String Functions or Methods:

Python supports so many functions or methods in String concept to handling the String data. They are ,

1. capitalize():

- This function converts first letter of first word in the given string into upper case format.

```
>>> str1 = 'python developer'
>>> str1.capitalize()              # 'Python developer'
```

2. title():

- This function converts first character of each word in the given string into upper case format.

```
>>> str1 = 'python developer'
>>> str1.title()              # 'Python Developer'
```

3. islower():

- This function checks whether the given string contains all lower case letters or not. If all are lower case then it will return True else False.

```
>>> str1='python developer'
>>> str1.islower()                    # True
>>> str2="Python"
>>> str2.islower()                    # False
```

4. isupper():

- This function checks whether the given string contains all upper case letters or not. If all are upper case then it will return True else False.

```
>>> str1='python developer'
>>> str1.isupper()                    # False
>>> str3='PYTHON'
>>> str3.isupper()                    # True
```

5. lower():

- This function converts all letters of given string into lower case.

```
>>> str3='PYTHON'
>>> str3.lower()                      'python'
```

6. upper():

- This function converts all letters of given string into upper case.

```
>>> str1='python developer'
>>> str1.upper()                    'PYTHON DEVELOPER'
```

7. len():

- This function counts the number of characters in the given string and returns count value.

```
>>> str1='python developer'
>>> len(str1)                    16
```

8. count(element, start_index_position, end_index_position):

- This function counts number of occurrences of a specific character in the given string and returns count value.
- here start_index_position value is starts with 0 by default and end_index_position ends with -1 by default.

```
>>> str1='python developer'
>>> str1.count('o')                  2
>>> str1.count('o',5)                1
```

9. find(element, start_index_position, end_index_position):

Syntax : str.find(element , index_position)

- This function finds the index position of specific character in the given string.
- By default index position value is taking zero (0), means element searching from 0 index place onwords.
- If we specifiying any index value then searching starts from that index value only.
- If searching element not available in given string then it returns -1 value.

```
>>> str1='python developer'
>>> str1.find('o')              4      --->> for first occurrence of 'o'
>>> str1.find('o', 5 )                 12     --->> for second occurrence of 'o'
```

10. index(element, start_index_position, end_index_position):

- The index() method finds the first occurrence of the specified value.
- The index() method raises an exception if the value is not found.
- The index() method is almost the same as the find() method, the only difference is that the find() method returns -1 if the value is not found.

Syntax : string.index(element_value, start_index_value, end_index_value)

Parameter	Description
value	Required. The value to search for

start	Optional. Where to start the search. Default is 0
end	Optional. Where to end the search. Default is to the end of the string

11. split():

- This function splits the given string into multiple strings and returns in the form of list of strings.
- By default space is taken as splitting parameter.

```
>>> str1='python developer'
>>> str1.split()                    ['python', 'developer']
```

- If you want to splitting the string by using any special parameter insted of space then we use that special parameter using split().

```
>>> str1='python developer'
>>> str1.split('o')          ['pyth', 'n devel', 'per']
```

12. splitlines()

- By using this **splitlines()** method we can find total number of rows from a given string and it returns in the form of list and every single line as a one strings .
- It is used \n as a separator.

```
>>> s1 = """Python is esay.
Python is more simple.
Python is a language
"""
>>> print(s1.splitlines())
['Python is esay.', 'Python is more simple.', 'Python is a language']
>>> print(len(s1.splitlines()))
3
```

13. swapcase():

- This function swaps all lower case letters into upper case and swaps all upper case letters into lower case letters.

```
>>> str1='PyThOn'
>>> str1.swapcase()                 'pYtHoN'
```

14. reversed():

- This function reverses the given string and returns reversed object only but not values directly.

