Alignment methods

1. ljust
2. rjust
3. center

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).

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).

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).

Example:

```
>>> str1="nit"
>>> str1.ljust(15)
'nit            '
>>> str1.ljust(15,'*')
'nit************'
>>> str1.ljust(15,'$')
'nit$$$$$$$$$$$$'
>>> str1.rjust(15)
'            nit'
>>> str1.ljust(15,'*')
'nit************'
>>> str1.rjust(15,'*')
'************nit'
>>> str1.center(15)
'      nit      '
>>> str1.center(15,'*')
'******nit******'
```

Example:

```
namesList=['naresh','ramesh','kishore','suresh','kiran']
for name in namesList:
    print(name.center(20,'*'))


studList=[['naresh','python'],
          ['kishore','java'],
          ['ramesh','oracle'],
          ['kiran','mysql']]

for stud in studList:
    name,course=stud
    print(name.ljust(15,'#'),course.center(15,'*'))
```

Output

```
*******naresh*******
*******ramesh*******
******kishore*******
*******suresh*******
*******kiran********
naresh########## *****python****
kishore######### ******java*****
ramesh########## *****oracle****
kiran########### *****mysql*****
```

https://www.hackerrank.com/challenges/swap-case/problem?isFullScreen=true

```
def swap_case(s):
    s1=''
    for ch in s:
        if ch.islower():
            s1=s1+ch.upper()
        elif ch.isupper():
            s1=s1+ch.lower()
        else:
            s1=s1+ch
```

```
    return s1

if __name__ == '__main__':
    s = input()
    result = swap_case(s)
    print(result)
```

https://www.hackerrank.com/challenges/python-string-split-and-join/problem?isFullScreen=true

```
def split_and_join(line):
    # write your code here
    list1=line.split()
    str1="-".join(list1)
    return str1

if __name__ == '__main__':
    line = input()
    result = split_and_join(line)
    print(result)
```

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.

Example:

```
>>> str1="python"
>>> str1.find('h')
3
>>> str2="java"
>>> str2.find('a')
1
>>> str2.find('p')
-1
>>> str2.find('a',2)
3
```

str.join(iterable)

Return a string which is the concatenation of the strings in iterable.

```
>>> list1=['a','b','c','d']
>>> str1=''.join(list1)
>>> print(list1)
['a', 'b', 'c', 'd']
>>> print(str1)
abcd
>>> str2=','.join(list1)
>>> print(str2)
a,b,c,d
>>> str3='-'.join(list1)
>>> print(str2)
a,b,c,d
>>> print(str3)
a-b-c-d
```

https://www.hackerrank.com/challenges/python-mutations/problem?isFullScreen=true

```
def mutate_string(string, position, character):
    list1=list(string)
    list1[position]=character
    str1=''.join(list1)

    return str1

if __name__ == '__main__':
    s = input()
    i, c = input().split()
    s_new = mutate_string(s, int(i), c)
    print(s_new)
```

String examine methods

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:

```
>>> str1="python"
>>> str1.endswith('n')
True
>>> str2="java"
>>> str2.endswith('a')
True
>>> str2.endswith('A')
False
>>> str2.endswith(('a','A'))
True
>>> str3="programming"
>>> str3.endswith('g',0,3)
False
>>> str3.endswith('i',3,9)
True
```

Example:

```
names=['naresh',
       'suresh',
       'kishore',
       'rajesh',
       'kiran',
       'raman']

for name in names:
    if name.endswith('h'):
        print(name)


for name in names:
    if name.endswith(('h','n')):
       print(name)
```

Output:

```
naresh
suresh
rajesh
naresh
```

suresh
rajesh
kiran
raman

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:

```
>>> str1="python"
>>> str1.startswith('p')
True
>>> str2="java"
>>> str2.startswith('j')
True
>>> str2.startswith('J')
False
```

Example:

```
names=['naresh',
       'suresh',
       'kishore',
       'rajesh',
       'kiran',
       'raman']

for name in names:
    if name.startswith('r'):
        print(name)

print("---------------------------")
for name in names:
    if name.startswith(('r','k')):
        print(name)
print("-----------------------------")
```

```
for name in names:
    if name.startswith(('r','k')) and name.endswith('n'):
        print(name)
```

Output:

```
rajesh
raman
---------------------------
kishore
rajesh
kiran
raman
-----------------------------
kiran
raman
```

Example:

```
>>> str1="rama rao"
>>> str1.endswith('rao')
True
>>> str2="Mr.Rama"
>>> str2.startswith('Mr.')
True
```

Partition methods

1. partition
2. rpartition