SNS COLLEGE OF ENGINEERING
Kurumbapalayam (Po), Coimbatore – 641 107

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

19IT103 – COMPUTATIONAL THINKING AND PYTHON PROGRAMMING

- **A readable, dynamic, pleasant, flexible, fast and powerful language**

Session wise Agenda

- **session 1 - List (Operations, Slice, Methods)**
- **Session 2 - List (Loop, Mutability)**
- Session 3 - List (Aliasing, Cloning, Parameters)
- Session 4 - Tuples (Assignment, as return value)
- Session 5 - Dictionaries (operations and methods)
- Session 6 - Advance List processing, List Comprehension
- Session 7 - Simple Sort, Histogram
- Session 8 - Student Mark Statement
- Session 9 - Retail Bill preparation

Recap

- List is a sequence data type that hold heterogenous data.
- The following are the operations can be performed in List: creation, deletion, concatenation, slicing, repetition
- Python has in-built methods to perform on list such as append(), pop(), remove(), len(), index()

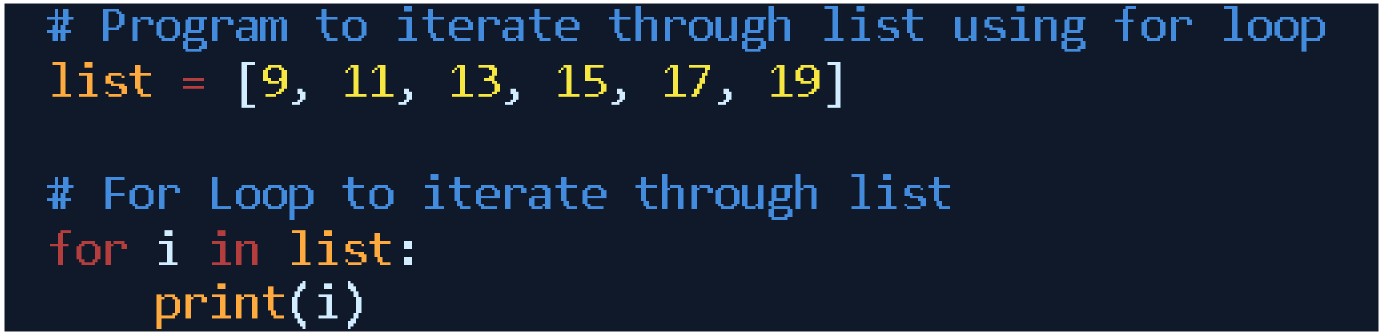
List Traversal / Looping

- A list is an ordered sequence of elements.
- It is a non-scalar data structure and mutable in nature.
- A list can contain distinct data types in contrast to arrays storing elements belonging to the same data types.
- List is equivalent to arrays in other languages, with the extra benefit of being dynamic in size.
- In Python, the list is a type of container in Data Structures, which is used to store multiple data at the same time.
- Unlike Sets, lists in Python are ordered and have a definite count.

List Traversal / Looping

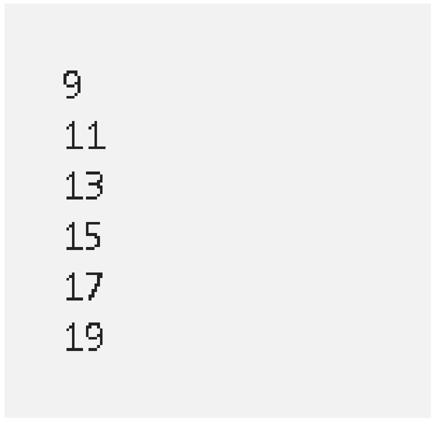
- There are different ways in which List can be traversed:
  - Iterate Through List in Python using For Loop
  - Iterate Through List in Python using While Loop
  - Iterate Through List in Python using Loop and Range
  - Iterate Through List in Python using negative indexes
  - Iterate Through List in Python using sliced list
  - Iterate Through List in Python using Enumerate Method
  - Iterate Through List in Python using Iter() and Next()
  - Iterate Through List in Python using zip()

Iterate Through List in Python using For Loop



```python
# Program to iterate through list using for loop
list = [9, 11, 13, 15, 17, 19]

# For Loop to iterate through list
for i in list:
    print(i)
```

Output



```
9
11
13
15
17
19
```

Iterate Through List in Python using While Loop

```python
# Program to loop through the list using while loop
list = [9, 11, 13, 15, 17, 19]

# Finding length of the list
length = len(list)
i = 0

# While Loop to iterate through list
while i < length:
    print(list[i])
    i += 1
```

