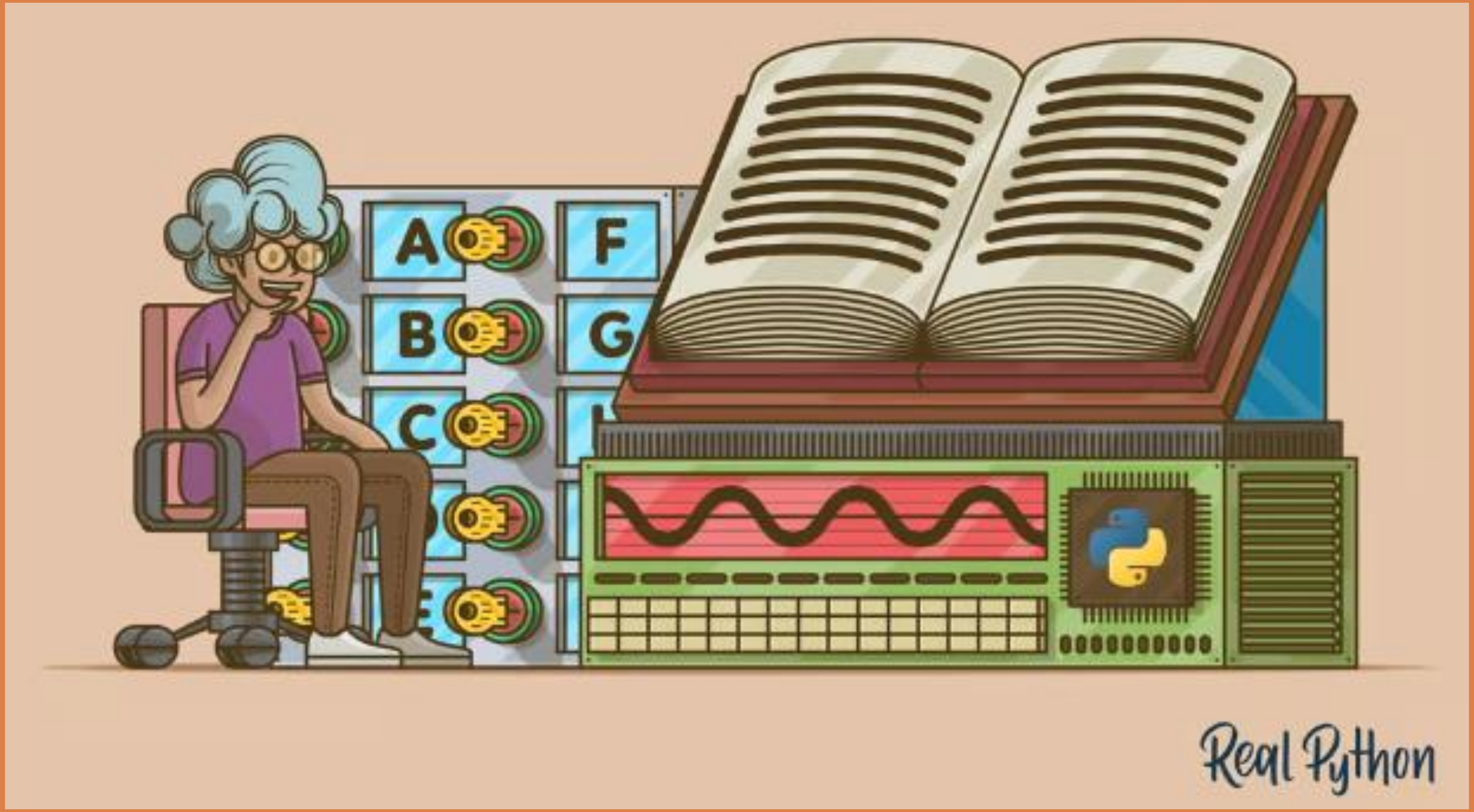


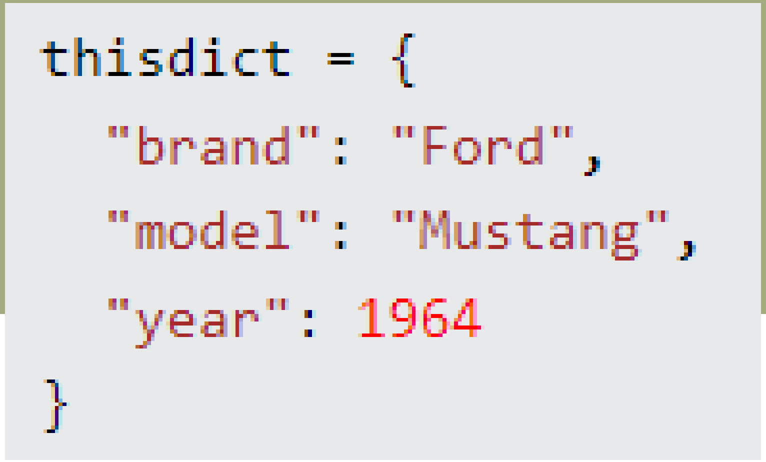
PYTHON – DICTIONARIES

Course Unit 7: Week 9

OBJECTIVES:

1. Discuss the python dictionaries and arrays.
2. Demonstrate programs using python dictionaries and arrays.
3. Develop a python program using dictionaries and arrays.

- **Python – Dictionaries**
- Accessing Dictionary Items
- Changing Dictionary Items
- Add Dictionary Items
- Remove Dictionary Items
- Dictionary View Objects
- Loop Dictionaries
- Copy Dictionaries
- Nested Dictionaries
- Dictionary Methods



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

PYTHON – DICTIONARIES

Course Unit 7: Week 9

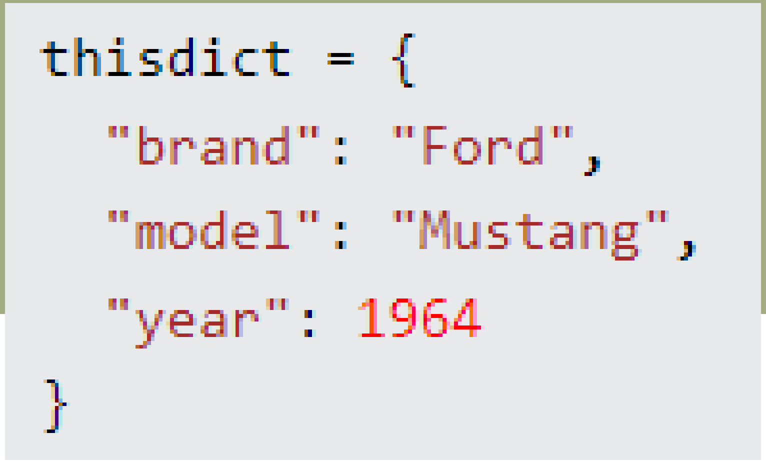
PYTHON – DICTIONARIES

- one of the built-in data types in Python
- example of mapping type (A mapping object 'maps' value of one object with another.)
- In a language dictionary we have pairs of word and corresponding meaning. Two parts of pair are key (word) and value (meaning).
- a collection of key: value pairs.
- pairs are separated by comma and put inside curly brackets {}.
- To establish mapping between key and value, the colon ':' symbol is put between the two.

PYTHON – DICTIONARIES

- used to store multiple items in a single variable
- used to store data values in key: value pairs
- written with curly brackets { }
- 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.
- can be of any data type
- defined as objects with the data type 'dict'

- Python – Dictionaries
- **Accessing Dictionary Items**
- Changing Dictionary Items
- Add Dictionary Items
- Remove Dictionary Items
- Dictionary View Objects
- Loop Dictionaries
- Copy Dictionaries
- Nested Dictionaries
- Dictionary Methods



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

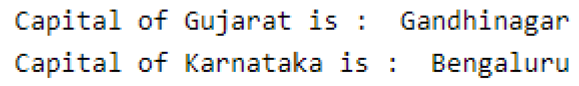
PYTHON – DICTIONARIES

Course Unit 7: Week 9

ACCESSING DICTIONARY ITEMS

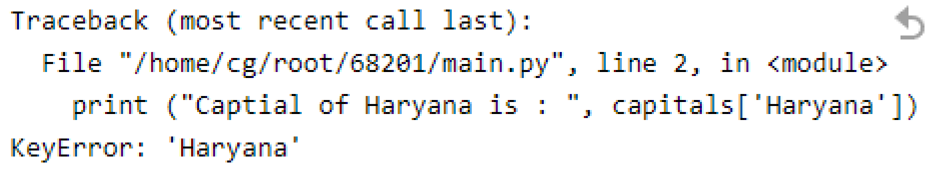
❖Using the "[ ]" Operator

```
capitals = {"Maharashtra":"Mumbai", "Gujarat":"Gandhinagar",
"Telangana":"Hyderabad", "Karnataka":"Bengaluru"}

print ("Capital of Gujarat is : ", capitals['Gujarat'])

print ("Capital of Karnataka is : ", capitals['Karnataka'])
```



```
Capital of Gujarat is :  Gandhinagar
Capital of Karnataka is :  Bengaluru
```

```
capitals = {"Maharashtra":"Mumbai", "Gujarat":"Gandhinagar",
"Telangana":"Hyderabad", "Karnataka":"Bengaluru"}

print ("Captial of Haryana is : ", capitals['Haryana'])
```



```
Traceback (most recent call last):
  File "/home/cg/root/68201/main.py", line 2, in <module>
    print ("Captial of Haryana is : ", capitals['Haryana'])
KeyError: 'Haryana'
```

