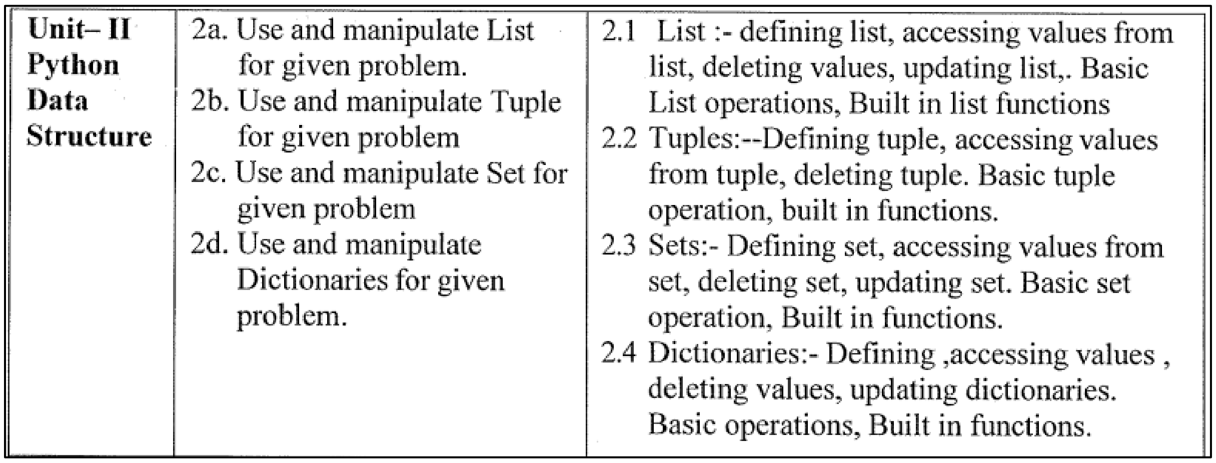
Contents

UNIT II

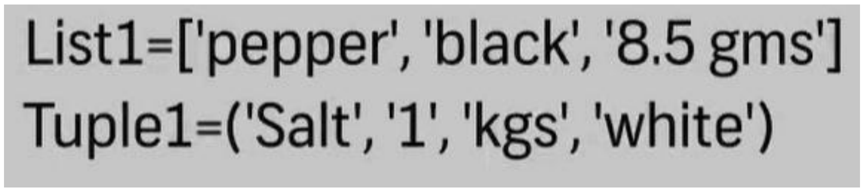
PYTHON DATA STRUCTURE



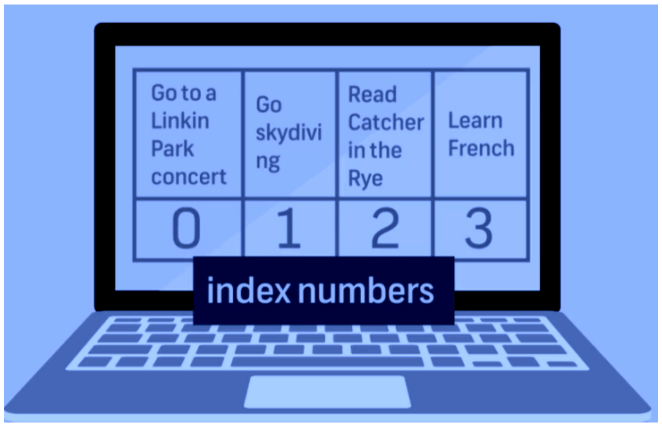
Unit– II Python Data Structure	2a. Use and manipulate List for given problem. 2b. Use and manipulate Tuple for given problem 2c. Use and manipulate Set for given problem 2d. Use and manipulate Dictionaries for given problem.	2.1 List :- defining list, accessing values from list, deleting values, updating list,. Basic List operations, Built in list functions 2.2 Tuples:--Defining tuple, accessing values from tuple, deleting tuple. Basic tuple operation, built in functions. 2.3 Sets:- Defining set, accessing values from set, deleting set, updating set. Basic set operation, Built in functions. 2.4 Dictionaries:- Defining ,accessing values , deleting values, updating dictionaries. Basic operations, Built in functions.

<u>2.1, 2.2 Lists and Tuples:</u>

List and tuples are single storage unit which hold multiple data items together. These data items may or may not be of same types. They may contain mix of numeric and string data types.



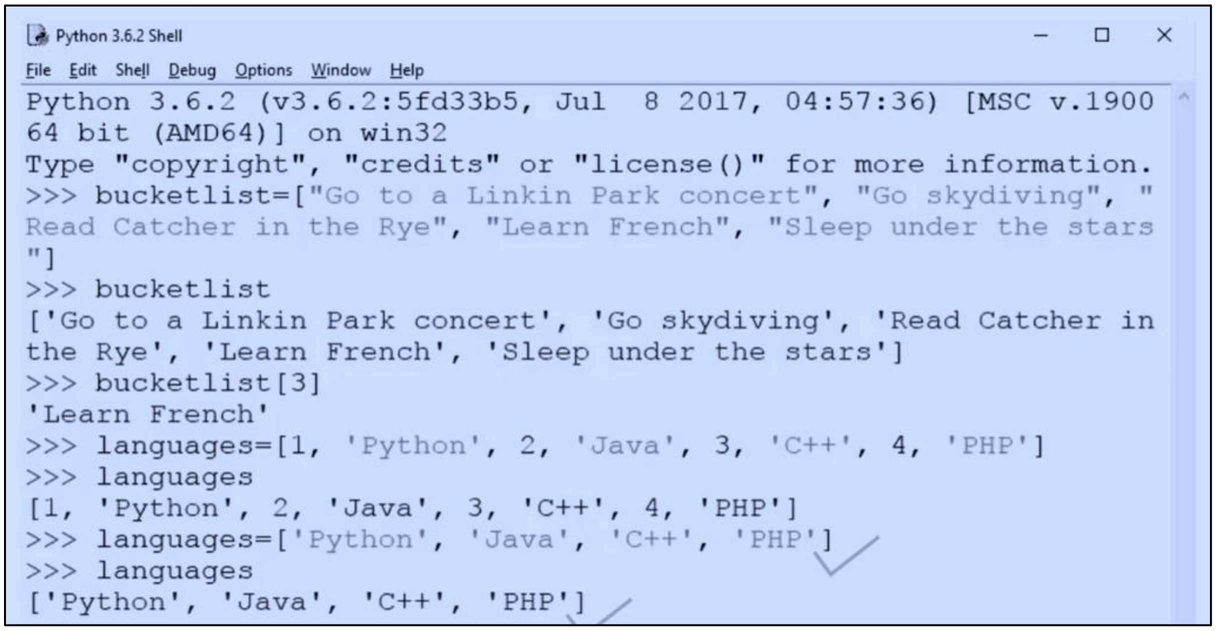
```
List1=['pepper', 'black', '8.5 gms']
Tuple1=('Salt', '1', 'kgs', 'white')
```



List:

Here is my bucket list, you can see, I have written thing randomly here. In python, this list is created in a computer memory with each item being stored in a separate memory location. The memory location and number with numbering starting from zero(0).Numbers are called index numbers.

