analogous to a hash table or a map. Dictionaries are surrounded by a pair of braces {…}. A dictionary is not ordered.

```
my_dict = {'name':'Rijami', 'age':42, 'height':72}
my_dict              # dictionary items are un-ordered
'Output': {'age': 42, 'height': 72, 'name': 'Rijami'}
my_dict['age']       # get dictionary value by indexing on keys
'Output': 42
my_dict['age'] = 35  # change the value of a dictionary item
my_dict['age']
'Output': 35
```

More on Lists

As earlier mentioned, because list items are mutable, they can be changed, deleted, and sliced to produce a new list.

```
my_list = [4, 8, 16, 32, 64]
my_list
'Output': [4, 8, 16, 32, 64]
my_list[1:3]      # slice the 2nd to 4th element (indexed from 0)
'Output': [8, 16]
my_list[2:]       # slice from the 3rd element (indexed from 0)
'Output': [16, 32, 64]
my_list[:4]       # slice till the 5th element (indexed from 0)
'Output': [4, 8, 16, 32]
my_list[-1]       # get the last element in the list
'Output': 64
min(my_list)      # get the minimum element in the list
'Output': 4
max(my_list)      # get the maximum element in the list
'Output': 64
sum(my_list)      # get the sum of elements in the list
'Output': 124
my_list.index(16) # index(k) - return the index of the first occurrence of
item k in the list
'Output': 2
```

When modifying a slice of elements in the list, the right-hand side can be of any length depending that the left-hand size is not a single index.

```
# modifying a list: extended index example
my_list[1:4] = [43, 59, 78, 21]
my_list
'Output': [4, 43, 59, 78, 21, 64]
my_list = [4, 8, 16, 32, 64]  # re-initialize list elements
my_list[1:4] = [43]
my_list
'Output': [4, 43, 64]
```

```
# modifying a list: single index example
my_list[0] = [1, 2, 3]      # this will give a list-on-list
my_list
'Output': [[1, 2, 3], 43, 64]
my_list[0:1] = [1, 2, 3]    # again - this is the proper way to extend lists
my_list
'Output': [1, 2, 3, 43, 64]
```

Some useful list methods include

```
my_list = [4, 8, 16, 32, 64]
len(my_list)          # get the length of the list
'Output': 5
my_list.insert(0,2)   # insert(i,k) - insert the element k at index i
my_list
'Output': [2, 4, 8, 16, 32, 64]
my_list.remove(8) # remove(k) - remove the first occurrence of element k in
                                the list
my_list
'Output': [2, 4, 16, 32, 64]
my_list.pop(3)    # pop(i) - return the value of the list at index i
'Output': 32
my_list.reverse() # reverse in-place the elements in the list
my_list
'Output': [64, 16, 4, 2]
```

```
my_list.sort()    # sort in-place the elements in the list
my_list
'Output': [2, 4, 16, 64]
my_list.clear()   # clear all elements from the list
my_list
'Output': []
```

The append() method adds an item (could be a list, string, or number) to the end of a list. If the item is a list, the list as a whole is appended to the end of the current list.

```
my_list = [4, 8, 16, 32, 64]  # initial list
my_list.append(2)             # append a number to the end of list
my_list.append('wonder')      # append a string to the end of list
my_list.append([256, 512])    # append a list to the end of list
my_list
'Output': [4, 8, 16, 32, 64, 2, 'wonder', [256, 512]]
```

The extend() method extends the list by adding items from an iterable. An iterable in Python are objects that have special methods that enable you to access elements from that object sequentially. Lists and strings are iterable objects. So extend() appends all the elements of the iterable to the end of the list.

```
my_list = [4, 8, 16, 32, 64]
my_list.extend(2)             # a number is not an iterable
Traceback (most recent call last):

  File "<ipython-input-24-092b23c845b9>", line 1, in <module>
    my_list.extend(2)

TypeError: 'int' object is not iterable

my_list.extend('wonder')      # append a string to the end of list
my_list.extend([256, 512])    # append a list to the end of list
my_list
'Output': [4, 8, 16, 32, 64, 'w', 'o', 'n', 'd', 'e', 'r', 256, 512]
```

We can combine a list **with another list** by overloading the operator +.

```
my_list = [4, 8, 16, 32, 64]
my_list + [256, 512]
'Output': [4, 8, 16, 32, 64, 256, 512]
```

Strings

Strings in Python are enclosed by a pair of single quotes (' … '). Strings are immutable. This means they cannot be altered when assigned or when a string variable is created. Strings can be indexed like a list as well as sliced to create new lists.

```
my_string = 'Schatz'
my_string[0]      # get first index of string
'Output': 'S'
my_string[1:4]    # slice the string from the 2nd to the 5th element
                    (indexed from 0)
'Output': 'cha'
len(my_string)    # get the length of the string
'Output': 6
my_string[-1]     # get last element of the string
'Output': 'z'
```

We can operate on string values with the boolean operators.

```
't' in my_string
'Output': True
't' not in my_string
'Output': False
't' is my_string
'Output': False
't' is not my_string
'Output': True
't' == my_string
'Output': False
't' != my_string
'Output': True
```

We can concatenate two strings to create a new string using the overloaded operator +.

```
a = 'I'
b = 'Love'
c = 'You'
```