Python Wiederholung 2

Ergänzungsfach Informatik, 2021/2022, pro@kswe.ch

16. November 2021

1 Funktionen

Wir haben bereits eingebaute Funktionen, wie zum Beispiel `print` oder `type`, eingesetzt. Nun geht es darum, wie man eigene Funktionen erstellen kann.

Erklärung 1.1 (Funktion)**.** *Eine Funktion fasst einen ausführbaren Teil des Quellcodes unter einem Namen zusammen.*

1.1 Funktionsdefinition

Bevor man eine Funktion durch einen Funktionsaufrufen "verwenden" kann, muss man die Funktion definieren. Das folgende Beispiel zeigt eine Funktionsdefinition.

```
1 import random
2
3 def begruessung():
4     print("Hallo!")
5     zahl = random.randint(1, 10)
6     print("Zufallszahl:", zahl)
```

Eine Funktionsdefinition besteht aus einem Funktionskopf und einem Funktionskörper. Zuerst wird der Funktionskopf notiert, dann der Funktionskörper.

1.1.1 Funktionskopf

Der Funktionskopf beginnt immer mit dem Schlüsselwort `def`. Dies ist eine Abkürzung für **definition**. Das Schlüsselwort def hat die Bedeutung, dass nun eine Funktionsdefinition beginnt. Mit einem Leerzeichen folgt der Funktionsname. Es gelten für den Funktionsnamen dieselben Regeln, wie für Variablennamen. Nach dem Funktionsnamen folgen immer runde Klammern. Eine öffnende Klammer (und eine schliessende Klammer). Die Klammern gehören zur Funktionsdefinition, auch wenn diese momentan noch "leer" sind. Danach folgt zwingend ein Doppelpunkt :.

1.1.2 Funktionskörper

Alle Befehle des Funktionskörpers sind unterhalb des Funktionskopfs in einer neuen Zeile eingerückt. Nur die Befehle die eingerückt sind, gehören zum Funktionskörper und somit zur Funktionsdefinition. Diese Befehle werden beim Funktionsaufruf ausgeführt. Es können beliebige Befehle innerhalb des Funktionskörpers notiert werden.

1.2 Funktionsaufruf

Nachdem wir die Funktionsdefinition erläutert haben, schauen wir uns die Ausführung einer Funktion an.

Wichtig!. *Wenn man eine Funktion definiert, dann werden die Befehle der Funktion noch nicht ausgeführt.*

Wir müssen Python erst "sagen", dass die Funktion ausgeführt werden soll. Mit einem Funktionsaufruf können wir das Ausführen einer Funktion bewirken. Wir sagen Python quasi, führe bitte die Befehle der Funktion (genauer die Befehle im Funktionskörper) jetzt aus. Ein Funktionsaufruf erfolgt durch die Notation des Funktionsnamens und der Klammern. Wir kennen den Funktionsaufruf bereits. Wir haben stets Funktionen aufgerufen, welche andere Programmierer erstellt haben (z.B. `print("Hello, World!")`). Nun rufen wir unsere eigenen Funktionen auf! Wir können eine Funktion beliebig oft aufrufen. Egal wann wir eine Funktion aufrufen, es werden immer die Befehle innerhalb des Funktionskörpers exakt einmal pro Funktionsaufruf ausgeführt. Im folgenden Beispiel ist erfolgt in Zeile 10 der Funktionsaufruf.

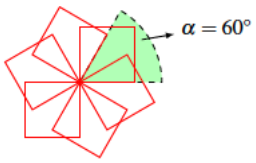
```
1  import random
2
3  def begruessung():
4      print("Hallo!")
5      zahl = random.randint(1, 10)
6      print("Zufallszahl:", zahl)
7
8  begruessung()
```

1.3 Aufgaben

Erstellen Sie in PyCharm einen Ordner `python_wiederholung`. Erstellen Sie darin eine neue Python-Datei `wdh_3.py`. Lösen Sie darin die folgenden Teilaufgaben.

