Empfohlene Unterrichtsinhalte als Vorbereitung für dieses Thema: 02-03, 04-01 (Listen, Dictionaries)

Einführung in die computerlinguistische Programmierung mit Python

08-01: Sets 🛍️

Und noch ein veränderlicher Datentyp

Mengen kennen Sie schon aus der Vorlesung "Mathematische Grundlagen der Computerlinguistik". Mengen (englisch: Sets) ähneln in Python Listen, haben aber einige Eigenschaften, die sie von Listen unterscheiden:

- Sets sind unsortiert
- Sets enthalten jedes Item nur einmal

Genau wie Listen sind Sets ebenfalls **mutable**, das bedeutet, sie können "in place" verändert werden. Durch die Art, wie Sets im Arbeitsspeicher angelegt werden, kann mit ihnen effizienter gearbeitet werden als mit Listen. *Wir verwenden also immer dann Sets, wenn die Sortierung und die Anzahl der Vorkommen einzelner Elemente keine Rolle spielen*.

In [1]:
```
set1 = {1,2,3,3,3} # Nur wohlunterschiedene Elemente
print(set1)

print("-----")

set1_anders = set1 # set1_anders zeigt auf den selben Wert wie set1
set1_anders.add(4) # Wir fügen dem Set etwas hinzu
print(set1)
print(set1_anders)
```

Da Sets unsortiert sind, gibt es auch keinen Index mit dem man auf einzelne Elemente zugreifen kann. Slicing ist auf Mengen deshalb ebenfalls nicht möglich.

In [1]:
```
set1 = {1,2,3,3,3}

print(set1[1])
print(set1[0:2])
```

Das Fehlen der Indexinformationen bei Mengen führt u.a. dazu, dass sie vom Python-Interpreter effizienter verarbeitet werden können als Listen.

Erzeugen von Sets

Sets auf verschiedene Weise erzeugt werden:

In [1]:
```
set1 = set()  # leere Menge erstellen...

for i in [1,2,3,3,4,4,5]:
    print(i)
    set1.add(i)    # ... und nach und nach füllen

print("Set 1: " + str(set1))
```

In [1]:
```
set2 = set([1,2,3,3,4,4,5])  # Menge direkt beim Erstellen mit Elementen fu

print("Set 2: " + str(set2))
```

In [1]:
```
set3 = {1,2,3,3,4,4,5}    # Keine Verwechslungsgefahr mit Dictionarys:
                          # Sets haben keine Values, nur Keys
print("Set 3: " + str(set3))
```

Der Unterschied zwischen `set2` und `set3` ist, dass `set2` mit Hilfe eines Funktionsaufrufs erzeugt wurde. Wir erkennen Funktionen inzwischen an den runden Klammern: `set()`, analog zu den schon bekannten Funktionen `str()` oder `bool()`. Die dritte Menge `set3` dagegen wurde direkt mithilfe einer bestimmten Art von Klammern als Menge erstellt; analog zu den Datentypen, die wir schon länger kennen, z.B. Listen (dort verwenden wir eckige Klammern, hier sind es geschweifte Klammern).

Wie immer sollten wir auch beim Erstellen von Mengen sollten wir immer darauf achten, dass unser Code auch das tut was er soll:

In [1]:
```
set4 = set("This is the boss, and I'm sick of waiting")
print(set4)
```

Hier wird durch die `set()` -Funktion das String-Argument in einzelne Zeichen aufgesplittet; diese werden dann in die Menge aufgenommen.

In [1]:
```
set5 = {}
print(type(set5))
```

Wir können mit Hilfe der geschweiften Klammern keine leere Menge erstellen. Leere geschweifte Klammern `{}` werden vom Pythoninterpreter immer als Dictionary interpretiert.

In [1]:
```
set6 = set("This is the boss, and I'm sick of waiting".split())
print(set6)
```

Operationen auf Mengen

Operation	**Bedeutung**
`len(s)`	Anzahl der Elemente in `s`
`s.add(e)`	Füge dem Set `s` das Element `e` hinzu
`s1.update(s2)`	Ergänze s1 um alle Elemente aus s2
`s1.intersection(s2)` oder `s1 & s2`	Erzeuge die Schnittmenge von `s1` und `s2`
`s1.union(s2)` oder `s1 | s2`	Erzeuge die Vereinigungsmenge von `s1` und `s2`
`s1.difference(s2)` oder `s1 - s2`	Erzeuge die Differenzmenge von `s1` und `s2`
`s1.issubset(s2)`	`True`, wenn `s1` eine Teilmenge von `s2` ist
`e in s`	`True`, wenn das Element `e` in `s` enthalten ist; sonst `False`
`s.discard(e)`	Entferne das Element `e` aus `s`, falls es enthalten ist

Zusammenfassung

Wir haben gelernt

- wie Mengen in Python erstellt werden und worin sie sich von Listen unterscheiden.
- welche grundlegenden Operationen auf Mengen vorgenommen werden können.
- Mengen werden von Python schneller verarbeitet als Listen und sind diesen falls möglich vorzuziehen.

Weitere Themen dieser Woche

- Mutability 🤷