PYTHON PROGRAMMING

UNIT-III: Strings: Strings-Compound data types- len function- String Slices-Strings are Immutable-String Traversal-Escape Characters-String formatting operators and functions. Lists: Values and accessing elements-lists are mutable-Traversing and deleting elements –Built-in operators and methods.

TEXT BOOK

1. E.Balagurusamy, "Introduction to Computing and Problem Solving Using Python", McGraw Hill Education Private Limited, 1st Edition, New Delhi.

STRINGS AND LISTS

STRINGS

- Strings are created by enclosing various characters within quotes. Strings are one of the most popular data types
- Python does not distinguish between single quotes and double quotes.

Ex:

```
V1="hello python"
V2='welcome'
V3=""""this is triple quoted string""""
Print v1
Print v2
Print v3
```

Output

```
Hello python
Welcome
This is triple quoted string
```

- Strings are of literal or scalar type
- Python interpreter treats string as a single value
- Strings are immutable. If we want to change an element of a string, we have to create a new string.
- Triple quoted strings can span to multiple lines

```
>>>v1=“””welcome
    to
    python
    programming”’’
>>>print v1
```

output

```
    welcome
     to
     python
     programming
```

1.Compound data type

- Strings are made up of smaller pieces of characters. The data types that are made up of smaller pieces are known as compound data types.
- To access a part of the string ([]) must be used.

```
>>>string=“hello”
>>>letter=string[4]
>>>print letter     #5th letter of string
```

Output

```
O
```

Getting first letter of the string

```
>>string=“hello”
>>>letter=string[0]
>>>print letter
```

Output

```
h
```

2.Len function

- len is a built-in function in python. When used with a string, len returns the length or number of characters in the string.

Ex:

```
>>>v1="hello python!"
>>>len(v1)
```

Output

```
13
```

Access the last letter of out string

```
>>>v1="hello python!"
>>>l=len(v1)        #hello python!
   last=v1[-1]
   print last
```

output

```
    !
```

Ex:

```
>>>var="hello world"
      last=var[-1]      #negative indices for accessing the string from last
      second_last=var[-2]       #second last element of the string
       print last
       print second_last
```

Output

```
d
l
```

3. String Slices

- A piece or subset of a string is known as slice
- Slice operator is applied to a string with the use of square braces([])
- Operator [n:m] will give a substring consists of letters between n and m indices[i.e., nth index to (m-1)th index.
- Operator [n:m:s] $\Longrightarrow$ nth index to (m-1)th index, where s is called the step value ie., n,n+s, n+2s,n+3s……..

Ex:

```
>>>var='hello python'
>>>print var[0:4]
>>>print var[6:12]
```

Output

```
hell
python
```

```
>>>al='abcdefghij'
>>>print al[1:8:3]
>>>print al[1:8:2]
```

Output

```
beh
bdfh
```

```
>>>var='banana'
>>>var[:4]
>>>var[4:0]
>>>var[ : ]
```

```
>>>var='banana'
>>>var[4:3]
>>>var[ : :-1]
```

Output

```
bana
na
'banana'
''                    #the second index is smaller than first index, then the
                      output will be empty and represented in single quotes.
'ananab'              #step is -1, and no value for n and m, then  it will print the
                      string in reverse order.
```

An empty string has a length 0. though it does not contain any character, it is still a string.

4. Strings are immutable.

- We cannot change any element of a string.

```
>>>v1=‘hello’
v1[0]=‘p’
```

Output: type error ‘str’ object does not support item assignment.

Solution: generate a new string rather than change the old string

```
>>>var=‘hello python’
>>>new_var=‘p’+var[1:]
>>>print new_var
```

Output

```
Pello python
```

5. String Traversal

Traversal is a process in which we access all the elements of the string one by one using some conditional statements such as for loop, while loop etc.,

Ex:

```
var1='hello python‘
while i<len(var1):
     letter=var1[i]
      print letter
      i=i+1
```

Output

```
h
e
l
l
o
p
y
t
h
o
n
```

Ex:

```
var='hello python'
for char in var:
      print char
```

Output

```
h
e
l
l
o
p
y
t
h
o
n
```

Ex: you have been given a string ‘I live in cochin. I love pets’. Divide this string in such a way that the two sentences in it are separated and stored in different variables. Print them.

```
var=‘I live in cochin.  I love pets’
var1=var[:17]
var2=var[18:30]
print var1
print var2
```

Output

```
I live in cochin
I love pets
```

Ex: searching within strings

Ex:

```
def find(string,char)
index=0
while index<len(string):
    if string[index]==char:
        return index
index=index+1
return -1
```

A function find takes a string and a character as input. A while loop traverses the string until the end and compares every element of the string with the character passed by the user. If it matches any element of the string then the index of that element is returned by the function. otherwise it returns -1.

