String Indexing

```
str1 = "Python"
```

-6	-5	-4	-3	-2	-1
P	y	t	h	o	n
0	1	2	3	4	5

str1[0]= P= str1[-6]
str1[1]= y= str1[-5]
str1[2]= t= str1[-4]
str1[3]= h= str1[-3]
str1[4]= o= str1[-2]
str1[5]= n= str1[-1]

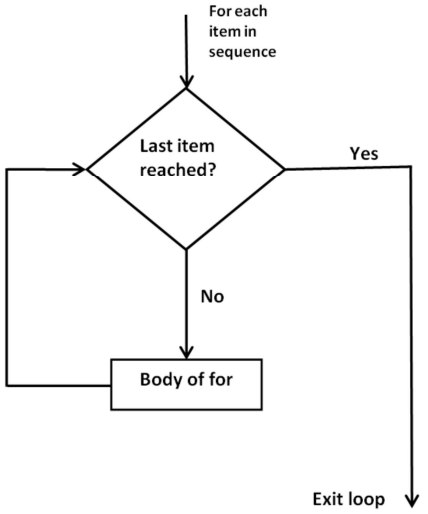
Membership Operators

in and **not in** are membership operators, used to test whether a value or variable is in a sequence or not. These operators return **True** or **False** boolean values.

Operator	Illustration	Syntex
in	True: value is in the sequence False: value is not in the sequence	a in b
not in	True: value is not in the sequence False: value is in the sequence	a not in b

For Loop

- A for loop is used to iterate over a sequence (that is either a string, a tuple, a set, a list, or a dictionary).




```
for x in sequence:
  body of for loop
```

range() Function

- A range() function allows user to generate a sequence of numbers with in given range.

Syntax:

range(start, stop, step)

start: integer value (starting point of the sequence, optional, by default takes 0)
stop: integer value (before which the sequence of integers is to be returned, does not return stop value in sequence)
step: integer value (increment size, optional, by default takes 1)

Example

```
str1 = "Python"
print("Printing str1 without using index")
for x in str1:
  print(x,end="")
print("\nPrinting str1 using index")
for x in range(len(str1)):
  print(str1[x],end="")
```

Output

```
Printing str1 without using index
Python
Printing str1 using index
Python
```

len()

- len() returns length of the sequence.

Syntax:

len(sequence)

```
>>>str1 = "Python "
>>>print(len(str1))
7
>>>str2 = ""
>>>print(len(str2))
0
>>>
```

split()

- split() returns a list of words delimited by the provided substring.
- If maximum number of split is specified, it is done from the left.

Syntax:

split(separator, maxsplits)

separator(optional): split the string at specified separator. If the separator is not specified, then space and new line are treated as separator.

maxsplits(optional): maximum number of splits.

split()

```
>>>str1 = "Python is a high level language"
>>>print(str1.split())
['Python', 'is', 'a', 'high', 'level', 'language']
>>str2 = "Python, is, a, high, level, language"
>>>print(str2.split(','))
['Python', ' is', ' a', ' high', ' level', ' language']
>>str3 = "Python is a high level language"
>>>print(str3.split(','))
['Python is a high level language']
>>str4 = "Python is a high level language"
>>>print(str4.split(' ',3))
['Python', 'is', 'a', 'high level language']
```

slice()

- The slice() method creates a slice object representing the set of indices specified by range(start, stop, step).

Syntax:

slice(start, stop, step)

start(optional): starting integer from where the slicing of the object starts. If the value of start is not specified, then 0 is taken as start value.
stop: integer until which the slicing takes place. The slicing stops at index **stop - 1**.
step(optional): increment size, if the value of step is not specified, then 1 is taken as step value.

slice()

```
>>>str1 = "Python"
>>>objects = slice(4)
>>>print(str1[objects])
Pyth
>>str2 = "Python"
>>>objects = slice(1,5,2)
>>>print(str2[objects])
yh
```

isdecimal()

- The isdecimal() returns:
  - True: if all characters in the string are decimal characters.
  - False: if at least one character is not decimal character.

