zahl_y
zahl_b = zahl_x - 20
zahl_c = zahl_x * zahl_y
zahl_d = zahl_x / 5
zahl_e = zahl_x ** 4
```

# Einseitige Auswahlstruktur

```
if bedingung:
   anweisung(en)
```

#### Beispiel

```
if x > y:
   turtle.forward(5)
```

# Zweiseitige Auswahlstruktur

Die Alternative besteht aus einer Bedingung, deren Wahrheitswert überprüft wird. Je nach Ergebnis dieser Prüfung wird einer von zwei Anweisungsblöcken ausgeführt.

```
if bedingung:
   anweisung(en)
else:
   anweisung(en)
```

#### Beispiel

```
if x > y:
   turtle.forward(5)
else:
   turtle.forward(20)
```

# Mehrseitige Auswahlstruktur

```
if bedingung:
   anweisung(en)
elif bedingung:
   anweisung(en)
else:
   anweisung(en)
```

#### Beispiel

```
if x > y:
   turtle.forward(5)
elif y > z:
   turtle.forward(10)
else:
   turtle.forward(15)
```

# Zählschleife

```
for Variable in Sequenz:
   Anweisung1
   Anweisung2
   ...
   Anweisungn
```

#### Beispiel

```
for i in range(0,5):
   turtle.forward(100)
   turtle.left(90)
```

# Variablen

Variablen sind also ein Speicher für Werte, die man im Laufe des Programms immer wieder benötigt. Mit der Deklaration benennen wir eine Variable und machen diese dem Compiler bekannt. Mittels der Initialisierung können wir die Variable auf einen initialen Anfangswert setzen.

```
#Deklaration der Variable x mit dem initalen Anfangswert 10
x = 10

#Zuweisung eines neuen Werts: Das was rechts des = steht, wird der Variable,
die links des = steht, zugewiesen
```

```
x = 11
x = x + 1 #x ist somit 12

#Deklaration der Variable erste_frage mit dem initalen Anfangswert "Wie geht
es dir?"
erste_frage = "Wie geht es dir?"
```

# Eingabe

Eingaben einsammeln mit input(): Mit der Input-Funktion können wir einen Benutzer zu einer Eingabe auffordern. Die Eingabe können wir dann z.B. einer Variable zuweisen und damit für eine spätere Verwendung speichern.

```
lieblingszahl_benutzer = input("Wie lautet deine Lieblingszahl?")
```

### Umwandlung von Datentypen

- Mit `float()` eine Zeichenkette in eine Gleitkommazahl umwandeln
- Mit `int()` eine Zeichenkette in eine Ganzzahl umwandeln

# Ausgabe

Ausgabe von Daten mit print()

```
#Beispiel 1
print("Hallo Du da!")

#Beispiel 2
frage_eins = "Wie geht es dir?"
print(frage_eins)

#Beispiel 3
x = 5
y = 10
z = x + y
print(x, y, z)
```

# Zufallszahlen

https://docs.python.org/3/library/random.html

Eine natürliche Zufallszahl gibt uns folgende Anweisung zurück:

```
random.randint(m, n)
```

Der Aufruf `random.randint(1, 10)` liefert also eine natürliche Zahl zwischen 1 und 10, der Aufruf `random.randint(5, 8)` eine natürliche Zahl zwischen 5 und 8.

```
#Paket importieren
import random
#Anweisung
random.randint(ersteZahl, letzeZahl)
```

**Beispiel**

```
import random
zahl = random.randint(1, 100)
print(zahl)

turtle.forward(random.randint(1, 10))
```

# Funktionen ohne Parameter

```
def bezeichner():
    Anweisungen
```

**Beispiel**

```
#Funktionsblock
#Hier werden Funktionen definiert, z.B.
def vor_und_zurueck():
   turtle.forward(100)
   turtle.right(180)
   turtle.forward(100)
   turtle.right(180)


######################
#Hauptblock
#Hier folgt der Aufruf der Funktionen, z.B.
vor_und_zurueck()
```

# Funktionen mit Parameter

Das sind Funktionen, denen man Informationen übermittelt, die dann von der Funktion verarbeitet werden und bei jedem Aufruf ein darauf aufbauendes Ergebnis liefern. Die Informationen, die die Funktionen erhalten, nennt man **Parameter**.

### Beispiel

```
#Funktion mit Parametern
def nutzer_gruessen(vorname, nachname):
    print("Herzlich willkommen,", vorname, nachname)

#Funktionsaufruf
nutzer_gruessen("Manfred", "Meyer")

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

#Eingabeaufforderung zur Veranschaulichung eines weiteren Beispiels
vornameEingabe = input("Bitte Vornamen eingeben: ")
nachnameEingabe = input("Bitte Nachname eingeben: ")

