String Methods

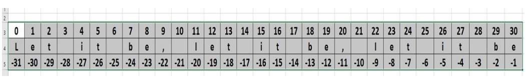
In this notebook we will discuss some of the many String methods.
Note that each of the methods is applied (or called) using the following syntax `variablename.methodname(argument values if any)`

Also keep in mind that calling the method as above on a string variable does not change the value of the original variable. If you want to retain the results, you will need to store it in another variable

Assume you have a string variable called **`my_str`**.

1. `my_str.find(sub [, start[, end]])`
    - The `find()` method returns the index of first occurrence of the substring (if found). If not found, it returns -1.
    - if the start value is provided, then the search begins from the start value
    - if the end value is provided, the search stops at one index before the end value
2. `my_str.capitalize()`
    - the `capitalize()` method converts first character of a string to uppercase letter and lowercases all other characters, if any.
3. `my_str.lower()`
    - `lower()` method converts all uppercase characters in a string into lowercase characters and returns it.
4. `my_str.islower()`
    - `islower()` method returns True if all alphabets in a string are lowercase alphabets. If the string contains at least one uppercase alphabet, it returns False.
5. `my_str.upper()`
    - `lower()` method converts all uppercase characters in a string into lowercase characters and returns it
6. `my_str.isupper()`
    - `isupper()` method returns True if all alphabets in a string are uppercase alphabets. If the string contains at least one lowercase alphabet, it returns False.
7. `my_str.lstrip([chars])`
    - `lstrip()` removes characters from the left based on the argument (a string specifying the set of characters to be removed). Default is whitespace
8. `my_str.rstrip([chars])`
    - `rstrip()` removes characters from the right based on the argument (a string specifying the set of characters to be removed). Default is whitespace
9. `my_str.strip()`
    - `my_str()` method removes both leading and trailing characters as specified by the argument.
10. `my_str.split([separator [, maxsplit]])`
    - `my_str.split()` method breaks up a string at the specified separator and returns a list of strings.
11. `my_str.replace(old, new [, count])`
    - `replace()` method returns a copy of the string where all occurrences of a substring is replaced with another substring. The optional count argument specifies the number of times you want to replace the old substring with the new substring
12. `my_str.startswith(prefix[, start[, end]])`
    - `str.startswith()` method returns True if a string starts with the specified prefix(string). If not, it returns False.

Note that all the methods listed above keep the original string unchanged.



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
L	e	t		i	t		b	e	,		l	e	t		i	t		b	e	,		l	e	t		i	t		b	e
-31	-30	-29	-28	-27	-26	-25	-24	-23	-22	-21	-20	-19	-18	-17	-16	-15	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

In [19]:
```python
'''
The following code will return the index of the first occurence of 'be'.
'''
quote = 'Let it be, let it be, let it be'
s = quote.find('be')
print(s)
print(quote)
```
```
7
Let it be, let it be, let it be
```

In [20]:
```python
'''
The following code will return the index of the first occurence, after index 27, of 'be'.
'''
quote = 'Let it be, let it be, let it be'
s = quote.find('be', 28)
print(s)
print(quote)
```
```
29
Let it be, let it be, let it be
```

In [21]:
```python
'''
The following code will return the index of the first occurence of 'be', between the indexes 8 and 24.
'''
quote = 'Let it be, let it be, let it be'
s = quote.find('be', 8, 25)
print(s)
print(quote)
```
```
18
Let it be, let it be, let it be
```

In [22]:
```python
quote = 'Let it be, let it be, let it be'
s = quote.find('be', 0, 8)
print(s)
print(quote)
```
```
-1
Let it be, let it be, let it be
```

In [23]:
```python
quote = 'Let it be, let it be, let it be'
s = quote.find('be', 0, 15)
print(s)
print(quote)
```
```
7
Let it be, let it be, let it be
```

`my_str.capitalize().` Note that the original string remains unchanged.

In [24]:
```python
my_string = "1python is AWes09ome."

capitalized_string = my_string.capitalize()

print('Old String: ', my_string)
print('Capitalized String:', capitalized_string)
```
```
Old String:  1python is AWes09ome.
Capitalized String: 1python is awes09ome.
```

In [25]:
```python
my_string = "python is AWes09ome."

capitalized_string = my_string.capitalize()

print('Old String: ', my_string)
print('Capitalized String:', capitalized_string)
```
```
Old String:  python is AWes09ome.
Capitalized String: Python is awes09ome.
```

`my_str.lower().` Note that the original string remains unchanged.

