# Python Bootcamp & Masterclass

# string methods 2

gknxt

# .find( )

The `.find(sub,start,end)` method returns the lowest index in the string where the substring (`sub`) is found starting at `start` (optional) up to `end` (optional). Returns `-1` if `sub` is not found (`start` and `end` behave the same way as they do in slicing)

The `.find( )` works only with strings, not with other sequences.

```
k = 'It was the best of times, it was the worst of times'   # 51 character string
k.find('was')           # sub = 'was' (present twice: at 3 and 29 positions in the string)
k.find('its')           # sub = 'its' (not present in the string)
k.find('was', 4)        # sub = 'was', start = 4
k.find('time', 10, 30)  # sub = 'time', start = 10, end = 30
k.find('st', 300)       # sub = 'st', start = 300 (start is out of range)
k.find('st', 25, 300)   # sub = 'st', start = 25, end = 300 (end is out of range)
k.find('st', 100, 300)  # sub = 'st', start = 100, end = 300 (start and end are out of range)
```

```
3

-1

29

19

-1

40

-1
```

# .rfind( )

The `.rfind(sub,start,end)` method returns the highest index in the string where the substring (`sub`) is found starting at `start` (optional) up to `end` (optional). Returns `-1` if `sub` is not found (`start` and `end` behave the same way as they do in slicing)

The `.find( )` works only with strings, not with other sequences.

```
k = 'It was the best of times, it was the worst of times'   # 51 character string
k.rfind('was')           # sub = 'was' (present twice: at 3 and 29 positions in the string)
k.rfind('its')           # sub = 'its' (not present in the string)
k.rfind('was', 2, 20)    # sub = 'was', start = 2, end = 20
k.rfind('time', 10, 30)  # sub = 'time', start = 10, end = 30
k.rfind('st', 300)       # sub = 'st', start = 300 (start is out of range)
k.rfind('st', 25, 300)   # sub = 'st', start = 25, end = 300 (end is out of range)
k.rfind('st', 100, 300)  # sub = 'st', start = 100, end = 300 (start and end are out of range)
```

```
29

-1

3

19

-1

40

-1
```

# .count( )

The `.count(sub,start,end)` method returns the number of non-overlapping occurrences of substring `sub` (`start` and `end` behave the same way as they do in slicing)

If multiple counting for a single character is needed, Counter from collections module could be a better option

```
my_str = "It was the best of times, it was the worst of times"
my_str.count('t')
my_str.count('t', 6)
my_str.count('t', 6, 30)
my_str.count('time', 6, 300)
```

```
8

7

4

2
```

```
from collections import Counter
my_str = "It was the best of times, it was the worst of times"
counter = Counter(my_str)
print (counter['m'])
print(counter)
```

```
2
Counter({' ': 11, 't': 8, 's': 6, 'e': 5, 'w': 3, 'o': 3, 'i': 3, 'a': 2, 'h': 2, 'f': 2, 'm': 2, 'I': 1, 'b': 1, ',': 1,
'r': 1})
```

# .startswith( )

The `.startswith(x,start,end)` method returns `True` if the string starts with str `x` or with any of the strings in tuple `x`; otherwise, returns `False`.

```
ws = "https://gknxt.com"
ws.startswith('https:')
ws.startswith(('https:', 'http:', 'www.'))
```

```
True

True
```

# .endswith( )

The `.endswith(x,start,end)` method returns `True` if the string ends with str `x` or with any of the strings in tuple `x`; otherwise, returns `False`.

```
img = "gknxt.png"
img.endswith('png')
img.endswith(('png', 'jpeg', 'jpg', 'JPEG', 'JPG'))
```

```
True

True
```

# .reversed( )

The **`reversed(i)`** function returns an iterator that returns the items from iterator **`i`** in reverse order. The iterator yields characters directly from the original string (doesn't create a new reversed string), so it is efficient in terms of memory usage and speed.

```
greet = reversed("Hello")
print(greet)
next(greet)
next(greet)
next(greet)
next(greet)
next(greet)
next(greet)                  # StopIteration Error if the iterator is accessed out of its range
```

```
<reversed object at 0x000001FFC26195E0>
'o'
'l'
'l'
'e'
'H'
---------------------------------------------------------------------------
StopIteration                             Traceback (most recent call last)
<ipython-input-19-46834bc95541> in <module>
      6 next(greet)
      7 next(greet)
----> 8 next(greet)                  # StopIteration Error if the iterator is accessed out of its range

StopIteration:
```

```
for _ in reversed("Hello"):
    print(_)
```

```
o
l
l
e
H
```