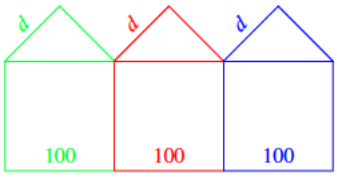
1. Erstellen Sie eine Funktion `quadratzahl`. Berechnen Sie darin die Quadratzahl einer zufälligen Zahl zwischen 1 und 100. Rufen Sie die Funktion dreimal auf.

2. Erstellen Sie eine Funktion `quadrat`. Die Funktion soll mit der Turtle (`import turtle`) ein Quadrat mit der Seitenlänge 100 zeichnen. Zeichnen Sie damit folgende Figur.



3. Innerhalb einer Funktionsdefinition können Sie andere Funktionen aufrufen. Folgendes Beispiel zeigt dieses Konzept. In der Konsole erscheint zuerst `Foo` und dann `Bar`.

```
1  def bar():
2      print("Bar")
3
4  def foo():
5      print("Foo")
6      bar()
7
8  foo()
```



Drei Häuser nebeneinander.
Für die Dachseite gilt: $d = \frac{100 \cdot \sqrt{2}}{2}$.

Erstellen Sie ein Programm mit drei Funktionen: `geschoss`, `dach` und `haus`. Die Funktion `haus` soll mithilfe von `geschoss` und `dach` gezeichnet werden. Rufen Sie dann `haus` dreimal auf, damit die drei Häuser aus der Abbildung von oben gezeichnet werden.

1.4 Parameter und Argumente

Wir können in der Funktionsdefinition zwischen den Klammern einen oder mehrere Parameter definieren.

1.4.1 Parameter

Ein Parameter ist eine lokale Variable der Funktion. Man definiert einen Parameter bei der Funktionsdefinition zwischen den runden Klammern. Da es eine lokale Variable darstellt, muss wie bei einer Variable ein gültiger Name verwendet werden. Mehrere Parameter werden durch Komma getrennt. Bei mehreren Parametern muss jeder eindeutig sein. Das folgende Beispiel zeigt eine Funktionsdefinition mit einem Parameter.

```
1 def begruessung(name):
2     print("Hallo,", name, "!")
```

1.4.2 Argumente

Beim Funktionsaufruf muss für jeden Parameter ein Argument vorhanden sein. Man muss jedem Parameter einen Wert übergeben. Beispiel:

```
1 def begruessung(name):
2     print("Hallo,", name, "!")
3
4 begruessung("Katja")
5 begruessung("Bernd")
```

Erklärung 1.2 (Parameter und Argument)**.** *Ein Parameter ist die Variable, die in der Funktionsdefinition in Klammern aufgeführt ist. Ein Argument ist der Wert, der an die Funktion übergeben wird, wenn sie aufgerufen wird. Innerhalb der Funktion speichert der Parameter den Übergabewert.*

Im Beispiel speichert der Parameter `name` den Wert `"Katja"` beim Funktionsaufruf in Zeile 4. Beim Funktionsaufruf in Zeile 5 wird der Wert `"Bernd"` gespeichert.

Wichtig!. *Besitzt eine Funktionsdefinition einen Parameter, dann muss beim Funktionsaufruf auch ein Argument notiert werden. Stimmt die Anzahl der Parameter mit der Anzahl der Argumente nicht überein, wird ein Error generiert und das Programm stürzt ab.*

Bei mehreren Parametern muss beim Funktionsaufruf die Reihenfolge der Argumente mit der Reihenfolge der Parameter übereinstimmen.

```
1 def begruessung(name, alter):
2     print("Hallo,", name, "! Du bist", alter, "Jahre alt.")
3
4 begruessung("Katja", 32)
```

Tauscht man die Argumente beim Funktionsaufruf `begruessung(32, "Katja")`, dann wird `Hallo, 32 ! Du bist Katja Jahre alt.` ausgegeben. Nicht immer ist eine korrekte Ausführung gewährleistet. Es kann auch vorkommen, dass bei einer falschen Reihenfolge das Programm abstürzt.

1.4.3 Keyword Arguments

Typischerweise gibt man bei einem Funktionsaufruf die Argumente in der korrekten Reihenfolge an. Die Zuordnung von Argument und Parameter erfolgt über die Position beim Funktionsaufruf. Man nennt diese Art von Argumente deshalb auch **Positional Arguments**. Alle bisherigen Beispiele verwenden Positional Arguments. Möchte man die Argumente unabhängig von der Position korrekt übergeben, dann kann man beim Funktionsaufruf den Parameternamen verwenden. Folgendes Beispiel zeigt die Verwendung. Neben den Argumenten müssen auch die Parameternamen mit einem Gleichheitszeichen notiert werden. Gibt man Argumente mit dieser Variante beim Funktionsaufruf an, dann spricht man von **Keyword Arguments**.

```
1 def begruessung(name, alter):
2     print("Hallo,", name, "! Du bist", alter, "Jahre alt.")
3
4 begruessung(alter=32, name="Katja")
```

Man kann **Positional Arguments** und **Keyword Arguments** kombinieren. Man **muss** jedoch zuerst die Positional Arguments angeben und dann die Keyword Arguments. Beispiel:

```
1 def begruessung(name, alter):
2     print("Hallo,", name, "! Du bist", alter, "Jahre alt.")
3
4 begruessung("Katja", alter=32)
```

1.4.4 Standardwerte für Parameter

Bei der Funktionsdefinition kann man für jeden Parameter einen Standardwert (eng. default value) angeben. Dieser wird benutzt, falls beim Funktionsaufruf kein Argument für diesen Parameter vorhanden ist. Dadurch kann man **optionale** Parameter realisieren.

```
1 def begruessung(name, alter=42):
2     print("Hallo,", name, "! Du bist", alter, "Jahre alt.")
3
4 begruessung("Katja")
```

Soll ein Parameter optional sein und keinen Wert besitzen, dann kann man das Schlüsselwort `None` bei der Funktionsdefinition für diesen Parameter verwenden. Optionale Parameter erfordern meist zusätzlichen Code, damit das Programm auch bei einem fehlenden Argument sinnvoll arbeitet.

```
1 def begruessung(name, alter=None):
2     if alter is None:
3         print("Hallo,", name, "!")
4     else:
5         print("Hallo,", name, "! Du bist", alter, "Jahre alt.")
6
7 begruessung("Katja")
```

1.5 Aufgaben

Überarbeiten Sie die vorherigen Aufgaben.

1. Die Funktion `quadratzahl` soll nicht für eine zufällige Zahl die Quadratzahl berechnen, sondern einen Parameter verwenden. Für den Parameter soll die Quadratzahl berechnet werden. Wie können Sie nun für eine zufällige Zahl die Quadratzahl berechnen?

2. Die `quadrat`-Funktion soll zwei Parameter besitzen. Die Seitenlänge und die Farbe der Quadrate soll beim Funktionsaufruf durch zwei Argumente bestimmt werden. Rufen Sie die Funktion mit Keyword Arguments auf.

3. Bauen Sie die drei Funktionen `geschoss`, `dach` und `haus` so um, dass die Seitenlänge ein Parameter mit dem Standardwert 100 ist und die Farbe der drei Häuser ebenfalls als Parameter übergeben werden kann. Der Parameter soll eine Liste sein und den Standardwert `["green", "red", "blue"]` besitzen.

4. Informieren Sie sich über Rekursion. Programmieren Sie dann eine Funktion `fakultaet(x)`, welche die Fakultät von x berechnet. Beispiel: Falls `fakultaet(5)` muss $5 \cdot 4 \cdot 3 \cdot 2 \cdot 1$ berechnet werden.