String

It is one of the most used data types in Python. Thus, string is prioritized by python as well.
For example:

```
input('____')
```

Here, the input keyword takes string, by default.

String is known as the array of characters. In python specifically, string is anything enclosed by single quotation, double quotation or triple quotation (triple single quotation or triple double quotation).

Example:

```
a='apple'
print(type(a))
```

Output:

```
apple
<class 'str'>
```

There is no concept of char(character) in Python, be it a single character or multiple character, all are strings.
The string with triple double quotation is also known as docstring, which is a special kind of string which lets us print multi-line strings.

For example:

```
a= """ Learning
Python
is
fun """
print(a)
```

Output:

```
Learning
Python
is
fun
```

Note: Triple quotation is generally only used to represent docstring, or it's a good practice.

Escape sequence:
Incase of strings, when we need to represent special characters we use escape sequences, denoted by backslash '\'.
Example:

```
a= 'Python\'s classes'
print(a)
a= "Python\nclass"
print(a)
```

Output:

```
Python's classes
Python
class
```

Here, such string literals are called escape sequences.
In python, we have multiple escape sequences such as:

\n : newline
\t : tab
\r : carriage return
\b : backspace
\v : vertical tab
\' : single quote
\" : double quote
\\ : slash

Example:

```
a = 'Python\\Lava class'
print(a)
```

Output:

```
Python\Lava class
```

Accessing the characters of a string:

- **Indexing**:

Indexing always starts from 0 where each index represents a character. In python, there is the concept of both positive and negative indexing.
Example 1:

```
s= "python is fun"
print(s[4])
```

```
print(s[-9])
```

Output:

```
o
o
```

Example 2:

```
s="python is fun"
i=0
print("Index of given string in both positive and negative index")
for x in s:
print(f"The character {x} is present at positive {i} index and negative
{i-len (s)} index")
i =i+1
```

Output:

```
Index of given string in both positive and negative index
The character p is present at positive 0 index and negative -13 index
The character y is present at positive 1 index and negative -12 index
The character t is present at positive 2 index and negative -11 index
The character h is present at positive 3 index and negative -10 index
The character o is present at positive 4 index and negative -9 index
The character n is present at positive 5 index and negative -8 index
The character   is present at positive 6 index and negative -7 index
The character i is present at positive 7 index and negative -6 index
The character s is present at positive 8 index and negative -5 index
The character   is present at positive 9 index and negative -4 index
The character f is present at positive 10 index and negative -3 index
The character u is present at positive 11 index and negative -2 index
The character n is present at positive 12 index and negative -1 index
```

Slice Operator:

Slice operator in Python extracts a portion (or "slice") of a string using a specific range of indices. This is useful for accessing substrings within a larger string.

Syntax:

string[start : end : step]

For example:

```
s= 'Hello world this is python'
print(s[0:5:1])
print(s[0:5:2])
```

```
print(s[::])  //none values are actually compulsory
```

Output:

```
Hello
Hlo
Hello world this is python
```

Using negative index:

```
s= 'Hello world this is python'
print(s[::-1])
```

Output:

```
nohtyp si siht dlrow olleH
```

Above is similar to:

```
s= 'Hello world this is python'
b=reversed(s)
For x in b:
print(x,end= ' ')
```

Output:

```
nohtyp si siht dlrow olleH
```

Behavior of slice operator:

For a string ‘s’, the syntax is:

s [begin : end-1 : step]

Here,

begin : can be -ve or +ve and is optional

end: can be -ve or +ve and is optional

[In forward direction, if this value is 0 then we get an empty string.

In backward direction, if this value is -1 then we get an empty string.]

step: can be -ve (backward direction/ right to left) or +ve (forward direction / left to right) and is optional.

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)

Default Value For Step = -1

For example:

```
s= 'abcdefghijkl'
print(s[1:6:1])
print(s[::1])
print(s[::-1])
print(s[3:7:-1])
print(s[7:4:-1])
print(s[-4:1:-1])
```

Output:

```
bcdef
lkjihgfedcba
   (gives empty string )
hgf
ihgfedc
```

Operations in string:

Addition:

The addition operator is used in string as a means of concatenation. But both the concatenated data types should be string, not number.

Multiplication:

Multiplication operator, on the other hand, needs to have a string and an integer number(not float or float)

Example:

```
str1='Ram '
str2='Sita'
print(str1+str2)
print(str1 * 4) #to multiply, there needs to be one string and one int
(only int)
print(str1 + '10')
print(str1 * int('10'))
print(str1 * True)
print(str1 * False)   #denotes 0 times repetition
```

Output:

```
Ram Sita
Ram Ram Ram Ram
Ram 10
Ram Ram Ram Ram Ram Ram Ram Ram Ram Ram
Ram
```

Note:
To calculate length of string:

```
a= 'python'
print(len(a))
print(a.__len__())
```

Output:

```
6
6
```

Membership operator in string:
Membership operators include: in and not in.
Example:

```
a='python'
print("p" in a)
print("p" not in a)
```

Output:

```
True
False
```

Comparison and Equality Operator:
It includes: <, >, <=, >=, ==, !=
Example:

```
a='apple'
b="apple"
print(a==b)
print(a!=b)
print(a>b)
print(a<=b)
c='pple'
print(a<c)
```

Output:

```
True
False
False
True
True
```

strip(), rstrip(), lstrip():
These functions are used to ignore the blank spaces from the string entered.

Example:

```
name= input("Enter a name: ").strip()
if name == 'Ram':
 print("Correct Name")
else:
 print("Incorrect Name")
```

Output:

```
Enter a name:     Ram
Correct Name
```

Finding Substring:

There are about four functions that is needed to find substrings:
find(), index(), rfind() and rindex().
Out of these four functions, find() and index() finds in forward direction whereas the rest two in backward direction. However, all give a positive index. They always give the first occurance. find()/rfind() gives -1 if the string is not found whereas index()/rindex() gives value error if the string is not found.

Example 1:

```
a='Learning python is fun'
print(a.find('r'))
print(a.rfind('i'))
print(a.find('i'))
```

Output:

```
3
```

Example 2:

```
a='Learning python is fun'
print(a.find('i',7,60))
```

Output:

```
16
```

To count the number of substrings:

```
a='Learning python is fun'
print(a.count('i'))
print(a.count('i',7,60))
```

Output:

```
2
```

```
1
```

Replacing Substring:
Example:

```
a = "learning Python is beautiful beautiful.."
print(a.replace('beautiful','fun'))
print(a)
```

Output:

```
learning Python is fun fun..
learning Python is beautiful beautiful..
```

Splitting of String:
Inorder to split string into different substrings, split() function is used. It's return type is list.
Example:

```
a = "learning Python is beautiful beautiful..".split()
for i in a:
 print(i)
```

Output:

```
learning
Python
is
beautiful
beautiful..
```

Example 2:

```
a="2081-5-12".split('-')
for i in a:
 print(i)
```

Output:

```
2081
5
12
```

Joining of String:
The reverse operation of split is join. It can join not only list of strings but also tuples.
Example:

```
l=['Learning','Python','is','fun']
t=('Learning','Python','is','fun')
```

```
a=' '.join(l)
b=' '.join(t)
print(a)
print(b)
```

Output:

```
Learning Python is fun
Learning Python is fun
```

Changing case of string:

There are different functions to change case of strings such as: upper(), lower(), swapcase(), title(), capitalize()

Example:

```
s="Learning Python is Fun"
print(s.upper())
print(s.lower())
print(s.swapcase())
print(s.title())
print(s.capitalize ())
```

Output:

```
LEARNING PYTHON IS FUN
learning python is fun
lEARNING pYTHON IS fUN
Learning Python Is Fun
Learning python is fun
```

Checking starting and ending part of string:

Two functions are used to check these, i.e. , startswith() and endswith().

Example:

```
s="Learning Python is Fun"
print(s.startswith('Learning'))
print(s.endswith('Learning'))
print(s.endswith('n'))
```

Output:

```
True
False
True
```

To check the type of characters present in a string:

Example:

```
print("Learning Python3.12".isalnum())
print("Learning Python3 12".isalnum())
print("Learning Python".isalpha())
print("32".isdigit())
print("hi".islower())
print("Python Is Fun".istitle())
```

Output:

```
False
False
False
True
True
True
```

Formatting of strings:

There are quite a few ways of formatting strings.

Example 1:

```
name='Ram'
age='100'
print(f"My name is {name} and my age is {age}")
print("My name is {0} and my age is {1}".format(name,age))
print("My name is {a} and my age is {b}".format(a=name,b=age))
```

Output:

```
My name is Ram and my age is 100
My name is Ram and my age is 100
My name is Ram and my age is 100
```

Example 2:

```
print("The integer number is {}".format(123))
print("The integer number is {:d}".format(123))
print("The integer number is {:5d}".format(123))
print("The integer number is {:05d}".format(123))
```

Output:

```
The integer number is 123
The integer number is 123
The integer number is   123
The integer number is 00123
```