Strings formatieren

May 7, 2020

0.1 Strings formatieren

Manchmal möchtest du in einen String irgendwelche Werte einsetzen. Also z. B. möchtest du die Ausgabe erzeugen, "Ich habe 5 Hunde". Bisher war das immer recht unhandlich, mit der str()-Funktion:

[1]:
```
n = 5
print("Ich habe " + str(n) + " Hunde")
```

```
Ich habe 5 Hunde
```

In dieser Lektion lernst du eine neue, praktische Methode kennen, um eine solche Ausgabe zu erzeugen!

0.1.1 Schauen wir uns dazu mal ein Beispiel an

Stell dir vor, du möchtest deine Anwendung übersetzen. Spätestens dann stößt du auf Probleme:

[2]:
```
n = 5
print("Ich habe " + str(n) + " Hunde")
print("I got " + str(n) + " dogs")
```

```
Ich habe 5 Hunde
I got 5 dogs
```

Wie bekommst du es jetzt hin, dass der Sprachbaustein austauschbar ist und nicht mit dem `+ str(n) +`Befehl "verwoben" ist?

Idee: Du könntest die Sprachbausteine in einem Dictionary definieren und einen Platzhalter per `.replace()` einsetzen:

[3]:
```
translations = {
    "number_of_dogs": "Ich habe XXX Hunde"
}
print(translations["number_of_dogs"].replace("XXX", str(n)))
```

```
Ich habe 5 Hunde
```

Problem: Das wird aber schnell unübersichtlich.

Lösung: Glücklicherweise stellt uns Python hier eine `.format()`-Methode zur Verfügung, die das ganze sehr viel einfacher macht. Hierbei verwenden wir `{0}` für die Position, wo wir den Parameter

`n` des `.format(n)`-Aufrufs einsetzen möchten:

[4]:
```
print("Ich habe {0} Hunde".format(n))
```

```
Ich habe 5 Hunde
```

Ergebnis: Unserer Übersetzungs-Code ist sehr viel angenehmer:

[5]:
```
translations = {
    "number_of_dogs": "Ich habe {0} Hunde"
}
print(translations["number_of_dogs"].format(n))
```

```
Ich habe 5 Hunde
```

Diese `.format()`-Methode funktioniert auch mit mehreren Parametern. Hierbei definiert dann `{0}` die Position für den ersten Parameter, `{1}` die Position, wo der 2. Parameter hin gesetzt werden soll.

Hier in folgendem Fall wird also die `{1}` durch `"Katzen"` ersetzt, und `{0}` durch die Zahl 5:

[6]:
```
print("Ich habe {1} {0}x".format(5, "Katzen"))
```

```
Ich habe Katzen 5x
```

Kommazahlen und runden Der `format()`-Befehl erlaubt es, Kommazahlen komfortabel zu runden. Hierbei wird an einen Formatierungs-Befehl (das war z. B. `{0}` oder `{1}`) noch innerhalb der geschweiften Klammern ein `f` drangehängt.

Dadurch sagen wir Python, dass diese Zahl als Kommazahl betrachtet werden soll. Entsprechend wird hier jetzt die Zahl 5 eingesetzt, die `.000000` werden aber ergänzt, weil es als Kommazahl ausgegeben wird:

[9]:
```
print("So viele Katzen habe ich: {0:f}".format(5))
```

```
So viele Katzen habe ich: 5.000000
```

Wenn wir die Anzahl der Stellen beschränken möchten, können wir das tun, indem wir nach dem Doppelpunkt `.2` schreiben, um z. B. die Kommazahl auf 2 Nachkommastellen zu limitieren:

[10]:
```
print("So viele Katzen habe ich: {0:.2f}".format(5))
```

```
So viele Katzen habe ich: 5.00
```

Das können wir jetzt auch für echte Kommazahlen nutzen, um diese zu runden! :-)

[44]:
```
print("Pi hat den Wert: {0:.3f}".format(3.141529))
```

```
Pi hat den Wert: 3.142
```

0.1.2 Parameter bennennen

Wenn du einen String hast, in dem viele Platzhalter eingesetzt werden, wird die Schreibweise `{0}`, `{1}`, `{2}`, `{3}`, `{4}`, `{5}`, … irgendwann unübersichtlich.

Glücklicherweise können wir die Parameter auch benennen. Hierbei ist wichtig, dass, wenn auf der linken Seiten ein Parameter z. B. `{animal}` heißt, der Parameter `animal` dann entsprechend auch der `.format()`-Funktion übergeben wird.

Hier in dem Fall ist es also so, dass an die Stelle, an der `{animal}` steht, der Wert vom `.format()`-Aufruf `animal = "Hunde"` (konkret das Wort "Hunde") eingesetzt wird:

[47]:
```
print("Ich habe {number:.3f} {animal}".format(number = 5, animal = "Hunde"))
```

```
Ich habe 5.000 Hunde
```

[ ]:
```

```