Sr.	Escape Sequence	Description
6.	\b	ASCII Backspace(BS)
7.	\f	ASCII Formfeed
8.	\n	ASCII Linefeed
9.	\r	ASCII Carriege Return(CR)
10.	\t	ASCII Horizontal Tab
11.	\v	ASCII Vertical Tab
12.	\ooo	Character with octal value
13	\xHH	Character with hex value.

Here is the simple example of escape sequence.

In [ ]:
```
print("Python1 \
Python2 \
Python3")
```

In [ ]:
```
print("Python1\nPython2\nPython3")
```

In [ ]:
```
print("C:\\Users\\Milind Bhatt\\Python32\\Lib")
```

In [ ]:
```
print("This is the \n multiline quotes")
print("This is \x48\x45\x58 representation")
```

In [ ]:
```
print("Hello \r World")
```

In [ ]:
```
print("Hello\tWorld")
```

In [ ]:
```
print("Hello\v World\v We All Are\v From\v PSIT")
```

In [ ]:
```
print("\110\145\154\154\157")
```

In [ ]:
```
print("\x48\x65\x6c\x6c\x6f")
```

In [ ]:
```
print("call me Milind Sir\\Bhatt Sir")
```

In [ ]:
```
print('Mr Milind\'s class')
```

In [ ]:
```
print("Hello \aWorld")
```

Raw String

- We can ignore the escape sequence from the given string by using the raw string.
- We can do this by writing r or R in front of the string.
- Useful for specifying file paths on Windows or writing regular expressions.

Example

In [ ]:
```
print(r"C:\\Users\\MILIND BHATT\\Python37")
```

The format() method

- The format() method is the most flexible and useful method in formatting strings.
- The curly braces {} are used as the placeholder in the string and replaced by the format() method argument

Examples:

In [ ]:
```
# Using Curly braces
print("{} and {} both are the best friend".format("Ajay","Vijay"))
## equ'nt st in c will look like:
# printf("%s and %s both are the best friend","Ajay","Vijay")
```

In [ ]:
```

```

In [ ]:
```
a,b= "Ajay","Vijay"
print("{} and {} both are the best friend".format(a,b))
```

In [ ]:
```
#Positional Argument
print("Now a days {1} and {0} are the best players of Indian Cricket  ".format("Virat"
```

In [ ]:
```
#Keyword Argument
print("{a},{c},{b}".format(a = "James", b = "Peter", c = "Ricky"))
```

String Format :

- 1. Using f-string methods

- 2. Using % Operator

1. fromatted string using f-string methods

In [ ]:
```
a,b= "Ajay","Vijay"
print("{} and {} both are the best friend".format(a,b))
```

In [ ]:
```
a= "Ajay"
b= "Vijay"
print(f"{b} and {a} both are the best friend")
```

2. Formatted String Formatting Using % Operator

- Python allows us to use the format specifiers used in C's printf statement.
- The format specifiers in Python are treated in the same way as they are treated in C.

- However, Python provides an additional operator %, which is used as an interface between the format specifiers and their values.
- **In other words, we can say that it binds the format specifiers to the values.**

Example:

In [ ]:
```
Integer = 10;
Float = 1.290
String1 = "Milind"
print(" Hi I am Integer ... My value is %d \n Hi I am float ... \
My value is %f \n Hi I am string ... My value is %s"%(Integer,Float,String1))
#  printf("................." ,Integer,Float,String1 );
```

In [ ]:
```

```

String Functions

Method	Description
capitalize()	It capitalizes the first character of the String. This function is deprecated in python3
casefold()	It returns a version of s suitable for case-less comparisons.
center(width, fillchar)	It returns a space padded string with the original string centred with equal number of left and right spaces.
count(string, begin, end)	It counts the number of occurrences of a substring in a String between begin and end index.
decode(encoding = 'UTF8', errors = 'strict')	Decodes the string using codec registered for encoding.
encode()	Encode S using the codec registered for encoding. Default encoding is 'utf-8'.
endswith(suffix ,begin=0, end=len(string))	It returns a Boolean value if the string terminates with given suffix between begin and end.
expandtabs(tabsize = 8)	It defines tabs in string to multiple spaces. The default space value is 8.
find(substring, beginIndex, endIndex)	It returns the index value of the string where substring is found between begin index and end index.
format(value)	It returns a formatted version of S, using the passed value.
index(subsring, beginIndex, endIndex)	It throws an exception if string is not found. It works same as find() method.