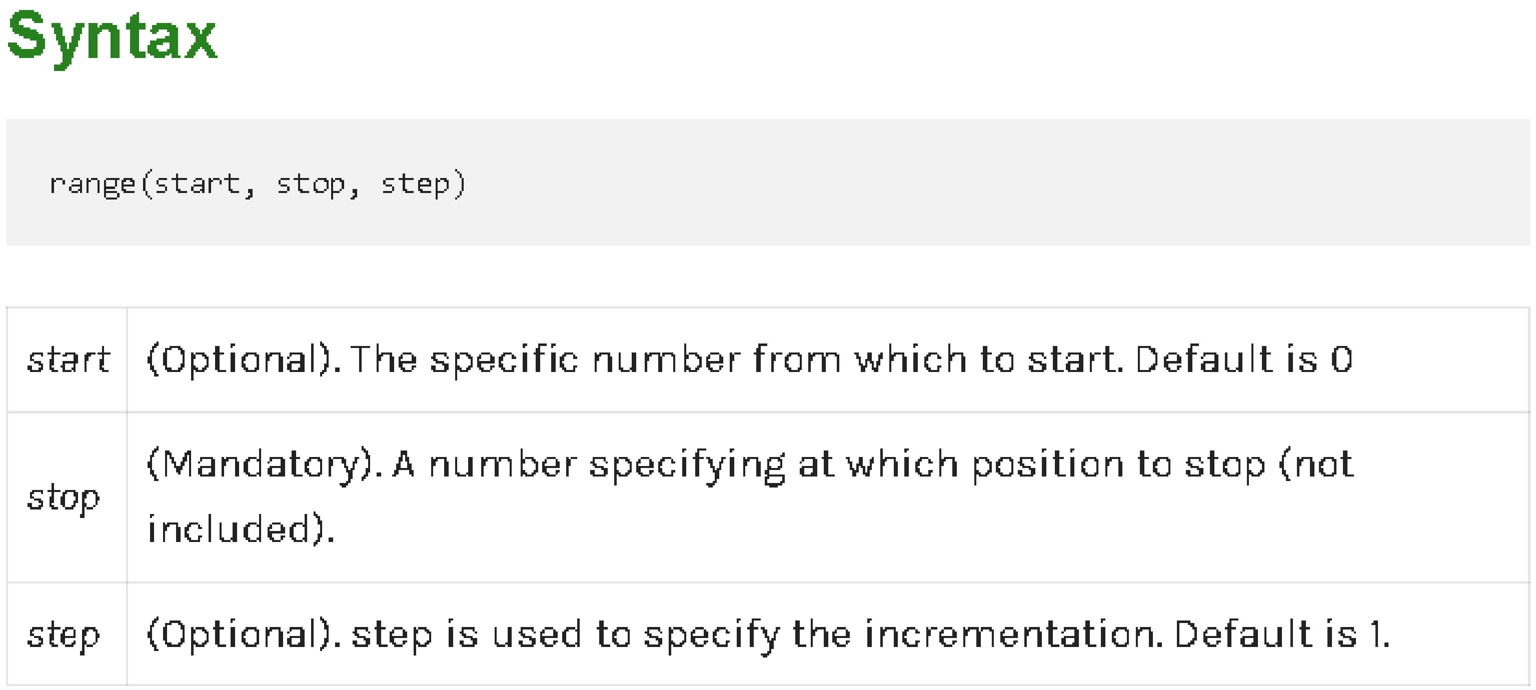
Output

```
9
11
13
15
17
19
```

Iterate Through List in Python using Loop and Range

- The range method can be used as a combination with for loop to traverse and iterate through a list.
- The range() returns a sequence of numerals, starting from 0 (default), and by default increment by 1, and stops before a specified number.

Iterate Through List in Python using Loop and Range

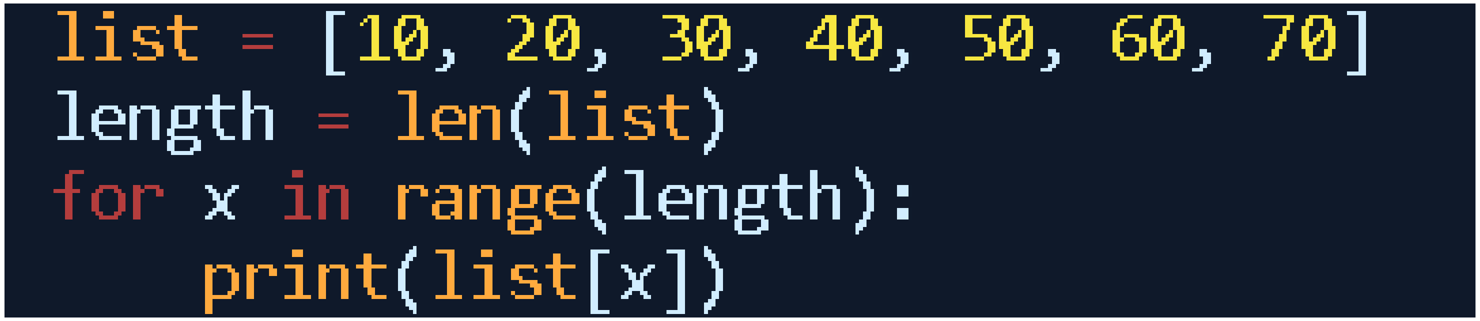


Syntax

```
range(start, stop, step)
```

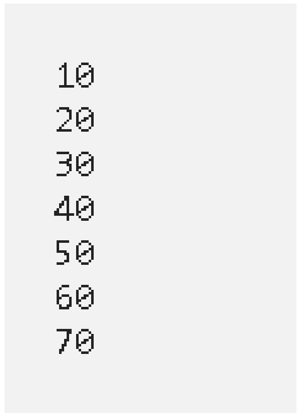
start	(Optional). The specific number from which to start. Default is 0
stop	(Mandatory). A number specifying at which position to stop (not included).
step	(Optional). step is used to specify the incrementation. Default is 1.

