Day 20: Dictionaries

- There are 4 built-in data types in Python used to store collections of data:
  1. List
  2. Tuple
  3. Set
  4. Dictionary

Dictionary

- Dictionaries are used to store data values in key: value pairs.
- A dictionary is a collection which is:
  - Ordered
  - Changeable
  - Do not allow duplicates

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

Access Specific Elements of the Dictionary

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
```

```
}
print(thisdict["brand"])
```

Ordered

- When we say that dictionaries are ordered, it means that the items have a defined order, and that order will not change.

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

Changeable

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

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

# Changeable
thisdict["year"] = 1965
```

Duplicates not allowed

- Dictionaries cannot have two items with the same key.

```
thisdict = {
  "brand": "Ford",
```

```
  "model": "Mustang",
  "year": 1964,
  "year": 2020
}
print(thisdict)
# duplicate value will overwrite the existing value
```

Dictionary Length

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

print(len(thisdict))
# return 3
```

Dictionary Items - Data Types

- Dictionaries allow all data types

```
thisdict = {
  "brand": "Ford", # String
  "electric": False, # Boolean
  "year": 1964, # Integer
  "colors": ["red", "white", "blue"] # List
}
```

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)
```

Access 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"]
```

get() method

- Returns the value for the mentioned key.

```
x = thisdict.get("model")
```

keys() method

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

```
x = thisdict.keys()

# view of the dictionary
car = {
"brand": "Ford",
"model": "Mustang",
"year": 1964
```

```
}

x = car.keys()

print(x) #before the change

car["color"] = "white"

print(x) #after the change
```

values() method

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

```
x = thisdict.values()

# view of the dictionary
car = {
"brand": "Ford",
"model": "Mustang",
"year": 1964
}

x = car.values()

print(x) #before the change

car["year"] = 2020
car["color"] = "red"

print(x) #after the change
```

items() method

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

```
x = thisdict.items()
```

Exercise:

1. Check whether the specified key is present in a dictionary or not:

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

Change Items

```
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})
```

📝 Test