ACCESSING DICTIONARY ITEMS

❖Using the get() Method

Syntax

Val = dict.get("key")

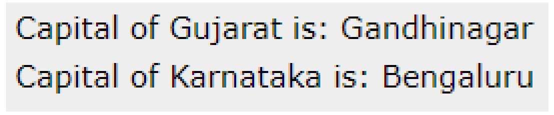
Parameters

key – An immutable object used as key in the dictionary object

Return Value

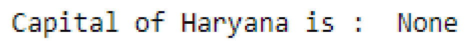
The get() method returns the object mapped with the given key.

```
capitals = {"Maharashtra":"Mumbai",
"Gujarat":"Gandhinagar", "Telangana":"Hyderabad",
"Karnataka":"Bengaluru"}
print ("Capital of Gujarat is: ", capitals.get('Gujarat'))
print ("Capital of Karnataka is: ", capitals.get('Karnataka'))
```



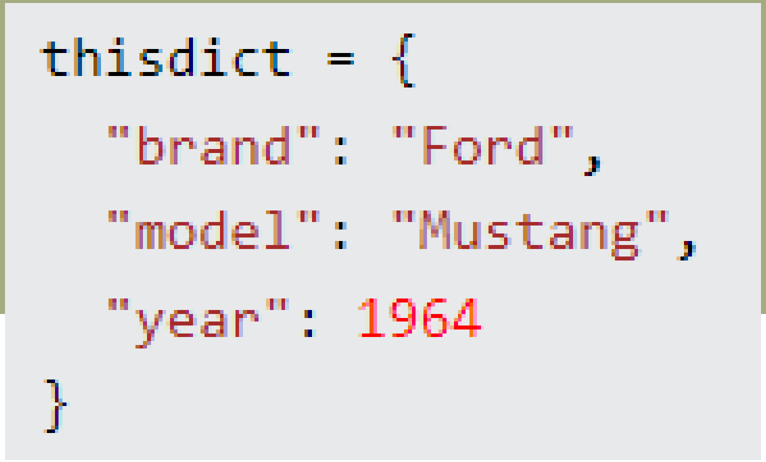
```
Capital of Gujarat is: Gandhinagar
Capital of Karnataka is: Bengaluru
```

```
capitals = {"Maharashtra":"Mumbai", "Gujarat":"Gandhinagar",
"Telangana":"Hyderabad", "Karnataka":"Bengaluru"}
print ("Capital of Haryana is : ", capitals.get('Haryana'))
```



```
Capital of Haryana is :  None
```

- Python – Dictionaries
- Accessing Dictionary Items
- **Changing Dictionary Items**
- Add Dictionary Items
- Remove Dictionary Items
- Dictionary View Objects
- Loop Dictionaries
- Copy Dictionaries
- Nested Dictionaries
- Dictionary Methods



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

PYTHON – DICTIONARIES

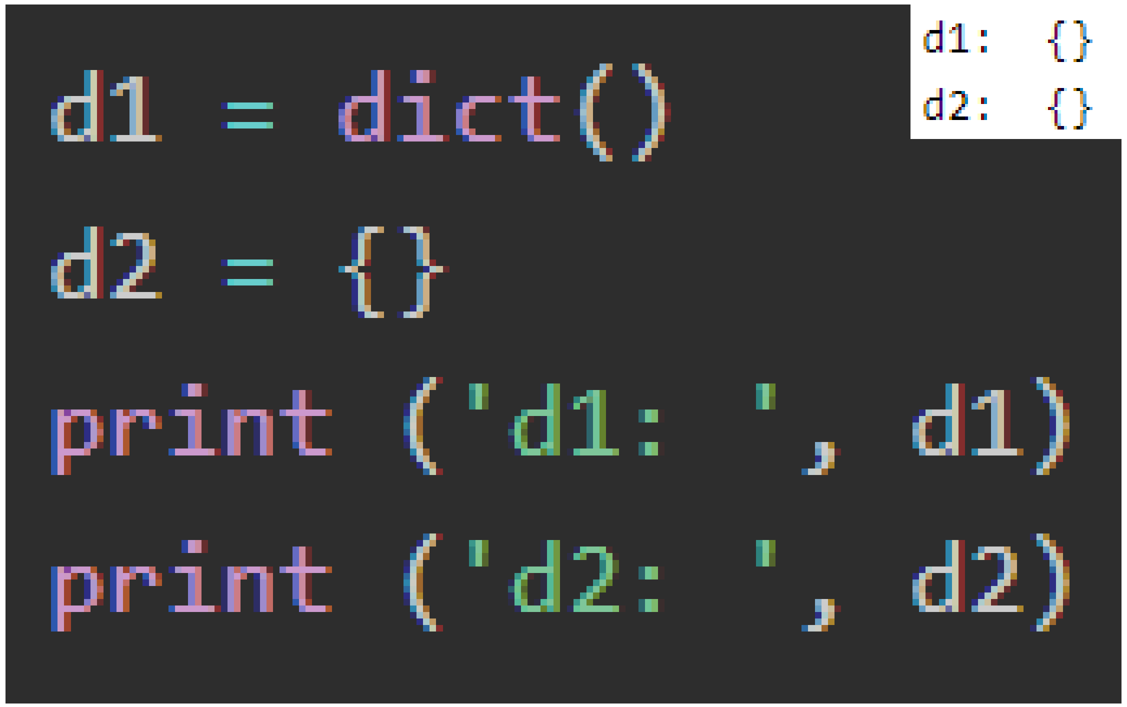
Course Unit 7: Week 9

CHANGE DICTIONARY ITEMS

❖Empty Dictionary

Using **dict()** function without any arguments creates an empty dictionary object.

It is equivalent to putting nothing between curly brackets.



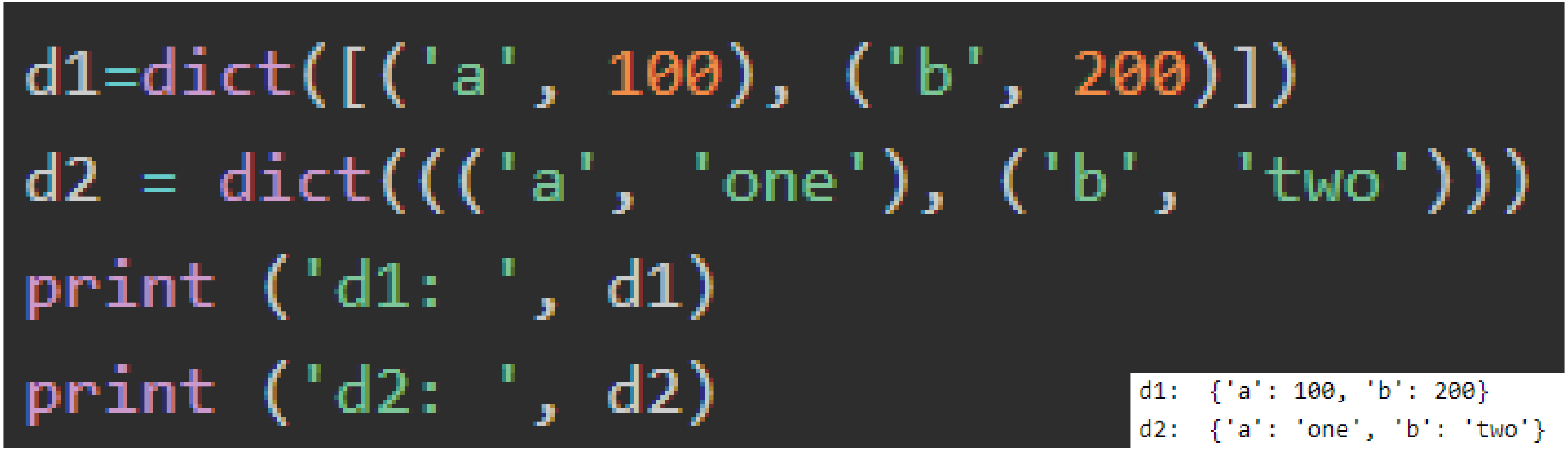
```
d1 = dict()
d2 = {}
print ('d1: ', d1)
print ('d2: ', d2)
```

```
d1:  {}
d2:  {}
```

CHANGE DICTIONARY ITEMS

❖Dictionary from List of Tuples

The **dict()** function constructs a dictionary from a list or tuple of two-item tuples. First item in a tuple is treated as key, and the second as its value.



