Empfohlene Unterrichtsinhalte als Vorbereitung für dieses Thema: 02-01 (Strings), 05-03 (Module), 09-01 (Regular Expressions schreiben)

Einführung in die computerlinguistische Programmierung mit Python

09-02: Reguläre Ausdrücke in Python verarbeiten 🤓

Mit regulären Ausdrücken können wir Strings beschreiben und auf bestimmte selbstdefinierte Eigenschaften prüfen. Im nächsten Schritt setzen wir diese Werkzeuge mit Pythoncode um.

`re.search()`: Strings matchen mit dem `re`-Modul

Für die Arbeit mit regulären Ausdrücken steht uns das Modul `re` zur Verfügung. Wir können es importieren und dann alle Funktionen verwenden, die im Modul definiert sind. Die offizielle Dokumentation zum Modul ist hier zu finden:
https://docs.python.org/3.8/library/re.html#module-contents

Um in Python einen String mit einem regulären Ausdruck zu prüfen, brauchen wir die zwei Funktionen `re.compile()` und `re.search()`.

Mit `re.compile()` **definieren** wir einen regulären Ausdruck. Dabei fügen wir als Argument einen String ein, der den gesamten regulären Ausdruck enthält, den wir entworfen haben. Das Ergebnis dieser Funktion ist ein **Objekt vom Typ regulärer Ausdruck**; wir können diesen Wert in einer Variable speichern, um später damit weiter zu arbeiten.

In [ ]:
```
# RegEx-Modul importieren:
import re

# RegEx erzeugen:
my_regex = re.compile("^Man(go|ko|ga)$")
print(type(my_regex))
```

Ab jetzt können wir verschiedene Strings daraufhin **prüfen**, ob sie die definierten Bedingungen erfüllen oder nicht. Dazu verwenden wir `re.search()`. Das **erste Argument** ist der reguläre Ausdruck, den wir im vorigen Schritt definiert haben. Das **zweite Argument** ist der String, den wir mit dem regulären Ausdruck prüfen wollen.

In [ ]:
```
# RegEx-Modul importieren:
import re

# RegEx erzeugen:
my_regex = re.compile("^Man(go|ko|ga)$")

# Strings anhand der RegEx prüfen:
print(re.search(my_regex, "Mango"))
print(re.search(my_regex, "Manga"))
print(re.search(my_regex, "Manko"))
print(re.search(my_regex, "Mangold"))
```

Beim Prüfen von Strings mit regulären Ausdrücken bekommen wir **als Rückgabe ein Objekt vom Typ** `re.Match` - es sei denn, der String passt nicht zum regulären Ausdruck. Dann erhalten wir als Rückgabewert `None`.

Wenn wir einen Match gefunden haben, enthält die Rückgabe die folgenden Informationen:

- `span=(0, 5)`: Von Index 0 bis Index 5 des geprüften Strings haben wir einen Match für den regulären Ausdruck gefunden.
- `match='Mango'`: Der Match wurde für den Teilstring "Mango" gefunden.

Die Information zur Spanne ist interessant, weil reguläre Ausdrücke auf **Strings unterschiedlicher Länge** matchen können. Zum Beispiel, wenn wir optionale Sequenzen mit dem Quantifizierer `*` oder `?` definieren:

In [ ]:
```
# RegEx-Modul importieren:
import re

# RegEx erzeugen:
my_regex = re.compile("^Mango(ld)?$")

# Strings anhand der RegEx prüfen:
print(re.search(my_regex, "Mango"))
print(re.search(my_regex, "Manga"))
print(re.search(my_regex, "Manko"))
print(re.search(my_regex, "Mangold"))
```

`re.sub()`: Reguläre Ausdrücke zum Ersetzen von Teilstrings verwenden

Wenn ein Match gefunden wird, können wir einen neuen String erzeugen, in dem genau der gematchte Teilstring durch etwas anderes **ersetzt** wird.

Im folgenden Beispiel geht es um Kontonummern, die anonymisiert werden sollen. Wir könnten alle Zahlen durch ein Platzhalterzeichen ersetzen:

In [ ]:
```
import re

secret = 'Kontonummer: 108010222202'

# Hier definieren wir das Muster für Zahlen:
zahlen = re.compile('[0-9]') # \d

# re.sub ersetzt Teilstrings durch beliebige andere Zeichensequenzen:
print(re.sub(zahlen, 'X', secret))

print(secret)
```

Die Funktion `re.sub(pattern, replacement, string)` sucht im übergebenen `string` nach dem übergebenen `pattern` und ersetzt jede Stelle im String, die dem Muster entspricht, durch das gewünschte `replacement`. Der Rückgabewert ist vom Typ String.

Da Strings in Python **immutable** sind, verändern wir nicht den Wert der Originalvariable `secret`. Stattdessen wird bei `re.sub()` ein neuer String erzeugt, der teilweise dem alten String entspricht und teilweise Änderungen enthält.

`re.split()`: Strings mit regulären Ausdrücken zerteilen

Wir können reguläre Ausdrücke auch verwenden, um Strings zu **zerlegen**. Wir kennen bereits die Funktion `string.split(sep)`, bei der der gegebene String an allen Vorkommen des Substrings `sep` aufgeteilt wird. Das Ergebnis von `split()` ist eine Liste (vgl. Thema 02-01, Strings).

Fast analog dazu funktioniert die Funktion `re.split(pattern, string)`. Das übergebene `pattern` beschreibt in der Syntax regulärer Ausdrücke den Separator, der jetzt nicht mehr exakt bekannt sein muss, sondern mithilfe von **Bedingungen** beschrieben werden kann.

Im folgenden Code wird ein Text an allen Satzendzeichen aufgeteilt. Das Ziel ist es, eine Liste aller einzelnen Sätze zu erhalten.

In [ ]:
```
import re

# Der Text enthält mehrere Sätze, die durch Satzendzeichen
# voneinander getrennt sind
heine_brief = """Sehr liebenswürdige und charmante Person! Ich bedauere sehr, daß ic

# Das Muster soll auf alle Satzendzeichen matchen.
satzende = re.compile('[!\.?]\s-?\s*')

brief_saetze = re.split(satzende, heine_brief)
for satz in brief_saetze:
    print(satz)
    print("+++")
```

Wie man sieht, werden die Sätze erfolgreich separiert, aber die Satzendzeichen gehen dabei verloren. Wie man so lange Texte besser mit regulären Ausdrücken verarbeiten kann, besprechen wir in der nächsten Themeneinheit, 09-03.

Zusammenfassung

- Das Modul `re` stellt alle Funktionen zur Verfügung, die wir brauchen, um in Python mit regulären Ausdrücken zu arbeiten.
- Mit `re.compile()` erzeugen wir einen regulären Ausdruck, mit dem wir dann arbeiten können.
- Mit `re.search()` prüfen wir, ob ein String die Bedingungen eines definierten regulären Ausdrucks erfüllt.
- Falls der String auf das Muster *matcht*, erhalten wir als Rückgabe unter anderem die Information, an welchen Positionen der Match gefunden wurde und welcher Teilstring dort zu finden ist.
- Mit `re.sub()` ersetzen wir gematchte Teilstrings durch beliebige andere Strings.
- Mit `re.split()` zerteilen wir einen String an allen Stellen, an denen ein Match für den angegebenen regulären Ausdruck gefunden wurde.

Weitere Themen dieser Woche:

- 09-01: Regular Expressions schreiben
- 09-03: Regular Expressions und Gruppen