Python  Kurs für Anfänger

Jochen Leeder

Was ist Python?

- Python ist eine hochgradig lesbare Programmiersprache.
- Sie ist einfach zu erlernen und vielseitig einsetzbar.
- Wird für Webentwicklung, Datenanalyse, KI und mehr verwendet.

Warum Python?

- Klare, einfache Syntax, die die Programmierung verständlicher macht.
- Große Community und umfangreiche Bibliotheken.
- Plattformunabhängig: Läuft auf Windows, Mac und Linux.

Python installieren

1. Gehe zu python.org
2. Lade den Installer für dein Betriebssystem herunter.
3. Führe den Installer aus und folge den Anweisungen.

Dein erstes Python-Programm

- Öffne einen Texteditor z. B. in MacOS den coteditor
- Schreibe:

```
print("Hallo Welt!")
```

- Speichere die Datei als `hallo.py` in ein dafür angelegtes Verzeichnis z. B. Python_Kurs

Dein erstes Programm aufrufen

- Unter macOS musst du im Terminal zunächst in das Verzeichnis navigieren in der die Datei Python Datei liegt `cd /dein Verzeichnis/`
- Führe sie im Terminal mit `/usr/local/bin/ python3 hallo.py` aus.

Variablen

- Speichern von Daten für die spätere Verwendung.

```
# Variable als string
name = "Welt"
# Variable als Integer damit kann man rechnen!
alter = 30

print(name)
print(alter)
```

Grundlegende Datentypen

- Ganzzahlen (Integer) `5`
- Fließkommazahlen (Float) `5.7`
- Zeichenketten (Strings) `"Hallo Welt"`
- Boolesche Werte (Boolean) `"True" , "False"`

```
a = 5 # Datentyp Integer (Damit kann man rechnen)
b = "5" #Datentyp String (damit kann man nicht rechnen)
c = "5.8" #Datentyp String (damit kann man nicht rechnen)
d = int(a) # jetzt kann man mit c rechnen , da der Datentyp geändert wurde!
e = float(b) # e ist jetzt eine Fließkommazahl, wichtig ist einen . zu setzen und kein Komma ,
bool_value_1 = True
bool_value_2 = False
```

Operatoren

- Addition: `+`
- Subtraktion: `-`
- Multiplikation: `*`
- Division: `/`

Beispiele zu Operatoren

```
a = 5
b = 3
c = a + b

print (c)

d = a * b

print (d)
```

Input Befehle

Der Input Befehl wird genutzt, wenn man User etwas eingeben soll
z.b. sein Alter?

```
alter = int(input("Wie alt bist du?: "))
```

Bedingungen

- `if`, `elif`, `else`
- Beispiel:

```
if alter > 18:
    print("Du bist volljährig.")
else:
    print("Du bist minderjährig.")
```

Schleifen

- Schleifen werden verwendet, um Code mehrfach auszuführen.
- Es gibt zwei Haupttypen von Schleifen in Python: `for` -Schleifen und `while` -Schleifen.

`for` -Schleife

- Wird verwendet, um über eine Sequenz (wie eine Liste oder Zeichenkette) zu iterieren.
- Beispiel:

```
# Iteration über eine Liste
früchte = ["Apfel", "Banane", "Kirsche"]
for frucht in früchte:
    print(frucht)

# Iteration über eine Zeichenkette
wort = "Python"
for buchstabe in wort:
    print(buchstabe)
```

`for` -Schleife in range

Schleifen die einen Bereich abdecken

```
for a in range (3,9):
    print(a)
```

Gibt die Zahlen von 3-8 aus

`while` -Schleife

- Führt Code so lange aus, wie eine Bedingung wahr ist.
- Beispiel:

```
# Zählen von 1 bis 5
zahl = 1
while zahl <= 5:
    print(zahl)
    zahl += 1  # Erhöht die Zahl um 1
```

Schleifen mit `break` und `continue`

- `break` beendet die Schleife vorzeitig.
- `continue` überspringt den aktuellen Schleifendurchlauf und macht mit dem nächsten weiter.

```
# Verwendung von break
for zahl in range(1, 11):
    if zahl == 5:
        break
    print(zahl)
# Verwendung von continue
for zahl in range(1, 11):
    if zahl == 5:
        continue
```

Übung: Schleifen

1. Schreibe eine `for`-Schleife, die die Zahlen von 1 bis 10 ausgibt.
2. Schreibe eine `while`-Schleife, die die Zahlen von 10 bis 1 in umgekehrter Reihenfolge ausgibt.

Lösung: Schleifen

```
# Lösung für Übung 1
for zahl in range(1, 11):
    print(zahl)

# Lösung für Übung 2
zahl = 10
while zahl >= 1:
    print(zahl)
    zahl -= 1
```

Zusammenfassung Schleifen

- Schleifen ermöglichen die Wiederholung von Codeblöcken.
- `for` -Schleifen eignen sich gut für die Iteration über Sequenzen.
- `while` -Schleifen sind nützlich, wenn die Anzahl der Durchläufe nicht im Voraus bekannt ist.
- `break` und `continue` ermöglichen eine feinere Kontrolle über den Schleifenfluss.

Listen

- Listen sind geordnete Sammlungen von Elementen.
- Sie können verschiedene Datentypen enthalten.
- Listen sind veränderbar, d.h., man kann Elemente hinzufügen, entfernen oder ändern.

