ZUSAMMENFASSUNG Funktionen und Namensbereiche

EINFÜHRUNG

Funktionen benötigt man vor allem, wenn man langen und komplexen Code schreibt. Denn mit ihnen hat man die Möglichkeit den Code besser zu strukturieren, indem einzelne Funktionalitäten des Programms in unterschiedliche Funktionen ausgelagert werden. Eine Funktion ist wie ein Unterprogramm, das man mit dem Funktionsnamen aufrufen kann und mit dem man alle Anweisungen der aufgerufenen Funktion ausführt.

Vorteile von Funktionen

1. Bessere Strukturierung von Code
2. Gleicher Code kann einfacher wiederholt werden
3. Code kann einfacher verbessert werden
4. Fehleranfälligkeit des Code sinkt

FUNKTIONEN DEFINIEREN UND AUFRUFEN

Eine Funktion wird definiert mit dem Schlüsselwort "def". Darauf folgen ein rundes Klammerpaar und ein Doppelpunkt. Der darauffolgende Funktionskörper wird wieder eingerückt und enthält Anweisungen.

WICHTIG Man muss eine Funktion zuerst definieren bevor man sie im weiteren Programm aufrufen kann.

Funktionen kann man aufrufen, indem man den Bezeichner der Funktion, gefolgt von einem Klammerpaar hinschreibt.
Mit dem Funktionsaufruf springt Python ins Unterprogramm und führt alle Anweisungen der Reihe nach aus, und springt dann wieder dahin zurück, wo die Funktion aufgerufen wurde.

FUNKTIONEN MIT PARAMETERN DEFINIEREN

Parameter einer Funktion sind variable Stellen im Funktionskörper, die man mit den Werten befüllt, die man beim Funktionsaufruf übergibt.

Zum Beispiel:

```
def greeting(name):
    print("Hallo" + name)
```

Hier definieren wir eine Funktion. In dieser Funktion ist “name” ein Parameter, er funktioniert wie ein Platzhalter. Denn wir können nun ein Argument an die Funktion übergeben, das in das Parameter geladen wird.

Wenn wir die Funktion nun aufrufen, muss in der Klammer ein Argument stehen, damit sie ausgeführt werden kann. Dieses Argument wird dann in den Parameter geladen. Beispielsweise:

greeting("Hendrik")
Hier wird das Argument "Hendrik" in den Parameter geladen und "Hallo Hendrik" wird ausgegeben.

WICHTIG Auch Funktionen mit mehreren Parametern sind möglich! Hierbei muss auf die Position der Parameter geachtet werden.

FUNKTIONEN MIT RÜCKGABEWERTEN

Jede Funktion besitzt einen Rückgabewert. Jedes Mal, wenn eine Funktion aufgerufen und komplett abgearbeitet wird, wird in das Hauptprogramm ein Wert zurückgegeben.

Manche Funktionen benötigen keinen Rückgabewert, weil kein Ergebnis durch den Funktionsaufruf erwartet wird. In diesem Fall wird "None" zurückgegeben, was für die Abwesenheit eines Werts steht.
"None" hat den Datentyp NoneType, welcher ganz speziell nur für den Wert "None" gilt.

Möchte man aber bewusst einen Wert zurückgeben, kann man das Schlüsselwort "return" verwenden. Wenn dieses Schlüsselwort aufgerufen wird, wird die Funktion sofort beendet und der Wert nach "return" wird ausgegeben.

OPTIONALE PARAMETER

Parameter kann man in einer Funktion auch optional übergeben.
Ein Beispiel ist hier wieder die range() Funktion. Man muss auf jeden Fall eine Ganzzahl übergeben, die definiert bis zu welcher Zahl ausgegeben werden soll, z.B. range(5).
Optional kann auch ein Startwert mit angeben werden z.B. range(2,5).
Als dritte Möglichkeit kann man noch den Zählerschritt angeben, z.B. range(2,5,3).

Definiert man eine Funktion mit optionalen Parametern aber selbst, muss man den gewünschten Parameter als optional markieren, indem man ihm einen Standardwert zuweist. Dieser wird dann immer von der Funktion verwendet, falls man keinen Parameter übergibt.

WICHTIG Der optionale Parameter muss immer in der Parameterliste stehen, nach den positionsbezogenen Parametern.

SCHLÜSSELWORT PARAMETER

Mit Schlüsselwortparametern können Argumente in einer beliebigen Reihenfolge übergeben werden. Anstatt einen konkreten Wert als Argument zu übergeben, schreibt man vor den Wert den entsprechenden Namen des Parameters, sowie ein „=“.

Zum Beispiel:
greeting(academic_title=“Dr.“, last_name=”Mustermann”, first_name=”Max”)

Man gibt also direkt bei der Übergabe an, in welchen Parameter die Argumente geladen werden sollen.

