Data Structure: In computer science, a data structure is a data organization, management, and storage format that enables efficient access and modification.

Data structures are two types.

1. Primitive
    - Integer numbers.
    - Floating-point numbers
    - Fixed-point numbers.
    - Booleans.
    - Characters and strings.
    - Numeric data type ranges.

2. Non-Primitive
    - Array
    - List
    - Tuple
    - Set
    - Dictionary
    - File

Here, We will talk about Non-Primitive data types mainly.

List

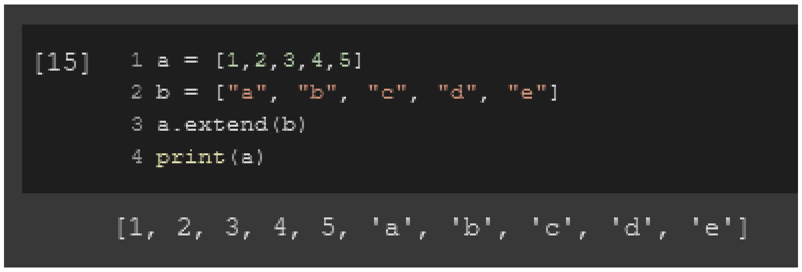
A list is an ordered and mutable Python container. To create a list, the elements are placed inside square brackets “ [ ] ”, separated by commas. Index starts from 0!

EG: list = [“A”, ”B”, 3, 5, 6]
print(list(2)) = 3 ; print(list(0)) = A

• To sort a list, use “.sort()” method.
list.sort() => list = [3, 5, 6, “A”, ”B” “P”]

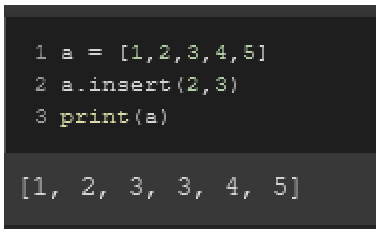
• To insert an element in a list, list.append(“P”)
list = [“A”, ”B”, 3, 5, 6, “P”]

• To add one list with another list, We use list.extend(list_2)



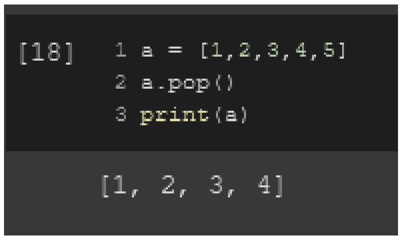
```
[15]  a = [1,2,3,4,5]
      b = ["a", "b", "c", "d", "e"]
      a.extend(b)
      print(a)

    [1, 2, 3, 4, 5, 'a', 'b', 'c', 'd', 'e']
```

• To insert an element at a specific indexed place of a list, We use list.insert(new_item,index_number)



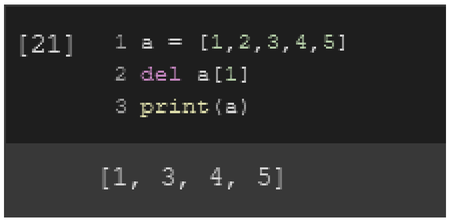
```
a = [1,2,3,4,5]
a.insert(2,3)
print(a)

[1, 2, 3, 3, 4, 5]
```

• If we want to delete an element from the last point of a list, We can use pop method.



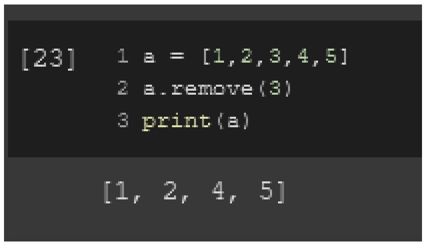
```
[18]  a = [1,2,3,4,5]
      a.pop()
      print(a)

    [1, 2, 3, 4]
```

• If we want to delete an element from the last point of a list following index, We use del



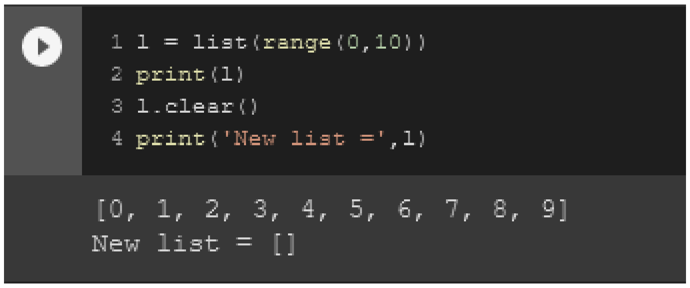
```
[21]  a = [1,2,3,4,5]
      del a[1]
      print(a)

      [1, 3, 4, 5]
```

• If we want to delete an element from the last point of a list element wise, We use remove



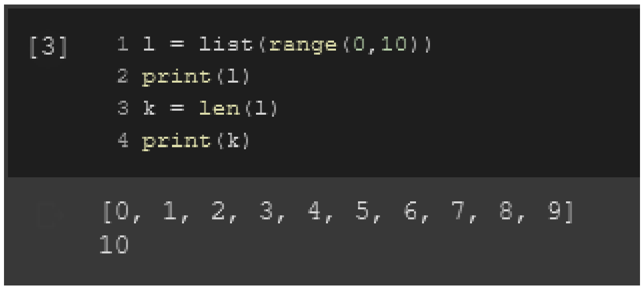
```
[23]  a = [1,2,3,4,5]
      a.remove(3)
      print(a)

      [1, 2, 4, 5]
```

• to clear a list, We use l.clear() [l = list name]



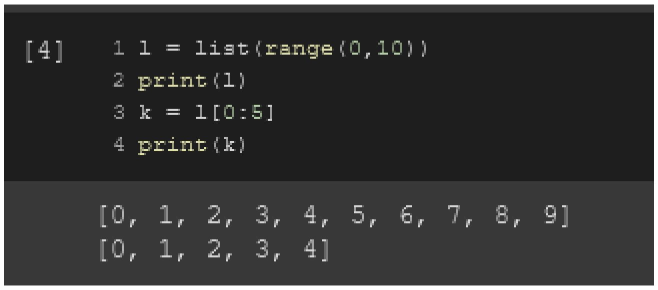
```
l = list(range(0,10))
print(l)
l.clear()
print('New list =',l)

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
New list = []
```

