PYTHON PROGRAMMING - I

Chap - 3

Python Strings

By-

Prof. A. P. Chaudhari

(M.Sc. Computer Science, SET)

HOD,

Department of Computer Science

S.V.S's Dadasaheb Rawal College,

Dondaicha

Chap – 3 Python Strings

- Introduction to String
- String Literals
- Assign String to a Variable
- Multiline Strings
- Operations on Strings, Index Operator: Working with the Characters of a String, String Methods, Length, The Slice Operator, String Comparison,
- Concepts of Python Lists: Creating, Initializing and Accessing elements in lists, Traversing, Updating and deleting elements from Lists.
- List Operations: Concatenation, List Indexing, Slices
- Built- in List functions and methods
- Aliasing, Cloning Lists

Introduction to String :

Strings are amongst the most popular types in Python. We can create them simply by enclosing characters in quotes. Python treats single quotes the same as double quotes. Creating strings is as simple as assigning a value to a variable. For example −

```
var1 = 'Hello World!'
var2 = "Python Programming"
```

String Literals :

In Python programming language, string literals are enclosed by either single quotation marks, or double quotation marks. For example the two string literals ‘hello’ and “hello” are same. You can use print() function to display a string literal.

e.g.: print(“Python Programming”)

Assigning String to a Variable :

In Python programming we can assign a string value to a variable trough the variable name followed by an equal sign.

The following syntax shows how to assign string to a variable.

cname = “Dadasaheb Rawal College”

We will display the value in string variable by using print() function. E.g.: print(cname)

Multiline String:

You can assign a multiline string to a variable by using three quotes:

e.g.:

```
address = """Plot No. 10,
Ganesh Colony,
Pathardi Phata,
Nashik, (Maharashtra)"""
print (address)
```

O/P:

```
Plot No. 10,
Ganesh Colony,
Pathardi Phata,
Nashik, (Maharashtra)
```

Operations on String :

Substring:

Python does not support a character type; these are treated as strings of length one, thus also considered a substring.

To access substrings, use the square brackets for slicing along with the index or indices to obtain your substring. For example −

e.g.: var1 = 'Hello World!‘

var2 = "Python Programming“

print "var1[0]: ", var1[0]

print "var2[1:5]: ", var2[1:5]

O/P: var1[0]: H

var2[1:5]: ytho

Operations on String :

Updating Strings

You can "update" an existing string by reassigning a variable to another string. The new value can be related to its previous value or to a completely different string altogether. For example −

```
var1 = 'Hello World!'
print "Updated String :- ", var1[:6] + 'Python'
```

When the above code is executed, it produces the following result −

```
Updated String :-  Hello Python
```

String special operator:

Assume string variable **a** holds 'Hello' and variable **b** holds 'Python', then −

Operator	Description	Example
+	Concatenation - Adds values on either side of the operator	a + b will give HelloPython
*	Repetition - Creates new strings, concatenating multiple copies of the same string	a*2 will give -HelloHello
[]	Slice - Gives the character from the given index	a[1] will give e
[:]	Range Slice - Gives the characters from the given range	a[1:4] will give ell
in	Membership - Returns true if a character exists in the given string	H in a will give 1
not in	Membership - Returns true if a character does not exist in the given string	M not in a will give 1

Built-in String Functions:

Python includes the following some built-in methods to manipulate strings −

1) **capitalize() :-** Python String **capitalize()** method returns a copy of the string with only its first character capitalized.

e.g.: str = "this is string example....wow!!!";
print str.capitalize()

O/P: This is string example....wow!!!

2) center() :- Python string method **center()** returns centered in a string of length width. Padding is done using the specified fillchar. Default filler is a space.

Syntax: str.center(width [, fillchar])

e.g.: str = "this is string example....wow!!!"
print str.center(40,'@')

O/P: @@@@this is string example....wow!!!@@@@

Built-in String Functions:

3) count():- Python string method **count()** returns the number of occurrences of substring sub in the range [start, end]. Optional arguments start and end are interpreted as in slice notation.

Syntax: **str.count(sub, start, end)**

Parameters: **sub** − This is the substring to be searched.

start − Search starts from this index. By default search starts from 0 index.

end − Search ends from this index. By default search ends at the last index.

e.g.: str = "this is string example....wow!!!“

sub = "i"

print sub, ‘ counts ‘, str.count(sub, 4, 40)

sub = "wow"

print sub, ‘ counts ‘, str.count(sub)

O/P: i counts 2

wow counts 1

Built-in String Functions:

4) find():- Python string method **find()** determines index, if string *str* occurs in string, or in a substring of string if starting index *beg* and ending index *end* are given.

Syntax: str.find(str, beg, end)

This method return index number if str found otherwise -1.

e.g.: str1 = "this is string example....wow!!!“

str2 = "exam"

print str1.find(str2)

print str1.find(str2, 10)

print str1.find(str2, 40)

O/P: 15

15

-1

Built-in String Functions:

5) index():- Python string method **index()** determines index, if string *str* occurs in string, or in a substring of string if starting index *beg* and ending index *end* are given. This method is same as find(), but raises an exception if sub is not found.

Syntax: str.index(str, beg, end)

This method return index number if str found otherwise exception.

e.g.: str1 = "this is string example....wow!!!“

```
str2 = "exam"
print str1.index(str2)
print str1.index(str2, 10)
print str1.index(str2, 40)
```

```
O/P: 15
     15
Traceback (most recent call last):
  File "C:\Python27\z1.py", line 5, in <module>
  print str1.index(str2, 40)
ValueError: substring not found
```

Built-in String Functions:

6) islower():- Python string method **islower()** checks whether all the case-based characters (letters) of the string are lowercase.

Syntax: str.islower()

e.g.1: str = "THIS is string example....wow!!!"

print str.islower() O/P: False

e.g.2: str = "this is string example....wow!!!"

print str.islower() O/P: True

7) isupper():- Python string method **isupper()** checks whether all the case-based characters (letters) of the string are uppercase.

Syntax: str.isupper()

e.g.1: str = "THIS IS STRING EXAMPLE....WOW!!!"

print str.isupper() O/P: True

e.g.2: str = “THIS is string example....wow!!!"

print str.isupper() O/P: False

Built-in String Functions:

8) join():- Python string method **join()** returns a string in which the string elements of sequence have been joined by *str* separator.

Syntax: str.join(sequence)

e.g.: s = "-";

seq = ("a", "b", "c"); # This is sequence of strings.

print s.join(seq) O/P: a-b-c

9) len():- Python string method **len()** returns the length of the string.

Syntax: len(str)

e.g.: str = "this is string example....wow!!!"

print "Length of the string: ", len(str)

O/P: Length of the string: 32

Built-in String Functions:

10) lower():- Python string method **lower()** returns a copy of the string in which all case-based characters have been lowercased.

Syntax: str.lower()

e.g.: str = "THIS IS STRING EXAMPLE....WOW!!!"

print str.lower()

O/P: this is string example....wow!!!

11) upper():- Python string method **upper()** returns a copy of the string in which all case-based characters have been uppercased.

Syntax: str.upper()

e.g.: str = " this is string example....wow!!!"

print str.upper()

O/P: THIS IS STRING EXAMPLE....WOW!!!

List:

The most basic data structure in Python is the **sequence**. Each element of a sequence is assigned a number - its position or index. The first index is zero, the second index is one, and so on.

There are certain things you can do with all sequence types. These operations include indexing, slicing, adding, multiplying, and checking for membership. In addition, Python has built-in functions for finding the length of a sequence and for finding its largest and smallest elements.

The **list** is a most versatile datatype available in Python which can be written as a list of comma-separated values items between square brackets. Important thing about a list is that items in a list need not be of the same type.

For example −

```
list1 = ['physics', 'chemistry', 1997, 2000];
list2 = [1, 2, 3, 4, 5 ];
list3 = ["a", "b", "c", "d"]
```

Accessing Values in Lists:

To access values in lists, use the square brackets for slicing along with the index or indices to obtain value available at that index.

For example −

```
list1 = ['physics', 'chemistry', 1997, 2000];
list2 = [1, 2, 3, 4, 5, 6, 7 ];
print "list1[0]: ", list1[0]
print "list2[1:5]: ", list2[1:5]
```

When the above code is executed, it produces the following result −

```
list1[0]:  physics
list2[1:5]:  [2, 3, 4, 5]
```

Updating Lists:

You can update single or multiple elements of lists by giving the slice on the left-hand side of the assignment operator, and you can add to elements in a list with the append method.

For example −

```
list = ['physics', 'chemistry', 1997, 2000];
print "Value available at index 2 : “, list[2]
list[2] = 2001;
print "New value available at index 2 : “, list[2]
```

When the above code is executed, it produces the following result −

```
Value available at index 2 : 1997
New value available at index 2 : 2001
```

Delete List Elements:

To remove a list element, you can use either the del statement if you know exactly which elements you are deleting or the remove method if you do not know.

