Python Programming

RGM College of Engineering & Technology (Autonomous)

Department of Computer Science & Engineering

Academic Year : 2020-2021

UNIT – III:

Strings: Introduction to strings, Defining and Accessing strings, Operations on string - String slicing, Mathematical Operators for String, Membership operators on string, Removing spaces from the string, Finding Substrings, Counting substring in the given String, Replacing a string with another string, Splitting of Strings, Joining of Strings, Changing case of a String, Checking starting and ending part of the string, checking type of characters present in a string. Illustrative examples on all the above topics.

Files: Opening files, Text files and lines, Reading files, Searching through a file, Using try, except and open, Writing files, debugging.

STRINGS IN PYTHON – I -1

Guido Van Rossum

Learning Mantra

If you really strong in the basics, then remaining things will become so easy.

Agenda

1. Introduction
2. How to access characters of a String
3. Behaviour of slice operator

INTRODUCTION

The most commonly used object in any project and in any programming language is String only. Hence we should aware complete information about String data type.

What is String?

- Any sequence of characters within either single quotes or double quotes is considered as a String.

Syntax:

```
s='karthi'

s="karthi“
```

Note:

- In most of other languages like C, C++,Java, a single character with in single quotes is treated as char data type value. But in Python we are not having char data type. Hence it is treated as String only.

Eg:

```
ch = 'a'
print(type(ch))          ➔<class 'str'>
```

How to define multi-line String literals?

1. We can define multi-line String literals by using triple single or double quotes.

Eg:

```
s = '''karthi
sahasra
sri'''
print(s)                 # Multi line strings
```

Output:

```
karthi
sahasra
sri
```

Eg: Multi line strings

```
s = """karthi
sahasra
sri"""
print(s)
```

Output:

```
karthi
sahasra
sri
```

2. We can also use triple quotes to use single quotes or double quotes as symbol inside String literal.

Eg:

s='This is ' single quote symbol‘ ➔**SyntaxError:** invalid syntax

s='This is \' single quote symbol'

print(s) ➔This is ' single quote symbol

s="This is ' single quote symbol"

print(s) ➔This is ' single quote symbol

s='This is " double quotes symbol'

print(s) ➔This is " double quotes symbol

s='The "Python Notes" by ‘ABC' is very helpful‘ ➔**SyntaxError**: invalid syntax

s="The "Python Notes" by ‘ABC' is very helpful“ ➔**SyntaxError**: invalid syntax

s='The \"Python Notes\" by \‘ABC\' is very helpful'

print(s) ➔The "Python Notes" by ‘ABC' is very helpful

s='"The "Python Notes" by ‘ABC' is very helpful"'

print(s) ➔The "Python Notes" by ‘ABC' is very helpful

How to access characters of a String?

- We can access characters of a string by using the following ways.
    1. By using index
    2. By using slice operator

1. By using index:

- Python supports both +ve and -ve index.
- +ve index means left to right(Forward direction)
- -ve index means right to left(Backward direction)

Eg:

```
s = 'Karthi'
print(s[0])      ➔K
print(s[5])      ➔i
print(s[-1])     ➔i
print(s[19])     ➔ IndexError: string index out of range
```

Eg: Q 1. Write a program to accept some string from the keyboard and display its characters by index wise(both positive and negative index)

```
s=input("Enter Some String: ")
i=0
for x in s:
    print("The character present at positive index {} and at negative index {} is {}".format(i,i-len(s),x)
    i=i+1
```

Enter Some String: karthikeya

The character present at positive index 0 and at negative index -10 is k

The character present at positive index 1 and at negative index -9 is a

The character present at positive index 2 and at negative index -8 is r

The character present at positive index 3 and at negative index -7 is t

The character present at positive index 4 and at negative index -6 is h

The character present at positive index 5 and at negative index -5 is i

The character present at positive index 6 and at negative index -4 is k

The character present at positive index 7 and at negative index -3 is e

The character present at positive index 8 and at negative index -2 is y

The character present at positive index 9 and at negative index -1 is a

2. Accessing characters by using slice operator:

- string slice means a part of the string (i.e., Sub string).

Syntax:

string_Name [beginindex:endindex:step]

Here,

- beginindex: From where we have to consider slice(substring)
- endindex: We have to terminate the slice(substring) at endindex-1
- step: incremented / decremented value

Note :

- Slicing operator returns the sub string form **beginindex** to **endindex – 1**
- If we are not specifying begin index then it will consider from beginning of the string.
- If we are not specifying end index then it will consider up to end of the string.
- The default value for step is 1.

Eg 1:

```
s = 'abcdefghijk'
print(s[2:7])      ➔cdefg
print(s[:7])       ➔abcdefg
print(s[2:])       ➔cdefghijk
print(s[:])        ➔abcdefghijk
print(s[2:7:1])    ➔cdefg
print(s[2:7:2])    ➔ceg
print(s[2:7:3])    ➔cf
print(s[::1])      ➔abcdefghijk
print(s[::2])      ➔acegik
print(s[::3])      ➔adgj
```

Eg 2:

s="Learning Python is very very easy!!!"

s[1:7:1] ➔earnin

s[1:7] ➔earnin

s[1:7:2] ➔eri

s[:7] ➔Learnin

s[7:] ➔ g Python is very very easy!!!

s[::] ➔Learning Python is very very easy!!!

s[:] ➔Learning Python is very very easy!!!

s[::-1] ➔!!!ysae yrev yrev si nohtyP gninraeL **#Reverse of the string**

Behaviour of slice operator:

s[begin:end:step]

- Here, step value can be either Positive or Negative.
- If Positive then it should be forward direction(left to right) and we have to consider begin to end-1
- If Negative then it should be backward direction(right to left) and we have to consider begin to end+1

Note:

- In the backward direction if end value is -1 then result is always empty.
- In the forward direction if end value is 0 then result is always empty.

In forward direction:

- default value for begin: 0
- default value for end: length of string
- default value for step: +1

In backward direction:

- default value for begin: -1
- default value for end: -(length of string + 1)

Note:

- Either forward (or) backward direction, we can take both Positive and Negative values for begin and end index.

Slice Operator (Review):

Syntax:

s[begin:end:step]

Here,

i. begin ======> starting Index

ii. end =======> Ending Index

iii. step ======> Incrementing value

- step value can be either positive or negative.
- if step is positive ➔ We required to move forward direction and we have to move from begin to end-1.
- if step is negative ➔We required to move backward direction and we have to move from begin to end+1.

Assume that, if We take,

s[begin:end:-1]

Rule:

- If the step is negative, end should not be -1 (or) end+1 is 0 then result is always an empty string.

s = '0123456789'

print(s[0:5:1]) ➔ 01234

print(s[4:-1:-1]) ➔ end + 1 = 0 so we will get empty string

print(s[-7:-2:-1]) ➔ from -7 to -1 we can't move so empty string will return

print(s[-2:-7:-1]) ➔87654

print(s[-2:-3:-2]) ➔8

print(s[0:0:-1]) ➔from 0 to 0+1 =1 (i.e.,end +1) in backward direction we can't move

print(s[0:-2:-1]) ➔from 0 to -2+1 =-1 (i.e.,end +1) in backward direction we can't move

print(s[-1:-2:-1]) ➔9

print(s[7:-1:-1]) ➔Empty string

print(s[1:7:-1]) ➔Empty string

Note:

If we consider the following statement,

s[x:-1:-Z] =====> Irrespective of **x** and **z** values we are going to return an empty string.

Any question?

If you try to practice programs yourself, then you will learn many things automatically

Spend few minutes and then enjoy the study

Thank You