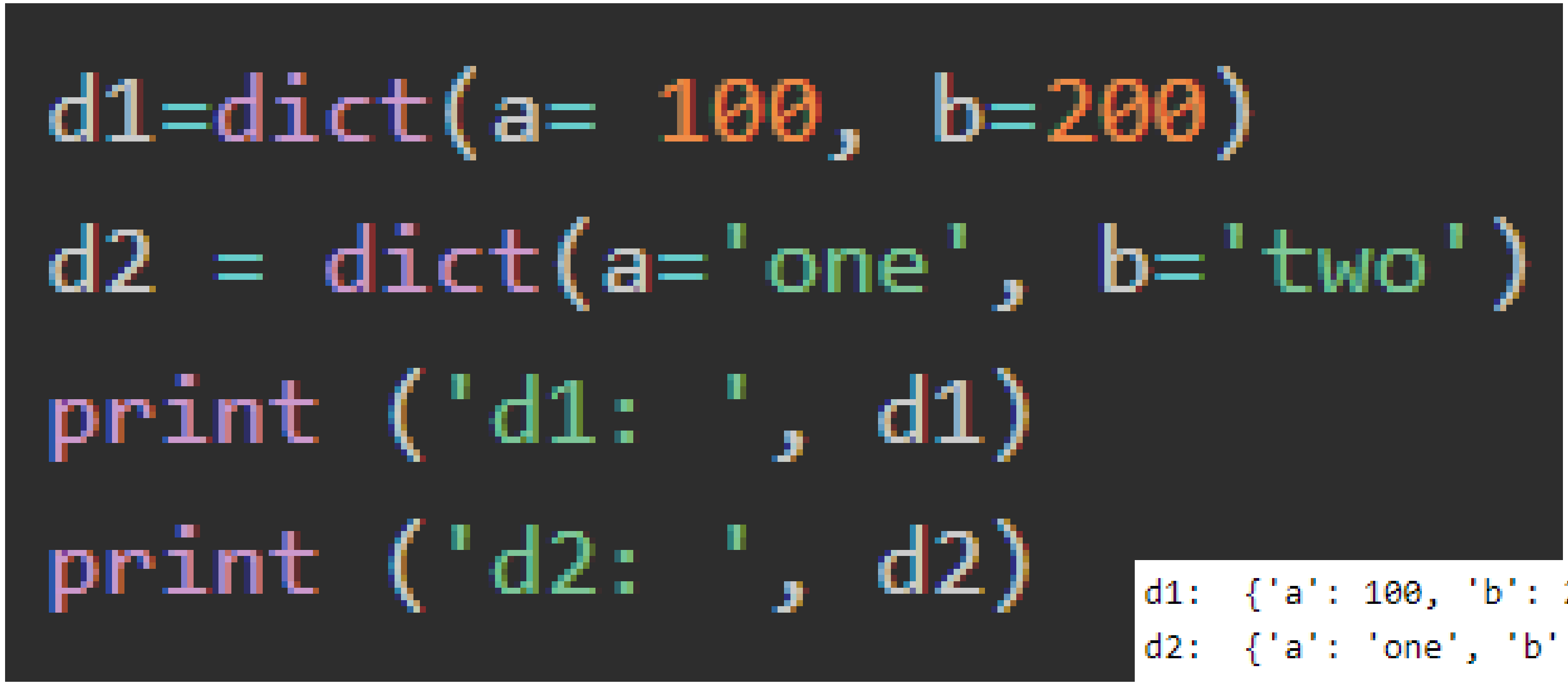
```python
d1=dict([('a', 100), ('b', 200)])
d2 = dict((('a', 'one'), ('b', 'two')))
print ('d1: ', d1)
print ('d2: ', d2)
```

```
d1:  {'a': 100, 'b': 200}
d2:  {'a': 'one', 'b': 'two'}
```

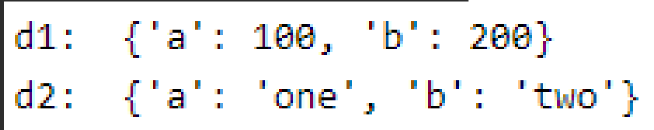
CHANGE DICTIONARY ITEMS

❖Dictionary from Keyword Arguments

The **dict()** function can take any number of keyword arguments with name=value pairs. It returns a dictionary object with the name as key and associates it to the value.

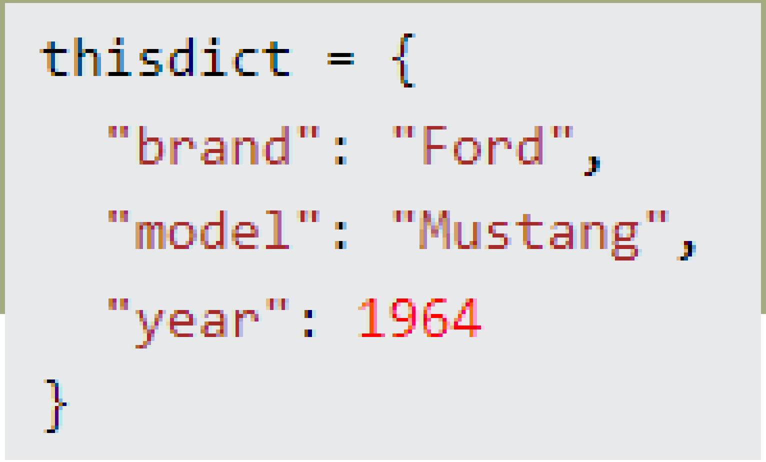


```
d1=dict(a= 100, b=200)
d2 = dict(a='one', b='two')
print ('d1: ', d1)
print ('d2: ', d2)
```



```
d1:  {'a': 100, 'b': 200}
d2:  {'a': 'one', 'b': 'two'}
```

- Python – Dictionaries
- Accessing Dictionary Items
- Changing Dictionary Items
- **Add Dictionary Items**
- Remove Dictionary Items
- Dictionary View Objects
- Loop Dictionaries
- Copy Dictionaries
- Nested Dictionaries
- Dictionary Methods



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

PYTHON – DICTIONARIES

Course Unit 7: Week 9

ADD DICTIONARY ITEMS

❖Using the Operator

The "[]" operator (used to access value mapped to a dictionary key) is used to update an existing key-value pair as well as add a new pair.

Syntax

```
dict["key"] = val
```

If the key is already present in the dictionary object, its value will be updated to val. If the key is not present in the dictionary, a new key-value pair will be added.

ADD DICTIONARY ITEMS

❖**Using the Operator**

```
marks = {"Savita":67, "Imtiaz":88, "Laxman":91, "David":49}
print ("marks dictionary before update: ", marks)
marks['Laxman'] = 95
print ("marks dictionary after update: ", marks)
```

```
marks dictionary before update: {'Savita': 67, 'Imtiaz': 88, 'Laxman': 91, 'David': 49}
marks dictionary after update: {'Savita': 67, 'Imtiaz': 88, 'Laxman': 95, 'David': 49}
```

ADD DICTIONARY ITEMS

❖**Using the Operator**

```
marks = {"Savita":67, "Imtiaz":88, "Laxman":91, "David":49}
print ("marks dictionary before update: ", marks)
marks['Krishan'] = 74
print ("marks dictionary after update: ", marks)
```

```
marks dictionary before update:  {'Savita': 67, 'Imtiaz': 88, 'Laxman': 91, 'David': 49}
marks dictionary after update:  {'Savita': 67, 'Imtiaz': 88, 'Laxman': 91, 'David': 49, 'Krishan': 74}
```

ADD DICTIONARY ITEMS

❖**Using the update() Method – Update with Another Dictionary**

the **update()** method's argument is another dictionary. Value of keys common in both dictionaries is updated. For new keys, key-value pair is added in the existing dictionary.

Syntax

d1.update(d2)

Return value

The existing dictionary is updated with new key-value pairs added to it.

ADD DICTIONARY ITEMS

❖**Using the update() Method – Update with Another Dictionary**

```
marks = {"Savita":67, "Imtiaz":88, "Laxman":91, "David":49}
print ("marks dictionary before update: \n", marks)
marks1 = {"Sharad": 51, "Mushtaq": 61, "Laxman": 89}
marks.update(marks1)
print ("marks dictionary after update: \n", marks)
```

```
marks dictionary before update:
{'Savita': 67, 'Imtiaz': 88, 'Laxman': 91, 'David': 49}
marks dictionary after update:
{'Savita': 67, 'Imtiaz': 88, 'Laxman': 89, 'David': 49, 'Sharad': 51, 'Mushtaq': 61}
```

ADD DICTIONARY ITEMS

❖**Using the update() Method – Update with Iterable**

If the argument to update() method is a list or tuple of two item tuples, an item each for it is added in the existing dictionary, or updated if the key is existing.

Syntax

d1.update([(k1, v1), (k2, v2)])

Return value

Existing dictionary is updated with new keys added.

ADD DICTIONARY ITEMS

❖**Using the update() Method – Update with Iterable**

```
marks = {"Savita":67, "Imtiaz":88, "Laxman":91, "David":49}
print ("marks dictionary before update: \n", marks)
marks1 = [("Sharad", 51), ("Mushtaq", 61), ("Laxman", 89)]
marks.update(marks1)
print ("marks dictionary after update: \n", marks)
```

```
marks dictionary before update:
 {'Savita': 67, 'Imtiaz': 88, 'Laxman': 91, 'David': 49}
marks dictionary after update:
 {'Savita': 67, 'Imtiaz': 88, 'Laxman': 89, 'David': 49, 'Sharad': 51, 'Mushtaq': 61}
```

ADD DICTIONARY ITEMS

❖**Using the update() Method – Update with Keyword Arguments**

Third version of update() method accepts list of keyword arguments in name=value format. New k-v pairs are added, or value of existing key is updated.

Syntax

d1.update(k1=v1, k2=v2)

Return value

Existing dictionary is updated with new key-value pairs added.

ADD DICTIONARY ITEMS

❖**Using the update() Method – Update with Keyword Arguments**

```
marks = {"Savita":67, "Imtiaz":88, "Laxman":91, "David":49}
print ("marks dictionary before update: \n", marks)
marks.update(Sharad = 51, Mushtaq = 61, Laxman = 89)
print ("marks dictionary after update: \n", marks)
```

```
marks dictionary before update:
 {'Savita': 67, 'Imtiaz': 88, 'Laxman': 91, 'David': 49}
marks dictionary after update:
 {'Savita': 67, 'Imtiaz': 88, 'Laxman': 89, 'David': 49, 'Sharad': 51, 'Mushtaq': 61}
```

ADD DICTIONARY ITEMS

❖Using the Unpack Operator

The "**" symbol prefixed to a dictionary object unpacks it to a list of tuples, each tuple with key and value. Two dict objects are unpacked and merged together and obtain a new dictionary.

