

7.8 STRING METHODS AND BUILT-IN FUNCTIONS

Python provides several built-in functions associated with the strings. These functions allow us to easily modify and manipulate strings. We can think of functions as being actions that we perform on elements of a string in our code. Built-in functions/string manipulation functions are pre-defined in Python programming language and are readily available for use. The syntax is:

String_Object.functionName()

Let us understand these methods.

1. **len()**—This method returns the length of the string.

Syntax:

```
len(str)
```

Here str is the string.

For example,

```
>>> word='Good Morning'
```

```
>>> len(word)
```

```
12
```

```
>>> str1='This is Meera\'s pen'
```

```
>>> len(str1)
```

```
19
```

Here \' is counted as one character.

2. **capitalize()**—This method returns the exact copy of the string with the first letter in uppercase.

Syntax:

```
str.capitalize()
```

Here str is the string.

For example,

```
>>> str1='welcome'
```

```
>>> str1.capitalize()
```

```
'Welcome'
```

```
>>> str1='Welcome'
```

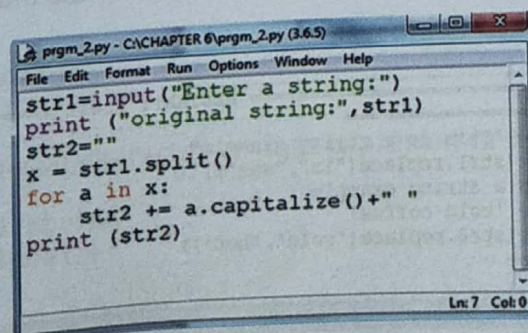
```
>>> str1.capitalize()
```

```
'Welcome'
```

Note: If the string has its first character as capital, then it returns the original string.

Practical Implementation-4

Write a program to accept a string (a sentence) and return a string having first letter of each word in capital letter.



```
prgm_2.py - C:\CHAPTER 0\prgm_2.py (3.6.5)
File Edit Format Run Options Window Help
str1=input("Enter a string:")
print ("original string:",str1)
str2=""
x = str1.split()
for a in x:
    str2 += a.capitalize()+" "
print (str2)
```



```

Python 3.6.5 Shell
File Edit Shell Debug Options Window Help
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 16:07:46) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:\CHAPTER 6\prgm_2.py =====
Enter a string:we are learning python
original string:we are learning python
We Are Learning Python
>>>
Ln: 8 Col: 4

```

3. **split()**—The `split()` method breaks up a string at the specified separator and returns a list of substrings.

Syntax:

```
str.split([separator, [maxsplit]])
```

The `split()` method takes a maximum of 2 parameters:

- **separator** (optional)—The separator is a delimiter. The string splits at the specified separator. If the separator is not specified, any whitespace (space, newline, etc.) string is a separator.
- **maxsplit** (optional)—The `maxsplit` defines the maximum number of splits. The default value of `maxsplit` is `-1`, which means no limit on the number of splits.

For example,

```

===== RESTART: Shell =====
>>> x= 'blue;red;green'
>>> x.split(";")
['blue', 'red', 'green']
>>> text='Love your country'
>>> print(text.split())
['Love', 'your', 'country']
>>>

```

Here, separator is not specified, so by default, space is a string separator.

The second argument 'maxsplit' is optional and its default value is zero. If an integer value 'n' is given for the second argument, the string is split into 'n+1' strings.

```

===== RESTART: Shell =====
>>> grocery= 'Red:Blue:Orange:Pink'
>>> print(grocery.split(':',2))
['Red', 'Blue', 'Orange:Pink']
>>> print(grocery.split(':',1)) #maxsplit is 1
['Red', 'Blue:Orange:Pink']
>>> print(grocery.split(':',5))
['Red', 'Blue', 'Orange', 'Pink']
>>> print(grocery.split(':',0)) #maxsplit is 0, so it will return the string
['Red:Blue:Orange:Pink']
>>>
Ln: 29 Col: 4

```

4. **replace()**—This function replaces all the occurrences of the old string with the new string.