Iterate Through List in Python using Loop and Range



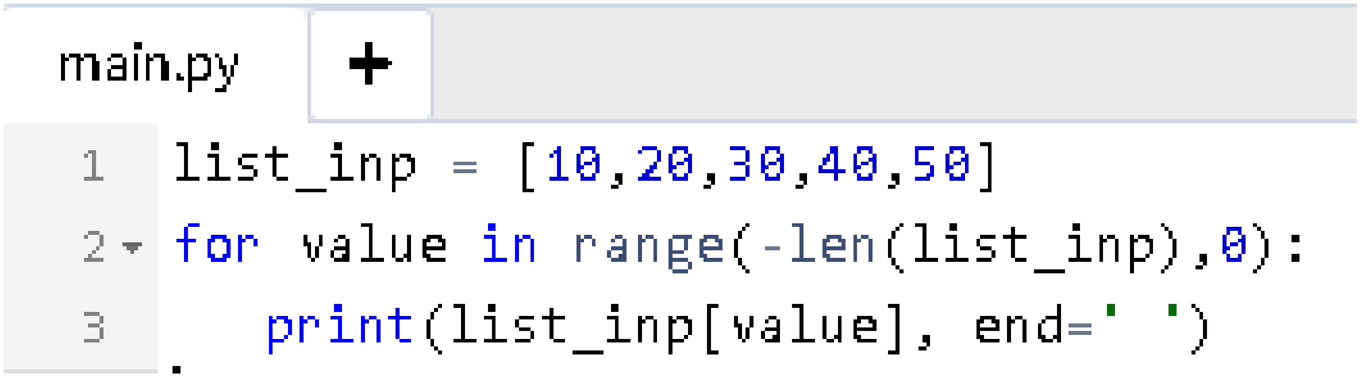
```python
list = [10, 20, 30, 40, 50, 60, 70]
length = len(list)
for x in range(length):
    print(list[x])
```

Output

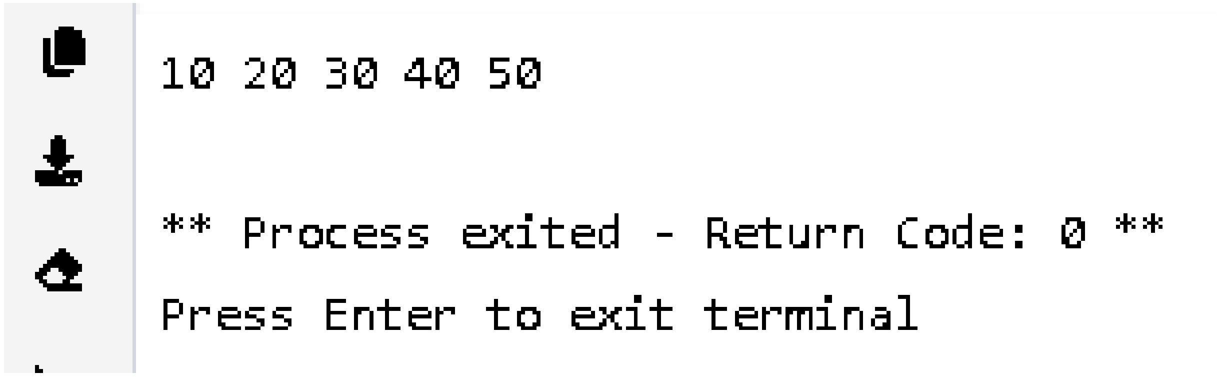


```
10
20
30
40
50
60
70
```

Iterate Through List in Python using negative indexes

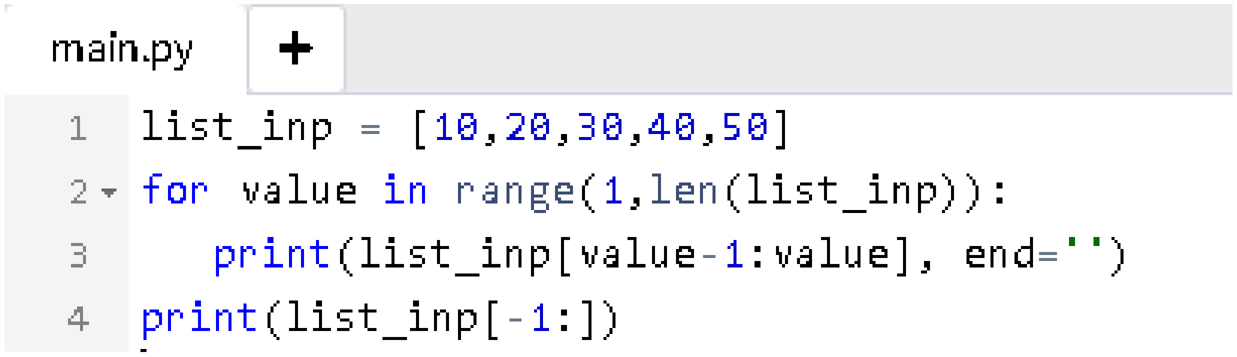


```python
list_inp = [10,20,30,40,50]
for value in range(-len(list_inp),0):
    print(list_inp[value], end=' ')
```

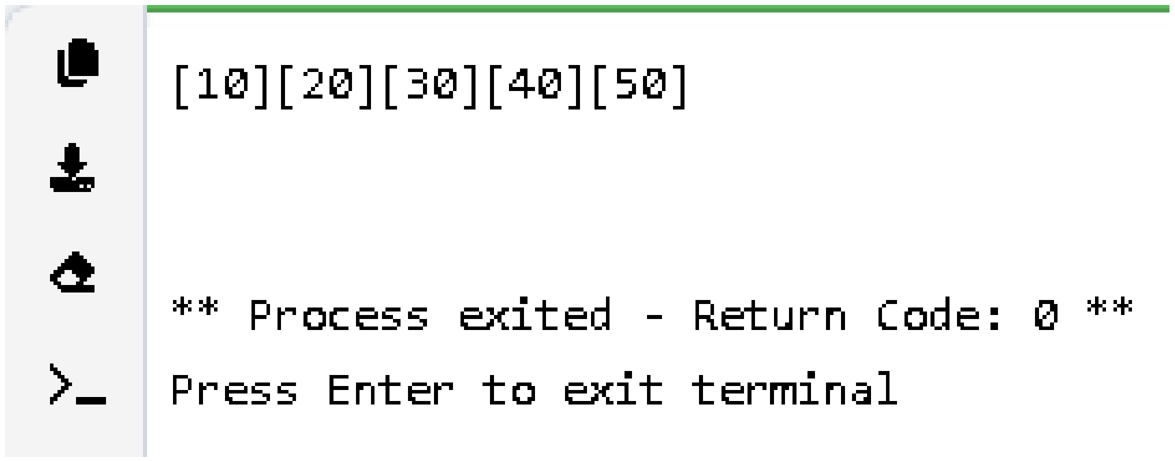


```
10 20 30 40 50


** Process exited - Return Code: 0 **
Press Enter to exit terminal
```

Iterate Through List in Python using sliced list


main.py

```python
list_inp = [10,20,30,40,50]
for value in range(1,len(list_inp)):
    print(list_inp[value-1:value], end='')
print(list_inp[-1:])
```

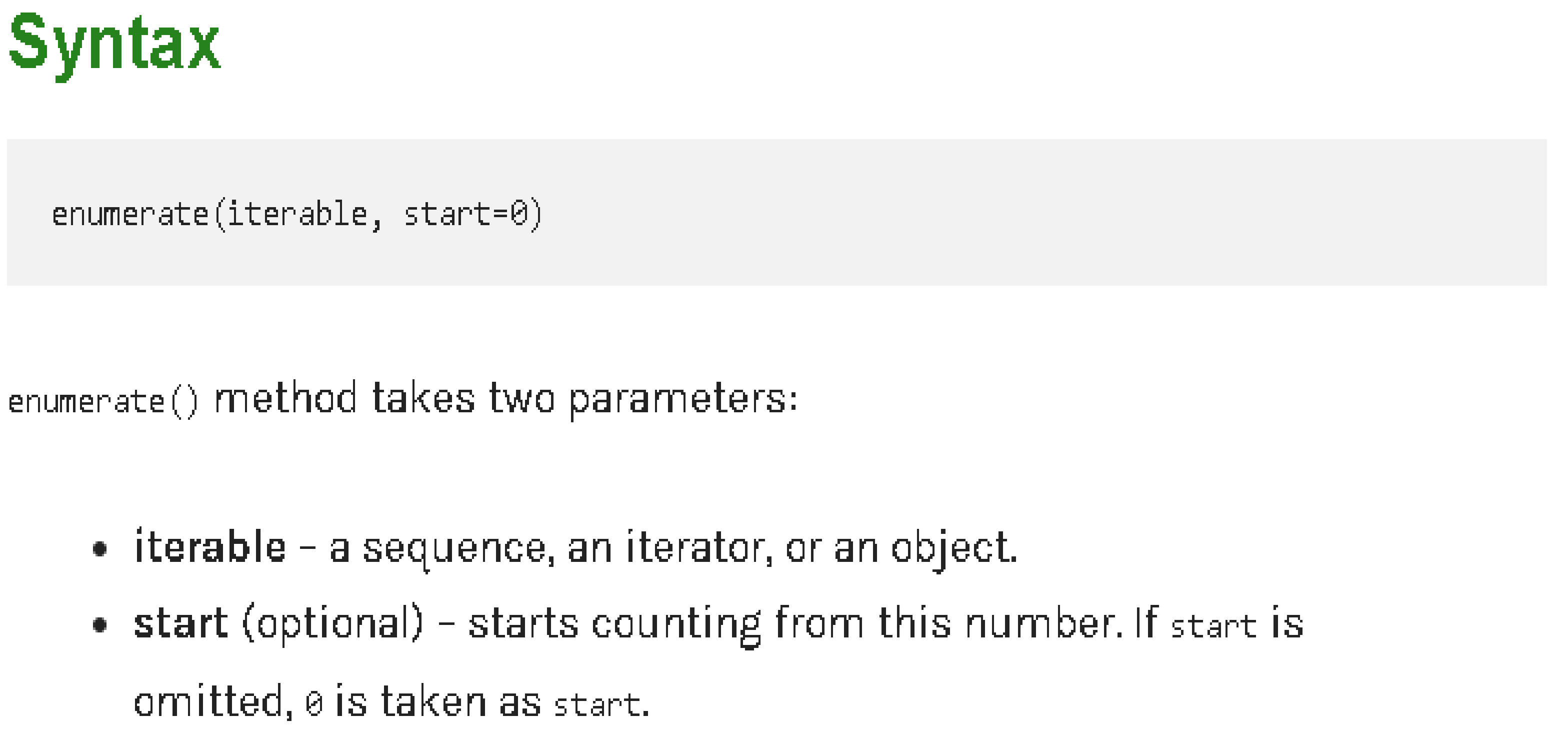


```
[10][20][30][40][50]



** Process exited - Return Code: 0 **
Press Enter to exit terminal
```

Iterate Through List in Python using Enumerate Method

- The **enumerate()** method adds counter to an iterable and returns it. And whatever the enumerate method returns, it will be a enumerate object.
- The main advantage of using the enumerate method is you can convert enumerate objects to **list and tuple** using the list() and tuple() method, respectively.

Iterate Through List in Python using Enumerate Method



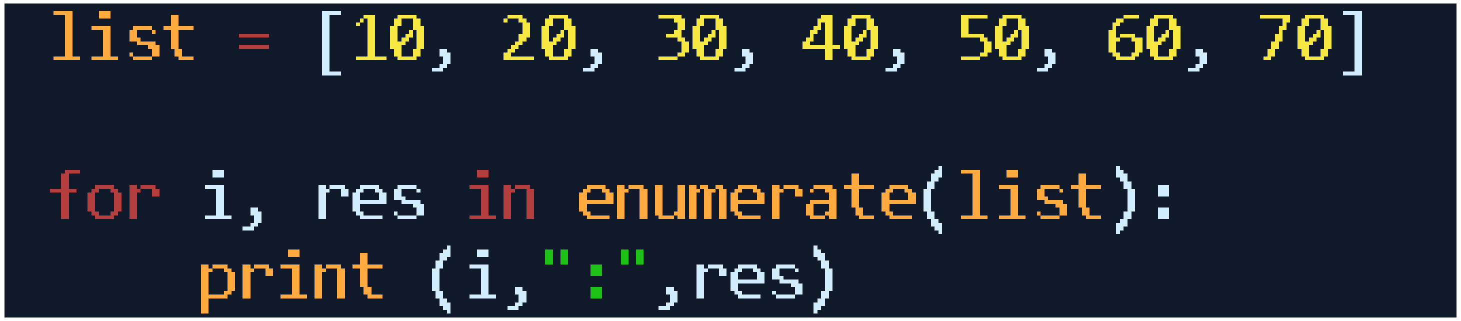
Syntax

```
enumerate(iterable, start=0)
```

`enumerate()` method takes two parameters:

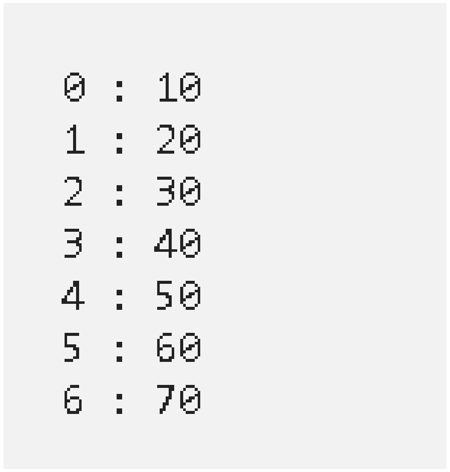
- **iterable** - a sequence, an iterator, or an object.
- **start** (optional) - starts counting from this number. If `start` is omitted, `0` is taken as `start`.

Iterate Through List in Python using Enumerate Method



```python
list = [10, 20, 30, 40, 50, 60, 70]

for i, res in enumerate(list):
    print (i,":",res)
```

