Python String Methods

Python string is a sequence of Unicode characters that is enclosed in the quotations marks. In this article, we will discuss the in-built function i.e. the functions provided by the Python to operate on strings.

Note: Every string method does not change the original string instead returns a new string with the changed attributes.

Case Changing of Strings

The below functions are used to change the case of the strings.

- **lower():** Converts all uppercase characters in a string into lowercase
- **upper():** Converts all lowercase characters in a string into uppercase
- **title():** Convert string to title case

Example: Changing the case of Python Strings.

```
# Python3 program to show the
# working of upper() function
text ='geeKs For geEkS'

# upper() function to convert
# string to upper case
print("\nConverted String:")
print(text.upper())

# lower() function to convert
# string to lower case
print("\nConverted String:")
print(text.lower())

# converts the first character to
# upper case and rest to lower
case
print("\nConverted String:")
print(text.title())

# original string never changes
print("\nOriginal String")
print(text)
```

Output:

```
Converted String:
GEEKS FOR GEEKS


Converted String:
geeks for geeks


Converted String:
Geeks For Geeks


Original String
geeKs For geEkS
```

Table of Python String Methods

Function Name	Description
capitalize()	Converts the first character of the string to a capital (uppercase) letter
casefold()	Implements caseless string matching
center()	Pad the string with the specified character.
count()	Returns the number of occurrences of a substring in the string.
encode()	Encodes strings with the specified encoded scheme

endswith()	Returns “True” if a string ends with the given suffix
expandtabs()	Specifies the amount of space to be substituted with the “\t” symbol in the string
find()	Returns the lowest index of the substring if it is found
format()	Formats the string for printing it to console
format_map()	Formats specified values in a string using a dictionary
index()	Returns the position of the first occurrence of a substring in a string
isalnum()	Checks whether all the characters in a given string is alphanumeric or not
isalpha()	Returns “True” if all characters in the string are alphabets
isdecimal()	Returns true if all characters in a string are decimal
isdigit()	Returns “True” if all characters in the string are digits
isidentifier()	Check whether a string is a valid identifier or not
islower()	Checks if all characters in the string are lowercase
isnumeric()	Returns “True” if all characters in the string are numeric characters
isprintable()	Returns “True” if all characters in the string are printable or the string is empty
isspace()	Returns “True” if all characters in the string are whitespace characters
istitle()	Returns “True” if the string is a title cased string
isupper()	Checks if all characters in the string are uppercase
join()	Returns a concatenated String
ljust()	Left aligns the string according to the width specified
lower()	Converts all uppercase characters in a string into lowercase
lstrip()	Returns the string with leading characters removed
maketrans()	Returns a translation table
partition()	Splits the string at the first occurrence of the separator
replace()	Replaces all occurrences of a substring with another substring
rfind()	Returns the highest index of the substring
rindex()	Returns the highest index of the substring inside the string
rjust()	Right aligns the string according to the width specified
rpartition()	Split the given string into three parts
rsplit()	Split the string from the right by the specified separator
rstrip()	Removes trailing characters
splitlines()	Split the lines at line boundaries
startswith()	Returns “True” if a string starts with the given prefix
strip()	Returns the string with both leading and trailing characters
swapcase()	Converts all uppercase characters to lowercase and vice versa
title()	Convert string to title case
translate()	Modify string according to given translation mappings
upper()	Converts all lowercase characters in a string into uppercase
zfill()	Returns a copy of the string with ‘0’ characters padded to the left side of the string

Note: For more information about Python Strings, refer to Python String Tutorial.

string capitalize() in Python

In Python, the **capitalize()** method returns a copy of the original string and converts the first character of the string to a capital **(uppercase)** letter while making all other characters in the string **lowercase** letters.

Syntax:

string_name.capitalize()

string_name:

```
It is the name of string of

whose first character we want

to capitalize.
```

Parameter: The capitalize() function does not takes any parameter.
Return value: The capitalize() function returns a string with the first character in the capital.
Below is the python program to illustrate capitalize() function:

```
# Python program to demonstrate the
# use of capitalize() function

# capitalize() first letter of string
# and make other letters lowercase
name ="geeks FOR geeks"

print(name.capitalize())

# demonstration of individual words
# capitalization to generate camel case
name1 ="geeks"
name2 ="for"
name3 ="geeks"
print(name1.capitalize() +name2.capitalize()
                         +name3.capitalize()
)
```

Output:

```
Geeks for geeks

GeeksForGeeks
```

Python String casefold() Method

Python String **casefold()** method is used to implement caseless string matching. It is similar to lower() string method but case **removes all the case distinctions** present in a string. i.e ignore cases when comparing.

Syntax:

string.casefold()

Parameters:

The casefold() method doesn’t take any parameters.

Return value:

It returns the case folded string the string converted to lower case.

Example 1: Convert String in Lower Case

```
# Python program to convert string in lower
case
string ="GEEKSFORGEEKS"

# print lowercase string
print("lowercase string: ",string.casefold())
```

Output:

```
lowercase string: geeksforgeeks
```

Example 2: Check if a string is palindrome

```
# Program to check if a string
#  is palindrome or not

# change this value for a different output
str='geeksforgeeks'

# make it suitable for caseless comparison
str=str.casefold()

# reverse the string
rev_str ="".join(reversed(str))
# check if the string is equal to its
reverse
ifstr==rev_str:
      print("palindrome")
else:
      print("not palindrome")
```

Output:

```
not palindrome
```

Example 3: Count Vowels in a String

```
# Program to count
# the number of each
# vowel in a string

# string of vowels
v ='aeiou'

# change this value for a different
result
str='Hello, have you try geeksforgeeks?'

# input from the user
# str = input("Enter a string: ")

# caseless comparison
str=str.casefold()

# make a dictionary with
# each vowel a key and value 0
c ={}.fromkeys(v,0)

# count the vowels
forchar instr:
          ifchar inc:
                  c[char] +=1
print(c)
```

Output:

```
{'a': 1, 'e': 6, 'i': 0, 'o': 3, 'u': 1}
```

Python String center() Method

Python String **center()** method creates and returns a new string that is padded with the specified character.

Syntax:

string.center(length[, fillchar])

Parameters:

- **length:** length of the string after padding with the characters.
- **fillchar:** (optional) characters which need to be padded. If it’s not provided, space is taken as the default argument.

Returns:

Returns a string padded with specified fillchar and it doesn’t modify the original string.

Example 1: center() Method With Default fillchar

```
# Python program to illustrate
# string center() in python
string ="geeks for geeks"

new_string =string.center(24)

# here fillchar not provided so takes space by
default.
print"After padding String is: ", new_string
```

Output:

```
After padding String is:      geeks for geeks
```

Example 2: center() Method With # fillchar

```
# Python program to illustrate
# string center() in python
string ="geeks for geeks"

new_string =string.center(24, '#')

# here fillchar is provided
print"After padding String is:",
new_string
```

Output:

```
After padding String is: ####geeks for geeks#####
```

Python String count() Method

Python String **count()** function is an inbuilt function in python programming language that returns the number of occurrences of a substring in the given string.

Syntax:

string.count(substring, start=…, end=…)

Parameters:

- The count() function has one compulsory and two optional parameters.
- **Mandatory parameter:**
- substring – string whose count is to be found.
- **Optional Parameters:**
- *start (Optional)* – starting index within the string where the search starts.
- *end (Optional)* – ending index within the string where the search ends.

Return Value:

count() method returns an integer that denotes **number of times** a substring occurs in a given string.

Example 1: Implementation of the count() method without optional parameters

```
# Python program to demonstrate the use of
# count() method without optional parameters

# string in which occurrence will be checked
string ="geeks for geeks"

# counts the number of times substring occurs
in
# the given string and returns an integer
print(string.count("geeks"))
```

Output:

```
2
```

Example 2: Implementation of the count() method using optional parameters

```
# Python program to demonstrate the use of
# count() method  using optional parameters

# string in which occurrence will be checked
string ="geeks for geeks"

# counts the number of times substring occurs in
# the given string between index 0 and 5 and
returns
# an integer
```

```
print(string.count("geeks", 0, 5))

print(string.count("geeks", 0, 15))
```

Output:

```
1

2
```

Python Strings encode() method

In today's world, security holds the key in many applications. Thus the need for secure storage of passwords in the database is required and hence there is to save encoded versions of strings. To achieve this, python in its language has defined "**encode()**" that encodes strings with the specified encoding scheme. There are several encoding schemes defined in the language. The Python String **encode()** method encodes the string, using the specified encoding. If no encoding is specified, UTF-8 will be used.

Syntax:

encode(encoding, error)

Parameters:

- **encoding:** Specifies the encoding on the basis of which encoding has to be performed.
- **error:** Decides how to handle the errors if they occur, e.g 'strict' raises Unicode error in case of exception and 'ignore' ignores the errors that occurred. There are six types of error response
- strict – default response which raises a UnicodeDecodeError exception on failure
- ignore – ignores the unencodable unicode from the result
- replace – replaces the unencodable unicode to a question mark ?
- xmlcharrefreplace – inserts XML character reference instead of unencodable unicode
- backslashreplace – inserts a \uNNNN escape sequence instead of unencodable unicode
- namereplace – inserts a \N{...} escape sequence instead of unencodable unicode

Returns:

Returns the string in the encoded form

Example 1: Code to print encoding schemes available

```
# Python3 code to print
# all encodings available

fromencodings.aliases importaliases

# Printing list available
print("The available encodings are :
")
print(aliases.keys())
```

