Python - Befehlsübersicht

1 Grundlagen

1.1 Kommentare

Kommentare gehören nicht zum eigentlichen Programm, sondern sind als Hilfe für den Programmierer gedacht.

```
# Ein Kommentar, der bis ans Zeilenende geht

""" Ein Kommentar, der
über mehrere Zeilen geht.
"""
```

1.2 Variablen

Variablen sind Wertespeicher. Man kann darin einen Wert ablegen und diesen wieder abfragen. Sie sind wie eine kleine Box, die einen Namen hat und einen Wert enthält. Um einer Variablen einen Wert zuzuweisen, muss sie links von einem =-Zeichen stehen.

```
# Eine neue Variable Zahl mit dem Namen 'meineZahl' und Wert 42
meineZahl = 42

# Eine Text-Variable. Texte stehen in "..".
meinText = "Hallo Welt"
```

1.3 Ausgabe

```
# Gibt einen Text aus
print("Hallo Welt")

# Gibt den Wert einer Variablen aus
name = "Mark"
print("Hallo")
print(name)

# 'print' gibt auch mehrere Werte mit Leerzeichen getrennt aus
print("Hallo", name)
```

1.4 Rechnen mit Zahlen

Man kann mit Zahlen und Variablen ganz normal rechnen.

```
# Mit Zahlen kann ganz normal rechnen
wert = 10 + 20 # = 30
wert = 10 - 20 # = -10
wert = 10 / 2 # = 5
wert = 10 * 2 # = 20
wert = 10 % 3 # = 1, da 10 / 3 = 3 Rest 1
wert = 10 ** 3 # = 1000 = 10 * 10 * 10
```

1.5 Eingabe durch den Benutzer

1.5.1 input - Text einlesen

Die Funktion `input(..)` fordert den Benutzer auf einen Text einzugeben.

```
eingabeText = input("Bitte Text eingeben: ")
print(eingabeText) # gibt den eingelesenen Text aus
```

1.5.2 input - Eine Zahl einlesen

Eine Zahl, die als Text vorliegt (in Anführungszeichen) konvertieren wir mittels `int(..)` in eine Zahl:

```
eingabeText = input("Bitte eine Zahl eingeben: ");

eingabeZahl = int(eingabeText)
# ACHTUNG: wird keine Zahl eingeben, so stürzt das Programm ab!

print(eingabeZahl)
```

2 Text

Ein Text ist im Prinzip nur eine Liste von Buchstaben und Zeichen. Man kann daher Listenfunktionen wir `len(..)`, den Zugriff auf Element mittels `text[1]` verwenden oder einen Text Buchstabenweiße mit einer `for`-Schleife durchlaufen. Mehr dazu in der Sektion über Listen.

2.1 Texte kombinieren

Texte kann man '+' aneinander anfügen.

```
begruessung = "Hallo "
text = begruessung + "Mark" # "Hallo Mark"
print(text)

# Texte kann man auch multiplizieren/wiederholen
text = "Ha" * 3 # HaHaHa
print(text)
```

2.2 Länge eines Texts

Die Funktion `len(..)` ermittelt die Länge von Texten, Listen, etc.

```
text = "Hallo Welt"
laenge = len(text) # = 10
print(laenge)
```

2.3 Einen einzelnen Buchstaben abfragen

Einen Buchstaben kann man über seine Position im Text abfragen:

```
text = "Hallo Welt"
buchstabe = text[1]
print(buchstabe) # = 'a'
```

ACHTUNG: Wir beginnen bei 0 zu zählen, die 1 beschreibt also den zweiten Buchstaben!