Syntax

```
d3 = {**d1, **d2}
```

Return value

Two dictionaries are merged and a new object is returned.

ADD DICTIONARY ITEMS

❖**Using the Unpack Operator**

```
marks = {"Savita":67, "Imtiaz":88, "Laxman":91, "David":49}
print ("marks dictionary before update: \n", marks)
marks1 = {"Sharad": 51, "Mushtaq": 61, "Laxman": 89}
newmarks = {**marks, **marks1}
print ("marks dictionary after update: \n", newmarks)
```

```
marks dictionary before update:
 {'Savita': 67, 'Imtiaz': 88, 'Laxman': 91, 'David': 49}
marks dictionary after update:
 {'Savita': 67, 'Imtiaz': 88, 'Laxman': 89, 'David': 49, 'Sharad': 51, 'Mushtaq': 61}
```

ADD DICTIONARY ITEMS

❖Using the Union Operator (|)

Python introduces the "|" (pipe symbol) as the union operator for dictionary operands. It updates existing keys in dict object on left, and adds new key-value pairs to return a new dict object.

Syntax

d3 = d1 | d2

Return value

The Union operator return a new dict object after merging the two dict operands

ADD DICTIONARY ITEMS

❖**Using the Union Operator (|)**

```
marks = {"Savita":67, "Imtiaz":88, "Laxman":91, "David":49}
print ("marks dictionary before update: \n", marks)
marks1 = {"Sharad": 51, "Mushtaq": 61, "Laxman": 89}
newmarks = marks | marks1
print ("marks dictionary after update: \n", newmarks)
```

```
marks dictionary before update:
 {'Savita': 67, 'Imtiaz': 88, 'Laxman': 91, 'David': 49}
marks dictionary after update:
 {'Savita': 67, 'Imtiaz': 88, 'Laxman': 89, 'David': 49, 'Sharad': 51, 'Mushtaq': 61}
```

ADD DICTIONARY ITEMS

❖Using "|=" Operator

The "|=" operator is an augmented Union operator. It performs in-place update o n the dictionary operand on left by adding new keys in the operand on right, and updating the existing keys.

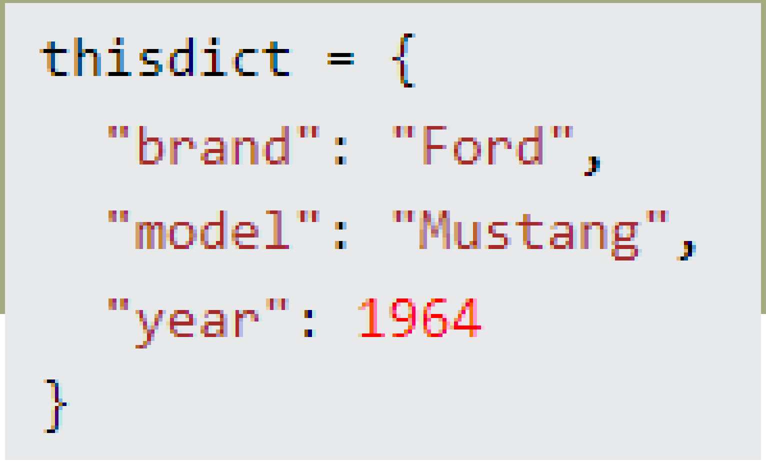
Syntax

```
d1 |= d2
```

```
marks = {"Savita":67, "Imtiaz":88, "Laxman":91, "David":49}
print ("marks dictionary before update: \n", marks)
marks1 = {"Sharad": 51, "Mushtaq": 61, "Laxman": 89}
marks |= marks1
print ("marks dictionary after update: \n", marks)
```

```
marks dictionary before update:
 {'Savita': 67, 'Imtiaz': 88, 'Laxman': 91, 'David': 49}
marks dictionary after update:
 {'Savita': 67, 'Imtiaz': 88, 'Laxman': 89, 'David': 49, 'Sharad': 51, 'Mushtaq': 61}
```

- Python – Dictionaries
- Accessing Dictionary Items
- Changing Dictionary Items
- Add Dictionary Items
- **Remove Dictionary Items**
- Dictionary View Objects
- Loop Dictionaries
- Copy Dictionaries
- Nested Dictionaries
- Dictionary Methods



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

PYTHON – DICTIONARIES

Course Unit 7: Week 9

REMOVE DICTIONARY ITEMS

❖Using del Keyword

Python's **del** keyword deletes any object from the memory. Here we use it to delete a key-value pair in a dictionary.

Syntax

del dict['key']

```
numbers = {10:"Ten", 20:"Twenty", 30:"Thirty",40:"Forty"}
print ("numbers dictionary before delete operation: \n", numbers)
del numbers[20]
print ("numbers dictionary before delete operation: \n", numbers)
```

```
numbers dictionary before delete operation:
 {10: 'Ten', 20: 'Twenty', 30: 'Thirty', 40: 'Forty'}
numbers dictionary before delete operation:
 {10: 'Ten', 30: 'Thirty', 40: 'Forty'}
```

REMOVE DICTIONARY ITEMS

❖Using pop() Method

The **pop()** method of dict class causes an element with the specified key to be removed from the dictionary.

Syntax

val = dict.pop(key)

Return value

The pop() method returns the value of the specified key after removing the key-value pair.

REMOVE DICTIONARY ITEMS

❖**Using pop() Method**

```
numbers = {10:"Ten", 20:"Twenty", 30:"Thirty",40:"Forty"}
print ("numbers dictionary before pop operation: \n", numbers)
val = numbers.pop(20)
print ("numbers dictionary after pop operation: \n", numbers)
print ("Value popped: ", val)
```

```
numbers dictionary before pop operation:
 {10: 'Ten', 20: 'Twenty', 30: 'Thirty', 40: 'Forty'}
numbers dictionary after pop operation:
 {10: 'Ten', 30: 'Thirty', 40: 'Forty'}
Value popped:  Twenty
```

REMOVE DICTIONARY ITEMS

❖**Using popitem() Method**

The **popitem()** method in dict() class doesn't take any argument. It pops out the last inserted key-value pair, and returns the same as a tuple

Syntax

val = dict.popitem()

Return Value

The popitem() method return a tuple contain key and value of the removed item from the dictionary

REMOVE DICTIONARY ITEMS

❖ **Using popitem() Method**

```
numbers = {10:"Ten", 20:"Twenty", 30:"Thirty",40:"Forty"}
print ("numbers dictionary before pop operation: \n", numbers)
val = numbers.popitem()
print ("numbers dictionary after pop operation: \n", numbers)
print ("Value popped: ", val)
```

```
numbers dictionary before pop operation:
 {10: 'Ten', 20: 'Twenty', 30: 'Thirty', 40: 'Forty'}
numbers dictionary after pop operation:
 {10: 'Ten', 20: 'Twenty', 30: 'Thirty'}
Value popped:  (40, 'Forty')
```

REMOVE DICTIONARY ITEMS

❖Using clear() Method

The **clear()** method in dict class removes all the elements from the dictionary object and returns an empty object.

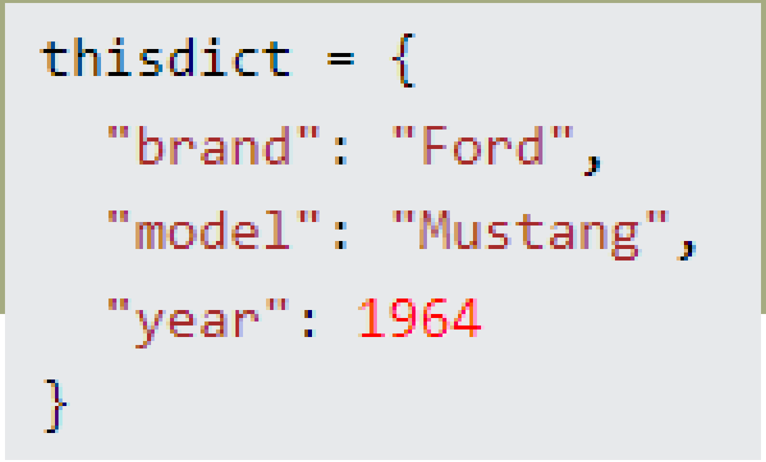
Syntax

dict.clear()

```
numbers = {10:"Ten", 20:"Twenty", 30:"Thirty",40:"Forty"}
print ("numbers dictionary before clear method: \n", numbers)
numbers.clear()
print ("numbers dictionary after clear method: \n", numbers)
```

```
numbers dictionary before clear method:
 {10: 'Ten', 20: 'Twenty', 30: 'Thirty', 40: 'Forty'}
numbers dictionary after clear method:
 {}
```

- Python – Dictionaries
- Accessing Dictionary Items
- Changing Dictionary Items
- Add Dictionary Items
- Remove Dictionary Items
- **Dictionary View Objects**
- Loop Dictionaries
- Copy Dictionaries
- Nested Dictionaries
- Dictionary Methods



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

PYTHON – DICTIONARIES

Course Unit 7: Week 9

DICTIONARY VIEW OBJECTS

- The items(), keys() and values() methods of **dict** class return view objects. These views are refreshed dynamically whenever any change occurs in the contents of their source dictionary object.

DICTIONARY VIEW OBJECTS

❖items() Method

The **items()** method returns a dict_items view object. It contains a list of tuples, each tuple made up of respective key, value pairs.

Syntax

Obj = dict.items()

Return value

The items() method returns dict_items object which is a dynamic view of (key,value) tuples.