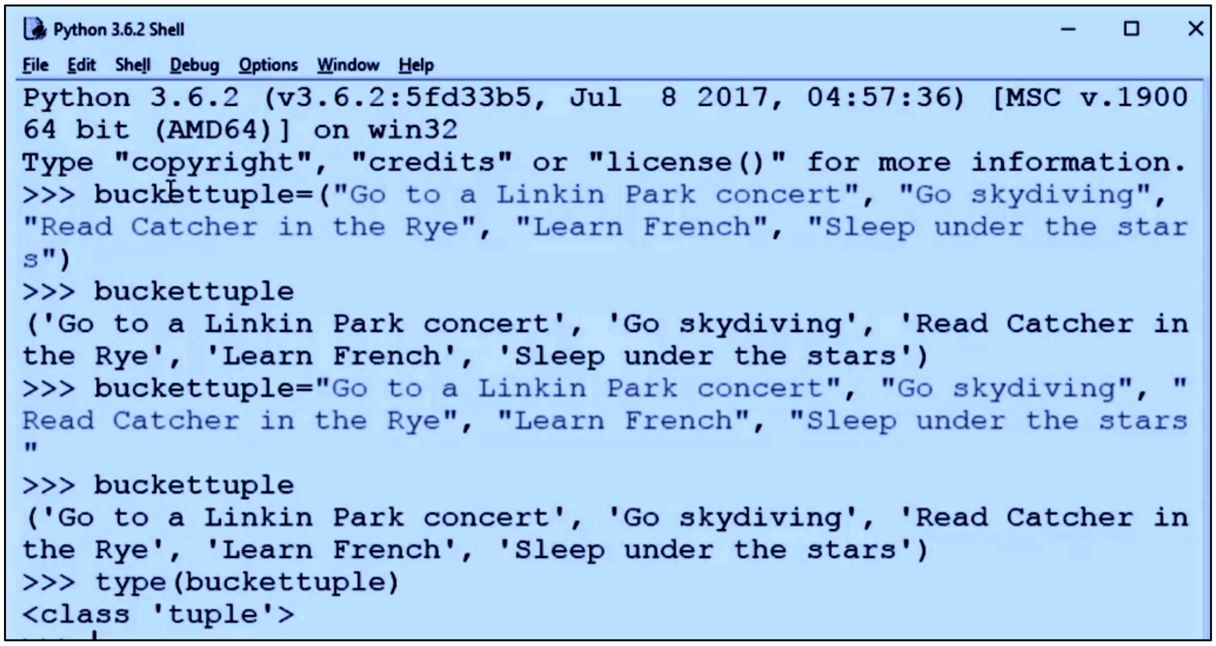
To create a list in python, we first give it a name, let us say bucket list. Then we write items separated by commas and enclosed in square bracket. We can access items using its index number. Just remember, start counting from zero instead of one when counting a index.



```
Python 3.6.2 (v3.6.2:5fd33b5, Jul  8 2017, 04:57:36) [MSC v.1900
64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> bucketlist=["Go to a Linkin Park concert", "Go skydiving", "
Read Catcher in the Rye", "Learn French", "Sleep under the stars
"]
>>> bucketlist
['Go to a Linkin Park concert', 'Go skydiving', 'Read Catcher in
the Rye', 'Learn French', 'Sleep under the stars']
>>> bucketlist[3]
'Learn French'
>>> languages=[1, 'Python', 2, 'Java', 3, 'C++', 4, 'PHP']
>>> languages
[1, 'Python', 2, 'Java', 3, 'C++', 4, 'PHP']
>>> languages=['Python', 'Java', 'C++', 'PHP']
>>> languages
['Python', 'Java', 'C++', 'PHP']
```

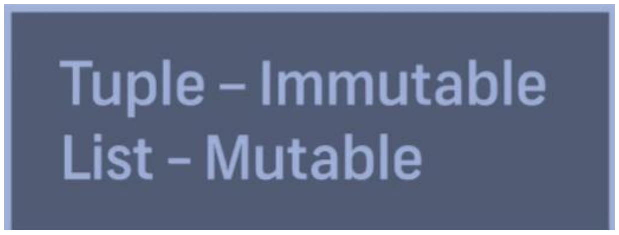
Tuple:

Tuple is similar to list except that it uses round brackets or parenthesis instead of square brackets. Note that parenthesis is optional. Even if parentheses are omitted, object type is treated as tuple. Tuple is immutable i.e., you cannot change its contents after it is created but remember a list is mutable.

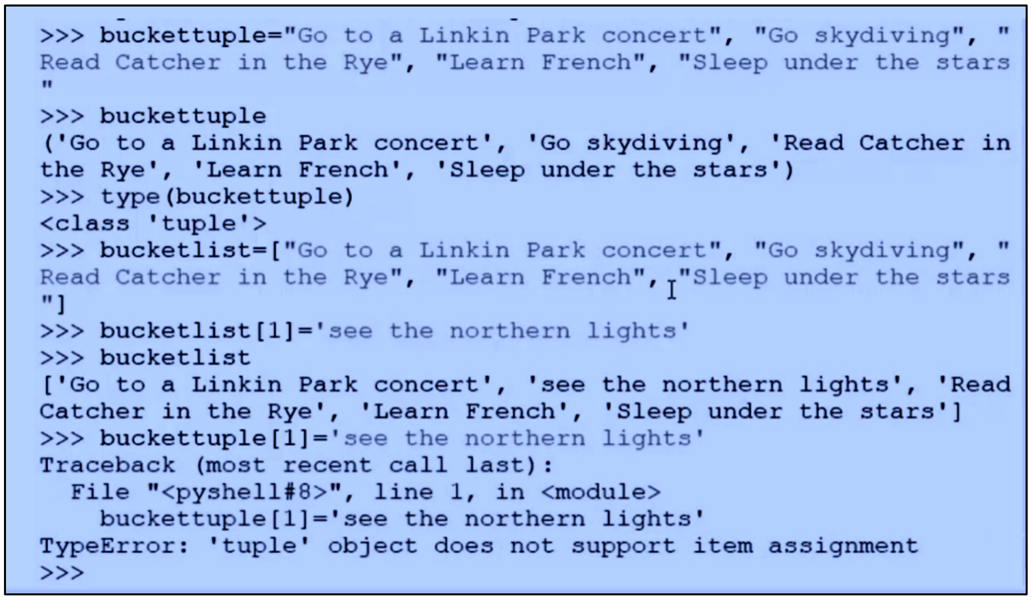


```
Python 3.6.2 (v3.6.2:5fd33b5, Jul  8 2017, 04:57:36) [MSC v.1900
64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> buckettuple=("Go to a Linkin Park concert", "Go skydiving",
"Read Catcher in the Rye", "Learn French", "Sleep under the star
s")
>>> buckettuple
('Go to a Linkin Park concert', 'Go skydiving', 'Read Catcher in
the Rye', 'Learn French', 'Sleep under the stars')
>>> buckettuple="Go to a Linkin Park concert", "Go skydiving", "
Read Catcher in the Rye", "Learn French", "Sleep under the stars
"
>>> buckettuple
('Go to a Linkin Park concert', 'Go skydiving', 'Read Catcher in
the Rye', 'Learn French', 'Sleep under the stars')
>>> type(buckettuple)
<class 'tuple'>
```

Tuple does not support item assignment. So tuple is **immutable.**

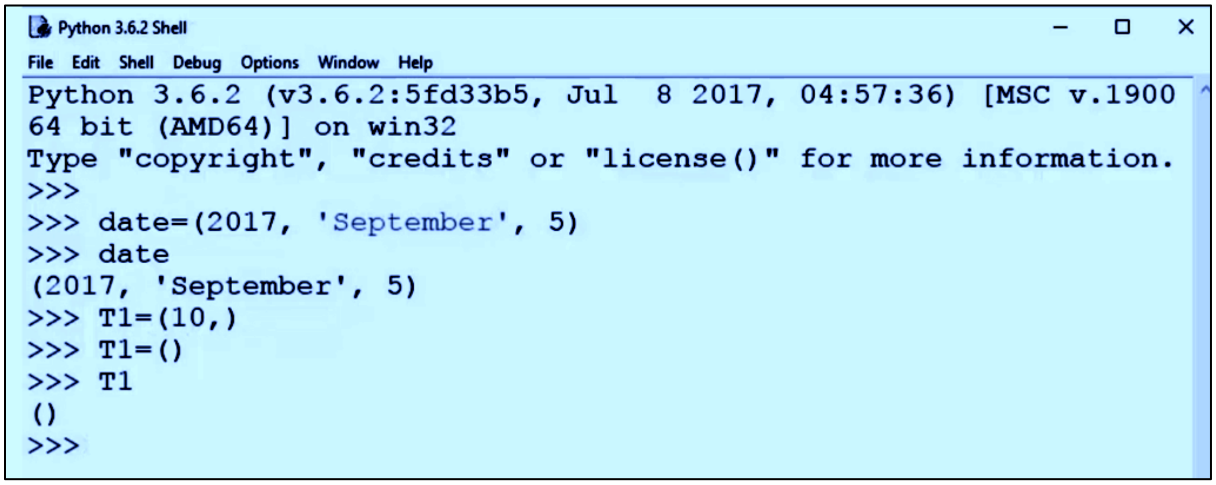


Tuple – Immutable
List – Mutable



```
>>> buckettuple="Go to a Linkin Park concert", "Go skydiving", "
Read Catcher in the Rye", "Learn French", "Sleep under the stars
"
>>> buckettuple
('Go to a Linkin Park concert', 'Go skydiving', 'Read Catcher in
the Rye', 'Learn French', 'Sleep under the stars')
>>> type(buckettuple)
<class 'tuple'>
>>> bucketlist=["Go to a Linkin Park concert", "Go skydiving", "
Read Catcher in the Rye", "Learn French", "Sleep under the stars
"]
>>> bucketlist[1]='see the northern lights'
>>> bucketlist
['Go to a Linkin Park concert', 'see the northern lights', 'Read
Catcher in the Rye', 'Learn French', 'Sleep under the stars']
>>> buckettuple[1]='see the northern lights'
Traceback (most recent call last):
  File "<pyshell#8>", line 1, in <module>
    buckettuple[1]='see the northern lights'
TypeError: 'tuple' object does not support item assignment
>>>
```

