----:String in Python:----

A group of characters enclosed within single or double or triple quotes is called a string. We can say the string is a sequential collection of characters.

```
s1 = "Welcome to 'python' learning"
s2 = 'Welcome to "python" learning'
s3 = """Welcome "to" 'python' learning"""
print(s1)
print(s2)
print(s3)

O/P:--
Welcome to 'python' learning
Welcome to "python" learning
Welcome "to" 'python' learning
```

In-built functions:--

1. len()	- To check how many objects/characters present in string.
2. max()	-To check which object/character may have maximum ASCII value.
3. min()	-To check which object/character may have minimum ASCII value.
4. type()	-To check data-type
5. str()	-For type casting
6. ord()	-Character to ASCII value
7. chr()	-ASCII value to character

```
str1="Neeraj"
print(max(str1))
print(min(str1))
print(len(str1))
print(type(str1))

# find ACII value of any charactor(ord() take only one argument)
for i in str1:
   print(ord(i))
```

```
# find ACII value of any special symbol(ord() take only one argument)
x='#'
print("ASCII value of X=",ord(x))
```

```

O/P:--
r
N
6
<class 'str'>
78
101
101
114
97
106
ASCII value of X= 35
```

Accessing string characters in python:

We can access string characters in python by using,

1. Indexing
2. Slicing

Indexing:

Indexing means a position of string's characters where it stores. We need to use square brackets [] to access the string index. String indexing result is string type. String indices should be integer otherwise we will get an error. We can access the index within the index range otherwise we will get an error.

Python supports two types of indexing

1. **Positive indexing:** The position of string characters can be a positive index from left to right direction (we can say forward direction). In this way, the starting position is 0 (zero).

2. **Negative indexing:** The position of string characters can be negative indexes from right to left direction (we can say backward direction). In this way, the starting position is -1 (minus one).

Slicing:--

A substring of a string is called a slice. A slice can represent a part of a string from a string or a piece of string. The string slicing result is string type. We need to use square brackets [] in slicing. In slicing, we will not get any Index out-of-range exception. In slicing indices should be integer or None or __index__ method otherwise we will get errors.

Different Cases:

wish = “Hello World”

1. wish [::] => accessing from 0th to last
2. wish [:] => accessing from 0th to last
3. wish [0:9:1] => accessing string from 0th to 8th means (9-1) element.
4. wish [0:9:2] => accessing string from 0th to 8th means (9-1) element.
5. wish [2:4:1] => accessing from 2nd to 3rd characters.
6. wish [:: 2] => accessing entire in steps of 2
7. wish [2 ::] => accessing from str[2] to ending
8. wish [:4:] => accessing from 0th to 3 in steps of 1
9. wish [-4: -1] => access -4 to -1

Note: If you are not specifying the beginning index, then it will consider the beginning of the string. If you are not specifying the end index, then it will consider the end of the string. The default value for step is 1

```
wish = "Hello World"
print(wish[::])
print(wish[:])
print(wish[0:9:1])
print(wish[0:9:2])
print(wish[2:4:1])
print(wish[::2])
print(wish[2::])
print(wish[:4:])
print(wish[-4:-1])

O/P:--
Hello World
Hello World
Hello Wor
HloWr
ll
HloWrd
```

```
llo World
Hell
orl
```

Strings are immutable in Python:

Once we create an object then the state of the existing object cannot be changed or modified. This behavior is called immutability. Once we create an object then the state of the existing object can be changed or modified. This behavior is called mutability. A string having immutable nature. Once we create a string object then we cannot change or modify the existing object.

```
name = "Python"
print(name)
print(name[0])
name[0]="X"
O/P:-

Python
P
Traceback (most recent call last):
  File "e:\DataSciencePythonBatch\string.py", line 15, in <module>
    name[0]="X"
TypeError: 'str' object does not support item assignment
```

Mathematical operators on string objects in Python

We can perform two mathematical operators on a string. Those operators are:

1. Addition (+) operator.
2. Multiplication (*) operator.

Addition operator on strings in Python:

The + operator works like concatenation or joins the strings. While using the + operator on the string then compulsory both arguments should be string type, otherwise, we will get an error.

```
a = "Python"
b = "Programming"
print(a+b)
```

```
O/P:--

PythonProgramming
```

```
a = "Python"
b = "Programming"
print(a+" "+b)

O/P:--

Python Programming
```

Multiplication operator on Strings in Python:

This is used for string repetition. While using the * operator on the string then the compulsory one argument should be a string and other arguments should be int type.

```
a = "Python"
b = 3
print(a*b)

O/P:--
PythonPythonPython
```

