Python Data Type: DICTIONARY

Github: @Aaryacode19

Theory: - 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, unchangeable, and unindexed. No duplicate members.
- **Dictionary** is a collection which is ordered and changeable. No duplicate members.

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. Dictionaries are written with curly brackets, and have keys and values.

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

Ordered
Dictionaries are ordered, it means that the items have a defined order, and that order will not change.

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.

Create and print a dictionary:

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

Output:

```
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
```

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

```
print(len(thisdict))
```
Output:
```
3
```

Dictionary Items - Data Types
The values in dictionary items can be of any data type:
String, int, boolean, and list data types:

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

Output:
```
'brand': 'Ford', 'electric': False, 'year': 1964, 'colors': ['red',
'white', 'blue']}
```

Accessing Items
You can access the items of a dictionary by referring to its key name, inside square brackets:
Get the value of the "model" key:

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

print(x)
```

Output:
```
Mustang
```

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

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

Output:

```
Mustang
```

Get Keys

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

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

Output:

```
dict_keys(['brand', 'model', 'year'])
```

Get Values

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

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

Output:

```
dict_values(['Ford', 'Mustang', 1964])
```

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")
```

Output:

```
Yes, 'model' is one of the keys in the thisdict dictionary
```

Change Values

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

Change the "year" to 2018:

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

Output:

```
{'brand': 'Ford', 'model': 'Mustang', '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.
Add new key value pair:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict.update({"price": 20000})
thisdict.update({"color": "red"})
```

Output:

```
{'brand': 'Ford', 'model': 'Mustang', 'year': 2020, ‘pirce’ : 20000,
‘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)
```

Output:

```
{'brand': 'Ford', 'year': 1964}
```

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

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

Output:

```
{'brand': 'Ford', 'year': 1964}
```

The **del** keyword can also delete the dictionary **completely**:

```
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)
```

Output:

```
{}
```

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)
```

Output:

```
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964
```

Nested Dictionaries

A dictionary can contain dictionaries, this is called nested dictionaries.
Create a dictionary that contain three dictionaries:

```
myfamily = {
  "child1" : {
    "name" : "Ram",
    "year" : 2004
  },
  "child2" : {
    "name" : "Sham",
    "year" : 2007
  },
  "child3" : {
    "name" : "Hari",
    "year" : 2011
  }
}
```

Output:

```
 {'child1': {'name': 'Ram', 'year': 2004}, 'child2': {'name': 'Sham',
'year': 2007}, 'child3': {'name': 'Hari', 'year': 2011}}
```

Result: Practical has been studied successfully.