strings

string literale

Normale Stringliterale

Zwischen Hochkommas oder doppelten Hochkommas. Wahl ist "beliebig"...

- **Aber**: ich persönlich würde empfehlen, für Texte, welche dem Benutzer angezeigt werden und potentiell übersetzt werden könnten, doppelte Hochkommas zu verwenden.
- Einfache Hochkommas sind eher für Schlüssel, feststehende Strings für Bibliotheken

Zusammengesetze Strings (string concatenation)

[f-strings (https://realpython.com/python-f-strings/)](https://realpython.com/python-f-strings/) sind zu bevorzugen.

In [1]:
```
name = "Gion Antoni Derungs"
print("Guten Tag " + name)
print("Guten Tag {0} ".format(name))
print(f"Guten Tag {name}")
```

```
Guten Tag Gion Antoni Derungs
Guten Tag Gion Antoni Derungs
Guten Tag Gion Antoni Derungs
```

Bei string-Formatierungen is es jeweils auch möglich, weitere Formatierungsangaben mitzugeben. Dies ist meisten bei Fliesskommazahlen relevant.

Bei floats ist der Syntax `<value>:<width>.<precision>f`

In [2]:
```
val = 1.0 / 3.01

# Two digits after the dot, no minimal width specified
print(f"Value: {val:.2f}")
print("Value: {0:.2f}".format(val*100))

#Three digits after the dot, minimal width will be 7 chars
print(f"Value: {val:7.3f}")
print("Value: {0:7.3f}".format(val * 100))
print("      |1234567|")
```

```
Value: 0.33
Value: 33.22
Value:   0.332
Value:  33.223
      |1234567|
```

In diesen Arten von Strings müssen Zeilenumbrüche explizit angegeben werden. \n wird als De-

Facto-Standart auch auf Windows-Plattformen verwendet

In [3]:
```
multiline_string = "Guten Tag\nSie haben in der Lotterie gewonnen."
print(multiline_string)
```

```
Guten Tag
Sie haben in der Lotterie gewonnen.
```

docstrings

Alles in drei Hochkommas wird auch docstrings genannt. Zeilenumbrüche werden so übernommen, wie sie im Quellcode vorkommen.

Sie werden auch verwendet, um z. B. Funktionen zu dokumentieren, da sie im Gegensatz zu Kommentaren vom Interpreter **nicht** entfernt werden.

In [4]:
```
my_doc_string = """Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt
ut labore et dolore magna aliqua."""
print(my_doc_string)
```

```
Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt
ut labore et dolore magna aliqua.
```

In [5]:
```
def max(a, b):
    """ This function return the bigger value of a and b. """
    if a>b:
        return a
    return b

print(max.__doc__)
```

```
 This function return the bigger value of a and b.
```

String Methoden

Abfragen

len liefert die Anzahl Zeichen (characters)

In [6]:
```
my_string = "Hallo Welt😊"
print(f"my_string ist {len(my_string)} Zeichen lang")
```

```
my_string ist 11 Zeichen lang
```

Der **Index-Operator** wird verwendet, um ein Zeichen abzufragen

In [7]:
```
print(f"Das 11te Zeichen in my_string ist '{my_string[10]}'")
```

```
Das 11te Zeichen in my_string ist '😊'
```

Slicing wird verwendet, um einen Teilstring zu erhalten

substring = my_string[6:10]

6: Erstes Zeichen, null-basiert, inklusive

10: Letztes Zeichen, null basiert, exclusive

In [8]:
```
substring = my_string[6:10]
print(f"Zeichen 7 bis 10 in substring sind '{substring}'")
```

```
Zeichen 7 bis 10 in substring sind 'Welt'
```

von oder *bis* kann auch weggelassen werden:

start = my_string[:5]

end = my_string[6:]

In [9]:
```
print(f"first 5 chars of my_string are '{my_string[:5]}'")
print(f"my_string from position 7 on is '{my_string[6:]}'")
```

```
first 5 chars of my_string are 'Hallo'
my_string from position 7 on is 'Welt😊'
```

in kann verwendet werden, um zu prüfen, ob ein Teilstring vorkommt.