Syntax: `str.replace(old, new)`

```

===== RESTART: Shell =====
>>> str1= "This is a string example"
>>> print(str1.replace("is","was"))
Thwas was a string example
>>> str2= 'cold coffee'
>>> print(str2.replace('cold','hot'))
hot coffee
>>>
Ln: 37 Col: 4

```


5. **find()**—This function is used to search the first occurrence of the substring in the given string. The find() method returns the lowest index of the substring if it is found in the given string. If the substring is not found, it returns -1. Its syntax is:

```
str.find(sub, start, end)
```

Here,

sub: is the substring which needs to be searched in the given string

start: starting position where substring is to be checked within the string

end: end position is the index of the last value for specified range

```
>>> word = 'Green revolution'
>>> result= word.find('Green')
>>> print (result)
0
>>> result= word.find('green')
>>> print (result)
-1
>>> result=word.find("e",4)
>>> print (result)
7
>>> result=word.find("en",5)
>>> print (result)
-1
>>> result=word.find('o',11,14) #Finding letter 'o'
>>> print (result) # from 11th to 14th position
-1
```

In the last statement, letter 'o' is to be searched for starting from 11th to 13th position (as index 14 is excluded in search). Since 'o' lies at 14th position, it will result in an unsuccessful search and shall return -1 as the output.

6. **index()**—This function is quite similar to find() function as it also searches the first occurrence and returns the lowest index of the substring if it is found in the given string but raises an exception if the substring is not present in the given string.

Syntax:

```
index(substring, start, end)
```

```
>>> str1 = "Hello ITS all about STRINGS!!"
>>> str1.index('Hello')
0
>>> str1.index('Hi')
Traceback (most recent call last):
  File "<pyshell#9>", line 1, in <module>
    str1.index('Hi')
ValueError: substring not found
>>>
```

7. **isalpha()**—This function checks for alphabets in an inputted string. It returns True if the string contains only letters, otherwise returns False.

Syntax:

```
str.isalpha()
```

```
>>> str = "Good"
>>> print(str.isalpha())
True
```

Returns True as no special character or digit is present in the string.

```
>>> str1="This is a string"
>>> print(str1.isalpha())
False
```

Returns False as the string contains spaces.


```
>>> str1="Working with ...Python!!"
>>> print(str1.isalpha())
False
```

Returns False as the string contains special characters and spaces.

8. **isalnum()**—The `isalnum()` method returns True if all the characters are alphanumeric, i.e., alphabet letters (A-Z, a-z) and numbers (0-9).

Example of characters that are not alphanumeric: (space) ! # % & ? etc.

Syntax:

```
string.isalnum()
```

For example,

```
text1="Python 3.8"
x = text1.isalnum()
print(x)
```

```
>>>
RESTART: C:/Users/preeti
func.py
False
>>>
```

The above function shall return output as False since the text contains space also.

```
text1="Python38"
x = text1.isalnum()
print(x)
```

```
>>>
RESTART: C:/Users/preeti
func.py
True
```

As is evident from the second example, the output returned is True since text includes only text and numbers.

9. **isdigit()**—This function returns True if the string contains only digits, otherwise False.

Syntax: `str.isdigit()`

```
>>> str1 = "123456"
>>> print(str1.isdigit())
True
```

Returns True as the string contains only digits.

```
>>> str1 = "Ram bagged 1st position"
>>> print(str1.isdigit())
False
```

Returns False because apart from digits, the string contains letters and spaces.

10. **title()**—This function returns the string with first letter of every word in the string in uppercase and rest in lowercase.

Syntax: `str.title()`

```
>>> str1 = "hello ITS all about STRINGS!!"
>>> str1.title()
'Hello Its All About Strings!!'
>>>
```

Thus, `title()` returns a version of the string where each word is title-cased.

11. **count()**—This function returns number of times substring str occurs in the given string. If we do not give start index and end index then searching starts from index 0 and ends at length of the string.

Syntax: `str.count(substring, start, end)`

```
>>> str1 = 'Hello World! Hello Hello'
>>> str1.count('Hello',12,25)
2
>>> str1.count('Hello')
3
```

12. **lower()**—This function converts all the uppercase letters in the string into lowercase.

Syntax: `str.lower()`

```
>>> str1= "Learning PYTHON"
>>> print(str1.lower())
learning python
```

Converts uppercase letters only to lowercase. If already in lowercase, then it will simply return the string.

```
>>> str1= "learning python"
>>> print(str1.lower())
learning python
```

13. **islower()**—This function returns True if all the letters in the string are in lowercase.

Syntax:

`str.islower()`

```
>>> str1="python"
>>> print(str1.islower())
True
>>> str1 = "Python"
>>> print(str1.islower())
False
```

14. **upper()**—This function converts lowercase letters in the string into uppercase.

Syntax:

`str.upper()`

```
>>> var1= 'Welcome'
>>> print(var1.upper())
WELCOME
```

If already in uppercase, then it will simply return the string.

```
>>> var1= 'WELCOME'
>>> print(var1.upper())
WELCOME
```

15. **isupper()**—This function returns True if the string is in uppercase.