Length of a string in Python:--

We can find the length of the string by using the len() function. By using the len() function we can find groups of characters present in a string. The len() function returns int as result.

```
course = "Python"
print("Length of string is:",len(course))

O/P:--
Length of string is: 6
```

Membership Operators in Python:

We can check, if a string or character is a member/substring of string or not by using the below operators:

1. In
2. not in

in operator:---

in operator returns True, if the string or character found in the main string.

```
print('p' in 'python')
print('z' in 'python')
print('on' in 'python')
print('pa' in 'python')

O/P:--
True
False
True
False
```

```
main=input("Enter main string:")
s=input("Enter substring:")
if s in main:
   print(s, "is found in main string")
else:
   print(s, "is not found in main string")

O/P:--
Enter main string:Neeraj
Enter substring:raj
raj is found in main string
```

Pre-define methods:--

1. **upper()** – This method converts all characters into upper case

```
str1 = 'python programming language'
print('converted to using title():', str1.upper())

str2 = 'JAVA proGramming laNGuage'
```

```
print('converted to using upper():', str2.upper())

str3 = 'WE ARE SOFTWARE DEVELOPER'
print('converted to using upper ():', str3.upper())

O/P:--
converted to using upper(): PYTHON PROGRAMMING LANGUAGE
converted to using upper (): JAVA PROGRAMMING LANGUAGE
converted to using upper (): WE ARE SOFTWARE DEVELOPER
```

2. **lower()** – This method converts all characters into lower case

```
str1 = 'python programming language'
print('converted to using lover():', str1.lower())

str2 = 'JAVA proGramming laNGuage'
print('converted to using lover ():', str2.lower())

str3 = 'WE ARE SOFTWARE DEVELOPER'
print('converted to using lover ():', str3.lower())

O/P:--
converted to using lover(): python programming language
converted to using lover (): java programming language
converted to using lover (): we are software developer
```

3. **swapcase()** – This method converts all lower-case characters to uppercase and all upper-case characters to lowercase

```
str1 = 'python programming language'
print('converted to using swapcase():', str1.swapcase())
str2 = 'JAVA proGramming laNGuage'
print('converted to using swapcase ():', str2.swapcase())

str3 = 'WE ARE SOFTWARE DEVELOPER'
print('converted to using swapcase ():', str3.swapcase())

O/P:--
converted to using title(): PYTHON PROGRAMMING LANGUAGE
converted to using title(): java PROgRAMMING LAngUAGE
converted to using title(): we are software developer
```

4. **title()** – This method converts all character to title case (The first character in every word will be in upper case and all remaining characters will be in lower case)

```
str1 = 'python programming language'
print('converted to using title():', str1.title())

str2 = 'JAVA proGramming laNGuage'.title()
print('converted to using title():', str2.title())

str3 = 'WE ARE SOFTWARE DEVELOPER'.title()
print('converted to using title():', str3.title())

O/P:--
converted to using title(): Python Programming Language
converted to using title(): Java Programming Language
converted to using title(): We Are Software Developer
```

5. **capitalize()** – Only the first character will be converted to upper case and all remaining characters can be converted to lowercase.

```
str1 = 'python programming language'
print('converted to using capitalize():', str1.capitalize())

str2 = 'JAVA proGramming laNGuage'
print('converted to using capitalize ():', str2.capitalize())

str3 = 'WE ARE SOFTWARE DEVELOPER'
print('converted to using capitalize ():', str3.capitalize())

O/P:--
converted to using capitalize(): Python programming language
converted to using capitalize (): Java programming language
converted to using capitalize (): We are software developer
```

6. **center():-**Python String center() Method tries to keep the new string length equal to the given length value and fills the extra characters using the default character (space in this case).

```
str = "python programming language"

new_str = str.center(40)
# here fillchar not provided so takes space by default.
print("After padding String is: ", new_str)

O/P:--
      python programming language      .
```

```
str = "python programming language"

new_str = str.center(40,'#')
# here fillchar not provided so takes space by default.
print("After padding String is: ", new_str)

O/P:--
######python programming language#######
```

```
str = "python programming language"
new_str = str.center(15,'#')
# here fillchar not provided so takes space by default.
print("After padding String is: ", new_str)

O/P:--
python programming language
```

7. **count():-- count()** function is an inbuilt function in Python programming language that returns the number of occurrences of a substring in the given string.