DICTIONARY VIEW OBJECTS

- **items() Method**

```
numbers = {10:"Ten", 20:"Twenty", 30:"Thirty",40:"Forty"}
obj = numbers.items()
print ('type of obj: ', type(obj))
print (obj)
print ("update numbers dictionary")
numbers.update({50:"Fifty"})
print ("View automatically updated")
print (obj)
```

```
type of obj:  <class 'dict_items'>
dict_items([(10, 'Ten'), (20, 'Twenty'), (30, 'Thirty'), (40, 'Forty')])
update numbers dictionary
View automatically updated
dict_items([(10, 'Ten'), (20, 'Twenty'), (30, 'Thirty'), (40, 'Forty'), (50, 'Fifty')])
```

DICTIONARY VIEW OBJECTS

❖**keys() Method**

The **keys()** method of dict class returns dict_keys object which is a list of all keys defined in the dictionary. It is a view object, as it gets automatically updated whenever any update action is done on the dictionary object

Syntax

Obj = dict.keys()

Return value

The keys() method returns dict_keys object which is a view of keys in the dictionary.

DICTIONARY VIEW OBJECTS

❖keys() Method

```
numbers = {10:"Ten", 20:"Twenty", 30:"Thirty",40:"Forty"}
obj = numbers.keys()
print ('type of obj: ', type(obj))
print (obj)
print ("update numbers dictionary")
numbers.update({50:"Fifty"})
print ("View automatically updated")
print (obj)
```

```
type of obj: <class 'dict_keys'>
dict_keys([10, 20, 30, 40])
update numbers dictionary
View automatically updated
dict_keys([10, 20, 30, 40, 50])
```

DICTIONARY VIEW OBJECTS

❖values() Method

The values() method returns a view of all the values present in the dictionary. The object is of dict_value type, which gets automatically updated.

Syntax

Obj = dict.values()

Return value

The values() method returns a dict_values view of all the values present in the dictionary.

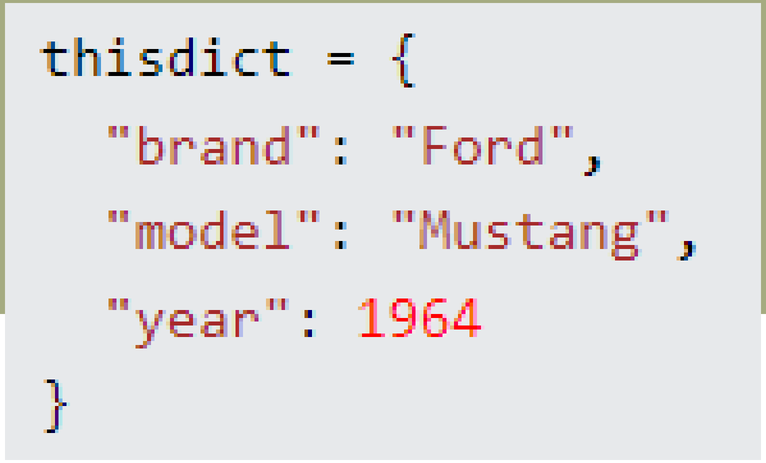
DICTIONARY VIEW OBJECTS

❖**values() Method**

```
numbers = {10:"Ten", 20:"Twenty", 30:"Thirty",40:"Forty"}
obj = numbers.values()
print ('type of obj: ', type(obj))
print (obj)
print ("update numbers dictionary")
numbers.update({50:"Fifty"})
print ("View automatically updated")
print (obj)
```

```
type of obj: <class 'dict_values'>
dict_values(['Ten', 'Twenty', 'Thirty', 'Forty'])
update numbers dictionary
View automatically updated
dict_values(['Ten', 'Twenty', 'Thirty', 'Forty', 'Fifty'])
```

- Python – Dictionaries
- Accessing Dictionary Items
- Changing Dictionary Items
- Add Dictionary Items
- Remove Dictionary Items
- Dictionary View Objects
- **Loop Dictionaries**
- Copy Dictionaries
- Nested Dictionaries
- Dictionary Methods



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

PYTHON – DICTIONARIES

Course Unit 7: Week 9

LOOP DICTIONARIES

Unlike a list, tuple or a string, dictionary data type in Python is not a sequence, as the items do not have a positional index. However, traversing a dictionary is still possible with different techniques.

LOOP DICTIONARIES

- Running a simple **for** loop over the dictionary object traverses the keys used in it.

```
numbers = {10:"Ten", 20:"Twenty", 30:"Thirty",40:"Forty"}
for x in numbers:
   print (x)
```

```
10
20
30
40
```

LOOP DICTIONARIES

- Once we are able to get the key, its associated value can be easily accessed either by **using square brackets** operator or with **get() method.**

```
numbers = {10:"Ten", 20:"Twenty", 30:"Thirty",40:"Forty"}
for x in numbers:
   print (x,":",numbers[x])
```

```
10 : Ten
20 : Twenty
30 : Thirty
40 : Forty
```

LOOP DICTIONARIES

The items(), keys() and values() methods of dict class return the view objects dict_items, dict_keys and dict_values respectively. These objects are iterators, and hence we can run a for loop over them.

LOOP DICTIONARIES

❖The **dict_items** object is a list of key-value tuples over which a for loop can be run as follows:

```
numbers = {10:"Ten", 20:"Twenty", 30:"Thirty",40:"Forty"}
for x in numbers.items():
   print (x)
```

```
(10, 'Ten')
(20, 'Twenty')
(30, 'Thirty')
(40, 'Forty')
```

LOOP DICTIONARIES

- On previous, "x" is the tuple element from the dict_items iterator. We can further unpack this tuple in two different variables.

```
numbers = {10:"Ten", 20:"Twenty", 30:"Thirty",40:"Forty"}
for x,y in numbers.items():
   print (x,":", y)
```

```
10 : Ten
20 : Twenty
30 : Thirty
40 : Forty
```

LOOP DICTIONARIES

❖Similarly, the collection of keys in dict_keys object can be iterated over.

```
numbers = {10:"Ten", 20:"Twenty", 30:"Thirty",40:"Forty"}
for x in numbers.keys():
   print (x, ":", numbers[x])
```

```
10 : Ten
20 : Twenty
30 : Thirty
40 : Forty
```

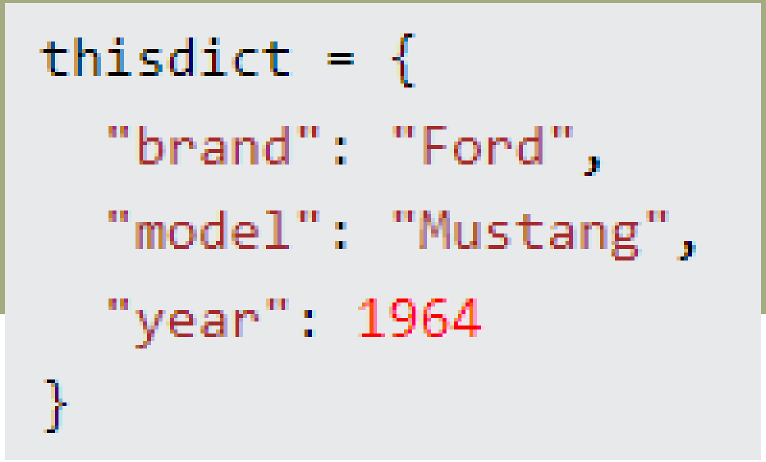
LOOP DICTIONARIES

- Respective Keys and values in dict_keys and dict_values are at same index. In the following example, we have a for loop that runs from 0 to the length of the dict, and use the looping variable as index and print key and its corresponding value.

```
numbers = {10:"Ten", 20:"Twenty", 30:"Thirty",40:"Forty"}
l = len(numbers)
for x in range(l):
   print (list(numbers.keys())[x], ":", list(numbers.values())[x])
```

```
10 : Ten
20 : Twenty
30 : Thirty
40 : Forty
```

- Python – Dictionaries
- Accessing Dictionary Items
- Changing Dictionary Items
- Add Dictionary Items
- Remove Dictionary Items
- Dictionary View Objects
- Loop Dictionaries
- **Copy Dictionaries**
- Nested Dictionaries
- Dictionary Methods



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

PYTHON – DICTIONARIES

Course Unit 7: Week 9

COPY DICTIONARIES

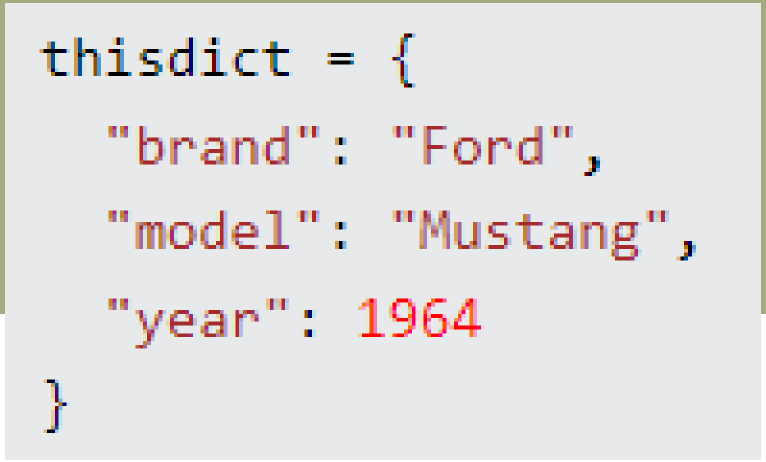
- use the copy() method instead of assignment.

```
d1 = {"a":11, "b":22, "c":33}
d2 = d1.copy()
print ("id:", id(d1), "dict: ",d1)
print ("id:", id(d2), "dict: ",d2)
d1["b"] = 100
print ("id:", id(d1), "dict: ",d1)
print ("id:", id(d2), "dict: ",d2)
```