Note: the return statement in a while loop works in the same way as the break statement.

6. ESCAPE CHARACTERS

- Backslash(\) character is used to escape characters
- If we want quotation marks in the output, then we will make use of escaping characters.

Ex:

```
>>>print “I am 5\” tall”
```

Output

```
I am 5” tall
```

.

Following table is a list of escape or non-printable characters that can be represented with backslash notation.

An escape character gets interpreted in a single quoted as well as double quoted strings

An escape character gets interpreted; in a single quoted as well as double quoted strings.

Escape Sequence	Meaning
\newline	ignored
\\	Backslash(\)
\’	Single quote(‘)
\”	Double quote(“)
\a	ASCII bell
\f	ASCII backspace
\n	ASCII linefeed
\r	ASCII carriage return
\t	ASCII horizontal tab
\v	ASCII vertical tab
\ooo	ASCII character using octal value ooo
\xhhh	ASCII character with Hex value

Examples using escape sequences

```
txt = "We are the so-called "Vikings" from the north.“
#You will get an error if you use double quotes inside a string that are
surrounded by double quotes:

txt = "We are the so-called \"Vikings\" from the north."
print(txt)
```

output

```
We are the so-called "Vikings" from the north.
```

```
txt = 'It\'s alright.'
print(txt)
```

output

```
It's alright.
```

```
txt = "This will insert one \\ (backslash)."
print(txt)
```

output

```
This will insert one \ (backslash).
```

```
txt = "Hello\nWorld!"
print(txt)
```

output

```
Hello
World!
```

```
txt = "Hello\rWorld!"
print(txt)
```

Output

```
Hello
World!
```

```
txt = "Hello\tWorld!"
print(txt)
```

Output

```
Hello   World!
```

```
#This example erases one character (backspace):
txt = "Hello \bWorld!"
print(txt)
```

Output

```
HelloWorld!
```

```
#A backslash followed by three integers will result in a octal value:
txt = "\110\145\154\154\157"
print(txt)
```

Output

```
Hello
```

7. String formatting operator

Format Symbol	Conversion
%c	Character
%s	String conversion
%i	Signed Decimal integer
%d	Signed Decimal integer
%u	Unsigned decimal integer
%o	Octal integer
%x	Hexadecimal integer(lowercase letters)
%X	Hexadecimal integer(uppercase letters)
%e	Exponential notation(with lowercase 'e')
%E	Exponential notation(with uppercase 'E')
%f	Floating point real number
%g	The shorter of %f and %e
%G	The shorter of %f and %E

Ex:

```
>>>print(“the first letter of %s is %c” %(‘python’ , ’p’))
>>>print(“the sum=%d” %(-15))
>>>print(“the sum=%i” %(-15))
>>>print(“the sum=%u” %(15))
>>>print(“%o is the octal equivalent of %d”, %(9,9))
>>>print(“%x is the hexadecimal equivalent of %d” %(12,12))
>>>print(“%X is the hexadecimal equivalent of %f” %(8.98354,8.98354))
>>>print(“%E is the exponential equivalent of %f” %(8.98354,8.98354))
```

Output

```
The first letter of python is p
The sum=-15
The sum=-15
The sum=15
11 is the octal equivalent of 9
c is the hexadecimal equivalent of 12
C is the hexadecimal equivalent of 12
8.983540e+00 is the exponential equivalent of 8.983540
8.983540E+00 is the exponential equivalent of 8.983540
```

8.String formatting functions

Built-in functions for strings

1. capitalize()

    makes the first letter of the string capital

ex:

a='banana‘

x=a.capitalize()

print(x)

Output:Banana

2. center(width,fillchar())

    Returns a space-padded string with the original string centred to a total width columns.

