String Methods

find()

- It gives the index number in the string

In [1]:
```
help(str.find)
```
```
Help on method_descriptor:

find(...)
    S.find(sub[, start[, end]]) -> int

    Return the lowest index in S where substring sub is found,
    such that sub is contained within S[start:end].  Optional
    arguments start and end are interpreted as in slice notation.

    Return -1 on failure.
```

In [2]:
```
name = "I am data scientist"
```

In [3]:
```
name.find('a')
```
Out[3]:
```
2
```

In [9]:
```
name.find('a',7)
```
Out[9]:
```
8
```

rfind()

- It gives the right side index number in string

In [10]:
```
help(str.rfind)
```
```
Help on method_descriptor:

rfind(...)
    S.rfind(sub[, start[, end]]) -> int

    Return the highest index in S where substring sub is found,
    such that sub is contained within S[start:end].  Optional
    arguments start and end are interpreted as in slice notation.

    Return -1 on failure.
```

In [12]:
```
name = "I am data scientist"
```

In [13]:
```
name.rfind('a')
```
Out[13]:
```
8
```

In [17]:
```
name.rfind('a',2,4)
```
Out[17]:
```
2
```

count()

- It gives the frequency occurancy of a particular value

In [18]:
```
string = "Malayalam"
```
In [24]:
```
string.count('y')
```
Out[24]:
```
1
```
In [25]:
```
string.count('a')
```
Out[25]:
```
4
```

index()

- It gives the index number in the string

In [27]:
```
help(str.index)
```
```
Help on method_descriptor:

index(...)
    S.index(sub[, start[, end]]) -> int

    Return the lowest index in S where substring sub is found,
    such that sub is contained within S[start:end].  Optional
    arguments start and end are interpreted as in slice notation.

    Raises ValueError when the substring is not found.
```
In [28]:
```
name = "malayalam"
```
In [29]:
```
name.index('a')
```
Out[29]:
```
1
```
In [30]:
```
name.index('y')
```
Out[30]:
```
4
```
In [31]:
```
name.find('a'), name.index('a')
```
Out[31]:
```
(1, 1)
```
In [32]:
```
name.find('z')
name.index('z')          # uncomment to see the error
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[32], line 2
      1 name.find('z')
----> 2 name.index('z')

ValueError: substring not found
```

rindex()

- It gives the right side index number in the string

In [36]:
```
help(str.rindex)
```

```
Help on method_descriptor:

rindex(...)
    S.rindex(sub[, start[, end]]) -> int

    Return the highest index in S where substring sub is found,
    such that sub is contained within S[start:end].  Optional
    arguments start and end are interpreted as in slice notation.

    Raises ValueError when the substring is not found.
```

In [33]:
```
name = "malayalam"
```

In [34]:
```
name.rindex('a')
```

Out[34]:
```
7
```

replace()

- It replace the words

In [37]:
```
string = "I am python programmer"
```

In [40]:
```
string.replace("python","c")
```

Out[40]:
```
'I am c programmer'
```

In [41]:
```
m = "malayalam"
```

In [42]:
```
m.replace('m','M')
```

Out[42]:
```
'MalayalaM'
```

In [43]:
```
m.replace('a','A',2)
```

Out[43]:
```
'mAlAyalam'
```

split()

- It split the words

In [46]:
```
help(str.split)
```

```
Help on method_descriptor:

split(self, /, sep=None, maxsplit=-1)
    Return a list of the substrings in the string, using sep as the separator stri
ng.

      sep
        The separator used to split the string.

        When set to None (the default value), will split on any whitespace
        character (including \\n \\r \\t \\f and spaces) and will discard
        empty strings from the result.
      maxsplit
        Maximum number of splits (starting from the left).
        -1 (the default value) means no limit.

    Note, str.split() is mainly useful for data that has been intentionally
    delimited.  With natural text that includes punctuation, consider using
    the regular expression module.
```

In [44]:
```
string = "I am data scientist"
```

In [45]:
```
string.split()
```

Out[45]:
```
['I', 'am', 'data', 'scientist']
```

In [48]:
```
string.split(' ',1)
```

Out[48]:
```
['I', 'am data scientist']
```

In [50]:
```
string.split(' ',2)
```

Out[50]:
```
['I', 'am', 'data scientist']
```

rsplit()

- It split the right side words

In [52]:
```
help(str.rsplit)
```

```
Help on method_descriptor:

rsplit(self, /, sep=None, maxsplit=-1)
    Return a list of the substrings in the string, using sep as the separator stri
ng.

      sep
        The separator used to split the string.

        When set to None (the default value), will split on any whitespace
        character (including \\n \\r \\t \\f and spaces) and will discard
        empty strings from the result.
      maxsplit
        Maximum number of splits (starting from the left).
        -1 (the default value) means no limit.

    Splitting starts at the end of the string and works to the front.
```

In [51]:
```
string = "I am data scientist"
```

In [53]:
```
string.rsplit(' ',2)
```

Out[53]:
```
['I am', 'data', 'scientist']
```

In [55]:
```
string.split('am')
```

Out[55]:
```
['I ', ' data scientist']
```

partition()

- It divided into 3 partitions

In [61]:
```
help(str.partition)
```

```
Help on method_descriptor:

partition(self, sep, /)
    Partition the string into three parts using the given separator.

    This will search for the separator in the string.  If the separator is found,
    returns a 3-tuple containing the part before the separator, the separator
    itself, and the part after it.

    If the separator is not found, returns a 3-tuple containing the original strin
g
    and two empty strings.
```

In [58]:
```
m = "malayalam"
```

In [59]:
```
m.partition('a')
```

Out[59]:
```
('m', 'a', 'layalam')
```

In [60]:
```
m.partition('a'),m.partition('y')
```

Out[60]:
```
(('m', 'a', 'layalam'), ('mala', 'y', 'alam'))
```

In [62]:
```
name = "I am data scientist"
```

In [64]:
```
name.partition('m')
```

Out[64]:
```
('I a', 'm', ' data scientist')
```