gk nxt

# .join( )

The `.join(seq)` method returns the concatenation of every item in the sequence `seq`, with string (which can be empty) between each one. A `TypeError` will be raised if there are any non-string values in `seq`. To concatenate a sequence of strings, `.join(seq)` is the preferred and faster way than string concatenation using `+` operator.

```
'z'.join(['a','b', 'c'])
','.join(('a', 'z'))
''.join(['a'])
''.join(('x','y','z'))*3
'#'.join(('x','y','z'))*3
'*'.join(('x','y','z')*2)
```

```
'azbzc'

'a,z'

'a'

'xyzxyzxyz'

'x#y#zx#y#zx#y#z'

'x*y*z*x*y*z'
```

```
''.join(('a', 2, 'z'))
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-61-57ae3d4de8a8> in <module>
----> 1 ''.join(('a', 2, 'z'))

TypeError: sequence item 1: expected str instance, int found
```

```
greeting = "Hello!"
"".join(reversed(greeting))
```

```
'!olleH'
```

# .split( )

The `.split(sep=None, maxsplit=- 1)` method returns a list of the words in the string, using `sep` as the delimiter. If `maxsplit` is given, at most `maxsplit` splits are done (the resulting list will have at most `maxsplit+1` elements). If `maxsplit` is not specified or -1, then all possible splits are made.

```
'a,b,c'.split()
'a,b,c'.split(',')
'a,b,c'.split(',', 1)
'1,2,,3,'.split(',')
'1 2 3'.split()
'  1 2   3    '.split()
```

```
['a,b,c']

['a', 'b', 'c']

['a', 'b,c']

['1', '2', '', '3', '']

['1', '2', '3']

['1', '2', '3']
```

```
record = "Mahatma Gandhi*1869-10-02*1948-01-30"
record.split()
record.split('*')
record.replace(' ','*').split('*')  # re is more suitable for multi-char splits
```

```
['Mahatma', 'Gandhi*1869-10-02*1948-01-30']

['Mahatma Gandhi', '1869-10-02', '1948-01-30']

['Mahatma', 'Gandhi', '1869-10-02', '1948-01-30']
```

```
record = "Mahatma Gandhi*1869-10-02*1948-01-30"
print(record.split("*")[0] + " lived about " +
    str(int(record.split("*")[2].split("-")[0]) -
    int(record.split("*")[1].split("-")[0])) + " years")
```

```
Mahatma Gandhi lived about 79 years
```

# .rsplit( )

The `.rsplit(sep=None, maxsplit=- 1)` method returns a list of the words in the string, using `sep` as the delimiter. If `maxsplit` is given, at most `maxsplit` rightmost splits are done. If `maxsplit` is not specified or -1, then all possible splits are made.

```
'a,b,c'.rsplit()
'a,b,c'.rsplit(',')
'a,b,c'.rsplit(',', 1)
'1,2,,3,'.rsplit(',')
'1 2 3'.rsplit()
'  1 2   3    '.rsplit()
```

```
['a,b,c']

['a', 'b', 'c']

['a,b', 'c']

['1', '2', '', '3', '']

['1', '2', '3']

['1', '2', '3']
```

```
record = "Mahatma Gandhi*1869-10-02*1948-01-30"
record.rsplit()
record.rsplit('*')
record.rsplit('*', 2)
record.replace(' ','*').rsplit('*')  # re is more suitable for multi-char splits
```

```
['Mahatma', 'Gandhi*1869-10-02*1948-01-30']

['Mahatma Gandhi', '1869-10-02', '1948-01-30']

['Mahatma Gandhi', '1869-10-02', '1948-01-30']

['Mahatma', 'Gandhi', '1869-10-02', '1948-01-30']
```

# .splitlines( )

The `.splitlines(keepends=False)` method returns a list of the lines, breaking at line boundaries. Line breaks are not included in the resulting list unless **keepends** is set to **True**. Windows uses **carriage return + line feed** for newline ('\r\n') and UNIX uses **line feed** for newline ('\n') Mac's newline depends on the version of Mac OS

```
s = 'ab c\n\nde fg\r\nkl\r\n'
print(s)     # ab c on line1, two blank lines, de fg line 4, kl on line 5
s.splitlines()
s.splitlines(True)
```

```
ab c

de fg
kl


['ab c', '', 'de fg', 'kl']

['ab c\n', '\n', 'de fg\r\n', 'kl\r\n']
```

```
t = ''
t.splitlines()
```

```
[]
```

```
u = 'One line\n'
v = 'One line\nAnotherline'
print(u)                    # prints One line and then a blank line below
u.splitlines()
u.splitlines(True)
v.splitlines()
v.splitlines(True)
```

```
One line


['One line']

['One line\n']

['One line', 'Anotherline']

['One line\n', 'Anotherline']
```


# Online Resources

For best python resources, please visit:

gknxt.com/python/

gknxt

# Python Bootcamp & Masterclass

# Thank You

for your Rating & Review

gknxt