Useful Python String Methods

June 30, 2022

0.1 Useful String Methods in Python

Learn about some of Python's built-in methods that can be used on strings

[1]:
```
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"
```

0.2 1. center( )

The `center()` method center aligns a string. The alignment is done using a specified character (whitespace is default)

0.2.1 Syntax

`str.center(length, fillchar), where :`

- **length** is the length of the string [*Required*]
- **fillchar** is the character which specifies the alignment [*Optional*]

[2]:
```
sentence = 'algorithm'
sentence.center(15,'*')
```

[2]:
```
'***algorithm***'
```

0.3 2. count( )

The `count()` method returns the count or the number of times a particular value appears in a string.

0.3.1 Syntax

`str.count(value, start, end), where :`

- **value** is the substring which is to be searched in the string [*Required*]
- **start** is the starting index within the string where the search for the specified value starts [*Optional*]
- **end** is the ending index within the string where the search for the specified value ends [*Optional*]

[3]:
```
sentence = 'She sells seashells by the seashore. The shells she sells are␣
↪surely seashells'
```

```
sentence.count('s')
sentence.count('z')
sentence.count('seashells',9,25)
```

[3]: 16

[3]: 0

[3]: 1

0.3.2 3. find( )

The `find()` method returns the lowest index of a particular substring in a string. If the substring is not found, -1 is returned.

0.3.3 Syntax

`str.find(value, start, end), where :`

- **value** or substring which is to be searched in the string [*Required*]
- **start** is the starting index within the string where the search for the specified value starts [*Optional*]
- **end** is the ending index within the string where the search for the specified value ends [*Optional*]

0.3.4 Types

rfind() : The rfind() method is similar to find() except that it returns the highest index value of the substring

[4]:
```
sentence = 'She sells seashells by the seashore. The shells she sells are␣
↪surely seashells'
sentence.find('seashells')
sentence.find('seashells',0,9)
sentence.find('s',5,10)
sentence.rfind('seashells')
```

[4]: 10

[4]: -1

[4]: 8

[4]: 69

0.4 4. swapcase( )

The `swapcase()` method returns a copy of the string with all its uppercase letters converted into lower case and vice versa.

0.4.1 Syntax

`string.swapcase()`

[5]:
```
sentence = 'Queue IS another FUNDAMENTAL data STRucture AND IS a close COUSIN␣
↪of the STACK'
sentence.swapcase()
```

[5]: `'qUEUE is ANOTHER fundamental DATA strUCTURE and is A CLOSE cousin OF THE stack'`

0.5 5. startswith( ) and endswith( )

The `startswith()` method returns True if the string starts with the specified value, otherwise it returns False. The `endswith()` function, on the other hand, returns True if the string endswith the specified value, else it returns False.

0.5.1 Syntax

```
string.startswith(value, start, end)
string.endsswith(value, start, end)
```

- **value** is the string to look for in the string *[Required]*
- **start** is the starting index within the string where the search for the specified value starts *[Optional]*
- **end** is the ending index within the string where the search for the specified value ends *[Optional]*

[6]:
```
#string.startswith()

sentence = 'Binary Search is a classic recursive algorithm'
sentence.startswith("Binary")
sentence.startswith("Search",7,20)
sentence.endswith('classic')
```

[6]: `True`

[6]: `True`

[6]: `False`

0.6 6. split( )

The `split()` method returns a list of words in a string where default separator is any whitespace.

0.6.1 Syntax

`string.split(sep, maxsplit)`

- **sep**: The separator to be used for splitting the string. if nothing is specified, whitespace is the default separator[Optional]
- **maxsplit**: denotes the number of splits. Default is -1 which means "all occurrences"[Optional]

- **Output Will be a list**

0.6.2 Version

rsplit(): splits a string from the right.