You can store any type of item in a tuple. It is typically used to represent a data structure such as a date which may contain objects of different types. To define a tuple with only one item we put a comma after it inside the parenthesis. An empty tuple have nothing inside the parenthesis.



```
Python 3.6.2 (v3.6.2:5fd33b5, Jul  8 2017, 04:57:36) [MSC v.1900
64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
>>> date=(2017, 'September', 5)
>>> date
(2017, 'September', 5)
>>> T1=(10,)
>>> T1=()
>>> T1
()
>>>
```

Lists and Tuples Operatoros

- **+ Concatenation**

Appends the second list or tuple to the first.

The data types being concatenated should be of the same type. For example, you cannot concatenate a list and a tuple.

List

```
>>> founders=["Iron Man", "Thor", "Ant-Man", "Hulk", "Wasp"]
>>> recruits=["Captain America", "Hawkeye"]
>>> founders + recruits
['Iron Man', 'Thor', 'Ant-Man', 'Hulk', 'Wasp', 'Captain America', 'Hawkeye']
>>>
```

Tuple

```
>>> stark=("Ned", "Catelyn", "Brandon", "Sansa", "Arya", "Robb")
>>> mormont=("Jeor", "Maege", "Jorah", "Lyanna")
>>> stark + mormont
('Ned', 'Catelyn', 'Brandon', 'Sansa', 'Arya', 'Robb', 'Jeor', 'Maege', 'Jorah', 'Lyanna')
>>>
```

- *** Repetition**

Concatenates multiple copies of the same list or tuple.

List

```
>>> dna=["A", "G","T","C"]
>>> dna * 3
['A', 'G', 'T', 'C', 'A', 'G', 'T', 'C', 'A', 'G', 'T', 'C']
>>>
```

Tuple

```
>>> glucose=("C6", "H12", "O6")
>>> glucose*6
('C6', 'H12', 'O6', 'C6', 'H12', 'O6', 'C6', 'H12', 'O6', 'C6', 'H12', 'O6', 'C6', 'H12', 'O6', 'C6', 'H12',
'O6')
>>>
```

- **[] Slice**

Returns the item at given index in a list or tuple.
A negative index counts the position from right with the count starting at -1.

Note: Recall that the index starts at 0.

List

```
>>> founders=["Iron Man", "Thor", "Ant-Man", "Hulk", "Wasp"]
>>> founders[3]
'Hulk'
>>>
>>> founders[-3]
'Ant-Man'
>>>
```

Tuple

```
>>> stark=("Ned", "Catelyn", "Brandon", "Sansa", "Arya", "Robb")
>>> stark[2]
'Brandon'
>>>
>>> stark[-1]
'Robb'
>>>
```

- **[:] Range Slice**

Fetches items in a range specified by the two index operands separated by the colon [:] symbol.

If the first operand is omitted, the range starts at zero index.If the second operand is omitted, range goes upto the end of the list or tuple.

List

```
>>> avengers=["Iron Man", "Thor", "Ant-Man", "Hulk", "Wasp", "Captain America",
"Hawkeye"]
>>> avengers[2:5]
['Ant-Man', 'Hulk', 'Wasp']
>>>
>>> avengers[3:]
```

```
['Hulk', 'Wasp', 'Captain America', 'Hawkeye']
>>>
>>> avengers[:3]
['Iron Man', 'Thor', 'Ant-Man']
>>>
```

Tuple

```
>>> got=("Ned", "Catelyn", "Brandon", "Sansa", "Arya", "Robb", "Jeor", "Maege", "Jorah",
"Lyanna")
>>> got[1:4]
('Catelyn', 'Brandon', 'Sansa')
>>>
>>> got[3:]
('Sansa', 'Arya', 'Robb', 'Jeor', 'Maege', 'Jorah', 'Lyanna')
>>> got[4:]
('Arya', 'Robb', 'Jeor', 'Maege', 'Jorah', 'Lyanna')
>>>
```

- **in Membership**

Returns true if an object exists in the given list or tuple.

Note: If the list or tuple comprises string items, then the items are case sensitive.

List

```
>>> avengers=["Iron Man", "Thor", "Ant-Man", "Hulk", "Wasp", "Captain America",
"Hawkeye"]
>>> "Wasp" in avengers
True
>>>
>>> "Black Widow" in avengers
False
>>>
```

Tuple

```
>>> got=("Ned", "Catelyn", "Brandon", "Sansa", "Arya", "Robb", "Jeor", "Maege", "Jorah",
"Lyanna")
>>> "Arya" in got
True
>>>
>>> "Jon" in got
False
>>>
```

- **not in Membership**

Returns true if an object does not exist in the given list or tuple

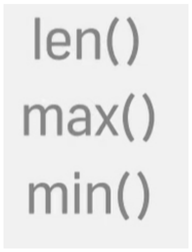
List

```
>>> avengers=["Iron Man", "Thor", "Ant-Man", "Hulk", "Wasp", "Captain America",
"Hawkeye"]
>>> "Wasp" not in avengers
False
>>>
>>> "Black Widow" not in avengers
True
>>>
```

Tuple

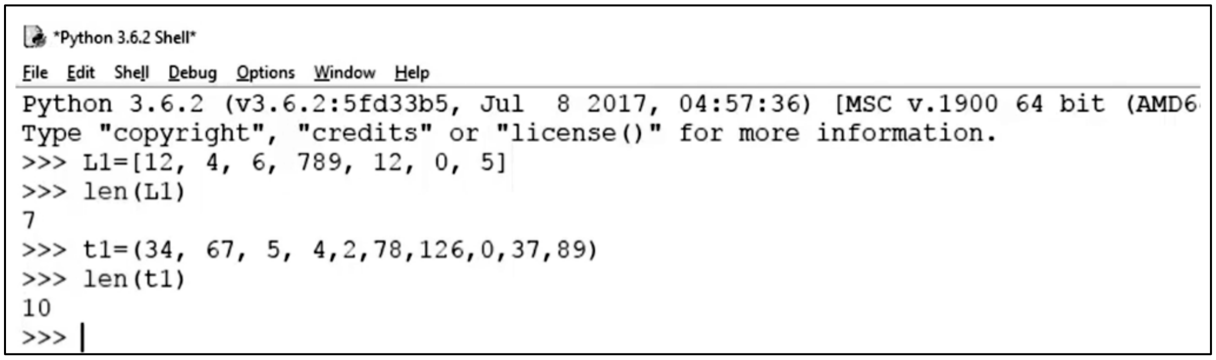
```
>>> got=("Ned", "Catelyn", "Brandon", "Sansa", "Arya", "Robb", "Jeor", "Maege", "Jorah",
"Lyanna")
>>> "Arya" not in got
False
>>>
>>> "Jon" not in got
True
>>>
```