• to calculate the length of the list, we use len(list)



```
[3]  l = list(range(0,10))
     print(l)
     k = len(l)
     print(k)

     [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
     10
```

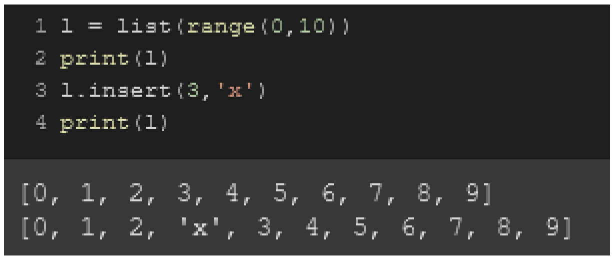
• to slice a list, we use list_name(start_index : finishing_index)



```
[4]  l = list(range(0,10))
     print(l)
     k = l[0:5]
     print(k)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[0, 1, 2, 3, 4]
```

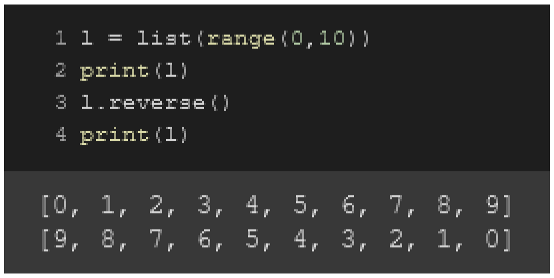
• to insert an element in a specific position or index, we use list_name(index, “element”)



```
l = list(range(0,10))
print(l)
l.insert(3,'x')
print(l)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[0, 1, 2, 'x', 3, 4, 5, 6, 7, 8, 9]
```

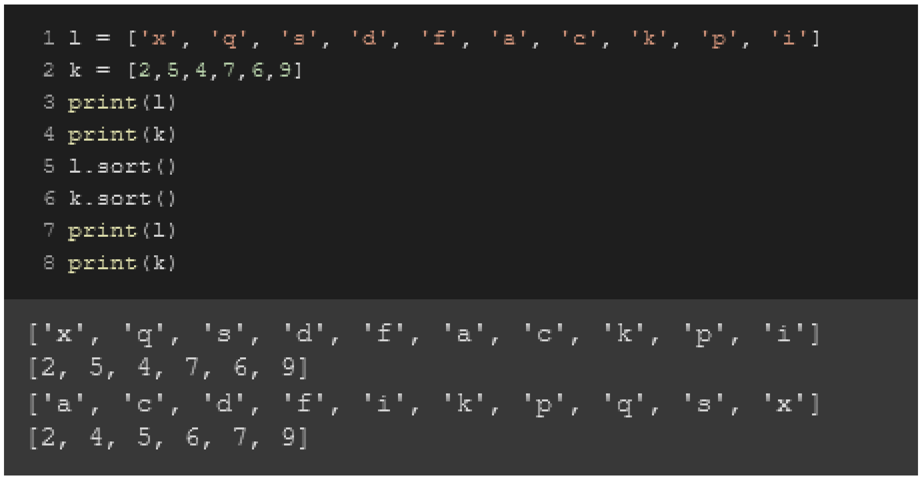
• to reverse a list, we use list_name.reverse()



```
l = list(range(0,10))
print(l)
l.reverse()
print(l)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
```

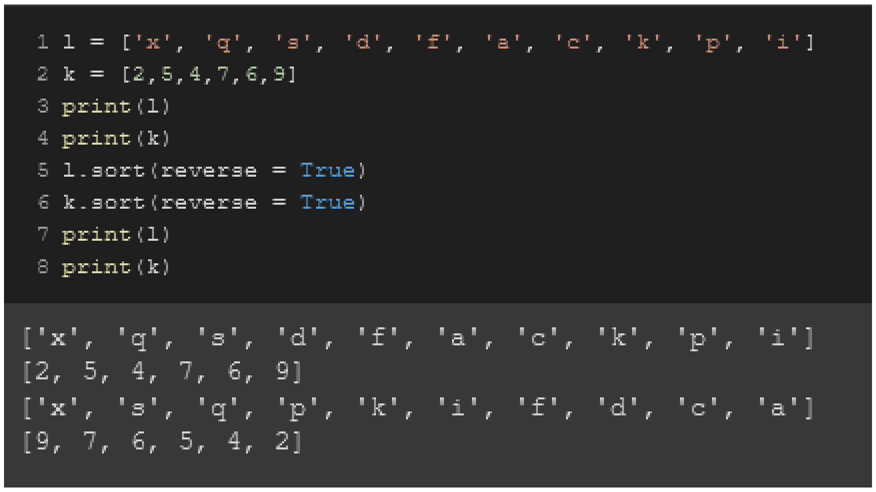
• to sort a list, we use list_name.sort()



```
l = ['x', 'q', 's', 'd', 'f', 'a', 'c', 'k', 'p', 'i']
k = [2,5,4,7,6,9]
print(l)
print(k)
l.sort()
k.sort()
print(l)
print(k)
```

```
['x', 'q', 's', 'd', 'f', 'a', 'c', 'k', 'p', 'i']
[2, 5, 4, 7, 6, 9]
['a', 'c', 'd', 'f', 'i', 'k', 'p', 'q', 's', 'x']
[2, 4, 5, 6, 7, 9]
```

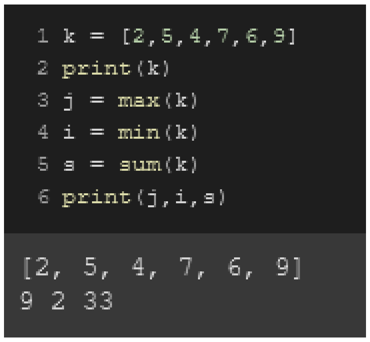
• to sort a list in reverse order, we use list_name.sort(reverse = True)



```
l = ['x', 'q', 's', 'd', 'f', 'a', 'c', 'k', 'p', 'i']
k = [2,5,4,7,6,9]
print(l)
print(k)
l.sort(reverse = True)
k.sort(reverse = True)
print(l)
print(k)
```

```
['x', 'q', 's', 'd', 'f', 'a', 'c', 'k', 'p', 'i']
[2, 5, 4, 7, 6, 9]
['x', 's', 'q', 'p', 'k', 'i', 'f', 'd', 'c', 'a']
[9, 7, 6, 5, 4, 2]
```

• to find the maximum value, minimum value and sum of all the elements in a list, we respectively use max(list_name), min(list_name), sum(list_name)

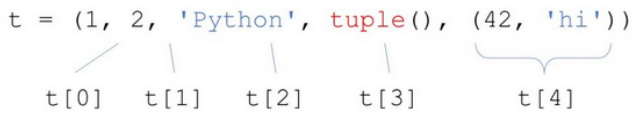


```
k = [2,5,4,7,6,9]
print(k)
j = max(k)
i = min(k)
s = sum(k)
print(j,i,s)
```

```
[2, 5, 4, 7, 6, 9]
9 2 33
```

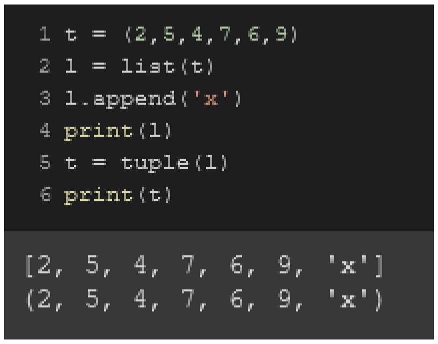
Tuple

Python tuples are a data structure that store an ordered sequence of values. Tuples are immutable. This means you cannot change the values in a tuple. Tuples are defined with parenthesis. Tuples are a core data structure in Python.
Tuples placed inside round brackets “ ( ) ”, separated by commas. Index starts from 0!



```
t = (1, 2, 'Python', tuple(), (42, 'hi'))
```

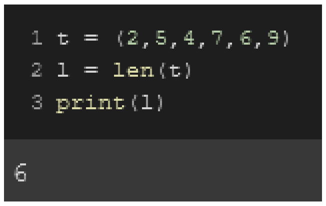
• Tuples are immutable, to insert a new element in a tuple, convert it into a list first and then add the element in the list and convert it back to tuple again.



```
t = (2,5,4,7,6,9)
l = list(t)
l.append('x')
print(l)
t = tuple(l)
print(t)
```

```
[2, 5, 4, 7, 6, 9, 'x']
(2, 5, 4, 7, 6, 9, 'x')
```

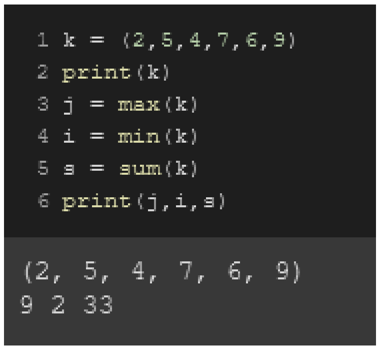
• to calculate the length of a tuple, we use len(tuple)



```
t = (2,5,4,7,6,9)
l = len(t)
print(l)
```

```
6
```

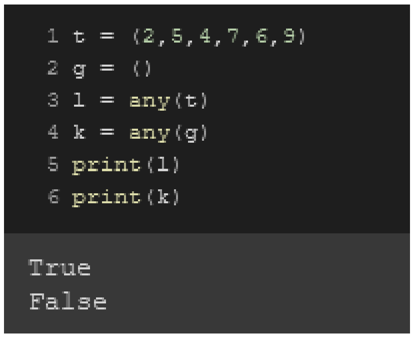
• to find the maximum value, minimum value and sum of all the elements in a tuple, we respectively use max(tuple_name), min(tuple_name), sum(tuple_name)



```
k = (2,5,4,7,6,9)
print(k)
j = max(k)
i = min(k)
s = sum(k)
print(j,i,s)
```

```
(2, 5, 4, 7, 6, 9)
9 2 33
```

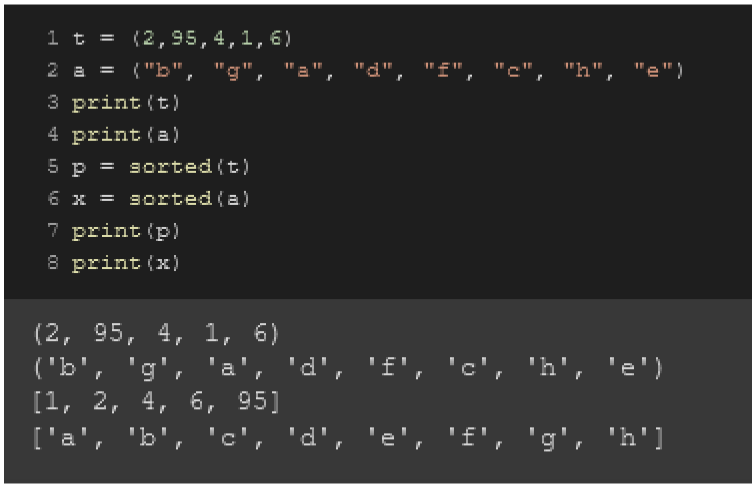
• any() function tells if a tuple is filled or empty. Prints “True” & “False” for respectively filled and empty tuples.



```
t = (2,5,4,7,6,9)
g = ()
l = any(t)
k = any(g)
print(l)
print(k)
```

```
True
False
```

• to sort a tuple, we use sorted(tuple)



```
t = (2,95,4,1,6)
a = ("b", "g", "a", "d", "f", "c", "h", "e")
print(t)
print(a)
p = sorted(t)
x = sorted(a)
print(p)
print(x)
```

```
(2, 95, 4, 1, 6)
('b', 'g', 'a', 'd', 'f', 'c', 'h', 'e')
[1, 2, 4, 6, 95]
['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h']
```

SET

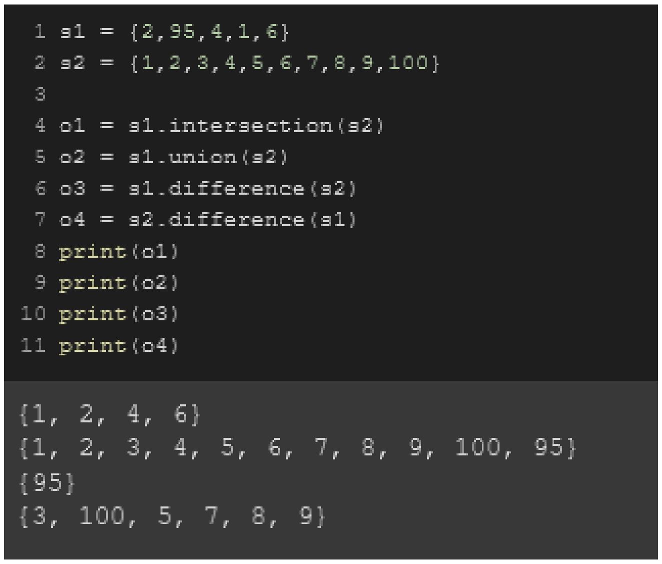
Set does not support duplicate elements. Set placed inside curly brackets “ { } ”, separated by commas. Index starts from 0!

These are operations we can do using set.

Operation on Set:

1. Access Items
2. Add Items
3. Remove Items
4. Join Set
5. Set Methods
6. Loop Set

In between 2 sets, We can do intersection, union and difference operation using “s1.intersection(s2)”, “s1.union(s2)”, “s1.difference(s2)” respectively.

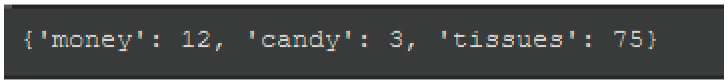


```
s1 = {2,95,4,1,6}
s2 = {1,2,3,4,5,6,7,8,9,100}

o1 = s1.intersection(s2)
o2 = s1.union(s2)
o3 = s1.difference(s2)
o4 = s2.difference(s1)
print(o1)
print(o2)
print(o3)
print(o4)
```

```
{1, 2, 4, 6}
{1, 2, 3, 4, 5, 6, 7, 8, 9, 100, 95}
{95}
{3, 100, 5, 7, 8, 9}
```

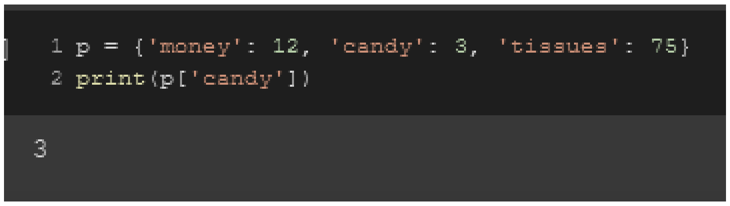
Dictionary

Dictionary is basically set with keys along with values. Same as set, dictionary placed inside curly brackets “ { } ”, separated by commas. Index starts from 0!
A dictionary has a key and a corresponding value which are stored in {key1:value1, key2:value2} format



```
{'money': 12, 'candy': 3, 'tissues': 75}
```

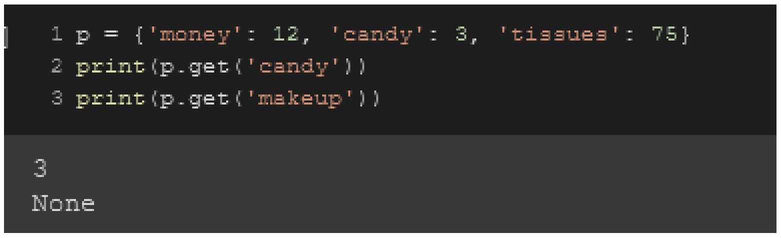
• To recall a value from the dictionary, We use the key name to call out the value.



```
p = {'money': 12, 'candy': 3, 'tissues': 75}
print(p['candy'])
```

```
3
```

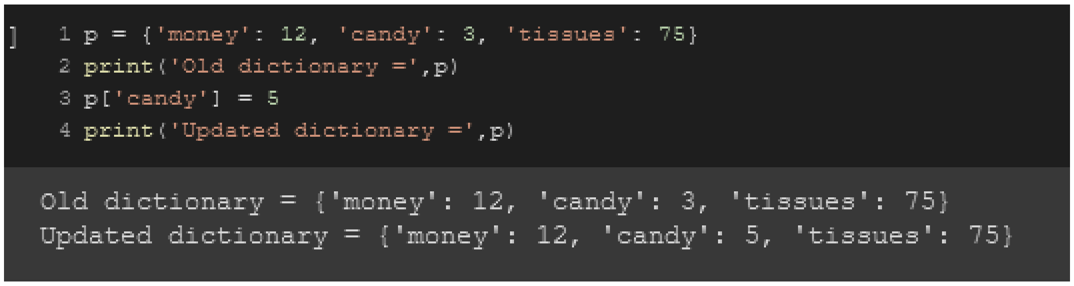
• To see if a key exists in a dictionary, we can use the “get()” function.



```
p = {'money': 12, 'candy': 3, 'tissues': 75}
print(p.get('candy'))
print(p.get('makeup'))
```

```
3
None
```