Syntax:

`str.isupper()`

```
>>> str1= "PYTHON"
>>> print(str1.isupper())
True
>>> str1= "PythOn"
>>> print(str1.isupper())
False
```


16. **lstrip()**—This function returns the string after removing the space(s) from the left of the string.

Syntax:

```
str.lstrip()
```

or

```
str.lstrip(chars)
```

chars (optional)—a string specifying the set of characters to be removed from the left. All combinations of characters in the *chars* argument are removed from the left of the string until left character of the string mismatches.

```
>>> str1= "  Green Revolution"
>>> print(str1.lstrip())
Green Revolution
```

Here no argument is given, hence, it removed all leading whitespaces from the left of the string.

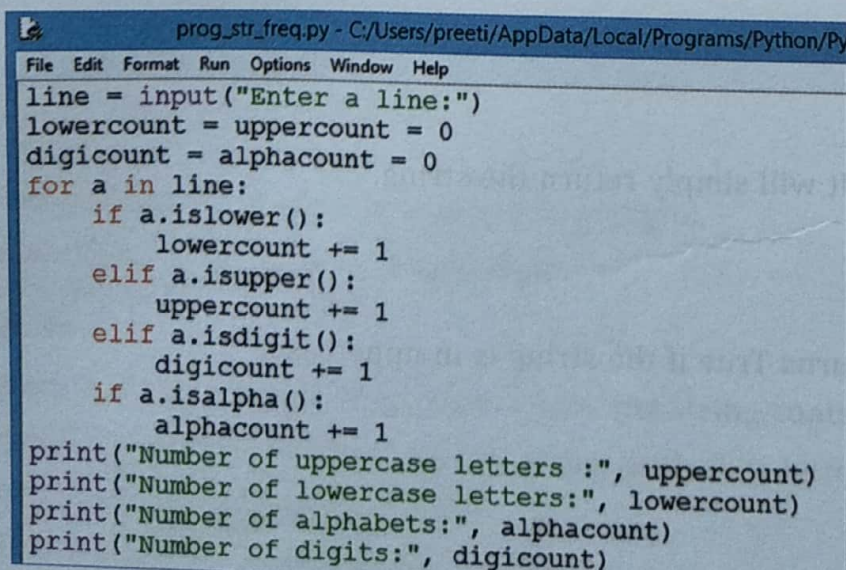
```
>>> str2= "Green Revolution"
>>> print(str2.lstrip("Gr"))
een Revolution
>>> str2= "Green Revolution"
>>> print(str2.lstrip("rG"))
een Revolution
```

Here, all elements of the given argument are matched with left of the str2 and, if found, are removed.

Practical Implementation-5

Write a program that reads a line and prints its frequency chart like,

- Number of uppercase letters
- Number of lowercase letters
- Number of alphabets
- Number of digits



```
prog_str_freq.py - C:/Users/preeti/AppData/Local/Programs/Python/Py
File Edit Format Run Options Window Help
line = input("Enter a line:")
lowercount = uppercount = 0
digicount = alphacount = 0
for a in line:
    if a.islower():
        lowercount += 1
    elif a.isupper():
        uppercount += 1
    elif a.isdigit():
        digicount += 1
    if a.isalpha():
        alphacount += 1
print("Number of uppercase letters :", uppercount)
print("Number of lowercase letters:", lowercount)
print("Number of alphabets:", alphacount)
print("Number of digits:", digicount)
```



```
>>>
RESTART: C:/Users/preeti/AppData/Local
q.py
Enter a line:Python for BIG data 2020
Number of uppercase letters : 5
Number of lowercase letters: 11
Number of alphabets: 16
Number of digits: 4
```

17. **rstrip()**—This function removes all the trailing whitespaces from the right of the string.

Syntax:

```
rstrip()
```

or

```
str.rstrip(chars)
```

chars (optional)—a string specifying the set of characters to be removed from the right.

All combinations of characters in the *chars* argument are removed from the right of the string until the right character of the string mismatches.

```
>>> str1= "Green Revolution  "
>>> print(str1.rstrip())
Green Revolution
```

Here, no argument is given, hence, it removed all leading whitespaces from the right of the string.

```
>>> str1= "Computers"
>>> print(str1.rstrip("rs"))
Compute
```

Here, the letters 'rs' are passed as an argument; it is matched from the right of the string and removed from the right of the str1.

18. **strip()**—This function returns the string after removing the spaces both on the left and the right of the string.

Syntax:

```
str.strip()
```

```
>>> str1 = "    Hello ITS all about STRINGS!!    "
>>> str1.strip()
'Hello ITS all about STRINGS!!!'
>>>
```

19. **isspace()**—This function returns True if the string contains only whitespace characters, otherwise returns False.