Output

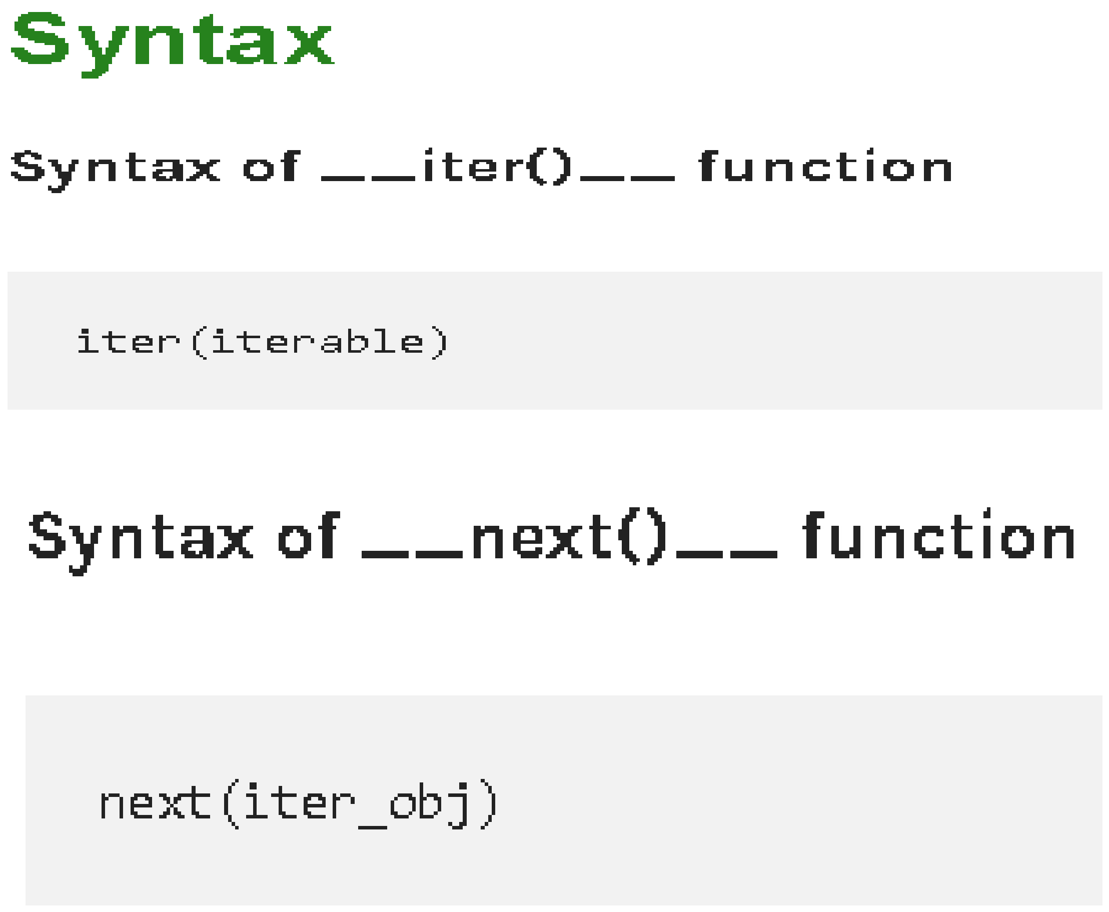


```
0 : 10
1 : 20
2 : 30
3 : 40
4 : 50
5 : 60
6 : 70
```

Iterate Through List in Python using Iter() and Next()

- To iterate a list using iterators in python we will use __iter()__ and __next()__ methods.
- In Python the __iter__() and __next__() are collectively knows as iterator protocol.
- Iterators are generally implemented within loops, comprehensions, generators, etc.
- They are simply an object that can be iterated upon (one element at a time). Internally, the for loop creates an iterator object, iter_obj by calling iter() on the iterable.

Iterate Through List in Python using Iter() and Next()



Syntax

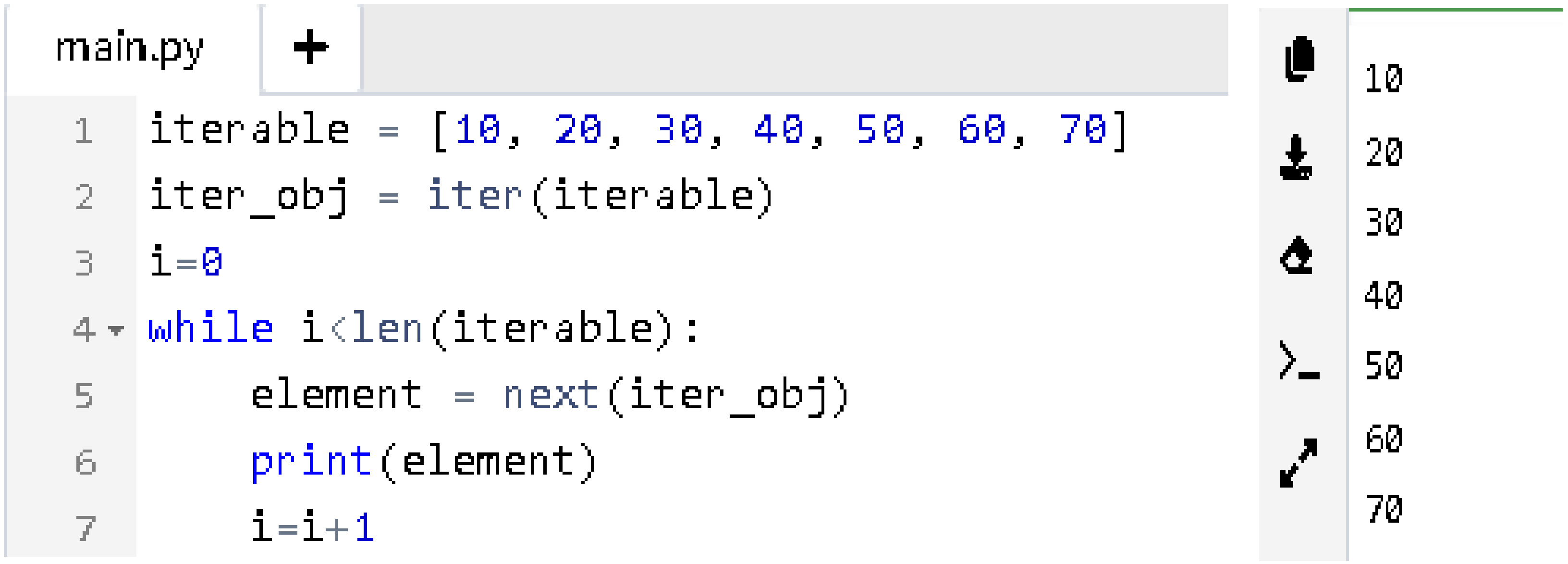
Syntax of __iter()__ function

```
iter(iterable)
```

