Listen

Erstellen und Erweitern

Wir habe bereits gesehen, wie Listen initialisiert werden können (*my_list = ["1", "23", 1.43]*)
Natürlich können sie auch leer initialisiert werden: *list=[]*

Danach können Listen erweitert werden:

- **append**: Fügt ein Element am Schluss an
- **extend**: Fügt mehrere Elemente am Schluss an
- **insert**: fügt ein Element an einer beliebigen stelle ein

In [1]:
```
list = []

list.append("first entry")
list.append("second entry")
list.append(range(3)) # append adds one entry, here a Range
list.extend(range(3)) # append adds multiple entries, here a Range
list.extend('abc') # append adds multiple entries, here 'a', 'b' and 'c'
list.insert(2, "third entry") # inserts an entry at a certain position

print(f"list = {list}")
```

```
list = ['first entry', 'second entry', 'third entry', range(0, 3), 0, 1, 2, '
a', 'b', 'c']
```

Zu der Unterschied zwischen *append* und *extend* kommt dann richtig zum tragen, wenn eine ganze Liste eingefügt werden soll: *append* fügt die Liste als ein Element hinzu, während *extend* jedes einzelne Element in der hinzuzufügenden Liste einzeln anfügt.

Abfragen und Überschreiben von Elementen

Um ein Element an einer bestimmten Position abzufragen oder zu ändern, verwendet man die eckigen Klammern.

In [2]:
```
var = list[0]
print(f"Element at index 0 is '{var}'")
```

```
Element at index 0 is 'first entry'
```

In [3]:
```
list[0] = 'premier entry'
var = list[0]
print(f"Element at index 0 is '{var}'")
```

```
Element at index 0 is 'premier entry'
```

Position von Elementen Abfragen

Die Position von Elementen in der Liste kann mit der Funktion **index(element)** abgefragt werden:

In [4]:
```
print(f"index of element 'third entry': {list.index('third entry')}")
```

```
index of element 'third entry': 2
```

Falls das Element nicht in der Liste vorhanden ist, wird eine Exception geworfen:

In [5]:
```
print(f"index of element 'Tatzelwurm': {list.index('Tatzelwurm')}")
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In [5], line 1
----> 1 print(f"index of element 'Tatzelwurm': {list.index('Tatzelwurm')}")

ValueError: 'Tatzelwurm' is not in list
```

Testen, ob ein Element in der Liste vorkommt

Die Fehlermeldung von oben könnte verhindert werden, wenn zuvor geprüft wird, ob das gesuchte Element überhaupt in der Liste vorkommt. Dazu gibt es in Python den **in**-Operator.

In [6]:
```
searched_element = "third entry"

if searched_element in list:
    print(f"index of element '{searched_element}': {list.index(searched_elemen
else:
    print(f"Element '{searched_element}' is not in list")
```

```
index of element 'third entry': 2
```

Elemente löschen und Liste leeren

Mit dem **del**-Operator können Elemente aus der Liste entfernt werden. Es ist auch möglich, ganze Bereiche zu entfernen. Dazu wird slicing verwendet, das wir hier aber nicht eingehend behandeln werden. Da es bei strings manchmal gebracht wird, erwähne ich slicing hier kurz.

In [7]:
```
del list[3] # deletes the 4th entry (-> zero based), here the Range object
del list[3:5] #deletes the 4th and the 5th entry, here 0 and 1
print(f"list = {list}")
print(f"len(list) = {len(list)}")
```

```
list = ['premier entry', 'second entry', 'third entry', 2, 'a', 'b', 'c']
len(list) = 7
```

Die **clear()**-Funktion leert die gesamte Liste.

In [8]:
```
list.clear()
print(f"len(list) = {len(list)}") # Output: len(list) = 0
```

```
len(list) = 0
```

Anzahl Element und länge der Liste

len() liefert die Längeeiner Liste.Der Befehl **count()** kann verwendet werden, um zu zählen, wie häufig ein Element in der Liste vorkommt:

In [9]:
```
list = ['a', 'b', 'c', 'a', 3]
print(f"Element 'a' is {list.count('a')} times in the list")
print(f"There are {len(list)} elements in total in the list")
```

```
Element 'a' is 2 times in the list
There are 5 elements in total in the list
```

Dictionaries

Erstellen und initialisieren

Um ein Dictionary zu erstellen und initialisieren, werden die geschweiften Klammern verwendet: *phone_numbers = {"Hans": "041 234", "Peter": "041 322", "Kunigunde": "052 431"}*

Um einen leeres Dictionary zu erstellen, verwendet man die Funktion **dict()**: *empty = dict()*

In [10]:
```
phone_numbers = {"Hans": "041 234", "Peter": "041 322", "Kunigunde": "052 431"
empty = dict()

print(f"Phone numbers:    {phone_numbers}")
print(f"empty dictionary: {empty}")
```

```
Phone numbers:    {'Hans': '041 234', 'Peter': '041 322', 'Kunigunde': '052 4
31'}
empty dictionary: {}
```

Abfragen, Hinzufügen und Überschreiben von

Elementen

In [11]:
```
hans_phone_number = phone_numbers["Hans"] # Return value associated with key '
print(f"Hans' phone number: {hans_phone_number}")
```

```
Hans' phone number: 041 234
```

Hinzufügen oder Überschreiben hat den selben Syntax - es kommt darauf an, ob es bereits ein Element mit dem gegebenen Schlüssel gibt:

In [12]:
```
phone_numbers["Hans"] = "000 000" # changes the value associated with key 'Han
phone_numbers["New Kid on the Block"] = "043 678" # adds a new key/value pair

print(f"Phone numbers: {phone_numbers}")
```

```
Phone numbers: {'Hans': '000 000', 'Peter': '041 322', 'Kunigunde': '052 431
', 'New Kid on the Block': '043 678'}
```

In [ ]:

In [ ]:

In [ ]:

In [ ]: