Python Dictionaries

S1 MCA

Dictionary

- Dictionaries are used to store data values in key:value pairs.
- A dictionary is a collection which is ordered*, changeable and do not allow duplicates.
- As of Python version 3.7, dictionaries are *ordered*. In Python 3.6 and earlier, dictionaries are *unordered*.
- Dictionaries are written with curly brackets, and have keys and values

```
>>>    thisdict = {
       "brand": "Ford",
       "model": "Mustang",
       "year": 1964
       }
       print(thisdict)
```

Dictionary Items

- Dictionary items are ordered, changeable, and do not allow duplicates.
- Dictionary items are presented in key:value pairs, and can be referred to by using the key name.

```
>>>  thisdict = {
     "brand": "Ford",
     "model": "Mustang",
     "year": 1964
     }
     print(thisdict["brand"])
```

Changeable

- Dictionaries are changeable, meaning that we can change, add or remove items after the dictionary has been created.

Duplicates Not Allowed

- Dictionaries cannot have two items with the same key:
- Duplicate values will overwrite existing values:

```
>>>  thisdict = {
     "brand": "Ford",
      "model": "Mustang",
      "year": 1964,
     "year": 2020
     }
     print(thisdict)
```

Dictionary Length

To determine how many items a dictionary has, use the len() function

```
>>>     print(len(thisdict))
```

Dictionary Items - Data Types

- The values in dictionary items can be of any data type:

```
>>>     thisdict = {
        "brand": "Ford",
        "electric": False,
         "year": 1964,
         "colors": ["red", "white", "blue"]
        }
```

type()

From Python's perspective, dictionaries are defined as objects with the data type 'dict':

```
<class 'dict'>
```

- **The dict() Constructor**

It is also possible to use the dict() constructor to make a dictionary.

```
>>>    thisdict = dict(name = "John", age = 36, country = "Norway")
       print(thisdict)
```

Accessing Items

- You can access the items of a dictionary by referring to its key name, inside square brackets:

```
>>>    thisdict = {
       "brand": "Ford",
        "model": "Mustang",
         "year": 1964
       }
       x = thisdict["model"]
```

There is also a method called **get()** that will give you the same result:

- >>> x = thisdict.get("model")

- Get Keys

The **keys**() method will return a list of all the keys in the dictionary.

```
>>>     x = thisdict.keys()
```

- The list of the keys is a *view* of the dictionary, meaning that any changes done to the dictionary will be reflected in the keys list.

```
>>>     car = {
        "brand": "Ford",
        "model": "Mustang",
        "year": 1964
        }

        x = car.keys()

        print(x) #before the change

        car["color"] = "white"

        print(x) #after the change
```

Get Values

The **values**() method will return a list of all the values in the dictionary.

```
>>>     x = thisdict.values()
```

- The list of the values is a *view* of the dictionary, meaning that any changes done to the dictionary will be reflected in the values list.

```
>>>     car = {
        "brand": "Ford",
        "model": "Mustang",
        "year": 1964
        }
        x = car.values()
        print(x) #before the change
        car["year"] = 2020
        print(x) #after the change
```

- Add a new item to the original dictionary, and see that the values list gets updated as well:

```
>>>    car = {
       "brand": "Ford",
       "model": "Mustang",
       "year": 1964
       }

       x = car.values()

       print(x) #before the change

       car["color"] = "red"

       print(x) #after the change
```

Get Items

- The **items**() method will return each item in a dictionary, as tuples in a list.

Get a list of the key:value pairs

```
>>>     thisdict = {
        "brand": "Ford",
        "model": "Mustang",
        "year": 1964
        }
        x = thisdict.items()
        print(x)
```

```
dict_items([('brand', 'Ford'), ('model', 'Mustang'), ('year', 1964)])
```

- Add a new item to the original dictionary, and see that the items list gets updated as well

```
>>>  car = {
     "brand": "Ford",
     "model": "Mustang",
     "year": 1964
     }

     x = car.items()

     print(x) #before the change

     car["color"] = "red"

     print(x) #after the change
```

```
dict_items([('brand', 'Ford'), ('model', 'Mustang'), ('year', 1964)])
dict_items([('brand', 'Ford'), ('model', 'Mustang'), ('year', 1964), ('color', 'red'
```

Check if Key Exists

- To determine if a specified key is present in a dictionary use the in keyword:

```
>>>     thisdict = {
        "brand": "Ford",
         "model": "Mustang",
         "year": 1964
        }
        if "model" in thisdict:
         print("Yes, 'model' is one of the keys in the thisdict dictionary")
```

→ Check if "model" is present in the dictionary

Change Values

- You can change the value of a specific item by referring to its key name

```
>>>    thisdict = {
       "brand": "Ford",
        "model": "Mustang",
        "year": 1964
       }
       thisdict["year"] = 2018
```

Update Dictionary

- The update() method will update the dictionary with the items from the given argument.
- The argument must be a dictionary, or an iterable object with key:value pairs.

```
>>>    thisdict = {
       "brand": "Ford",
       "model": "Mustang",
       "year": 1964
       }
       thisdict.update({"year": 2020})
```

Adding Items

- Adding an item to the dictionary is done by using a new index key and assigning a value to it:

```
>>>    thisdict = {
        "brand": "Ford",
        "model": "Mustang",
        "year": 1964
       }
       thisdict["color"] = "red"
       print(thisdict)
```

Update Dictionary

- The update() method will update the dictionary with the items from a given argument. If the item does not exist, the item will be added.
- The argument must be a dictionary, or an iterable object with key:value pairs.

```
>>>  thisdict = {
     "brand": "Ford",
     "model": "Mustang",
     "year": 1964
     }
     thisdict.update({"color": "red"})
```

Removing Items

- There are several methods to remove items from a dictionary:

The **pop()** method removes the item with the specified key name:

```
>>>    thisdict = {
       "brand": "Ford",
       "model": "Mustang",
       "year": 1964
       }
       thisdict.pop("model")
       print(thisdict)
```

- The **popitem()** method removes the last inserted item (in versions before 3.7, a random item is removed instead)

```
>>>  thisdict = {
     "brand": "Ford",
     "model": "Mustang",
     "year": 1964
     }
     thisdict.popitem()
     print(thisdict)
```

- The **del** keyword removes the item with the specified key name

```
>>>  thisdict = {
      "brand": "Ford",
      "model": "Mustang",
      "year": 1964
     }
     del thisdict["model"]
     print(thisdict)
```

- The del keyword removes the item with the specified key name

```
>>>  thisdict = {
      "brand": "Ford",
      "model": "Mustang",
      "year": 1964
     }
     del thisdict
     print(thisdict) #this will cause an error because "thisdict" no longer exists.
```

- The **clear()** method empties the dictionary

```
>>>  thisdict = {
     "brand": "Ford",
      "model": "Mustang",
      "year": 1964
     }
     thisdict.clear()
     print(thisdict)
```

Loop Through a Dictionary

You can loop through a dictionary by using a for loop.

When looping through a dictionary, the return value are the *keys* of the dictionary, but there are methods to return the *values* as well.

- Print all key names in the dictionary, one by one

```
>>>    thisdict =      {
       "brand": "Ford",
        "model": "Mustang",
        "year": 1964
       }
       for x in thisdict:
        print(x)
```

- Print all *values* in the dictionary, one by one:

```
>>>    thisdict =      {
       "brand": "Ford",
       "model": "Mustang",
        "year": 1964
       }
       for x in thisdict:
        print(thisdict[x])
```

- You can also use the **values()** method to return values of a dictionary

```
>>>    for x in thisdict.values():
        print(x)
```

- You can use the **keys()** method to return the keys of a dictionary

```
>>>    for x in thisdict.keys():
                print(x)
```

Loop through both keys and values, by using the items() method

```
>>>    for x, y in thisdict.items():
                print(x, y)
```

Copy a Dictionary

- You cannot copy a dictionary simply by typing dict2 = dict1,

because: dict2 will only be a reference to dict1, and changes made in dict1 will automatically also be made in dict2.

- There are ways to make a copy, one way is to use the built-in Dictionary method **copy().**

```
>>>  thisdict = {
     "brand": "Ford",
     "model": "Mustang",
     "year": 1964
     }
     mydict = thisdict.copy()
     print(mydict)
```

- Another way to make a copy is to use the built-in function dict().

```
>>>    thisdict = {
       "brand": "Ford",
        "model": "Mustang",
        "year": 1964
       }
       mydict = dict(thisdict)
       print(mydict)
```

Nested Dictionaries

- A dictionary can contain dictionaries, this is called nested dictionaries.

```
>>>    myfamily = {
       "child1" : {
        "name" : "Emil",
         "year" : 2004
       },
       "child2" : {
       "name" : "Tobias",
        "year" : 2007
       },
       "child3" : {
       "name" : "Linus",
       "year" : 2011
       }
      }
```

- Create three dictionaries, then create one dictionary that will contain the other three dictionaries:

```
>>>  child1 = {
     "name" : "Emil",
     "year" : 2004
     }
     child2 = {
     "name" : "Tobias",
     "year" : 2007
     }
     child3 = {
     "name" : "Linus",
     "year" : 2011
     }

     myfamily = {
     "child1" : child1,
     "child2" : child2,
     "child3" : child3
     }
```

Access Items in Nested Dictionaries

- To access items from a nested dictionary, you use the name of the dictionaries, starting with the outer dictionary:

```
>>> print(myfamily["child2"]["name"])
```

Loop Through Nested Dictionaries

- You can loop through a dictionary by using the items() method like this

```
>>>     for x, obj in myfamily.items():
            print(x)

        for y in obj:
            print(y + ':', obj[y])
```

Dictionary Methods

Method	Description
clear()	Removes all the elements from the dictionary
copy()	Returns a copy of the dictionary
fromkeys()	Returns a dictionary with the specified keys and value
get()	Returns the value of the specified key
items()	Returns a list containing a tuple for each key value pair
keys()	Returns a list containing the dictionary's keys
pop()	Removes the element with the specified key
popitem()	Removes the last inserted key-value pair
setdefault()	Returns the value of the specified key. If the key does not exist: insert the key, with the specified value
update()	Updates the dictionary with the specified key-value pairs
values()	Returns a list of all the values in the dictionary