Interesting Inference

Using the same string slicing technique, you will find that

- for any index *n*, s[:n] + s[n:] will give you original string s.

This works even for *n* negative or out of bounds.

Consider the string namely word storing *'amazing'*.

```
>>> word[3:], word [:3]
'zing' 'ama'
```



```
>>> word[:3] + word[3:]
'amazing'
>>> word[:-7], word [-7:]
'' 'amazing'
>>> word[:-7] + word[-7:]
'amazing'
```

NOTE

String [: : –1] is an easy way to reverse a string.

- Index out of bounds causes error with strings but slicing a string outside the bounds does not cause error.

```
s = "Hello"
print (s[5])
```

Will cause error because 5 is invalid index-out of bounds, for string "Hello"

But if you give

```
s = "Hello"
print (s[4 : 8])
print (s[5 : 10])
```

One limit is outside the bounds (length of Hello is 5 and thus valid indexes are 0-4)

Both limits are outside the bounds

the above will not give any error and print output as :

```
o

```

empty string

i.e., letter o followed by empty string in next line.

2.2.5 String Functions

Python also offers many built-in functions and methods for string manipulation. The string manipulation methods that are being discussed below can be applied to strings as per following syntax :

```
<stringObject>.<method name> ()
```

For instance, if you have a string namely **str** = "Rock the World" and you want to find its length, you will write the code somewhat like shown below :

```
>>> str = "Rock the World."
>>> str.length( )
15
>>> str2 = "New World"
>>> str2.length( )
9
```

see the string object is **str** *and method name is* **length( )**.

Do you know that following websites and web applications have used Python extensively : *Instagram, Dropbox, Google, Netflix, Spotify, Quora, Reddit, Facebook,* and many others ?

Table 2.1 *Python's built-in string manipulation methods*

`string.capitalize()`	Returns a copy of the ***string*** with its first character capitalized.
	Example
	`>>> ' i love my India'.capitalize()`
	`I love my India`

`string.find (sub[, start[, end]])`	Returns the lowest index in the ***string*** where the substring ***sub*** is found within the slice range of *start* and *end*. Returns -1 if ***sub*** is not found.
	Example
	`>>> string = 'it goes as - ringa ringa roses'`
	`>>> sub = 'ringa'`
	`>>> string.find(sub, 15, 22)`
	`-1`
	`>>> string.find(sub, 15, 25)`
	`19`

`string.isalnum()`
`string.isalpha()`
`string.isdigit()`
	Returns True if the characters in the ***string*** are alphanumeric (alphabets or numbers) and there is at least one character, False otherwise.
	Returns True if all characters in the ***string*** are alphabetic and there is at least one character, False otherwise.
	Returns True if all the characters in the ***string*** are digits. There must be at least one digit, otherwise it returns False.
	Examples
	`>>> string = "abc123"`
	`>>> string2 = 'hello'`
	`>>> string3 = '12345'`
	`>>> string4 = ' '`
	`>>> string.isalnum()` `True`
	`>>> string2.isalnum()` `True`
	`>>> string3.isalnum()` `True`
	`>>> string4.isalnum()` `False`
	`>>> string.isalpha()` `False`
	`>>> string2.isalpha()` `True`
	`>>> string3.isalpha()` `False`
	`>>> string4.isalpha()` `False`
	`>>> string.isdigit()` `False`
	`>>> string2.isdigit()` `False`
	`>>> string3.isdigit()` `False`
	`>>> string4.isdigit()` `True`

`string.isspace()`	Returns True if there are only whitespace characters in the ***string***. There must be at least one character. It returns False otherwise.
	Example
	`>>> string = "  "  # stores three spaces`
	`>>> string2 = ""  # an empty string`
	`>>> string.isspace()`
	`True`
	`>>> string2.isspace()`
	`False`

string.islower()

string.isupper()

Returns **True** if all cased characters in the ***string*** are lowercase. There must be at least one cased character. It returns **False** otherwise.

Tests whether all cased characters in the ***string*** are uppercase and requires that there be at least one cased character. Returns **True** if so and **False** otherwise.

Examples

```
>>> string = 'hello'
>>> string2 = 'THERE'
>>> string3 = 'Goldy'
>>> string.islower()
True
>>> string2.islower()
False
>>> string3.islower()
False
```

```
>>> string = "HELLO"
>>> string2 ="There"
>>> string3 = "goldy"
>>> string.isupper()
True
>>> string2.isupper()
False
>>> string3.isupper()
False
>>> string4.isupper()
True
>>> string5.isupper()
False
```

string.lower()

Returns a copy of the ***string*** converted to lowercase. **Example**

```
>>> string.lower()          #string = "HELLO"
'hello'
```

string.upper()

Returns a copy of the ***string*** converted to uppercase. **Example**

```
>>> string.upper()          #string = "hello"
'HELLO'
```

string.lstrip([chars])

string.rstrip([chars])

Returns a copy of the ***string*** with leading characters removed.

If used without any argument, it removes the leading whitespaces.

One can use the optional **chars** argument to specify a set of characters to be removed.

The **chars** argument is not a prefix ; rather, all combinations of its values (all possible substrings from the given string argument **chars**) are stripped when they lead the ***string***.

Returns a copy of the *string* with trailing characters removed.

If used without any argument, it removes the leading whitespaces.

The **chars** *argument* is a ***string*** specifying the set of characters to be removed.

The **chars** argument is not a suffix; rather, all combinations of its values are stripped.

Examples

```
>>> string2 = 'There'
'There'
>>> string2.lstrip( 'The' )
're'
>>> "saregamapadhanisa".lstrip( "tears" )
'gamapadhanisa'
>>> string2.rstrip('care')
'Th'
>>> "saregamapadhanisa".rstrip( "tears" )
'saregamapadhani'
```

'The', 'Th', 'he', 'Te', 'T', 'h', 'e'" and their reversed strings are matched, if any of these found, is removed from the left of the string 'The' found , hence removed