String Functions:1:

capitalize()

- It capitalizes the first character of the String.

- This function is deprecated in python3

In [ ]:
```
str1 = "ezylearning"

str2 = str1.capitalize()

print("Old value:", str1)
print("New value:", str2)
```

String Functions:2:

casefold()

- It returns a version of s suitable for case-less comparisons.

In [ ]:
```
str = "EZYlearning"

str2 = str.casefold()

print("Old value:", str)
print("New value:", str2)
```

String Functions:3:

center(width ,fillchar)

- It returns a space padded string with the original string centred with equal number of left and right spaces.

In [ ]:
```
# Example-1

str = "Hello PSIT"

str2 = str.center(30)

print("Old value:", str)
print("New value:", str2)

str3 = str.center(30,'*')
print("New filled value:", str3)
```

String Functions:4:

count(string,start,end)

- It counts the number of occurrences of a substring in a String between begin and end index.

Syntax: count(sub[, start[, end]])

Parameters

- sub (required)
- start (optional)
- end (optional)

Return Type

- It returns number of occurrences of substring in the range.

In [ ]:
```
str = "Hello PSIT"
str2 = str.count('l')
# Displaying result
print("occurences:", str2)
```

In [ ]:
```
str = "ab bc ca de ed ad da  ab bc ca"
ct = str.count('a')
# Displaying result
print("occurences:", ct)
```

In [ ]:
```
str = "ab bc ca de ed ad da ab c ca"
ct = str.count('a')
print("occurences:", ct)

ct = str.count('a', 3) # 3-> from where to start
print("occurences:", ct)

ct = str.count('a', 3, 8)  # 3--> start  8--> end
print("occurences:", ct)
```

String Functions:5:

Decode()

- Decodes the string using codec registered for encoding.

In [ ]:
```
# Define a bytes object
str1='Hello, Python!'
print(str1)

bytes_data = b'Hello, Python!'
print(bytes_data)

# Decode bytes object to string using UTF-8 encoding
text = bytes_data.decode('utf-8')

print(text)  # Output: Hello, Python!
```

String Functions:6:

encode() function

- Python encode() method encodes the string according to the provided encoding standard.

- By default Python strings are in unicode form but can be encoded to other standards also.
- Encoding is a process of converting text from one standard code to another.

Syntax

encode(encoding="utf-8", errors="strict")

Parameters

- encoding : encoding standard, default is UTF-8
- errors : errors mode to ignore or replace the error messages.
- Both are optional. Default encoding is UTF-8.

Error parameter has a default value strict and allows other possible values 'ignore', 'replace', 'xmlcharrefreplace', 'backslashreplace' etc too.

Return Type

- It returns an encoded string.

In [45]:
```
# Example-2

# Latin caharacter Ë into default encoding.
# Variable declaration

str = "HËLLO"
ecd = str.encode()

print("Old value", str)
print("Encoded value", ecd)
```

```
Old value HËLLO
Encoded value b'H\xc3\x8bLLO'
```

In [ ]:
```
# Example 3
#We are encoding latin character into ascii, it throws an error. See the example below

str = "HËLLO"
ecd = str.encode("ascii")

# Displaying result
print("Old value", str)
print("Encoded value", ecd)
```

In [ ]:
```
# Example 4
# If we want to ignore errors, pass ignore as the second parameter.

str = "HËLLO"
ecd = str.encode("ascii","ignore")

print("Old value", str)
print("Encoded value", ecd)
```