2.4 Texte sind unveränderlich

Man kann in einen Text nicht ohne weiteres verändern:

```
text = "Hallo Welt"
text[1] = "o" # Erzeugt einen Fehler!
```

3 Bedingungen und if-Abfragen

3.1 Bedingung

Eine Bedingung ist entweder Wahr (True) oder Falsch (False).

```
bedingung = 10 > 20 # False
print("Die Bedingung ist:", bedingung)
```

3.2 if-Abfragen

Eine if-Abfrage überprüft ob eine Bedingung Wahr oder Falsch ist.

```
wert = 10 # Eine Variable mit einem beliebigen Wert (hier 10)

if wert > 20: # wird NUR ausgeführt wenn die Bedingung wahr ist
    print("Der Wert ist größer als 20") # (ACHTUNG: Einrückung)

# Hier gehts weiter (ACHUTUNG: Nicht eingerückt)
```

3.3 if-else-Abfragen

Mit einer if-else Abfrage kann man auch auf eine nicht erfüllte Bedingung mit dem 'else'-Zweig reagieren.

```
if wert > 20: # wird ausgeführt wenn die Bedingung erfüllt ist
    print("Der Wert ist größer als 20")

else: # wird ausgeführt wenn die Bedingung nicht erfüllt ist
    print("Wert ist kleiner oder gleich 20.")

# Hier gehts weiter (nicht eingerückt)
```

3.4 Vergleichsoperationen:

Alle Vergleichsoperationen liefern entweder Wahr oder Falsch. Diese können als Bedingung in einer `if`-Abfrage verwendet werden.

3.4.1 Standardvergleichsoperationen

```
a = 10
b = 20
```

```
a == b # = False
a != b # = True
a > b  # = False
a >= b # = False
a < b  # = True
a <= b # = True
```

3.4.2 Überprüfen, ob ein Textstück in einem Text enthalten ist

```
a = "Hallo Welt"

"allo" in a # = True
"oo" in a   # = False
```

3.4.3 Überprüfung auf None

```
a = "Hallo Welt"

a is None # = False
a is not None # = True
```

4 Schleifen

4.1 while-Schleife

Eine `while`-Schleife führt den eingerückten Code aus, solange die Bedingung erfüllt ist.

```
zahl = 1
while zahl < 10: # solange die Bedingung wahr ist...
    # ...wird dieser eingerückte Code wird wiederholt
    print("Zahl:", zahl)
    zahl = zahl + 1 # Zähler erhöhen

# Hier gehts normal weiter (Achtung: nicht eingerückt)
```

Achtung: Ist die Bedingung immer wahr läuft das Programm in einer Endlosschleife.

4.2 for-Schleife

Eine `for`-Schleife ird verwendet um eine Liste, Text, o.ä. zu durchzulaufen. Der eingerückte Code wird für jedes Listenelment einmal ausgeführt.

4.2.1 Eine Liste durchlaufen:

```
liste = [1, 2, 3, 4]

for zahl in liste:
    print("Zahl:", zahl)

# Hier gehts normal weiter (ACHTUNG: nicht eingerückt)
```

4.2.2 Einen Text buchstabenweiße durchlaufen:

```
text = "abcdefghijklmnopqrstuvwxyz"

for buchstabe in text:
    print("Buchstabe:", buchstabe)
```

4.3 Geschachtelte Schleifen

Bei geschachtelten Schleifen befindet sich eine Schleife innerhalb einer anderen Schleife! D.h. die innere Schleife und somit der Code innerhalb der inneren Schleife wird mehrmals ausgeführt!
In diesem Beispiel wird die äußere `for`-Schleife 9 mal ausgeführt. Die innere `for`-Schleife 4 mal. Der Code in der inneren Schleife wird also: `9 * 4 = 36` mal ausführt!

```
for zahlAussen in range(1, 10): # nimmt Werte 1-9 an
    # Achtung Einrückung!

    for zahlInnen in range(1, 5): # nimmt Werte 1-4 an
        # Achtung nochmal einrücken!
        print("Außen:", zahlAussen, "Innen:", zahlInnen)
```

Hinweis: `range(untergrenze, obergrenze)` erzeugt eine Liste der Zahlen von der Untergrenze bis zur Obergrenze. Die Obergrenze ist allerdigs nicht enthalten.