Syntax:

string.isdecimal()

```
>>>str1 = "123456"
>>>print(str1.isdecimal())
True
>>str2 = "123abc456"
>>>print(str2.isdecimal())
False
```

find()

- **Syntax:**

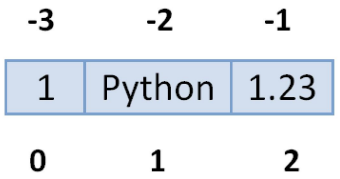
  string.find(substring, start, end)
- **substring:** substring to be searched in the string.
- **start** and **end** (optional): substring is searched within the string[start:end]
- find() returns index of the first occurrence of the substring if it is present in the string, else, returns -1.

```
>>>str1 = "Python is a high level language"
>>>print(str1.find("high"))
12
>>str2 = "Python is a high level language"
>>>print(str2.find("high",1,10))
-1
```

List

- A list is created by placing all the elements inside a square bracket [ ], separated by commas.
- The elements of a list can be of different data-type.

```
list1 = [1, "Python", 1.23]
```



list1[0]= 1= list1[-3]

list1[1]= "Python"= list1[-2]

list1[2]= 1.23= list1[-1]

List Methods

Method	Explanation
append()	Inserts an element at the end of the list
extend()	Add all elements of a list to the another list
insert()	Insert an item at the defined index
count()	Returns the count of number of items passed as an argument
index()	Returns the index of the first matched item
remove()	Removes an item from the list

List Methods

pop()	Removes and returns an element at the given index
reverse()	Reverse the order ot items in the list
sort()	Sort items in a list in specified order
copy()	Returns a copy of the list
clear()	Clears the list by removing all items from it and return an empty list.

append() Method

- append() method adds an element/item at the end of the list.

Syntax:

list.append(item)

```
>>>Courses = ['IC101P', 'IC110', 'IC140', 'IC152']
>>>Courses.append('IC160')
>>>print(Courses)
['IC101P', 'IC110', 'IC140', 'IC152', 'IC160']
>>>Credits = [2, 3, 4, 4, 3]
>>>Courses.append(Credits)
>>>print(Courses)
['IC101P', 'IC110', 'IC140', 'IC152', 'IC160', [2, 3, 4, 4,
    3]]
```

extend() Method

Syntax:

list1.extend(list2)

- extend() method adds elements of the list2 to list1 at the end.

```
>>>Courses = ['IC101P', 'IC110', 'IC140', 'IC152', 'IC160']
>>>Credits = [2, 3, 4, 4, 3]
>>>Courses.extend(Credits)
>>>print(Courses)
['IC101P', 'IC110', 'IC140', 'IC152', 'IC160', 2, 3, 4, 4,
    3]
```

? Questions:
Can we use any other sequence instead of list data-type in place of list2 in extend() method (i.e. list1.extend('string'), list1.extend(('a',1,1.23)) etc.)? If, yes, then how it will work?

insert() Method

- insert() method inserts an element to the list at a given index.

Syntax:

list.insert(index, element)

```
>>>Courses = [ 'IC110', 'IC140', 'IC152', 'IC160']
>>>Courses.insert(3,'HS106')
>>>print(Courses)
['IC110', 'IC140', 'IC152', 'HS106', 'IC160']
>>>Courses.insert(0,'IC101P')
>>>print(Courses)
['IC110P', 'IC110', 'IC140', 'IC152', 'HS106', 'IC160']
```

count() Method

- count() method return the number of occurrences of an element in the list.

Syntax:

list.count(element)

```
>>>list1 = [2, 'a', '2', {'a',2}, (2,2), [2,'a'], 2]
>>>print(list1.count(2))
2
>>>print(list1.count('a'))
1
>>>print(list1.count((2,2)))
1
>>>print(list1.count([2, 'a']))
1
```

index() Method

- index() method returns the index of the first occurrence of the given element in the list.

Syntax:

list.index(element)

```
>>>Courses = ['IC140', 'IC160', 'IC110', 'IC152','HS106',
    'IC152']
>>>print(Courses.index('IC152'))
3
>>>Courses = ['IC140', 'IC160', 'IC110', 'IC152','HS106',
    'IC152']
>>>print(Courses.index('IC152P'))
ValueError: 'IC152P' is not in list
```

remove() Method

- remove() method searches for the given element in the list and removes the first matching element.
- If the given element does not exist in the list then remove() method through an error.

Syntax:

list.remove(element)

```
>>>Courses = ['IC110', 'IC140', 'IC152', 'IC160', 'IC152']
>>>Courses.remove('IC152')
>>>print(Courses)
['IC110', 'IC140', 'IC160', 'IC152']
>>>Courses.remove('HS106')
ValueError: list.remove(x): x not in list
```

pop() Method

- pop() method removes the item at the given index from the list and also returns the removed item.