Ex:

a=‘banana’

x=a.center(20)

print(x)

Output:

--7 blanksplace--   banana ---7 blank space----

Ex:

```
a='banana‘
x=a.center(20,"0")
print(x)
```

output

```
0000000banana0000000
```

Ex:

```
a='hello, and welcome to my world‘
x=a.capitalize()
print(x)
```

output

```
Hello, and welcome to my world
```

3.count(str,beg=0,end=len(string()))

counts the number of times string occurs in the string or in a substring provided that starting index is beg and ending index is end.

Ex:

```
 txt="i love apples”
x=txt.count("apple")
print(x)
```

Output

```
1
```

optional

optional

Syntax: $\text{string.count(value,}\overset{\text{optional}}{\text{start}}\text{,}\overset{\text{optional}}{\text{end}}\text{)}$

```
txt="i love apples, apple are my favourite fruit”
x=txt.count("apple“,10,24)
print(x)
```

Output

```
1
```

LISTS

I. values and accessing elements

- List is a collection of items or elements
- Sequence of data in a list is ordered
- Elements in the list can be accessed by their positions [indices]
- Lists are defined before they are used
- Creation of list using []

Ex:

List1=[2,-1,0,-2,8]

List2=[‘lilly’, ‘jasmine’, ‘rose’]

List3=[‘python’, 5.5,8]

List4=[‘python’, 5.6, [20,40]] $\longrightarrow$ list is contained in another list.ie nested list

Copying the list

duplicate or copy of an existing list

```
org1=[1,2,3,4]
cp1=org1
```

Modifying org1

```
>>>org1.append(10)
>>>print org1
[1,2,3,4,10]
>>>print cp1
[1,2,3,4,10]
```

Note: the modification made in org1 will also take place in cp1.

Two methods to make copy of a list

1. Using [:] operator
2. Using built-in copy function

1. Using [:] operator

```
>>>org1=[1,2,3,4]
cp1=org1[:]
Print cp1
```

Output

```
[1,2,3,4]
```

Making changes in the original list

```
>>>org1.append(10)
>>>print org1
[1,2,3,4,10]        #original list is changed
>>>print cp1
[1,2,3,4]           #copied list is unchanged
```

2. Using built-in functions

- To copy the list use import function

Ex:

```
>>>from copy import copy #import library copy
>>>org1=[1,2,3,4]
>>>cp1=copy(org1)
>>>print cp1
```

output

```
[1,2,3,4]
```

An empty list also can be created using enclosing brackets with no elements, inside them

```
>>>a=[]
```

Printing a list assigning it to a variable:

```
>>>list=[10,20,30,'hello']
>>>print list
[10,20,30,'hello']
```

II list are mutable

- The value of any element inside the list can be changed at any point of time
- The elements accessible by their index value.
- Index value starts with 0 and ends with n-1

Ex:

```
 list=[10,20,30,40]
Print list[1]
```

Output

```
20
```

Changing the value in the list

```
List[3]=50
Print list
```

Output

```
[10,20,30,50]
```

Indices in a list work in the same way as in string:

- Any integer expression can be used as an index number
- If any element that does not exists in the list is accessed there will be index error
- If the indices are given in negative, then counting happens from the end of the list(backward)

III .Traversing a list

- Accessing all the elements of the list
- Traversing is performed using any conditional statement in python, but prefer for loop.

Ex:

```
l1=['a','b','c','d']
for x in l1:
    print x
```

Output

```
a
b
c
d
```

Ex:

```
list=[10,20,30,40]
for i in range(len(list)):
        list[i]=list[i]+4
print list
```

Output

```
[14,24,34,44]
```

Note:range and len are functions

Range $\longrightarrow$ returns the indices

Len $\longrightarrow$ return the length

IV DELETING ELEMENTS FROM THE LIST

a. Pop operator

Ex:

```
List=[10,20,30,40]
a=list.pop(2)
Print list
Print a
```

output

```
[10,20,40]
30
```

The pop operator deletes the element on the provided index and stores that element in a variable for further use.

b. Del Operator

The del operator deletes the value on the provided index, but it does not store the value for further use.

