str.istitle()
Return True if the string is a titlecased string and there is at least one character, for example uppercase characters may only follow uncased characters and lowercase characters only cased ones. Return False otherwise.

str.isupper()
Return True if all cased characters in the string are uppercase and there is at least one cased character, False otherwise

```
>>> str1="Python Language"
>>> str1.istitle()
True
>>> str2="Python language"
>>> str2.istitle()
False
>>> str3="PYTHON"
>>> str3.isupper()
True
>>> str4="python"
>>> str4.isupper()
False
```

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

```
>>> l1=['A','B','C','D']
>>> str1=",".join(l1)
>>> print(str1)
A,B,C,D
>>> l2=["python","programming","language"]
>>> str3=" ".join(l2)
>>> print(str3)
python programming language
```

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).

```
>>> stud_dict={'naresh':'python',
          'kishore':'c++',
          'ramesh':'c',
          'kiran':'oracle'}
>>> for name,course in stud_dict.items():
    print(name.ljust(15),course.ljust(15))



naresh          python
kishore         c++
ramesh          c
kiran           oracle
>>> for name,course in stud_dict.items():
...     print(name.ljust(15,'*'),course.ljust(15,"$"))
...
...
naresh********* python$$$$$$$$$
kishore******** c++$$$$$$$$$$$$
ramesh********* c$$$$$$$$$$$$$$
kiran********** oracle$$$$$$$$$
```

str.lower()
Return a copy of the string with all the cased characters converted to lowercase.

```
>>> str1="PYTHON"
>>> str2="python"
>>> str1==str2
False
>>> str1.lower()==str2.lower()
True
>>> str3=str1.lower()
>>> print(str3)
python
```

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"
>>> str2="    nit"
>>> str1==str2
False
>>> str3=str2.lstrip()
>>> print(str1)
nit
>>> print(str2)
    nit
>>> print(str3)
nit
>>> str1==str3
True
>>> str1==str2.lstrip()
True
>>> s1="******nit"
>>> s2=s1.lstrip("*")
>>> print(s1)
******nit
>>> print(s2)
nit
>>> s3="**#$**$@$$nit"
>>> s4=s3.lstrip("$@#*")
>>> print(s3)
**#$**$@$$nit
>>> print(s4)
nit
```

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

```
>>> t1=str.maketrans("aeiou","!@#$%")
>>> t2=str.maketrans("!@#$%","aeiou")
>>> str1="programming"
>>> str2=str1.translate(t1)
>>> print(str1)
programming
>>> print(str2)
pr$gr!mm#ng
>>> str3=str2.translate(t2)
>>> print(str3)
programming
t1=str.maketrans("abcdefghijklmnopqrstuvwxyz","!@#$%^&*()_+{}|:'<>?123
456")
str1="python"
str2=str1.translate(t1)
print(str1)
python
>>> print(str2)
:5?*|}
>>>
t2=str.maketrans("!@#$%^&*()_+{}|:'<>?123456","abcdefghijklmnopqrstuv
wxyz")
>>> str3=str2.translate(t2)
>>> print(str3)
python
```

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.

```
>>> str1="a,b,c,d,e"
t=str1.partition(",")
print(t)
('a', ',', 'b,c,d,e')
name="rama rao"
>>> t=name.partition(" ")
>>> print(t)
('rama', ' ', 'rao')
```

```
>>> str2="a,b,c"
>>> t=str2.partition(":")
>>> t
('a,b,c', '', '')
```

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="python java oracle java .net oracle python"
>>> str2=str1.replace("java","python")
>>> print(str1)
python java oracle java .net oracle python
>>> print(str2)
python python oracle python .net oracle python
>>> str3=str1.replace("java","python",1)
>>> print(str3)
python python oracle java .net oracle python
```

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.

```
>>> s1="python java python java python java"
>>> i=s1.find("java")
>>> print(i)
7
>>> i=s1.rfind("java")
>>> print(i)
31
>>> i=s1.rfind("java",20,30)
>>> print(i)
-1
```

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).

```
>>> emp_dict={'naresh':5000.0,
         'suresh':6000.0,
         'kishore':7000.0}
>>> for name,sal in emp_dict.items():
    print(name.ljust(15),str(sal).rjust(10))


naresh             5000.0
suresh             6000.0
kishore            7000.0
>>> for name,sal in emp_dict.items():
...     print(name.ljust(15,"*"),str(sal).rjust(10,"&"))
...
...
naresh********* &&&&5000.0
suresh********* &&&&6000.0
kishore******** &&&&7000.0
```

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.

```
>> str1="a,b,c,d,e"
>>> t=str1.partition(",")
>>> print(t)
('a', ',', 'b,c,d,e')
>>> t=str1.partition(",")
>>> print(t)
('a', ',', 'b,c,d,e')
```

```
>>> t=str1.rpartition(",")
>>> print(t)
('a,b,c,d', ',', 'e')
```

Example:

```
str1="python programming language"
l=str1.split()
>>> l1=[s[::-1] for s in l]
>>> print(l)
['python', 'programming', 'language']
>>> print(l1)
['nohtyp', 'gnimmargorp', 'egaugnal']
>>> str2=" ".join(l1)
>>> print(str2)
nohtyp gnimmargorp egaugnal
>>>
```

str.rsplit(sep=None, maxsplit=- 1)
Return a list of the words in the string, using sep as the delimiter string. If maxsplit is given, at most maxsplit splits are done

str.split(sep=None, maxsplit=- 1)
Return a list of the words in the string, using sep as the delimiter string. If maxsplit is given, at most maxsplit splits are done