NOTICE

www.DataScienceInTamil.com

Day 23 - Batch 3 - Python Language

Chapter 12 – String (ALL string methods are covered)

To watch the recorded Python and Data Science videos in YouTube:

Day 19- Batch 3 - String, Str, str - ALL methods - Part 1

https://youtu.be/CfAkVkfE5fs

Day 20- Batch 3 - String, Str, str - ALL methods - Part 2

https://youtu.be/UBRZU_nbQqo

Day 21- Batch 3 - String, Str, str - ALL methods - Part 3

https://youtu.be/mbJ6F7H95C4

Day 22- Batch 3 - String, Str, str - ALL methods - Part 4

https://youtu.be/sT_SN86Wm-M

Day 23- Batch 3 - String, Str, str - ALL methods - Part 5 (FINAL PART)

https://youtu.be/uGyE3yUf74U

Official Website:

https://DataScienceInTamil.com/

மேலும் முக்கிய கேள்விகள் பதில்களுக்கு :

https://www.DatascienceInTamil.com/#faq

To join DataScienceInTamil Telegram group:

இந்த குழுவில் உங்கள் நண்பர்களை இணைக்க விரும்பினால் அதற்கான லிங்க்

https://t.me/joinchat/lUZEsr-zidpjZjEx

To Join the class, please fill the form :

https://forms.gle/QFpLHwAoinFaX2cE6

Join Zoom Meeting (From Sep 26 2022 to Oct 26 2022)

https://us06web.zoom.us/j/88900302653?pwd=MVBFUlhqTTE1LzFFRUVpTzZ2S1Vsdz09

Meeting ID: 889 0030 2653

Passcode: 1234

Monday through Friday 8 PM to 10 PM IST (From Sep 26 2022 to Oct 26 2022)

We support open-source products to spread Technology to the mass.

- This is completely a FREE training course to provide introduction to Python language
- All materials / contents / images/ examples and logo used in this document are owned by the respective companies / websites. We use those contents for FREE teaching purposes only.
- We take utmost care to provide credits whenever we use materials from external source/s. If we missed to acknowledge any content that we had used here, please feel free to inform us at info@DataScienceInTamil.com.
- All the programming examples in this document are for FREE teaching purposes only.

Thanks to all the open-source community and to the below websites from where we take references / content /code example, definitions, etc., please use these websites for further reading:

- Book : Python Notes For Professionals

- https://www.w3schools.com
- https://www.geeksforgeeks.org
- https://www.askpython.com
- https://docs.python.org
- https://www.programiz.com/
- https://www.openriskmanagement.com/
- https://pynative.com/python-sets/
- https://www.alphacodingskills.com/
- https://codedestine.com/
- https://appdividend.com/
- https://freecontent.manning.com/
- https://stackoverflow.com/
- https://datagy.io/python-isdigit
- https://www.datacamp.com/community/tutorials/functions-python-tutorial
- https://data-flair.training/blogs/python-function/
- https://problemsolvingwithpython.com/07-Functions-and-Modules/07.07-Positional-and-Keyword-Arguments/
- https://www.tutorialsteacher.com/python/callable-method

WHAT TO COVER TODAY IN STRING

All 47 string functions will be covered

String Methods (ALL string methods are covered)

1. Changing the capitalization of a string
2. str.translate: Translating characters in a string
3. str.format and f-strings: Format values into a string
4. String module's useful constants
5. Stripping unwanted leading/trailing characters from a string
6. Reversing a string
7. Split a string based on a delimiter into a list of strings
8. Replace all occurrences of one substring with another substring
9. Testing what a string is composed of
10. String Contains
11. Join a list of strings into one string
12. Counting number of times a substring appears in a string
13. Case insensitive string comparisons
14. Justify strings
15. Test the starting and ending characters of a string
16. Conversion between str or bytes data and unicode characters

1. **What is string**
2. **capitalize**
3. **casefold**
4. **center**

5. count
6. encode
7. endswith
8. expandtabs
9. find
10. format
11. format_map
12. index
13. isalnum
14. isalpha
15. isascii
16. isdecimal
17. isdigit
18. isidentifier
19. islower
20. isnumeric
21. isprintable
22. isspace
23. istitle

24. **isupper**
25. **join**
26. **ljust**
27. **lower**
28. **lstrip**
29. **maketrans**
30. **partition**
31. **removeprefix**
32. **removesuffix**
33. **replace**
34. **rfind**
35. **rindex**
36. **rjust**
37. **rpartition**
38. **rsplit**
39. **rstrip**
40. **split**
41. **splitlines**
42. **startswith**

43. **strip**
44. **swapcase**
45. **title**
46. **translate**
47. **upper**
48. **zfill**
49. **The type of outermost quote cannot be used inside (Single, double and Triple quote rules**

Methods available in String class

print (dir (str))
total methods 80

special or magic methods 33

output
['__add__', '__class__', '__contains__', '__delattr__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__getitem__', '__getnewargs__', '__gt__', '__hash__', '__init__', '__init_subclass__', '__iter__', '__le__', '__len__', '__lt__', '__mod__', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__rmod__', '__rmul__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__',

Regular methods 47

['capitalize', 'casefold', 'center', 'count', 'encode', 'endswith', 'expandtabs', 'find', 'format', 'format_map', 'index', 'isalnum', 'isalpha', 'isascii', 'isdecimal', 'isdigit', 'isidentifier', 'islower', 'isnumeric', 'isprintable', 'isspace', 'istitle', 'isupper', 'join', 'ljust', 'lower', 'lstrip', 'maketrans', 'partition', 'removeprefix', 'removesuffix', 'replace', 'rfind', 'rindex', 'rjust', 'rpartition', 'rsplit',

'rstrip', 'split', 'splitlines', 'startswith', 'strip', 'swapcase', 'title', 'translate', 'upper', 'zfill']

Strings are one of the most commonly used data types in any programming language.

It represents **textual data** and **sequences of characters**. In Python, strings are **immutable** and **ordered**. This is very important while implementing a function to operate a string. Fortunately, Python has rich in-built functions that manipulate strings.

capitalize()

```
def capitalize(self) -> str

def capitalize(self, *args, **kwargs): # real signature unknown
    """
    Return a capitalized version of the string.

    More specifically, make the first character have upper case and the rest lower
```

```
case.
    """
    pass
```

Return a capitalized version of the string.

More specifically, make the first character have upper case and the rest lower case

```
s = 'monty 123 python'
print (s.capitalize())
```

Monty 123 python

Note: The first character only convert to upper case in the given string.

```
s = 'MONty 123 python'
print (s.capitalize())
```

Monty 123 python

Note: The first character convert to upper case and then remaining characters are converted to lower case

==========

Title()

```
def title(self) -> str
```

```
def title(self, *args, **kwargs): # real signature unknown
    """
    Return a version of the string where each word is titlecased.

    More specifically, words start with uppercased characters and all remaining
    cased characters have lower case.
    """
    pass
```

Return a version of the string where each word is titlecased.
More specifically, words start with uppercased characters and all remaining cased characters have lower case.

```
s = "i am learning PYTHON FOR EVERY DAY"
print(s.title())
```
I Am Learning Python For Every Day

```
s = "#1i_2am_learning_PYTHON_FOR_EVERY_DAY"
print(s.title())
```
output

#1I_2Am_Learning_Pyt2Hon_For_Every_Day

Note: Special symbols separates a word

```
s = "#1i_2am_learning_PYT2HON_FOR_EV$ERY_DAY" # numerals and special chars are considered as separators in a string
print(s.title())

s = "#1i_2am_lear.ning_PYT2HON_FOR_EV$ERY_DAY"
print(s.title())
```

#1I_2Am_Learning_Pyt2Hon_For_Ev$Ery_Day
#1I_2Am_Lear.Ning_Pyt2Hon_For_Ev$Ery_Day

```
print ("dataதscienceமி*ழ்in&tamil".title())
```
output
Dataதscienceமி*ழ்In&Tamil

```
s = "i am learning python for every day"
print(s.title())
```

I Am Learning Python For Every Day

Note: If the given string is lowercases or upper cases and include with multiple words, all the words first letter converted to upper cases and the remaining characters are converted to lower cases.

```
s = 'monty 123 python'
print (s.title())
```

Monty 123 Python

Watch closely the below code to see how the title works with '-' and ',' (with special chracters)

```
print('ab cd-ef'.title())
print('ab cd,ef'.title())
print('ab cd ef'.title())
```

output

Ab Cd-Ef
Ab Cd,Ef
Ab Cd Ef

Special symbols terminate a word.

==========

Istitle()

```
def istitle(self) -> bool

def istitle(self, *args, **kwargs): # real signature unknown
    """
    Return True if the string is a title-cased string, False otherwise.

    In a title-cased string, upper- and title-case characters may only
    follow uncased characters and lowercase characters only cased ones.
    """
    pass
```

Return True if the string is a title-cased string, False otherwise.
In a title-cased string, upper- and title-case characters may only follow uncased characters and lowercase characters only cased ones.

The **istitle**() **method** returns True if each word in a text starts with a upper case letter, AND the rest of the chracters in the word are lower case letters, otherwise False.

```
a = "HELLO, AND WELCOME TO MY WORLD"
b = "Hello"
c = "22 Names"
d = "This Is %'!?"

print(a.istitle())
print(b.istitle())
print(c.istitle())
print(d.istitle())
e = "Th2Is Is %'!?"
```

```
    False
    True
    True
    True
True
```

```
b = "lawra"
print(b.istitle())
```
False

```
b = "HEllo"
print(b.istitle())
```
False

```
a = "Hello, And"
print(a.istitle())
a = "Hello, ANd"
print(a.istitle())
```

True
False

Note: All the strings start with first character is upper case, and the remaining characters are lower case in a given string.
Note: the given string must have atleast one character length (ie it should not be empty. If so it is false)- see below

```
a = ' '
print(a.istitle())
```

False

```
a = ' ,  '
print(a.istitle())
```

False

```
a = ',A '
print(a.istitle())
```

True

Note: The chracters should be between **a-z and A-Z**

```
a = '1Abc'
print(a.istitle())
```

True

Note: string starts with numeric the numeric value will be discarded first, the the evaluation starts

```
a = '1Abc, 2Bcb'
print(a.istitle())
True
------
a = '111'
print(a.istitle())
 False
--------
```

```
a="HellஅOw" # H is caps but o is  in lower, so the output is False
print(a.istitle())
False
```

```
a="HellஅOw" # H and O are in caps, so the output is True
print(a.istitle())

True
```

==========

Center()

```
def center(self,          __width: int,          __fillchar: str = ...) -> str

def center(self, *args, **kwargs): # real signature unknown
    """
    Return a centered string of length width.
        Padding is done using the specified fill character (default is a space).
    """
    pass
```

Return a centered string of length width.
Padding is done using the specified fill character (default is a space).

```
str1 = 'monty 123 python'
print (len(str1))
print (str1.center(25,'G'))
```

16
GGGGGmonty 123 pythonGGGG

Real time example (When we write check amount) see below

```
str1 = 'Rs 3000/'
print (len(str1))
print (str1.center(20,'*'))
```

output

8
******Rs 3000/******

Note:The given string must be centered and add ‘G’ in prefix and suffix of the given string depend upon the integer(25).