Listen erstellen

- Eine Liste wird mit eckigen Klammern `[]` erstellt.
- Beispiel:

```
# Eine leere Liste
leere_liste = []

# Eine Liste mit verschiedenen Datentypen
meine_liste = [1, "Hallo", 3.5, True]

print(meine_liste)
```

Elemente zugreifen

- Auf Elemente einer Liste greift man mit dem Index zu.
- Indizes beginnen bei 0.
- Beispiel:

```
früchte = ["Apfel", "Banane", "Kirsche"]

# Erstes Element
print(früchte[0])

# Letztes Element
print(früchte[-1])
```

Listen bearbeiten

- Listen sind veränderbar. Man kann Elemente hinzufügen, entfernen oder ändern.

Elemente hinzufügen

- Mit der Methode `append()` wird ein Element am Ende der Liste hinzugefügt.
- Beispiel:

```
früchte = ["Apfel", "Banane"]
früchte.append("Kirsche")
print(früchte)
```

Elemente entfernen

- Mit der Methode `remove()` wird das erste Vorkommen eines Elements entfernt.
- Beispiel:

```
früchte = ["Apfel", "Banane", "Kirsche"]
früchte.remove("Banane")
print(früchte)
```

Elemente ändern

- Man kann ein Element ändern, indem man seinen Index verwendet.
- Beispiel:

```
früchte = ["Apfel", "Banane", "Kirsche"]
früchte[1] = "Erdbeere"
print(früchte)
```

Listen durchsuchen

- Mit einer `for`-Schleife kann man alle Elemente einer Liste durchsuchen.
- Beispiel:

```
früchte = ["Apfel", "Banane", "Kirsche"]
for frucht in früchte:
    print(frucht)
```

Nützliche Listenmethoden

- `len(liste)` : Gibt die Anzahl der Elemente in der Liste zurück.
- `sort()` : Sortiert die Liste in aufsteigender Reihenfolge.
- `reverse()` : Kehrt die Reihenfolge der Elemente um.

```
zahlen = [3, 1, 4, 1, 5, 9]
print(len(zahlen))
zahlen.sort()
print(zahlen)
zahlen.reverse()
print(zahlen)
```

Übung: Listen

1. Erstelle eine Liste mit deinen Lieblingsfilmen.
2. Füge einen weiteren Film zur Liste hinzu.
3. Ändere den zweiten Film in der Liste.
4. Entferne den letzten Film aus der Liste.
5. Durchsuche die Liste und gib jeden Film aus.

Lösung: Listen

```python
# Lösung für Übung 1
filme = ["Inception", "Matrix", "Interstellar"]

# Lösung für Übung 2
filme.append("The Dark Knight")
print(filme)

# Lösung für Übung 3
filme[1] = "Blade Runner"
print(filme)

# Lösung für Übung 4
filme.pop()
print(filme)

# Lösung für Übung 5
for film in filme:
    print(film)
```

Zusammenfassung Listen

- Listen sind flexible, geordnete Sammlungen von Elementen.
- Sie können verschiedene Datentypen enthalten und sind veränderbar.
- Mit Listenmethoden wie `append()`, `remove()` und `sort()` kann man Listen effizient bearbeiten.

Funktionen

- Funktionen sind wiederverwendbare Codeblöcke, die eine bestimmte Aufgabe ausführen.
- Sie helfen, den Code zu strukturieren und zu modularisieren.

Funktionen definieren

- Eine Funktion wird mit dem Schlüsselwort `def` definiert.
- Beispiel:

```
def hallo():
    print("Hallo, Welt!")
```

Funktionen aufrufen

- Eine Funktion wird durch ihren Namen aufgerufen.
- Beispiel:

```
hallo()
```

Funktionen mit Parametern

- Funktionen können Parameter annehmen, um Daten zu verarbeiten.
- Beispiel:

```
def hallo(name):
    print("Hallo, " + name + "!")

hallo("Jochen")
```

Funktionen mit Rückgabewerten

- Funktionen können Werte zurückgeben, die weiterverwendet werden können.
- Beispiel:

```
def addiere(a, b):
    return a + b

ergebnis = addiere(3, 5)
print(ergebnis)
```

Beispiel: Funktion zur Berechnung des Quadrats einer Zahl

```
def quadrat(x):
    return x * x

zahl = 4
print(quadrat(zahl))
```

Funktionen mit Standardparametern

- Parameter können Standardwerte haben, die verwendet werden, wenn keine Argumente übergeben werden.
- Beispiel:

```
def hallo(name="Welt"):
    print("Hallo, " + name + "!")

hallo()
hallo("Jochen")
```

Übung: Funktionen

1. Schreibe eine Funktion, die den Umfang eines Kreises berechnet (`umfang(radius)`).
2. Schreibe eine Funktion, die prüft, ob eine Zahl gerade ist (`ist_gerade(zahl)`).

Lösung: Funktionen

```
import math

# Lösung für Übung 1
def umfang(radius):
    return 2 * math.pi * radius

print(umfang(5))

# Lösung für Übung 2
def ist_gerade(zahl):
    return zahl % 2 == 0

print(ist_gerade(4))
print(ist_gerade(7))
```

Nützliche Tipps für Funktionen

- Funktionen sollten nur eine Aufgabe erfüllen.
- Funktionen sollten klar benannt sein, um ihre Aufgabe zu beschreiben.
- Vermeide zu viele Parameter, um Funktionen übersichtlich zu halten.

Zusammenfassung Funktionen

- Funktionen sind wiederverwendbare Codeblöcke.
- Sie können Parameter annehmen und Werte zurückgeben.
- Funktionen helfen, den Code zu strukturieren und zu modularisieren.

Nächste Schritte

- Dateien einlesen und schreiben: Mit externen Daten arbeiten.
- Fehlerbehandlung: Programme robust machen.
- Objektorientierte Programmierung: Komplexe Programme strukturieren.