WICHTIG Man kann positionsbezogene und Schlüsselwortargumente vermischen. Hierbei müssen die positionsbezogenen allerdings am Anfang des Funktionsaufruf stehen.

TUPEL VS. LISTE

Listen sind veränderliche Datenstrukturen, das heißt man kann die Werte einer bestehenden Liste nachträglich verändern.

Tupel sind unveränderliche Datenstrukturen und dabei ähnlich wie Listen. Man definiert sie allerdings mit runden Klammern.
Zum Beispiel:
tupel1= (1,2,3,4)

Ähnlich wie bei Listen, kann man durch eckige Klammern bestimmte Elemente des Tupel ausgeben lassen:
print(tuple1[1])

Durch die Verwendung von Tupeln kann man dem Leser eines Programms verdeutlichen, dass die Werte des Tupels nicht verändert werden sollen.
Definiert man Tupel allerdings nur mit einem Wert, legt Python nur eine normale Variable an. Fügt man nach der Zahl allerdings noch ein Komma hinzu, erkennt Python ein Tupel.
Zum Beispiel
tupel1 =(1,)

Unpackaging von Tupeln

Möchte man aus einem Tupel einzelne Werte rausziehen und diese als einzelne Variablen abspeichern, kann man das mit Hilfe von unpackaging machen.
Zum Beispiel:
first_name, last_name, age = person1

Die erste Variable enthält das erste Element des Tupels, die zweite das zweite und so weiter. Hierbei ist natürlich die Reihenfolge wieder sehr wichtig.

Wieso braucht man Tupel?

Mit Hilfe von Tupeln kann man sich mehrere Rückgabewerte einer Funktion ausgeben lassen. Das funktioniert, indem man alle Rückgabewerte in ein Tupel packt, return() aufruft und das zurückgegebene Tupel entpackt.

BELIEBIGE ANZAHL AN PARAMETER ÜBERGEBEN

Man kann Funktionen so definieren, dass man diesen beliebig viele Parameter übergeben kann. Anstatt alle Parameter einzeln aufzulisten, kann man einen Stern schreiben und dann den beliebigen Bezeichner.
Zum Beispiel:
def multiply(*numbers)

Hierdurch können beliebig viele Werte entgegengenommen werden. Die Parameter, die man an die Funktion übergibt, werden innerhalb der Funktion als Tupel bereitgestellt.

REINE SCHLÜSSELWORT PARAMETER

Es gibt Parameter, in die man nur Werte laden kann, wenn man sie auch wirklich als Schlüsselwort Parameter übergibt.
Ein Beispiel hierfür gibt es in der Funktion print(). Man kann beispielsweise den Parameter "end" verwenden, der bestimmt was am Ende nach einem ausgegebenen String passieren soll. Hierfür muss man den Wert immer als Schlüsselwort Parameter übergeben.
Der end Parameter ist ein reiner Schlüsselwort Parameter, da man nur mit Hilfe eines Schlüsselwortbezogenen Arguments einen Wert reinbekommt.

Ein weiteres Beispiel in der print() Funktion ist "sep", mit dem man einen Separator festlegen kann. Standardmäßig ist das ein Leerzeichen, man kann aber auch ein Komma verwenden.

REINE SCHLÜSSELWORT PARAMETER

Bei Funktionen gibt es lokale und globale Namensbereiche.

Lokaler Namensbereich

Bei der Definition einer Funktion ist alles innerhalb des Funktionskörpers im lokalen Namensbereich der Funktion.
Somit sind alle Variablen und Referenzen, denen innerhalb der Funktion ein Wert zugewiesen wird, nur innerhalb des lokalen Namensbereichs gültig. Man bezeichnet diese Variablen auch als lokale Variablen.

Man kann also nur innerhalb der Funktion auf diese zugreifen. Nachdem der Funktionskörper abgearbeitet wurde, wird der lokale Namensbereich der Funktion zerstört.

Globaler Namensbereich

Der globale Namensbereich wird erzeugt, sobald ein Programm ausgeführt wird. Es ist ein Hauptprogramm, das außerhalb der Funktion festgelegt wird. Der globale Bereich ist während der kompletten Programmlaufzeit gültig. Wenn das Programm komplett durchgelaufen ist, wird der Bereich zerstört.

➔ Im globalen Bereich können wir nicht auf lokale Variablen zugreifen

Das Schlüsselwort Global

Hiermit ist es möglich, Python zu zeigen, dass man keine neue lokale Variable mit dem gleichen Namen wie die globale Variable erzeugt, sondern dass man wirklich die globale Variable meint.
Das macht man, indem man "global" an den Beginn der Variable schreibt.