In [26]:
```python
my_str = "THIS Should BE LOWERCASE!"
my_lc_str = my_str.lower()
print(my_str)
print(my_lc_str)
```
```
THIS Should BE LOWERCASE!
this should be lowercase!
```

`my_str.lower().` The string can contain a mix of alphanumeric characters. Non-numeric characters will convert to lower case.

In [27]:
```python
my_str = "Th!s Sh0uLd B3 L0w3rCas3!"
print(my_str.lower())
print(my_str)
```
```
th!s sh0uld b3 l0w3rcas3!
Th!s Sh0uLd B3 L0w3rCas3!
```

In [28]:
```python
my_str = "THIS is LOWERCASE!"
print(my_str.islower())
```
```
False
```

`my_str.isupper()` and `my_str.upper()` work similarly for upper case

`my_str.lstrip([chars]).` The original string remains unchanged. `lstrip()` takes an optional string argument. If this argument is provided, any permutation of the characters (or subset of characters) in the optional argument is removed if they form the start of the string.

In [29]:
```python
random_string = '   this is good '

# Leading whitepsace are removed
print('string is:', random_string)
print('length is:', len(random_string))
print('substring is:', random_string.lstrip())
print('length is:', len(random_string.lstrip()))
```
```
string is:    this is good
length is: 16
substring is: this is good
length is: 13
```

In [30]:
```python
# Argument doesn't contain leading spaces.  So no characters will be removed.
random_string = 'this is good '
print('string is:', random_string)
print('length is:', len(random_string))
print('substring is:', random_string.lstrip())
print('length is:', len(random_string.lstrip()))
```
```
string is: this is good
length is: 13
substring is: this is good
length is: 13
```

In [31]:
```python
#Any permutation of the characters prioivided in the optional argument is removed if they form the leading character
s.
random_string = 'this is good '
print('string is:', random_string)
print('length is:', len(random_string))
print('substring is:', random_string.lstrip('thi'))
print('length is:', len(random_string.lstrip('thi')))
```
```
string is: this is good
length is: 13
substring is: s is good
length is: 10
```

In [32]:
```python
'''
Any permutation of the characters (or subset of characters) provided in the optional argument is removed if they for
m the
leading characters.  In the example below, the first eight chacracters in the string are a permutation of the charac
ters
in the argument.  Hence they will be removed.
'''
random_string = 'this is good '
print('string is:', random_string)
print('length is:', len(random_string))
print('substring is:', random_string.lstrip('htis '))
print('length is:', len(random_string.lstrip('htis ')))
```
```
string is: this is good
length is: 13
substring is: good
length is: 5
```

In [33]:
```python
'''
A few more examples.
'''
random_string = 'thisis! good !'
print('string is:', random_string)
print('length is:', len(random_string))
print('substring is:', random_string.lstrip('htis'))
print('length is:', len(random_string.lstrip('htis')))
```
```
string is: thisis! good !
length is: 14
substring is: ! good !
length is: 8
```

In [34]:
```python
'''
A few more examples.
'''
random_string = 'thisis good !'
print('string is:', random_string)
print('length is:', len(random_string))
print('substring is:', random_string.lstrip('hti'))
print('length is:', len(random_string.lstrip('hti')))
```
```
string is: thisis good !
length is: 13
substring is: sis good !
length is: 10
```

In [35]:
```python
'''
A few more examples.
'''

random_string = 'thisis good '
print('string is:', random_string)
print('length is:', len(random_string))
print('substring is:', random_string.lstrip('hti '))
print('length is:', len(random_string.lstrip('hti ')))
```
```
string is: thisis good
length is: 12
substring is: sis good
length is: 9
```

`my_str.rstrip([chars])`. Removes trailing characters. The original string remains unchanged.

In [36]:
```python
random_string = '.  this is good'

# No trailing whitepsaces,so nothing is removed.
print(random_string.rstrip())

# Argument doesn't contain 'd'.  So no characters will be removed.
print(random_string.rstrip('oo'))

#Since the last three characters in the string are a permutation of the optional argument, they will be removed.
print(random_string.rstrip('doo'))

website = 'www.programiz.com/'
print(website.rstrip('/m'))
```
```
.  this is good
.  this is good
.  this is g
www.programiz.co
```

`my_str.strip()` removes both leading and trailing characters as specified by the argument.