[7]:
```
#string.split()

fruits = 'apples, mangoes, bananas, grapes, papaya, oranges'
fruits.split()
```

[7]:
```
['apples,', 'mangoes,', 'bananas,', 'grapes,', 'papaya,', 'oranges']
```

[8]:
```
fruits.split(",")
```

[8]:
```
['apples', ' mangoes', ' bananas', ' grapes', ' papaya', ' oranges']
```

[9]:
```
spiltFruits = fruits.split(",",maxsplit = 4)
spiltFruits
```

[9]:
```
['apples', ' mangoes', ' bananas', ' grapes', ' papaya, oranges']
```

[10]:
```
#string.rsplit()
fruits.rsplit(",",maxsplit = 1)
```

[10]:
```
['apples, mangoes, bananas, grapes, papaya', ' oranges']
```

0.7 7.Join

The **join( )** string method returns a string by joining all the elements of an iterable (list, string, tuple), separated by a string separator.

[11]:
```
print(''.join(spiltFruits)) #" " is seperator here
```

```
apples mangoes bananas grapes papaya, oranges
```

0.8 8.String Capitalization

0.8.1 1. capitalize()

The **capitalize()** method capitalizes only the first character of the given string.

0.8.2 Syntax

```
string.capitalize()
```

[12]:
```
"san francisco".capitalize()
```

[12]:
```
'San francisco'
```

0.8.3 2. upper( ) & lower( )

The **upper()** method converts the string to uppercase whereas **lower()** converts to lowercase

0.8.4 Syntax

```
string.upper()
string.lower()
```

[13]:
```
"san francisco".upper()
```

[13]:
```
'SAN FRANCISCO'
```

[14]:
```
'SAN FRANCISCO'.lower()
```

[14]:
```
'san francisco'
```

0.8.5 3. string.title( )

The **title()** method capitalizes all the first letters of the string.

0.8.6 Syntax

```
string.title()
```

[15]:
```
"san francisco".title()
```

[15]:
```
'San Francisco'
```

0.9 9. ljust( ) and rjust( )

The **ljust()** method returns a left-justified version of the given string using a specified character, whitespace being default. The **rjust()** methods aligns the string to the right.

0.9.1 Syntax

```
string.rjust/ljust(length, character)
```

- **length**: length of the string which is to be returned *[Required]*
- **character**: Character used for filling in the missing space where whitespace is default *[Optional]*

[16]:
```
#str.rjust
text = 'Binary Search'
print(text.rjust(25,"*"),"is a classic recursive algorithm")
```

```
**********Binary Search is a classic recursive algorithm
```

[17]:
```
#str.ljust
text = 'Binary Search'
print(text.ljust(25),"is a classic recursive algorithm")
```

```
Binary Search             is a classic recursive algorithm
```

0.10 10. strip( )

The **strip()** method returns a copy of the string with the leading and trailing characters removed. Default character to be removed is whitespace.

0.10.1 Syntax

```
string.strip(character)
```

- **character**: set of characters to be removed [**Optional**]

0.10.2 Versions

```
rstrip(): strips characters from the right of a string.
lstrip(): strips characters from the left of a string.
```

[18]:
```
#str.strip
string = '#.......Section 3.2.1 Issue #32......'
string.strip('.#!')
```

[18]:
```
'Section 3.2.1 Issue #32'
```

[19]:
```
#str.rstrip
string.rstrip('.#!')
string.lstrip('.#!')
```

[19]:
```
'#…Section 3.2.1 Issue #32'
```

[19]:
```
'Section 3.2.1 Issue #32…'
```

0.11 11. zfill( )

The **zfill()** method adds zeros(0) at the beginning of the string. The length of the returned string depends on the width provided.

0.11.1 Syntax

```
string.zfill(width)
```

width: specifies the length of the returned string. However, no zeros are added if the width parameter is less than the length of the original string.

[20]:
```
'7'.zfill(3)
'796'.zfill(3)
'-21'.zfill(5)
'Python'.zfill(10)
'Python'.zfill(3)
```

[20]:
```
'007'
```

[20]: '796'

[20]: '-0021'

[20]: '0000Python'

[20]: 'Python'