Note: The fill character must be exactly one character long
If we give print (str1.center('GG')), it is a TypeError

print (str1.center(25,' ')) // add padding of 5 spaces in front and 4 spaces at end , since the spaces are non printable characters, we cant see it, to see the non printable charactes use repr(). See down how to use repr()

```
str1 = 'monty 123 python'
print (str1.center('G')) # int ie, width is not given so it is an error
TypeError: 'str' object cannot be interpreted as an integer

str1 = 'monty 123 python'
print (str1.center(‘5’)) # Width is not given , hence error
 TypeError: 'str' object cannot be interpreted as an integer
-----
```

```
st = 'Lara'
x = st.center(20) # if fill charactes is not given, it takes SPACE as it padding
print(repr(x))
```

output

```
'        Lara        '
```

Another example

```
st = ' Python is for "Data Science"'
print(st.center(50, '&'))
```

output

```
&&&&&&&&&& Python is for "Data Science"&&&&&&&&&&
```

```
str1 = 'monty 123 python'
print (len(str1))
a =  (str1.center(50,' '))
print(a)
print(repr(a))
```

output

```
'                monty 123 python                '
```

```
s = 'Tamil '

print(s.center(20,'Z'))
print(s.center(-20,'Z')) # witdh must be positive number
```

ZZZZZZZTamil ZZZZZZZ
Tamil

```
s = 'Tamil '
print(s.center(10.5,'Z')) # width must be in integer, else error
```

TypeError: 'float' object cannot be interpreted as an integer

Count()

```
def count(self, x: str,  __start: Optional[int] = ..., __end: Optional[int] = ...) -> int

def count(self, sub, start=None, end=None): # real signature unknown; restored from __doc__
    """
    S.count(sub[, start[, end]]) -> int

    Return the number of non-overlapping occurrences of substring sub in
    string S[start:end].  Optional arguments start and end are
```

```
    interpreted as in slice notation.
    """
    return 0
```

S.count(sub[, start[, end]]) -> int

Return the number of non-overlapping occurrences of **substring** sub in string S[start:end]. Optional arguments start and end are interpreted as in slice notation.

```
str1 = 'monty monty 123 python'
print (str1.count('monty'))
```

2

Note: Now we are checking how many substring of ‘monty’ is available in the given string, so we get the output 2

```
str1 = 'monty monty 123 python'
print (str1.count('y',4,18))
```

3

Note: now we check how many substring ‘y’ is available in the given string. And also we asked python to check the number of substring from 4th index to 18th index. There will 3 occraances of substring ‘y’ in the given string

```
str1 = 'monty monty 123 pythyon'
str2 = 'monty' # declare the variable and pass the same to count() as its parameter
print (str1.count(str2))
output 2
```

```
str1 = 'monty monty 123 python'
print (str1.count('y',5,18))
```

2

Note: now we check how many substring ‘y’ is available in the given string. And also we asked python to check the number of substring from 5th index to 18th index. There will 2 occraances of substring ‘y’ in the given string

---

```
str1 = 'monty monty 123 python'
print (len(str1))
print (str1.count('monty monty 123 python'))
```

Note: it counts how many repeated sub strings available in the given string

22

1

Count(), the first parameter (str) is mandatory, else it gives error as below

```
str1 = 'monty monty 123 python'
print (str1.count())
```

```
    print (str1.count())
TypeError: count() takes at least 1 argument (0 given)
```

```
str1 = "SSaaSSbbSS"
for index, char in enumerate(str1):
    print(index, char)
print(str1.count('S',3,6))
```

output

```
0 S
1 S
2 a
3 a
4 S
5 S
6 b
7 b
8 S
```

9 S
2

Count () with slicing

```
str1 = 'monty monty 123 python o'
print (str1[2:10].count('o'))
```

output
1

```
str1 = 'monty monty 123 python o'
print(str1)
print (str1[2:10])
print (str1[2:10].count('o'))
```

output
monty monty 123 python o
nty mont
1

==========

Endswith()

```
def endswith(self,suffix: Union[str, Tuple[str, ...]],start: Optional[int] = ...,
             end: Optional[int] = ...) -> bool
```

```
def endswith(self, suffix, start=None, end=None): # real signature unknown;
restored from __doc__
    """
    S.endswith(suffix[, start[, end]]) -> bool

    Return True if S ends with the specified suffix, False otherwise.
    With optional start, test S beginning at that position.
    With optional end, stop comparing S at that position.
    suffix can also be a tuple of strings to try.
    """
    return False
```

S.endswith(suffix[, start[, end]]) -> bool

Return True if S ends with the specified suffix, False otherwise. With optional start, test S beginning at that position. With optional end, stop comparing S at that position. suffix can also be a tuple of strings to try.

```
s = 'monty 123 python'
print (s.endswith('n'))
s = 'monty 123 python'
print (s.endswith('h'))
```

Note: This method consider for last character of given string. If the argument passed last character of given string, the output shows as True and otherwise shows as Fail.

```
s = 'monty 123 python'
print (s.endswith('n'))
print(s[2:12])
print (s.endswith('n',2,12))
```

True

Note: The above code put the integer value is fulfill of count in last character(index) of given string. So the output shows as True.

```
s = 'monty 123 python'
print(s)
```

```
subString = (s[2:18])
print(subString)
for item in enumerate(subString):
    print(item)
print (s.endswith('n',2,18))
--------------


s = 'monty 123 python'
print(s)
print("Length is ", len(s))
print(s[2:15])
print (s.endswith('o',2,15))
print (s.endswith('h',2,15))
```

output
monty 123 python
Length is 16
nty 123 pytho
True
False

Note: The above code put the integer value is not fulfill of count in last character of given string. So the output shows as False.

```
s = 'monty 123 python'
print (s.endswith('python'))
```

```
s = 'monty 123 python'
print (s.endswith())
 TypeError: endswith() takes at least 1 argument (0 given)
```

Expandtabs()

```
def expandtabs(self, tabsize: int = ...) -> str

def expandtabs(self, *args, **kwargs): # real signature unknown
    """
    Return a copy where all tab characters are expanded using spaces.
        If tabsize is not given, a tab size of 8 characters is assumed.
    """
    pass
```

Return a copy where all tab characters are expanded using spaces.
If tabsize is not given, a tab size of 8 characters is assumed

```
str1 = "WE are LEARNING 12345 datascience" # this can be expanded
print("original string ", str1)
print(len(str1))
print("==========")

str2 = "WE \tare \tLEARNING \t12345 \tdatascience" # this can be expanded
print("Expanded string with single tab", str2)
print(len(str2))
print("==========")

# str2 = "WE \t\tare \t\tLEARNING \t\t12345 \t\tdatascience" # this can be expanded
# print("Expanded string with DOUBLE tab", str2)
# print(len(str2))
# print("==========")

Expanded string with DOUBLE tab
 WE             are             LEARNING    12345                datascience
40
--------------------------
str2 = "WE are \tLEARNING 12345 \tdatas\tcience"
print('Original String is \n:', str)
print(len(str2))
print("==========")
```

```
# no tab size is given
print('Tabsize is nothing:', str2.expandtabs())
print('Tabsize is nothing:', len(str2.expandtabs()))
print("==========")
#
# tabsize is set to 2
print('Tabsize 2:', str2.expandtabs(2))
print('Tabsize 2:', len(str2.expandtabs(2)))
print("==========")

# # tabsize is set to 3
print('Tabsize 3:', str2.expandtabs(3))
print('Tabsize 3:', len(str2.expandtabs(3)))
print("==========")

# # tabsize is set to 4
print('Tabsize 4:', str2.expandtabs(4))
print('Tabsize 4:', len( str2.expandtabs(4)))
print("==========")

# # tabsize is set to 5
print('Tabsize 5:', str2.expandtabs(5))
print('Tabsize 5:', len(str2.expandtabs(5)))
print("==========")

# # tabsize is set to 6
print('Tabsize 6:', str2.expandtabs(20))
print('Tabsize 6:', len(str2.expandtabs(20)))
```

output

```
original string  WE are LEARNING 12345 datascience
33
==========
Expanded string with single tab WE	are	LEARNING	12345	datascience
37
==========
Original String is : WE are LEARNING 12345 datascience
33
==========
Tabsize is nothing: WE	are	LEARNING	12345	datascience
Tabsize is nothing: 51
==========
Tabsize 2: WE  are   LEARNING  12345   datascience
Tabsize 2: 39
==========
Tabsize 3: WE   are   LEARNING   12345   datascience
Tabsize 3: 44
==========
Tabsize 4: WE  are    LEARNING   12345   datascience
```

```
Tabsize 4: 43
==========
Tabsize 5: WE   are  LEARNING  12345    datascience
Tabsize 5: 41
==========
Tabsize 6: WE              are              LEARNING          12345
datascience
Tabsize 6: 91
```

Note: This method used for expandable of the given string based on the int value.(using ‘ \t ‘ of the given string).

==========

Rfind() and rindex()

Both give the same result but the difference is

rindex() method is similar to rfind() method for strings. The only difference is that rfind() returns -1 if the substring is not found, whereas rindex() throws an exception.

```
s = "example of me rindex method"
print(s.rindex('meS')) # ValueError: substring not found
print(s.rfind('meS')) # gives -1 if there is no such substring
```

```
def rfind(self,
          sub: str,
          __start: Optional[int] = ...,
          __end: Optional[int] = ...) -> int
```

```
def rfind(self, sub, start=None, end=None): # real signature unknown; restored from __doc__
    """
    S.rfind(sub[, start[, end]]) -> int

    Return the highest index in S 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.
```

```
    """
    return 0
```

S.rfind(sub[, start[, end]]) -> int

Return the highest index in S 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.

Definition and Usage

The rfind() method finds the last occurrence of the specified value.

The rfind() method returns -1 if the value is not found.

The rfind() method is almost the same as the rindex() method.

Parameter Values

Parameter	Description
value	Required. The value to search for
start	Optional. Where to start the search. Default is 0
end	Optional. Where to end the search. Default is to the end of the string

Note: The major difference between find() and rfind() is:

The find(),find the position of a given string in staring of the sequence.
The rfind(),find the position of a given string in ending of the sequence.

```
s = 'i learn python'
print(len(s))
print (s.rfind('n'))
```

```
 14
  13
```

Note: Here the total length of the string is 14. The find the string 'n' is in last occurrence of the sequence in place of 13(n-1 ie 14-1=13).

```
s = 'i learn python you learn it'
print (s.rfind('l'))
```

```
 19
```

Note:Consider the given string('l') it is the position of 2,19(two times of 'l' character in a sequecnc). The result gets the last occurrence of the sequence(left to right) consider at 19th position only.

```
s = 'i learn python you learn it'
print (s.rfind('i'))
```

```
 25
```

Note: Consider the given string('i') it is the position of 0,25(two times of 'i' character in a sequecnc). The result gets the last occurrence of the sequence(left to right) consider at 25th position only.

```
s = 'i learn python you learn it'
print (s.rfind('i',2,26))
 25
```

Note: Consider the given string(‘i’) it is the position of 0,25(two times of ‘i’ character in a sequecnc). But the parameter pass to consider the positon of 2 to 26. So that 0^{th} position of the ‘i’ is not considered. Only consider the output takes 25^{th} position of i(n-1 ie 26-1=25).

```
s = 'i learn python you learn it'
print (s.rfind('i',2,25))
 -1
```

Note: The rfind() method returns -1 if the value is not found.

```
s = 'monty 123 python'
print (s.rfind())
 TypeError: rfind() takes at least 1 argument (0 given)
```

==========

Rindex()

```
def rindex(self,
           sub: str,
           __start: Optional[int] = ...,
           __end: Optional[int] = ...) -> int

def rindex(self, sub, start=None, end=None): # real signature unknown; restored from __doc__
    """
    S.rindex(sub[, start[, end]]) -> int

    Return the highest index in S 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.

    Raises ValueError when the substring is not found.
    """
    return 0
```

S.rindex(sub[, start[, end]]) -> int (From Dhana – from DS)

Return the **highest index in S where substring is found,** such that sub is contained within S[start:end]. Optional arguments start and end are interpreted as in slice notation.

