Programming and Problem Solving through Python Language
O Level / A Level

Chapter - 5: Sequence Data Types

Python Collections (Arrays)

There are four collection data types in the Python programming language:

- **List** is a collection which is ordered and changeable. Allows duplicate members.
- **Tuple** is a collection which is ordered and unchangeable. Allows duplicate members.
- **Set** is a collection which is unordered and unindexed. No duplicate members.
- **Dictionary** is a collection which is unordered, changeable and indexed. No duplicate members.

Dictionary

- A dictionary is a collection which is unordered, changeable and indexed. In Python dictionaries are written with curly brackets, and they have keys and values.
- Each key is separated from its value by a colon (:)
- Keys are unique within a dictionary while values may not be.
- The values of a dictionary can be of any type, but the keys must be of an immutable data type such as strings, numbers, or tuples.

Creating Dictionary

```
dict1 = {
            "brand" : "Ford",
            "model" : "Mustang",
            "year" : 1964
        }
dict2 = {'Name' : 'Zara',  'Age' : 7,  'Class' : 'First' }
dict3 = { }        #Empty Dictionary
```

Access Items

- We can access the items of dictionary by referring to its key name, inside square brackets.
- **get( )** method will be used to get the value of key.

```
dict1 = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}
print ( dict1 )
print ( dict1['Age'] )
x= dict1['Class']
print ( x)
x= dict1.get('Name')
print ( x)
```

Output−

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

Updating Dictionary

We can update a dictionary by adding a new entry or a key-value pair, modifying an existing entry, or deleting an existing entry

```
dict1 = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}
dict1['Age'] = 10
print ( dict1['Age'] )
```

Output

```
{'Name': 'Zara', 'Age': 10, 'Class': 'First'}
```

Adding new item in Dictionary

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

```
dict1 = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}
print ( dict1 )
dict1['Height'] = 4
print ( dict1 )
```

Output

```
{'Name': 'Zara', 'Age': 7, 'Class': 'First'}
{'Name': 'Zara', 'Age': 7, 'Class': 'First', 'Height': 4}
```

Loop Through a Dictionary

- When looping through a dictionary, the return values are the *keys* of the dictionary, but there are methods to return the *values* as well.
- use the **keys**() function to return keys of a dictionary.
- use the **values**() function to return values of a dictionary.
- Loop through both *keys* and *values*, by using the **items**() function

```
dict1 = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}
for x in dict1:              #Print all key names in the dictionary, one by one
    print(x)

for x in dict1:              #Print all values in the dictionary, one by one
    print(dict1[x])

for x in dict1.values( ):  #Print all values in the dictionary, using the values() function
    print(x)
```

```
for x , y in dict1.items( ): #Print  both keys and values, by using the items() function
        print(x , y)
```

Output

```
Name
Age
Class

Zara
7
First

Zara
7
First

Name Zara
Age 7
Class First
```

Check if Item Dictionary

To determine if a specified item is present in a Dictionary use the "**in**" keyword:

```
dict1 = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}
if  "Age" in dict1:
    print("Yes, 'Age' is the key in the Dictionary")
```

Length of Dictionary

To determine how many items (key-value pairs) a **Dictionary** has, use the **len( )** function.
e.g. print(len(dict1))

Removing Item from the Dictionary

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

```
dict1 = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}
dict1.pop('Name')
print(dict1)
```

- The **popitem( )** method removes the last inserted item.

```
dict1 = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}
dict1.popitem( )
print(dict1)
```

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

```
dict1 = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}
dict1.del('Name')
print(dict1)
```

- The **del** keyword can also delete the dictionary completely.

```
dict1 = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}
del dict1
print(dict1)
```

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

```
dict1 = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}
dict1.del('Name')
print(dict1)
```

Copy a Dictionary

- We 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.
- To make a copy, one way is to use the built-in Dictionary method **copy**( ).
- Another way to make a copy is to use the built-in method or constructor **dict** ( ).

```
dict1 = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}
dict2= dict1.copy( )
print(dict2)

dict3=dict(dict1)
print(dict2)
```

Nested Dictionaries

A dictionary can also contain many dictionaries, this is called nested dictionaries.

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

Output

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

Assignment

1. Define Dictionary
2. Write the output from the following code:

```
A={1:100,2:200,3:300,4:400,5:500}
 print (A.items() )
 print (A.keys())
 print (A.values())
```