In [ ]:
```
# Example 5
#It ignores error and replace character with ? mark.
```

```
str = "HËLLO"
ecd = str.encode("ascii","replace")

print("Old value", str)
print("Encoded value", ecd)
```

String Functions:7:

endswith() Method

- It returns true of the string ends with the specified substring, otherwise returns false. ### **Syntax:** #### **endswith(suffix[, start[, end]]) Parameters**
- suffix : a substring
- start : start index of a range (**Optional**)
- end : last index of the range(**Optional**)
- **Start and end both parameters are optional.**

Return Type

- It returns a boolean value either True or False.

In [ ]:
```
#Example 1
#A simple example which returns true because it ends with dot (.).

str = "Hello this is EzyLearning@PSIT."
isends = str.endswith("PSIT.")

print(isends)
```

In [ ]:
```
# Example 2
# It returns false because string does not end with is.
str = "Hello this is EzyLearning@PSIT."
isends = str.endswith("is")

print(isends)
```

In [ ]:
```
# Example 3
#Here, we are providing start index of the range from where method starts searching.

str = "Hello this is EzyLearning@PSIT."
isends = str.endswith("is",10)  # 10--> start index

print(isends)
```

In [ ]:
```
# Example 4
# It returns true because third parameter stopped
# the method at index 13.


str = "Hello this is Kanpur."
isends = str.endswith("is",0,13)
```

```
print(isends)
```

String Functions:8:

expandtabs() Method

Python expandstabs() method replaces all the characters by sepecified spaces. By default a single tab expands to 8 spaces which can be overridden according to the requirement.

We can pass 1, 2, 4 and more to the method which will replace tab by the these number of space characters.

Syntax

- **expandtabs(tabsize=8)**

Parameters

- tabsize : It is optional and default value is 8.

Return Type

- It returns a modified string.

In [ ]:
```
# Example-1

str = "Welcome \t to \t the \t PSIT."

str2 = str.expandtabs(10)

print(str2)
```

String Functions:9:

find() Method

- Python finds () method finds a substring in the whole String and returns the index of the first match.
- It returns -1 if the substring does not match.

Syntax: string.find(substring [, start[, end]])

Parameters:

- **String** is the String in which you want to search for the substring
- **Substring** is the String you want to search for
- **Start** is the index from which the search should begin (**optional, defaults is 0**)
- **End** is the index at which the search should end (**optional, defaults to the end of the String**)

Example 1:

In [ ]:
```
string = "Hello, world!"
substring = "world"
index = string.find(substring)
print(index)
```

In [ ]:
```
index = "Hello, world!".find("world" )
print(index)
```

In [ ]:
```
# Example-2

string = "Hello, world!"
substring = "l"
start = 3
end = 8
index = string.find(substring, start,end)
print(index)
```

In [ ]:
```
# Example-3

string = "Hello, world!"
substring = "python"
index = string.find(substring)
print(index)
```

In [ ]:
```
# Example-4

## Finding a substring in a string with multiple occurrences:

string = "Hello, world!"
substring = "l"
index = string.find(substring)
while index != -1:
    print(index)
    index = string.find(substring, index + 1)
```

In [ ]:
```
# now jump string function set-2
```

String Functions:10:

String format() Method

- format() method, which helps the developers to format and customize data output
- One can enhance Python programming abilities by effectively using format().

Syntax

format(*args, **kwargs)

Parameters

- *args : substring
- **kwargs : start index a range

Return Type

- It returns a formatted string.

Simple Formatting:

- A string with one or more placeholders is needed to begin using the format() method.
- Placeholders are set apart by curly braces, with optional positional or named arguments inside.
- **Example**

In [ ]:
```
fname = "EzyLearning"
age = 42
formatted_string = "My firm\'s name is {} and I'm {} years old.".format(fname, age)
print(formatted_string)
```

String format() method : As Positional and Named Arguments:

- The `configuration()` strategy supports both positional and named arguments, offering flexibility in how you provide values..