Syntax:

list.pop(index)

index: integer value (optional, by default takes -1 if value is not specified)

```
>>>Courses = ['IC110', 'IC140', 'IC152', 'IC160', 'IC152']
>>>A = Courses.pop(2)
>>>print(A)
IC152
>>> print(Courses)
['IC110', 'IC140', 'IC160', 'IC152']
>>>Courses.pop()
>>>Courses
['IC110', 'IC140', 'IC160']
```

reverse() Method

- reverse() method reverses the elements of a list.

Syntax:

list.reverse()

```
>>>List = [1, 2, 3, 4, 5, 'a', 'b', 'c']
>>>List.reverse()
>>>print(List)
['c', 'b', 'a', 5, 4, 3, 2, 1]
>>>Courses = ['IC140', 'IC160', 'IC110', 'IC152']
>>>Courses.reverse()
>>>print(Courses)
['IC152', 'IC110', 'IC160', 'IC140']
```

sort() Method

```
>>>NumList = [5, 15, 0, 20, 10]
>>>NumList.sort()
>>>print(NumList)
[0, 5, 10, 15, 20]
>>>Courses = ['IC140', 'IC160', 'IC110', 'IC152','HS106',
    'IC152']
>>>Courses.sort(reverse=True)
>>>print(Courses)
['IC160', 'IC152', 'IC152', 'IC140', 'IC110', 'HS106']
>>>TupleList = [(1,5), (5, 2), (2, 3), (1, 3),  (2,1)]
>>>TupleList.sort()
>>>print(TupleList)
[(1, 3), (1, 5), (2, 1), (2, 3), (5, 2)]
>>>TupleList = [(1,5), (5, 2), (2, 3), (1, 3),  (2,1)]
>>>TupleList.sort(key = lambda x: x[1])
>>>print(TupleList)
[(2, 1), (5, 2), (2, 3), (1, 3), (1, 5)]
```

copy() Method

- copy() method returns a shallow copy of the list.

Syntax:

list.copy()

```
>>>old_list = ['a', 1, 'b', 2]
>>>new_list = old_list
>>>print("Old List:", old_list)
Old List: ['a', 1, 'b', 2]
>>>print("New List:", new_list)
New List: ['a', 1, 'b', 2]
>>>old_list.append('c')
>>>new_list.append(3)
>>>print("Old List:", old_list)
Old List: ['a', 1, 'b', 2, 'c', 3]
>>>print("New List:", new_list)
New List: ['a', 1, 'b', 2, 'c', 3]
```

copy() Method

```
>>>print(id(old_list))
2851368521352
>>>print(id(new_list))
2851368521352
>>>old_list = ['a', 1, 'b', 2]
>>>new_list = old_list.copy()
>>>print("Old List:", old_list)
Old List: ['a', 1, 'b', 2]
>>>print("New List:", new_list)
New List: ['a', 1, 'b', 2]
>>>old_list.append('c')
>>>new_list.append(3)
>>>print("Old List:", old_list)
Old List: ['a', 1, 'b', 2, 'c']
>>>print("New List:", new_list)
New List: ['a', 1, 'b', 2, 3]
>>>print(id(old_list))
2851368103432
>>>print(id(new_list))
2851368423624
```

clear() Method

- clear() method removes all items from the list.

Syntax:

list.clear()

```
>>>List = ['a', 1, 'b', 2]
>>>List.clear()
>>>print("Updated List:", List)
Updated List: []
```

del Statement

Syntax:

del **object_name**

object_name can be variables, user-defined objects, lists, items within lists, dictionaries etc.

```
>>>List = ['a', 1, 'b', 2]
>>>del List[1]
>>>print("Updated List:", List)
Updated List: ['a', 'b', 2]
>>>del List[:]
>>>print("Updated List:", List)
Updated List: []
>>>del List
>>>print(List)
NameError: name 'List' is not defined
```

Tuple

- A tuple is created by placing all the elements inside a parenthesis ( ), separated by commas or just by separating items by comma.
- The elements of a tuple can be different data-type.

```
tup = (1, "Python", 1.23)
```

-3	-2	-1
1	Python	1.23
0	1	2

tup[0]= 1= tup[-3]

tup[1]= "Python"= tup[-2]

tup[2]= 1.23= tup[-1]

Concatenation and Repetition of Tuples

Questions:
Do tuple support append, insert and extend methods?

- + operator is used for concatenation of tuples.
- * operator is used for the repetition of tuples.

```
tup1 = (1, "Python", 1.23)
tup2 = ('a', 'b', 3)
tup3 = tup1 + tup2
tup4 = tup3*2
print(tup3)
(1, 'Python', 1.23, 'a', 'b', 3)
print(tup4)
(1, 'Python', 1.23, 'a', 'b', 3, 1, 'Python', 1.23, 'a',
    'b', 3)
```

Question

Questions:
Do all the methods we have seen for list, will work for tuple? Figure out which will work and why?

sorted() Function

- sorted() function sorts any sequence (tuple, list, string etc.) and returns a list with the elements in sorted manner, without modifying the original sequence.

