LIST

* LISTS

A list is a sequence of values. They can be of any type. The values in a list are called elements or items.

* Creating a List

There are several ways to create a new list

a) Enclose the elements in square brackets ([ ]).

E.g: list = [10, 'Jovita', '2017', '2.6']

b) List with no elements is called an empty list

E.g: empty_List = []

c) A list that is an element of another list is called nested list

E.g: ['Dell', 2.0, [50, 100]]

* Accessing List elements

Accessing the elements of a list is same as for accessing the characters of a string that is using the bracket operator.

The expression inside the brackets specifies the index. The indicies start as 0 and the negative indicies start from -1 which refers to the last item in a list.

* E.g:

```
A = [1, 2, 3, 4, 'A', 'B', ['C', 'D']]
print ("A[0] = ", A[0], "and ", "A[3] = ", A[3])
```

Output:

A[0] = 1 and A[3] = 4

LIST OPERATIONS

* Concatenation operation

Concatenation means joining two operands by linking them end-to-end. In list concatenation, + operator concatenate two lists with each other and produce a third list.

E.g:

```
a = [1,2,3]
b = [4,5,7]
c = a + b
print (c)
```

Output :

[1,2,3,4,5,7]

* Repeat operation

Lists can be replicated or repeated or repeatedly concatenated with the asterisk operator "*". The * operator repeats a list in given number of times.

E.g:

```
odd = [1,3,5]
even = [2,4,6]
A = odd + even
print ("A * 3 =", A * 3)
```

Output :

A * 3 = [1,3,5,2,4,6,1,3,5,2,4,6,1,3,5,2,4,6]

LIST SLICES

An individual element of a list is called a slice. (selecting an element(s) of a list)

⇒ ([] and [m:n]) Subsets of list can be taken using the slice operator with two indices in square brackets separated by a colon. A range of items in a list can be accessed using the slicing operator.

E.g :

```
birds = ['parrot', 'Dove', 'duck', 'cuckoo']
print ("birds [0:2] = ", birds [0:2])
print ("birds [:] = ", birds [:])
print ("birds [:3] = ", birds [:3])
print ("birds [2:] = ", birds [2:])
```

Output :

```
birds [0:2] = ['parrot', 'Dove']
birds [:] = ['parrot', 'Dove', 'duck', 'cuckoo']
birds [:3] = ['parrot', 'Dove', 'duck']
birds [2:] = ['duck', 'cuckoo']
```

LIST METHODS

Method	Meaning / Eg :	

append()	Add an element to the end of the list	>>> t = ['a', 'b', 'c']
		>>> t.append ('d')
		>>> t ['a', 'b', 'c', 'd']

extend()	Add all elements of a list to another list.	>>> t1 = ['a', 'b']
		>>> t2 = ['c', 'd']
		>>> t1.extend (t2)
		>>> t1
		['a', 'b', 'c', 'd']

insert()	Insert an item at the defined index.
	• List.insert(index, obj)
		>>> fls = ['a', 'c']
		>>> fls.insert(1, 'b')
		>>> print (fls)
		['a', 'b', 'c']

remove()	Removes an item from the list.
	• list.remove (element)
		>>> nos = [1, 2, 3]
		>>> nos.remove (3)
		>>> print (nos)
		[1, 2]

pop()	Removes and returns an element at the given index. • list.pop(pos)	>>> al = ['aA', 'bB']
		>>> al.pop(1)
		>>> banana

clear()	Removes all item in the list • list.clear()	>>> lst = ['$', 'ch']
		>>> lst.clear()
		>>> print(lst)
		[]

index()	Returns the index of the first matched item. • List.index(element)	>>> frs = ['grape', 'apple']
		>>> frs.index('apple')
		1

count()	Returns the count number of items passed as an argument. • List.count(value)	>>> ns = ['b', 'b', 'o', 'h']
		>>> ns.count('b')
		2

sort()	Sort items in a list in ascending order. • list.sort(reverse = True | False, key = myFunc)	def myFunc(e): return len(e) bi = ['Mine', 'tech'] bi.sort(key = myFunc) print(bi) Output: ['Mine', 'tech']

reverse()	Reverse the order of items in the list. • List.reverse()	>>> lst = [1, 2, 3]
		>>> lst.reverse()
		>>> print(lst)
		[3, 2, 1]

copy()	Returns a shallow copy of the list. • list.copy()	>>> lst = ['a', 'b', 'c']
		>>> lst.copy()
		['a', 'b', 'c']

Len ()	Returns the number of elements in a list.
	• len (List)
		>>> lst1 = [1,2,3]
		>>> print (len (lst1))
		3

max ()	Returns the maximum value from the list
	• max (List)
		>>> Lst 2 = [2, 8, 9]
		>>> print (max (Lst 2))
		9

min ()	Returns the minimum value from the list
	• min (List)
		>>> Lst 3 = [10, 12, 1]
		>>> print (min (lst3))
		1

LIST LOOP [TRAVERSING A LIST]

1. Traversing a list means, process or go through each element of a list sequentially. When the list elements are processed within a loop, the loop variable or a separate counter is used as an index into the list which prints each counter position elements till the end-1. This patteren of computation is called a list traversal.

* Syntax : for <List_Item> in <List> :
statement to process <List_Item>

* Code :

```
for icecreams in icecreams :
        print (icecreams)
```

2. The 'for loop' works well to read the elements of the list. But to write or update the elements, it is needed to specify the indices. A common way to do that is to combine the built-in functions range and len.

* Syntax : for <Index> in range (len(List)) :
Statement to process <List.[Index]>

*. Code :

```
icecreams = ['vanila', 'strawberry', 'mango']
for i in range (len (icecream)) :
    print ( icecream [i])
```

MUTABILITY

The lists are mutable (Changeable). When the bracket operator appears on the left side of an assignment, it identifies the element of the list that will be assigned.

* Code :

```
birds = ['parrot', 'pigeon', 'dove', 'owl', 'penguin']
birds[1] = 'duck'
print ('Birds', birds)
```

Output :

```
Birds ['parrot', 'duck', 'dove', 'owl', 'penguin']
```

LIST MEMBERSHIP

in and not in are the membership operators in Python. They are used to test whether a value or variable is found in a sequence or not.

- in Operator

The in operator tests whether an element is a member of a list or not. If the element is a member in the list, then it will produce True otherwise False.

* Code :

```
icecreams = ['vanilla', 'strawberry', 'mango']
print ('strawberry' in icecreams)
Print ('chocolate' in icecreams)
```

Output :

True
False

- not in Operator

The not in operator evaluates to true if it doesn't find the element in the list and otherwise false.

* Code :

```
icecreams = ['vanilla', 'strawberry', 'mango']
print('strawberry' not in icecreams)
print('chocolate' not in icecreams)
```

Output :

False
True

~ Thankyou