Raises ValueError when the substring is not found.

Note: The difference between index() and rindex() is:
If the sub strings(arguments value) are repeated,

1. The index(), get the position of the particular string(sub string) in a starting level.
2. The rindex(), get the highest position of the particular string(sub string) in a ending level.

```
s = "example of rindex method"
print(s.rindex('x'))
```

16

Note : This method is used to get the position of the sub string in a given string. Here the sub string 'x' is repeated on two times in a given string. But using th rindex() take the position of the substring in the index to end of the sequence. So that, sub steing 'x' position of the index 1,16. The 1 is the starting level of the index and 16 is the ending level of the index. So that, rindex() take the

position of the sub string result is 16. But incase of the index() take the position of the sub string result is 1.

```
s = "example of rindex method"
print(s.rindex('x',1,16))
 1
```

Note: In this example, get the sub string ‘x’ and the integer value is 1, 16 (parameter int value). It take the result as 1, because the parameter int 16th (n-1 ie 16-1 = 15) did not get the result of 15 th position of ‘x’. The ‘x’ is placed on 16 th position of the index(17-1=16), so that result gets in 1. If the sub string parameter integer passed 17 (print(s.rindex('x',1,17)))we get the result 16. Given below:

```
s = "example of rindex method"
print(s.rindex('x',1,17))
 16
```

```
s = "example of rindex method"
print(s.rindex('x',2,16))
 ValueError: substring not found
```

Note: Here the sub string ‘x’ is not found in between position of int 2, 16.

```
s = 'monty 123 python'
print (s.rindex())
                    TypeError: rindex() takes at least 1 argument (0 given)
==========================
```

Isalpha()

```
def isalpha(self) -> bool

def isalpha(self, *args, **kwargs): # real signature unknown
    """
    Return True if the string is an alphabetic string, False otherwise.

    A string is alphabetic if all characters in the string are alphabetic and there
    is at least one character in the string.
    """
    pass
```

Return True if the string is an alphabetic string, False otherwise.

A string is alphabetic if **all** characters in the string are alphabetic and there is at least one character in the string.

```
s = '' # empty string
print (s.isalpha())
```

output

False

```
s = "sample string" # False, 'cos of SPACE
print (s.isalpha())
```

```
s = "" # False, 'cos of EMPTY
print (s.isalpha())
```

```
s = 'montypython'
print (s.isalpha())
```

True

Note: The given string is an alphabet, it shows the output True.

```
s = 'monty123python'
print (s.isalpha())
```

False

Note: The given string is an alphanumeric, it shows the output False.

```
s = 'montypyth,on' # string contains comma
print (s.isalpha())
```

False

isascii()

```
def isascii(self) -> bool
```

```
def isascii(self, *args, **kwargs): # real signature unknown
    """
    Return True if all characters in the string are ASCII, False otherwise.

    ASCII characters have code points in the range U+0000-U+007F.
    Empty string is ASCII too.
    """
    pass
```

Return True if all characters in the string are ASCII, False otherwise.

ASCII characters have code points in the range U+0000-U+007F. Empty string is ASCII too.

```
s = 'monty 123 python'
print (s.isascii())
```
True

```
s = '123456'
print (s.isascii())
```
True
Note: If all the characters are string shows the output is True.

```
s = 123456
print (s.isascii())
```
AttributeError: 'int' object has no attribute 'isascii'
Note: Integer object has no a attribute values in ‘isascii’.

```
s = 'monty123python'
print (s.isascii('m'))
```
TypeError: isascii() takes no arguments (1 given)

```
s = 'monty 123 python'
print (s.isascii())

str1="தமிழ்"
print(str1.isascii()) # False


s = '123456'
print (s.isascii())

s = '123.456'
print (s.isascii()) #True

s = 'ABC'
print (s.isascii())

s = '0'
print (s.isascii())

s = "My name is Ståle"
print(s.isascii())

x = ascii("My name is Ståle") #watch closely å
print(x)
y = "My name is Ståle"
print(y.isascii())
```

output
True
True
True
True
True
False
'My name is St\xe5le'

isdecimal()

isdecimal()	isdigit()	isnumeric()	Example
True	True	True	"038", "০৩৮", "０３８"
False	True	True	"³⁸", "0₃ ₈", "⓪③⑧"
False	False	True	"⅗⅛⅘", "Ⅰ ⅢⅧ", "⑩⑬㊿", "壹貳參"
False	False	False	"abc", "38.0", "-38"

```
def isdecimal(self) -> bool

def isdecimal(self, *args, **kwargs): # real signature unknown
    """
    Return True if the string is a decimal string, False otherwise.

    A string is a decimal string if all characters in the string are decimal and
    there is at least one character in the string.
    """
    pass
```

Return True if the string is a decimal string, False otherwise.

A string is a decimal string if all characters in the string are decimal and there is at least one character in the string.

Python String **isdecimal**() Method
The **isdecimal**() method returns True if all the characters are decimals (0-9). This method is used on unicode objects.

```
s = '11'
print (s.isdecimal())
```

True
Note: If all the values are decimals (0-9) in the given string , the output shows as True.

```
s = '1.1'
print (s.isdecimal())
```
False
Note: If all the values are floating in the given string , the output shows as False.

```
s = 'abc'
print (s.isdecimal())
```
False
Note: If all the values are alphabetic character in the given string , the output shows as False.

```
s = 'monty123python'
print (s.isdecimal())
```
False

```
s = '11.0'
print(type(s))
print (s.isdecimal())
print("============") False


s = '0'
print(type(s)) # True

print (s.isdecimal()) #True
print("============")

s = '0.0' # False
print(type(s))
print (s.isdecimal())
print("============")

a = "10"
b = "20"
print(a+b) #1020
print("===========")

a = "10"
b = "20"
print(int(a)+int(b)) # 30
print(type(int(a)+int(b)))
print("===========")

a = "10.5"
b = "20.5"
```

```
print(a+b) #10.520.5
print("===========")

a = "10"
b = "20"
print(float(a)+float(b)) # 30.0
print(type(int(a)+int(b)))
print("===========")

a = "10.5"
b = "20.5"
print(int(a)+int(b)) #ERROR
print("===========")

a = "10"
b = "20"
print(float(a)+float(b)) #NO ERROR
print("===========")
```

The Quick Answer: A Summary of the Methods

String Type	Example	Python `.isdecimal()`	Python `.isdigit()`	Python `.isnumeric()`
Base 10 Numbers	`'0123'`	True	True	True
Fractions and Superscripts	`'⅗'`, `'2²'`	False	True	True
Roman Numerals	`'Ⅾ'`	False	False	True

Differences between the Python `.isdecimal()`, `.isdigit()`, and `.isnumeric()` methods

Isdigit()

1. The isdigit method

- This is a built-in method which checks the string for the presence of digits (0-9) in it.
- This method returns True, if the string contains only numbers.
- The string should contain atleast one character, since an empty string returns False.
- Roman numerals, fractions, and currencies are **NOT considered as digits**. Hence, this method will return False for such strings.
- The subscript, superscript and decimal characters are **considered to be digits.** Hence, this method will return True for such strings (Doubt: more light to be shed on this point / yet to be confirmed)

Syntax of isdigit method - string.isdigit()

Example:

```
x = '123'
print(x.isdigit())

z='0'
```

```
print(z.isdigit())

z='0.0'
print(z.isdigit())

z='5.6'
print(z.isdigit())

z='III'
print(z.isdigit())

y = '123abc'
print(y.isdigit())

z=""
print(z.isdigit())
```

output

```
True

True

False

False
```

False

False

False

isnumeric() method:

- This is a built-in method which checks the string for the presence of numeric values.
- The numeric values refer to **Unicode characters** which includes integers, subscript, superscript, roman numerals, and fractions(1/2) BUT NOT decimals (0.5)
- It returns True if the characters are numerical values
- It returns False, if the string is empty or has other characters like alphabets and special characters.

Errors and Exceptions:

1. It does not contain any arguments, therefore, it returns an error if a parameter is passed.
2. **Whitespaces** are not considered to be numeric, therefore, it returns “False”

3. Subscript, Superscript, Fractions, Roman numerals (all written in Unicode)are all considered to be numeric, Therefore, it returns "True"

Syntax of isnumeric method - string.isnumeric()

```
my_string = '\u00BC'
print(my_string.isnumeric())

my_string = '1230579'
print(my_string.isnumeric())

f = "ⅠⅢⅧ" # Note this Roman numeals copied from google / output True
print(f.isnumeric())
print(f)


my_string = '123abx'
print(my_string.isnumeric())

my_string = ''
print(my_string.isnumeric())

my_string = "四"
print(my_string.isnumeric())
```

output

True

True

False

False

False

True

For example, in Chinese, we count 1, 2, 3, 4, 5 as 一， 二， 三， 四， 五. It turns out that the Chinese characters for numbers will return False for str.isdigit, but True for str.isnumeric, behaving differently from their 0-9 counterparts

isdigit() isnumeric():

Example of string with digits:

"12345"

"1233"

"3"

```
a = "\u0030" #unicode for 0
b = "\u00B2" #unicode for ²
c = "10km2"
d = "-1" # 1 is  numeric but -1 is not
e = "1.5"

print(a.isnumeric())
print(b.isnumeric())
print(c.isnumeric())
print(d.isnumeric())
print(e.isnumeric())
```

output

True

True

False

False

False

Returns ‘true’ if all characters of the string are digits.

Example of string with numerics:

"12345"

"½¼"

"½"

"12345½"

Returns ‘true if all characters of the string are numeric

a = "½"
b = "0.5"

```
print(a.isnumeric())
print(b.isnumeric())
```

output

True

False

```
a = "-½"
```

False (Watch the minus sign in the fraction,ie positive fractions on, not the negative ones)

---

def isdigit(self) -> bool

```
def isdigit(self, *args, **kwargs): # real signature unknown
    """
    Return True if the string is a digit string, False otherwise.

    A string is a digit string if all characters in the string are digits and there
    is at least one character in the string.
```

```
    """
    pass
```

Return True if the string is a digit string, False otherwise.
A string is a digit string if all characters in the string are digits and there is at least one character in the string.

Isidentifier()

```
def isidentifier(self) -> bool

def isidentifier(self, *args, **kwargs): # real signature unknown
    """
    Return True if the string is a valid Python identifier, False otherwise.

    Use keyword.iskeyword() to test for reserved identifiers such as "def" and
    "class".
    """
    pass
```

Return True if the string is a valid Python identifier, False otherwise.
Use keyword.iskeyword() to test for reserved identifiers such as "def" and "class".

The isidentifier() method returns True if the string is a valid identifier, otherwise False.