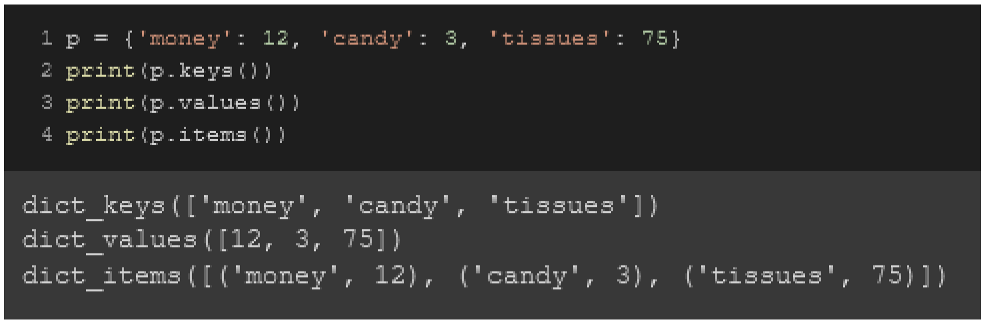
• To keywise update a dictionary, we use dictionary_name[‘key’] = new_value



```
p = {'money': 12, 'candy': 3, 'tissues': 75}
print('Old dictionary =',p)
p['candy'] = 5
print('Updated dictionary =',p)
```

```
Old dictionary = {'money': 12, 'candy': 3, 'tissues': 75}
Updated dictionary = {'money': 12, 'candy': 5, 'tissues': 75}
```

• To see all the keys, values & key-values pairs of a dictionary, we print keys(), values(), items functions respectively



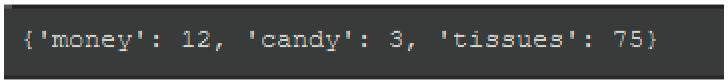
```python
p = {'money': 12, 'candy': 3, 'tissues': 75}
print(p.keys())
print(p.values())
print(p.items())
```

```
dict_keys(['money', 'candy', 'tissues'])
dict_values([12, 3, 75])
dict_items([('money', 12), ('candy', 3), ('tissues', 75)])
```

.

```
purse = dict()
purse['money'] = 12
purse['candy'] = 3
purse['tissues'] = 75
print(purse)
```

Output:-

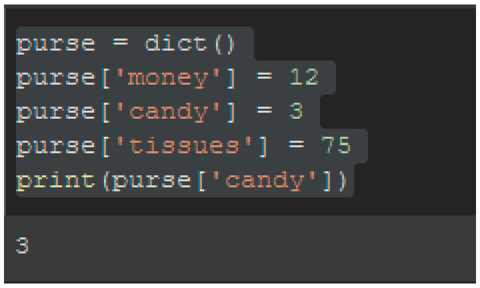


```
{'money': 12, 'candy': 3, 'tissues': 75}
```

Code:-

```
purse = dict()
purse['money'] = 12
purse['candy'] = 3
purse['tissues'] = 75
print(purse['candy'])
```

Output:-

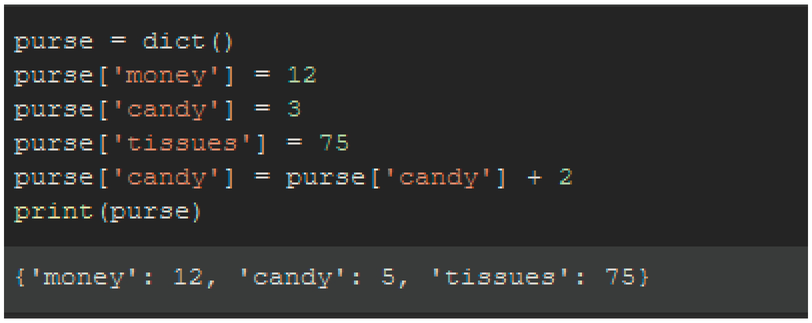


```
purse = dict()
purse['money'] = 12
purse['candy'] = 3
purse['tissues'] = 75
print(purse['candy'])

3
```

Code:-

```
purse = dict()
purse['money'] = 12
purse['candy'] = 3
purse['tissues'] = 75
purse['candy'] = purse['candy'] + 2
print(purse)
```

Output:-



```
purse = dict()
purse['money'] = 12
purse['candy'] = 3
purse['tissues'] = 75
purse['candy'] = purse['candy'] + 2
print(purse)
```

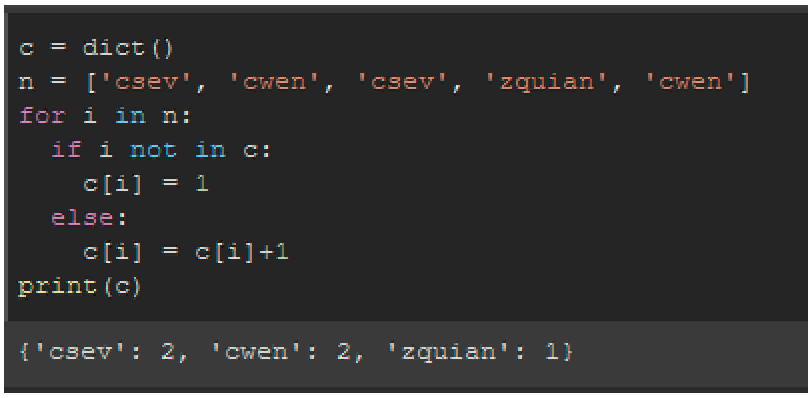
```
{'money': 12, 'candy': 5, 'tissues': 75}
```

(Using Dictionaries)

Code:-

```
c = dict()
n = ['csev', 'cwen', 'csev', 'zquian', 'cwen']
for i in n:
  if i not in c:
    c[i] = 1
  else:
    c[i] = c[i]+1
print(c)
```

Output:-



```
c = dict()
n = ['csev', 'cwen', 'csev', 'zquian', 'cwen']
for i in n:
  if i not in c:
    c[i] = 1
  else:
    c[i] = c[i]+1
print(c)
```

```
{'csev': 2, 'cwen': 2, 'zquian': 1}
```

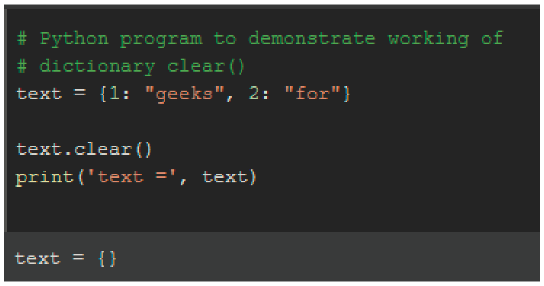
#4 (Clearing a dictionary)

Let there be an already occupied dictionary which needs to be cleared, Using the **CLEAR()** method, We can empty a previously dictionary list and use it again.