Output:

```
The available encodings are :

dict_keys(['ibm039', 'iso_ir_226', '1140', 'iso_ir_110', '1252',
'iso_8859_8', 'iso_8859_3', 'iso_ir_166', 'cp367', 'uu', 'quotedprintable',
'ibm775', 'iso_8859_16_2001', 'ebcdic_cp_ch', 'gb2312_1980', 'ibm852', 'uhc',
'macgreek', '850', 'iso2022jp_2', 'hz_gb_2312', 'elot_928', 'iso8859_1',
'eucjp', 'iso_ir_199', 'ibm865', 'cspc862latinhebrew', '863', 'iso_8859_5',
'latin4', 'windows_1253', 'csisolatingreek', 'latin5', '855', 'windows_1256',
'rot13', 'ms1361', 'windows_1254', 'ibm863', 'iso_8859_14_1998', 'utf8_ucs2',
'500', 'iso8859', '775', 'l7', 'l2', 'gb18030_2000', 'l9', 'utf_32be',
'iso_ir_100', 'iso_8859_4', 'iso_ir_157', 'csibm857', 'shiftjis2004',
'iso2022jp_1', 'iso_8859_2_1987', 'cyrillic', 'ibm861', 'ms950', 'ibm437',
'866', 'csibm863', '932', 'iso_8859_14', 'cskoi8r', 'csptcp154', '852',
'maclatin2', 'sjis', 'korean', '865', 'u32', 'csshiftjis', 'dbcs',
'csibm037', 'csibm1026', 'bz2', 'quopri', '860', '1255', '861', 'iso_ir_127',
'iso_celtic', 'chinese', 'l8', '1258', 'u_jis', 'cspc850multilingual',
'iso_2022_jp_2', 'greek8', 'csibm861', '646', 'unicode_1_1_utf_7', 'ibm862',
'latin2', 'ecma_118', 'csisolatinarabic', 'zlib', 'iso2022jp_3', 'ksx1001',
'858', 'hkscs', 'shiftjisx0213', 'base64', 'ibm857', 'maccentraleurope',
'latin7', 'ruscii', 'cp_is', 'iso_ir_101', 'us_ascii', 'hebrew',
'ansi_x3.4_1986', 'csiso2022jp', 'iso_8859_15', 'ibm860', 'ebcdic_cp_us',
'x_mac_simp_chinese', 'csibm855', '1250', 'maciceland', 'iso_ir_148',
'iso2022jp', 'u16', 'u7', 's_jisx0213', 'iso_8859_6_1987',
'csisolatinhebrew', 'csibm424', 'quoted_printable', 'utf_16le', 'tis260',
'utf', 'x_mac_trad_chinese', '1256', 'cp866u', 'jisx0213', 'csiso58gb231280',
'windows_1250', 'cp1361', 'kz_1048', 'asmo_708', 'utf_16be', 'ecma_114',
'eucjis2004', 'x_mac_japanese', 'utf8', 'iso_ir_6', 'cp_gr', '037',
'big5_tw', 'eucgb2312_cn', 'iso_2022_jp_3', 'euc_cn', 'iso_8859_13',
'iso_8859_5_1988', 'maccyrillic', 'ks_c_5601_1987', 'greek', 'ibm869',
'roman8', 'csibm500', 'ujis', 'arabic', 'strk1048_2002', '424',
'iso_8859_11_2001', 'l5', 'iso_646.irv_1991', '869', 'ibm855', 'eucjisx0213',
'latin1', 'csibm866', 'ibm864', 'big5_hkscs', 'sjis_2004', 'us',
'iso_8859_7', 'macturkish', 'iso_2022_jp_2004', '437', 'windows_1255',
's_jis_2004', 's_jis', '1257', 'ebcdic_cp_wt', 'iso2022jp_2004', 'ms949',
'utf32', 'shiftjis', 'latin', 'windows_1251', '1125', 'ks_x_1001',
'iso_8859_10_1992', 'mskanji', 'cyrillic_asian', 'ibm273', 'tis620', '1026',
'csiso2022kr', 'cspc775baltic', 'iso_ir_58', 'latin8', 'ibm424',
'iso_ir_126', 'ansi_x3.4_1968', 'windows_1257', 'windows_1252', '949',
'base_64', 'ms936', 'csisolatin2', 'utf7', 'iso646_us', 'macroman', '1253',
'862', 'iso_8859_1_1987', 'csibm860', 'gb2312_80', 'latin10', 'ksc5601',
'iso_8859_10', 'utf8_ucs4', 'csisolatin4', 'ebcdic_cp_be', 'iso_8859_1',
'hzgb', 'ansi_x3_4_1968', 'ks_c_5601', 'l3', 'cspc8codepage437',
'iso_8859_7_1987', '8859', 'ibm500', 'ibm1026', 'iso_8859_6', 'csibm865',
'ibm866', 'windows_1258', 'iso_ir_138', 'l4', 'utf_32le', 'iso_8859_11',
'thai', '864', 'euc_jis2004', 'cp936', '1251', 'zip', 'unicodebigunmarked',
'csHPRoman8', 'csibm858', 'utf16', '936', 'ibm037', 'iso_8859_8_1988', '857',
'csibm869', 'ebcdic_cp_he', 'cp819', 'euccn', 'iso_8859_2', 'ms932',
'iso_2022_jp_1', 'iso_2022_kr', 'csisolatin6', 'iso_2022_jp', 'x_mac_korean',
'latin3', 'csbig5', 'hz_gb', 'csascii', 'u8', 'csisolatin5',
'csisolatincyrillic', 'ms_kanji', 'cspcp852', 'rk1048', 'iso2022jp_ext',
'csibm273', 'iso_2022_jp_ext', 'ibm858', 'ibm850', 'sjisx0213',
'tis_620_2529_1', 'l10', 'iso_ir_109', 'ibm1125', '1254', 'euckr',
'tis_620_0', 'l1', 'ibm819', 'iso2022kr', 'ibm367', '950', 'r8', 'hex',
'cp154', 'tis_620_2529_0', 'iso_8859_16', 'pt154', 'ebcdic_cp_ca', 'ibm1140',
'l6', 'csibm864', 'csisolatin1', 'csisolatin3', 'latin6', 'iso_8859_9_1989',
'iso_8859_3_1988', 'unicodelittleunmarked', 'macintosh', '273', 'latin9',
'iso_8859_4_1988', 'iso_8859_9', 'ebcdic_cp_nl', 'iso_ir_144'])
```

Example 2: Code to encode the string

```
# Python code to demonstrate
# encode()

# initializing string
str="geeksforgeeks"

# printing the encoded string
print("The encoded string in utf8 format is :
",)
print(str.encode('utf8', 'ignore'))
```

Output:

```
The encoded string in utf8 format is :

b'geeksforgeeks'
```

Example 3: Encoding with the error parameter

```
# unicode string
string ='GeeksforGeeks'

# print string
print('The string is:', string)

# ignore error
print(string.encode("ascii", "ignore"))

# replace error
print(string.encode("ascii",
"replace"))
```

Output:

```
The string is: GeeksforGeeks

b'GeeksforGeeks'

b'GeeksforGeeks'
```

Python String endswith() Method

Python String endswith() method returns True if a string ends with the given suffix otherwise returns False.

Syntax:

str.endswith(suffix, start, end)

Parameters:

- **suffix:** Suffix is nothing but a string that needs to be checked.
- **start:** Starting position from where suffix is needed to be checked within the string.
- **end:** Ending position + 1 from where suffix is needed to be checked within the string.

Note: If start and end index is not provided then by default it takes 0 and length -1 as starting and ending indexes where ending index is not included in our search.

Returns:
It returns True if the string ends with the given suffix otherwise return False.

Example 1: Working of endswith() method Without start and end Parameters

```
# Python code shows the working of
# .endswith() function

text ="geeks for geeks."

# returns False
result =text.endswith('for geeks')
print(result)

# returns True
result =text.endswith('geeks.')
print(result)

# returns True
result =text.endswith('for geeks.')
print(result)

# returns True
result =text.endswith('geeks for
geeks.')
print(result)
```

Output:

```
False

True

True
```

```
True
```

Example 2: Working of endswith() method With start and end Parameters

```
# Python code shows the working of
# .endswith() function

text ="geeks for geeks."

# start parameter: 10
result =text.endswith('geeks.', 10)
print(result)

# Both start and end is provided
# start: 10, end: 16 - 1
# Returns False
result =text.endswith('geeks', 10,
16)
printresult

# returns True
result =text.endswith('geeks', 10,
15)
printresult
```

Output:

```
True

True

False
```

expandtabs() method in Python

expandtabs is a method specified in Strings in Python 3.

Sometimes, there is a need of specifying the space in the string, but the amount of space to be left is uncertain and depending upon the environment and conditions. For these cases, the need to modify the string, again and again, is a tedious task. Hence python in its library has “expandtabs()” which specifies the amount of space to be substituted with the “\t” symbol in the string.

Syntax : **expandtabs(space_size)**

Parameters :
space_size : Specifies the space that is to be replaced with the “\t” symbol in the string. By default the space is 8.

Returns : Returns the modified string with tabs replaced by the space.

Code #1 : Code to demonstrate expandtabs()

```
# Python3 code to demonstrate
# working of expandtabs()

# initializing string
str="i\tlove\tgfg"

# using expandtabs to insert spacing
print("Modified string using default spacing: ", end
="")
print(str.expandtabs())

print("\r")

# using expandtabs to insert spacing
print("Modified string using less spacing: ", end ="")
print(str.expandtabs(2))

print("\r")

# using expandtabs to insert spacing
print("Modified string using more spacing: ", end ="")
print(str.expandtabs(12))

print("\r")
```

Output:

```
Modified string using default spacing: i       love    gfg



Modified string using less spacing: i love  gfg
Modified string using more spacing: i           love        gfg
```

Exception :
The exception of using this method is that it doesn’t accept the floating-point number if we want to decide the exact precision of the space we require.

Code #2 : Code to demonstrate exception of expandtabs()

```
# Python3 code to demonstrate
# exception of expandtabs()

# initializing string
st ="i\tlove\tgfg"

# using expandtabs to insert spacing
try:
    print("Modified string using default spacing:
")
    print(st.expandtabs(10.5))

exceptException as e:
    print("Error !! The error occurred is :")
    print(str(e))
```

Output:

```
Modified string using default spacing:

Error !! The error occurred is :

integer argument expected, got float
```

Applications :
There are many possible applications where this can be used such as text formatting or documentation where user requirements keep on changing.

Python String find() method

Python String find() method returns the lowest index of the substring if it is found in a given string. If it is not found then it returns -1.

Syntax:

str.find(sub, start, end)

Parameters:

- **sub:** It's the substring that needs to be searched in the given string.
- **start:** Starting position where the sub needs to be checked within the string.
- **end:** Ending position where suffix needs to be checked within the string.

Note #1: If start and end indexes are not provided then by default it takes 0 and length-1 as starting and ending indexes where ending indexes are not included in our search.

Returns:

Returns the lowest index of the substring if it is found in a given string. If it's not found then it returns -1.

Note #2: The find() method is similar to index(). The only difference is find() returns -1 if the searched string is not found and index() throws an exception in this case.

