Chapter 7:Dictionaries

Prepared by: Hanan Hardan

Python Dictionaries

- Dictionaries are used to store data values in key: value pairs.
- A dictionary is a collection which is ordered, changeable and does not allow duplicates.
- Dictionaries are written with curly brackets, and have keys and values
- Index values are called keys
- Keys are unique within a dictionary while values may not be.
- It is generally used when we have a huge amount of data. Dictionaries are optimized for retrieving data. We must know the key to retrieve the value.

Python Dictionaries

- Create and print a dictionary:

```
dict = {'Name': 'Zara',
        'Age': 7,
        'Class': 'First'
       }

print(dict)
```

Python Dictionaries

- Create and print a dictionary:

```
dict = { }
dict[‘Name'] = ‘Zara'
dict[‘Age']=7
dict[‘Class'] = ‘First'
```

Dictionary Items

- Dictionary items are presented in key : value pairs, and can be referred to by using the key name.

```
dict = {'Name': 'Zara',
         'Age': 7,
         'Class': 'First'
        }

print(dict['Age'])
```

Dictionary Items

- Dictionary items are ordered
  - When we say that dictionaries are ordered, it means that the items have a defined order, and that order will not change.
- Dictionary items are changeable
  - Dictionaries are changeable, meaning that we can change, add or remove items after the dictionary has been created.

Dictionary Items

- Dictionary items are does not allow duplicates.
  - Dictionaries cannot have two items with the same key:
  - Duplicate values will overwrite existing values:

```
dict = {'Name': 'Zara',
         'Age': 7,
         'Class': 'First‘
         'Class': ‘Last‘
       }
print(dict)
```

Dictionary Length

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

Example: Print the number of items in the dictionary:

```
dict = {'Name': 'Zara',
        'Age': 7,
        'Class': 'First‘
}
print(len(dict))
```

Dictionary Items - Data Types

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

Example: String, int, boolean, and list data types:

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

Access Dictionary Items

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

```
dict = {'Name': 'Zara',
        'Age': 7,
        'Class': 'First‘
}
print(dict[‘Name’])
```

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

Example: Get the value of the “Class" key:

```
x = dict.get(“Class")
```

Access Dictionary Items

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

Example: Get a list of the keys:

x = dict.keys()

Access Dictionary Items

- 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.

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

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

Access Dictionary Items

Get Items

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

Example

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

x = car.items()
print(x)
```

Access Dictionary Items

Check if Key Exists

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

Example

Check if "model" is present in the dictionary:

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

Change Dictionary Items

Change Values

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

Example: Change the "year" to 2018:

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

Change Dictionary Items

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.

Example: Update the "year" of the car by using the update() method:

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

Add Dictionary Items

Adding Items

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

Example

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


car["color"] = "red"
print(car)
```

Add Dictionary Items

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.

Example: Add a color item to the dictionary by using the update() method:

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

Remove Dictionary Items

- There are several methods to remove items from a dictionary:
- The pop() method removes the item with the specified key name:

Example:

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

Remove Dictionary Items

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

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

Remove Dictionary Items

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

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

Remove Dictionary Items

- The del keyword can also delete the dictionary completely:

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

Remove Dictionary Items

- The clear() method empties the dictionary:

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

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.

Example: Print all key names in the dictionary, one by one:

```
car = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
for x in car:
  print(x)
#Print all values in the dictionary, one by one:
for x in car:
  print(car[x])
```

Loop Through a Dictionary

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

```
for x in car.values():
  print(x)
```

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

```
for x in car.keys():
  print(x)
```

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

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

Sort A dictionary

- display a dictionary in sorted order on keys

```
dict = {
        'Name': 'Zara',
        'Age': 7,
        'Class': 'First',
   'Grade':70
}

for x in sorted(dict):
        print (x, dict[x])
```

Example 1

Write a Python script to generate and print a dictionary that contains a number (between 1 and n) in the form (x, x*x).

```
n=int(input("Input a number "))
d = dict()

for x in range(1,n+1):
    d[x]=x*x

print(d)
```

Example 2

Write a Python program to sum all the items in a dictionary.

```
my_dict = {'data1':100,'data2':-54,'data3':247}

sum=0
for i in my_dict:
    sum+=my_dict[i]
print(sum)
```

Example 3

- What if we want to know the keys that are associated with an item?
- Can do this one at a time or build a complete reverse dictionary.

original = {'A':1, 'B':3, 'C':3, 'D':4, 'E': 1, 'F': 3}

reverse = {1: ['A', 'E'], 3: ['C', 'B', 'F'], 4: ['D'] }

Solution:

```
def buildReverse(dictionary):
    reverse = { }
    for key,value in dictionary.items():
        if value in reverse:
            reverse[value].append(key)
        else:
            reverse[value] = [key]
    return reverse
```

Filter a dictionary by conditions on keys or values

```
dictOfNames = {7 : 'sami',8: 'jana',9: 'rami',10: 'rana',11 : 'reem',
               12 : 'salma'
               }
```

Suppose we want to filter above dictionary by keeping only elements whose keys are even. For that we can just iterate over all the items of dictionary and add elements with even key to an another dictionary

```
newDict = dict()
for (key, value) in dictOfNames.items():
    if key % 2 == 0:
     newDict[key] = value

print(newDict)
```

Filter a Dictionary by keys in Python using dict comprehension

Let’s filter items in dictionary whose keys are even i.e. divisible by 2 using dict comprehension ,

```
dictOfNames = {7 : 'sami',8: 'jana',9: 'rami',10: 'rana',11 : 'reem',
                    12 : 'salma'
                    }

ND ={ key:value for (key,value) in dictOfNames.items() if key % 2 == 0}

print(newDict)
```

Example 4

Write a Python program to drop empty Items from a given Dictionary.

Sample Solution:

Python Code:

```
dict1 = {'c1': 'Red', 'c2': 'Green', 'c3':None}
print("Original Dictionary:")
print(dict1)
print("New Dictionary after dropping empty items:")
dict1 = {key:value for (key, value) in dict1.items() if value is not None}
print(dict1)
```

Example 5

Write a Python program to filter a dictionary based on marks greater than 70.

```
marks = {‘1': 75,
         ‘2': 80,
         ‘3': 65,
         ‘4': 90}

print("Original Dictionary:")
print(marks)
print("Marks greater than 70:")
result = {key:value for (key, value) in marks.items() if value > 70}
print(result)
```