Code:-

```
# Python program to demonstrate working of
# dictionary clear()
text = {1: "geeks", 2: "for"}

text.clear()
print('text =', text)
```

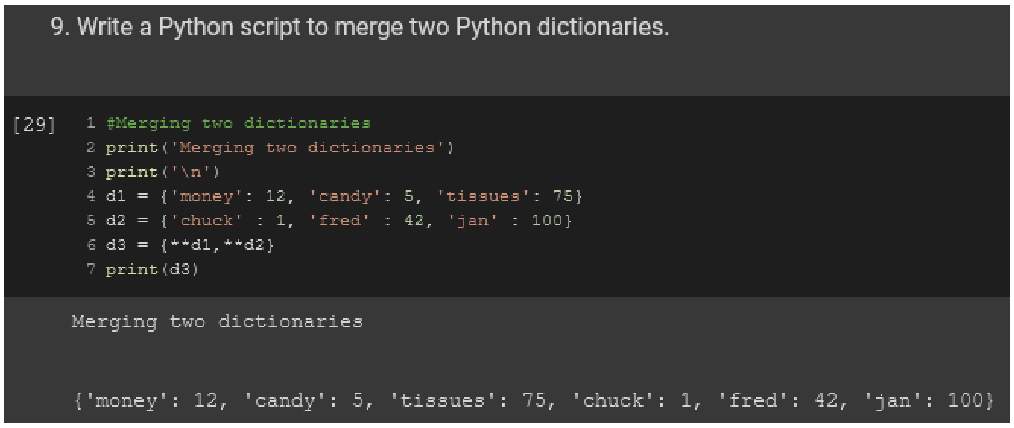
Output:-



```
# Python program to demonstrate working of
# dictionary clear()
text = {1: "geeks", 2: "for"}

text.clear()
print('text =', text)

text = {}
```

#5 (Merging two dictionaries)

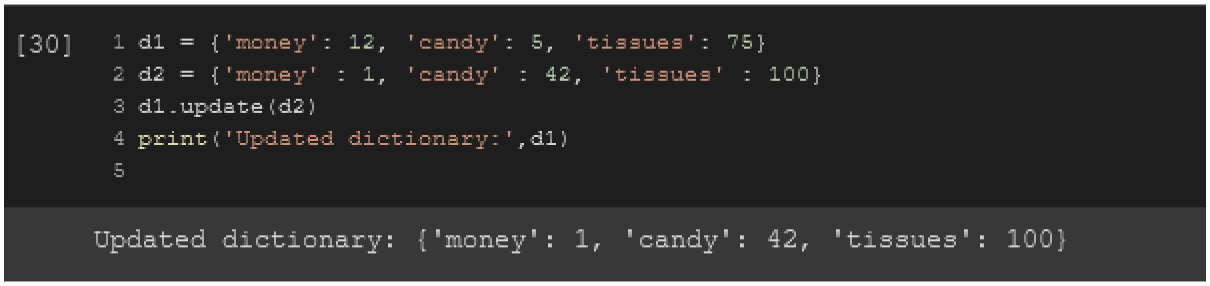
Code & Output:-



9. Write a Python script to merge two Python dictionaries.

```
[29]  #Merging two dictionaries
      print('Merging two dictionaries')
      print('\n')
      d1 = {'money': 12, 'candy': 5, 'tissues': 75}
      d2 = {'chuck' : 1, 'fred' : 42, 'jan' : 100}
      d3 = {**d1,**d2}
      print(d3)

   Merging two dictionaries


   {'money': 12, 'candy': 5, 'tissues': 75, 'chuck': 1, 'fred': 42, 'jan': 100}
```

#6 (Updating a dictionaries)



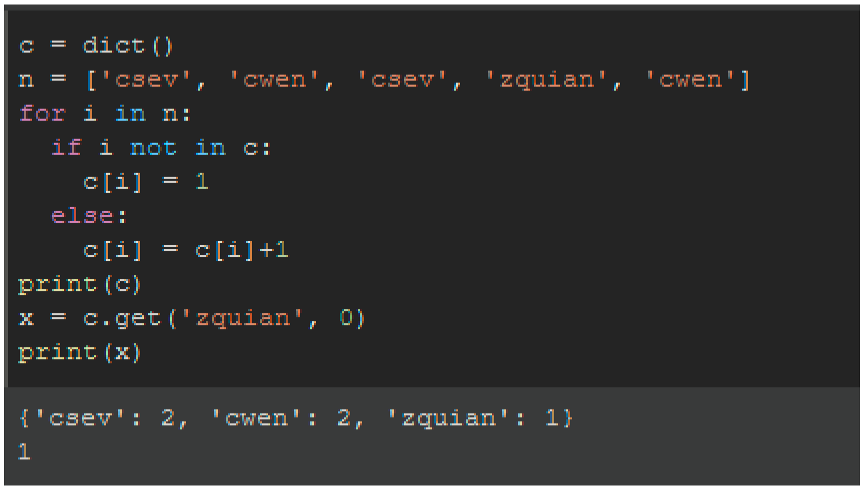
```
[30]
d1 = {'money': 12, 'candy': 5, 'tissues': 75}
d2 = {'money' : 1, 'candy' : 42, 'tissues' : 100}
d1.update(d2)
print('Updated dictionary:',d1)