A string is considered a valid identifier if it only contains alphanumeric letters (a-z) and (0-9), or underscores (_). A valid identifier cannot start with a number, or contain any spaces.

```
s = 'python'
print (s.isidentifier())
 True

s = '123'
print (s.isidentifier())
 False

a = "MyFolder"
b = "Demo002"
c = "2bring"
d = "my demo" # watch the space in between
```

```
e = _”Name”
f = _123

print(a.isidentifier())
print(b.isidentifier())
print(c.isidentifier())
print(d.isidentifier())
```

```
 True
     True
     False
     False
True
True
```

Note: If the given string start at number and that string in spaces, the result gets False, otherwise gets True.

```
s = 'monty 123 python'
print (s.isidentifier())
```

```
False
```

```
SS = "_"
print (SS.isidentifier())

SS = "aabb"
print (SS.isidentifier())

SS = "BBZz"
print (SS.isidentifier())

SS = "4BB" # False
print (SS.isidentifier())

SS = "AAA+" # False
print (SS.isidentifier())

SS = "AAA&" # False
print (SS.isidentifier())
```

==========

Isnumeric()

def isnumeric(self) -> bool

```
def isnumeric(self, *args, **kwargs): # real signature unknown
    """
    Return True if the string is a numeric string, False otherwise.

    A string is numeric if all characters in the string are numeric and there is at
    least one character in the string.
    """
    pass
```

Return True if the string is a numeric string, False otherwise.
A string is numeric, if all characters in the string are numeric and there is at least one character in the string.

```
s = '123'
print (s.isnumeric())
 True
--------------

s = 'abc'
print (s.isnumeric())
 False
------------------
```

```
s = '2abc'
print (s.isnumeric())
```
False

```
s = 'monty 123 python'
print (s.isnumeric())
```
False

```
s = ''
print (s.isnumeric())
```
False (Empty string is false)

Note: Except numeric string, all are False (The numeric string must not have any sign characters. If so, it is false, see below

```
s = '-1'
print (s.isnumeric())
```
False

```
s = '+0881'
print (s.isnumeric())
```
False

Note: If the numeric value of given string (without + or – sign), it gets the result True, otherwise False.

```
s = '0'
print (s.isnumeric())
```
True

```
s = '3i+6'
print (s.isnumeric())
```
False

Isprintable()

```
def isprintable(self) -> bool

def isprintable(self, *args, **kwargs): # real signature unknown
    """
    Return True if the string is printable, False otherwise.

    A string is printable if all of its characters are considered printable in
    repr() or if it is empty.
    """
    pass
```

Return True if the string is printable, False otherwise.
A string is printable if all of its characters are considered printable in repr() or if it is empty.

```
s = '2abc'
print (s.isprintable())
True
---------------
s = 'monty 123 python'
print (s.isprintable())
True
-------------

s = 'abc\n'
print (s.isprintable())
False
-------
s = 'abc\n\n'
print (s.isprintable())
print(s)
print(repr(s))
```

output
False
abc
'abc\n\n'

Note: New line characters are the indicators that the line ends there. It is not printable
Note : If the string is printable the result gets True, otherwise False.

Isspace()

```
def isspace(self) -> bool
def isspace(self, *args, **kwargs): # real signature unknown
    """
    Return True if the string is a whitespace string, False otherwise.

    A string is whitespace if all characters in the string are whitespace and there
    is at least one character in the string.
    """
    pass
```

Return True if the string is a whitespace string, False otherwise.

A string is whitespace if all characters in the string are whitespace and there is at least one character in the string.

```
s = '      '
print (s.isspace())
 True
----------
s = '  p    '
print (s.isspace())
 False
--------------
s = 'monty 123 python'
print (s.isspace())
 False
-----
s =''
print(s.isspace())
False
------
s ='\n'
print(s.isspace())
True
```

```
s ='\t'
print(s.isspace())
```
True

Note: non printable characters are actually creates SPACE – but not visible

Use repr(), to check

Note: Empty string is different from empty space string (a sring that has only empty space)

Note: If the string is a whitespace string(empty with space), the result gets True, otherwise False.

==========

Join()

def join(self, __iterable: Iterable[str]) -> str

```
def join(self, ab=None, pq=None, rs=None): # real signature unknown; restored from __doc__
    """
    Concatenate any number of strings.

    The string whose method is called is inserted in between each given string.
    The result is returned as a new string.

    Example: '.'.join(['ab', 'pq', 'rs']) -> 'ab.pq.rs'
    """
    pass
```

Concatenate any number of strings.
The string whose method is called is inserted in between each given string. The result is returned as a new string.

Example: '.' .join(['ab', 'pq', 'rs']) -> 'ab.pq.rs'

Python String **join**() **Method**
The **join**() **method** takes all items in an iterable and **joins** them into one string. A string must be specified as the separator.

```
str3 = (["AA", "BB",  "cc", "&&&&&&", "SSSS", "XXXXX", "ZZ"])
str3 = "K: ".join(str3)
print(str3)
```

output
AAK: BBK: ccK: &&&&&&K: SSSSK: XXXXXK: ZZ
Note: the string separator is added with every element in the sequence in the list except the last string

Note: we can give the “String separater” separately or with join (), here we give the string separator along with the join()

The above pgm can be rewritten as below

```
str1 = (["AA", "BB",  "cc", "&&&&&&", "SSSS", "XXXXX", "ZZ"])
str2 = "K: "
print(str2.join(str1))
```

output
AAK: BBK: ccK: &&&&&&K: SSSSK: XXXXXK: ZZ
Note: we can give the “String seprater” separately or with join (), here we give the string separator separatley

---
Another pgm how the string separator is added with every element of the list[]

```
str1 = (["AA", "BB",  "cc", "&&&&&&", "SSSS", "XXXXX", "ZZ"])
str2 = ' StringSperator '
str3 = str2.join(str1)
print(str3)
```

output

AA StringSperator BB StringSperator cc StringSperator &&&&&& StringSperator SSSS StringSperator XXXXX StringSperator ZZ

Note: here the String Sperator is “StringSperator”

```
str1 = (["AA", "BB",  "cc", "&&&&&&", "SSSS", "XXXXX", "ZZ"])
str2 = ","
str3 = str2.join(str1)
print(str3)
print(type(str3))
```

output

AA,BB,cc,&&&&&&,SSSS,XXXXX,ZZ

<class 'str'>

Note: we join all the list elemets into **one big string with** comma(,)

```
str1 = (["AA", "BB",  "cc", "&&&&&&", "SSSS", "XXXXX", "ZZ"])
str2 = "||"
str3 = str2.join(str1)
print(str3)
print(type(str3))
```

output

AA||BB||cc||&&&&&&||SSSS||XXXXX||ZZ

Note: the string separarot is ||

```
str1={'data': 1, 'science': 2}
str2="@"
str3=str2.join(str1)
print(str3)
```

output

data@science

```
str1={'data': 1, 'science': 2, 13:200}
str2="@"
str3=str2.join(str1)
print(str3)
```

output is error
reason being the key 13 is not a string

How to use the join method in real time (Ex, split a sentence to words using (split()), then add custom strings to evey word

```
str1 = "Lara is studing Data Science"
lst = list()
for word in str1.split():
    lst.append(word)
print(lst)
a = lst
b= " QQ ".join(a)
print (b)
```

output
['Lara', 'is', 'studing', 'Data', 'Science']
Lara QQ is QQ studing QQ Data QQ Science

```
s = ("lara","Alex","Antony")
a = ":"
print(a.join(s))
 lara:Alex:Antony
------
s = ("lara","Abdulla","Murugan")
a = "join"
print(a.join(s))
 larajoinAlexjoinAntony
------

s = ("123","456","789")
a = "join"
print(a.join(s))
 123join456join789
----------

s = ("lara","Alex","Antony")
a = "123"
print(a.join(s))
 lara123Alex123Antony
---------
```

```
s = ("4lara","4Alex","4Antony")
a = "123"
print(a.join(s))
 4lara1234Alex1234Antony
----------

s = {"name":"lawra","country":"USA"}
a = "Age"
print(a.join(s))
 nameAgecountry
```

If dict is given, the join(), joins the string only with dict keys, not with values. And again the last element is omitted and it is not participated in the join() process- see below

```
s = {"name": "lawra","country": "USA", "Job": "Data Scientist"}
a = " Age "
print(a.join(s))
```

output

name Age country Age Job

Note the "Age sring is added only with dict keys , leaving the last element for readability purpose we added leading and traling spaces btween the join string "Age"

'''

output

name Age country Age Job

```
s = {"lawra":"name","USA":"country"}
a = "Age"
print(a.join(s))
```

lawraAgeUSA

Note: This method used for join two or more sequence of the string will convert to one sequence of the string. It is iterable.

```
s = ('monty 123 python') # this is one string, not a tuple
a = "ZZZ"
print(a.join(s))
# mZZZoZZZnZZZtZZZyZZZ ZZZ1ZZZ2ZZZ3ZZZ ZZZpZZZyZZZtZZZhZZZoZZZn

s = ('monty 123 python', ) # this tuple but only one tuple, so ZZZ will not joined
```

```
a = "ZZZ"
print(a.join(s))
# monty 123 python

s = ('monty',  ' python', ) # this tuple with 2 len, the ZZZ is added with first string 'monty', the ZZZ will not be added woth last stirng
a = "ZZZ"
print(a.join(s))
# montyZZZ python
----------
s = ('monty 123 python',) # this is one string, not a tuple
print(type(s))
a = "ZZZ  "
print(a.join(s))
# the join string WILL NOT be joined with element; reason : the iterable has one only element
# if the iterable had more than 1 element, the  JOIN STRING could have added
---------

s = 'monty 123 python'
print (s.join())
 TypeError: join() takes exactly one argument (0 given)
==========================
```

Ljust()

Str.center(), it pads both side based on the width

Ljust and Rjust pads left or right side

```
def ljust(self,
          __width: int,
          __fillchar: str = ...) -> str

def ljust(self, *args, **kwargs): # real signature unknown
    """
    Return a left-justified string of length width.
        Padding is done using the specified fill character (default is a space).
    """
    pass
```

Return a left-justified string of length width.
Padding is done using the specified fill character (default is a space).

Note : Difference between ljust() and rjust():

1. The ljust(), in the padding value(character) added by after the string sequence.
2. The rjust(), in the padding value(character) added by before the string sequence.

```
s = 'monty 123 python'
print (s.ljust(25,'a'))
 monty 123 pythonaaaaaaaaa
```

Note: The above example of a program we assign 25 character of a total sequence of the string, but the original string length was 16 character only, so in that program argument passed('a') to the sequence the 25 characters are filled in the string (25-16=9 remaining characters are added in the sequence suffix only)

```
txt = 'python'
x = txt.ljust(15)
print(x,"is very interested")
 python          is very interested
```

Note: In this method, argument passed(int, string). Incase of not passed a string value, only passed as an integer value the result shows as a blank space in between of the sequence. The blank space depend the integer value.

```
txt = '12345'
x = txt.ljust(15) # padding char is not given so it takes SPACE as its padding substring
print(x)
```

```
print(repr(x))
print(len(x))
```

output

```
12345
'12345          '
15
```

```
s = 'monty 123 python'
print (s.ljust())
 TypeError: ljust() takes at least 1 argument (0 given)
===============
```

Rjust()

```
def rjust(self,
          __width: int,
          __fillchar: str = ...) -> str
```

```
def rjust(self, *args, **kwargs): # real signature unknown
    """
    Return a right-justified string of length width.

    Padding is done using the specified fill character (default is a space).
    """
    pass
```

Return a right-justified string of length width.
Padding is done using the specified fill character (default is a space).

Definition and Usage

The rjust() method will right align the string, using a specified character (space is default) as the fill character.

Syntax

string.rjust(*length*, *character*)

Note : Difference between ljust() and rjust():

- The ljust(), in the padding value(character) added by after the string sequence.
- The rjust(), in the padding value(character) added by before the string sequence.

```
s = "example"
print(s.rjust(20))
```

```
            example
```

Note: In the result, there are actually 13 whitespaces to the left of the word example. (13 whitespaces + 7characters of given string= totally 20 characters)

```
s = "example"
print(s.rjust(20),"of rjust method")
```

```
            example of rjust method
```

```
s = "example"
print(s.rjust(20,'r'))
```

```
rrrrrrrrrrrrexample
```

Note: Here using the letter ‘r’ as the padding character. The padding character is fill the whitespaces.

```
s = "example"
print(s.rjust(10,'1'))
 111example
```

```
s = "example"
print(s.rjust(20,'$'))
print(s.ljust(20,'$'))
```

output

```
$$$$$$$$$$$$example
example$$$$$$$$$$$$
```

```
s = "example"
print(s.rjust(10,1))
 TypeError: The fill character must be a unicode character, not int
```

```
s = 'monty 123 python'
print (s.rjust())
 TypeError: rjust() takes at least 1 argument (0 given)
============
```

==========

Upper()

```
def upper(self) -> str

def upper(self, *args, **kwargs): # real signature unknown
    """ Return a copy of the string converted to uppercase. """
    pass
```

Return a copy of the string converted to uppercase.

```
s = "i am learning python for every day"
print(s.upper())
```

I AM LEARNING PYTHON FOR EVERY DAY

Note: This method used to convert lower case to upper case.

```
s = "I AM LEARNING python for every day"
print(s.upper())
```

I AM LEARNING PYTHON FOR EVERY DAY

```
s = "123I AM LEARNING python for every day123"
print(s.upper())
 123I AM LEARNING PYTHON FOR EVERY DAY123

s = 'monty 123 python'
print (s.upper())
 MONTY 123 PYTHON
==================
```

Isupper()

```
def isupper(self) -> bool

def isupper(self, *args, **kwargs): # real signature unknown
    """
    Return True if the string is an uppercase string, False otherwise.

    A string is uppercase if all cased characters in the string are uppercase and
    there is at least one cased character in the string.
    """
    pass
```

Return True if the string is an uppercase string, False otherwise.
A string is uppercase if all cased characters in the string are uppercase and there is at least one cased character in the string.

```
s = 'PYTHON'
print (s.isupper())
```
True

```
s = '123PYTHON'
print (s.isupper())
```
True

Note: This method not considered as numbers, only consider for upper case.

```
s = '123python'
print (s.isupper())
```
False

```
s = 'PYTHOn'
print (s.isupper())
```

```
 False
-----------

s = 'python'
print (s.isupper())
 False
-----------
s = 'monty 123 python'
print (s.isupper())
 False
```

Note: If the given string is all the characters only upper case, output gets True, otherwise False.

==========

Maketrans()

Return a translation table usable for str.translate().

ASCII control characters

00	NULL	(Null character)
01	SOH	(Start of Header)
02	STX	(Start of Text)
03	ETX	(End of Text)
04	EOT	(End of Trans.)
05	ENQ	(Enquiry)
06	ACK	(Acknowledgement)
07	BEL	(Bell)
08	BS	(Backspace)
09	HT	(Horizontal Tab)
10	LF	(Line feed)
11	VT	(Vertical Tab)
12	FF	(Form feed)
13	CR	(Carriage return)
14	SO	(Shift Out)
15	SI	(Shift In)
16	DLE	(Data link escape)
17	DC1	(Device control 1)
18	DC2	(Device control 2)
19	DC3	(Device control 3)
20	DC4	(Device control 4)
21	NAK	(Negative acknowl.)
22	SYN	(Synchronous idle)
23	ETB	(End of trans. block)
24	CAN	(Cancel)
25	EM	(End of medium)
26	SUB	(Substitute)
27	ESC	(Escape)
28	FS	(File separator)
29	GS	(Group separator)
30	RS	(Record separator)
31	US	(Unit separator)
127	DEL	(Delete)

ASCII printable characters

32	space	64	@	96	`
33	!	65	A	97	a
34	"	66	B	98	b
35	#	67	C	99	c
36	$	68	D	100	d
37	%	69	E	101	e
38	&	70	F	102	f
39	'	71	G	103	g
40	(	72	H	104	h
41	)	73	I	105	i
42	*	74	J	106	j
43	+	75	K	107	k
44	,	76	L	108	l
45	-	77	M	109	m
46	.	78	N	110	n
47	/	79	O	111	o
48	0	80	P	112	p
49	1	81	Q	113	q
50	2	82	R	114	r
51	3	83	S	115	s
52	4	84	T	116	t
53	5	85	U	117	u
54	6	86	V	118	v
55	7	87	W	119	w
56	8	88	X	120	x
57	9	89	Y	121	y
58	:	90	Z	122	z
59	;	91	[	123	{
60	<	92	\	124	|
61	=	93	]	125	}
62	>	94	^	126	~
63	?	95	_		

Extended ASCII characters

128	Ç	160	á	192	└	224	Ó
129	ü	161	í	193	┴	225	ß
130	é	162	ó	194	┬	226	Ô
131	â	163	ú	195	├	227	Ò
132	ä	164	ñ	196	─	228	õ
133	à	165	Ñ	197	┼	229	Õ
134	å	166	ª	198	ã	230	µ
135	ç	167	º	199	Ã	231	þ
136	ê	168	¿	200	╚	232	Þ
137	ë	169	®	201	╔	233	Ú
138	è	170	¬	202	╩	234	Û
139	ï	171	½	203	╦	235	Ù
140	î	172	¼	204	╠	236	ý
141	ì	173	¡	205	═	237	Ý
142	Ä	174	«	206	╬	238	¯
143	Å	175	»	207	¤	239	´
144	É	176	░	208	ð	240	≡
145	æ	177	▒	209	Ð	241	±
146	Æ	178	▓	210	Ê	242	‗
147	ô	179	│	211	Ë	243	¾
148	ö	180	┤	212	È	244	¶
149	ò	181	Á	213	ı	245	§
150	û	182	Â	214	Í	246	÷
151	ù	183	À	215	Î	247	¸
152	ÿ	184	©	216	Ï	248	°
153	Ö	185	╣	217	┘	249	¨
154	Ü	186	║	218	┌	250	·
155	ø	187	╗	219	█	251	¹
156	£	188	╝	220	▄	252	³
157	Ø	189	¢	221	¦	253	²
158	×	190	¥	222	Ì	254	■
159	ƒ	191	┐	223	▀	255	nbsp

A = 65
Z = 90
a = 97

z 122

0 = 48
1 =49
9 = 57

```
def maketrans(__x: Union[Dict[int, _T], Dict[str, _T], Dict[Union[str, int], _T]])
  -> Dict[int, _T]
Possible types:
• (__x: Union[Dict[int, _T], Dict[str, _T], Dict[Union[str, int], _T]]) -> Dict[int, _T]
• (__x: str, __y: str, __z: Optional[str]) -> Dict[int, Optional[int]]

def maketrans(self, *args, **kwargs): # real signature unknown
    """
```

***If there is only one argument**, it must be a dictionary mapping Unicode ordinals (integers) or characters to Unicode ordinals, strings or None. Character keys will be then converted to ordinals.*

***If there are two arguments**, they must be strings of equal length, and in the*

resulting dictionary, each character in x will be mapped to the character at the same position in y.

If there is a third argument, *it must be a string, whose characters will be mapped to None in the result.*

"""

pass

Return a translation table usable for str.translate().

Follow the three rules as below

If there is only one argument, it must be a dictionary mapping Unicode ordinals (integers) or characters to Unicode ordinals, strings or None. Character keys will be then converted to ordinals.

If there are two arguments, they must be **strings of equal length**, and in the resulting dictionary, each character in x will be mapped to the character at the same position in y.

If there is a third argument, it must be a string, whose characters will be mapped to None in the result.

Python String maketrans()

The maketrans() method returns a mapping table for translation usable for translate() method. This is a static method that creates a one to one mapping of a character to its translation/replacement. This method creates a Unicode representation of each character for translation.

The syntax of maketrans() method is −

```
string.maketrans(x[, y[, z]])
y and z are optional arguments.
```

String maketrans() Parameters

The maketrans() method has 3 parameters.

x - In this case if only one argument is supplied, **it must be a dictionary** and the dictionary should contain 1-to-1 mapping from a single character string to its translation OR a Unicode number (97 for 'a') to its translation. if you give only one argument to maketrans it must be a **dict**

Program samples for Rule 1

In this case if only one argument is supplied in the form of dict

```
dict1 = {'a': 'A', 'b':'B'}
'''
Dict key's  value is STRING of one length, then it is converted
into ASCII and then it is  mapped / assigned
corresponding  Dict's value (Value can be string or numerical")
'''
print("Dict keys are string, value also string {'a': 'A', 'b':'B'}")
print(str.maketrans(dict1))
```

output

```
{97: 'A', 98: 'B'}
```

```
#==========
dict2 = {1: 'A', 2:'B'}
'''
If Dict key's  are numerical value, there will be NO
 ASCII for numerical, simply the key ares mapped to value")
'''
print("Dict keys are numerical, value is string {1: 'A', 2:'B'}")
print(str.maketrans(dict2))
```

Output

```
{1: 'A', 2: 'B'}

#==========
dict3 = {'a': 1, 'b':2}
'''
Dict key's  value is STRING of one length, then it is converted
into ASCII and then it is  mapped / assigned
corresponding  Dict's value (Value can be string or numerical  ")
'''
print("Dict keys are string, value is numerical  {'a': 1, 'b':2}")
print(str.maketrans(dict3))
Output
{97: 1, 98: 2}

#==========
dict4 = {1: 10, 2:20}
'''
If Dict key's  are numerical value, there will be NO  ASCII for
numerical, simply the key ares mapped to value", (Value can be
string or numerical")
'''
print("Dict keys are numerical, value also numerical  {1: 10,
```

```
2:20}")
print(str.maketrans(dict4))
```

Output

{1: 10, 2: 20}

Program samples for Rule 2

In this case if two arguments are passed

In this case if two arguments are passed, it **must be two strings with equal length and** each character in the first string is a replacement to its corresponding index in the second string. **first maketrans argument must be a string if there is a second argument**. the first two maketrans arguments must have equal length

```
str1 = 'ABC'
str2 = 'xyz'
print("Both arguments are STRING: (str1 = 'ABC', str2 = 'xyz')")
a = str.maketrans(str1,str2)
'''
converted into ASCII:
If there are two arguments, they must be strings of equal length,
and in the resulting dictionary, each character in x will be
mapped to the character at the same position in y. (Both
arguments can be string)
'''
print(a)

output
{65: 120, 66: 121, 67: 122}
#===========

str1 = "ABC"
str2 = '123'
print("One argument is string another is STRING NUMERALS:
(str1 = 'ABC', str2 = '123')")
a = str.maketrans(str1,str2)
```

```
'''
converted into ASCII : (EVEN THE NUMERALS IS CONVERTED
TO ASCII)

If there are two arguments, they must be strings of equal length,
and in the resulting dictionary, each character in x will be mapped
to the character at the same position in y. (one arguments can be
string another one can be numerical string(Ex '123')
'''
print(a)
```

Output

{65: 49, 66: 50, 67: 51}

Program samples for Rule 3

In this case if three arguments are passed

Case 3 - In this case if **three** arguments are passed, **it must be a string**, whose characters will be mapped to None in the result each character in the third argument is mapped to none.

```
str1 = 'ABC'
str2 = 'xyz'
str3 = 'RST'
print("All 3 argumetns are STRING: (str1 = 'ABC', str2 = 'xyz', str3
= 'RST')")
a = str.maketrans(str1,str2,str3)
'''
converted into ASCII:
If there is a third argument, it
must be a string, whose characters will be mapped to None in
the result
'''
print(a)


output
{65: 120, 66: 121, 67: 122, 82: None, 83: None, 84: None}
#============

str1 = 'ABC'
```

```
str2 = 'xyz'
str3 = '123'
print("2 argumetns are STRINGS, third is STRING NUMERALS:
(str1 = 'ABC', str2 = 'xyz', str3 = '123')")
a = str.maketrans(str1,str2,str3)
print(a)
output
{65: 120, 66: 121, 67: 122, 49: None, 50: None, 51: None}
'''
converted into ASCII:(EVEN THE NUMERALS IS CONVERTED TO
ASCII)
If there is a third argument, it must be a string, whose
characters will be mapped to None in the result
'''
#===========
str1 = '123'
str2 = '456'
str3 = '789'

print("All 3 argumetns are STRING NUMERALS: (str1 = '123', str2
= '456', str3 = '789')")
```

```
a = str.maketrans(str1,str2,str3)
print(a)
```

