CLASS-XI

String Manipulation

STUDY NOTE

Period-01

Learning Outcomes

- An in-depth understanding string.
- Introduce the fundamentals of Mutable & Immutable Types.
- Working with different types of Operators.
- Concepts of sequence
- Concept of string functions & methods
- Debugging Concepts

Introduction:-

You all have basic knowledge about Python strings. You know that Python strings are characters enclosed in quotes of any type – single quotation marks, double quotation marks and triple quotation marks. You have also learnt things like – an empty string is a string that has 0 characters (i.e., it is just a pir of quotation marks) and that Python strings are immutable. You have used strings in earlier chapters to store text type of data.

You know by now that strings are sequence of characters, where each character has a unique position –id/index. The indexes of a string begin from 0 to (length – 1) in forward direction and -1,-2,-3, ..., -length in backward direction.

In this chapter, you are going to learn about many more string manipulation techniques offered by Python like operators, methods etc.

TRAVERSING A STRING

You know that individual characters of a string are accessible through in unique index of each character. Using the indexes, you can traverse a string character by character. Traversing refers to iterating through the elements of a string, one character at a time. You are already traversed through strings, though unknowingly, when we talked about sequences along with for loops. To traverse through a string, you can write a loop like :

Fsdfdasffdsaf

The information that you have learnt till now is sufficient to create wonderful programs to manipulate strings. Consider the following programs that use the Python string indexing to display strings in multiple ways.

➢ Program to read a string and display it in reverse order – display one character per line. Do not create a reverse string, just display in reverse order.

```
string1 = input(“Enter a string:”)
print (“The”, string1, “in reverse order is”☺
length = len(string1)
for a in range (-1, (-length-1),-1) :
        print (string1[a])
```

Sample run of above program is :

```
Enter a string : python
The python in reverse order is :
n
o
h
t
y
p
```

➢ Program to read a string and display it in the form :

```
        First character        last character
        Second character       second last character
                :                       :
```

For example, string “try” should print as :

```
        t       y
        r       r
        y       t
```

```
        string1 = input (“Enter a string : “)
        length = len(string)
        i = 0
        for a in range(-1, (-length-1), -1 :
                print(string1[1], "\t", string1[a])
                i +=1
```

Sample run of above program is :

```
Enter a string : Python
 p       n
 y       o
 t       h
 h       t
 o       y
 n       p
```

STRING OPERATIONS

In this section, you'll be learning to work with various operators that can be used to manipulate strings in multiple ways. We'll be talking about basic operators + and*, membership operators in and not in and comparisons operators (all relational operators) for strings.

- **Basic Operators**

The two basic operators of strings are : + and *. You have use these operators as arithmetic operators before for addition and multiplication respectively. But when used with strings. + operator performs concatenation rather than addition and * operator performs replication rather than multiplication. Let us see, how.

Also, before we proceed, recall that strings are immutable i.e., un-modifiable. Thus every time you perform something on a string that changes it, Python will internally create a new string ratter than modifying the old string in place.

String Concatenation Operator +

The + operator creates a new string by joining the two operand strings, e.g.,

"tea" + "pot"

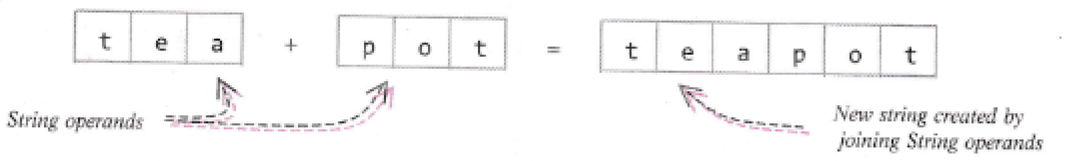
Will result into

DDD

Consider some more examples :

Expression	**Will result into**
‘l’ +’l’	‘11’
“a” + “0”	‘a0’
‘123’ + ‘abc’	‘123abc’

Let us see how concatenation takes place internally. Python creates a new string the memory by storing the individual characters of first string operand followed by the individual characters of second string operand. (see below).



Original strings are not modified as strings are immutable ; new strings can be created but existing strings cannot be modified.

Caution !

Another important thing that you need to know about + operator is that this operator can work with numbers and strings separately for addition and concatenation respectively, but in the same expression, you cannot combine numbers and strings as operands with a + operator.

For example,

2 + 3 = 5	# addition – VALID
‘2’ + ‘3’ = ‘23’	# concatenation – VALID

But the expression

‘2’ + 3

Is invalid. It will produce an error like :

```
>>> ‘2’ + 3
 Traceback (most recent call last) :
    File “<pyshell#2>”, line 1, in <module>
      ‘2’ + 3
```

Thus we can summarize + operator as follows :

Table : Working of Python + Operator :

Operands' data type	Operation performed by +	Example
numbers	addition	9 + 9 = 18
string	concatenation	"9"+"9" = "99"

Period-02

String Replication Operator *

The *operator when used with numbers (i.e., when both operands are numbers), it performs multiplication and returns the product of the two number operands.

To use a * operator with strings, you need two types of operands – a string and a number, i.e., as number * string or string * number.

Where string operand tells the string to be replicated and number operand tells the number of times, it is to be repeated; Python will create a new string that is a number of repetitions of the string operand.

For example,

3 * “go!”

Will return

‘go!go!go!’

Consider some more examples :

Expression	will result into
“abc” * 2	“abcabc”
5*”@”	“@@@@@”
“:-“ * 4	“:-:-:-:-“
“1” * 2	“11”

Caution!

Another important thing that you need to know about* operator is that this operator can work with numbers as both operands for multiplication and with a string and a number for replication respectively, but in the same expression. You cannot have string as both the operands with a* operator.

For example.

```
2 * 3 = 6            # multiplication – VALID
“2”*3 = “222”        #replication – VALID
```

But the expression

```
“2” * “3”
```

Is invalid. It will produce an error like :

```
>>>”2”*”3”
Traceback(most recent call last) :
  File “<pyshell#0>”, line 1, in <module>
    “2”*”3”
TypeError : can’t multiply sequence by non-int of type ‘str’
```

Thus we an summarize + operator as follows :

Table 2.2 : Working of Python * operator