```

```
Updated dictionary: {'money': 1, 'candy': 42, 'tissues': 100}
```

#7 (Using get method)

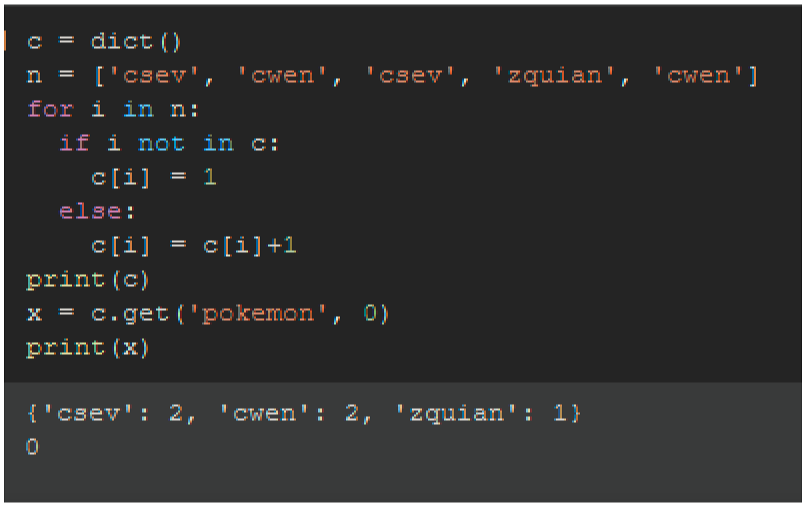
The **GET method** basically simplifies count. Two examples with explanations are given below.

Code:-



```
c = dict()
n = ['csev', 'cwen', 'csev', 'zquian', 'cwen']
for i in n:
  if i not in c:
    c[i] = 1
  else:
    c[i] = c[i]+1
print(c)
x = c.get('zquian', 0)
print(x)
```

```
{'csev': 2, 'cwen': 2, 'zquian': 1}
1
```



```
c = dict()
n = ['csev', 'cwen', 'csev', 'zquian', 'cwen']
for i in n:
  if i not in c:
    c[i] = 1
  else:
    c[i] = c[i]+1
print(c)
x = c.get('pokemon', 0)
print(x)
```

```
{'csev': 2, 'cwen': 2, 'zquian': 1}
0
```

<u>Explanation:-</u>

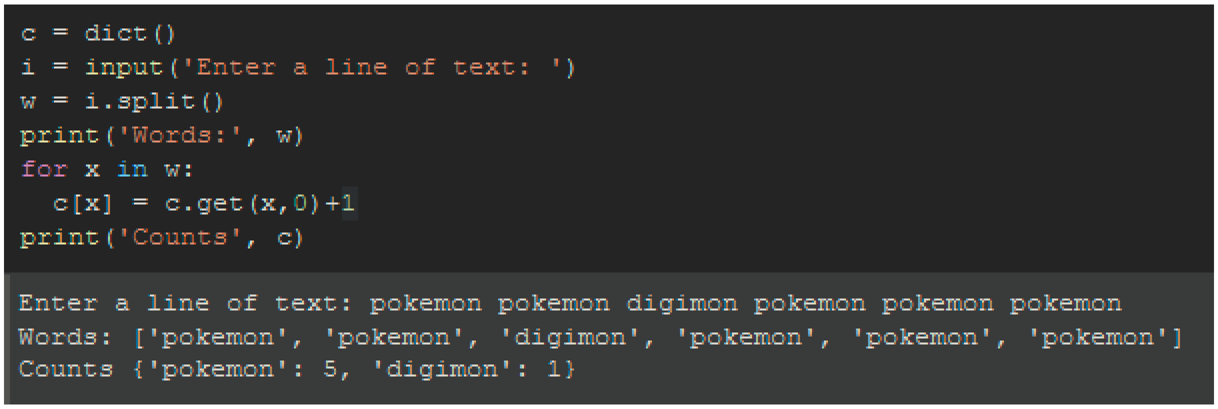
Here in the line **“x = c.get(‘zquian’, 0)”** the **get()** method states that go to the **c** dictionary, Use the key **‘zquian’** to look up the dictionary **c** how many times is that there. And the **0** is set as the default value if the answer if the given **KEYWORD** is not there like for the second case where the keyword **‘pokemon’** is given.
It is showing **0** as the x’s answer because there s no keyword named **pokemon** in the dictionary **c**.

<u>#6 (Counting pattern)</u>

Code:-

```
c = dict()
i = input('Enter a line of text: ')
w = i.split()
print('Words:', w)
for x in w:
  c[x] = c.get(x,0)+1
print('Counts', c)
```

Output:-



```python
c = dict()
i = input('Enter a line of text: ')
w = i.split()
print('Words:', w)
for x in w:
  c[x] = c.get(x,0)+1
print('Counts', c)
```

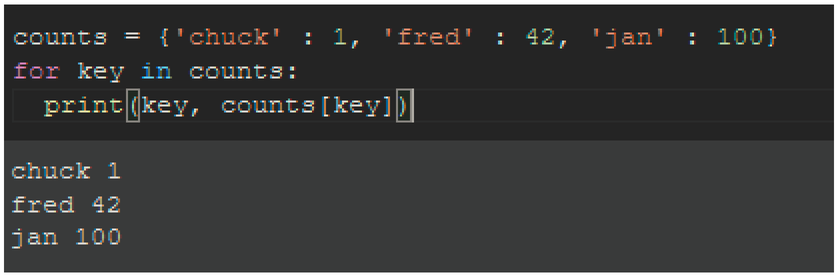
```
Enter a line of text: pokemon pokemon digimon pokemon pokemon pokemon
Words: ['pokemon', 'pokemon', 'digimon', 'pokemon', 'pokemon', 'pokemon']
Counts {'pokemon': 5, 'digimon': 1}
```