5 Listen

5.1 Eine Liste erstellen

```
# Eine leere Liste erstellen
liste = []

# eine Liste mit Einträgen erstellen
liste = [1, 2, 3, 4, 5]
```

5.2 Einen Eintrag aus der Liste lesen

Ein Wert aus der Liste wird über seinen Index, seine Position in der Liste, abgefragt.

```
liste = ["hallo", "test", "welt"]

ersterEintrag = liste[0] # = "hallo".
```

Achtung: Wir beginnen bei 0 zu zählen! Das erste Element hat den Index 0.

5.3 Einen Eintrag von Ende der Liste lesen

```
# Negative Indices beginnen am Ende zu zählen
letzterEintrag = liste[-1] # = "welt".

print(ersterEintrag, letzterEintrag) # => "Hallo Welt"
```

Achtung: wir beginnen hinten bei -1 zu zählen! Das letzte Element hat den Index -1.

5.4 Länge einer Liste ermitteln

```
liste = [1, 2, 3]

# Die Länge einer Liste (die Anzahl der Elemente) abfragen:
laenge = len(liste) # = 3
print(laenge)
```

5.5 Einen Eintrag hinzufügen

Die `.append(..)`-Funktion fügt einen Eintrag ans Ende der Liste an.

```
liste = [1, 2]

# einen Eintrag anfügen
liste.append(3)
```

5.6 Eine Liste mit einer `for`-Schleife durchlaufen

Die einzelnen Elemente der Liste werden mittels einer `for`-Schleife durchlaufen.

```
for element in liste:
    print(element)
```

5.6.1 Hilfsfunktion für das Ausgeben einer Liste

Gibt die Anzahl der Elemente und deren Werte aus.

```
def liste_ausgeben(liste):
    print("Liste mit", len(liste), "Einträgen")

    # die Elemente ausgeben
    for element in liste:
        print(element)

# Diese ruft man so auf
liste = [1, 2, "Haus", "Katze"]
liste_ausgeben(liste)
```

5.7 Überprüfen, ob ein Eintrag vorhanden ist

Mittles `x in liste` wird überprüft ob der Wert von `x`, z.B. 1 in der Liste vorhanden ist

```
liste = [1, 2, "hallo", "welt"]

if 1 in liste:
    print("Enthalten")
else:
    print("Nicht entalten")
```

5.8 Einen Eintrag entfernen

Die `.remove(..)`-Funktion sucht den ersten Eintrag mit diesem Wert und entfernt ihn.

```
liste = [1, 2, 6]
liste.remove(6) # = [1, 2]

liste_ausgeben(liste)
```

Achtung: Hat es keinen solchen Wert, wird ein Fehler erzeugt!

5.9 Zwei Listen kombinieren

Wie bei Texten kann man mit + zwei Listen kombinieren.

```
liste = [1, 2, 3, 4, 5, 6]
liste = liste + [7, 8, 9] # = [1, 2.., 9]

liste_ausgeben(liste)
```

6 Funktionen

6.1 Eine einfache Funktion

Funktionen bestehen aus einem Block von eingerückten Code-Zeilen. Diese werden erst ausgeführt, wenn die Funktion aufgerufen wird.

```
# Neue Funktion mit dem Namen 'sageHallo' definieren
def sageHallo():
    print("Hallo Welt") # Einrückung!

# Funktion aufrufen (keine Einrückung)
sageHallo()
```

6.2 Funktion mit Übergabeparametern

Funktion können Parameter (Variablen, die nur innerhalb der Funktion existieren) haben. Diese *müssen* beim Aufrufen mitübergeben werden.

```
# Funktion definieren
def sageHallo(name, alter):
    print("Hallo", name)
    print("Du bist", alter, "Jahre alt")
```

```
# Funktion aufrufen
sageHallo("Mark", 22)
```

6.3 Funktion mit Rückgabewerten

Funktion können einen Wert zurückgeben, müssen aber nicht. Wenn nichts zurückgeben wird, wird automatisch `None` zurückgeben.

```
# Funktion definieren
def addiere(zahl1, zahl2):
    summe = zahl1 + zahl2
    return summe

# Funktion aufrufen
ergebnis = addiere(12, 22)
print(ergebnis)
```