Syntax: string. Count(substring, start= …., end= ….)
Parameters:
The count() function has one compulsory and two optional parameters.
Mandatory parameter:
    substring – string whose count is to be found.
Optional Parameters:

start (Optional) – starting index within the string where the search starts.
end (Optional) – ending index within the string where the search ends.

```
str = "python programming language"
count = str.count('o')
# here fillchar not provided so takes space by default.
print("count of given charactor is: ", count)

O/P:--
ount of given charactor is:  2
```

```
str = "python programming language"
count = str.count('o',5,9)
# here fillchar not provided so takes space by default.
print("count of given charactor is: ", count)

O/P:--
count of given charactor is:  0
```

8. **Join():---** The string join() method returns a string by joining all the elements of an iterable (list, string, tuple), separated by the given separator.

The join() method takes an iterable (objects capable of returning its members one at a time) as its parameter.Some of the example of iterables are: Native data types - List, Tuple, String, Dictionary and Set.

```
str = ['Python', 'is', 'a', 'programming', 'language']
# join elements of text with space
print(' '.join(str))

O/P:--
Python is a programming language
```

```
str = ['Python', 'is', 'a', 'programming', 'language']
# join elements of text with space
print('_'.join(str))
O/P:-
Python_is_a_programming_language
```

```
# .join() with lists
numList = ['1', '2', '3', '4']
separator = ', '
print(separator.join(numList))

O/P:--
1, 2, 3, 4
# .join() with tuples
numTuple = ('1', '2', '3', '4')
print(separator.join(numTuple))
O/P:--
1, 2, 3, 4
s1 = 'abc'
s2 = '123'
# each element of s2 is separated by s1
# '1'+ 'abc'+ '2'+ 'abc'+ '3'
print('s1.join(s2):', s1.join(s2))

O/P:--
s1.join(s2): 1abc2abc3

# each element of s1 is separated by s2
# 'a'+ '123'+ 'b'+ '123'+ 'b'
print('s2.join(s1):', s2.join(s1))

O/P:--
s2.join(s1): a123b123c
```

```
# .join() with sets
test = {'2', '1', '3'}
s = ', '
print(s.join(test))

O/P:--
2, 3, 1

test = {'Python', 'Java', 'Ruby'}
s = '->->'
print(s.join(test))
```

```
O/P:--
Ruby->->Java->->Python
```

```
# .join() with dictionaries
test = {'mat': 1, 'that': 2}
s = '->'

# joins the keys only
print(s.join(test))

O/P:--
mat->that
```

9. **split():--** The split() method splits a string at the specified separator and returns a list of substrings.

```
str = "Python is a programming language"
print(str.split(" "))
str = "Python is a programming language"
print(str.split(",",2))
print(str.split(":",4))
print(str.split(" ",1))
print(str.split(" ",0))

O/P:--
['Python', 'is', 'a', 'programming', 'language']
['Python is a programming language']
['Python is a programming language']
['Python', 'is a programming language']
['Python is a programming language']
```

String-related operators

1. **Concatenation Operator (+):--**

```
a = "Hello"
b = "World"
print(a + " " + b)

```

```
O/P:---
Hello World
```

2. **Repetition Operator (*):---**

```
text = "Ha"
print(text * 3)

O/P:---
HaHaHa
```

3. **Membership Operators (in, not in):------**

```
msg = "Python is awesome"
print("Python" in msg)
print("java" not in msg)

O/P:----
True
True
```

4. **Comparison Operators (==, !=, <, >, <=, >=) :-----**

```
print("apple" == "apple")
print("apple" != "banana")
print("apple" < "banana")

O/P:---
True
True
True
```

5. **Indexing Operator ([]) :----**

```
name = "Neeraj"
print(name[0])
print(name[-1])
O/P:---
N
j
```

6. **Slicing Operator ([:]) :-----**

```
word = "Python"
print(word[1:4])
print(word[:2])
print(word[2:])

O/P:---
yth
Py
thon
```

7. **Assignment Operators (=, +=, *=) :-----**

```
s = "Hello"
s += " World"
print(s)
s *= 2
print(s)

O/P:---
Hello World
HelloHello
```

8. Identity Operators (is, is not) :----

```
a = "hello"
b = "hello"
print(a is b)
x = "".join(["he", "llo"])
print(a == x)
print(a is x)

O/P:---
True
True
False
```

9. Logical Operators (and, or, not) (Indirect Use) :----

```
a = "Hello"
b = ""
```

```
print(a and b)
print(a or b)
print(not a)
print(not b)

O/P:---
Hello
False
True
```