#Funktionsaufruf mit Variablen
nutzer_gruessen(vornameEingabe, nachnameEingabe)
```

## Funktionen mit Rückgabewert

Programmierer können nicht nur Parameter an eine Funktion übergeben, sondern auch Ergebnisse mit einer Funktion an den Aufrufort zurückliefern. Diese Ergebnisse heißen **Rückgabewerte**. Rückgabewerte können sowohl Zahlen, Zeichenketten als auch Wahrheitswerte sein.

### Beispiel

Als Beispiel hier ein Programm, welches das Quadrat einer eingegebenen Zahl errechnet und ausgibt:

```
def berechne_quadratzahl(zahl):
    quadratzahl = zahl * zahl
    return quadratzahl

zahl = float(input("Geben Sie bitte eine Zahl ein: "))
ergebnis = berechne_quadratzahl(zahl)
print("Die Quadratzahl von", zahl, "ist", ergebnis)
```

## Turtle Grafik

Bibliothek `turtle graphics` - https://docs.python.org/3.3/library/turtle.html

### Häufige Anweisungen

```
#turtle.forward(distanz)
turtle.forward(10)
```

```
#turtle.right(winkel)
turtle.right(90)

#turtle.left(winkel)
turtle.left(180)

#turtle.back(distanz)
turtle.back(10)

#turtle.goto(x,y)
turtle.goto(100,-100)

#Ausrichtung der Turtle bestimmen (0, 90, 180 oder 270)
turtle.setheading(0)

#turtle.dot(size=None, *color)
turtle.dot(20, "blue")

#Ausfüllen einer Form
#To be called just before drawing a shape to be filled.
turtle.begin_fill()
#Fill the shape drawn after the last call to begin_fill().
turtle.end_fill()
```

### Bilddatei exportieren

```
import turtle

ts = turtle.getscreen()
ts.getcanvas().postscript(file="dateiname.eps")
```

Die .eps-Datei konvertieren (z.B. png oder jpg): https://www.epsconverter.com/de.html

### Farbmodus bestimmen

```
s1 = turtle.Screen()
#rgb-Farben
s1.colormode(255)

#Stiftfarbe bestimmen
turtle.pencolor(255,255,255)
```

rgb-Farbtabelle: http://www.am.uni-duesseldorf.de/de/Links/Tools/farbtabelle.html

### Kreisbogen zeichnen

```
import turtle
import random
import math

#Funktion Kreisbogen mit radius (r) und winkel (w)
def kreisbogen(r, w):
    for num in range (0,int(w/10)):
        turtle.left(2)
        turtle.forward((r*2*math.pi)/36)
        turtle.left(2)

kreisbogen(50,200)
```

## Listen

## Listen - Deklarierung und Initialisierung in Python

*Möglichkeit 1:* Erzeugen eines Arrays und direktes Füllen mit Werten:

```
#Deklaration und Initialisierung
namensliste = ["Uli", "Anna", "E-
               mil", "Sophia"]
```

Array erzeugen und füllen:

Deklaration und Initialisierung: namensliste als Array = ["Uli", "Anna", "Emil", "Sophia"]

*Möglichkeit 2:* Hier wird zunächst ein leeres Array erzeugt und dann schrittweise mit Werten gefüllt, indem die Werte mit Hilfe der Funktion `append()` nach und nach an das Ende des bestehenden Arrays angehängt werden:

```
#Deklaration
himmelsrichtungen = []

#Initialisierung
himmelsrichtungen.append("Nord")
himmelsrichtungen.append("Ost")
himmelsrichtungen.append("Süd")
himmelsrichtungen.append("West")
```

Array erzeugen und füllen:

- Deklaration: himmelsrichtungen als Array = []
- Zuweisung: himmelsrichtungen[0] = "Nord"
- Zuweisung: himmelsrichtungen[1] = "Ost"
- Zuweisung: himmelsrichtungen[2] = "Süd"
- Zuweisung: himmelsrichtungen[3] = "West"

## Zugriff auf die einzelnen Listenwerte mit dem Index

Auf die einzelnen Listenelemente kann mit dem Index zugegriffen werden. Dazu gibt man den Namen der Liste an und direkt dahinter in eckigen Klammern die Nummer des gewünschten Listenelementes, z.B.: `liste[3]`
Dabei gilt es zu beachten, dass auf das erste Element in einer Liste mit dem Index 0 zugegriffen wird. Für das zweite Element wird der Index 1 benutzt, für das dritte Element der Index 2, usw.:

**Beispiel**