Built in functions of LIST and TUPLE



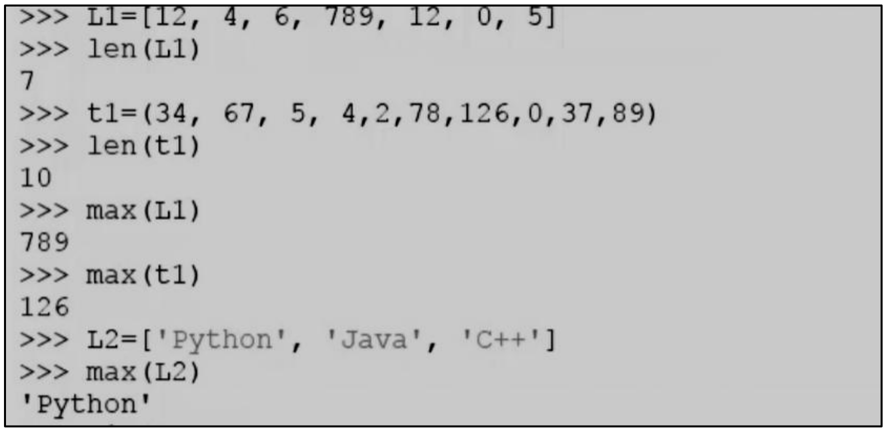
```
len()
max()
min()
```

The len() function helps to know howmany elements a list or tuple has.



```
Python 3.6.2 (v3.6.2:5fd33b5, Jul  8 2017, 04:57:36) [MSC v.1900 64 bit (AMD6
Type "copyright", "credits" or "license()" for more information.
>>> L1=[12, 4, 6, 789, 12, 0, 5]
>>> len(L1)
7
>>> t1=(34, 67, 5, 4,2,78,126,0,37,89)
>>> len(t1)
10
>>> |
```

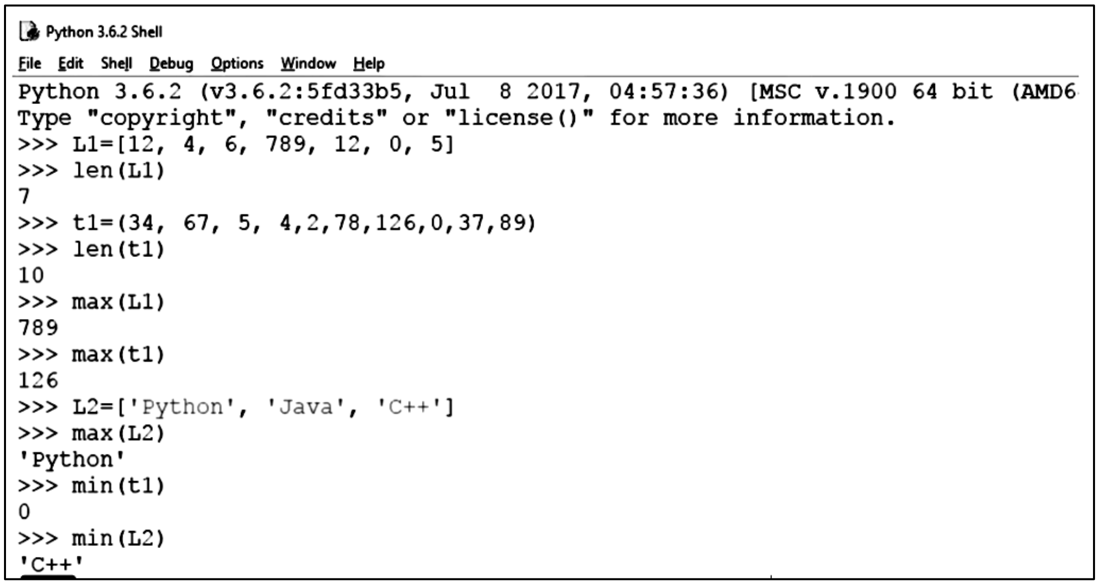
The max() function is useful when you need to find the highest number if a tuple or list contains numeric item.



```
>>> L1=[12, 4, 6, 789, 12, 0, 5]
>>> len(L1)
7
>>> t1=(34, 67, 5, 4,2,78,126,0,37,89)
>>> len(t1)
10
>>> max(L1)
789
>>> max(t1)
126
>>> L2=['Python', 'Java', 'C++']
>>> max(L2)
'Python'
```

List or tuple with string items, then item which comes last in the alphabetical order is returned.

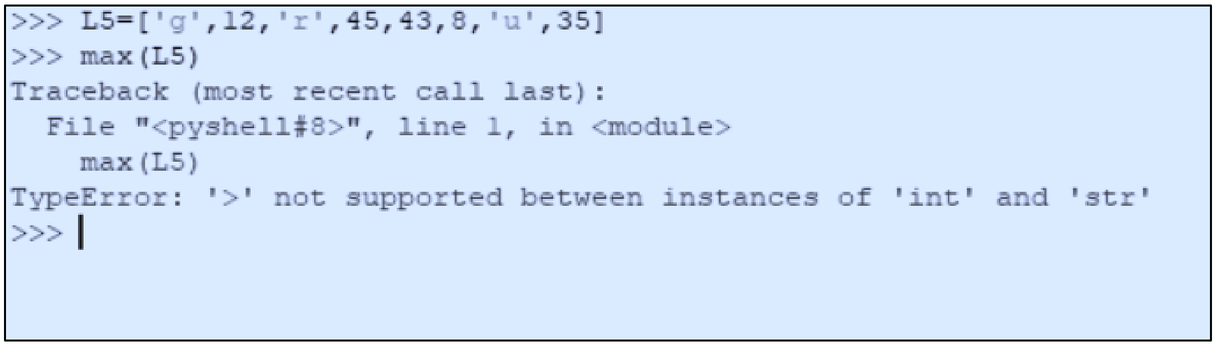
The min() function does the opposite of max() function. It returns the minimum value or the first occurrence in the alphabetical order.



```
Python 3.6.2 (v3.6.2:5fd33b5, Jul  8 2017, 04:57:36) [MSC v.1900 64 bit (AMD6
Type "copyright", "credits" or "license()" for more information.
>>> L1=[12, 4, 6, 789, 12, 0, 5]
>>> len(L1)
7
>>> t1=(34, 67, 5, 4,2,78,126,0,37,89)
>>> len(t1)
10
>>> max(L1)
789
>>> max(t1)
126
>>> L2=['Python', 'Java', 'C++']
>>> max(L2)
'Python'
>>> min(t1)
0
>>> min(L2)
'C++'
```

What will you think if the min() and max() function are used with the list or tuple containing mix of string and numeric values. Try it in IDLE and verify. We will get error that not supported between instances of ‘int’ and ‘str’.

```
L5=['g', 12, 'r', 45, 43, 8, 'u', 35]
t5=('one', 2, 'three', 4, 'five', 6)
```



```
>>> L5=['g',12,'r',45,43,8,'u',35]
>>> max(L5)
Traceback (most recent call last):
  File "<pyshell#8>", line 1, in <module>
    max(L5)
TypeError: '>' not supported between instances of 'int' and 'str'
>>> 
```

Manipulating Lists & Utility Functions

Manipulating Lists

You can modify the items within a list. Modifying a list means to change an item, or add a new item, or remove an existing item. Here are some methods of the built-in List class that help in modifying lists. Read through each function and then try it out in IDLE.

