# Strings

October 15, 2017

## 1 Strings

Wir können in Python nicht nur mit Zahlen, sondern auch mit Zeichenketten, sogenannten **Strings**, arbeiten. Darunter kannst du dir jegliche Abfolgen von Zeichen vorstellen, die in Anführungszeichen stehen. Zum Beispiel ist *"Hallo"* ein String und *"Hallo Welt"* ebenso, aber auch *"123.2"* oder *"!Achtung!"*

### 1.0.1 Allgemeines

Auch Strings können wir mit dem print()-Befehl ausgeben.

```
In [4]: print("Hallo Welt")

Hallo Welt
```

Du kannst Strings wie Zahlen in Variablen speichern.

```
In [3]: name = "Max"

In [4]: print(name)

Max
```

### 1.0.2 Strings zusammenfügen

Du kannst zwei oder mehr Zeichenketten auch mittels **+** zusammenfügen.

```
In [5]: print("Ich bin: " + "Max")

Ich bin: Max
```

```
In [7]: print("Ich bin: " + name + ". Und wer bist du?")

Ich bin: Max. Und wer bist du?
```

Allerdings kommt es zu einer Fehlermeldung, wenn du versuchst, Zahlen und Strings zu addieren.

```
In [19]: print("Ich bin: " + 4)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-19-9794e8874ec9> in <module>()
----> 1 print("Ich bin: " + 4)

TypeError: must be str, not int
```

### 1.0.3 Eine Zahl in einen String umwandeln

Du kannst diese Fehler korrigieren, indem du die Zahl in einen String umwandelst. Dazu hast du zwei Möglichkeiten.

1.) Du setzt Anführungszeichen um die Zahl und machst sie so zu einem String:

```
In [20]: print("Ich bin: " + "4")
```

```
Ich bin: 4
```

2.) Du wandelst die Zahl mit dem **str()**-Befehl in einen String um:

```
In [22]: age = 22
         print("Ich bin: " + str(age))
```

```
Ich bin: 22
```

**Beachte, dass du mit "4" oder str(age) nicht mehr rechnen kannst!**

### 1.0.4 Spiel doch jetzt ein wenig herum mit dem was du gelernt hast:

- gib einige mittels + zusammengesetzte Strings per print() aus :-)

```
In [ ]:
```

# Merkblatt - Strings

September 23, 2017

## 0.1 String - Methoden

In diesem Dokument möchte ich dir einen kurzen Überblick über wichtige String - Methoden geben.

Mit der `.upper()` bzw. `.lower()` - Methode kannst du dafür sorgen, dass alle Zeichen in GroSS- bzw. Kleinbuchstaben angezeigt werden.

```
In [6]: w = "Hallo"
        print(w.upper())
        print("Hallo".upper())
```

```
HALLO
HALLO
```

```
In [7]: w = "Hallo"
        print(w.lower())
        print("Hallo".lower())
```

```
hallo
hallo
```

Mit der `.startswith()` bzw. `.endswith()` - Methode kannst du prüfen, ob ein String mit einem anderen String beginnt / aufhört:

```
In [12]: sentence = "Ist das Wetter heute gut???"

         if sentence.endswith("???"):
             print("Der Satz endet mit drei Fragezeichen")

         if sentence.startswith("Ist"):
             print("Der Satz beginnt mit einem 'ist'")
```

```
Der Satz endet mit drei Fragezeichen
Der Satz beginnt mit einem 'ist'
```

### 0.1.1 Die `.strip()` - Methode

StandardmäSSig entfernt die `.strip()` - Methode Leerzeichen vom Anfang und vom Ende des Strings.

```
In [1]: "   Hallo Welt.    ".strip()

Out[1]: 'Hallo Welt.'
```

Du kannst der `strip()` - Methode aber auch als Parameter übergeben, welche Zeichen entfernt werden sollen. Hier in den nächsten Beispielen sagen wir z.B., dass nur Unterstrichte und Punkte entfernt werden sollen.

Die `.lstrip()` bzw. `.rstrip()` - Methode funktioniert analog der `.strip()` - Methode, wobei aber `.lstrip()` nur die linke Seite und `.rstrip()` nur die rechte Seite betrachtet:

```
In [23]: word = "____Hallo.__"
         print(word.strip("_."))
         print(word.lstrip("_"))
         print(word.rstrip("_"))

         sentence = "Ist das Wetter heute, und morgen gut???"
         print(sentence.rstrip("!?.,"))

Hallo
Hallo.__
____Hallo.
Ist das Wetter heute, und morgen gut
```

### 0.1.2 Die `.find()` - Methode

Mit der `.find()` - Methode kannst du herausfinden, an welcher Stelle ein Zeichen in einem String vorkommt. Beispielsweise können wir so herausfinden, dass das Komma an der 21. Stelle (Position 20 also) vorkommt.

Wenn die `.find()` - Methode die Zahl `-1` zurückgibt, bedeutet dass, dass das Zeichen im String nicht vorkommt.

```
In [26]: sentence = "Ist das Wetter heute, und morgen gut???"
         print(sentence.find(","))
         print(sentence.find("!"))

20
-1
```

### 0.1.3 Zeichen ersetzen (`.replace()`)

Mit der `.replace()` - Methode kannst du eine Ersetzung durchführen. Beispielsweise kannst du so z.B. das Komma durch ein Semikolon ersetzen lassen, etc.

In [29]:
```
sentence = "Ist das Wetter heute, und morgen gut???"

        print(sentence.replace(",", ";"))
        print(sentence.replace("u", "ü"))
        print(sentence.replace("und", "oder"))
```

```
Ist das Wetter heute; und morgen gut???
Ist das Wetter heüte, ünd morgen güt???
Ist das Wetter heute, oder morgen gut???
```

In [ ]:

# Strings formatieren

September 23, 2017

## 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:

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

```
Ich habe 5 Hunde
```

In dieser Lektion lernst du eine neue, praktsiche 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:

```
In [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:

```
In [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:

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

```
Ich habe 5 Hunde
```

**Ergebnis:** Unserer Übersetzungs-Code ist sehr viel angenehmer:

```
In [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, an die Stelle, 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.

```
In [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:

```
In [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 schreiben `.2`, um z.B. die Kommazahl auf 2 Nachkommastellen zu limitieren:

```
In [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! :)

```
In [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 den 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, dass der Parameter `animal` dann entsprechend auch der `.format()` - Funktion übergeben wird.

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

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

```
Ich habe 5.000 Hunde
```

```
In [ ]:
```

### 1.5.3 re

Ermöglicht mit reguläre Ausdrücken sehr flexibel Strings zu durchsuchen.

#### Modul einbinden

```
In [33]: import re
```

#### Anwendung

```
In [34]: sentence = "Habe 30 Hunde, die jeweils 4 Liter Wasser brauchen und 2 kg Nahrung."
         re.findall("[0-9]+", sentence)
```

```
Out[34]: ['30', '4', '2']
```

**Weitere Infos:** https://docs.python.org/3.6/library/re.html