Syntax of __next()__ function

```
next(iter_obj)
```

Iterate Through List in Python using Iter() and Next()



main.py

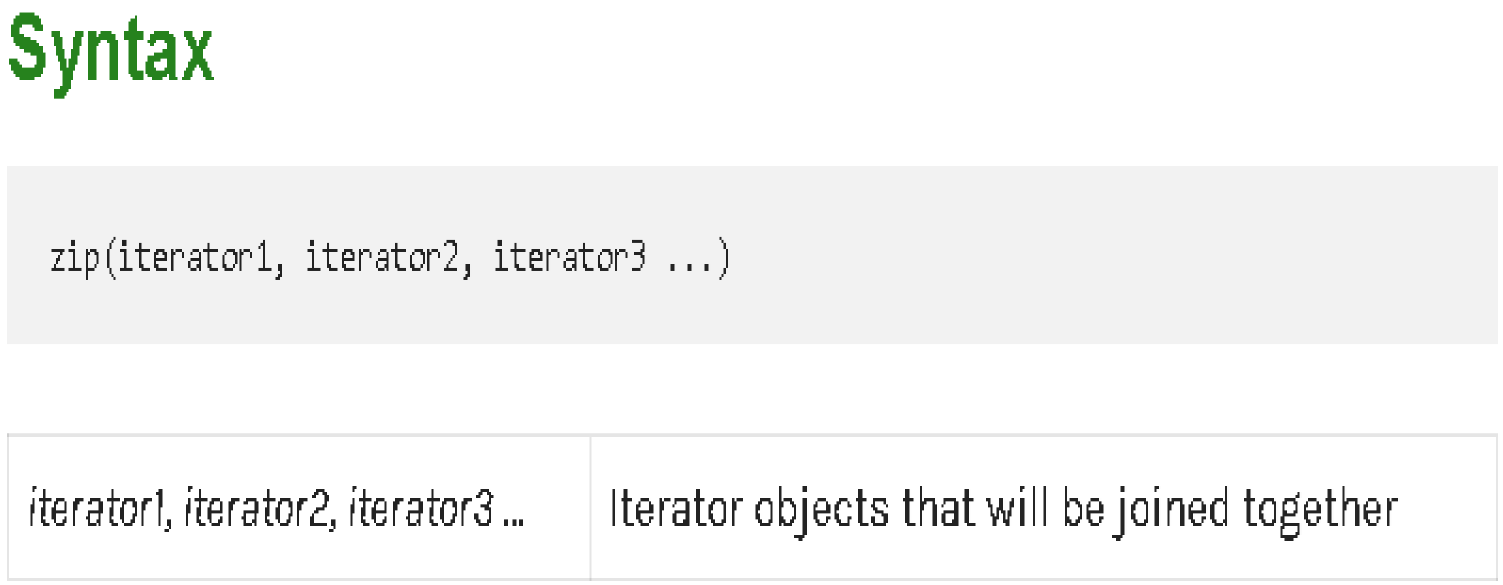
```python
iterable = [10, 20, 30, 40, 50, 60, 70]
iter_obj = iter(iterable)
i=0
while i<len(iterable):
    element = next(iter_obj)
    print(element)
    i=i+1
```

```
10
20
30
40
50
60
70
```

Iterate Through List in Python using zip()

- If you want to iterate through two lists simultaneously you can use the zip() method in Python. So what the zip() function does is it creates an iterator that will aggregate elements from two or more iterables.
- The **zip()** function in Python generates a zip object, which is an iterator of tuples.

Iterate Through List in Python using zip()



Syntax

```
zip(iterator1, iterator2, iterator3 ...)
```

iterator1, iterator2, iterator3 ...	Iterator objects that will be joined together

Iterate Through List in Python using zip()

```
num = [1, 2, 3, 4]
daypart = ['moring', 'afternoon', 'evening', 'night']
for (a, b) in zip(num, daypart):
    print (a, b)
```

