

analysis methods

count(): returns the number of times a specified set of characters is repeated.

```
"Hello world".count("Hello")
>> 1
```

find() e **index()** return the **location** (starting at 0) where the given argument is found. They differ in that index raises **ValueError** when the argument is not found, while find returns -1.

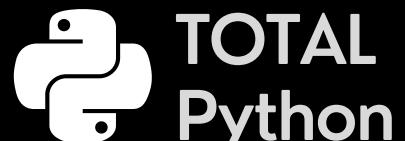
```
"Hello world".find("world")
>> -1
```

rfind() & rindex() To search for a set of characters starting from the end.

```
"C:/python36/python.exe".rfind("/")
>> 11
```

startswith() & **endswith()** indicate whether the string in question begins or ends with the set of characters passed as an argument, and return True or False accordingly.

```
"Hello world".startswith("Hello")
>> True
```



analysis methods

isdigit(): returns True if all the characters in the string are digits, or can form numbers, including those corresponding to oriental languages.

```
"abc123".isdigit()
>> False
```

isnumeric(): returns True if all characters in the string are numbers, it also includes characters with numeric connotation that are not necessarily digits (for example, a fraction).

```
"1234".isnumeric()
>> True
```

isdecimal(): returns True if all characters in the string are decimals, that is, formed by digits from 0 to 9.

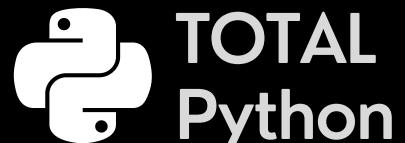
```
"1234".isdecimal()
>> True
```

isalnum(): returns True all characters in the string are alphanumeric.

```
"abc123".isalnum()
>> True
```

isalpha(): returns True if all characters in the string are alphabetic.

```
"abc123".isalpha()
>> False
```



analysis methods

islower(): returns True if all characters in the string are lowercase.

```
"abcdef".islower()
>> True
```

isupper(): returns True if all characters in the string are uppercase.

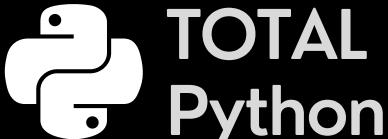
```
"ABCDEF".isupper()
>> True
```

isprintable(): returns True if all characters in the string are printable (that is, not special characters indicated by \...).

```
"Hello \t world!".isprintable()
>> False
```

isspace(): returns True if all characters in the string are spaces.

```
"Hello world".isspace()
>> False
```



transformation methods

Strings are actually immutable objects, so in reality all the methods below do not act on the original object but return a new one.

capitalize() returns the string with its first letter capitalized.

```
"hello world".capitalize()
>> 'Hello world'
```

encode() encodes the string with the specified character map and returns an instance of type bytes.

```
"Hello world".encode("utf-8")
>> b'Hello world'
```

replace() replaces one string with another.

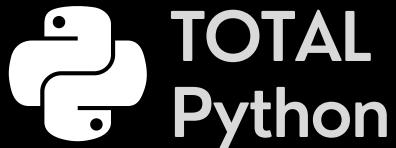
```
"Hello world".replace("world", "everyone")
>> 'Hello everyone'
```

lower() returns a copy of the string with all its letters in lowercase.

```
"Hello World".lower()
>> 'hello world'
```

upper() returns a copy of the string with all its letters in uppercase.

```
"Hello world".upper()
>> 'HELLO WORLD'
```



transformation methods

swapcase() change uppercase to lowercase and vice versa.

```
"Hello World".swapcase()
>> 'hELLO wORLD'
```

strip(), Istrip() & rstrip() remove whitespaces that precede and/or follow the string.

```
" Hello world ".strip()
>> 'Hello world'
```

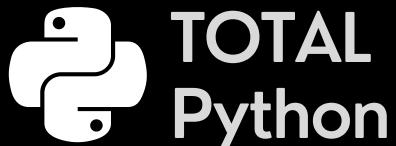
center(), ljust() & rjust() align a string to the center, left, or right. A second argument indicates with which character to fill the empty spaces (by default: blank space).

```
"hello".center(9, "*")
>> '**hello**'
```

splitting and joining methods

split() splits a string based on a separator character (defaults: blanks). A second argument indicates the maximum number of splits that can take place (-1 by default, meaning an unlimited number of splits).

```
"Hello world!\nHello everyone!".split()
>> ['Hello', 'world!', 'Hello', 'everyone!']
```



strings: methods Python

splitting and joining methods

splitlines() splits a string with each occurrence of a line break.

```
"Hello world!\nHello everyone!".splitlines()
>> ['Hello world!', 'Hello everyone!']
```

partition() returns a tuple of three elements: the block of characters before the first occurrence of the separator, the separator itself, and the block after.

```
"Hello world! Hello everyone!".partition(" ")
>> ('Hello', ' ', 'world! Hello everyone!')
```

rpartition() operates in the same way as the previous one, but starting from right to left.

```
"Hello world! Hello everyone!".rpartition(" ")
>> ('Hello world! Hello', ' ', 'everyone!')
```

join() must be called from a string that acts as a separator to join the elements of a list into the same resulting string.

```
", ".join(["C", "C++", "Python", "Java"])
>> 'C, C++, Python, Java'
```