Ex:

List=[‘W’,’X’,’Y’,’Z’]

del list(1)

Print list

Output

[‘w’,’y’,’z’]

c.Remove operator

This operator is used to remove or delete from the list.

Ex:

List=[10,20,30,40]

List.remove(10)

Print.list

Output

[20,30,40]

Delete more than one value from a list, del operator with slicing is used.

Ex:

List=[1,2,3,4,5,6,7,8]

del list[1:3]

Print list

Output

[1,4,5,6,7,8]

V. Built-in list operators

a. concatenation: this operator concatenates two strings (+) operator.

Ex:

List1=[10,20,30,40]

List2=[50,60,70]

List3=list1+list2

Print list3

Output:[10,20,30,40,50,60,70]

b. Repetition: repeats the list for a given number of times

Ex:

```
List1=[1,2,3]
List1*4
```

Output

```
[1,2,3,1,2,3,1,2,3,1,2,3,1,2,3]
[2]*6
[2,2,2,2,2,2]
```

c. In operator

The In operator tells the user whether the given string exists in the list or not. It gives a Boolean output true or false.

Ex:

```
List=['hello','python','program']
'hello' in list
True #output
```

‘world’ in list

False #output

Ex:

List=[10,20,30,40]

10 in list

True #output

Built-in list methods

1. len(list) – it determine how many items a list has

Ex:

List=[“apple”,”banana”,”cherry”]

Print(len(list))

Output

3

2. append() – To add item to the end of the list

Ex:

```
List=[1,2,3,4]
List.append(0)
Print list
```

Output

```
[1,2,3,4,0]
```

3. Insert() method- to add an item at the specified index

Ex:

```
List=[“apple”,”banana”,”cherry”]
List.insert[1, ”orange”]
```

Output

```
[‘apple’,’orange’,’banana’,’cherry’]
```

4.Remove() method removes the specified item.

Ex:

List=[“apple”,”banana”,”cherry”]

List.remove(“banana”)

Print(list)

Output

[‘apple’, ’cherry’]

5. pop() method removes the specified index,(or the last item if index is not specified)

Ex:

List=[“apple”, ”banana”, “cherry”]

List.pop()

Print(list)

Output

[‘apple’, ’banana’]

6. del() method- removes the specified index.

Ex:

List=[“apple”, “banana”, “cherry”]

del list[0]

Print(list)

Output

[‘banana’, ‘cherry’]

7.clear() method- empties the list.

Ex:

List=[“apple”,”banana”,”cherry”]

List.clear()

Print(list)

Output

[]

8.list.count(item)-item can be string, number, list, tuple etc.,

Ex:

```
list=["apple","banana","cherry"]
p= list.count('apple')
print(p)
```

Output

```
1
```

Ex:

```
P=[1,4,2,9,7,8,9,3,1]
X=p.count(9)
Print(x)
```

Output

```
2
```

9.extend()

List.extend(iterable)- it adds the specified list elements to the end of the current list.

Ex:

```
Fruits=['apple', 'banana', 'cherry']
Points=[1,4,5,8]
Fruits.extend(points)
Print(fruits)
```

Output

```
['apple', 'banana', 'cherry', 1,4,5,8]
```

10.index() method –position of the element is found

List.index(element)- element can be string, number,list

Ex:

list1=[4,55,64,32,16,32]

X=list1.index(32)

Print(x)

Output

3

11. reverse() method- method reverse the sorting order of the elements.

Ex:

Fruits=[‘apple’,’banana’,’cherry’]

Fruits.reverse()

Print(fruits)

Output:

[‘cherry’, ’banana’, ’apple’]

12. sort() method- The method sorts the list in ascending by default.

Ex:

```
Cars=[‘ford’, ‘bmw’, ‘volvo’]
Cars.sort()
Print(cars)
```

Output

```
[‘bmw’ ‘ford’, ‘volvo’]
```

13. max(list)- it returns the item that has the maximum value in the list.

14. min(list)- it returns the item that has the minimum value in the list.

Ex:

```
Cars=[‘ford’, ‘bmw’, ‘volvo’]
Max(cars)
Min(cars)
```

Output

```
Volvo
bmw
```