Output

{49: 52, 50: 53, 51: 54, 55: None, 56: None, 57: None}

==========

Another code

```
str1= "ABC"
str2 = "XYZ"
str3 = "RTY"
str4 = "XY"

dic1 = {1 : 10, 2 : 20, 3:30, 4:40}
dic2 = {"N": "Lawrence", "I" : "M"}
dic3 = {"S": "Man",  "M": 'Marks', 'P': "Pincode"}

print (str.maketrans(dic1))

print (str.maketrans(dic2))

print (str.maketrans(dic3))

print(str.maketrans(str1, str2))

print (str.maketrans(str1,str2,str3))
```

```
# print (str.maketrans(str1,str2,str3, str4))#TypeError: maketrans expected at most 3 arguments, got 4
```

If we pass 3 arguments, the first 2 must be of same length. The third parament can of any length – see below

```
str1="ABCD"
str2='1234'
str3 = "abcdef"

print (str.maketrans(str1, str2))
print (str.maketrans(str1, str2,str3))
```

output

```
{65: 49, 66: 50, 67: 51, 68: 52}
```

{65: 49, 66: 50, 67: 51, 68: 52, 97: None, 98: None, 99: None, 100: None, 101: None, 102: None}

Error: string keys in translate table must be of length 1 – see below

```
dic1 = {"NP": "Lawrence", "I" : "M"}
print (str.maketrans(dic1))
```

output

ValueError: string keys in translate table must be of length 1

.=================

```
dict= {"a": '123', "b": '456' , "c": '789'}
print(str.maketrans(dict))
```

output

{97: '123', 98: '456', 99: '789'}

Note1: In this example, dict is a dictionary and It contains a mapping of characters a,b and c to 123, 456 and 789 respectively. The maketrans() creates a mapping of the character's Unicode ordinal to its corresponding translation. So, the result is 97 ('a') is mapped to '123', 98 'b' to 456 and 99 'c' to 789.

Note2: The Unicode starts with a=97, b=98, c=99……. Z=122, and 0 to 9 is mapped to 48 to 56

A = 65
Z = 90
a = 97
z 122

0 = 48
1 =49
9 = 57

```
dict= {"a": '123', "b": '456' , "c": '789',"d": '321',"e": '525'}
print(str.maketrans(dict))
```

output

```
 {97: '123', 98: '456', 99: '789', 100: '321', 101: '525'}

dict= {"x": '123', "y": '456' , "z": '789',"a": '321',"b": '525'}
print(str.maketrans(dict))

output
 {120: '123', 121: '456', 122: '789', 97: '321', 98: '525'}

dict= {"x": '123', "y": '456' , "z": '789',"a": '321',"b": '525'}
my_string1 = ""
print(my_string1.maketrans(dict))
output

 {120: '123', 121: '456', 122: '789', 97: '321', 98: '525'}

Or
dict= {"x": '123', "y": '456' , "z": '789',"a": '321',"b": '525'}
print(str.maketrans(dict))

output
{120: '123', 121: '456', 122: '789', 97: '321', 98: '525'}
```

==========

What happens if we pass 2 parameters that are dict type

```
dict1 = {'a': 'A', 'b':'B'}
dict2 = {1:10, 2:20}

print(str.maketrans(dict1, dict2))
```

output is an error

TypeError: maketrans() argument 2 must be str, not dict

Partition()

```
def partition(self, __sep: str) -> Tuple[str, str, str]

def partition(self, *args, **kwargs): # real signature unknown
    """
```

```
    Partition the string into three parts using the given separator.

    This will search for the separator in the string.  If the separator is found,
    returns a 3-tuple containing the part before the separator, the separator
    itself, and the part after it.

    If the separator is not found, returns a 3-tuple containing the original string
    and two empty strings.
    """
    pass
```

Partition the string into three parts using the given separator.
This will search for the separator in the string. If the separator is found, returns a 3-tuple containing

a) the part before the separator,
b) the separator itself,
c) and the part after it.

If the separator is not found, returns a 3-tuple containing the original string and two empty strings.

The **partition**() method searches for a specified string, and splits the string into a tuple containing three elements.

The first element contains the part before the specified string.
The second element contains the specified string.
The third element contains the part after the string.

1 - everything before the "match"
2 - the "match"
3 - everything after the "match"

Note: This method search for the *first* occurrence of the specified string.

```
s = "I am learning python for every day"
print(s.partition("python"))
```

output
('I am learning ' , 'python' , ' for every day')

Note: This method used to three partition of given string.

1. The first element (‘ I am learning’) contains the part before the specified string.
2. The second element(‘python’) contains the specified string.
3. The third element(‘for every day’) contains the part after the specified string.

```
s = "I am learning python for every day"
print(s.partition("Java"))
```

output

('I am learning python for every day' , '', '')

Note1: If the specified value is not found, the partition() method returns a tuple containing:
1 - the whole string
2 - an empty string
3 - an empty string

Note2: Here ‘Java’ is not a specified string. So returns the value of the whole string and an empty string.

```
s = "I am learning  Java python for every day Java"
print(s.partition("Java"))
```

output
('I am learning ', 'Java', ' python for every day Java')

```
s = "I am learning  Java python for every day Java"
print(s.partition(" "))
```

output
('I', ' ', 'am learning Java python for every day Java')

```
s = 'monty 123 python'
print (s.partition())
```
TypeError: partition() takes exactly one argument (0 given)
==========

Rpartition()

```
def rpartition(self, __sep: str) -> Tuple[str, str, str]

def rpartition(self, *args, **kwargs): # real signature unknown
    """
    Partition the string into three parts using the given separator.
```

```
    This will search for the separator in the string, starting at the end. If
    the separator is found, returns a 3-tuple containing the part before the
    separator, the separator itself, and the part after it.

    If the separator is not found, returns a 3-tuple containing two empty strings
    and the original string.
    """
    pass
```

Partition the string into three parts using the given separator.

This will search for the separator in the string, starting at the end. If the separator is found, returns a 3-tuple containing the part before the separator, the separator itself, and the part after it.

If the separator is not found, returns a 3-tuple containing two empty strings and the original string.

Definition and Usage

The rpartition() method searches for the last occurrence of a specified string, and splits the string into a tuple containing three elements.

The first element contains the part before the specified string.

The second element contains the specified string.

The third element contains the part after the string.

Note: The partition() and the rpartition() are mostly same, The major difference is:

1.If the specified value is NOT found, the partition() method returns a tuple containing: 1 - the whole string, 2 - an empty string, 3 - an empty string.

2. If the specified value is NOT found, the rpartition() method returns a tuple containing: 1 - an empty string, 2 - an empty string, 3 - the whole string.

```
s = "I am learning python for every day"
print(s.rpartition("python"))
```
('I am learning ' , 'python', ' for every day')

Note: This method is used to create three partition based on the specified string.

```
s = "I am learning python for every day and do lab on python day"
print(s.rpartition("python"))
```
output

('I am learning python for every day and do lab on ', 'python', ' day')

```
s = "I am learning python for every day"
print(s.rpartition("Java"))
```
('', '', 'I am learning python for every day')

Note: If the specified string not found on the sequence, the rpartition() method returns a tuple containing: 1 - an empty string, 2 - an empty string, 3 - the whole string:

```
s = "I am learning python for every day python abc and xyz 1111"
print(s.rpartition("python"))
```

```
s = "I am learning python for every day python abc and xyz python 1111"
print(s.rpartition("python"))
```

```
s = 'monty 123 python'
print (s.rpartition())
 TypeError: rpartition() takes exactly one argument (0 given)
```

==========

Replace()

```
def replace(self,   __old: str,  __new: str,  __count: int = ...) -> str

def replace(self, *args, **kwargs): # real signature unknown
    """
    Return a copy with all occurrences of substring old replaced by new.

      count
        Maximum number of occurrences to replace.
        -1 (the default value) means replace all occurrences.
```

```
    If the optional argument count is given, only the first count occurrences are
    replaced.
    """
    pass
```

Return a copy with all occurrences of substring old replaced by new.

count
Maximum number of occurrences to replace. -1 (the default value) means replace all occurrences.

If the optional argument count is given, only the first count occurrences are replaced.

```
s = "I am learning Java for every day"
print(s.replace("Java","Python"))
```

I am learning Python for every day

Note: This method used to replaced for the specified sub string in a sequence. Here the old string ‘java’ is replaced to ‘python. The first parameter is a old string the second parameter is a new string.

```
s = "I am learning Java Java for every day"
print(s.replace("Java","Python",1))
```
I am learning Python Java for every day

```
s = "Java I am learning Java Java Java for every day"
print(s.replace("Java","Python",2))
```
Python I am learning Python Java Java for every day

Note: The first parameter is a old string, the second parameter is a new string, the third parameter is a integer value used to number of old strings are replaced.

```
s = 'monty 123 python'
print (s.replace())
```
TypeError: replace() takes at least 2 arguments (0 given)

Space also can replaced by any string or by space

```
s = "I am learning Java for Java every day Java"
print(s.replace(" "," w "))
```

Output

I w am w learning w Java w for w Java w every w day w Java

Multiple space also be given as NEW VALUE

```
s = "I am learning Java for Java every day Java"
print(s.replace(" ",  "        "))
```

output

I am learning Java for Java every day Java

```
s = "I am learning Java for Java every day Java"
print(s.replace(" "," SPACE "))
```

output

I SPACE am SPACE learning SPACE Java SPACE for SPACE Java SPACE every SPACE day SPACE Java

---

Real time usage in : when using OS.Path, we get absolute path, relative path, we get(single slash or double back slash (\ or \\). To remove the slash we can use this replace function

```
print(a.replace("\\", "\\\\")) # it is equqvelent to ("\", "\\")
```

output

E:\\PYTHON\\Machine Learning, Scikit-Learn\\MACHINE LEARNING - BOOKS AND VIDEOS\\Data Science & Machine Learning Projects

---

Special characters also can be given as a replaceable string

```
s = "I am learning Java for Java every day Java"
print(s.replace("am","@"))
```

output

I @ learning Java for Java every day Java

==========

If the third argument (Count) is -1, it replaces ALL the values

```
s = "I am learning Java for Java every day Java Java"
print(s.replace("Java",  "MatplotLib",-1))
```

output

I am learning MatplotLib for MatplotLib every day MatplotLib MatplotLib

---

Split()

def split(self,

```
        sep: Optional[str] = ...,  maxsplit: int = ...) -> List[str]

def split(self, *args, **kwargs): # real signature unknown
    """
```

Return a list of the words in the string, using sep as the delimiter string.

sep
The delimiter according which to split the string.
None (the default value) means split according to any whitespace, and discard empty strings from the result.

maxsplit
Maximum number of splits to do.
-1 (the default value) means no limit.

```
    """
    pass
```

Return a list of the words in the string, using sep as the delimiter string.

sep

The delimiter according which to split the string. None (the default value) means split according to any whitespace, and discard empty strings from the result.
maxsplit
Maximum number of splits to do. -1 (the default value) means no limit.

```
s = "I am learning python for every day"
print(s.split())
```
['I', 'am', 'learning', 'python', 'for', 'every', 'day']

```
s = "I am learning python for every day"
print(s.split("python"))
```