For example −

```
list1 = ['physics', 'chemistry', 1997, 2000];
print list1
del list1[2];
print "After deleting value at index 2 : "
print list1
```

When the above code is executed, it produces following result −

```
['physics', 'chemistry', 1997, 2000]
After deleting value at index 2 :
['physics', 'chemistry', 2000]
```

Basic List Operations:

Lists respond to the + and * operators much like strings; they mean concatenation and repetition here too, except that the result is a new list, not a string.

Python Expression	Results	Description
len[1,2,3]	3	Length
[1, 2, 3] + [4, 5, 6]	[1, 2, 3, 4, 5, 6]	Concatenation
['Hi!'] * 4	['Hi!', 'Hi!', 'Hi!', 'Hi!']	Repetition
3 in [1, 2, 3]	True	Membership
for x in [1, 2, 3]: print x,	1 2 3	Iteration

Built-in List Functions:

1) **cmp():** Python list method **cmp()** compares elements of two lists.

Syntax: **cmp(a, b)**

Parameters: a and b are two elements in which the comparison is being done.

Returns: **-1** if a<b, **0** if a=b **1** if a>b

e.g.1:
```
list1 = [1,2,3]
list2 = [4,5,6]
print cmp(list1, list2)
```
O/P: -1

e.g.2:
```
list1 = [1,2,3]
list2 = [4,5,6]
print cmp(list2, list1)
```
O/P: 1

e.g.3:
```
list1 = ['abc','mno']
list2 = ['abc','mno']
print cmp(list1, list2)
```
O/P: 0

Built-in List Functions:

2) len(): Python list method **len()** returns the number of elements in the *list*.

Syntax: **len(list)**

Parameters: list − This is a list for which number of elements to be counted.

Returns: This method returns the number of elements in the list.

e.g.: list1 = [101, ‘Atharv’, ‘Dhule’, 30000]

list2 = [‘English’, ‘Maths’, ‘Science’]

print “Number of elements in list1:”, len(list1)

print “Number of elements in list2:”, len(list2)

O/P: Number of elements in list1: 4

Number of elements in list2: 3

Built-in List Functions:

3) max(): Python list method **max()** returns the elements from the *list* with maximum value.

Syntax: **max(list)**

Parameters: list − This is a list from which max valued element to be returned.

Returns: This method returns the elements from the list with maximum value.

e.g.: list1 = [400, 100, 700, 300]

list2 = [‘Maths’, ‘English’, ‘Science’]

print “Maximum element in list1:”, max(list1)

print “Maximum element in list2:”, max(list2)

O/P: Maximum element in list1: 700

Maximum element in list2:: Science

Built-in List Functions:

4) min(): Python list method **min()** returns the elements from the *list* with minimum value.

Syntax: **min(list)**

Parameters: list − This is a list from which min valued element to be returned.

Returns: This method returns the elements from the list with minimum value.

e.g.: list1 = [400, 100, 700, 300]

list2 = [‘Maths’, ‘English’, ‘Science’]

print “Minimum element in list1:”, min(list1)

print “Minimum element in list2:”, min(list2)

O/P: Minimum element in list1: 100

Minimum element in list2:: English

Built-in List Functions:

5) list(): Python list method **list()** takes sequence types and converts them to lists. This is used to convert a given tuple into list.

Note − Tuple are very similar to lists with only difference that element values of a tuple can not be changed and tuple elements are put between parentheses instead of square bracket.

Syntax: **list(seq)**

Parameters: **seq** − This is a tuple to be converted into list.

Returns: This method returns the list.

e.g.: tuple1 = (101, 'Atharv', 'Dhule', 30000)

list1 = list(tuple1)

print "List Elements: ", list1

O/P: List Elements: [101, 'Atharv', 'Dhule', 30000]

Built-in List Methods:

1) append(): Python list method **append()** appends a passed *obj* into the existing list.

Syntax: **list.append(obj)**

Parameters: **obj** − This is the object to be appended in the list.

Returns: This method does not return any value but updates existing list.

e.g.: list1 = [101, 'Atharv', 'Dhule', 30000]

list1.append(‘Manager‘)

print “Updated List Elements: ", list1

O/P: Updated List Elements: [101, 'Atharv', 'Dhule', 30000, ‘Manager']

Built-in List Methods:

2) count(): Python list method **count()** returns count of how many times *obj* occurs in list.

Syntax: **list.count(obj)**

Parameters: **obj** − This is the object to be counted in the list.

Returns: This method returns count of how many times obj occurs in list.

e.g.:
```
list1 = [101, 'Atharv', 'Dhule', 101, 30000]
print "Count for 101: ",list1.count(101)
print "Count for Atharv: ",list1.count('Atharv')
```

O/P:
```
Count for 101:  2
Count for Atharv:  1
```

Built-in List Methods:

3) extend(): Python list method **extend()** appends the contents of *seq* to list.

Syntax: **list.extend(seq)**

Parameters: **seq** − This is the list of elements

Returns: This method does not return any value but add the content to existing list.

e.g.:
```
list1 = [101, 'Atharv', 'Dhule', 30000]
list2 = ['English', 'Maths', 'Science']
list1.extend(list2)
print "Extended List: ",list1
```

O/P: Extended List: [101, 'Atharv', 'Dhule', 30000, 'English', 'Maths', 'Science']

Built-in List Methods:

4) index(): Python list method **index()** returns the lowest index in list that *obj* appears.

Syntax: **list.index(obj)**

Parameters: **obj** − This is the object to be find out.

Returns: This method returns index of the found object otherwise raise an exception indicating that value does not find.

e.g.:
```
list1 = [101, 'Atharv', 'Dhule', 30000]
list2 = ['English', 'Maths', 'Science']
print "Index for Dhule: ",list1.index('Dhule')
print "Index for Maths: ",list2.index('Maths')
```

O/P:
```
Index for Dhule:  2
Index for Maths:  1
```

Built-in List Methods:

5) insert(): Python list method **insert()** inserts object *obj* into list at offset *index*.

Syntax: **list.insert(index, obj)**

Parameters: **index** − This is the Index where the object obj need to be inserted.

obj − This is the Object to be inserted into the given list.

Returns: This method does not return any value but it inserts the given element at the given index.

e.g.:
```
list1 = [101, 'Atharv', 'Dhule', 30000]
list1.insert(2,'Manager')
print "Final List: ",list1
```

O/P: Final List: [101, 'Atharv', 'Manager', 'Dhule', 30000]

Built-in List Methods:

6) pop(): Python list method **pop()** removes and returns last object or *obj* from the list.

Syntax: **list.pop(obj)**

Parameters: **obj** − This is an optional parameter, index of the object to be removed from the list.

Returns: This method returns the removed object from the list.

e.g.1:
```
list1 = [101, 'Atharv', 'Dhule', 30000]
print "Poped Element: ",list1.pop()
print "Final List: ",list1
```

O/P:
```
Poped Element:  30000
Final List:  [101, 'Atharv', 'Dhule']
```

e.g.2:
```
list1 = [101, 'Atharv', 'Dhule', 30000]
print "Poped Element: ",list1.pop(2)
print "Final List: ",list1
```

O/P:
```
Poped Element:  Dhule
Final List:  [101, 'Atharv', 30000]
```

Built-in List Methods:

7) remove(): Python list method **remove()** searches for the given element in the list and removes the first matching element.

Syntax: **list.remove(obj)**

Parameters: **obj** − This is the object to be removed from the list.

Returns: This Python list method does not return any value but removes the given object from the list.

e.g.1:
```
list1 = [101, 'Atharv', 'Dhule', 101, 30000]
list1.remove(101)
print "List after removing: ",list1
```

O/P: List after removing: ['Atharv', 'Dhule', 101, 30000]

e.g.2:
```
list1 = [101, 'Atharv', 'Dhule', 101, 30000]
list1.remove('Dhule')
print "List after removing: ",list1
```

O/P: List after removing: [101, 'Atharv', 101, 30000]

Built-in List Methods:

8) reverse(): Python list method **reverse()** reverses objects of list in place.

Syntax: **list.reverse()**

Returns: This method does not return any value but reverse the given object from the list.

e.g.1:
```
list1 = [101, 'Atharv', 'Dhule', 30000]
list1.reverse()
print "Reverse List: ",list1
```

O/P: Reverse List: [30000, 'Dhule', 'Atharv', 101]

Built-in List Methods:

9) sort(): Python list method **sort()** sorts objects of list.

Syntax: **list.sort()**

Returns: This method does not return any value but it changes from the original list.

e.g.1:
```
list1 = [101, 'Atharv', 'Dhule', 30000]
list1.sort()
print "Sorted List: ",list1
```

O/P: Sorted List: [101, 30000, 'Atharv', 'Dhule']