```
himmelsrichtungen = ["Nord", "Ost", "Süd", "West"]
print(himmelsrichtungen[3])    #Ausgabe: West
print(himmelsrichtungen[0])    #Ausgabe: Nord
```

## Hinzufügen und Verändern von Elementen

Listen (Arrays) sind in Python nicht abschließend, das heißt, es können, wie schon oben gezeigt, mit `append()` weitere Elemente hinzugefügt werden. Außerdem sind die Elemente auch nicht fest. Dadurch können bereits bestehende Elemente mit neuen Werten überschrieben werden. Das geschieht wie beim Auslesen der einzelnen Felder mit dem Index:

**Struktogramm**

| Hinzufügen und Verändern von Arrayelementen |
|---|
| Deklaration und Initialisierung: training als Array = ["Montag", "Donnerstag"] |
| Zuweisung: training[0] = "Dienstag" |
| Zuweisung: training[2] = "Freitag" |
| Zuweisung: training[2] = "Samstag" |

| **Python-Code** | **Inhalt des Arrays `training`:** |
|---|---|
| `training = ["Montag", "Donnerstag"]` | ["Montag", " Donnerstag "] |
| `#Verändern des ersten Wertes`<br>`training[0] = "Dienstag"` | ["Dienstag ", " Donnerstag "] |
| `#Hinzufügen am Ende`<br>`training.append("Freitag")` | ["Dienstag ", " Donnerstag ", "Freitag"] |
| `#Verändern des dritten Wertes`<br>`training[2]= "Samstag"` | ["Dienstag", "Donnerstag", "Samstag"] |

## Anzahl der Elemente eines Arrays

Bei der Programmierung ist es oft wichtig, die Anzahl der Listenelemente (Feldelemente) zu kennen. In Python kann dies mit der Funktion `len(liste)` ermittelt werden:

```
training = ["Montag", "Mittwoch", "Freitag"]
anzahl = len(training)
print("Trainingstage:", anzahl)
```

Dieses Programm gibt die folgende Zeile am Bildschirm aus:

```
Trainingstage: 3
```

**Struktogramm**

| Anzahl der Elemente eines Arrays |
|---|
| Deklaration und Initialisierung: training als Array = ["Montag", "Mittwoch", "Freitag"] |
| Deklaration und Initialisierung: anzahl als Ganzzahl = Anzahl der Elemente des Arrays training |
| Ausgabe: "Trainingstage: " + anzahl |

## Zugriff auf die Elemente eines Arrays mit einer for-Schleife

Häufig müssen in einem Programm alle Elemente einer Liste ausgegeben werden. Betrachte die folgende Liste:

```
zahlenliste = [12, 9, 17, 31, 8, 24, 51, 67, 2, 34]
```

Um dieses Array vollständig auszugeben, wäre folgender Programmcode denkbar:

```
print(zahlenliste[0])
print(zahlenliste[1])
...
print(zahlenliste[8])
print(zahlenliste[9])
```

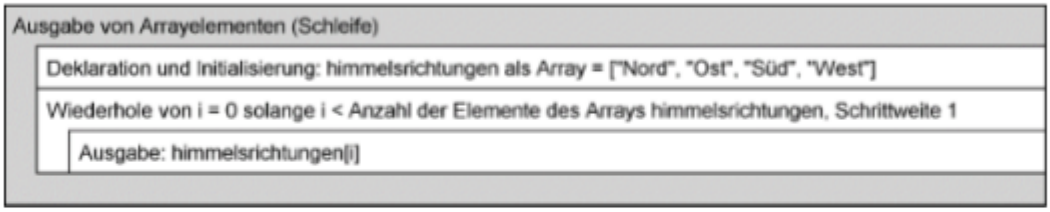
Wirklich befriedigend ist dieses Programm nicht. Das wird spätestens deutlich, wenn man sich vorstellt, dass man ein Array mit 100, 1000 oder 10.000 Zahlen hätte. 10.000 Zeilen Programmcode, um ein Array auszugeben? Das muss einfacher gehen!
An dieser Stelle kommt uns die schon bekannte for-Schleife zur Hilfe:

Soll das ganze Array ausgegeben werden, kann man eine for-Schleife für die Ausgabe verwenden:

```
himmelsrichtungen = ["Nord", "Ost", "Süd", "West"]

for i in range(len(himmelsrichtungen)):
    print(himmelsrichtungen[i])
```

**Struktogramm**




Dieser Ansatz sieht schon deutlich besser aus, denn hier werden alle Werte des Arrays mit einer Schleife ausgegeben. Das Gute dabei ist: auch wenn das Array statt 4 Werten z.B. 10.000 Werte hätte, wäre das Programm keine Zeile länger, da auch hier die Schleife alle Werte des Arrays ausgeben würde.

# Python im Unterrichtswiki

- Alternative
- Eingabe & Ausgabe
- Farben verwenden

- Funktionen mit Rückgabewert
- Funktionen ohne Parameter
- Python
- Python - Programmieraufgaben 1
- Spielwiese - Python Online Editor
- Turtle bewegen
- Variablen
- Verschachtelung
- Zählschleife

[informatik, arbeitsauftrag, computerkunst, lernpfad, python]