Syntax

sorted(sequence, key = ..., reverse= ...)

sorted() Function

```
>>>NumList = [5, 15, 0, 20, 10]
>>>print(sorted(NumList))
[0, 5, 10, 15, 20]
>>>print(NumList)
[5, 15, 0, 20, 10]
>>>Courses = 'IC140', 'IC160', 'IC110', 'IC152','HS106',
    'IC152'
>>>print(Courses)
('IC140', 'IC160', 'IC110', 'IC152', 'HS106', 'IC152')
>>>NewCourses = sorted(Courses, reverse=True)
>>>print(NewCourses)
['IC160', 'IC152', 'IC152', 'IC140', 'IC110', 'HS106']
>>>Tup = [(1,5), (5, 2), (2, 3), (1, 3),  (2,1)]
>>>New_Tup = sorted(Tup, key = lambda x: x[1])
>>>print(New_Tup)
[(2, 1), (5, 2), (2, 3), (1, 3), (1, 5)]
```

reversed() Function

- reversed() function returns the reversed iterator of the given sequence.

Syntax

reversed(sequence)

```
>>>NumList = [5, 15, 0, 20, 10]
>>>print(reversed(NumList))
<list_reverseiterator object at 0x00000297E2CD30B8>
>>>print(tuple(reversed(NumList)))
(10, 20, 0, 15, 5)
>>>Courses = 'IC140', 'IC160', 'IC110', 'IC152','HS106',
    'IC152'
>>>print(reversed(Courses))
<reversed object at 0x00000297E2CD30B8>
>>>print(list(reversed(Courses)))
['IC152', 'HS106', 'IC152', 'IC110', 'IC160', 'IC140']
```

Difference Between Tuple and List

- The main difference between list and a tuple is the fact that lists are mutable (i.e. elements of list can be modified after its creation) whereas tuples are immutable(i.e., elements of tuple cannot be changed or modified after its creation).
- List has more functionality than tuple.
- Lists has variable length, tuple has fixed length.
- Tuples are more memory efficient as compare to lists
- Tuple operations have smaller size than that of list, which makes it a bit faster.

Set

- A set() is an unordered collection of objects.
- Every element of the set is unique (no duplicates) and must be immutable (i.e. cannot be changed).
- The set itself is mutable, we can add or remove items from it.
- A set is created by placing all the elements inside curly braces {}, separated by comma.
- The elements of a set can be different data-type.
- Set does not support indexing or slicing.

Set

- A set() is an unordered collection of objects.
- Every element of the set is unique (no duplicates) and must be immutable (i.e. cannot be changed).
- The set itself is mutable, we can add or remove items from it.
- A set is created by placing all the elements inside curly braces {}, separated by comma.
- The elements of a set can be different data-type.
- Set does not support indexing or slicing.

```
>>>emptySet = set()
>>>print("Empty set :", emptySet)
Empty set: set()
>>>print(type(emptySet))
<class 'set'>
>>>mixedSet = {1, 'a', 1.23, 1, 'python'}
>>>print("Mixed set:", mixedSet)
Mixed set: {1, 'python', 1.23, 'a'}
```

Set Methods

Method	Explanation
add()	Add an element to the set
update()	Adds elements from a sequence (passed as an argument) to the set .
discard()	Removes an element from the set if it is a member
remove()	Removes an element from the set
pop()	Removes and returns an arbitrary element of set

Set Methods

union()	Returns the union of sets
intersection()	Returns the intersection of sets
difference()	Returns the difference of two or more sets
symmetric_difference()	Returns the symmetric difference of two sets

add() Method

- add() method adds a given element to set (if element is not in the set).

Syntax

set.add(element)

```
>>>NumSet = {5, 15, 0, 20, 10}
>>>NumSet.add(25)
>>>print(NumSet)
{0, 5, 10, 15, 20, 25}
>>>Courses = {'IC140', 'IC160', 'IC110', 'IC152','HS106',
    'IC152'}
>>>Courses.add('IC110')
>>>print(Courses)
{'HS106', 'IC152', 'IC140', 'IC110', 'IC160'}
```

update() Method

- update() method adds elements from a sequence (passed as an argument) to the set (calling the update() method).

Syntax

set.update(sequence)

```
>>>NumSet = {5, 15, 0, 20, 10}
>>>string = "hello"
>>>NumSet.update(string)
>>>print(NumSet)
{0, 5, 'e', 10, 'l', 15, 20, 'h', 'o'}
>>>List = [1, 'a', (5,'a')]
>>>NumSet.update(List)
>>>print(NumSet)
{0, 1, 5, 'e', 10, 'l', 15, 20, 'h', 'a', (5, 'a'), 'o'}
>>>Courses = {'IC140', 'IC160', 'IC110', 'IC152','HS106',
    'IC152'}
>>>Courses.update(NumSet)
>>>print(Courses)
{0, 1, 'IC140', 5, 'IC110', 'e', 10, 'l', 15, 'IC152',
    'IC160', 20, 'h', 'a', (5, 'a'), 'o', 'HS106'}
```

discard() Method

- discard() method removes an element from the set, if present.

Syntax

set.discard(element)

```
>>>NumSet = {5, 15, 0, 20, 10}
>>>NumSet.discard(0)
>>>print(NumSet)
{5, 10, 15, 20}
>>>NumSet.discard(25)
>>>print(NumSet)
{5, 10, 15, 20}
```

remove() Method

- remove() method removes the given element from the set if present, otherwise throughs an error.

Syntax

set.remove(element)

```
>>>NumSet = {5, 15, 0, 20, 10}
>>>NumSet.remove(0)
>>>print(NumSet)
{5, 10, 15, 20}
>>>NumSet.remove(25)
>>>print(NumSet)
KeyError: 25
```

pop() Method

- pop() method removes an arbitrary element from the set and return it.

Syntax

set.pop()

```
>>>NumSet = {5, 15, 0, 20, 10}
>>>print(NumSet.pop())
0
>>>print(NumSet)
{5, 10, 15, 20}
>>>Courses = {'IC140', 'IC160', 'IC110', 'IC152','HS106',
    'IC152'}
>>>print(Courses.pop())
IC140
>>>print(Courses)
{'IC110', 'IC152', 'IC160', 'HS106'}
```

union() Method

- union() method returns the union of the sets.

Syntax

SetA.union(SetB)

```
>>>NumSet1 = {5, 15, 0, 20, 10}
>>>NumSet2 = {10, 20, 30, 40, 50}
>>>NumSet3 = NumSet1.union(NumSet2)
>>>print(NumSet3)
{0, 5, 40, 10, 15, 50, 20, 30}
>>>Courses = {'IC140', 'IC160', 'IC110', 'IC152','HS106',
    'IC152'}
>>>Set = Courses.union(NumSet1, NumSet2)
>>>print(Set)
{0, 'IC140', 5, 'IC110', 40, 10, 'HS106', 15, 50, 'IC152',
    20, 'IC160', 30}
```

intersection() Method

- intersection() method returns the intersection of sets.

Syntax

SetA.intersection(SetB)

```
>>>NumSet1 = {5, 15, 0, 20, 10}
>>>NumSet2 = {10, 20, 30, 40, 50}
>>>NumSet3 = NumSet1.intersection(NumSet2)
>>>print(NumSet3)
{10, 20}
>>>Courses = {'IC140', 'IC160', 'IC110', 'IC152','HS106',
    'IC152'}
>>>Set = Courses.intersection(NumSet1, NumSet2)
>>>print(Set)
set()
```

difference() Method

- difference() method returns the difference of the sets.

Syntax

SetA.difference(SetB)

```
>>>NumSet1 = {5, 15, 0, 20, 10}
>>>NumSet2 = {10, 20, 30, 40, 50}
>>>NumSet3 = NumSet1.difference(NumSet2)
>>>print(NumSet3)
{0, 5, 15}
>>>Set4 = NumSet2 - NumSet1
>>>print(Set4)
{40, 50, 30}
```

symmetric_difference() Method

- symmetric_difference() method returns the symmetric difference of two sets. .

Syntax

SetA.symmetric_difference(SetB)

```
>>>NumSet1 = {5, 15, 0, 20, 10}
>>>NumSet2 = {10, 20, 30, 40, 50}
>>>NumSet3 = NumSet1.symmetric_difference(NumSet2)
>>>print(NumSet3)
{0, 50, 5, 40, 30, 15}
>>>NumSet4 = NumSet2.symmetric_difference(NumSet1)
>>>print(NumSet4)
{0, 50, 5, 40, 30, 15}
>>>Set = {5, 'a', 10}
>>>print(NumSet1.symmetric_difference(Set))
{0, 20, 'a', 15}
```