When "d1" is updated, "d2" will not change now because "d2" is the copy of dictionary object, not merely a reference.

```
id: 1586671734976 dict: {'a': 11, 'b': 22, 'c': 33}
id: 1586673973632 dict: {'a': 11, 'b': 22, 'c': 33}
id: 1586671734976 dict: {'a': 11, 'b': 100, 'c': 33}
id: 1586673973632 dict: {'a': 11, 'b': 22, 'c': 33}
```

- Python – Dictionaries
- Accessing Dictionary Items
- Changing Dictionary Items
- Add Dictionary Items
- Remove Dictionary Items
- Dictionary View Objects
- Loop Dictionaries
- Copy Dictionaries
- **Nested Dictionaries**
- Dictionary Methods



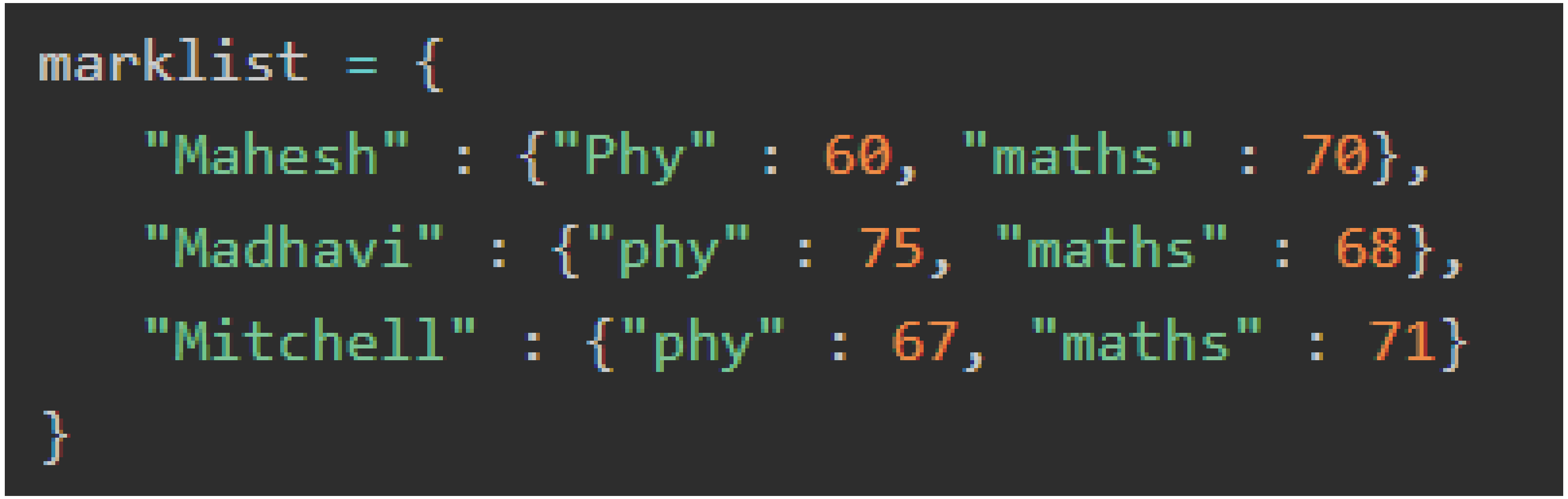
```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
```

PYTHON – DICTIONARIES

Course Unit 7: Week 9

NESTED DICTIONARIES

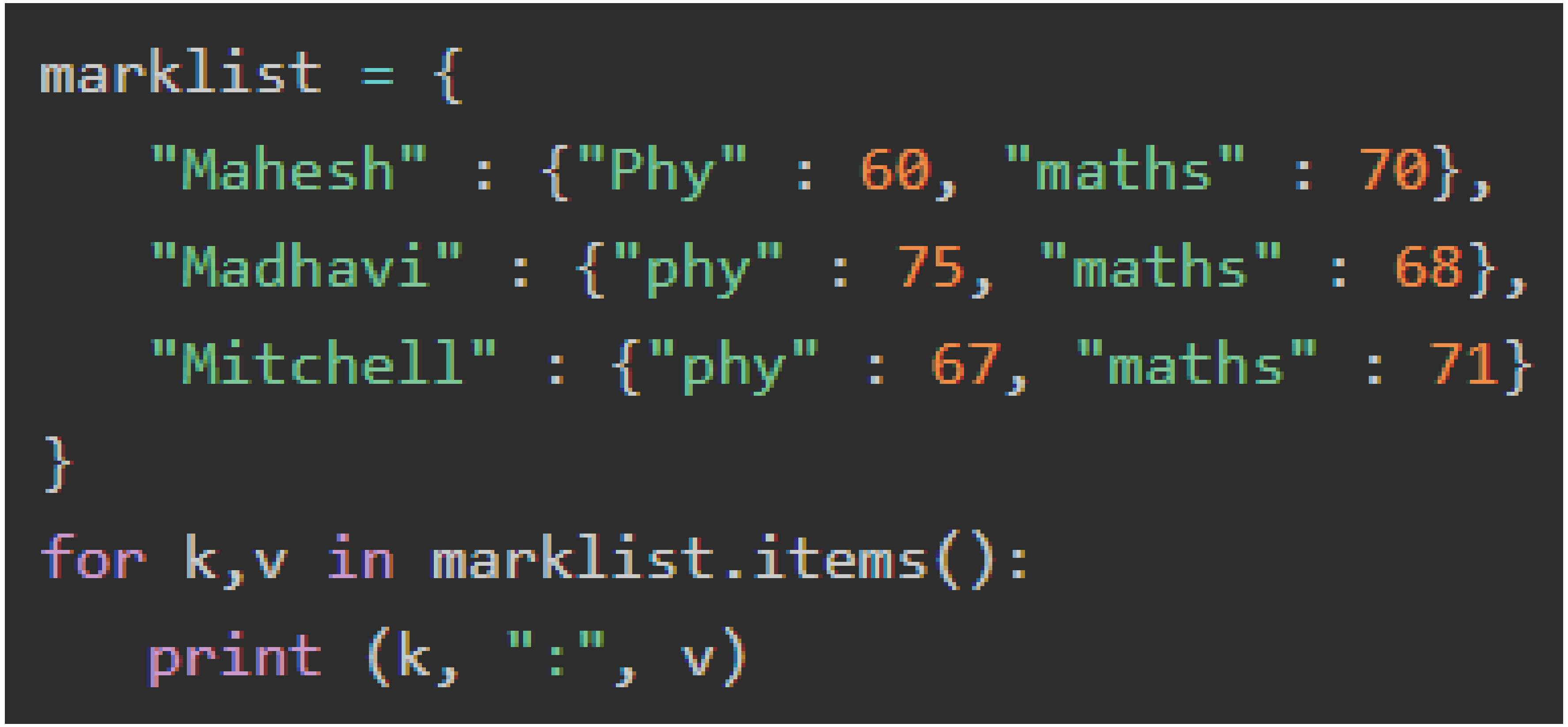
A Python dictionary is said to have a nested structure if value of one or more keys is another dictionary. A nested dictionary is usually employed to store a complex data structure.



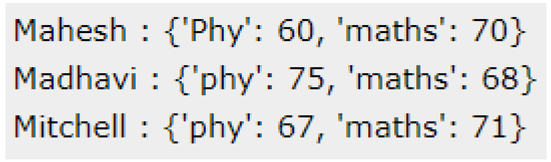
```
marklist = {
   "Mahesh" : {"Phy" : 60, "maths" : 70},
   "Madhavi" : {"phy" : 75, "maths" : 68},
   "Mitchell" : {"phy" : 67, "maths" : 71}
}
```

NESTED DICTIONARIES

❖constitute a for loop to traverse nested dictionary, as in the previous section.



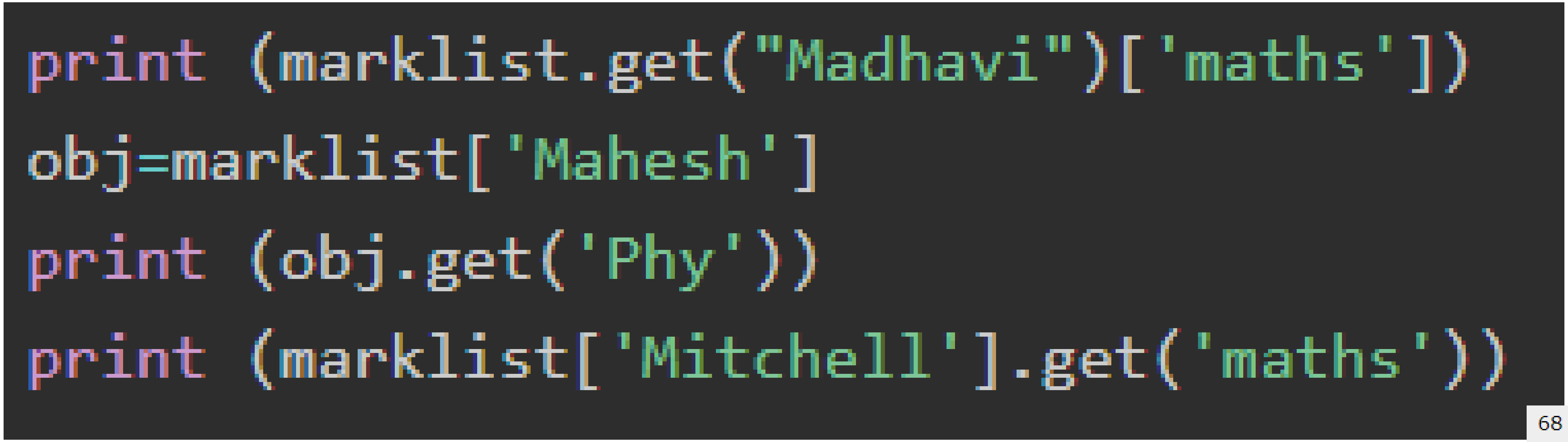
```
marklist = {
    "Mahesh" : {"Phy" : 60, "maths" : 70},
    "Madhavi" : {"phy" : 75, "maths" : 68},
    "Mitchell" : {"phy" : 67, "maths" : 71}
}
for k,v in marklist.items():
    print (k, ":", v)
```



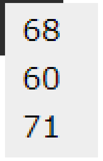
```
Mahesh : {'Phy': 60, 'maths': 70}
Madhavi : {'phy': 75, 'maths': 68}
Mitchell : {'phy': 67, 'maths': 71}
```

NESTED DICTIONARIES

- It is possible to access value from an inner dictionary with [] notation or get() method.