Example 1: find() With No start and end Argument

```
word ='geeks for geeks'

# returns first occurrence of Substring
result =word.find('geeks')
print("Substring 'geeks' found at index:",
result )

result =word.find('for')
print("Substring 'for ' found at index:", result )

# How to use find()
if(word.find('pawan') !=-1):
    print("Contains given substring ")
else:
    print("Doesn't contains given substring")
```

Output:

```
Substring 'geeks' found at index: 0

Substring 'for ' found at index: 6

Doesn't contains given substring
```

Example 2: find() With start and end Arguments

```
word ='geeks for geeks'

# Substring is searched in 'eks for
geeks'
print(word.find('ge', 2))

# Substring is searched in 'eks for
geeks'
print(word.find('geeks ', 2))

# Substring is searched in 's for g'
print(word.find('g', 4, 10))

# Substring is searched in 's for g'
print(word.find('for ', 4, 11))
```

Output:

```
10

-1

-1

6
```

Python String format() Method

Python format() function has been introduced for handling complex string formatting more efficiently. This method of the built-in string class provides functionality for complex variable substitutions and value formatting. This new formatting technique is regarded as more elegant. The general syntax of format() method is string.format(var1, var2,…)

Using a Single Formatter

Formatters work by putting in one or more replacement fields and placeholders defined by a pair of curly braces **{ }** into a string and calling the str.format(). The value we wish to put into the placeholders and concatenate with the string passed as parameters into the format function.

Syntax : { } .format(value)

Parameters :
(value) : Can be an integer, floating point numeric constant, string, characters or even variables.
Returntype : Returns a formatted string with the value passed as parameter in the placeholder position.

Example 1: Simple demonstration of format()

```
# Python3 program to demonstrate
# the str.format() method

# using format option in a simple string
print("{}, A computer science portal for
geeks."
      .format("GeeksforGeeks"))

# using format option for a
# value stored in a variable
str="This article is written in {}"
print(str.format("Python"))

# formatting a string using a numeric constant
print("Hello, I am {} years old !".format(18))
```

Output :

```
GeeksforGeeks, A computer science portal for geeks.

This article is written in Python

Hello, I am  18 years old!
```

Using Multiple Formatters

Multiple pairs of curly braces can be used while formatting the string. Let's say if another variable substitution is needed in the sentence, can be done by adding a second pair of curly braces and passing a second value into the method. Python will replace the placeholders with values in **order.**

Syntax : { } { } .format(value1, value2)

Parameters : (value1, value2) : Can be integers, floating point numeric constants, strings, characters and even variables. Only difference is, the number of values passed as parameters in format() method must be equal to the number of placeholders created in the string.

Errors and Exceptions :

IndexError : Occurs when string has an extra placeholder, and we didn't pass any value for it in the format() method. Python usually assigns the placeholders with default index in order like *0, 1, 2, 3….* to access the values passed as parameters. So when it encounters a placeholder whose index doesn't have any value passed inside as parameter, it throws IndexError.

Example 2: Python String format() Method IndexError

```
# Python program demonstrating Index error

# Number of placeholders are four but
# there are only three values passed

# parameters in format function.
my_string ="{}, is a {} {} science portal for {}"

print(my_string.format("GeeksforGeeks", "computer",
"geeks"))
```

Output :

```
IndexError: tuple index out of range
```

Example 3: Python String format() with multiple placeholders

```
# Python program using multiple place
# holders to demonstrate str.format() method

# Multiple placeholders in format() function
my_string ="{}, is a {} science portal for {}"
print(my_string.format("GeeksforGeeks", "computer",
"geeks"))

# different datatypes can be used in formatting
print("Hi ! My name is {} and I am {} years old"
      .format("User", 19))

# The values passed as parameters
# are replaced in order of their entry
print("This is {} {} {} {}"
      .format("one", "two", "three", "four"))
```

Output :

```
GeeksforGeeks, is a computer science portal for geeks

Hi! My name is User and I am 19 years old

This is one two three four
```

Formatting Strings using Escape Sequences

You can use two or more specially designated characters within a string to format a string or perform a command. These characters are called escape sequences. An Escape sequence in Python starts with a backslash (\). For example, \n is an escape sequence in which the common meaning of the letter n is literally escaped and given an alternative meaning – a new line.

Escape sequence	Description	Example
\n	Breaks the string into a new line	print('I designed this rhyme to explain in due time\nAll I know')
\t	Adds a horizontal tab	print('Time is a \tvaluable thing')
\\	Prints a backslash	print('Watch it fly by\\as the pendulum swings')
\'	Prints a single quote	print('It doesn\'t even matter how hard you try')
\"	Prints a double quote	print('It is so\"unreal\"')
\a	makes a sound like a bell	print('\a')

Formatters with Positional and Keyword Arguments

When placeholders **{ }** are empty, Python will replace the values passed through str.format() in order.

The values that exist within the str.format() method are essentially **tuple data types** and each individual value contained in the tuple can be called by its index number, which starts with the index number 0. These index numbers can be passed into the curly braces that serve as the placeholders in the original string.

Syntax : {0} {1}.format(positional_argument, keyword_argument)

Parameters : (positional_argument, keyword_argument)
Positional_argument can be integers, floating point numeric constants, strings, characters and even variables.
Keyword_argument is essentially a variable storing some value, which is passed as parameter.

Example 4:

```
# To demonstrate the use of formatters
# with positional key arguments.

# Positional arguments
# are placed in order
print("{0} love {1}!!".format("GeeksforGeeks",
                              "Geeks"))

# Reverse the index numbers with the
# parameters of the placeholders
print("{1} love {0}!!".format("GeeksforGeeks",
                              "Geeks"))


print("Every {} should know the use of {} {} programming and {}"
      .format("programmer", "Open", "Source",
              "Operating Systems"))


# Use the index numbers of the
# values to change the order that
# they appear in the string
print("Every {3} should know the use of {2} {1} programming and
{0}"
      .format("programmer", "Open", "Source", "Operating
Systems"))


# Keyword arguments are called
# by their keyword name
print("{gfg} is a {0} science portal for {1}"
      .format("computer", "geeks", gfg="GeeksforGeeks"))
```

Output :

```
GeeksforGeeks love Geeks!!

Geeks love GeeksforGeeks!!

Every programmer should know the use of Open Source programming and Operating Systems

Every Operating Systems should know the use of Source Open programming and programmer

GeeksforGeeks is a computer science portal for geeks
```

Type Specifying

More parameters can be included within the curly braces of our syntax. Use the format code syntax **{field_name:conversion}**, where *field_name* specifies the index number of the argument to the str.format() method, and conversion refers to the conversion code of the data type.

Example: %s – string conversion via str() prior to formatting

```
print("%20s"%('geeksforgeeks', ))
print("%-20s"%('Interngeeks', ))
print("%.5s"%('Interngeeks', ))
```

Output:

```
       geeksforgeeks

Interngeeks

Inter
```

Example: %c– character

```
type='bug'

result ='troubling'

print('I wondered why the program was %s me.
Then\
it dawned on me it was a %s .' %
      (result, type))
```

Output:

I wondered why the program was me troubling me. Then it dawned on me it was a bug.

Example: %i signed decimal integer and %d signed decimal integer(base-10)

```
match =12000

site ='amazon'

print("%s isso useful. I tried to look\
up mobile andthey had a nice one that cost %d rupees." %(site,
match))
```

Output:

amazon is so useful. I tried to look up mobiles and they had a nice one that cost 12000 rupees

Some another useful Type Specifying

- **%u** unsigned decimal integer
- **%o** octal integer
- **f** – floating point display
- **b** – binary
- **o** – octal
- **%x** – hexadecimal with lowercase letters after 9
- **%X**– hexadecimal with uppercase letters after 9

- **e** – exponent notation

You can also specify formatting symbols. The only change is using a colon (:) instead of %. For example, instead of %s use {:s} and instead of %d use (:d}

Syntax : String {field_name:conversion} Example.format(value)
Errors and Exceptions :
ValueError : Error occurs during type conversion in this method.

Example 5:

```
# Demonstrate ValueError while
# doing forced type-conversions

# When explicitly converted floating-point
# values to decimal with base-10 by 'd'
# type conversion we encounter Value-Error.
print("The temperature today is {0:d} degrees outside !"
      .format(35.567))

# Instead write this to avoid value-errors
''' print("The temperature today is {0:.0f} degrees
outside !"
                                            .format(35.567))'
''
```

Output :

```
ValueError: Unknown format code 'd' for object of type 'float'
```

Example 6 :

```
# Convert base-10 decimal integers
# to floating point numeric constants
print("This site is {0:f}% securely
{1}!!".
      format(100, "encrypted"))

# To limit the precision
print("My average of this {0} was {1:.2f}
%"
      .format("semester", 78.234876))

# For no decimal places
print("My average of this {0} was {1:.0f}
%"
      .format("semester", 78.234876))

# Convert an integer to its binary or
# with other different converted bases.
print("The {0} of 100 is {1:b}"
      .format("binary", 100))

print("The {0} of 100 is {1:o}"
      .format("octal", 100))
```

Output :

```
This site is 100.000000% securely encrypted!!

My average of this semester was 78.23%

My average of this semester was 78%

The binary of 100 is 1100100

The octal of 100 is 144
```

Padding Substitutions or Generating Spaces

Example 7: Demonstration of spacing when strings are passed as parameters

By default, strings are left-justified within the field, and numbers are right-justified. We can modify this by placing an alignment code just following the colon.

< : left-align text in the field **^** : center text in the field **>** : right-align text in the field

```
# To demonstrate spacing when
# strings are passed as parameters
print("{0:4}, is the computer science portal for
{1:8}!"
      .format("GeeksforGeeks", "geeks"))

# To demonstrate spacing when numeric
# constants are passed as parameters.
print("It is {0:5} degrees outside !"
      .format(40))

# To demonstrate both string and numeric
# constants passed as parameters
print("{0:4} was founded in {1:16}!"
      .format("GeeksforGeeks", 2009))


# To demonstrate aligning of spaces
print("{0:^16} was founded in {1:<4}!"
      .format("GeeksforGeeks", 2009))

print("{:*^20s}".format("Geeks"))
```

Output :

```
GeeksforGeeks, is the computer science portal for geeks   !

It is    40 degrees outside!

GeeksforGeeks was founded in             2009!

 GeeksforGeeks   was founded in 2009 !

*******Geeks********
```

Applications

Formatters are generally used to Organize Data. Formatters can be seen in their best light when they are being used to organize a lot of data in a visual way. If we are showing databases to users, using formatters to increase field size and modify alignment can make the output more readable.

