Python Grundkurs

Zusammenfassung

https://pitrium.de/

Python Grundkurs

Contents

1.1) Download und Installation

- Download von:
  - https://www.python.org/downloads/

1.2) Entwicklungsumgebung

- IDEs - Integrated Development Environments
  - Texteditoren mit zusätzlichen Funktionalitäten
  - Zum Beispiel: PyCharm, Jupyter Notebook, Spyder, Visual Studio, Sublime Text

1.3) Hello World – Dein erstes Programm

- Neues, leeres Dokument öffnen
- Code:

```
print("Hello World =)")
```

- Speichern
- Ausführen (F5 oder „Ausführen“ – „Modul Ausführen“)

2.1) Variablen Grundwissen

- Variablen dienen als Zwischenspeicher
  - Können Werte wie zum Beispiel Zahlen oder Text speichern

- Datentypen:

Datentyp	Bezeichnung	Gespeicherte Daten	Beispiel
String	str()	Text	„Hello World“
Integer	int()	Ganzzahl	123
Float	float()	Gleitkommazahl	12,34
Boolean	bool()	Wahr oder Falsch	True / False

- Namens Konventionen:
  - Camel Case: meinVariablenName
  - Pascal Case: MeinVariablenName
  - Underscores: mein_variablen_name

- Deklaration:

```
meineGanzzahl = 123
```

- Auslesen:

```
print(meineGanzzahl)

### ERGEBNIS / AUSGABE ###
123
```

- Neuen Wert zuweisen:

```
meineGanzzahl = 99
print(meineGanzzahl)

### ERGEBNIS / AUSGABE ###
99
```

2.2) Strings – Text Manipulation

- Deklaration:

```
meinString = "Das ist mein Text."
print(type(meinString), meinString)

### ERGEBNIS / AUSGABE ###
<class 'str'> Das ist mein Text.
```

- Escape Character:

```
print("Ich möchte gern ein \" in meinem Text haben.")

### ERGEBNIS / AUSGABE ###
Ich möchte gern ein " in meinem Text haben.
```

- Zeilenumbrüche:

```
print("Ich möchte gern \nin zwei Zeilen schreiben.")

### ERGEBNIS / AUSGABE ###
Ich möchte gern
in zwei Zeilen schreiben.
```

- Verketten:

```
print("Will" + "kom" + "men =)")

### ERGEBNIS / AUSGABE ###
Willkommen =)
```

- Formatierung:

```
wer = "Felix"
wieViele = 3
print(f"{wer} hat {wieViele} Äpfel gekauft.")

### ERGEBNIS / AUSGABE ###
Felix hat 3 Äpfel gekauft.
```

2.3) Integer, Float - Berechnungen

- Deklaration:

```
zahl1 = 2
zahl2 = 5.7
print(f"{zahl1} \t {type(zahl1)}")
print(f"{zahl2} \t {type(zahl2)}")

### ERGEBNIS / AUSGABE ###
2       <class 'int'>
5.7     <class 'float'>
```

- Grundrechenarten:

```
print(f"Addieren: {zahl1} + {zahl2} = {ergebnis}")
print(f"Subtrahieren: {zahl1} - {zahl2} = {zahl1 - zahl2}")
print(f"Multiplizieren: {zahl1} * {zahl2} = {zahl1 * zahl2}")
print(f"Dividieren: {zahl1} / {zahl2} = {zahl1 / zahl2}")

### ERGEBNIS / AUSGABE ###
Addieren: 2 + 5.7 = 7.7
Subtrahieren: 2 - 5.7 = -3.7
Multiplizieren: 2 * 5.7 = 11.4
Dividieren: 2 / 5.7 = 0.3508771929824561
```

- Berechnungs Reihenfolge: Klammern, Punktrechnung, Strichrechnung

```
print(f"1 + 1 * 2 = {1 + 1 * 2}")
print(f"(1 + 1) * 2 = {(1 + 1) * 2}")

### ERGEBNIS / AUSGABE ###
1 + 1 * 2 = 3
(1 + 1) * 2 = 4
```

- Potenzieren:

```
print(f"Potenzieren: {zahl1} ** {zahl2} = {zahl1 ** zahl2}")
print(f"Wurzel: {zahl1} = {zahl1 ** 0.5}")

### ERGEBNIS / AUSGABE ###
Potenzieren: 2 ** 5.7 = 51.98415336679908
Wurzel: 2 = 1.4142135623730951
```

2.4) Boolean – Vergleiche, Logische Verknüpfung

- Deklaration:

```
wahr = True
falsch = False
print(f"wahr = {wahr} \t {type(wahr)}")
print(f"falsch = {falsch} \t {type(falsch)}")

### ERGEBNIS / AUSGABE ###
wahr = True      <class 'bool'>
falsch = False   <class 'bool'>
```

- Vergleiche:

```
x = 2
y = 5
print(f"x < y --> {x < y}")
print(f"x <= y --> {x <= y}")
print(f"x > y --> {x > y}")
print(f"x >= y --> {x >= y}")
print(f"x == y --> {x == y}")
print(f"x != y --> {x != y}")

### ERGEBNIS / AUSGABE ###
x < y --> True
x <= y --> True
x > y --> False
x >= y --> False
x == y --> False
x != y --> True
```

- Logische Verknüpfungen:

```
print(f"Wahr UND Wahr --> {True and True}")
print(f"Wahr UND Falsch --> {True and False}")
print(f"Wahr ODER Wahr --> {True or True}")
print(f"Wahr ODER Falsch --> {True or False}")
print(f"NICHT Wahr UND Wahr --> {not True and True}")
print(f"NICHT Falsch UND Wahr --> {not False and True}")

### ERGEBNIS / AUSGABE ###
Wahr UND Wahr --> True
Wahr UND Falsch --> False
Wahr ODER Wahr --> True
Wahr ODER Falsch --> True
```

```
NICHT Wahr UND Wahr --> False
NICHT Falsch UND Wahr --> True
```

3.1) if, elif, else Abfragen

- Prüfung ob eine Bedingung erfüllt ist

- Ausführung des dazugehörigen Codes nur, wenn Ergebnis der Prüfung True ist

- Wenn (diese Bedingung erfüllt ist):
  dann wird dieser Code ausgeführt.
  Oder wenn (diese Bedingung erfüllt ist)
  dann wird dieser Code ausgeführt.
  Sonst:
  wird dieser Code ausgeführt.

- Syntax:

```
bedingung1 = True
bedingung2 = True

if bedingung1:
   print("bedingung1 ist True.")
elif bedingung2:
   print("bedingung1 ist False und bedingung2 ist True.")
else:
   print("bedingung1 und bedingung2 sind False.")

### ERGEBNIS / AUSGABE ###
bedingung1 ist True.
```

3.2) while Schleifen

- wiederholte Ausführung des dazugehörigen Codes solange eine Bedingung erfüllt ist

- Während (diese Bedingung erfüllt ist):
  wird dieser Code ausgeführt.

- Richtlinie: verwende eine while Schleife, wenn du nicht genau einschätzen kannst wie oft die Schleife durchlaufen werden soll

- Syntax:

```
i = 0
while i < 5:
   print(f"Bedingung True - i = {i}")
   i = i + 1

### ERGEBNIS / AUSGABE ###
Bedingung True - i = 0
Bedingung True - i = 1
Bedingung True - i = 2
Bedingung True - i = 3
Bedingung True - i = 4
```

- Endlosschleife mit Abbruchbedingung:

```
while True:
   print(f"Immer noch True - i = {i}")
   if i == 9:
      break
   i += 1

### ERGEBNIS / AUSGABE ###
Immer noch True - i = 5
Immer noch True - i = 6
Immer noch True - i = 7
Immer noch True - i = 8
Immer noch True - i = 9
```

3.3) for Schleifen

- mehrmalige Ausführung des dazugehörigen Codes

- Für jeden Wert der VARIABLE im Bereich (von – bis):
  wird dieser Code ausgeführt.

- Richtlinie: Verwende eine for Schleife, wenn du bereits weißt wie oft deine Schleife durchlaufen werden soll

- Syntax:

```
for i in range(5):
   print(f"Bedingung True - i = {i}")

### ERGEBNIS / AUSGABE ###
Bedingung True - i = 0
Bedingung True - i = 1
Bedingung True - i = 2
Bedingung True - i = 3
Bedingung True - i = 4
```

- range() Anpassung:

```
for i in range(1,10,2):
   print(f"Bedingung True - i = {i}")
for i in range(0,-5,-1):
   print(f"Bedingung True - i = {i}")

### ERGEBNIS / AUSGABE ###
Bedingung True - i = 1
Bedingung True - i = 3
Bedingung True - i = 5
Bedingung True - i = 7
Bedingung True - i = 9
Bedingung True - i = 0
Bedingung True - i = -1
Bedingung True - i = -2
Bedingung True - i = -3
Bedingung True - i = -4
```

4.1) Funktionen

- Wiederverwendbare Code Teilabschnitte

- Definition:

```
def NameMeinerFunktion():
   print("Code meiner Funktion wird ausgeführt.")

NameMeinerFunktion()

### ERGEBNIS / AUSGABE ###
Code meiner Funktion wird ausgeführt.
```

- Parameter:

```
def Addieren(zahl1, zahl2):
   print(zahl1 + zahl2)

Addieren(2, 3)

### ERGEBNIS / AUSGABE ###
5
```

- Rückgabewerte:

```
def RechteckFlaecheUndUmfang(laenge, breite):
   flaeche = laenge * breite
   umfang = 2 * (laenge + breite)
   return flaeche, umfang

laenge = 2
breite = 3
flaeche, umfang = RechteckFlaecheUndUmfang(laenge, breite)
print(f"Länge = {laenge}; Breite = {breite}; Fläche = {flaeche}; Umfang =
{umfang}")

### ERGEBNIS / AUSGABE ###
Länge = 2; Breite = 3; Fläche = 6; Umfang = 10
```

5.1) Module

- Sammlungen von bereits getesteten und optimierten Funktionen und Code

- Einbinden:

```
import time
from datetime import datetime
import math as m

for i in range(3):
   print(f"{datetime.now()} - {m.pi}")
   time.sleep(1)

### ERGEBNIS / AUSGABE ###
2020-04-23 19:07:08.299398 - 3.141592653589793
2020-04-23 19:07:09.314949 - 3.141592653589793
2020-04-23 19:07:10.315151 - 3.141592653589793
```

- Installation mit pip
  - pip install MODULNAME

- Auflisten aller installierter Module
  - pip list