`my_str.split([separator [, maxsplit]])`. The original string remains unchanged. The `split()` method separates a string into components delimited by the separator. The default value for the separator is a space. The individual components, separated by commas are returned as a list.

In [37]:
```python
text = 'Love thy neighbor'

# splits at space
print(text.split())
```
```
['Love', 'thy', 'neighbor']
```

In [38]:
```python
'''
    my_str.split([separator [, maxsplit]])
'''

grocery = 'Milke,Chicken,Bread'

# splits at ','
print(grocery.split(','))
```
```
['Milke', 'Chicken', 'Bread']
```

In [39]:
```python
'''
    my_str.split([separator [, maxsplit]])
'''

grocery = 'Milke,Chicken,Bread'

# splits at 'ke'
print(grocery.split('ke'))
```
```
['Mil', ',Chic', 'n,Bread']
```

In [40]:
```python
'''
    my_str.split([separator [, maxsplit]])
'''

grocery = 'Milk,Chicken,Bread'

# Splitting at ':'.  Since the string does not contain a :, no split will occur
print(grocery.split(':'))
```
```
['Milk,Chicken,Bread']
```

In [41]:
```python
'''
    my_str.split([separator [, maxsplit]])
'''

grocery = 'Milk:Chicken:Bread'

# Splitting at ':'.  Here the split will occur at each :
print(grocery.split(':'))
```
```
['Milk', 'Chicken', 'Bread']
```

In [42]:
```python
'''
    my_str.split([separator [, maxsplit]])
'''

grocery = 'Milk,Chicken,Bread,Eggs,Cookies'

#The optional argument indicates the maximum number of splits that can occur.  Since there are only 2 commas in the
 string
# both splits will occur.
print(grocery.split(',',3))
```
```
['Milk', 'Chicken', 'Bread', 'Eggs,Cookies']
```

In [43]:
```python
'''
    my_str.split([separator [, maxsplit]])
'''

grocery = 'Milk,Chicken,Bread'
#The optional argument indicates the maximum number of splits that can occur.  In this example only one split will o
ccur.
print(grocery.split(',', 1))
```
```
['Milk', 'Chicken,Bread']
```

`my_str.replace(old, new [, count])`. The original string remains unchanged. The `replace()` method returns a copy of the string where all occurrences of a substring are replaced with another substring.

The number of replacements will be less than or equal to the optional argument 'max'. If it is not provided, all occurences will be replaced

In [44]:
```python
song = 'cold, cold heart'
print (song.replace('cold', 'hurt')) # This will replace both occurences of 'cold' with 'hurt'
```
```
hurt, hurt heart
```

In [45]:
```python
song = 'cold, cold heart'
print(song.replace('cold', 'hurt', 3))
print(song.replace('cold', 'hurt', 2)) #This returns the same result as above
print(song.replace('cold', 'hurt'))
print(song.replace('cold', 'hurt', 1)) # This will only replace the first occurence of 'cold' with 'hurt'
```
```
hurt, hurt heart
hurt, hurt heart
hurt, hurt heart
hurt, cold heart
```

In [46]:
```python
song = 'Let it be, let it be, let it be, let it be'
print(song.lower().replace('let', "don't let", 2))
print(song)
```
```
don't let it be, don't let it be, let it be, let it be
Let it be, let it be, let it be, let it be
```

`startswith()` method returns True if a string starts with the specified prefix(string). If not, it returns False. The syntax is as follows:

`str.startswith(prefix[, start[, end]])`.

If the optional start and end parameters are provided, the method looks for the substring only within this range. The start value defaults to 0 and the end value defaults to the last index. The original string remains unchanged.

In [47]:
```python
text = "Python is easy to learn."

result = text.lower().startswith('python is ')
print('1:', result)

result = text.startswith('is easy')
print('2:', result)

result = text.startswith('is easy', 7)
print('3:', result)

result = text.startswith('Python is easy to learn.')
print('4:', result)

result = text.startswith('Python is easy to learn.', 0, 20) # Returns False because the end parameter is 20
print('5:', result)
print(len(text))
```
```
1: True
2: False
3: True
4: True
5: False
24
```