str.rstrip([*chars*])
Return a copy of the string with trailing characters removed.
The *chars* argument is a string specifying the set of characters to be removed. If omitted or None, the *chars* argument defaults to removing whitespace. The *chars* argument is not a suffix; rather, all combinations of its values are stripped:

```
>>> s1="nit       "
>>> s1=="nit"
False
>>> len(s1)
10
>>> s1.rstrip()=="nit"
True
>>> s2="nit****"
>>> s3=s2.rstrip("*")
>>> print(s2,s3,sep="\n")
nit****
nit
>>> s4="nit$$$***@@"
>>> s5=s4.rstrip("$*@")
>>> print(s4,s5,sep="\n")
nit$$$***@@
nit
```

str.lstrip([*chars*])
Return a copy of the string with leading characters removed.
The *chars* argument is a string specifying the set of characters to be removed. If omitted or None, the *chars* argument defaults to removing whitespace. The *chars* argument is not a prefix; rather, all combinations of its values are stripped:

```
>>> str1="  nit"
>>> str1=="nit"
```

```
False
>>> str1.lstrip()=="nit"
True
>>> str2=str1.lstrip()
>>> print(str2)
nit
>>> print(str1)
  nit
>>> str3="*****nit"
>>> str4=str3.lstrip("*")
>>> print(str3)
*****nit
>>> print(str4)
nit
>>> str5="$$$##**$$nit"
>>> str6=str5.lstrip("$#*")
>>> print(str5,str6,sep="\n")
$$$##**$$nit
nit
```

Partition methods

str.partition(*sep*)

Split the string at the first occurrence of *sep*, and return a 3-tuple containing the part before the separator, the separator itself, and the part after the separator. If the separator is not found, return a 3-tuple containing the string itself, followed by two empty strings.

```
>>> s1="a,b,c,d,e"
>>> t1=s1.partition(",")
>>> print(s1)
a,b,c,d,e
```

```
>>> print(t1)
('a', ',', 'b,c,d,e')
>>> t2=s1.partition(".")
>>> print(t2)
('a,b,c,d,e', '', '')
```

str.rpartition(*sep*)
Split the string at the last occurrence of *sep*, and return a 3-tuple containing the part before the separator, the separator itself, and the part after the separator. If the separator is not found, return a 3-tuple containing two empty strings, followed by the string itself.

```
>>> s1="a,b,c,d,e"
>>> t1=s1.rpartition(",")
>>> print(s1,t1,sep="\n")
a,b,c,d,e
('a,b,c,d', ',', 'e')
```

str.join(*iterable*)
Return a string which is the concatenation of the strings in *iterable*.

```
>>> list1=["a","b","c","d","e"]
>>> str1=':'.join(list1)
>>> print(list1)
['a', 'b', 'c', 'd', 'e']
>>> print(str1)
a:b:c:d:e
>>> str2=','.join(list1)
>>> print(str2)
a,b,c,d,e
>>> str3='*'.join(list1)
```

```
>>> print(str3)
a*b*c*d*e
>>> str4="\t".join(list1)
>>> print(str4)
a	b	c	d	e
>>>
>>> s1="abcde"
>>> s2=",".join(s1)
>>> print(s2)
a,b,c,d,e
```

You are given a string. Split the string on a " " (space) delimiter and join using a - hyphen.

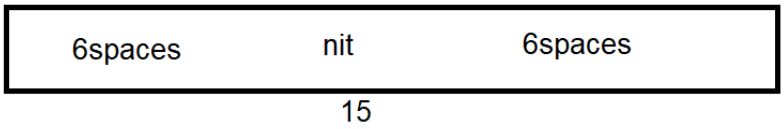
```
>>> s1="this is a string"
>>> list1=s1.split(" ")
>>> print(list1)
['this', 'is', 'a', 'string']
>>> s2="-".join(list1)
>>> print(s2)
this-is-a-string
```

Alignment methods or justification methods

str.center(*width*[, *fillchar*])

Return centered in a string of length *width*. Padding is done using the specified *fillchar* (default is an ASCII space). The original string is returned if *width* is less than or equal to len(s).

```
str1="nit"
print(str1.center(15))
```



Example:

```
str1="nit"
print(str1.center(15))
print(str1.center(15,"*"))
names=["naresh","ramesh","kishore","rajesh","kiran","raman"]
for name in names:
    print(name.center(12,"*"))
```

Output:

```
    nit
******nit******
***naresh***
***ramesh***
**kishore***
***rajesh***
***kiran****
***raman****
```

str.ljust(*width*[, *fillchar*])

Return the string left justified in a string of length *width*. Padding is done using the specified *fillchar* (default is an ASCII space). The original string is returned if *width* is less than or equal to len(s).

```
>>> str1="nit"
```

```
>>> print(str1.ljust(10))
nit
>>> print(str1.ljust(10,"*"))
nit*******
```

str.rjust(*width*[, *fillchar*])

Return the string right justified in a string of length *width*. Padding is done using the specified *fillchar* (default is an ASCII space). The original string is returned if *width* is less than or equal to len(s).

```
>>> str1="nit"
>>> print(str1.rjust(10))
       Nit
>>> print(str1.rjust(10,"*"))
*******nit
```

Finding and replacing methods

```
"prog" in "python is programming language"
   True                                   Result of "in" operator boolean value (True/False)
"java" in "python oracle .net"
   False
```

str.find(*sub*[, *start*[, *end*]])

Return the lowest index in the string where substring *sub* is found within the slice s[start:end]. Optional arguments *start* and *end* are interpreted as in slice notation. Return -1 if *sub* is not found.

```
>>> str1="python java oracle java python"
>>> index=str1.find("java")
>>> print(index)
```

```
7
>>> index=str1.find("python")
>>> print(index)
0
>>> index=str1.find("mysql")
>>> print(index)
-1
>>> index=str1.find("java",10)
>>> print(index)
19
>>> index=str1.find("java",10,15)
>>> print(index)
-1
```

str.rfind(*sub*[, *start*[, *end*]])

Return the highest index in the string 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.

```
>>> str1="python java oracle java python"
>>> index=str1.rfind("java")
>>> print(index)
19
>>> index=str1.rfind("java",0,12)
>>> print(index)
7
>>> index=str1.rfind("mysql")
>>> print(index)
-1
```

str.replace(*old*, *new*[, *count*])

Return a copy of the string with all occurrences of substring *old* replaced by *new*. If the optional argument *count* is given, only the first *count* occurrences are replaced.

```
>>> str1="jython python jython java rpython java"
>>> str2=str1.replace("java","python")
>>> print(str1)
jython python jython java rpython java
>>> print(str2)
jython python jython python rpython python
>>> str3=str1.replace("java","python",1)
>>> print(str3)
jython python jython python rpython java
>>> str4=str1.replace("oracle","mysql")
>>> print(str4)
jython python jython java rpython java
```

String conversion methods

str.maketrans(x[, y[, z]])
This method returns a translation table usable for str.translate().

str.translate(*table*)
Return a copy of the string in which each character has been mapped through the given translation table.