In [10]:
```
tests = ['Welt', 'e', '😊', 'Nix']

for test in tests:
    is_contained = test in my_string
    print(f"'{test}' kommt in my_doc_string vor: {is_contained}")
```

```
'Welt' kommt in my_doc_string vor: True
'e' kommt in my_doc_string vor: True
'😊' kommt in my_doc_string vor: True
'Nix' kommt in my_doc_string vor: False
```

split und **join** teilt einen string auf resp. setzt ihn zusammen.

In [11]:
```
sentence = "Ein kurzer Satz"
words = sentence.split(' ') # splits the string along the spaces
print(f"words are {words} which is of type {type(words)}")
new_sentence = '_'.join(words) # combine the words again with an underscore be
print(new_sentence)
```

```
words are ['Ein', 'kurzer', 'Satz'] which is of type <class 'list'>
Ein_kurzer_Satz
```

Diverse Methoden

Name	Beschreibung
`upper()`	Alles in Grossbuchstaben
`lower()`	Alles in Kleinbuchstaben
`replace(old, new)`	Ersetzt alle vorkommen von *old* mit *new*
`ljust(len)`	füllt links Zeichen (in der Regel Leerzeichen) ein, bis der String *len** lang ist.
`rjust(len)`	füllt rechts Zeichen (in der Regel Leerzeichen) ein, bis der String *len** lang ist.
`strip(), lstrip(), rstrip()`	Entfernt Zeichen (in der Regel Leerzeichen) am Anfang, am Ende oder Anfang und am Ende.
`startswith(start)`	testet, ob der string mit *start" anfängt

Mehr Methoden sind in [w3schools (https://www.w3schools.com/python/python_strings_methods.asp)](https://www.w3schools.com/python/python_strings_methods.asp) zu finden.

In [12]:
```
test = "   un geputzt\t\t   \n\r"
print(f"|{test.strip()}|")
stripped = test.strip()
stripped = stripped.replace(' ', '')
print(stripped)
```

```
|un geputzt|
ungeputzt
```

Escape Characters

Einige Zeichen können so direkt nicht verwendet werden, z.B. das Hochkomma, Tabulator, Zeilenumbrüche, etc. In diesen Fällen muss der backslash verwendet werden.

In [13]:
```
print("1\" sind 1/12 Fuss")
```

```
1" sind 1/12 Fuss
```

Weitere Escape-Sequenzen

Code	Beschreibung

Code	Beschreibung
\'	Hochkomma
\"	Doppeltes Hochkomma
\n	Zeilenumbruch
\t	Tabulator
\xHH	Hex-Wert
\xXXXX	Unicode Codepoint (16Bit)

In [14]:
```
print("Beispiele für Escape-Sequenzen:\n\t- Das Copyright Zeichen \xAE ist Cod
```

```
Beispiele für Escape-Sequenzen:
        - Das Copyright Zeichen ® ist Codepunkt 0xAE
        - Omega (Ω) ist u03A9
        - Das Emoticon 😀 ist 0x1F600
```

Regexps

Reguläre Ausdrücke werden verwendet, um String-Muster zu finden. Ein beliebtes Beispiel ist das Muster einer E-Mail-Adresse. Wir werden hier nicht sehr tief in die Welt der Regexps eintauchen, aber Sie sollten wissen, wie sie ein Python eingesetzt werden können.

Um Rexexps zu erstellen, testen und verstehen empfehle ich [regex101.com (https://www.regex101.com)](https://www.regex101.com).

In [15]:
```
import re

tests = ['mail@to.me', 'http://www.ecosia.com', 'Gion Antoni Derungs', 'mötele

regex_only_latin_lowercase_chars = r'([a-z._-]+)@[a-z._-]+'
regex_word_chars = r'([\w._-]+)@[\w._-]+'
regex = regex_only_latin_lowercase_chars
for test in tests:
    match = re.search(regex, test)
    if match:
        print(f"'{test}' does match. Group 1 is '{match.group(1)}'.")
    else:
        print(f"'{test}' does not match.")
```

```
'mail@to.me' does match. Group 1 is 'mail'.
'http://www.ecosia.com' does not match.
'Gion Antoni Derungs' does not match.
'möteleÿ.crew@yahoo.com' does match. Group 1 is '.crew'.
```