Example 8: To demonstrate the organization of large data using format()

```
# which prints out i, i ^ 2, i ^ 3,
#  i ^ 4 in the given range

# Function prints out values
# in an unorganized manner
defunorganized(a, b):
    fori inrange(a, b):
        print(i, i**2, i**3, i**4)

# Function prints the organized set of values
deforganized(a, b):
    fori inrange(a, b):

        # Using formatters to give 6
        # spaces to each set of values
        print("{:6d} {:6d} {:6d} {:6d}"
              .format(i, i **2, i **3, i **4))

# Driver Code
n1 =int(input("Enter lower range :-\n"))
n2 =int(input("Enter upper range :-\n"))

print("------Before Using Formatters-------")

# Calling function without formatters
unorganized(n1, n2)

print()
print("-------After Using
Formatters---------")
print()

# Calling function that contains
# formatters to organize the data
organized(n1, n2)
```

Output :

```
Enter lower range :-

3

Enter upper range :-

10

------Before Using Formatters-------
```

```
3 9 27 81

4 16 64 256

5 25 125 625

6 36 216 1296

7 49 343 2401

8 64 512 4096

9 81 729 6561


-------After Using Formatters---------


     3      9     27     81

     4     16     64    256

     5     25    125    625

     6     36    216   1296

     7     49    343   2401

     8     64    512   4096

     9     81    729   6561
```

Using a dictionary for string formatting

Using a dictionary to unpack values into the placeholders in the string that needs to be formatted. We basically use ****** to unpack the values. This method can be useful in string substitution while preparing an SQL query.

```
introduction ='My name is {first_name} {middle_name} {last_name} AKA the
{aka}.'
full_name ={
    'first_name': 'Tony',
    'middle_name': 'Howard',
    'last_name': 'Stark',
    'aka': 'Iron Man',
}

# Notice the use of "**" operator to unpack the values.
print(introduction.format(**full_name))
```

Output:

```
My name is Tony Howard Stark AKA the Iron Man.
```

Python format() with list

Given a list of float values, the task is to truncate all float values to 2-decimal digits. Let’s see the different methods to do the task.

```
# Python code to truncate float
# values to 2 decimal digits.

# List initialization
Input=[100.7689454, 17.232999, 60.98867,
300.83748789]

# Using format
Output =['{:.2f}'.format(elem) forelem inInput]

# Print output
print(Output)
```

Output:

```
['100.77', '17.23', '60.99', '300.84']
```

Python String format_map() Method

Python String***format_map()*** method is an inbuilt function in Python, which is used to return a dictionary key’s value.

Syntax:

string.format_map(z)

Parameters:

Here z is a variable in which the input dictionary is stored and string is the key of the input dictionary. input_dict: Takes a single parameter which is the input dictionary.

Returns:

Returns key’s values of the input dictionary.

Example 1: Python String format_map() method

```
# input stored in variable a.
a ={'x':'John', 'y':'Wick'}

# Use of format_map() function
print("{x}'s last name is
{y}".format_map(a))
```

Output:

```
John's last name is Wick
```

Example 2:

```
# input stored in variable a.
a ={'x':"geeksforgeeks",
'y':'b'}

# Use of format_map() function
print('{x} {y}'.format_map(a))
```

Output:

```
geeksforgeeks b
```

Example 3:

```
# Input dictionary
profession ={ 'name':['Barry', 'Bruce'],
               'profession':['Engineer', 'Doctor'],
               'age':[30, 31] }
```

```
# Use of format_map() function
print('{name[0]} is an {profession[0]} and he'
      ' is {age[0]} years
old.'.format_map(profession))

print('{name[1]} is an {profession[1]} and he'
      ' is {age[1]} years
old.'.format_map(profession))
```

Output:

```
Barry is an Engineer and he is 30 years old.

Bruce is an Doctor and he is 31 years old.
```

Example 4: Practical Application

The format_map() function can be used in any practical application.

```
# Python code showing practical
# use of format_map() function
defchk_msg(n):

    # input stored in variable a.
    a ={'name':"George", 'mesg':n}

    # use of format_map() function
    print('{name} has {mesg} new
messages'.format_map(a))

chk_msg(10)
```

Output:

```
George has 10 new messages
```

Python String index() Method

Finding the string(substring) in a string is an application that has many uses in day-to-day life. Python offers this using a function index(), which returns the position of the first occurrence of a substring in a string.

Syntax:

ch.index(ch1, begp, endp)

Parameters:

- **ch1 :** The string to be searched for.
- **begp (default : 0) :** This function specifies the position from where search has to be started.
- **endp (default : string_len) :** This function specifies the position from where search has to end.

Return Value:

Returns the first position of the substring found.

Exception:

Raises ValueError if argument string is not found.

Example 1

```
# Python code to demonstrate the working of
# index()

# initializing target string
ch ="geeksforgeeks"

# initializing argument string
ch1 ="geeks"

# using index() to find position of "geeks"
# starting from 2nd index
# prints 8
pos =ch.index(ch1,2)

print("The first position of geeks after 2nd index :
",end="")
print(pos)
```

Output:

```
The first position of geeks after 2nd index : 8
```

Note: The index() method is similar to find(). The only difference is find() returns -1 if the searched string is not found and index() throws an exception in this case.

Example 2: Exception

ValueError: This error is raised in the case when the argument string is not found in the target string.

```
# Python code to demonstrate the exception of
# index()

# initializing target string
ch ="geeksforgeeks"

# initializing argument string
ch1 ="gfg"

# using index() to find position of "gfg"
# raises error
pos =ch.index(ch1)

print("The first position of gfg is :
",end="")
print(pos)
```

Output:

```
Traceback (most recent call last):

  File "/home/aa5904420c1c3aa072ede56ead7e26ab.py", line 12, in

    pos = ch.index(ch1)

ValueError: substring not found
```

Example 3

Practical Application: This function is used to extract the **suffix or prefix length after or before the target word**. The example below displays the total bit length of instruction coming from AC voltage given information in a string.

```
# Python code to demonstrate the application of
# index()

# initializing target strings
voltages =["001101 AC", "0011100 DC", "0011100 AC", "001
DC"]

# initializing argument string
type="AC"

# initializing bit-length calculator
sum_bits =0

fori involtages :
    ch =i

    if(ch[len(ch)-2]!='D'):
       # extracts the length of bits in string
       bit_len =ch.index(type)-1

       # adds to total
       sum_bits =sum_bits +bit_len

print("The total bit length of AC is : ",end="")
print(sum_bits)
```

Output:

```
The total bit length of AC is : 13
```

Python String isalnum() Method

Python String **isalnum()** method checks whether all the characters in a given string are alphanumeric or not. **Alphanumeric** means a character that is either a letter or a number.

Syntax:

string_name.isalnum()

Parameter:

isalnum() method takes no parameters

Return:

- **True:** If all the characters are alphanumeric
- **False:** If one or more characters are not alphanumeric

Example 1: Working of isalnum()

```
# Python program to demonstrate the use of
# isalnum() method

# here a,b and c are characters and 1,2 and 3
# are numbers
string ="abc123"
print(string.isalnum())

# here a,b and c are characters and 1,2 and
3
# are numbers but space is not a
alphanumeric
# character
string ="abc 123"
print(string.isalnum())
```

Output:

```
True

False
```

Python String isalpha() Method

Python String **isalpha()** method is a built-in method used for string handling. The isalpha() methods returns “True” if all characters in the string are alphabets, Otherwise, It returns “False”. This function is used to check if the argument includes only alphabet characters (mentioned below).

ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz

Syntax:

string.isalpha()

Parameters:

isalpha() does not take any parameters

Returns:

- **True**: If all characters in the string are alphabet.
- **False**: If the string contains 1 or more non-alphabets.

Errors and Exceptions:

1. It contains no arguments, therefore an error occurs if a parameter is passed
1. Both uppercase and lowercase alphabets return “True”
1. Space is not considered to be the alphabet, therefore it returns “False”

Examples

```
Input : string = 'Ayush'
Output : True

Input : string = 'Ayush Saxena'
Output : False

Input : string = 'Ayush0212'
Output : False
```

Example 1: Working of isalpha()

```
# Python code for implementation of
isalpha()

# checking for alphabets
string ='Ayush'
print(string.isalpha())

string ='Ayush0212'
```

```
print(string.isalpha())

# checking if space is an alphabet
string ='Ayush Saxena'
print( string.isalpha())
```

Output:

```
True

False

False
```

Example 2: Practical Application

Given a string in python, count number of alphabets in the string and print the alphabets.

```
Input : string = 'Ayush Saxena'

Output : 11

         AyushSaxena



Input : string = 'Ayush0212'

Output : 5

         Ayush
```

Algorithm:

1. Initialize a new string and variable counter to 0.
1. Traverse the given string character by character upto its length, check if character is an alphabet.
1. If it is an alphabet, increment the counter by 1 and add it to a the new string, else traverse to the next character.
1. Print the value of the counter and the new string.

```
# Python program to illustrate
# counting number of alphabets
# using isalpha()

# Given string
string='Ayush Saxena'
count=0

# Initialising new strings
newstring1 =""
newstring2 =""

# Iterating the string and checking for alphabets
# Incrementing the counter if an alphabet is
found
# Finally printing the count
fora instring:
    if(a.isalpha()) ==True:
        count+=1
        newstring1+=a
print(count)
print(newstring1)

# Given string
string='Ayush0212'
count=0
fora instring:
    if(a.isalpha()) ==True:
        count+=1
        newstring2+=a
print(count)
print(newstring2)
```

Output:

```
11

AyushSaxena

5

Ayush
```

Python string | isdecimal()

isdecimal()-It is a function in Python that returns true if all characters in a string are decimal. If all characters are not decimal then it returns false.

Syntax:

string_name.isdecimal()
Here string_name is the string whose characters are to be checked

Parameters: This method does not takes any parameters .

Returns Value:

True – all characters are decimal
False – one or more then one character is not decimal.

Below is the Python3 implementation of isdecimal() method:

```
# Python3 program to demonstrate the
use
# of isdecimal()

s ="12345"
print(s.isdecimal())

# contains alphabets
s ="12geeks34"
print(s.isdecimal())

# contains numbers and spaces
s ="12 34"
print(s.isdecimal())
```

Output:

```
True
False
False
```

Python String isdigit() Method

Python String **isdigit()** method is a built-in method used for string handling. The isdigit() method returns “**True**” if all characters in the string are digits, Otherwise, It returns “False”. This function is used to check if the argument contains digits such as **0123456789**

Syntax:

string.isdigit()

Parameters:

isdigit() does not take any parameters

Returns:

- True – If all characters in the string are digits.
- False – If the string contains 1 or more non-digits.