- **append()**

Adds an item at the end of the list.

```
>>> L2=['Tailor Swift', 'Ed Sheeran', 'Imagine Dragons', 'Pink', 'Maroon 5']
>>> L2.append('Halsey')
>>> L2
['Tailor Swift', 'Ed Sheeran', 'Imagine Dragons', 'Pink', 'Maroon 5', 'Halsey']
>>>
```

- **insert()**

Inserts an item in list at the specified index.

```
>>> L2=['Tailor Swift', 'Ed Sheeran', 'Imagine Dragons', 'Pink', 'Maroon 5']
>>> L2.insert(2, 'Halsey')
>>> L2
['Tailor Swift', 'Ed Sheeran', 'Halsey', Imagine Dragons', 'Pink', 'Maroon 5']
>>>
```

- **remove()**

Removes the specified item from the list.

```
>>> L2=['Python', 'Perl', 'Java', 'C++']
>>> L2.remove('Java')
>>> L2
['Python', 'Perl', 'C++']
>>>
```

- **pop()**

Removes and returns the last object in list.

```
>>> L2=['Python', 'Perl', 'Java', 'C++']
>>> L2.pop()
'C++'
>>> L2
```

```
['Python', 'Perl', 'Java']
>>>
```

- **reverse()**

Reverses the order of items in a list.

```
>>> L2=['Python', 'Perl', 'Java', 'C++']
>>> L2.reverse()
>>> L2
['C++', 'Java', 'Perl', 'Python']
>>>
```

- **sort()**

Rearranges items in the list according to alphabetical order. Default is ascending order. For descending order put reverse=True as argument in function bracket.

```
>>> L2=['Python', 'C++', 'Java', 'Ruby']
>>> L2.sort()
>>> L2
['C++', 'Java', 'Python', 'Ruby']
>>>
>>> L2.sort(reverse=True)
>>> L2
['Ruby', 'Python', 'Java', 'C++']
>>>
```

Converting One Sequence Type to Another

The following utility functions help in converting one sequence data type to other.

Converts a tuple or string to a list.

```
>>> t2=('python', 'java', 'c++')
>>> list(t2)
['python', 'java', 'c++']
>>> s1="Intern"
>>> list(s1)
['I', 'n', 't', 'e', 'r', 'n']
>>>
```

Converts a list or string to a tuple.

```
>>> L2=['C++', 'Java', 'Python', 'Ruby']
>>> tuple(L2)
('C++', 'Java', 'Python', 'Ruby')
>>> s1="robotics"
>>> tuple(s1)
('r', 'o', b', 'o', 't', 'i', 'c', 's')
>>>
```

2.3 Set Data Type and Set Operations

Set Data type

Set is also a collection data type in Python. However, it is **not an ordered collection of objects**, like a list or tuple. Hence, indexing and slicing operations cannot be done on a set object. A set also doesn't allow duplicate objects to be stored, whereas, in list and tuple, the same object can appear more than once. Even if an object is put more than once in a set, only one copy is held. Set is a Python implementation of a set as defined in Mathematics. The set object has suitable methods to perform mathematical set operations like union, intersection, difference, etc. A set object contains one or more items, not necessarily of the same type which are separated by a comma and enclosed in curly brackets {}.

```
>>> S1={1, "Ravi", 75.50}
>>> S1
{1, 75.5, 'Ravi'}
>>> type(S1)
<class 'set'>
>>> S2={10,23,40,23,50,10}
>>> S2
{40, 10, 50, 23}
>>>
```

set() function

Python has an in-built function set() using which set object can be constructed out of any sequence like string, list or tuple object.

```
>>> S2=set([45,67,87,36,55])
>>> S2
{55, 67, 36, 45, 87}
>>> S3=set((10,25,15))
>>> S3
{25, 10, 15}
>>>
```

The order of elements in the set is not necessarily the same that is given at the time of assignment. Python optimizes the structure for performing operations over the set as defined in mathematics. Only immutable (and hashable) objects can be a part of a set object. Numbers (integer, float as well as complex), strings, and tuple objects are accepted but list and dictionary objects are not.

```
>>> S1={(10,10), 10,20}
>>> S1
{10, 20, (10, 10)}
>>> S2={[10,10], 10,20}
```

```
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    S2={[10,10], 10,20}
TypeError: unhashable type: 'list'
>>>
```

In the first case, (10,10) is a tuple, hence it becomes part of a set. In the second example though, since [10,10] is a list, an error message is displayed saying the list is unhashable. (Hashing is a mechanism in computer science that enables quicker searching of objects in the computer's memory. https://en.wikipedia.org/wiki/Hash_function) Even though mutable objects are not stored in a set, set itself is a mutable object. A set object can be modified by add(), update(), remove() and discard() methods.

- **add()**

```
>>> S1=set({"Python", "Java", "C++"})
>>> S1
{'Python', 'C++', 'Java'}
>>> S1.add("Perl")
>>> S1
{'Perl', 'Python', 'C++', 'Java'}
>>>
```

- **update()**

```
>>> S1={"Python", "Java", "C++"}
>>> S1.update(["C++", "Basic"])
>>> S1
{'C++', 'Java', 'Python', 'Basic'}
>>> S1.update(["Ruby", "PHP"])
>>> S1
{'Ruby', 'PHP', 'Java', 'C++', 'Python', 'Basic'}
>>>
```

- **clear()**

```
>>> S1.clear()
>>> S1
set()
>>>
```

- **copy()**

```
>>> S1={"Python", "Java", "C++"}
>>> S2=S1.copy()
>>> S2
{'C++', 'Java', 'Python'}
>>>
```

- **discard()**

```
>>> S1={"Python", "Java", "C++"}
>>> S1.discard("Java")
>>> S1
{'C++', 'Python'}
>>> S1.discard("SQL")
>>> S1
{'C++', 'Python'}
>>>
```

- **remove()**

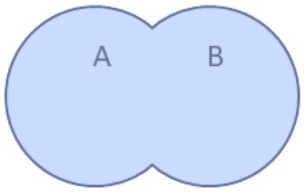
```
>>> S1={"Python", "Java", "C++"}
>>> S1.remove("C++")
>>> S1
{'Java', 'Python'}
>>> S1.remove("SQL")
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    S1.remove("SQL")
KeyError: 'SQL'
>>>
```

Set Operations

As mentioned earlier, set data type in Python implements set as defined in mathematics. Various Set operations can be performed using Python's set object. The operators |, &, - and ^ perform union, intersection, difference, and symmetric difference operations respectively. Each of these operators have a corresponding method associated with a built-in set class.

- **Union**

Union of two sets is a set of all elements in both.

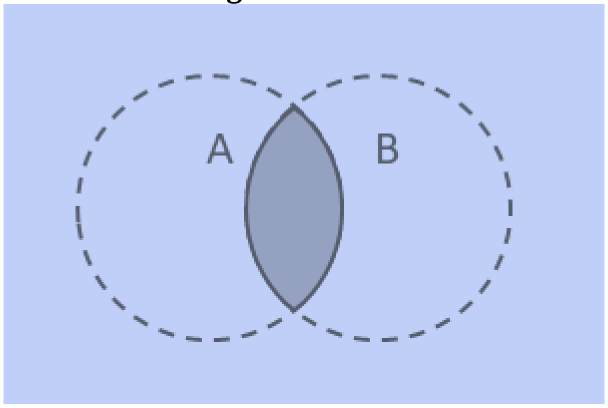


```
>>> s1={1,2,3,4,5}
>>> s2={4,5,6,7,8}
>>> s1|s2
{1, 2, 3, 4, 5, 6, 7, 8}
>>> s1.union(s2)
{1, 2, 3, 4, 5, 6, 7, 8}
>>>
```

Set is a specialized data type. One of the major applications of Python is an area of mathematical computing and data analysis in which set operations are important. We may drop this discussion considering it as not for a beginner (and also to curtail the size), but a learner (especially who intends to go in mathematical and scientific computing) should be encouraged to explore this section. We should emphasize this and provide this as a text for further reading.

- **Intersection**

Intersection of two sets is a set containing elements common to both

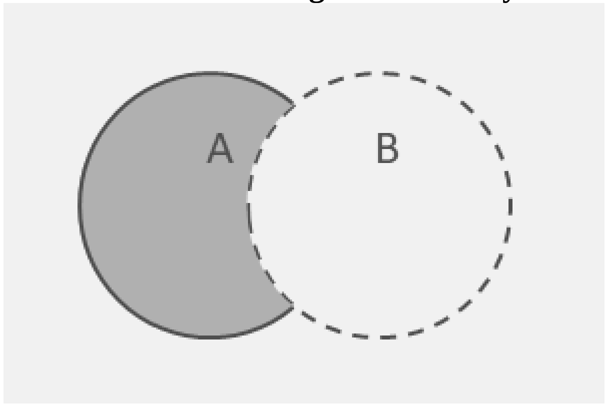


```
>>> s1={1,2,3,4,5}
>>> s2={4,5,6,7,8}
>>> s1&s2
{4, 5}
>>> s1.intersection(s2)
{4, 5}
>>>
```

Set is a specialized data type. One of the major applications of Python is an area of mathematical computing and data analysis in which set operations are important. We may drop this discussion considering it as not for a beginner (and also to curtail the size), but a learner (especially who intends to go in mathematical and scientific computing) should be encouraged to explore this section. We should emphasize this and provide this as a text for further reading.