- Named arguments are assigned based on their names, while positional arguments are filled in the order they are specified in the `format()` method.

- **Example**

In [ ]:
```
product = "LED Television"
price = 14999.999
item_details = "The {0} costs ${1:.2f}".format(product, price)
print(item_details)


#
# The positional placeholders "0" and "1" in this example correspond to the order of t
# passed to the format() method.
# The : .2f inside the subsequent placeholder guarantees that the cost is shown with t
# Also if your original cost is having more than TWO decimal ponts then the whole pric
# off to next number.
#
```

String Functions Set-2

Method	Description
split (str, num=string. count (str))	This function splits the string according to the delimiter str. The string splits according to the space if the delimiter is not provided. It returns the list of substring concatenated with the delimiter.
strip([chars])	It is used to perform lstrip() and rstrip() on the string.
swapcase()	It inverts case of all characters in a string.
title()	It is used to convert the string into the title-case i.e., The string meEruT will be converted to Meerut.
zfill(width)	Returns original string leftpadded with zeros to a total of width characters; intended for numbers, zfill() retains any sign given (less one zero).

String Function:12:

The split() Function

- This function splits the string according to the delimiter str.
- The string splits according to the space if the delimiter is not provided.
- It returns the list of substring concatenated with the delimiter.

Syntax:

split(sep=None, maxsplit=-1)

Parameters

sep: A string parameter acts as a seperator.

maxsplit: Number of times split perfome.

Return

It returns a comma separated list.

In [ ]:
```
## Example-1
str1 = "Python is a programming language"

str2 = str1.split()

print(str1)
print(str2)
```

In [ ]:
```
## Example-2
str1 = "Python is a programming language"

str2 = str1.split("Python")
```

```
print(str1)
print(str2)
```

In [ ]:
```
## Example-3
str1 = "abc@gmail.com"

str2 = str1.split("@")

print(str1)
print(str2)
```

In [ ]:
```
## Example-4
str1 = "Python is a programming language"

str2 = str1.split('o')

print(str1)
print(str2)
```

In [ ]:
```
## Using the maxsplit parameter
## maxsplit specifies the maximum number of splits that can be made.

string = "Milind,Bhatt,Python, Programming"
result = string.split(",",2)
print(result)
```

In [ ]:
```
## Example-4
str = "Java is a programming language"

str2 = str.split('a',1)
print(str2)

str2 = str.split('a',3)
print(str2)
```

Strip() Function:

- It is used to perform lstrip() and rstrip() sametime on the string.
- It removes the leading and trailing mentioned charactors from the string.

In [ ]:
```
## Example-5 : strip() function
# Example string with leading and trailing characters
text = ",,,$$ Hello, World!!!,,,"

# Remove leading and trailing commas and exclamation marks
cleaned_text = text.strip(",!$")

print(cleaned_text)  # Output: Hello, World
```

Method	Description
join(seq)	It merges the strings representation of the given sequence.
len(string)	It returns the length of a string.
lower ()	It converts all the characters of a string to Lower case.

Method	Description
upper ()	It converts all the characters of a string to upper case.
replace (old, new [, count])	It replaces the old sequence of characters with the new sequence. The max characters are replaced if max is given.
zip(iterables)	a built-in function that allows you to combine multiple iterables (like lists, tuples, or strings) into a single iterable of tuples. Each tuple contains elements from the corresponding positions of the input iterables

In [29]:
```
## Example 1 : join is reverse of split() function
lst = ['Hello', 'World', 'Python', 'Programming']
r = ' '.join(lst)
print(r)
```

```
Hello World Python Programming
```

In [31]:
```
## Example 2
str = ":"   # string
list = ['1','2','3']    # iterable
s2 = str.join(list)
print(s2)
```

```
1:2:3
```

In [37]:
```
str = "Java is a programming language"
str1=str.split('a')
print(str1)
st='a'
result=st.join(str1)
print(result)
```

```
['J', 'v', ' is ', ' progr', 'mming l', 'ngu', 'ge']
Java is a programming language
```