Output

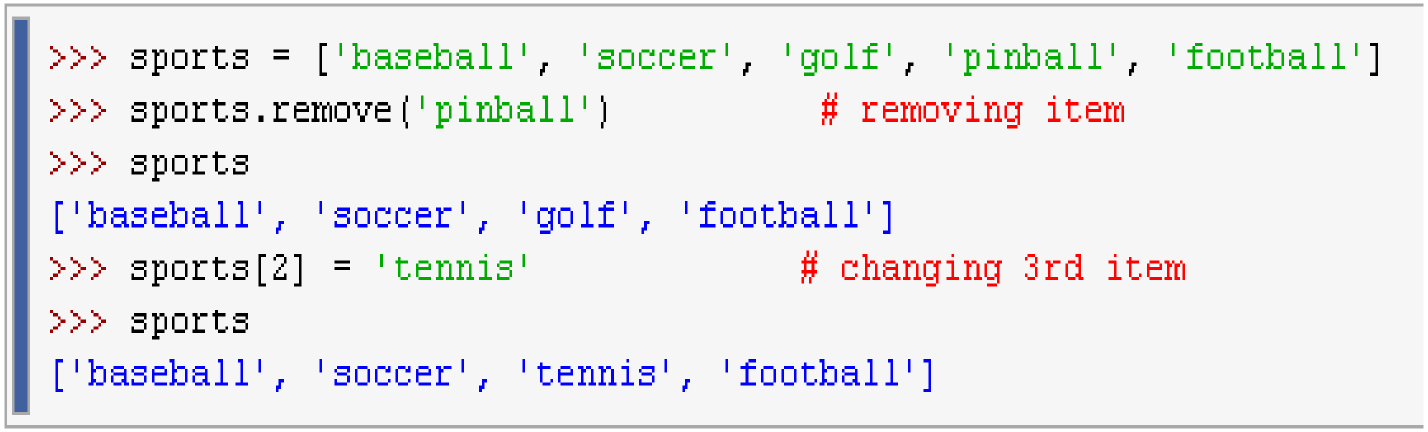
```
1 moring
2 afternoon
3 evening
4 night
```

List Mutability

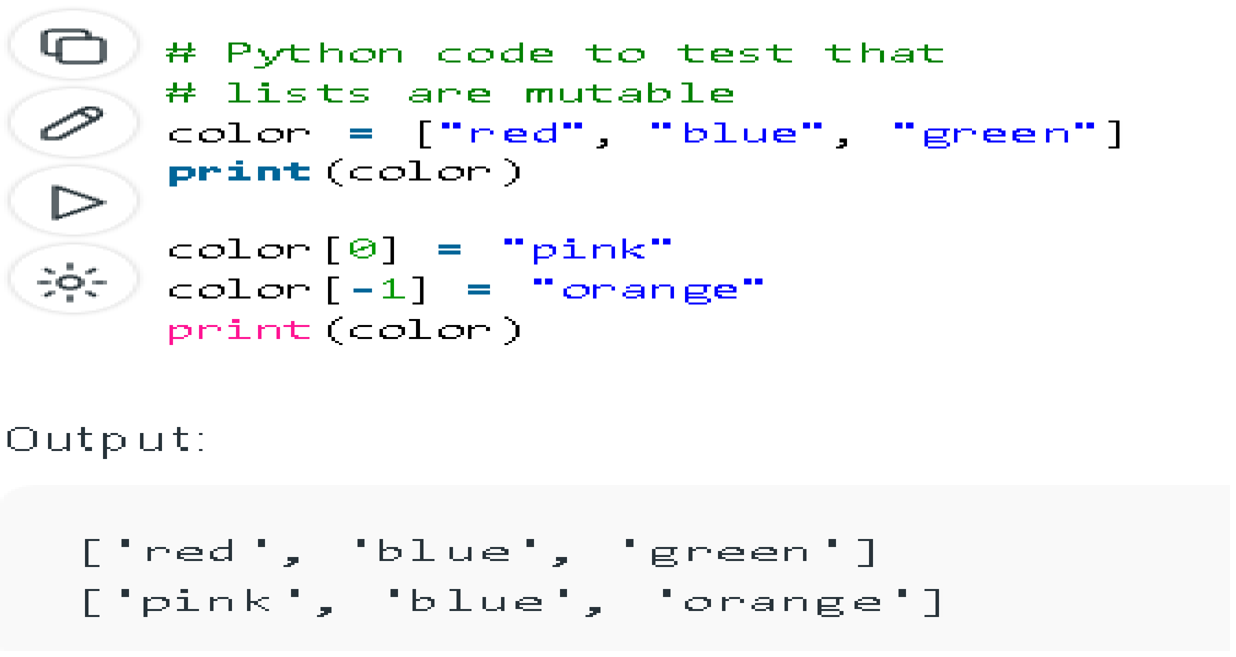
- List has mutable nature i.e., list can be changed or modified after its creation according to needs whereas tuple has immutable nature i.e., tuple can't be changed or modified after its creation.
- Mutable is a fancy way of saying that the internal state of the object is changed/mutated. So, the simplest definition is: An object whose internal state can be changed is mutable.

http://bit.ly/immutableThis – to learn more about mutability

List Mutability



```
>>> sports = ['baseball', 'soccer', 'golf', 'pinball', 'football']
>>> sports.remove('pinball')          # removing item
>>> sports
['baseball', 'soccer', 'golf', 'football']
>>> sports[2] = 'tennis'              # changing 3rd item
>>> sports
['baseball', 'soccer', 'tennis', 'football']
```



```
# Python code to test that
# lists are mutable
color = ["red", "blue", "green"]
print(color)

color[0] = "pink"
color[-1] = "orange"
print(color)
```

Output:

```
['red', 'blue', 'green']
['pink', 'blue', 'orange']
```

Summary

- List can be traversed or looped or visited in 8 ways.
- List is mutable – Changes can be made in the list using the index or position of the list.

THANK YOU