Syntax:

```
str.isspace()
```

```
>>> str1= "    "
>>> print(str1.isspace())
True

>>> str1="  Python"
>>> print(str1.isspace())
False
```

20. **istitle()**—The *istitle()* function doesn't take any arguments. It returns True if the string is properly "title-cased", else returns False if the string is not a "title-cased" string or an empty string.

Syntax:

```
str. istitle()
>>> str1= "All Learn Python"
>>> print(str1.istitle())
True
>>> s= "All learn Python"
>>> print(s.istitle())
False
>>> s= "This Is @ Symbol"
>>> print(s.istitle())
True
>>> s= "PYTHON"
>>> print(s.istitle())
False
```

21. **join(sequence)**—This function returns a string in which the string elements have been joined by a string separator.

Syntax:

```
str.join(sequence)
```

sequence—Join() takes an argument which is of sequence data type, capable of returning its element one at a time. This method returns a string which is the concatenation of each element of the string and the string separator between each element of the string.

```
>>> str1= '12345'
>>> s= '-'
>>> s.join(str1)
'1-2-3-4-5'
>>> str2= 'abcd'
>>> s='#'
>>> s.join(str2)
'a#b#c#d'
```

22. **swapcase()**—This function converts and returns all uppercase characters into lowercase and vice versa of the given string. It does not take any argument.

Syntax:

```
str.swapcase()
```

The swapcase() function returns a string with all the cases changed.

```
>>> str1= "Welcome"
>>> str1.swapcase()
'wELCOME'
>>> str2= "PYTHON"
>>> str2.swapcase()
'python'
>>> s= "pYThON"
>>> s.swapcase()
'PyThON'
```

23. **partition(Separator)**—Partition function is used to split the given string using the specified separator and return a tuple with three parts:

- substring before the separator;
- separator itself;
- a substring after the separator.

Syntax:

```
str.partition(Separator)
```

Separator: This argument is required to separate a string. If the separator is not found, it returns the string itself followed by two empty strings within parentheses as tuple.

```
>>> str3= 'xyz@gmail.com'
>>> str3.partition('@')
('xyz@gmail.com', '@', '')
```


Here, separator is not found, so returns the string itself, followed by two empty strings.

```
>>> str2= "Hardworkpays"  
>>> str2.partition('work')  
( 'Hard', 'work', 'pays' )
```

Here, str2 is separated in three parts:

- 1) substring before separator, i.e., 'Hard',
- 2) separator itself, i.e., 'work', and
- 3) substring part after separator, i.e., 'pays'.

```
>>> str4= str2.partition('-')  
>>> print(str4)  
( 'Hardworkpays', '', '' )
```

```
>>> str5= str3.partition('@')  
>>> print(str5)  
( 'xyz', '@', 'gmail.com' )
```

24. **endswith()**—This function returns True if the given string ends with the specified substring, else returns False.

Syntax:

```
str.endswith(substr)
```

```
>>> a="Artificial Intelligence"  
>>> a.endswith('Intelligence')  
True  
>>> a.endswith('Artificial')  
False
```

25. **startswith()**—This function returns True if the given string starts with the specified substring, else returns False.

Syntax:

```
str.startswith(substr)
```

```
>>> b='Machine Learning'  
>>> b.startswith('Mac')  
True  
>>> b.startswith('learning')  
False
```

Above functions in a Nutshell:

```
>>> "Hello World".upper().lower()  
'hello world'  
>>> "Hello World".lower().upper()  
'HELLO WORLD'  
>>> "Hello World".find("Wor",1,6)  
-1  
>>> "Hello World".find("Wor")  
6  
>>> "Hello World".isalpha()  
False  
>>> "Hello World".isalnum()  
False  
>>> "1234".isdigit()  
True  
>>> "123GH".isdigit()  
False  
>>> "Hello World".endswith("World")  
True  
>>> "Hello World".endswith("rld")  
True  
>>> "Hello World".endswith("Wor")  
False  
>>> "Hello World".startswith("Hello")  
True
```


7.9 OTHER FUNCTIONS

In internal storage or memory of the computer, the characters are stored in integer value. A specific value is used for a given character and it is based on ASCII code. There are different numbers assigned to capital letters and small letters.

Python provides two functions for character encoding: `ord()` and `chr()`.

☞ **`ord()`**—This function returns the ASCII/Unicode (Ordinal) of the character.

```
>>> ch= 'b'
>>> ord(ch)
98
>>> ord('A')
65
```

☞ **`chr()`**—This function returns the character represented by the inputted Unicode/ASCII number.

```
>>> chr(97)
'a'
>>> chr(66)
'B'
```