- If you want to get the reversed string for given string then we use join() method.
- reversed() method is not done any changes in the given main string and result will store into a separate variable.

```
>>> str='Python'
>>> print(str)                    Python
>>> str1=''.join(reversed(str))
>>> print(str1)                          nohtyP
>>> s
'Srinivas'
>>> reversed(s)
<reversed object at 0x0042D5D0>
# it prints reversed object only. It creates new object. use join() to get reverse string.
>>> ''.join(reversed(s))
'savinirS'
```

15. sorted(obj) :

- It will sorting the characters in assending order by default.

```
>>> st = 'Python'
>>> sorted(st)
['P', 'h', 'n', 'o', 't', 'y']
```

- To get as a string format then use join() like below,

```
>>> ''.join(sorted(st))
'Phnoty'
```

Q) How to display the given string in descending order?

```
>>> st="python"
>>> ''.join(reversed(sorted(st)))
'ytponh'
or
>>> ''.join(sorted(st,reverse=True))
'ytonhP'
```

Q) How to display the given string with two dots between each character?

```
>>> s1='python'
```

```
>>> s10='..'.join(s1)
>>> s10
'p..y..t..h..o..n'
```

16. replace():

- This function replaces an existing character(s) with new character(s).

syntax : str.replace(old_sub_string , new_sub_string)

```
>>> str1='python learner'
>>> print(str1)                    python learner
>>> str2=str1.replace('learner','developer')
>>> print(str2)                    python developer
>>> print(str1)                python learner
```

Note: we can remove any character(s) with non-empty space.

```
>>> str1='Python'
>>> print(str1)                    Python
>>> type(str1)                     <class 'str'>
>>> str2 = str1.replace('thon','')
>>> print(str2)                    Py
>>> type(str2)                     <class 'str'>
```

del :

- We can also remove string object by using "del" command, but we can not delete string object elements(charecters) because string is immutable object.

Example:

```
>>> str1="Python Srinivas"
>>> print(str1)              Python Srinivas
>>> type(str1)               <class 'str'>
>>> id(str1)                 63806464
>>> del str1                 #deleting str1
>>> print(str1)              #after deleting
```

NameError: name 'str1' is not defined

Note:

```
>>> del str1[0]
```

TypeError: 'str' object doesn't support item deletion

17. format() :

- The format() method formats the specified value(s) and insert them inside the string's placeholder.
- The placeholder is defined using curly brackets: {}.
- The placeholders can be identified using named indexes {price}, numbered indexes {0}, or even empty placeholders {}.
- The format() method returns the formatted string.

Syntax : string.format(value1, value2...)

Example:

```
>>> txt1 = "My name is {fname}, My age is {age}".format(fname = "Srinivas", age = 30)
>>> txt1
"My name is Srinivas, My age is 30"
>>> a = "Employee name is {0} and age is {1}".format(20,'Kiran')
>>> a
'Employee name is 20 and age is Kiran'
>>> a = "Employee name is {1} and age is {0}".format(20,'Kiran')
>>> a
'Employee name is Kiran and age is 20'
>>> name = 'Ramu'
>>> age = 30
>>> a = "Employee name is {} and age is {}".format('Ravi',40)
>>> a
'Employee name is Ravi and age is 40'
```

When Errors Coming:

```
a = "Employee name is {name} and age is {age}".format(age,name)
```

Outpput : KeyError: 'name'

```
a = "Employee name is {0} and age is {1}".format(age=20,name='Kiran')
```

Output : **IndexError**: tuple index out of range

18. endswith():

- The endswith() method returns True if the string ends with the specified value, otherwise False.

syntax : string.endswith(value, start_index_value, end_index_value)

f’string concept’

==========

```
>>> ename="Virat"
>>> age =30
>>> f'Employee name is {ename} and age is {age}'
'Employee name is Virat and age is 30'
```

Q. Check if the string ends with a punctuation sign (.):

```
>>> txt = "Hello, welcome to my world."
>>> x = txt.endswith(".")
>>> x
True
```

19. startswith() :

- Returns true if the string starts with the specified value

```
>>> txt = "Hello, welcome to my world."
>>> x = txt.startswith("wel", 7, 20)
>>> x
True
# use a dictionary with ascii codes to replace 83 (S) with 80 (P):
>>> mydict = {83:  80};
>>> txt = "Hello Sam!";
>>> print(txt.translate(mydict));
```

20. strip():

- The strip() method removes any leading (spaces at the beginning) and trailing (spaces at the end) characters (space is the default leading character to remove)

```
>>> txt = "     banana     "
>>> x = txt.strip()
>>> print("of all fruits", x, "is my favorite")
of all fruits banana is my favorite
```

21. rstrip() : Returns a right trim version of the string

22. lstrip() : Returns a left trim version of the string

Questions: