NAMESRÄUME, GÜLTIGKEITSBEREICHE

Verhalten von Variablen in Unterprogrammen

LOKALE/GLOBALE VARIABLEN

```
def add1(x):
    y=x+1
    return y

a=0
b=add1(a)
print(a,b)
```

- a und b sind globale Variablen des Hauptprogramms
- x und y sind lokale Variablen des Unterprogramms

NAMENSRÄUME

```
def add1(x):
    y=x+1
    return y

y=0
b=add1(y)
print(y,b)
```

- y ist globale als auch lokale Variable
- Kein Konflikt, da sie einen unterschiedlichen Namensraum / Gültigkeitsbereich haben

PARAMETER EINER FUNKTION

```
def add1(x):
    y=x+1
    return y

y=0
b=add1(y)
print(y,b)
```

- Parameter in einer Funktion werden wie lokale Variablen behandelt

ZUGRIFF AUF GLOBALE VARIABLEN

```
def add1():
    y=x+1
    return y

x=0
z=add1()
print(x,z)
```

- Auf globale Variablen kann **lesend** zugegriffen werden
- Vorraussetzung: es gibt keine lokale Variable mit gleichem Namen
- Schlechter Stil: möglichst vermeiden

ZUGRIFF AUF LOKALE VAR.

```
def add1():
    y=x+1
    return y

x=0
z=add1()
print(x,y,z)
```

- Eine lokale Variable ist nur innerhalb des Unterprogramms gültig
- Im Hauptprogramm kann nicht auf die lokale Variable zugegriffen werden

FEHLER

```
def add1():
    x=x+1
    return x

x=0
z=add1()
print(x,z)
```

- x ist lokale Variable
- Hat in Zeile 2 keinen Wert -> Fehler

SCHREIBENDER ZUGRIFF, GLOBAL

```
def add1():
    global x
    x=x+1
    return x

x=0
z=add1()
print(x,z)
```

- Schlüsselwort global
- Lesender und schreibender Zugriff auf globale Variable

PARAMETER, VERÄNDERLICHE OBJEKTE

PARAMETER ÄNDERN

- Funktionen können Parameter übergeben werden
- Den Einfluss der Funktion auf den Parameter hängt vom Objekt ab
  - Veränderliche Objekte: Listen, numerische Datentypen
  - Unveränderliche Objekte: Strings

BEISPIEL I

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

def foo(any_list):
    any_list.append(4)

foo(my_list)

print(my_list)
# Ergebnis: [1,2,3,4]
```

Das Argument kann innerhalb der Funktion geändert werden.
Auch außerhalb der Funktion ist diese Änderung gültig.

BEISPIEL 2

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

def foo(any_list):
    any_list = [1,2,3,4]

foo(my_list)

print(my_list)
# Ergebnis: [1,2,3]
```

Dem übergebenen Parameter wird ein neues Objekt (hier: [1,2,3,4]) zugewiesen.

Das entspricht der Definition einer lokalen Variablen. Any_List ist also lokal.

Die Änderung ist damit nur lokal.

BEISPIEL 3

```
my_string = 'Hello World!'

def foo(any_string):
  any_string.replace('World', 'Python')

foo(my_string)

print(my_string)
# Ergebnis: 'Hello World!'
```

Der übergebene Parameter ist ein String. Dieser gilt als unveränderlich. Es wird daher nur eine Kopie übergeben.

Die Änderung ist daher auch nur lokal.

BEISPIEL 4

```
my_string = 'Hello World!'

def foo(any_string):
  x = any_string.replace('World', 'Python')
  return x

my_string = foo(my_string)

print(my_string)
# Ergebnis: 'Hello Python!'
```

Eine neues Zuweisung zu einer String Variablen wird durch eine neue Zuweisung realisiert.