In [39]:
```
## Example -3

str = "------->  "   # string
list = {'Algol','c','java','c#','asp.NET','c++','python'}    # iterable --> sets are u


s2 = str.join(list)

print(s2)
```

```
asp.NET------->  c++-------->  python------->  c------->  Algol------->  c#------->  j
ava
```

In [41]:
```
str = "123456"

ln =len(str)
print(ln)
```

```
6
```

String Functions:11:

String replace() Method

- Return a copy of the string with all occurrences of substring old replaced by new.
- If the optional argument count is given, only the first count occurrences are replaced. ### Syntax: **replace(old, new[, count])**
- **Parameters**
- **old:** Here this parameter is used for old string which will be replaced.
- **new:** Here this parameter is used for new string which will replace the old string.
- **count:** The count parameter is used to describe the number of times to process the replace.

In [ ]:
```
str1 = "Apple is a fruit"

str2 = str1.replace(str1,"Tomato is also a fruit")

print(str2)
```

In [ ]:
```
str1 = "Apple is a fruit"

str1 = str1.replace(str1,"Tomato is also a fruit")

print(str1)
```

In [ ]:
```
## Example-2
str1 = "Java is a programming language."

str2 = str1.replace("Java","Python")


print("The Old String is: \n\t",str1)
print("The new String is:  \n\t",str2)
```

In [ ]:
```
## Example-2
str1 = "Java is a programming language. Java is Opps based Language"

str2 = str1.replace("Java","Python",1)


print("The Old String is: \n\t",str1)
print("The new String is:  \n\t",str2)
```

String Functions:12:

The zip() functio:

- The zip() a built-in function allows to combine multiple iterables (like lists, tuples, or strings) into a single iterable of tuples.
- Each tuple contains elements from the corresponding positions of the input iterables #### Syntax: #### `zip(*iterables)` #### Parameters: #### **iterables:**
  - One or more iterable objects (e.g., lists, tuples, strings). #### **Return Value**
  - Returns an iterator of tuples, where the i-th tuple contains the i-th element from each of the input iterables.

In [ ]:
```
keys = ['a', 'b', 'c']
values = [1, 2, 3]

# Using zip() and dict() to create a dictionary
result_dict = dict(zip(keys, values))
print(result_dict)
```

In [ ]:
```
keys = ['a', 'b', 'c','d','e']
values = [1, 2, 3,4]

# zipping will only follow 1:1 mapping format,
#if values or key are imbalance then, only min will be zipped.

# Using zip() and dict() to create a dictionary
result_dict = dict(zip(keys, values))
print(result_dict)
```

Method	Description
isalnum()	It returns true if the characters in the string are alphanumeric i.e., alphabets or numbers and there is at least 1 character. Otherwise, it returns false.
isalpha()	It returns true if all the characters are alphabets and there is at least one character, otherwise False.
isdecimal ()	It returns true if all the characters of the string are decimals.
isdigit()	It returns true if all the characters are digits and there is at least one character, otherwise False.
isidentifier()	It returns true if the string is the valid identifier.
islower()	It returns true if the characters of a string are in lower case, otherwise false.
isnumeric()	It returns true if the string contains only numeric characters.
isprintable()	It returns true if all the characters of s are printable or s is empty, false otherwise.
isupper()	It returns false if characters of a string are in Upper case, otherwise False.
isspace()	It returns true if the characters of a string are white-space, otherwise false.
istitle()	It returns true if the string is titled properly and false otherwise. A title string is the one in which the first character is upper-case whereas the other characters are lower-case.

Python String isalnum() Method Example

It returns True even the string is full of digits. See the example.