- **Difference**

Difference of two sets results in a set containing elements only in first but not in second set.

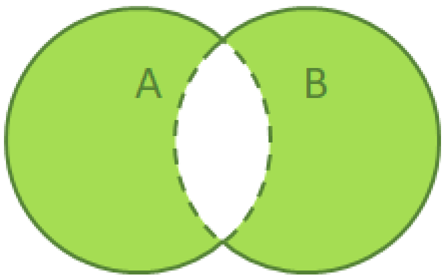


```
>>> s1={1,2,3,4,5}
>>> s2={4,5,6,7,8}
>>> s1-s2
{1, 2, 3}
>>> s2-s1
{8, 6, 7}
>>> s1.difference(s2)
{1, 2, 3}
>>> s2.difference(s1)
{8, 6, 7}
>>>
```

Set is a specialized data type. One of the major applications of Python is an area of mathematical computing and data analysis in which set operations are important. We may drop this discussion considering it as not for a beginner (and also to curtail the size), but a learner (especially who intends to go in mathematical and scientific computing) should be encouraged to explore this section. We should emphasize this and provide this as a text for further reading.

- **Symmetric Difference**

Result of Symmetric difference is a set consisting of elements in both sets excluding common elements



```
>>> s1={1,2,3,4,5}
>>> s2={4,5,6,7,8}
>>> s1^s2
{1, 2, 3, 6, 7, 8}
>>> s2^s1
{1, 2, 3, 6, 7, 8}
>>> s1.symmetric_difference(s2)
{1, 2, 3, 6, 7, 8}
>>> s2.symmetric_difference(s1)
{1, 2, 3, 6, 7, 8}
>>>
```

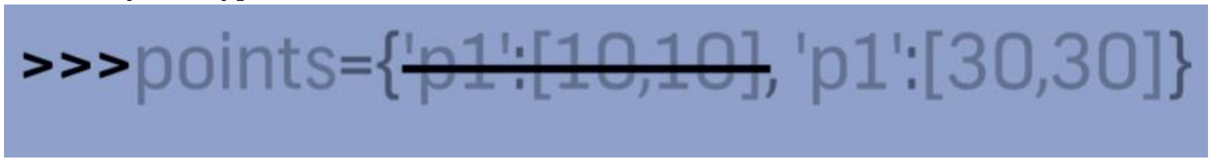
Set is a specialized data type. One of the major applications of Python is an area of mathematical computing and data analysis in which set operations are important. We may drop this discussion considering it as not for a beginner (and also to curtail the size), but a learner (especially who intends to go in mathematical and scientific computing) should be encouraged to explore this section. We should emphasize this and provide this as a text for further reading.

2.4 Dictionary data type:

Like list and tuple dictionary is also collection data type.

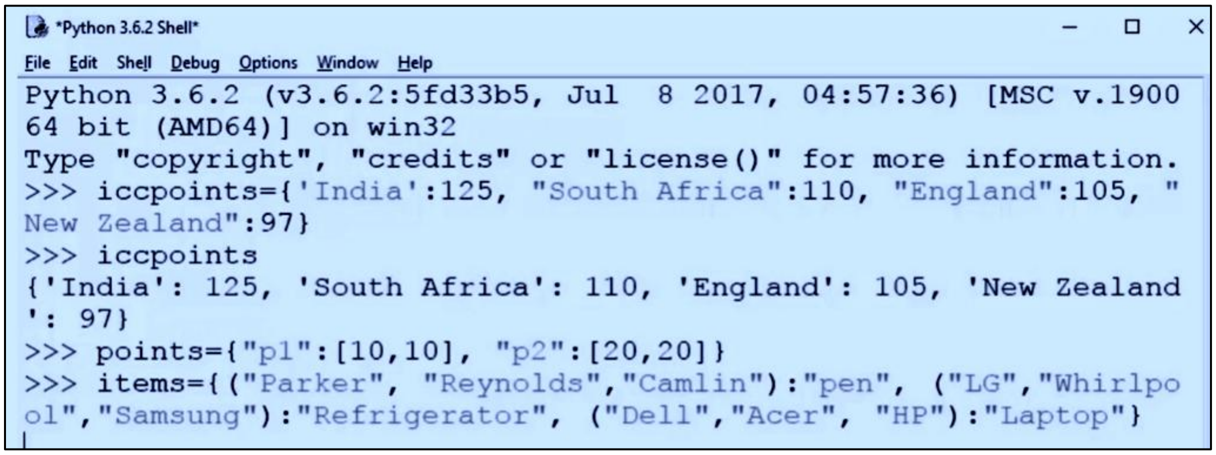
However, it is not a sequence. It is an unordered collection of items. Each item is a key value pair. For example, India is key and 125 is a value. In the dictionary object,

- the value is bound to the key by the colon (:) symbol.
- One or more key par separated by comma and put inside the curly brackets to form a dictionary object.
- A dictionary can contain other data types.
- It may have list as a value in the key value pair.
- In dictionary, the keys may be tuple.
- Key should be an immutable object.
- Number, String, Tuple can be used as key but list cannot be used as key.
- One key cannot be appeared more than one time in one dictionary. If a key appears more than once, only last one will be retained. The value however of any data type.



```
>>>points={~~'p1':[10,10]~~, 'p1':[30,30]}
```

- One value can be assigned to more than one key.



```
Python 3.6.2 (v3.6.2:5fd33b5, Jul  8 2017, 04:57:36) [MSC v.1900
64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> iccpoints={'India':125, "South Africa":110, "England":105, "
New Zealand":97}
>>> iccpoints
{'India': 125, 'South Africa': 110, 'England': 105, 'New Zealand
': 97}
>>> points={"p1":[10,10], "p2":[20,20]}
>>> items={("Parker", "Reynolds","Camlin"):"pen", ("LG","Whirlpo
ol","Samsung"):"Refrigerator", ("Dell","Acer", "HP"):"Laptop"}
```

Creating the Dictionary

The dictionary can be created by using multiple key-value pairs enclosed with the small brackets () and separated by the colon (:). The collections of the key-value pairs are enclosed within the curly braces {}.
The syntax to define the dictionary is given below.

#!/usr/bin/Python
dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}

Accessing values in a dictionary:

As List and Tuple have index no, Dictionary don’t have it because it not an ordered collection. Using a key with get() method, we can access values from dictionary.

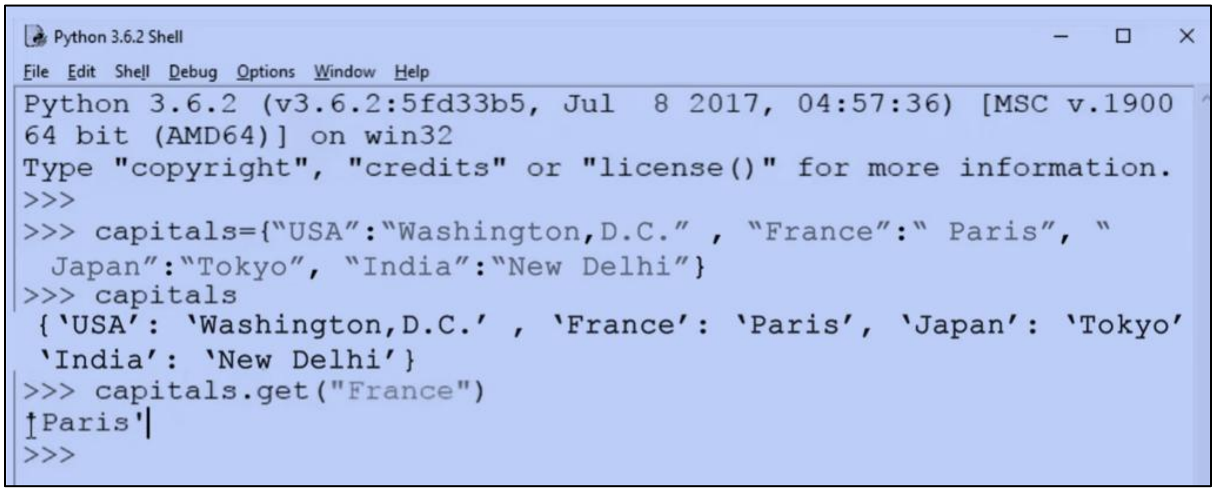
Accessing Values in Dictionary: To access dictionary elements, you can use the familiar square brackets along with the key to obtain its value.
Example:
dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}
print "dict['Name']: ", dict['Name']
print "dict['Age']: ", dict['Age']

Output:

dict['Name']: Zara

dict['Age']: 7

Let us have a dictionary called capitals with countries as a key and their capital as the values. To fetch the value from the key France, we use the syntax capiten.get(“France”).



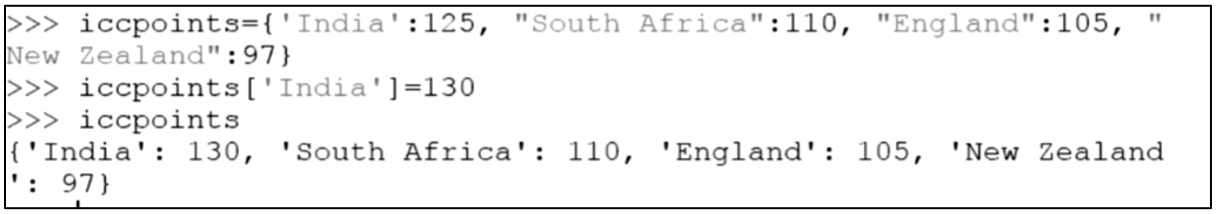
```
Python 3.6.2 (v3.6.2:5fd33b5, Jul  8 2017, 04:57:36) [MSC v.1900
64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
>>> capitals={“USA”:“Washington,D.C.” , “France”:“ Paris”, “
 Japan”:“Tokyo”, “India”:“New Delhi”}
>>> capitals
 {‘USA’: ‘Washington,D.C.’ , ‘France’: ‘Paris’, ‘Japan’: ‘Tokyo’
 ‘India’: ‘New Delhi’}
>>> capitals.get("France")
'Paris'
>>>
```

Updating values in a dictionary:

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

Example:
#!/usr/bin/Python
dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}
dict['Age'] = 8; # update existing entry

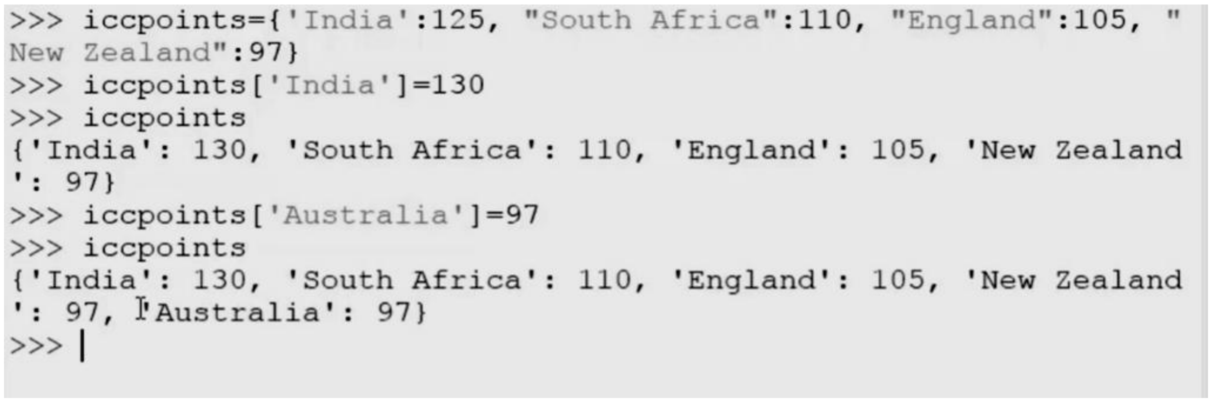
dict['School'] = "DPS School"; # Add new entry
print "dict['Age']: ", dict['Age']
print "dict['School']: ", dict['School']
Output:
dict['Age']: 8
dict['School']: DPS School



```
>>> iccpoints={'India':125, "South Africa":110, "England":105, "
New Zealand":97}
>>> iccpoints['India']=130
>>> iccpoints
{'India': 130, 'South Africa': 110, 'England': 105, 'New Zealand
': 97}
```

To change value from 125 to 130, simply use the above syntax.

We can also specify a new key and assign a value to it. It will get added to dictionary.



```
>>> iccpoints={'India':125, "South Africa":110, "England":105, "
New Zealand":97}
>>> iccpoints['India']=130
>>> iccpoints
{'India': 130, 'South Africa': 110, 'England': 105, 'New Zealand
': 97}
>>> iccpoints['Australia']=97
>>> iccpoints
{'India': 130, 'South Africa': 110, 'England': 105, 'New Zealand
': 97, 'Australia': 97}
>>> |
```

Delete Dictionary Elements:

You can either remove individual dictionary
elements or clear the entire contents of a dictionary. You can also delete entire dictionary in a single operation.
To explicitly remove an entire dictionary, just use the del statement.
Example:
#!/usr/bin/Python
dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}
del dict['Name']; # remove entry with key 'Name'
dict.clear(); # remove all entries in dict
del dict ; # delete entire dictionary

Looping through Dictionary:

A dictionary can be iterated using the for loop. If you
want to get both keys and the values in the output. You just have to add the keys and values as the argument of the print statement in comma separation. After each iteration

of the for loop, you will get both the keys its relevant values in the output.
Example

```
dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}
for key, value in dict.items():
print(key, ' - ', value)
```

The above example contains both the keys and the values in the output. The text ‘Related to’ in the output showing the given key is related to the given value in the output.

```
Name - Zara
Age - 7
Class – Firs
```

Built in functions of Dictionary:

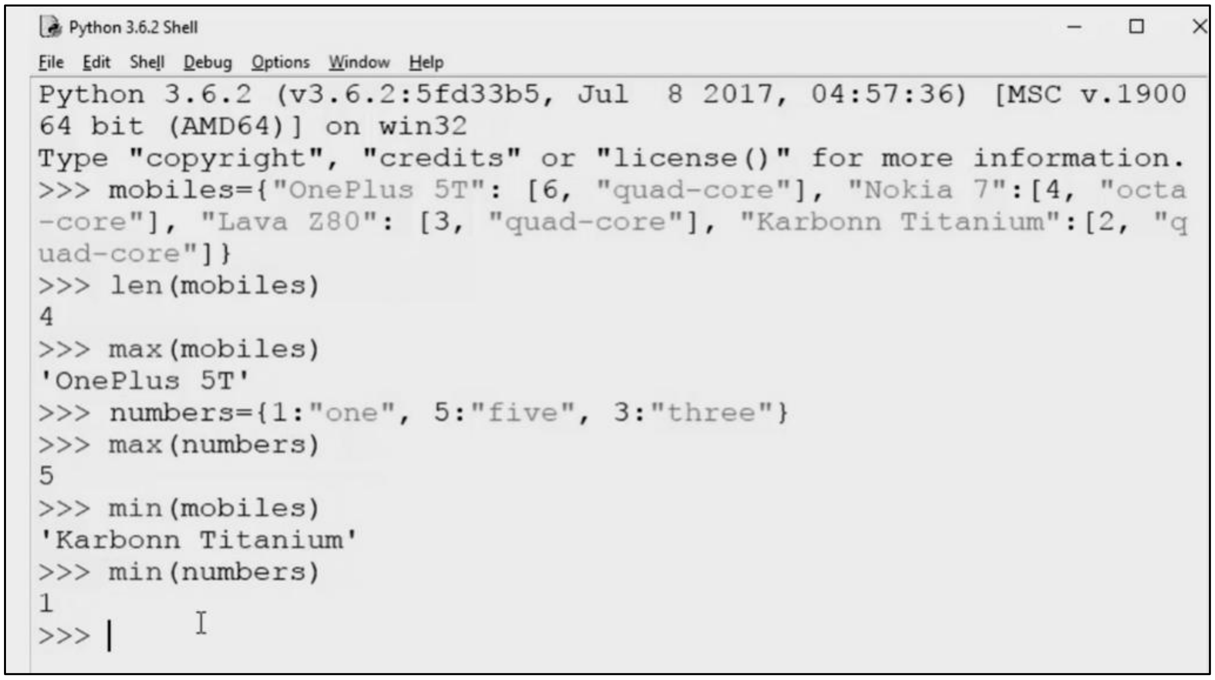
len()
max()
min()

len() function returns the number of key value pairs in a dictionary. Simply write len and name of the dictionary in the parenthesis.

The max() function returns the key which is either largest or last in alphabetical order.

For ex: In the dictionary mobiles max() function will return “Oneplus 5T” because ‘O’ comes last among the other keys.

min() function will do the opposite function of max() function. It returns the key which is either smallest or first in alphabetical order.



```
Python 3.6.2 (v3.6.2:5fd33b5, Jul  8 2017, 04:57:36) [MSC v.1900
64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> mobiles={"OnePlus 5T": [6, "quad-core"], "Nokia 7":[4, "octa
-core"], "Lava Z80": [3, "quad-core"], "Karbonn Titanium":[2, "q
uad-core"]}
>>> len(mobiles)
4
>>> max(mobiles)
'OnePlus 5T'
>>> numbers={1:"one", 5:"five", 3:"three"}
>>> max(numbers)
5
>>> min(mobiles)
'Karbonn Titanium'
>>> min(numbers)
1
>>> |
```

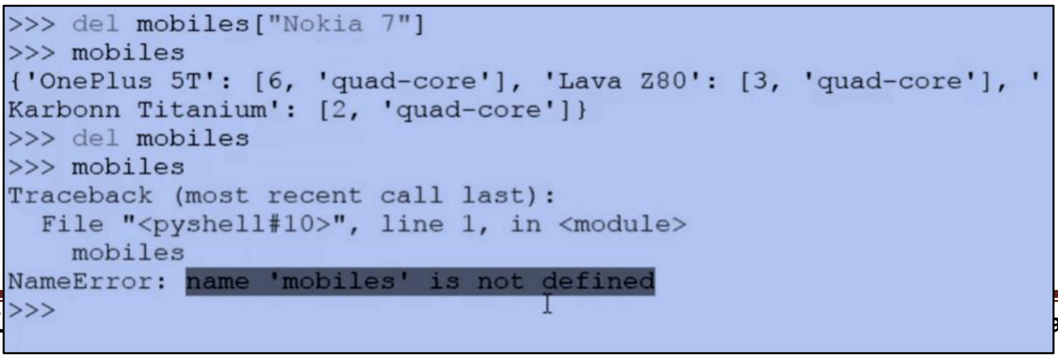
Deleting a dictionary item:

del
pop()
clear()

del:

We can delete key value pair from a dictionary. Python has a keyword called **del**, which can be used for this. To delete a pair, we use key as a parameter.

Same del keyword is used to delete dictionary itself. For this type del followed by dictionary name.



```
>>> del mobiles["Nokia 7"]
>>> mobiles
{'OnePlus 5T': [6, 'quad-core'], 'Lava Z80': [3, 'quad-core'], '
Karbonn Titanium': [2, 'quad-core']}
>>> del mobiles
>>> mobiles
Traceback (most recent call last):
  File "<pyshell#10>", line 1, in <module>
    mobiles
NameError: name 'mobiles' is not defined
>>>
```

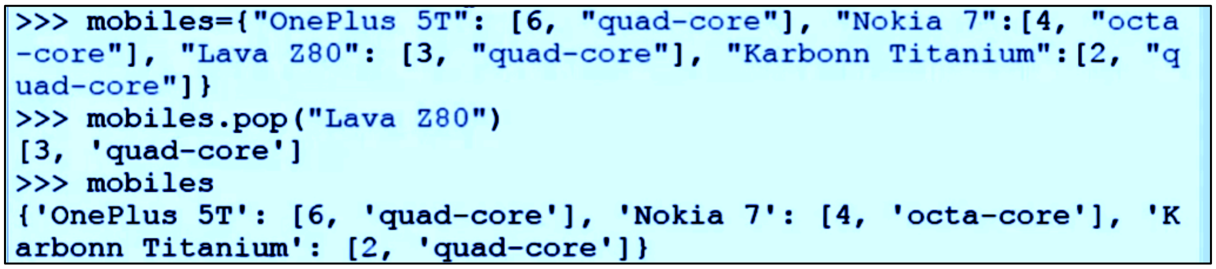
That means mobiles dictionary has been removed from memory.

<u>pop():</u>

Another way to remove key value pair is using pop method. We use syntax

Dictionary name. pop (“key name to be deleted”) and press enter.

The value linked with key is displayed. We can verify if the item is deleted by typing the name of dictionary at the prompt.



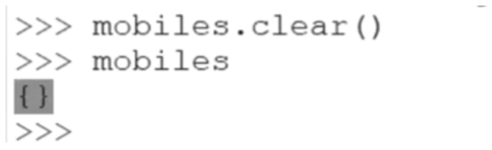
```
>>> mobiles={"OnePlus 5T": [6, "quad-core"], "Nokia 7":[4, "octa
-core"], "Lava Z80": [3, "quad-core"], "Karbonn Titanium":[2, "q
uad-core"]}
>>> mobiles.pop("Lava Z80")
[3, 'quad-core']
>>> mobiles
{'OnePlus 5T': [6, 'quad-core'], 'Nokia 7': [4, 'octa-core'], 'K
arbonn Titanium': [2, 'quad-core']}
```

<u>Clear():</u>

To make empty the dictionary,type at promt

Dictionary name. clear ( )

To verify, type the name of dictionary,we will see{} that means dictionary has become empty.



```
>>> mobiles.clear()
>>> mobiles
{}
>>>
```

<u>Methods of Dictionary Object:</u>

- **<u>items()</u>**

This method returns a list of tuples, with each tuple containing one key and and the corresponding value.

```
>>> captains={'England': 'Root', 'Australia': 'Smith', 'India': 'Virat', 'Pakistan': 'Sarfraz'}
>>> captains.items()
dict_items([('England', 'Root'), ('Australia', 'Smith'), ('India', 'Virat'), ('Pakistan', 'Sarfraz')])
>>>
```

- **keys()**

This method returns a list object comprising keys in the dictionary.

```
>>> captains={'England': 'Root', 'Australia': 'Smith', 'India': 'Virat', 'Pakistan': 'Sarfraz'}
>>> captains.keys()
dict_items(['England', 'Australia', 'India', 'Pakistan'])
>>>
```

- **values()**

This method returns a list object comprising values in the dictionary.

```
>>> captains={'England': 'Root', 'Australia': 'Smith', 'India': 'Virat', 'Pakistan': 'Sarfraz'}
>>> captains.values()
dict_items([Root', 'Smith', 'Virat', 'Sarfraz'])
>>>
```

Code Challenge

For each of the statements displayed below, write the code that will display the required result for the given dictionary.
villians={1:"magneto", 2:"Joker", 3:"Doctor Doom", 5:"Lex Luthor", 9:"Green Goblin", 6:"Loki"}

1. Display the length of the dictionary.
2. Display the key which has the least value.
3. Display the value of the key 9 as well as delete it simultaneously.
4. Display all the values in dictionary.
5. Delete all the elements in the dictionary while retaining the empty object.

Problem Statement	Ideal Solution
Display the length of the dictionary.	>>> len(villians) **Output** 6
Display the key which has the least value.	>>> min(villians) **Output** 1
Display the value of the key 9 as well as delete it simultaneously.	>>> villians.pop(9) **Output** 'Green Goblin'

Display all the values in dictionary.	>> villians.values() **Output** dict_values(['magneto', 'Joker', 'Doctor Doom', 'Lex Luthor', 'Loki'])
Delete all the elements in the dictionary while retaining the empty object.	>>> villians.clear() **Output** villians={}