['I am learning ', ' for every day']

```
s = "oops#java#python#sql"
print(s.split("#", 1))
```
['oops', 'java#python#sql']

```
s = "oops#java#python#sql"
print(s.split("#", 2))
```
['oops', 'java', 'python#sql']

```
s = "oops#java#python#sql"
print(s.split("#", 3))
 ['oops', 'java', 'python', 'sql']
```

Note: The above three programs using # in a sequence of the string, When the parameter passed on the “#” the int value like 1,2,3 based on remove(#) the given string, and split the sequence to separate strings.

```
st = ' Python for all and for "Data Science and for AI"'

print(st.split())
print(st.split('for',1))
print(st.split('for',2))
print(st.split('for',-1))
print(st.split('a', 2))
```

output

```
['Python', 'for', 'all', 'and', 'for', '"Data', 'Science', 'and', 'for', 'AI"']
[' Python ', ' all and for "Data Science and for AI"']
[' Python ', ' all and ', ' "Data Science and for AI"']
[' Python ', ' all and ', ' "Data Science and ', ' AI"']
[' Python for ', 'll ', 'nd for "Data Science and for AI"']
```

Note:

The output will remove the separator (ie, the string that is used for split, here we use the string 'for' as separator)

Even though many number of 'for' in the above statement, we can decide how many number of split we need using 'maxlimit'

Create code for the following

```
line = 'A lot                of spaces'  # watch the multiple spaces in this string
etc = line.split()
print(etc)
```

output
['A', 'lot', 'of', 'spaces'] # it removes ALL the spaces when spliting

---

```
line = 'first;second;third'  # this one whole string
thing = line.split() #split with None
print(thing)
```

output
['first;second;third']

```
> print(len(thing))
```

1

```
line = 'first;second;third'
thing = line.split(';') #split with ' ; '
print(thing)
print(len(thing))
```

output

```
['first', 'second', 'third']
3
```

- Note: When you do not specify a delimiter, **multiple spaces are treated like one delimiter**
- You can specify what delimiter string to use in the splitting

```
print('we are doing data science'.split('we'))
print('we are doing data science'.split('are'))
print('we are doing data science'.split('science'))
```

output

```
['', ' are doing data science']
['we ', ' doing data science']
['we are doing data ', '']
```

Note: if we split with the first or last substring, one of the output element in the list will have empty string (‘’) – see below

```
print('we are doing data science'.split('science'))
print('science we are doing data science'.split('science'))
```

output

```
['we are doing data ', '']
['', ' we are doing data ', '']
```

==========

Rsplit()*

```
def rsplit(self,
        sep: Optional[str] = ...,
        maxsplit: int = ...) -> List[str]
```

```
def rsplit(self, *args, **kwargs): # real signature unknown
    """
    Return a list of the words in the string, using sep as the delimiter string.

      sep
        The delimiter according which to split the string.
        None (the default value) means split according to any whitespace, and
discard empty strings from the result.
      maxsplit
        Maximum number of splits to do.
        -1 (the default value) means no limit.

    Splits are done starting at the end of the string and working to the front.
    """
    pass
```

Return a list of the words in the string, using sep as the delimiter string.

sep
The delimiter according which to split the string. None (the default value) means split according to any whitespace, and discard empty strings from the result.
maxsplit

Maximum number of splits to do. -1 (the default value) means no limit.
Splits are done starting at the end of the string and working to the front.

The rsplit() method splits a string into a list, starting from the right.

If no "max" is specified, this method will return the same as the split() method.

Note: When maxsplit is specified, the list will contain the specified number of elements *plus one*.

```
s = "I am learning python for every day"
print(s.rsplit())
```
['I', 'am', 'learning', 'python', 'for', 'every', 'day']
Note: The sequence of the given string split on the separate substring.

```
s = "I am learning python for every day"
print(s.rsplit("python"))
```
['I am learning ', ' for every day']
Note: The above example, split for two sub strings based on the specified string, but the specified string('python') is disappeared on the result.

Rsplit(), spilits from right side, based on the number of occurances (default is -1)

```
s = "I am learning Python for Python every day and do lab on Python"
print(s.rsplit("Python", 1))
print(len(s.rsplit("Python", 1)))

# ============
print(s.rsplit("Python", 2))
print(len(s.rsplit("Python", 2)))
```

output

```
['I am learning Python for Python every day and do lab on ', '']
2
['I am learning Python for ', ' every day and do lab on ', '']
3
--------------
s = "We  'are' learning Python"
print(s)
print(s.split('are')) # spliting with 'are' len 3
print(s.split("'are'")) # spliting with 'are' len 5

print(len("'are'"))
print(len('are'))
```

output

```
We  'are' learning Python
["We  '", "' learning Python"]
['We  ', ' learning Python']
5
3
```

```
s = "oops, java, python"
print(s.rsplit(", ", 1)) # comma with space
 ['oops, java', 'python']
```

Note: It is separate for two sub strings.

```
s = "oops,java,python"
print(s.rsplit(", ", 0)) # max count zero
 ['oops, java, python']
```

Note: It is a one sub string.

```
s = "oops,java,python"
s1 = "oops java python"
print(s.rsplit(" ", 0))
print(s1.rsplit(" ", -1))
print(len(s.rsplit(" ",)))
print(len(s1.rsplit(" ",)))
```

output
['oops,java,python']
['oops', 'java', 'python']
1
3

Note: join() vs split()
Joins makes the seq to single string, where as split() does the opposite

---

The Double Split Pattern

Sometimes we split a line one way, and then grab one of the pieces of the line and split that piece again

```
abc ='From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008'
splitedSubString=str.split(abc)
print(splitedSubString)
```

```
print (len(splitedSubString))
print (splitedSubString[1])
```

output

```
['From', 'stephen.marquard@uct.ac.za', 'Sat', 'Jan', '5', '09:14:16', '2008']
7
stephen.marquard@uct.ac.za
```

==========

Splitlines()

```
def splitlines(self, keepends: bool = ...) -> List[str]

def splitlines(self, *args, **kwargs): # real signature unknown
    """
    Return a list of the lines in the string, breaking at line boundaries.

    Line breaks are not included in the resulting list unless keepends is given and
    true.
```

```
    """
    pass
```

Return a list of the lines in the string, breaking at line boundaries.
Line breaks are not included in the resulting list unless keepends is given and true.

The splitlines() method splits a string into a list. The splitting is done at line breaks.

```
s = "I am learning\npython for every day"
print(s.splitlines())
```

```
['I am learning', 'python for every day']
```

Note: Split a string into a list where each line is a list item.

```
s = "I am learning\npython for every day"
print(s.splitlines(True))
```

```
['I am learning\n', 'python for every day']
```

Note: Split the string, but keep the line breaks.

```
s = "I am learning\npython for every day"
print(s.splitlines(False))
 ['I am learning', 'python for every day']
```

Note: : Split a string into a list where each line is a list item.

```
s = 'monty 123 python'
print (s.splitlines())
 ['monty 123 python']
```

Startswith()

```
def startswith(self,
               prefix: Union[str, Tuple[str, ...]],
               start: Optional[int] = ...,
               end: Optional[int] = ...) -> bool

ef startswith(self, prefix, start=None, end=None): # real signature unknown;
restored from __doc__
    """
    S.startswith(prefix[, start[, end]]) -> bool

    Return True if S starts with the specified prefix, False otherwise.
```

```
    With optional start, test S beginning at that position.
    With optional end, stop comparing S at that position.
    prefix can also be a tuple of strings to try.
    """
    return False
```

S.startswith(prefix[, start[, end]]) -> bool
Return True if S starts with the specified prefix, False otherwise. With optional start, test S beginning at that position. With optional end, stop comparing S at that position. prefix can also be a tuple of strings to try.

```
s = "I am learning python for every day"
print(s.startswith("I"))
```

True

Note: If the specified string start with prefix of first character gets the output True, otherwise false.

```
s = "I am learning python for every day"
print(s.startswith("a"))
```

False

```
s = "I am learning python for every day"
print(s.startswith("a",2,6))
```

True

Note: The specified string ‘a’ not to be a first letter in a given string, but the specified int value 2,6, so that the sequence assumed the start letter in 2nd position(int value 2), if check the second position in ‘a’ the condition is True, otherwise False.

```
s = "I am learning python for every day"
print(s.startswith("I am")) # True

s = "13 am learning python for every day"
print(s.startswith("13")) # True

s = " I am learning python for every day"
print(s.startswith(" ")) # True # the sentence starts with a space

s = "I am learning python for every day"
print(s.startswith("I am ")) # True "I am with a SPACE
```

```
s = 'monty 123 python'
print (s.startswith())
 TypeError: startswith() takes at least 1 argument (0 given)
=====================
```

Strip()

```
def strip(self, __chars: Optional[str] = ...) -> str

def strip(self, *args, **kwargs): # real signature unknown
    """
    Return a copy of the string with leading and trailing whitespace removed.

    If chars is given and not None, remove characters in chars instead.
    """
    pass
```

Return a copy of the string with leading and trailing whitespace removed.
If chars is given and not None, remove characters in chars instead.

The **strip**() method returns a copy of the string by removing both the leading and the trailing characters (based on the string argument passed). The **strip**() method removes characters from both left and right based on the argument (a string specifying the set of characters to be removed).

```
s = "      python          "
print("I am learning" ,s.strip(),"for everyday")
```

I am learning python for everyday

Note: This method used to strip both left and right of the string white spaces are removed.

```
s = ".....,,,pythonttttttt"
print(s.strip(".,t"))
```

python

Note: This method used to strip both left and right of the specified string characters are removed.

```
s = 'monty 123 python'
print (s.strip())
```

monty 123 python

```
s = "a.....,,,pythonttttttttaa"
print(s.strip("a.,t"))  # strip string  are in order,  it works

s = "a.....,,,pythonttttttttaa"
print(s.strip("at."))

s = "a.....,,,pythonttttttttaa"
print(s.strip("at.,")) # strip string  are jumpled, still it works

s = "a.....,,,PYthonttttttttaa"
print(s.strip("at.,PY")) # strip string  are jumpled, still it works

s = "a.....,,,Javatttttttaa..."
print(s.strip("ta.,@")) # Jav

s = "    Data Science in Tamil     "
print(len(s))
print(len(s.strip())) # by default the strip removes the space/s in both sides of the string
```

output

```
python
,,,python
python
hon
```

Jav
28
21

Swapcase()

```
def swapcase(self) -> str

def swapcase(self, *args, **kwargs): # real signature unknown
    """ Convert uppercase characters to lowercase and lowercase characters to uppercase. """
    pass
```

Convert uppercase characters to lowercase and lowercase characters to uppercase.

```
s = "i am learning python for every day"
print(s.swapcase())
```

```
 I AM LEARNING PYTHON FOR EVERY DAY
--------------
```

```
s = "I AM LEARNING PYTHON FOR EVERY DAY"
print(s.swapcase())
```

```
 i am learning python for every day
```

```
s = "i am learning PYTHON FOR EVERY DAY"
print(s.swapcase())
```

```
 I AM LEARNING python for every day
```

```
s = 'monty 123 python'
print (s.swapcase())
```

```
 MONTY 123 PYTHON
```

Note: This method is used to convert uppercase characters to lowercase and lowercase characters to uppercase.

==========

Translate()

To create dummy values in the exam paper we use this method
This method needs maketrans().

