Aggregierte Datentypen

Copy: Um Aggregierte Datentypen zu kopieren verwendet man oft das Paket `copy`. Dieses stellt ein shallow copy und ein deep copy zur Verfügung.

```
import copy

x = copy.copy(el) # shallow
x = copy.deepcopy(el) # deep copy
```

Tupel

Definition: Ein Tupel ist eine geordnete Menge an Daten. Tupel sind dabei immer Immutable.

Python - Syntax: Ein Tupel wird durch `(el1,el2,...)` runde Klammern erstellt. Ein Leeres Tupel ist einfach `()`, ein Tupel mit nur einem Element hingegen muss wie folgt erstellt werden `(el, )`. In den meisten Fällen kann man beim erstellen eines Tupels auch die runden Klammern weg lassen und die Elemente einfach durch Komma trennen.

Python - Operationen: Ein Tupel in Python ist ein Sequence Objekt und unterstützt somit Indizierung, Slicing, und noch andere Funktionen wie: Verkettung mittels +, Wiederholung mittels * aber auch die `len()` Funktion. Tupel sind außerdem iterierbar.

Listen

Definition: Eine Liste ist eine geordnete Menge an Daten. Eine Liste ist dabei immer Mutable.

Python - Syntax: Eine Liste erstellt man durch `[el1,el2,...]` eckige Klammern.

Python - Operationen: Eine Liste in Python ist ein Sequence Objekt und unterstützt somit Indizierung und Slicing. Listen sind außerdem iterierbar. Weitere Operationen sind:

- Mutable: `list_1.append(el)` : Fügt ein neues Element an.
- Mutable: `list_1.extend([el_1,el_2,el_3,...])` : Fügt alle Elemente einer Liste an.
- Mutable: `del(list_1[index])` : Entfernt ein Element bei Index `index` von der Liste.
- Mutable: `list_1.pop()` : Entfernt und returned das letzte Element aus der Liste.
- Mutable: `list_1.remove(el)` : Entfernt das erste Vorkommen eines ein Elements. Ist es nicht Vorhanden gibt es einen ValueError.
- Immutable: `list_1 + list_2` .
- Immutable: `sorted(list_1)` : Gibt die sortierte Liste zurück. Funktioniert nur wenn alle Elemente einer Liste von selben Typ sind. UND funktioniert auch bei allen anderen Iterables!
- Mutable: `list_1.sort()` : Sortiert die Liste. Funktioniert nur wenn alle Elemente einer Liste von selben Typ sind.
- Mutable: `list_1.reverse()` : Kehrt eine Liste um.
- `list(string_1)` : Wandelt einen String in eine Liste um.
- `string_1.split(trenner = ' ')` : Trennt einen String an den gegebenen Zeichen.
- `''.join(string_1)` : Verbindet eine Liste zu einem String.
- `list_1[:]` : Cloned eine Liste - shallow clone.

Sets

Definition: Eine Liste ist eine ungeordnete Menge an Daten in der jedes Element nur einmal vorkommen darf. Ein Set ist dabei immer Mutable. Allerdings können in ein Set nur hashable Objekte gespeichert werden! Konkret sind dies meist nur immutable Objekte (keine Liste)!

Python - Syntax: Ein Set erstellt man mit `{el_1,el_2,...}` geschweiften Klammern. Achtung, ein Leeres Set erstellt man aber so `set()`!

Python - Operationen: Ein Set in Python unterstützt somit Indizierung und Slicing nicht! Sets sind allerdings iterierbar. Weitere Operationen sind:

- Immutable: `el_1 in set_1`: Prüft auf Vorhandensein und ist viel schneller als in anderen Datenstrukturen.
- Immutable: `set_1 & set_2` = `set_1.intersection(set_2)`: Schnittmenge.
- Immutable: `set_1 | set_2` = `set_1.union(set_2)`: Vereinigung.
- Immutable: `set_1 ^ set_2` = `set_1.symmetric_difference(set_2)`: Symmetrische Differenz.
- Immutable: `set_1 - set_2` = `set_1.difference(set_2)`: Differenz.
- Immutable: `set_1 - set_2` = `set_1.isdisjoint(set_2)`: Prüft auf Disjunkt.
- Immutable: `set_1 <= set_2` = `set_1.issubset(set_2)`: Prüft ob set_1 in set_2 ist.
- Immutable: `set_1 >= set_2` = `set_1.issuperset(set_2)`: Testet ob set_2 in set_1 ist.
- Mutable: `set_1.add()`: Fügt ein Element hinzu.
- Mutable: `set_1.clear()`: Entfernt alle Elemente.
- Mutable: `set_1.pop()`: Entfernt ein zufälliges Element.
- Mutable: `set_1.discard(value)`: Entfernt value aus dem Set ohne Fehlermeldung.
- Mutable: `set_1.remove(value)`: Entfernt value aus dem Set mit KeyError.
- `sorted(set_1)`: Erzeugt eine sortierte Liste aus dem Set, dabei müssen aber alle Elemente im Set den selben Datentyp haben.
- `list(set_1)`: Erzeugt auch ein sortierte Liste aus dem Set, erlaubt aber unterschiedliche Datentypen im Set.

Frozenset

Definition: Eine Liste ist eine ungeordnete Menge an Daten in der jedes Element nur einmal vorkommen darf. Ein Forzenset ist dabei gleich einem Set allerdings Immutable!

Python - Syntax: Ein Forzenset erstellt man mit der entsprechenden Funktion `frozenset(iterable_1)`.

Python - Operationen: Aller Immutable Operationen eines Sets!

Dictionary

Definition: Eine Liste aus Schlüssel - Wert Paaren. Als Schlüssel kommen dabei nur Hashable Objekte in Frage, also immutable Objekte.

Python - Syntax: Ein Dictionary erzeugt man mittels geschwungener Klammern, wobei Schlüssel und Wert immer durch einen Doppelpunkt getrennt sind. `{key_1:value_1,...}`. Ein Leeres Dict erstellt man so `{}`. Man kann ein dict auch mittels `dict([[key_1,value_1], [key_2,value_2]])` oder `dict(key_1=value_1, key_2=value_2)` erstellen.

Python - Operationen: Iterriert man über ein Dict, so wird über die Schlüssel iterriert.

- `dict_1.clear()`: Entferne alle Elemente.
- `dict_1.copy()`: Shallow Copy.
- `dict_1.get(key, default=None)`: Gibt den Wert eines Keys zurück. Ist dieser nicht vorhanden wird default returned.
- `dict_1.keys()`: Ein Set an Schlüssel.
- `dict_1.items()`: Ein Set, aus Schlüssel - Wert Paaren..
- `dict_1.values()`: Ein Set aus Werten.
- `del dict_1[key_1]`: Löschen
- `dict_1.pop(key_1)`: Entfernt und returned das Element.