Strings in Python

What is the 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. Python strings are immutable.

for eg

```
var1 = "hello world"

var2 = 'hello'
```

Accessing Values in Strings

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, string element has both forward index as well as backward index.
str = "Python"

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

```
>>>str[0]  ------ P
>>>str[-1]----- n
```

Since **length** of string variable can be determined using function **len(<string>)**, we can say that

- first character of the string is at index 0 or –length.
- second character of the string is at index 1 or –(length-1)
- :
- second last character of string is at index (length-2) or -2
- last character of string is at index (length-1) or -1

Also another thing that you must know is that you cannot change the individual letter of a string by assignment because strings are immutable and hence **item assignment is not supported. i.e.**

```
name = "Hello"
name[0]='p'
name[0]='p'
TypeError: 'str' object does not support item assignment
```

However, you can assign to a string another string or an expression that returns a string using assignment e.g. following statement is valid.

```
name = "Hello"
name = "new"   # valid statement; String can be assigned
expression that give strings.
```

String Operator

To perform basic operations on String Python provides operators:

i) + Concatenation - Adds values on either side of the operator
```
a = 'Hello'
b = 'Python'
print(a+b)
```
------ 'HelloPython'

ii) * Repetition - Creates new strings, concatenating multiple copies of the same string
`print(a*3)` ------- 'HelloHelloHello'

iii) [] Slice - Gives the character from the given index
`a[0]` ---- H

iv) [:] Range Slice - Gives the characters from the given range
`a[1:4]` ---- ell
it extract from given range a[lowRange: upperRange] while upper range excluded.

v) in : Membership - Returns true if a character exists in the given string
`'H' in a` will give True

vi) not in : Membership - Returns true if a character does not exist in the given string
`'M' not in a` will give True

vii) r/R : Raw String - Suppresses actual meaning of Escape characters. The syntax for raw strings is exactly the same as for normal strings with the exception of the raw string operator, the letter "r," which precedes the quotation marks. The "r" can be lowercase (r) or uppercase (R) and must be placed immediately preceding the first quote mark.
```
>>> print(r'\n') ----------- \n
```

```
    >>>  print(R'\n') ----------- \n
```

String Formatting Operator

One of Python's coolest features is the string format operator %. This operator is unique to strings and makes up for the pack of having functions from C's printf() family. Following is a simple example:

```
print('My Name is %s and weight is %d' % ('Vikas', 60))
```

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

```
My Name is Vikas and weight is 60
```

Format Symbol	Conversion
%c	Character
%s	String conversion via str() prior to formatting
%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)
%f	Floating point real number