Translate() always need a maketrans(str1, str2) output

```
def translate(self, __table: Union[Mapping[int, Union[int, str, None]],
Sequence[Union[int, str, None]]])
  -> str

def translate(self, *args, **kwargs): # real signature unknown
    """
    Replace each character in the string using the given translation table.

      table
        Translation table, which must be a mapping of Unicode ordinals to
        Unicode ordinals, strings, or None.

    The table must implement lookup/indexing via __getitem__, for instance a
    dictionary or list.  If this operation raises LookupError, the character is
```

```
    left untouched.  Characters mapped to None are deleted.
    """
    pass
```

Replace each character in the string using the given translation table.

table
Translation table, which must be a mapping of Unicode ordinals to Unicode ordinals, strings, or None.
The table must implement lookup/indexing via __getitem__, for instance a dictionary or list. If this operation raises LookupError, the character is left untouched. Characters mapped to None are deleted.

Description

Python string method **translate()** returns a copy of the string in which all characters have been translated using *table* (constructed with the maketrans() function in the string module), optionally deleting all characters found in the string *deletechars*.

Python String translate()

The string translate() method returns a string where each character is mapped to its corresponding character in the translation table.

translate() method takes the translation table to replace/translate characters in the given string as per the mapping table.
The translation table is created by the static method maketrans().
Refer : https://www.programiz.com/python-programming/methods/string/translate

---

Description

The **translate()** method returns a copy of the string in which all characters have been translated using *table* (constructed with the maketrans() function in the string module), optionally deleting all characters found in the string *deletechars*.

Syntax

Following is the syntax for **translate()** method −

str.translate(table[, deletechars]);

Parameters

- **table** − You can use the maketrans() helper function in the string module to create a translation table.

Return Value

This method returns a translated copy of the string.

```
str1 = 'python'
str2 = '123456'

maktran = str.maketrans(str1, str2)
print(maktran)

str3 = "i am Learning python"
print(str3.translate(maktran))
```

output

```
{112: 49, 121: 50, 116: 51, 104: 52, 111: 53, 110: 54}
i am Lear6i6g 123456
```

Example :

```
str1 = 'aeiou'
str2 = '12345'
maktran = str.maketrans(str1, str2)
# print(maktran)
str3 = "i am Learning python"
print(str3.translate(maktran))
```

Output:

3 1m L21rn3ng pyth4n

Note: First need for maketrans() table, when using the translate().

```
str1 = 'aeiou'
str2 = '12345'
maktran = str.maketrans(str1, str2)
print(maktran)
str3 = "iamLearningpython"
print(str3.translate(maktran))
output
{97: 49, 101: 50, 105: 51, 111: 52, 117: 53}
31mL21rn3ngpyth4n
```

```
str1 = 'ae iou' # key is SPACE
str2 = '12#345' # corresponding value to space is #
maktran = str.maketrans(str1, str2)
```

```
print(maktran)
str3 = "iamL earningpython"
print(str3.translate(maktran))
```

output

{97: 49, 101: 50, 32: 35, 105: 51, 111: 52, 117: 53}

31mL#21rn3ngpyth4n

```
str1 = 'aeiouxm'
str2 = '1234512'
maktran = str.maketrans(str1, str2)
# print(maktran)
str3 = "this is string example"
print(str3.translate(maktran))
```

Output:

th3s 3s str3ng 2112pl2

Note: The same numerical string is assign for different alphabetic character.

Ref: https://www.tutorialspoint.com/python3/string_translate.htm

```
# first string
firstString = "abc"
secondString = "ghi"
thirdString = "ab"

string = "abcdef"
print("Original string:", string)

translation = string.maketrans(firstString, secondString, thirdString)
print(translation)
# translate string
print("Translated string:", string.translate(translation))
```

```
 Original string: abcdef
{97: None, 98: None, 99: 105}
Translated string: idef
```

```
s = 'monty 123 python'
print (s.translate())
```

```
 TypeError: translate() takes exactly one argument (0 given)
```

What will be the output of the following Python code snippet?

```
print('abcd'.translate('a'.maketrans('abc', 'bcd')))
```

a) bcde
b) abcd
c) error
d) bcdd

Answer: d
Explanation: The output is bcdd since no translation is provided for d.

The above code can be re written as below for understanding purpose

```
str1 = 'abc'
str2 = 'bcd'
makeTrans = 'a'.maketrans(str1, str2)
print(makeTrans)
c = 'abcdXY'.translate(makeTrans)
print(c)
```

output
{97: 98, 98: 99, 99: 100}
bcddXY
Note: translate() takes the str1 and replace with ‘str2’

See below for detailed explanation
str1 ‘a’ == str2 ‘b’
str1 ‘b’ == str2 ‘c’
str1 ‘c’ == str2 ‘d’
use the above values to map with ‘abcdXY’ asbelow
‘a’ maps with ‘b’ (b)
‘b’ maps with ‘c’ (bc)
‘c’ maps with ‘d’ (bcd)
‘d’ no maping value available, so it takes same value ‘d’ (bcdd)
‘X’ no maping value available, so it takes same value ‘X’ (bcddX)
‘Y’ no maping value available, so it takes same value ‘Y’ (bcddXY)
The final output is bcddXY

What will be the output of the following Python code snippet?

```
print('abcd'.translate({'a': '1', 'b': '2', 'c': '3', 'd': '4'}))
```

a) abcd
b) 1234
c) error
d) none of the mentioned

Answer: a
Explanation: The function translate expects a dictionary of integers. Use maketrans() instead of doing the above.
The above qn shoud be as below

```
print('abcd'.translate({100: 98, 97: 99, 103: 100, 104: 101}))
```

output
cbcb
Note: it checks only the keys in the dict and maps the corresponding value

How to get back original value

```
str1 = "aeiou"
str2 = "12345"

mk = str.maketrans(str1,str2)
print(mk)
str = "i am learning python"

tran = str.translate(mk)
```

```
print("After translate: ", tran)

mk1   = str.maketrans(str2,str1) # watch, the variables are interchanged
print("Again translate: ", tran.translate(mk1))
```

output

```
{97: 49, 101: 50, 105: 51, 111: 52, 117: 53}
After translate:  3 1m l21rn3ng pyth4n
Again translate:  i am learning python
```

Zfill()

```
def zfill(self, __width: int) -> str

def zfill(self, *args, **kwargs): # real signature unknown
    """
    Pad a numeric string with zeros on the left, to fill a field of the given width.

    The string is never truncated.
```

```
    """
    pass
```

Pad a numeric string with zeros on the left, to fill a field of the given width. The string is never truncated.

The zfill() method adds zeros (0) at the beginning of the string, until it reaches the specified length.

If the value of the len parameter is less than the length of the string, no filling is done.

```
s = "25"
print(s.zfill(5))
```

```
00025
```

Note: Fill the string with zeros until it is 5 characters long.

```
s = "255"
print(s.zfill(2))
```

```
255
```

Note: If the value of the len parameter is less than the length of the string, no filling is done.

```
a = "hello"
b = "welcome to Python"
c = "10.000"

print(a.zfill(10))
print(b.zfill(10))
print(c.zfill(10))
```

```
00000hello
    welcome to Python
    000010.000
```

What will be the output of the following Python code snippet?

```
print('+99'.zfill(5))
```

a) 00+99
b) 00099

c) +0099
d) +++99

Answer: c
Explanation: zeros are filled in between the first sign and the rest of the string. (sign is applicable only for + and -)

```
print('+99'.zfill(5))
print('-99'.zfill(5))
print('*99'.zfill(5))
print('&99'.zfill(5))
print('+999'.zfill(2))
```

output

```
+0099
-0099
00*99
00&99
```

```
print('+99'.zfill(5))
print('-99'.zfill(5))
print('A99'.zfill(5))
print('-A99'.zfill(5))
print('*99'.zfill(5))
```

output

```
+0099
-0099
00A99
-0A99
00*99
```

```
print('4+7j'.zfill(8))
print('A-99'.zfill(8))
```

Casefold()

Using of caseless comparisions

lower() vs casefold() and when to use, Details are given below.

"givenstring".lower()	"givenstring".casefold()
it identifies the ASCII characters in the "givenstring"on which lower method is applied and convert them to the lower case.	it identifies the Unicode Characters in the "givenstring" on which casefold is applied and covert them to the lower case.
ASCII standard contains 256 characters	Unicode Standard contains **143,859 characters**.
It's less effective while comparing two lower strings, as it can only lower 256 type of characters.	It's more effective while comparing two strings. As it has wide range of character which can be converted to lowercase.

The casefold() method is similar to the lower() method but it is more aggressive. This means the casefold() method converts more characters into lower case compared to lower() . For example, the German letter ß is already lowercase so, the lower() method doesn't make the conversion

def casefold(self) -> str

```
def casefold(self, *args, **kwargs): # real signature unknown
    """ Return a version of the string suitable for caseless comparisons. """
    pass
```

Return a version of the string suitable for caseless comparisons.

```
s = 'mel@gmail.com'
a= (s.casefold())
print(a)

s1 = 'MEL@GMAIL.COM'
b= (s1.casefold())
print(b)

print(s==s1)
print(a==b)
```

output

mel@gmail.com

mel@gmail.com

False

True

Note: The upper case string is converted to lower case, and the lowercase string is going to be as it is ..

To check the caseless comparisions we can can make use this method

```
text = "groß"

# convert text to lowercase using casefold()
print("Using casefold():", text.casefold())

# convert text to lowercase using lower()
print("Using lower():", text.lower())
```

output

Using casefold(): gross

Using lower(): groß

---

firstString = "der Fluß"

secondString = "der Fluss"

ß is equivalent to ss

```
if firstString.casefold() == secondString.casefold():
    print('The strings are equal.')
else:
    print('The strings are not equal.')
---------------
```

Encode()

Need more examples

```
def encode(self,
           encoding: str = ...,
           errors: str = ...) -> bytes

def encode(self, *args, **kwargs): # real signature unknown
    """
    Encode the string using the codec registered for encoding.

      encoding
        The encoding in which to encode the string.
      errors
```

```
        The error handling scheme to use for encoding errors.
        The default is 'strict' meaning that encoding errors raise a
        UnicodeEncodeError.  Other possible values are 'ignore', 'replace'
and
        'xmlcharrefreplace' as well as any other name registered with
        codecs.register_error that can handle UnicodeEncodeErrors.
    """
    pass
```

Encode the string using the codec registered for encoding.

encoding
The encoding in which to encode the string.
errors
The error handling scheme to use for encoding errors. The default is 'strict' meaning that encoding errors raise a UnicodeEncodeError. Other possible values are 'ignore', 'replace' and 'xmlcharrefreplace' as well as any other name registered with codecs.register_error that can handle UnicodeEncodeErrors.

The string **encode**() method returns encoded version of the given string. Since **Python** 3.0, strings are stored as Unicode, i.e. each character in

the string is represented by a code point. So, each string is just a sequence of Unicode code points.(Refer w3schools.com)

```
s = 'monty 123 python'
print (s.encode())
```

```
b'monty 123 python'
```

==========

String with Quotes

```
a= "hi 'how' are '''you''' Python" #if a string in double quote, it accepts single and triple
print(a)

# a='hi "how" are "'you'" Python' # if a string in single quote, it accepts only double quote
# print(a)
# #
a='''hi 'how' are ''you'' Python''' #if a string in treple  quote, it accepts single and double
print(a)


a='''\'hi 'how' are ''you'' Python\'  '''
print(a)
```

```
a='''\"hi 'how' are ''you'' Python\"  '''
print(a)

print('''    "  Deebwali 2022 Oct 23  "       ''')
```

output

```
hi 'how' are '''you''' Python
hi 'how' are '''you''' Python
hi 'how' are ''you'' Python
```