Errors And Exceptions:

1. It does not take any arguments, therefore it returns an error if a parameter is passed
1. Superscript and subscripts are considered digit characters along with decimal characters, therefore, isdigit() returns “True”.
1. The Roman numerals, currency numerators, and fractions are not considered to be digits. Therefore, the isdigit() returns “False”

Example 1:

```
Input : string = '15460'
Output : True

Input : string = '154ayush60'
Output : False
```

```
# Python code for implementation of
isdigit()

# checking for digit
string ='15460'
print(string.isdigit())

string ='154ayush60'
print(string.isdigit())
```

Output:

```
True

False
```

Example 2:

Application: Using **ASCII values** of characters, count and print all the digits using isdigit() function.

Algorithm:

1. Initialize a new string and a variable count=0.
1. Traverse every character using ASCII value, check if the character is a digit.
1. If it is a digit, increment the count by 1 and add it to the new string, else traverse to the next character.
1. Print the value of the counter and the new string.

```
# Python program to illustrate
# application of isdigit()
# initialising Empty string
newstring =''

# Initialising the counters to 0
count =0

# Incrementing the counter if a digit is
found
# and adding the digit to a new string
# Finally printing the count and the new
string

fora inrange(53):
    b =chr(a)
    ifb.isdigit() ==True:
        count+=1
        newstring+=b

print("Total digits in range :", count)
print("Digits :", newstring)
```

Output:

```
Total digits in range : 5

Digits : 01234
```

In Python, superscript and subscripts (usually written using Unicode) are also considered digit characters. Hence, if the string contains these characters along with decimal characters, isdigit() returns True. The Roman numerals, currency numerators, and fractions (usually written using Unicode) are considered numeric characters but not digits. The isdigit() returns False if the string contains these characters. To check whether a character is a numeric character or not, you can use isnumeric() method.

Example 3:

String Containing digits and Numeric Characters

```
s ='23455'
print(s.isdigit())

# s = '²3455'
# subscript is a digit
s ='\u00B23455'

print(s.isdigit())

# s = '½'
# fraction is not a
digit
s ='\u00BD'

print(s.isdigit())
```

Output:

```
True

True

False
```

Python String isidentifier() Method

Python String **isidentifier()** method is used to check whether a string is a valid identifier or not. The method returns True if the string is a valid identifier else returns False.

Syntax:

string.isidentifier()

Parameters:

The method does not take any parameters

Return Value:

The method can return one of the two values:

- **True:** When the string is a valid identifier.
- **False:** When the string is not a valid identifier.

Example: How isidentifier() works

```
# Python code to illustrate
# the working of
isidentifier()

# String with spaces
string ="Geeks for Geeks"
print(string.isidentifier())

# A Perfect identifier
string ="GeeksforGeeks"
print(string.isidentifier())

# Empty string
string =""
print(string.isidentifier())

# Alphanumerical string
string ="Geeks0for0Geeks"
print(string.isidentifier())

# Beginning with an integer
string ="54Geeks0for0Geeks"
print(string.isidentifier())
```

Output:

```
False

True

False

True
```

```
False
```

Python String islower() method

Python String islower() method checks if all characters in the string are lowercase. This method returns **True** if all alphabets in a string are lowercase alphabets. If the string contains at least one uppercase alphabet, it returns False.

Syntax:

string.islower()

Parameters:

None

Returns:

- True: If all the letters in the string are in lower case and
- False: If even one of them is in upper case.

Example 1: Demonstrating the working of islower()

```
# Python3 code to demonstrate
# working of islower()

# initializing string
islow_str ="geeksforgeeks"
not_islow ="Geeksforgeeks"

# checking which string is
# completely lower
print("Is geeksforgeeks full lower ? : "+str(islow_str.islower()))
print("Is Geeksforgeeks full lower ? : "+str(not_islow.islower()))
```

Output:

```
Is geeksforgeeks full lower ? : True

Is Geeksforgeeks full lower ? : False
```

Example 2: Practical Application

This function can be used in many ways and has many practical applications. One such application is checking for lower cases, checking proper nouns, checking for correctness of sentences that requires all lower cases. Demonstrated below is a small example showing the application of islower() method.

```
# Python3 code to demonstrate
# application of islower() method

# checking for proper nouns.
# nouns which start with capital letter

test_str ="Geeksforgeeks ismost rated Computer \
            Science portal andishighly recommended"

# splitting string
list_str =test_str.split()

count =0

# counting lower cases
fori inlist_str:
    if(i.islower()):
        count =count +1

# printing proper nouns count
print("Number of proper nouns in this sentence is : "
                            +str(len(list_str)-
count))
```

Output:

```
Number of proper nouns in this sentence is : 3
```

Python String isnumeric() Method

Python String **isnumeric()** method is a built-in method used for string handling. The isnumeric() method returns “True” if all characters in the string are numeric characters, Otherwise, It returns “False”. This function is used to check if the argument contains all numeric characters such as **integers, fractions, subscript, superscript, roman numerals**, **etc.(all written in Unicode)**

Syntax:

string.isnumeric()

Parameters:

isnumeric() does not take any parameters

Returns :

- True – If all characters in the string are numeric characters.
- False – If the string contains 1 or more non-numeric characters.

Errors and Exceptions:

1. It does not contain any arguments, therefore, it returns an error if a parameter is passed.
1. Whitespaces are not considered to be numeric, therefore, it returns “False”
1. Subscript, Superscript, Fractions, Roman numerals (all written in Unicode)are all considered to be numeric, Therefore, it returns “True”

Example 1:

```
Input : string = '1889345'
Output : True

Input : string = '\u00BD'
Output : True

Input : string = '123ayu456'
Output : False
```

```
# Python code for implementation of
isnumeric()

# checking for numeric characters
string ='123ayu456'
print(string.isnumeric())

string ='123456'
print( string.isnumeric())
```

Output:

```
False
```

```
True
```

Example 2:

Application: Given a string in python, count the number of numeric characters in the string and remove them from the string, and print the string.

```
Input : string = '123geeks456for789geeks'

Output : 9

         geeksforgeeks



Input : string = '123ayu456'

Output : 6

         ayu
```

Algorithm:

1. Initialize an empty NewString and variable count to 0.
1. Traverse the given string character by character up to its length, check if the character is a numeric character.
1. If it is a numeric character, increment the counter by 1 and do not add it to the new string, else traverse to the next character and keep adding the characters to the new string if not numeric.
1. Print the value of the counter and the NewString.

```
# Python implementation to count numeric characters
# in a string and print non numeric characters
# Given string
# Initialising the counter to 0
string ='123geeks456for789geeks'
count =0

newstring1 =""
newstring2 =""

# Iterating the string and checking for numeric
characters
# Incrementing the counter if a numeric character is
found
# And adding the character to new string if not numeric
# Finally printing the count and the newstring
fora instring:
    if(a.isnumeric()) ==True:
        count+=1
    else:
        newstring1+=a
print(count)
print(newstring1)

string ='123ayu456'
count =0
fora instring:
    if(a.isnumeric()) ==True:
        count+=1
    else:
        newstring2+=a
print(count)
print(newstring2)
```

Output:

```
9

geeksforgeeks

6

ayu
```

Python String isprintable() Method

Python String isprintable() is a built-in method used for string handling. The isprintable() method returns “True” if all characters in the string are printable or the string is empty, Otherwise, It returns “False”. This function is used to check if the argument contains any printable characters such as:

- Digits (0123456789)
- Uppercase letters (ABCDEFGHIJKLMNOPQRSTUVWXYZ)
- Lowercase letters (abcdefghijklmnopqrstuvwxyz)
- Punctuation characters (!”#$%&'()*+, -./:;?@[\]^_`{ | }~)
- Space ( )

Syntax:

string.isprintable()

Parameters:

isprintable() does not take any parameters

Returns:

- True – If all characters in the string are printable or the string is empty.
- False – If the string contains 1 or more nonprintable characters.

Errors Or Exceptions:

1. The function does not take any arguments, therefore no parameters should be passed, otherwise, it returns an error.
1. The only whitespace character which is printable is *space* or ” “, otherwise every whitespace character is non-printable and the function returns “False”.
1. The empty string is considered printable and it returns “True”.

Example 1

```
Input : string = 'My name is Ayush'
Output : True

Input : string = 'My name is \n Ayush'
Output : False

Input : string = ''
Output : True
```

```
# Python code for implementation of
isprintable()

# checking for printable characters
```

```
string ='My name is Ayush'
print(string.isprintable())

# checking if \n is a printable character
string ='My name is \n Ayush'
print(string.isprintable())

# checking if space is a printable character
string =''
print( string.isprintable())
```

Output:

```
True

False

True
```

Example 2: Practical Application

Given a string in python, count the number of non-printable characters in the string and replace non-printable characters with a space.

```
Input : string = 'My name is Ayush'

Output : 0

         My name is Ayush


Input : string = 'My\nname\nis\nAyush'

Output : 3

         My name is Ayush
```

Algorithm:

1. Initialize an empty new string and a variable count = 0.
1. Traverse the given string character by character up to its length, check if the character is a non-printable character.
1. If it is a non-printable character, increment the counter by 1, and add a space to the new string.
1. Else if it is a printable character, add it to the new string as it is.
1. Print the value of the counter and the new string.

```
# Python implementation to count
# non-printable characters in a string

# Given string and new string
string ='GeeksforGeeks\nname\nis\nCS
portal'
newstring =''

# Initialising the counter to 0
```

```
count =0

# Iterating the string and
# checking for non-printable characters
# Incrementing the counter if a
# non-printable character is found
# and replacing it by space in the
newstring

# Finally printing the count and newstring

fora instring:
    if(a.isprintable()) ==False:
            count+=1
            newstring+=' '
    else:
            newstring+=a
print(count)
print(newstring)
```

Output:

```
3

GeeksforGeeks name is CS portal
```

Python String isspace() Method

Python String **isspace()** is a built-in method used for string handling. The isspace() method returns “True” if all characters in the string are whitespace characters, Otherwise, It returns “False”. This function is used to check if the argument contains all whitespace characters such as:

- ‘ ‘ – Space
- ‘\t’ – Horizontal tab
- ‘\n’ – Newline
- ‘\v’ – Vertical tab
- ‘\f’ – Feed
- ‘\r’ – Carriage return

Syntax:

string.isspace()

Parameters:

isspace() does not take any parameters

Returns:

1. **True** – If all characters in the string are whitespace characters.
1. **False** – If the string contains 1 or more non-whitespace characters.

Example 1

```
Input : string = 'Geeksforgeeks'
Output : False

Input : string = '\n \n \n'
Output : True

Input : string = 'Geeks\nFor\nGeeks'
Output : False
```

```
# Python code for implementation of
isspace()

# checking for whitespace characters
string ='Geeksforgeeks'

print(string.isspace())

# checking if \n is a whitespace character
string ='\n \n \n'

print(string.isspace())

string ='Geeks\nfor\ngeeks'
print( string.isspace())
```

Output:

```
False

True

False
```

Example 2: Practical Application

Given a string in python, count the number of whitespace characters in the string.

```
Input : string = 'My name is Ayush'

Output : 3


Input : string = 'My name is \n\n\n\n\nAyush'

Output : 8
```

Algorithm:

1. Traverse the given string character by character up to its length, check if the character is a whitespace character.
1. If it is a whitespace character, increment the counter by 1, else traverse to the next character.
1. Print the value of the counter.

```
# Python implementation to count whitespace characters in a
string
# Given string
# Initialising the counter to 0
string ='My name is Ayush'
count=0

# Iterating the string and checking for whitespace characters
# Incrementing the counter if a whitespace character is found
# Finally printing the count
fora instring:
    if(a.isspace()) ==True:
        count+=1
print(count)

string ='My name is \n\n\n\n\nAyush'
count =0
fora instring:
    if(a.isspace()) ==True:
        count+=1
print(count)
```

Output:

```
3

8
```

Python String istitle() Method

Python String istitle() method returns True if the string is a titlecased string otherwise it returns False. **What is titlecased?** The string which has the first character in each word Uppercase and remaining all characters Lowercase alphabets.

Syntax:

string.istitle()

Parameters:

The istitle() method doesn’t take any parameters.

Returns:

True if the string is a titlecased string otherwise it returns False.

Example 1

```
# First character in each word is
# uppercase and remaining
lowercases
s ='Geeks For Geeks'
print(s.istitle())

# First character in first
# word is lowercase
s ='geeks For Geeks'
print(s.istitle())

# Third word has uppercase
# characters at middle
s ='Geeks For GEEKs'
print(s.istitle())

s ='6041 Is My Number'
print(s.istitle())

# word has uppercase
# characters at middle
s ='GEEKS'
print(s.istitle())
```

Output:

```
True

False

False

True
```

```
False
```

Example 2

```
s ='I Love Geeks For Geeks'

ifs.istitle() ==True:
    print('Titlecased String')
else:
    print('Not a Titlecased
String')

s ='I Love geeks for geeks'

ifs.istitle() ==True:
    print('Titlecased String')
else:
    print('Not a Titlecased
String')
```

Output:

```
Titlecased String

Not a Titlecased String
```

Python String isupper() method

Python String isupper() method returns whether or not all characters in a string are uppercased or not.

Syntax:

string.isupper()

Parameters:

The isupper() method doesn’t take any parameters.

Returns:

True if all the letters in the string are in the upper case and False if even one of them is in the lower case.

Example 1: Demonstrating the working of isupper()

```
# Python3 code to demonstrate
# working of isupper()

# initializing string
isupp_str ="GEEKSFORGEEKS"
not_isupp ="Geeksforgeeks"

# Checking which string is
# completely uppercase
print("Is GEEKSFORGEEKS full uppercase ? : "+str(isupp_str.isupper()))
print("Is Geeksforgeeks full uppercase ? : "+str(not_isupp.isupper()))
```

Output:

```
Is GEEKSFORGEEKS full uppercase ? : True

Is Geeksforgeeks full uppercase ? : False
```

Example 2: Practical Application

This function can be used in many ways and has many practical applications. One such application for checking the upper cases, checking Abbreviations (usually upper case), checking for correctness of sentence which requires all upper cases. Demonstrated below is small example showing the application of isupper() method.

```
# Python3 code to demonstrate
# application of isupper()

# checking for abbreviations.
# short form of work/phrase
test_str ="Cyware is US based MNC and works in IOT technology"
```

```
# splitting string
list_str =test_str.split()

count =0

# counting upper cases
fori inlist_str:
    if(i.isupper()):
        count =count +1

# printing abbreviations count
print("Number of abbreviations in this sentence is : "+str(count))
```

Output:

```
Number of abbreviations in this sentence is : 3
```

Python String join() Method

Python String join() method is a string method and returns a string in which the elements of the sequence have been joined by the str separator.

Syntax:

string_name.join(iterable)

Parameters:

The join() method takes iterable – objects capable of returning their members one at a time. Some examples are **List, Tuple, String, Dictionary**, **and Set**

Return Value:

The join() method returns a string concatenated with the elements of *iterable*.

Type Error:

If the iterable contains any non-string values, it raises a TypeError exception.

Example 1: Working of join() method

```
# Python program to demonstrate
the
# use of join function to join
list
# elements with a character.

list1 =['1','2','3','4']

s ="-"

# joins elements of list1 by '-'
# and stores in string s
s =s.join(list1)

# join use to join a list of
# strings to a separator s
print(s)
```

Output:

```
1-2-3-4
```

Example 2: Joining with an empty string

```
# Python program to demonstrate
the
# use of join function to join
list
# elements without any separator.

# Joining with empty separator
list1 =['g','e','e','k', 's']
print("".join(list1))
```

Output:

```
geeks
```

Python String lower() Method

Python String **lower()** method converts all uppercase characters in a string into lowercase characters and returns it.

Syntax:

string.lower()

Parameters:

The lower() method doesn't take any parameters.

Returns:

Returns a lowercase string of the given string

Example 1: String with only alphabetic characters

```
# Python3 program to show
the
# working of lower()
function
text ='GeEks FOR geeKS'

print("Original String:")
print(text)

# lower() function to
convert
# string to lower_case
print("\nConverted String:")
print(text.lower())
```

Output:

```
Original String:

GeEks FOR geeKS



Converted string:

geeks for geeks
```

Example 2: String with Alphanumeric Characters

```
# Python3 program to show
the
# working of lower()
function
text ='G3Ek5 F0R gE3K5'
```

```
print("Original String:")
print(text)

# lower() function to
convert
# string to lower_case
print("\nConverted String:")
print(text.lower())
```

Output:

```
Original String:

G3Ek5 F0R gE3K5



Converted String:

g3ek5 f0r ge3k5
```

Example 3

One of the common applications of the lower() method is to check if the two strings are the same or not.

```
# Python3 program to show the
# working of lower() function
text1 ='GEEKS For GEEKS'

text2 ='gEeKS fOR GeeKs'

# Comparison of strings using
# lower() method
if(text1.lower()
==text2.lower()):
    print("Strings are same")
else:
    print("Strings are not same")
```

Output:

```
Strings are same
```

Python String lstrip() method

Python String **lstrip()** method returns a copy of the string with leading characters removed (based on the string argument passed). If no argument is passed, it removes leading spaces.

Syntax:

string.lstrip(characters)

Parameters:

- **characters** [optional]: A set of characters to remove as leading characters.

Returns:

Returns a copy of the string with leading characters stripped.

Example 1

```
# Python3 program to demonstrate the use
of
# lstrip() method using default parameter

# string which is to be stripped
string ="   geeksforgeeks"

# Removes spaces from left.
print(string.lstrip())
```

Output:

```
geeksforgeeks
```

Example 2

```
# Python3 program to demonstrate the use
of
# lstrip() method using optional parameters

# string which is to be stripped
string ="++++x...y!!z* geeksforgeeks"

# Removes given set of characters from
left.
print(string.lstrip("+.!*xyz"))
```

Output:

```
geeksforgeeks
```

Example 3

```
# string which is to be stripped
string ="geeks for geeks"

# Argument doesn't contain leading
'g'
# So, no characters are removed
print(string.lstrip('ge'))
```

Output:

```
ks for geeks
```

Example 4

There is a runtime error when we try to strip anything except a string.

```
# Python3 program to demonstrate the use
of
# strip() method error

string =" geeks for geeks "
list=[1, 2, 3]

# prints the error message
print(list.lstrip())
```

Output:

```
print(list.lstrip())

AttributeError: 'list' object has no attribute 'lstrip'
```

Python String partition() Method

Python String partition() method splits the string at the first occurrence of the separator and returns a tuple containing the part before the separator, separator, and the part after the separator. Here separator is a string that is given as the argument.

Syntax:

string.partition(separator)

Parameters:

The partition() method takes a separator(a string) as the argument that separates the string at its first occurrence.

Returns:

Returns a tuple that contains the part before the separator, separator parameter, and the part after the separator if the separator parameter is found in the string. Returns a tuple that contains the string itself and two empty strings if the separator parameter is not found.

Example 1

```
string ="pawan is a good"

# 'is' separator is found
print(string.partition('is '))

# 'not' separator is not found
print(string.partition('bad '))

string ="pawan is a good, isn't it"

# splits at first occurrence of
'is'
print(string.partition('is'))
```

Output:

```
('pawan ', 'is ', 'a good')

('pawan is a good', '', '')

('pawan ', 'is', " a good, isn't it")
```

Example 2

```
string ="geeks is a good"

# 'is' separator is found
print(string.partition('is '))
```

```
# 'not' separator is not found
print(string.partition('bad '))

string ="geeks is a good, isn't it"

# splits at first occurrence of
'is'
print(string.partition('is'))
```

Output:

```
('geeks ', 'is ', 'a good')

('geeks is a good', '', '')

('geeks ', 'is', " a good, isn't it")
```

Python String | replace()

replace() is an inbuilt function in the Python programming language that returns a copy of the string where all occurrences of a substring are replaced with another substring.

Syntax :

```
string.replace(old, new, count)
```

Parameters :

old – old substring you want to replace.
new – new substring which would replace the old substring.
count – the number of times you want to replace the old substring with the new substring. (**Optional**)

Return Value :
It returns a copy of the string where all occurrences of a substring are replaced with another substring.

Note:

- If count is not specified then all the occurrences of the old substring are replaced with the new substring.
- This method returns the copy of the string i.e. it does not change the original string.

