Rückgabewert bei Funktionen

Funktionen sind praktisch um immer wieder verwendeten Code nutzen zu können. Bisher haben wir bei unseren Funktionen in Python immer fleißig Daten in die Funktion reingegeben. In diesem Kapitel lassen wir uns Ergebnisse aus einer Funktion herausgeben.

Mit den herausgegebenen Ergebnissen in Form von Variablen können wir dann im weiteren Programmcode nach Belieben weiteres anstellen.

Unsere Funktion haben dann folgenden Aufbau:

```
def bspfunktionfuerrueckgabe(eingabewert):
    rueckgabewert = eingabewert * 2
    return rueckgabewert

ergebnisausfunktion = bspfunktionfuerrueckgabe (5)
print(ergebnisausfunktion)
```

Wir übergeben in unserem obigen Beispiel die Zahl 5 in unserer Funktion mit dem vielsagenden Namen „`bspfunktionfuerrueckgabe`“. In der Funktion wird nun etwas mit dem hereingegeben Wert angestellt – im Beispiel einfach verdoppelt und dann über `return` das Ergebnis wieder aus der Funktion gegeben.

Außerhalb bekommt unser Funktionsaufruf vorneweg eine Variable, die das zurückgelieferte Ergebnis aufnehmen soll und ein Gleichheitszeichen.

Warum Variable über `return` übergeben

Warum müssen wir überhaupt die Variable über die `return`-Funktion zurückgeben? Eigentlich geben wir nicht die Variable, sondern den Wert der Variable zurück.

Die Variable steht außerhalb der Funktion nicht zur Verfügung. Probieren wir in unserem Python-Programm einfach nach Aufruf der Funktion direkt auf die Variable `rueckgabewert`, die nur innerhalb der Funktion benutzt wird, außerhalb der Funktion zu nutzen, erhalten wir die Fehlermeldung: „NameError: name 'rueckgabewert' is not defined“

```
def bspfunktionfuerrueckgabe(eingabewert):
    rueckgabewert = eingabewert * 2
    return rueckgabewert

ergebnisausfunktion = bspfunktionfuerrueckgabe (5)
print(ergebnisausfunktion)
print(rueckgabewert)
```

Dies ist eine extrem praktische Einrichtung, da wir beim Erstellen unserer Funktion nicht auf die genutzten Variablennamen außerhalb der Funktion achten müssen. Wir können alles nach Belieben verwenden.

Geltungsbereich/Gültigkeitsbereich von Variablen

Das Verständnis der Unterschiede zwischen globalen und lokalen Variablen ist extrem wichtig bei der Verwendung von Variablen innerhalb und außerhalb von Funktionen.

Bauen wir für das Verständnis ein kleines Python-Programm auf, dass nur für die Nutzung der Variablen da ist. Dazu haben wir eine Funktion und diese Funktion bekommt Funktionen integriert. Aber Schritt für Schritt:

```
variablenWert = "außerhalb der Funktion"
print("Variablenwert vor Funktion:", variablenWert)
def bspfunktion():
    print("Variablenwert in Funktion 1:", variablenWert)
    variablenWert = "IN der Funktion"
    print("Variablenwert in Funktion 2:", variablenWert)

bspfunktion()
print("Variablenwert nach Funktion:", variablenWert)
```

Ab jetzt sind Fehlermeldungen interessant. Das Programm läuft die ersten 2 Zeilen noch problemlos. Die Variable mit dem Namen `variablenWert` wird vor der Funktion gesetzt und direkt ausgegeben. Das funktioniert problemlos.

Danach wird die Funktion aufgerufen und in der Funktion soll sofort die außerhalb gesetzte Variable `variablenWert` ausgegeben werden. Jetzt erhalten wir unsere erste Fehlermeldung „UnboundLocalError: local variable 'variablenWert' referenced before assignment“. Die Variable existiert offensichtlich nicht innerhalb der Funktion.

Also nehmen wir diese Zeile raus und setzen der Wert der Variable neu innerhalb der Funktion.

```
variablenWert = "außerhalb der Funktion"
print("Variablenwert vor Funktion:", variablenWert)
def bspfunktion():
    variablenWert = "IN der Funktion"
    print("Variablenwert in Funktion:", variablenWert)

bspfunktion()
print("Variablenwert nach Funktion:", variablenWert)
```

Nun bekommen wir keine Fehlermeldung mehr aber der gleiche Variablennamen ist offensichtlich unterschiedlich vom Wert – je nachdem, ob er in oder außerhalb der Funktion benutzt wird.

Als Ergebnis sehen wir:

Variablenwert vor Funktion: außerhalb der Funktion

Variablenwert in Funktion: IN der Funktion

Variablenwert nach Funktion: außerhalb der Funktion

Globale Variablen

Nun steigern wir die Komplexität, da wir eine Variable auch als `global` definieren können.

Innerhalb der Funktion setzen wir unsere Variable "`variablenWert`" auf `global`

```
variablenWert = "außerhalb der Funktion"
print("Variablenwert vor Funktion:", variablenWert)
def bspfunktion():
    global variablenWert
    variablenWert = "IN der Funktion"
    print("Variablenwert in Funktion:", variablenWert)
```

```
bspfunktion()
print("Variablenwert nach Funktion:", variablenWert)
```

Jetzt bekommen wir als Ausgabe:

Variablenwert vor Funktion: außerhalb der Funktion

Variablenwert in Funktion: IN der Funktion

Variablenwert nach Funktion: IN der Funktion

Wir haben also den Wert der außerhalb gesetzten Variablen überschrieben mit einer Variablen in der Funktion.

Und noch eine Steigerung!

nonlocal in Python für Variablen

Zwischen `global` und `local` gibt es noch eine Zwischenform. Dazu muss man wissen, dass wir in Funktionen weitere Funktionen packen können, die aber nur für die jeweilige Funktion innerhalb der Funktion zur Verfügung steht. Und hier können wir in der Funktion in der Funktion ein Variable erstellen, die dann auch in der aufrufenden Funktion verfügbar ist.

Das ist etwas, was man bei Bedarf einfach nochmals durchlesen sollte. Hier der Vollständigkeit halber.

Im Beispiel wird es klarer:

```
variable = "Inhalt außerhalb gesetzt"
def grundfunktion():
    variable = "Inhalt innerhalb gesetzt"

    def fkt_local():
        variable = "Inhalt innerhalb local"

    def fkt_nonlocal():
        nonlocal variable
        variable = "Inhalt innerhalb nonlocal"

    def fkt_global():
        global varibale
        variable = "Inhalt innerhalb global"

    print("2. in Fkt: ", variable)
    fkt_local()
    print("3. in Fkt – flt_local aufgerufen: ", variable)
    fkt_nonlocal()
    print("4. in Fkt – flt_nonlocal aufgerufen: ", variable)
    fkt_global()
    print("5. in Fkt – flt_global aufgerufen: ", variable)

print("1. vor Funktionsaufruf: ", variable)
grundfunktion()
print("6. nach Funktionsaufruf: ", variable)
```

Was erhalten wir als Ergebnis?

1. vor Funktionsaufruf: Inhalt außerhalb gesetzt

2. in Fkt: Inhalt innerhalb gesetzt

3. in Fkt – flt_local aufgerufen: Inhalt innerhalb gesetzt

4. in Fkt – flt_nonlocal aufgerufen: Inhalt innerhalb nonlocal

5. in Fkt – flt_global aufgerufen: Inhalt innerhalb global

6. nach Funktionsaufruf: Inhalt innerhalb global

Schaut man sich das Ergebnis an, wird der Geltungsbereich der Variablen und die Möglichkeiten, diese durch `nonlocal` und `global` zu erweitern, deutlich.

Stichworte dazu sind: `local_scope`

Vorteile von Funktionen

Dadurch werden unsere Funktionen auch für weitere Projekte wiederverwertbar! Benötigen wir wieder diese Funktion, können wir unsere bereits aus einem alten Projekt erstellte Funktion in unser neues Projekt übernehmen (egal welche Variablennamen in der Funktion früher verwendet wurden).

Wie wir universelle verwendbare Funktionen einfach einbindet, sehen wir im folgenden Kapitel.