Strip methods or trim methods

These methods are used to remove leading or trailing characters from string.

1. lstrip()
2. rstrip()
3. strip()

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:

Example:

```
str1="      abc"
print(str1,len(str1))
str2=str1.lstrip()
print(str2,len(str2))
str3="******abc"
str4=str3.lstrip("*")
print(str3,str4,sep="\n")
str5="**$$###**@@$$abc"
str6=str5.lstrip("$#*@")
print(str5,str6,sep="\n")

# Application Development

user=input("UserName :")
pwd=input("Password :")
if user.lstrip()=="nit" and pwd=="n123":
   print(f'{user} welcome')
else:
   print("invalid username or password")
```

Output

```
      abc 10
```

abc 3
******abc
abc
**$$###**@@$$abc
abc
UserName : nit
Password :n123
  nit welcome

Example:

```
# Write a program to remove leading spaces from string without using predefined method

str1=input("Enter any string ")
str2=''
i=0
leading=input("Enter leading character to remove ")
for ch in str1:
    if ch in leading:
        i=i+1
    else:
        break

while i<len(str1):
    str2=str2+str1[i]
    i=i+1

print(str1)
print(str2)
```

Output

```
Enter any string **$$##@@nit
Enter leading character to remove *#@$
**$$##@@nit
nit
```

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:

Example:

```
str1="abc           "
str2=str1.rstrip()
print(str1,str2,sep="\n")
print(len(str1),len(str2),sep="\n")

str3="**a*bc**********"
str4=str3.rstrip("*")
print(str3,str4,sep="\n")

str5="abc**$$##@@"
str6=str5.rstrip("*$#@")
print(str5,str6,sep="\n")

str1=input("Enter Any String ")
tr=input("Enter string to remove ")
str2=''
c=0
for i in range(-1,-(len(str1)+1),-1):
    if str1[i] in tr:
        c=c+1
    else:
        break

for i in range(0,len(str1)-c):
    str2=str2+str1[i]

print(str1)
print(str2)
```

Output

```
abc
abc
15
3
**a*bc**********
**a*bc
abc**$$##@@
abc
Enter Any String abc$$%%^^
Enter string to remove %$^
abc$$%%^^
abc
```

str.strip([*chars*])

Return a copy of the string with the leading and 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 prefix or suffix; rather, all combinations of its values are stripped:

Example:

```
str1="      abc        "
str2=str1.strip()
print(str1,str2,sep="\n")
print(len(str1),len(str2),sep="\n")

str3="******nit*****"
str4=str3.strip("*")
print(str3,str4,sep="\n")

str5="**$$nit&&**$$"
str6=str5.strip("*$&")
print(str5,str6,sep="\n")

str1=input("Enter any String ")
r=input("Enter characters to remove ")
```

```
c1=0
for i in range(0,len(str1)):
    if str1[i] in r:
        c1=c1+1
    else:
        break
c2=0
for i in range(-1,-(len(str1)+1),-1):
    if str1[i] in r:
        c2=c2+1
    else:
        break

str2=str1[c1:-c2]
print(str1)
print(str2)
```

Output

```
      abc
abc
19
3
******nit*****
nit
**$$nit&&**$$
nit
Enter any String www.nareshit.com
Enter characters to remove w.com
www.nareshit.com
nareshit
```

Filtering methods or searching methods

1. startswith()
2. endswith()

These methods returns Boolean value (True/False)

str.startswith(*prefix*[, *start*[, *end*]])
Return True if string starts with the *prefix*, otherwise
return False. *prefix* can also be a tuple of prefixes to look for. With
optional *start*, test string beginning at that position. With
optional *end*, stop comparing string at that position.

Example

```
name="Mr. Ramesh"
if name.startswith('Mr'):
    print(f'{name} starts with Mr')
else:
    print(f'{name} not starts with Mr')


names_list=['naresh','ramesh','kishore','rajesh','kiran']
for name in names_list:
    if name.startswith('r'):
        print(name)

for name in names_list:
    if name.startswith(('n','k')):
        print(name)

for name in names_list:
    if name[0]=='r':
        print(name)

for name in names_list:
    if name[0] in "nk":
        print(name)
```

Output

```
Mr. Ramesh starts with Mr
ramesh
rajesh
naresh
```

kishore
kiran
ramesh
rajesh
naresh
kishore
kiran

str.endswith(*suffix*[, *start*[, *end*]])
Return True if the string ends with the specified *suffix*, otherwise return False. *suffix* can also be a tuple of suffixes to look for. With optional *start*, test beginning at that position. With optional *end*, stop comparing at that position.

Example:

```
names_list=['naresh','ramesh','kishore','rajesh','kiran']
for name in names_list:
    if name.endswith('h'):
        print(name)

print("-"*40)
for name in names_list:
    if name.endswith(('e','n')):
        print(name)

print("-"*40)
for name in names_list:
    if name[-1]=='h':
        print(name)

print("-"*40)
for name in names_list:
    if name[0] in "rk" and name[-1] in 'hn':
        print(name)
```

Output

```
naresh
```

ramesh
rajesh
---
kishore
kiran
---
naresh
ramesh
rajesh
---
ramesh
rajesh
kiran

partition methods

1. partition
2. rpartition

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.

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.

Example:

```
>>> str1="a,b,c,d,e"
>>> t1=str1.partition(",")
>>> print(str1)
```

a,b,c,d,e
>>> print(t1)
('a', ',', 'b,c,d,e')
>>> f1="1.456"
>>> t=f1.partition(".")
>>> print(t)
('1', '.', '456')
>>> str1="a,b,c,d,e"
>>> t=str1.rpartition(",")
>>> print(t)
('a,b,c,d', ',', 'e')

Example:

```
# input
str1="java"

# output aajv
list1=sorted(str1)
print(list1)
str2=''.join(list1)
print(str2)
```

Output

```
['a', 'a', 'j', 'v']
Aajv
```

Buffered data types or binary data type/collections

1. bytes → immutable sequence data type
2. bytearray → mutable sequence data type

bytes

Bytes objects are immutable sequences of single bytes. Since many major binary protocols are based on the ASCII text encoding, bytes objects offer several methods that are only valid when working with ASCII compatible data and are closely related to string objects in a variety of other ways.

Firstly, the syntax for bytes literals is largely the same as that for string literals, except that a b prefix is added:

- Single quotes: b'still allows embedded "double" quotes'
- Double quotes: b"still allows embedded 'single' quotes"
- Triple quoted: b'''3 single quotes''', b"""3 double quotes"""

Only ASCII characters are permitted in bytes literals (regardless of the declared source code encoding). Any binary values over 127 must be entered into bytes literals using the appropriate escape sequence.