Below is the code demonstrating ***replace()*** :

Example 1:

```
# Python3 program to demonstrate the
# use of replace() method

string ="geeks for geeks geeks geeks geeks"

# Prints the string by replacing all
# geeks by Geeks
print(string.replace("geeks", "Geeks"))

# Prints the string by replacing only
# 3 occurrence of Geeks
print(string.replace("geeks", "GeeksforGeeks",
3))
```

Output :

```
Geeks for Geeks Geeks Geeks Geeks

GeeksforGeeks for GeeksforGeeks GeeksforGeeks geeks geeks
```

Example 2:

```
# Python3 program to demonstrate the
# use of replace() method

string ="geeks for geeks geeks geeks
geeks"

# Prints the string by replacing
# e by a
print(string.replace("e", "a"))

# Prints the string by replacing only
# 3 occurrence of ek by a
print(string.replace("ek", "a", 3))
```

Output:

```
gaaks for gaaks gaaks gaaks gaaks

geas for geas geas geeks geeks
```

Python String rfind() Method

Python String ***rfind()*** method returns the highest index of the substring if found in the given string. If not found then it returns -1.

Syntax:

str.rfind(sub, start, end)

Parameters:

- **sub:** It’s the substring that needs to be searched in the given string.
- **start:** Starting position where the sub needs to be checked within the string.
- **end:** Ending position where suffix needs to be checked within the string.

Note: If start and end indexes are not provided then, by default it takes 0 and length-1 as starting and ending indexes where ending indexes are not included in our search.

Return:

Returns the highest index of the substring if it is found in the given string; if not found, then it returns -1.

Exception:

ValueError: This error is raised in the case when the argument string is not found in the target string.

Example 1

```
# Python program to demonstrate working of rfind()
# in whole string
word ='geeks for geeks'

# Returns highest index of the substring
result =word.rfind('geeks')
print("Substring 'geeks' found at index :",
result )

result =word.rfind('for')
print("Substring 'for' found at index :", result )

word ='CatBatSatMatGate'

# Returns highest index of the substring
result =word.rfind('ate')
print("Substring 'ate' found at index :", result)
```

Output:

```
Substring 'geeks' found at index : 10
Substring 'for' found at index : 6

Substring 'ate' found at index : 13
```

Example 2

```
# Python program to demonstrate working of
rfind()
# in a sub-string
word ='geeks for geeks'

# Substring is searched in 'eeks for geeks'
print(word.rfind('ge', 2))

# Substring is searched in 'eeks for geeks'
print(word.rfind('geeks', 2))

# Substring is searched in 'eeks for geeks'
print(word.rfind('geeks ', 2))

# Substring is searched in 's for g'
print(word.rfind('for ', 4, 11))
```

Output:

```
10

10

-1

6
```

Example 3: Practical Application

Useful in string checking. To check if the given substring is present in some string or not.

```
# Python program to demonstrate working of
rfind()
# to search a string
word ='CatBatSatMatGate'

if(word.rfind('Ate') !=-1):
    print("Contains given substring ")
else:
    print("Doesn't contains given substring")
```

Output:

```
Doesn't contains given substring
```

Python String | rpartition()

rpartition() function in Python split the given string into three parts. rpartition() start looking for separator from right side, till the separator is found and return a tuple which contains part of the string before separator, argument of the string and the part after the separator.

Syntax :

```
string.rpartition(separator)
```

Parameters :

separator - `separates the string at the first occurrence of it.`

Return Value :

1. It returns the part the string before the separator, separator parameter itself, and the part after the separator if the separator parameter is found in the string.
1. It returns two empty strings, followed by the given string if the separator is not found in the string.

Exception :

```
If separator argument is not supplied, it will throw TypeError.
```

Code #1 :

```
# Python3 code explaining rpartition()

# String need to split
string1 ="Geeks@for@Geeks@is@for@geeks"

string2 ="Ram is not eating but Mohan is
eating"

# Here '@' is a separator
print(string1.rpartition('@'))

# Here 'is' is separator
print(string2.rpartition('is'))
```

Output :

```
('Geeks@for@Geeks@is@for', '@', 'geeks')

('Ram is not eating but Mohan ', 'is', ' eating')
```

Code #2 :

```
# Python3 code explaining rpartition()

# String need to split
```

```
string ="Sita is going to school"

# Here 'not' is a separator which is
not
# present in the given string
print(string.rpartition('not'))
```

Output :

```
('', '', 'Sita is going to school')
```

Code #3 : TypeError

```
# Python3 code explaining
TypeError
# in rpartition()

str="Bruce Waine is Batman"

# Nothing is passed as separator
print(str.rpartition())
```

Output :

```
Traceback (most recent call last):

  File "/home/e207c003f42055cf9697001645999d69.py", line 7, in

    print(str.rpartition())

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

Code #4 : ValueError

```
# Python3 code explaining
ValueError
# in rpartition()

str="Bruce Waine is Batman"

# Nothing is passed as separator
print(str.rpartition(""))
```

Output :

```
Traceback (most recent call last):

  File "/home/c8d9719625793f2c8948542159719007.py", line 7, in

    print(str.rpartition(""))

ValueError: empty separator
```

Python String rsplit() Method

Python String ***rsplit()*** method returns a list of strings after breaking the given string from the right side by the specified separator.

Syntax:

str.rsplit(separator, maxsplit)

Parameters:

- **separator:** The is a delimiter. The string splits at this specified separator starting from the right side. It is not provided then any white space is a separator.
- **maxsplit:** It is a number, which tells us to split the string into a maximum of provided number of times. If it is not provided then there is no limit.

Return:

Returns a *list* of strings after breaking the given string from the right side by the specified separator.

Error:

We will not get any error even if we are not passing any argument.

Example 1

```
# Python code to split a string
# using rsplit.

# Splits at space
word ='geeks for geeks'
print(word.rsplit())

# Splits at 'g'. Note that we have
# provided maximum limit as 1. So
# from right, one splitting happens
# and we get "eeks" and "geeks, for, "
word ='geeks, for, geeks'
print(word.rsplit('g', 1))

# Splitting at '@' with maximum
splitting
# as 1
word ='geeks@for@geeks'
print(word.rsplit('@', 1))
```

Output:

```
['geeks', 'for', 'geeks']

['geeks, for, ', 'eeks']

['geeks@for', 'geeks']
```

Example 2

```
word ='geeks, for, geeks, pawan'

# maxsplit: 0
print(word.rsplit(', ', 0))

# maxsplit: 4
print(word.rsplit(', ', 4))

word
='geeks@for@geeks@for@geeks'

# maxsplit: 1
print(word.rsplit('@', 1))

# maxsplit: 2
print(word.rsplit('@', 2))
```

Output:

```
['geeks, for, geeks, pawan']

['geeks', 'for', 'geeks', 'pawan']

['geeks@for@geeks@for', 'geeks']

['geeks@for@geeks', 'for', 'geeks']
```

Example 3

```
word ='geeks for geeks'

# Since separator is 'None',
# so, will be splitted at space
print(word.rsplit(None, 1))

print(word.rsplit(None, 2))

# Also observe these
print('@@@@@geeks@for@geeks'.rsplit('@'))
print('@@@@@geeks@for@geeks'.rsplit('@',
1))
print('@@@@@geeks@for@geeks'.rsplit('@',
3))
print('@@@@@geeks@for@geeks'.rsplit('@',
5))
```

Output:

```
['geeks for', 'geeks']

['geeks', 'for', 'geeks']

['', '', '', '', '', 'geeks', 'for', 'geeks']

['@@@@@geeks@for', 'geeks']

['@@@@', 'geeks', 'for', 'geeks']
```

```
['@@', '', '', 'geeks', 'for', 'geeks']
```

Python String rstrip() Method

Python String **rstrip()** method returns a copy of the string with trailing characters removed (based on the string argument passed). If no argument is passed, it removes trailing spaces.

Syntax:

string.rstrip([chars])

Parameters:

chars (optional) – a string specifying the set of characters to be removed.

Returns:

Returns a copy of the string with trailing characters stripped

Example 1

```
# Python3 program to demonstrate the use
of
# rstrip() method using optional
parameters

# string which is to be stripped
string ="geekssss"

# Removes given set of characters from
# right.
print(string.rstrip('s'))
```

Output:

```
geek
```

Example 2

```
# Python3 program to demonstrate the use
of
# rstrip() method using optional
parameters

# string which is to be stripped
string ="   for    "

# Leading whitespaces are removed
print("Geeks"+string.rstrip() +" Geeks ")
```

Output:

```
Geeks   for Geeks
```

Example 3

```
# string which is to be stripped
string ="geeks for geeks"

# Argument doesn't contain trailing
's'
# So, no characters are removed
print(string.rstrip('ek'))
```

Output:

```
geeks for geeks
```

Example 4

```
# string which is to be stripped
string ="geeks for geeks"

# Removes given set of characters
from
# right.
print(string.rstrip('ske'))
```

Output:

```
geeks for g
```

Python String splitlines() Method

Python String ***splitlines()*** method is used to split the lines at line boundaries. The function returns a list of lines in the string, including the line break(optional).

Syntax:

string.splitlines([keepends])

Parameters:

keepends (optional)**:** When set to **True** line breaks are included in the resulting list. This **can be a number**, specifying the position of line break or, can be any **Unicode characters**, like "\n", "\r", "\r\n", etc as boundaries for strings.

Return Value:

Returns a *list* of the lines in the string, breaking at line boundaries.

splitlines() splits on the following line boundaries:

Representation	**Description**
\n	Line Feed
\r	Carriage Return
\r\n	Carriage Return + Line Feed
\x1c	File Separator
\x1d	Group Separator
\x1e	Record Separator
\x85	Next Line (C1 Control Code)
\v or \x0b	Line Tabulation
\f or \x0c	Form Feed
\u2028	Line Separator
\u2029	Paragraph Separator

Example 1

```
# Python code to illustrate splitlines()
string ="Welcome everyone to\rthe world of Geeks\
nGeeksforGeeks"

# No parameters has been passed
print(string.splitlines( ))

# A specified number is passed
```

```
print(string.splitlines(0))

# True has been passed
print(string.splitlines(True))
```

Output:

```
['Welcome everyone to', 'the world of Geeks', 'GeeksforGeeks']

['Welcome everyone to', 'the world of Geeks', 'GeeksforGeeks']

['Welcome everyone to\r', 'the world of Geeks\n', 'GeeksforGeeks']
```

Example 2

```
# Python code to illustrate splitlines()
string ="Cat\nBat\nSat\nMat\nXat\nEat"

# No parameters has been passed
print(string.splitlines( ))

# splitlines() in one line
print('India\nJapan\nUSA\nUK\nCanada\
n'.splitlines())
```

Output:

```
['Cat', 'Bat', 'Sat', 'Mat', 'Xat', 'Eat']

['India', 'Japan', 'USA', 'UK', 'Canada']
```

Example 3: Practical Application

In this code, we will understand how to use the concept of splitlines() to calculate the length of each word in a string.