<u>#7 (Looking up a dictionary using a dictionary)</u>

Code:-

```python
counts = {'chuck' : 1, 'fred' : 42, 'jan' : 100}
for key in counts:
  print(key, counts[key])
```

Output:-



```
counts = {'chuck' : 1, 'fred' : 42, 'jan' : 100}
for key in counts:
  print(key, counts[key])
```

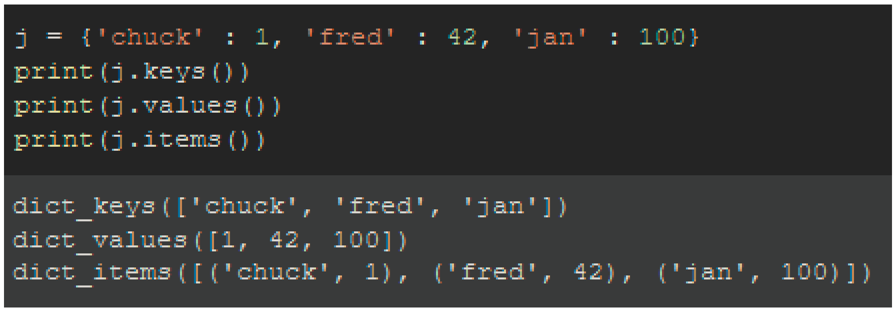
```
chuck 1
fred 42
jan 100
```

#8 (Retrieving lists of keys and values)

Code:-

```
j = {'chuck' : 1, 'fred' : 42, 'jan' : 100}
print(j.keys())
print(j.values())
print(j.items())
```

Output:-



```
j = {'chuck' : 1, 'fred' : 42, 'jan' : 100}
print(j.keys())
print(j.values())
print(j.items())
```

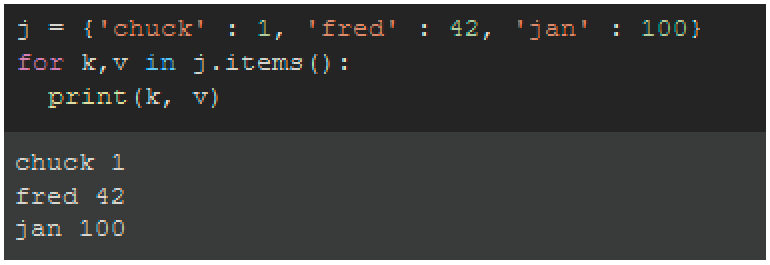
```
dict_keys(['chuck', 'fred', 'jan'])
dict_values([1, 42, 100])
dict_items([('chuck', 1), ('fred', 42), ('jan', 100)])
```

#9 (Two iteration variables)

Code:-

```
j = {'chuck' : 1, 'fred' : 42, 'jan' : 100}
for k,v in j.items():
  print(k, v)
```

Output:-



```
j = {'chuck' : 1, 'fred' : 42, 'jan' : 100}
for k,v in j.items():
  print(k, v)
```

```
chuck 1
fred 42
jan 100
```

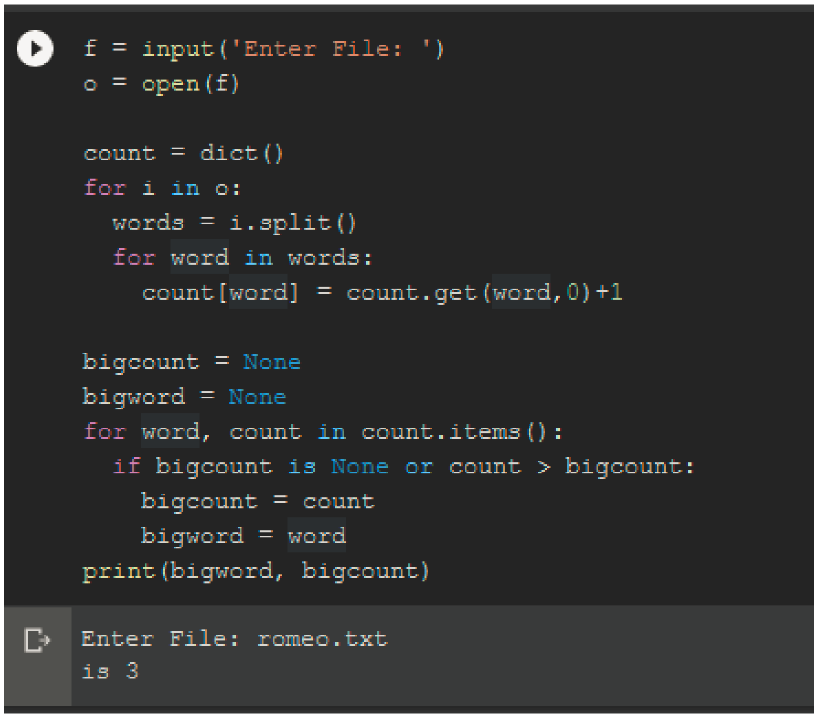
Explanation:-
Here, the two iteration variables K & V goes for respectively KEY & VALUES pairs.

#10 (Using two iteration variables in dictionary)

Code:-

```
f = input('Enter File: ')
o = open(f)

count = dict()
for i in o:
  words = i.split()
  for word in words:
    count[word] = count.get(word,0)+1

bigcount = None
bigword = None
for word, count in count.items():
  if bigcount is None or count > bigcount:
    bigcount = count
    bigword = word
print(bigword, bigcount)
```

Output:-



```
f = input('Enter File: ')
o = open(f)

count = dict()
for i in o:
  words = i.split()
  for word in words:
    count[word] = count.get(word,0)+1

bigcount = None
bigword = None
for word, count in count.items():
  if bigcount is None or count > bigcount:
    bigcount = count
    bigword = word
print(bigword, bigcount)
```

```
Enter File: romeo.txt
is 3
```