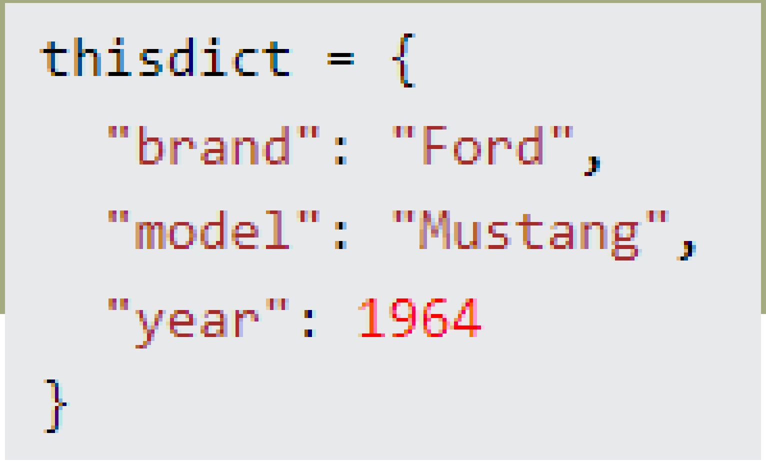


```
print (marklist.get("Madhavi")['maths'])
obj=marklist['Mahesh']
print (obj.get('Phy'))
print (marklist['Mitchell'].get('maths'))
```



```
68
60
71
```

- Python – Dictionaries
- Accessing Dictionary Items
- Changing Dictionary Items
- Add Dictionary Items
- Remove Dictionary Items
- Dictionary View Objects
- Loop Dictionaries
- Copy Dictionaries
- Nested Dictionaries
- **Dictionary Methods**

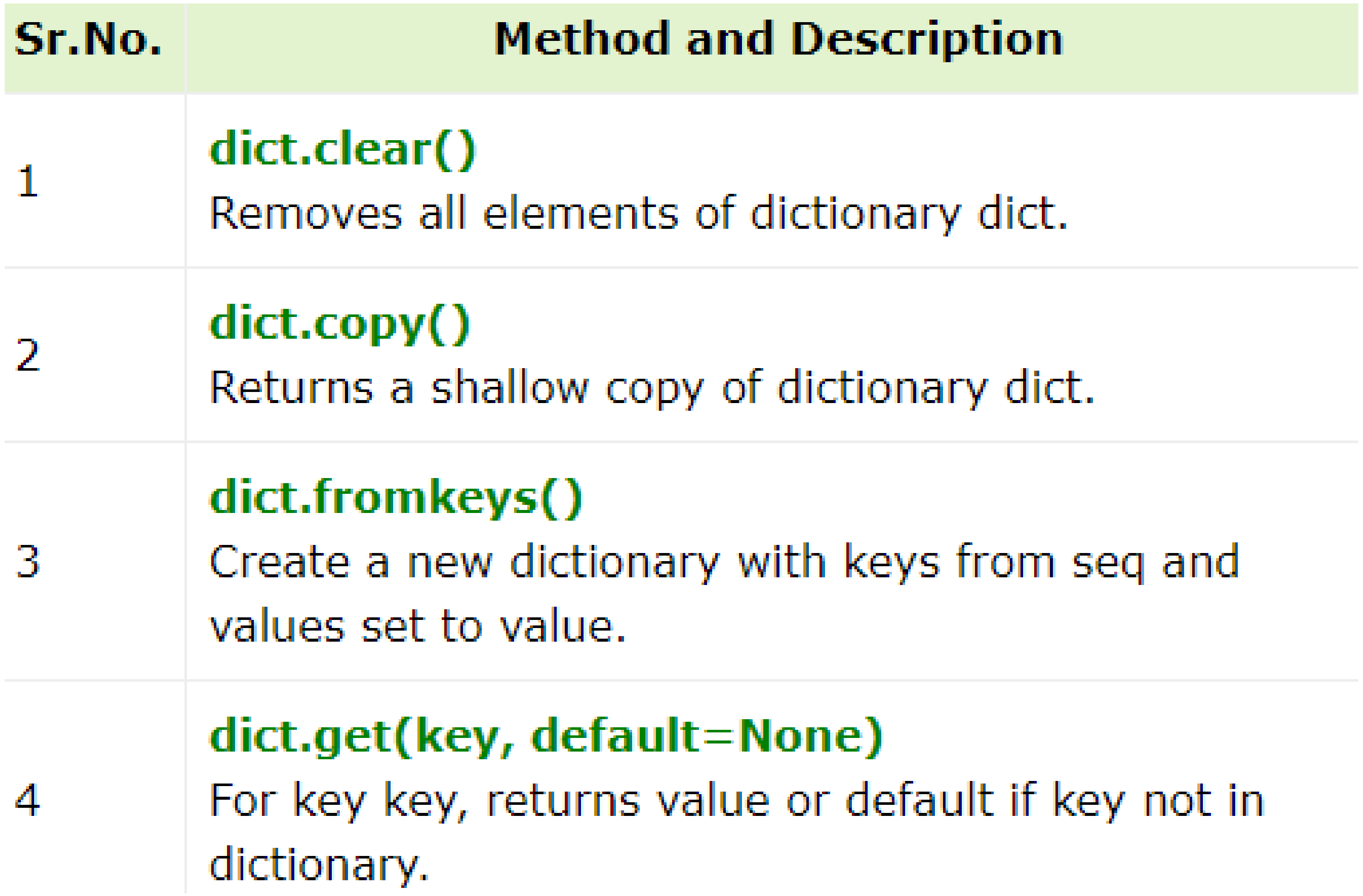


```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
```

PYTHON – DICTIONARIES

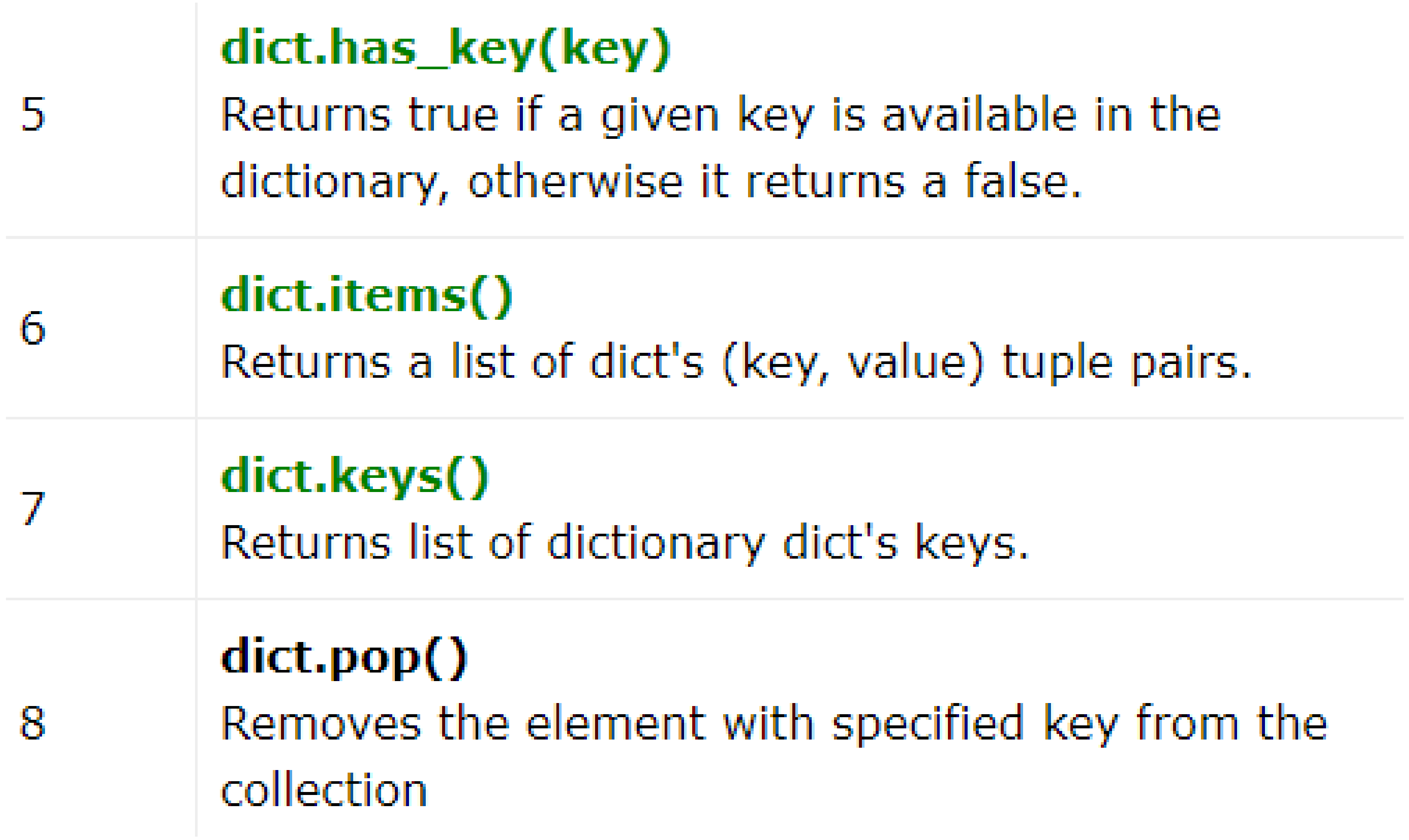
Course Unit 7: Week 9

DICTIONARY METHODS



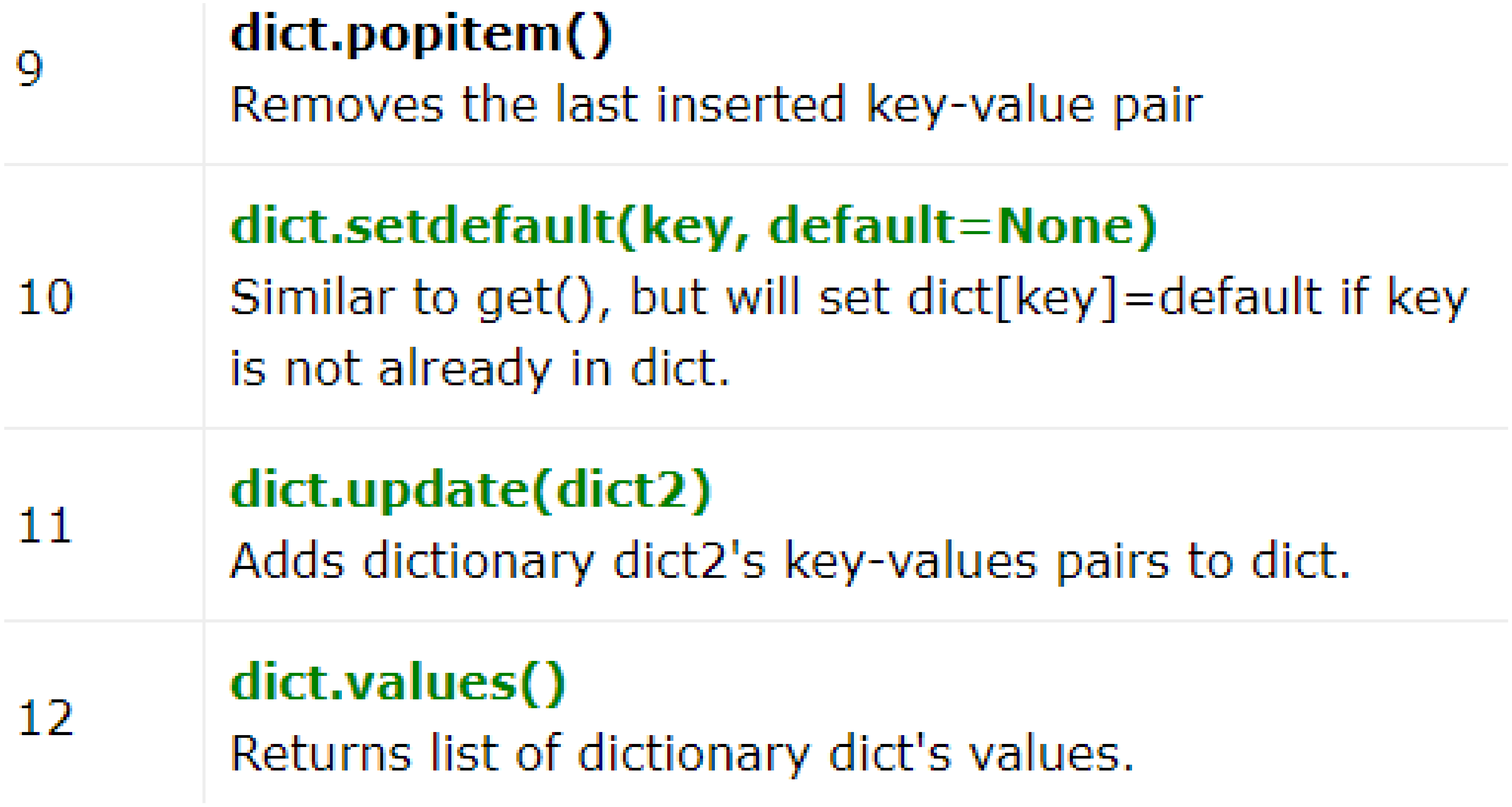
Sr.No.	Method and Description
1	**dict.clear()** Removes all elements of dictionary dict.
2	**dict.copy()** Returns a shallow copy of dictionary dict.
3	**dict.fromkeys()** Create a new dictionary with keys from seq and values set to value.
4	**dict.get(key, default=None)** For key key, returns value or default if key not in dictionary.

DICTIONARY METHODS

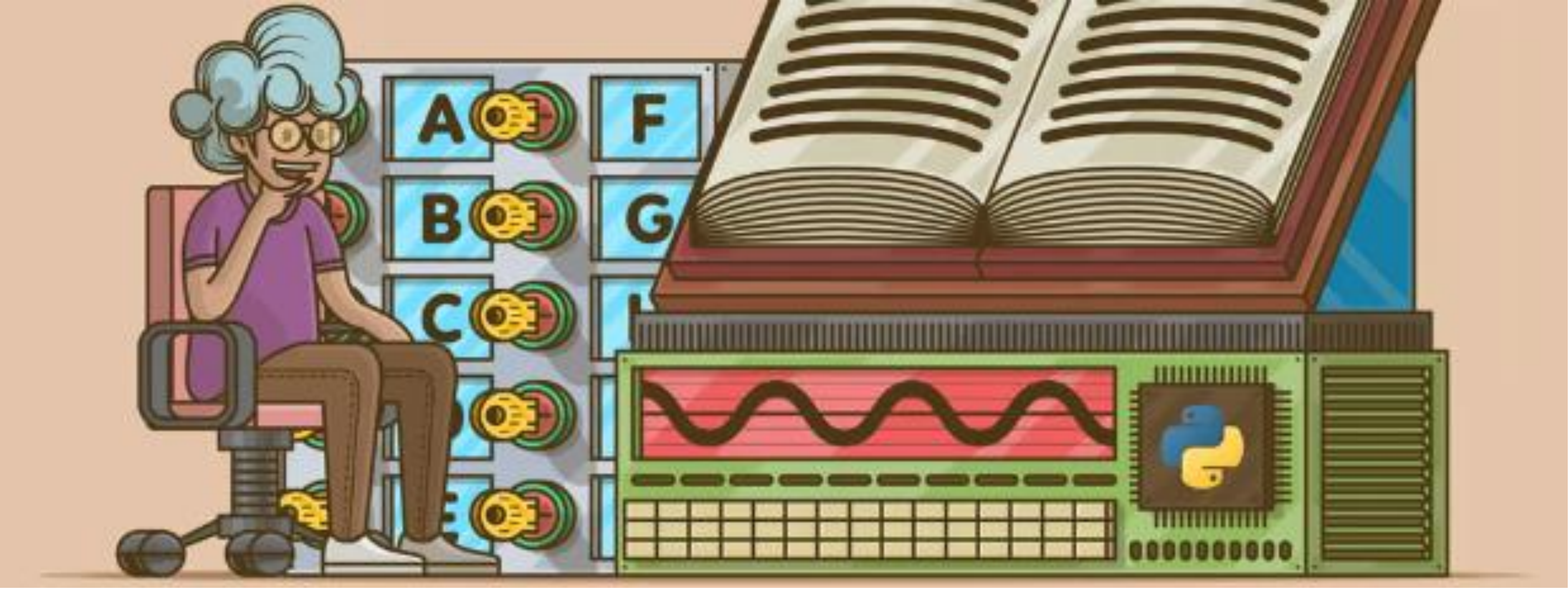


5	**dict.has_key(key)** Returns true if a given key is available in the dictionary, otherwise it returns a false.
6	**dict.items()** Returns a list of dict's (key, value) tuple pairs.
7	**dict.keys()** Returns list of dictionary dict's keys.
8	**dict.pop()** Removes the element with specified key from the collection

DICTIONARY METHODS



9	**dict.popitem()** Removes the last inserted key-value pair
10	**dict.setdefault(key, default=None)** Similar to get(), but will set dict[key]=default if key is not already in dict.
11	**dict.update(dict2)** Adds dictionary dict2's key-values pairs to dict.
12	**dict.values()** Returns list of dictionary dict's values.



References

learn python programming. (2023). https://www.tutorialsteacher.com/python
python tutorial. (2022). https://www.w3resource.com/python/python-tutorial.php
python tutorial. (n.d.). https://www.tutorialspoint.com/python/index.htm
python tutorial. (n.d.). https://www.w3schools.com/python/default.asp