```
# Python code to get length of each words
defCal_len(string):

    # Using splitlines() divide into a
list
    li =string.splitlines()
    print(li)

    # Calculate length of each word
    l =[len(element) forelement inli]
    returnl

# Driver Code
string ="Welcome\rto\rGeeksforGeeks"
print(Cal_len(string))
```

Output:

```
['Welcome', 'to', 'GeeksforGeeks']
```

```
[7, 2, 13]
```

Python | String startswith()

The startswith() method returns True if a string starts with the given prefix otherwise returns False.

Syntax :

`str.startswith(prefix, start, end)`

Parameters :

```
prefix : prefix ix nothing but a string which needs to be checked. start :
Starting position where prefix is needs to be checked within the string.
end : Ending position where prefix is needs to be checked within the string.
```

NOTE : If start and end index is not provided then by default it takes 0 and length-1 as starting and ending indexes where ending indes is not included in our search.

Returns :

```
It returns True if strings starts with the given
prefix otherwise returns False.
```

Examples:

```
Input : text = "geeks for geeks."
        result = text.startswith('for geeks')
Output : False

Input : text = "geeks for geeks."
        result = text.startswith('geeks', 0)
Output : True
```

Error : ValueError : This error is raised in the case when the argument string is not found in the target string

```
# Python code shows the working of
# .startsswith() function

text ="geeks for geeks."

# returns False
result =text.startswith('for geeks')
print(result)

# returns True
result =text.startswith('geeks')
print(result)

# returns False
```

```
result =text.startswith('for geeks.')
print(result)

# returns True
result =text.startswith('geeks for
geeks.')
print(result)
```

Output:

```
False
True
False
True
```

Python string | strip()

strip() is an inbuilt function in Python programming language that returns a copy of the string with both leading and trailing characters removed (based on the string argument passed).
Syntax:

`string.strip([chars])` **Parameter:** There is only one optional parameter in it: 1)**chars** - a string specifying the set of characters to be removed. If the optional chars parameter is not given, all leading and trailing whitespaces are removed from the string. **Return Value:** Returns a copy of the string with both leading and trailing characters removed.

```
# Python3 program to demonstrate the use of
# strip() method

string ="""    geeks for geeks    """

# prints the string without stripping
print(string)

# prints the string by removing leading and trailing
whitespaces
print(string.strip())

# prints the string by removing geeks
print(string.strip(' geeks'))
```

Output:
```
    geeks for geeks
geeks for geeks
for
```

```
# Python Program to demonstrate use of strip() method

str1 ='geeks for geeks'
# Print the string without stripping.
print(str1)

# String whose set of characters are to be
# remove from original string at both its ends.
str2 ='ekgs'

# Print string after stripping str2 from str1 at both its
end.
print(str1.strip(str2))
```

Output:
```
geeks for geeks
 for
```

Working of above code :
We first construct a string str1 = ‘geeks for geeks’
Now we call strip method over str1 and pass str2 = ‘ekgs’ as argument.
Now python interpreter trace str1 from left.It remove the character of str1 if it is present in str2.

Otherwise it stops tracing.
Now python interpreter trace str1 from right. It remove the character of str1 if it is present in str2.
Otherwise it stop tracing.
Now at last it returns the resultant string.
When we call strip() without argument, it removes leading and trailing spaces.

```
# Python Program to demonstrate use of strip() method without any
argument
str1 ="""    geeks for geeks    """

# Print the string without stripping.
print(str1)

# Print string after removing all leading
# and trailing whitespaces.
print(str1.strip())
```

Input: `    geeks for geeks`

Output:

```
geeks for geeks
```

Practical application:
Given a string remove occurrence of word “the” from the beginning and the end.

```
# Python3 program to demonstrate the practical application
# strip()


string =" the King has the largest army in the entire world
the"

# prints the string after removing the from beginning and end
print(string.strip(" the"))
```

Input: `the King has the largest army in the entire world the`

Output:

```
King has the largest army in the entire world
```

Python string | swapcase()

The string swapcase() method converts all uppercase characters to lowercase and vice versa of the given string, and returns it.

Syntax:

string_name.swapcase()
Here string_name is the string whose cases are to be swapped.

Parameter: The swapcase() method does not takes any parameter.

Return value:

The swapcase() method returns a string with all the cases changed.

Below is the Python implementation of the swapcase() method

```
# Python program to demonstrate the use
of
# swapcase() method

string ="gEEksFORgeeks"

# prints after swappong all cases
print(string.swapcase())

string ="striver"
print(string.swapcase())
```

Output:

```
GeeKSforGEEKS
STRIVER
```

Python String Title method

The title() function in python is the Python String Method which is used to convert the first character in each word to Uppercase and remaining characters to Lowercase in the string and returns a new string.

Syntax:

str.title()parameters:str is a valid string which we need to convert. **return:** This function returns a string which has first letter in each word is uppercase and all remaining letters are lowercase.

```
# Python title() Method Example

str1 ='geeKs foR geEks'
str2 =str1.title()
print'First Output after Title() method is = ', str2

# observe the original string
print'Converted String is = ', str1.title()
print'Original String is = ', str1

# Performing title() function directly
str3 ='ASIPU pawan kuMAr'.title()
print'Second Output after Title() method is = ',
str3

str4 ='stutya kUMari sHAW'.title()
print'Third Output after Title() method is = ', str4

str5 ='6041'.title()
print'Fourth Output after Title() method is = ',
str5
```

Output:

```
First Output after title() method is =  Geeks For Geeks

Converted String is =  Geeks For Geeks

Original String is =  geeKs foR geEks

Second Output after title() method is =  Asipu Pawan Kumar

Third Output after title() method is =  Stutya Kumari Shaw

Fourth Output after title() method is =  6041
```

Python | String translate()

translate() returns a string that is modified string of givens string according to given translation mappings.

There are two ways to translate :

Providing a mapping as a dictionary

Parameters :

string.translate(mapping)

`mapping` – a dictionary having mapping between two characters.
Returns : Returns modified string where each character is mapped to its corresponding character according to the provided mapping table.

```
# Python3 code to demonstrate
# translations without
# maketrans()

# specifying the mapping
# using ASCII
table ={ 119: 103, 121: 102, 117: None}

# target string
trg ="weeksyourweeks"

# Printing original string
print("The string before translating is : ", end
="")
print(trg)

# using translate() to make translations.
print("The string after translating is : ", end ="")
print(trg.translate(table))
```

Output:
```
The string before translating is : weeksyourweeks
The string after translating is : geeksforgeeks
```

One more example:

```
# Python 3 Program to show working
# of translate() method

# specifying the mapping
# using ASCII
translation ={103: None, 101: None, 101:
None}

string ="geeks"
print("Original string:", string)
```

```
# translate string
print("Translated string:",
       string.translate(translation))
```

Output:

```
Original string: geeks
Translated string: ks
```

Providing a mapping using [maketrans()]()

Syntax : maketrans(str1, str2, str3)
Parameters :
str1 : Specifies the list of characters that need to be replaced.
str2 : Specifies the list of characters with which the characters need to be replaced.
str3 : Specifies the list of characters that needs to be deleted.

Returns : Returns the translation table which specifies the conversions that can be used by translate()

```
# Python 3 Program to show working
# of translate() method

# First String
firstString ="gef"

# Second String
secondString ="eks"

# Third String
thirdString ="ge"

# Original String
string ="geeks"
print("Original string:", string)

translation =string.maketrans(firstString,
                               secondString,
                               thirdString)

# Translated String
print("Translated string:",
       string.translate(translation))
```

Output :

```
Original string: geeks
Translated string: ks
```

Python String upper()

upper() method converts all lowercase characters in a string into uppercase characters and returns it

Syntax :

```
string.upper()
```

Parameters :

The `upper()` method doesn’t take any parameters.

Returns :

```
returns a uppercased string of the given string
```

CODE 1: String with only alphabetic characters

```
# Python3 program to show the
# working of upper() function
text ='geeKs For geEkS'

print("Original String:")
print(text)

# upper() function to
convert
# string to upper_case
print("\nConverted String:")
print(text.upper())
```

Output :

```
Original String:
geeKs For geEkS

Converted String:
GEEKS FOR GEEKS
```

CODE 2: String with alphanumeric characters

```
# Python3 program to show the
# working of upper() function
text ='g3Ek5 f0r gE3K5'

print("Original String:")
print(text)

# upper() function to
convert
```

```
# string to upper_case
print("\nConverted String:")
print(text.upper())
```

Output :

```
Original String:
g3Ek5 f0r gE3K5

Converted String:
G3EK5 F0R GE3K5
```

Application: One of the common application of upper() method is to check if the two strings are same or not

```
# Python3 program to show the
# working of upper() function
text1 ='geeks fOr geeks'

text2 ='gEeKS fOR GeeKs'

# Comparison of strings using
# upper() method
if(text1.upper()
==text2.upper()):
    print("Strings are same")
else:
    print("Strings are not same")
```

Output:

```
Strings are same
```

Python String | zfill()

zfill() method returns a copy of the string with ‘0’ characters padded to the leftside of the given string.

Syntax :

```
str.zfill(length)
```

Parameters :

length : length is the length of the returned string from zfill() with ‘0’ digits filled to the leftside.

Return :

```
Returns a copy of the string with '0' characters
padded to the leftside of the given string.
```

CODE 1

```
text ="geeks for geeks"

print(text.zfill(25))

print(text.zfill(20))

# Given length is less than
# the length od original
string
print(text.zfill(10))
```

Output :

```
0000000000geeks for geeks
00000geeks for geeks
geeks for geeks
```

CODE 2

```
number ="6041"
print(number.zfill(8))

number ="+6041"
print(number.zfill(8))

text ="--anything%(&%(%)*^"
print(text.zfill(20))
```

Output :

```
00006041
+0006041
-0-anything%(&%(%)*^
```