In [ ]:
```
str = "123z456"

str2 = str.isalnum()
str3 = str.isnumeric()
print(str2)
print(str3)
```

In [ ]:
```
str = "123456"

str2 = str.isdecimal()
print(str2)
```

In [ ]:
```
#str = "12ZZ3456"
str = "abCd"
str2 = str.islower()
print(str2)
```

Less Frequently used String Functons of Python String

Method	Description
partition()	It searches for the separator sep in S, and returns the part before it, the separator itself, and the part after it. If the separator is not found, return S and two empty strings.
maketrans()	It returns a translation table to be used in translate function.
rfind(str,beg=0,end=len(str))	It is similar to find but it traverses the string in backward direction.
rindex(str,beg=0,end=len(str))	It is same as index but it traverses the string in backward direction.
rjust(width,[,fillchar])	Returns a space padded string having original string right justified to the number of characters specified.
rstrip()	It removes all trailing whitespace of a string and can also be used to remove particular character from trailing.

rsplit(sep=None, maxsplit = -1) |It is same as split() but it processes the string from the backward direction. It returns the list of words in the string. If Separator is not specified then the string splits according to the white-space. |splitlines(num=string.count('\n')) |It returns the list of strings at each line with newline removed. |startswith(str,beg=0,end=len(str)) |It returns a Boolean value if the string starts with given str between begin and end. |rpartition() |It splits the string from the last occurrence of parameter and returns a tuple. The tuple contains the three parts before the separator, the separator itself, and the part after the separator |translate(table,deletechars = '') |It translates the string according to the translation table passed in the function .

Program-1

Write a Python Program to Remove Punctuations From a String

In [ ]:
```
# define punctuation
punctuations = '''!()-[]{};:'"\,<>./?@#$%^&*_~'''

#str1 = "Hello!!!, Yes this is the right time to: learn!!!! ----> python."
str1=input("Enter Your String : ")

# remove punctuation from the string
```

```
clean_str = ""
for ch in str1:
    if ch not in punctuations:
        clean_str = clean_str + ch

print(clean_str)
```

Program-2

Write a Python Program to Sort Words in Alphabetic Order

In [ ]:
```
# Program to sort alphabetically the words form a string provided by the user

str1= "Hey, Yes This is the right TIME to Learn Python"

# split and convert each word into lower case
words = [word.upper() for word in str1.split()]

words.sort()

print("The sorted words are:")
for word in words:
    print(word)
```

Program-3

Write a Python Program to Count the Number of Each Vowel

In [ ]:
```
# Program to count the number of each vowels

# string of vowels
vowels = 'aeiou'

str1 = 'Hey, Yes This is the right TIME to Learn Python'

# make it suitable for caseless comparisions
str1 = str1.casefold()

for v in vowels:
    count=0
    for ch in str1:
        if ch==v:
            count+=1
    print(v,count)

# using list comprehension
#counts = [sum(1 for ch in str1 if ch == v) for v in vowels]
#print(v,[sum(1 for ch in str1 if ch == v) for v in vowels])
```

Program-4

Write a Python Program to Check If Two Strings are Anagram

- Two strings are said to be anagram if we can form one string by arranging the characters of another string.
- For example, Race and Care. Here, we can form Race by arranging the characters of Care.
- We solve this with sorted() function.
- Some anagram words
- - care and race
- - Restful and fluster
- - Cheater and teacher
- - Planter and replant

In [ ]:
```
str1 = "Cheater"
str2 = "Teacher"

# convert both the strings into lowercase
str1 = str1.lower()
str2 = str2.lower()

# check if length is same
if(len(str1) == len(str2)):

    # sort the strings
    sorted_str1 = sorted(str1)
    sorted_str2 = sorted(str2)

    # if sorted char arrays are same
    if(sorted_str1 == sorted_str2):
        print(str1 + " and " + str2 + " are anagram.")
    else:
        print(str1 + " and " + str2 + " are not anagram.")

else:
    print(str1 + " and " + str2 + " are not anagram.")
```

==========Python String End==========

In [ ]:

In [ ]:
```
# what will be the output of -

2**1**2*2**3
```

In [ ]:

In [ ]:

In [ ]: