Module5_Strings

October 8, 2024

1 String Creation

[2]:
```
str2='Hi, Welcome'
str2
```

[2]:
```
'Hi, Welcome'
```

[3]:
```
str1 = "HELLO PYTHON"
str1
```

[3]:
```
'HELLO PYTHON'
```

[5]:
```
str3="""Hello
        World"""
print(str3)
```

```
Hello
        World
```

[6]:
```
str3="""Hello
World"""
print(str3)
```

```
Hello
World
```

[7]:
```
str3="""Hello
        World"""
str3
```

[7]:
```
'Hello\n        World'
```

[11]:
```
str4='''Hello
        World'''
print(str4)
```

```
Hello
        World
```

[12]:
```
str4='''Hello
World'''
print(str4)
```

```
Hello
World
```

[13]:
```
str4='''Hello
        World'''
str4
```

[13]: 'Hello\n   World'

[14]:
```
str4='''Hello
World'''
str4
```

[14]: 'Hello\nWorld'

[28]:
```
mystr = ('Happy '
         'Monday '
 'Everyone')
print(mystr)
```

```
Happy Monday Everyone
```

[18]:
```
mystr2 = 'Woohoo '
mystr2 = mystr2*5  #replication operator
mystr2
```

[18]: 'Woohoo Woohoo Woohoo Woohoo Woohoo '

2 Iterating through a String

[29]:
```
str1="Happy Life"
for i in str1:
    print(i)
```

```
H
a
p
p
y

L
i
f
e
```

[30]:
```
str1="Happy Life"
for i in range(len(str1)):
    print(i)
```

```
0
1
2
3
4
5
6
7
8
9
```

[31]:
```
str1="Happy Life"
for i in range(str1):
    print(i)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-31-9b82d0540dd6> in <module>
      1 str1="Happy Life"
----> 2 for i in range(str1):
      3     print(i)

TypeError: 'str' object cannot be interpreted as an integer
```

[37]:
```
str1="Happy Life"
n=len(str1)
i=0
while i<=n-1:
    print(str1[i])
    i+=1
```

```
H
a
p
p
y

L
i
f
e
```

[40]:
```
str1="Happy Life"
for i, x in enumerate(str1):
    print(i,x)
```

```
0 H
1 a
2 p
3 p
4 y
5
6 L
7 i
8 f
9 e
```

[44]:
```
str1="Happy Life"
for i, x in enumerate(str1,start=5):
    print(i,x)
```

```
5 H
6 a
7 p
8 p
9 y
10
11 L
12 i
13 f
14 e
```

[42]:
```
str1="Happy Life"
for i in enumerate(str1):
    print(i)
```

```
(0, 'H')
(1, 'a')
(2, 'p')
(3, 'p')
(4, 'y')
(5, ' ')
(6, 'L')
(7, 'i')
(8, 'f')
(9, 'e')
```

[64]:
```
list(enumerate(str1))
```

[64]:
```
[(0, 'H'),
 (1, 'a'),
```

```
 (2, 'p'),
 (3, 'p'),
 (4, 'y'),
 (5, ' '),
 (6, 'L'),
 (7, 'i'),
 (8, 'f'),
 (9, 'e'),
 (10, ','),
 (11, 'W'),
 (12, 'e'),
 (13, 'l'),
 (14, 'c'),
 (15, 'o'),
 (16, 'm'),
 (17, 'e')]
```

3 In-built functions in strings

4

len() – Find out the length of characters in string. min() – Smallest value in a string based on ASCII values. max()- Largest value in a string based on ASCII values.

[4]:
```
str1= "If you want to shine like a sun, first burn like a sun."
print(len(str1))
```

```
55
```

[3]:
```
str1= "If you want to shine like a sun, first burn like a sun."
print(len(str1))
print(max(str1))
print(min(str1))
```

```
55
y
```

[5]:
```
min("abcdefghijklmnopqrstuvwxyz")
```

[5]: 'a'

[7]:
```
max("abcdefghijklmnopqrstuvwxyz")
```

[7]: 'z'

[8]:
```
str1= "Ifyouwanttoshinelikeasun,firstburnlikeasun."
print(len(str1))
```

```
print(max(str1))
print(min(str1))
```

```
43
y
,
```

[9]:
```
str1= "Ifyouwanttoshinelikeasunfirstburnlikeasun."
print(len(str1))
print(max(str1))
print(min(str1))
```

```
42
y
.
```

[10]:
```
str1= "HELLo"
print(len(str1))
print(max(str1))
print(min(str1))
```

```
5
o
E
```

5 String indexing

5.1 Method 1: Using the slice() method

The slice() constructor creates a slice object representing the set of indices specified by range(start, stop, step).

Syntax:

slice(stop) slice(start, stop, step)

Parameters: start: Starting index where the slicing of object starts. stop: Ending index where the slicing of object stops. step: It is an optional argument that determines the increment between each index for slicing. Return Type: Returns a sliced object containing elements in the given range only.

[11]:
```
String = 'ASTRING'

# Using slice constructor
s1 = slice(3)
s2 = slice(1, 5, 2)
s3 = slice(-1, -12, -2)

print(String[s1])
print(String[s2])
```

```
print(String[s3])
```

```
AST
SR
GITA
```

[13]:
```
String = 'ASTRING'

# Using slice constructor
s1 = slice(3)
s2 = slice(1, 5, 2)
s3 = slice(-1, -12, -2)

s1
```

[13]:
```
slice(None, 3, None)
```

[16]:
```
String = 'ASTRING'

# Using slice constructor
s1 = slice(3)
s2 = slice(1, 5, 2)
s3 = slice(-1, -12, -2)

String[s1]
```

[16]:
```
'AST'
```

[17]:
```
String = 'ASTRING'

# Using slice constructor
s1 = slice(3)
s2 = slice(1, 5, 2)
s3 = slice(-1, -12, -2)

String(s1)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-17-d3b31a25b862> in <module>
      6 s3 = slice(-1, -12, -2)
      7
----> 8 String(s1)

TypeError: 'str' object is not callable
```

5.2 Method 2: Using the List/array slicing [::] method

arr[start:stop] # items start through stop-1 arr[start:] # items start through the rest of the array arr[:stop] # items from the beginning through stop-1 arr[:] # a copy of the whole array arr[start:stop:step] # start through not past stop, by step

[57]:
```
str1="FAIL means 'First attempt in learning'"
print(str1)
print(len(str1))
```

```
FAIL means 'First attempt in learning'
38
```

[58]:
```
print(str1[0]) # First character in string "str1"
print(str1[-1]) # Last character in string
print(str1[-4])
print(str1[6]) #Fetch 7th element of the string
```

```
F
'
i
e
```

[51]:
```
print(str1[len(str1)-1])# Last character in string using len function
```

```
'
```

[55]:
```
str1="Happy Life,Welcome"
print(str1[0:5])
print(str1[6:12])
print(str1[-4:]) # Retreive last four characters of the string
print(str1[:4])  # Retreive first four characters of the string
print(str1[::-1]) #reversal
print(str1[-4:-8]) #no output
print(str1[-8:-4])
```

```
Happy
Life,W
come
Happ
emocleW,efiL yppaH

,Wel
```

[56]:
```
str1="Happy Life,Welcome"
str1[-4:-8]
```

[56]: ''

6 Update & Delete String

#Strings are immutable which means elements of a string cannot be changed once they have been assigned.

[59]:
```
str1="Happy Life,Welcome"
str1[0:3]="Hai"
print(str1)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-59-f41a7f68a655> in <module>
      1 str1="Happy Life,Welcome"
----> 2 str1[0:3]="Hai"
      3 print(str1)

TypeError: 'str' object does not support item assignment
```

[60]:
```
str2="Welcome to home"
print(str2)
del str2
print(str2)
```

```
Welcome to home
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-60-ad75d97ca58d> in <module>
      2 print(str2)
      3 del str2
----> 4 print(str2)

NameError: name 'str2' is not defined
```

7 String concatenation

[61]:
```
s1 = "Hello"
s2 = "Asif"
s3 = s1 + s2
print(s3)
```

```
HelloAsif
```

[62]:
```
s1 = "Hello"
s2 = "Asif"
```

```
s3 = s1 + " " + s2
print(s3)
```

```
Hello Asif
```

8 String Membership

[65]:
```
mystr1 = "Hello Everyone"
print ('Hello' in mystr1)
print ('Everyone' in mystr1)
print ('Hi' in mystr1)
```

```
True
True
False
```

[66]:
```
mystr1 = "Hello Everyone"
print ('Hello' not in mystr1)
print ('Everyone' not in mystr1)
print ('Hi' not in mystr1)
```

```
False
False
True
```

9 String functions

10 String partioning

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

string.partition(value)

The first element contains the part before the specified string.

The second element contains the specified string.

The third element contains the part after the string.

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

[4]:
```
str1="Welcome to the Python lab"
x=str1.partition("the")
print(type(x))
print(x) # the output is  a tuple
```

```
<class 'tuple'>
('Welcome to ', 'the', ' Python lab')
```

[6]:
```
str1="Welcome to the Python lab"
x=str1.partition("and")
print(type(x))
print(x) # the output is  a tuple
```

```
<class 'tuple'>
('Welcome to the Python lab', '', '')
```

[7]:
```
str1="Welcome to the Python lab"
x=str1.partition('hi')
print(type(x))
print(x) # the output is  a tuple
```

```
<class 'tuple'>
('Welcome to the Python lab', '', '')
```

[11]:
```
str2="The earth is tHe beautiful place to live"
x=str2.partition('tHe') #checks for the perfect match
print(type(x))
print(x)
```

```
<class 'tuple'>
('The earth is ', 'tHe', ' beautiful place to live')
```

[13]:
```
str1="Welcome to the 'Welcome' Python lab"
x=str1.partition('Welcome')
print(type(x))
print(x)
```

```
<class 'tuple'>
('', 'Welcome', " to the 'Welcome' Python lab")
```

[14]:
```
str1="I like to live, I like to live"
x=str1.partition('live')
print(type(x))
print(x)
```

```
<class 'tuple'>
('I like to ', 'live', ', I like to live')
```

11 rpartition() method

It searches for the last occurrence of a specified string, and splits the string into a tuple containing three elements.

The first element contains the part before the specified string.

The second element contains the specified string.

The third element contains the part after the string.

[15]:
```
str1="I like to live, I like to live"
x=str1.rpartition('live')
print(type(x))
print(x)
```

```
<class 'tuple'>
('I like to live, I like to ', 'live', '')
```

[16]:
```
str1="I like to live, I like to live my life"
x=str1.rpartition('live')
print(type(x))
print(x)
```

```
<class 'tuple'>
('I like to live, I like to ', 'live', ' my life')
```

[17]:
```
str1="I like to live, I like to live"
x=str1.rpartition('and')
print(type(x))
print(x)
```

```
<class 'tuple'>
('', '', 'I like to live, I like to live')
```

12 rsplit()

string.rsplit(separator, maxsplit)

The rsplit() method in Python is used to split a string into a list, just like split(), but with a key difference: it splits the string from the right side rather than the left. This is particularly useful when you want to limit the number of splits starting from the right side.

[26]:
```
text = "apple, banana, cherry"
result = text.rsplit(", ",1 )  # Split once from the right
print(type(result))
print(result) # result is  a list
```

```
<class 'list'>
['apple, banana', 'cherry']
```

[35]:
```
text = "apple,banana,cherry"
result = text.rsplit(", ",1 )  # Split once from the right , Note the difference
print(type(result))
print(result) # result is  a list
```

```
<class 'list'>
['apple,banana,cherry']
```

[38]:
```
text = "apple,banana,cherry,grapes,citrus"
result = text.rsplit(",", 2)  # Split once from the right
```

```
print(type(result))
print(result)
```

```
<class 'list'>
['apple,banana,cherry', 'grapes', 'citrus']
```

[41]:
```
text = "a-b-c-d-e-f"
result = text.rsplit("-", 2)  # Split at most 2 times from the right
print(type(result))
print(result)
```

```
<class 'list'>
['a-b-c-d', 'e', 'f']
```

[40]:
```
text = "a-b-c-d-e-f"
result = text.split("-", 2)  # Split at most 2 times from the left
print(type(result))
print(result)
```

```
<class 'list'>
['a', 'b', 'c-d-e-f']
```

13 Practice:

You are given a string that represents a file path, such as:

"/home/user/documents/project/file.txt"

Task 1: Using split() Write a Python program that splits this string using / as a separator to break it into individual directories and the filename. Print the resulting list of directories and the filename.

Task 2: Using rsplit() Modify the program to split the string into two parts: The file path before the last /. The filename (the part after the last /).

[42]:
```
# Input string (file path)
file_path = "/home/user/documents/project/file.txt"

# Task 1: Using split()
path_split = file_path.split('/')
print("Using split():", path_split)

# Task 2: Using rsplit() with maxsplit = 1
path_rsplit = file_path.rsplit('/', 1)
print("Using rsplit():", path_rsplit)
```

```
Using split(): ['', 'home', 'user', 'documents', 'project', 'file.txt']
Using rsplit(): ['/home/user/documents/project', 'file.txt']
```

[48]:
```
str2 = "one two Three one two two three"
x=str2.count('two')
```

```
y=str2.count('one')
print(x)
print(y)
```

```
3
2
```

[74]:
```
str2 = "one two Three one two two three"
print(len(str2))
x1=str2.count('two',10,30)
x2=str2.count('two')
y=str2.count('one')
print(x1)
print(x2)
print(y)
```

```
31
2
3
2
```

[50]:
```
str2.startswith('one')
```

[50]:
```
True
```

[51]:
```
str2.startswith('two')
```

[51]:
```
False
```

[52]:
```
str2.endswith('one')
```

[52]:
```
False
```

[53]:
```
str2.endswith('three')
```

[53]:
```
True
```

14 f-string and format()

F-String was introduced in Python 3.6, and is now the preferred way of formatting strings.

Before Python 3.6 we had to use the format() method.

f-strings:

To format values in an f-string, add placeholders {}, a placeholder can contain variables, operations, functions, and modifiers to format the value.

A placeholder can also include a modifier to format the value.

A modifier is included by adding a colon : followed by a legal formatting type, like .2f which means fixed point number with 2 decimals:

[66]:
```
name = "Jane"
age = 25

print("Hello, %s! You're %s years old." % (name, age)) #note the placement of␣
 ↪operators
```

```
Hello, Jane! You're 25 years old.
```

[54]:
```
price = 59  #f-string
txt = f"The price is {price:.2f} dollars"
print(txt)
```

```
The price is 59.00 dollars
```

[55]:
```
price = 59.425
txt = f"The price is {price:.2f} dollars"
print(txt)
```

```
The price is 59.42 dollars
```

[56]:
```
price = 59
print(f"The price is {price:.2f} dollars")
```

```
The price is 59.00 dollars
```

[57]:
```
price = 59
tax = 0.25
txt = f"The price is {price + (price * tax)} dollars"  #using operators
print(txt)
```

```
The price is 73.75 dollars
```

[58]:
```
price = 49
txt = f"It is very {'Expensive' if price>50 else 'Cheap'}"

print(txt)
```

```
It is very Cheap
```

[60]:
```
price = 5900
txt = f"The price is {price:,} dollars"  # using comma as thousand operator
print(txt)
```

```
The price is 5,900 dollars
```

[61]:
```
price = 59000
txt = f"The price is {price:,} dollars"
print(txt)
```

```
The price is 59,000 dollars
```

[63]:
```
price = 590
txt = f"The price is {price:,} dollars"
print(txt)
```

```
The price is 590 dollars
```

[67]:
```
full_name = "Trey Hunner"
costs = [1.10, 0.30, 0.40, 2]
print(f"Variables: {full_name=}, {costs=}")
```

```
Variables: full_name='Trey Hunner', costs=[1.1, 0.3, 0.4, 2]
```

[68]:
```
num = 255
print(f"{num:d}")  # Decimal (255)
print(f"{num:b}")  # Binary (11111111)
print(f"{num:x}")  # Hexadecimal (ff)
print(f"{num:o}")  # Octal (377)
```

```
255
11111111
ff
377
```

[69]:
```
pos_num = 42
neg_num = -42
print(f"{pos_num:+}")  # +42
print(f"{neg_num:+}")  # -42, Always show the sign (either + or -).
print(f"{pos_num: }")  #  42 (with space)
```

```
+42
-42
 42
```

[75]:
```
pos_num = 42
neg_num = -42
print(f"{pos_num:+}")
print(f"{neg_num:-}")
print(f"{pos_num: }")
print(f"{neg_num: }")
```

```
+42
-42
 42
-42
```

[77]:
```
pos_num = 42
neg_num = -42
print(f"{pos_num:+}")
```

```
print(f"{pos_num:-}")
print(f"{pos_num: }")
print(f"{neg_num: }")
print(f"{neg_num:+}")
print(f"{neg_num:-}")
```

```
+42
42
 42
-42
-42
-42
```

[72]:
```
num = 42
print(f"{num:05}")  # Pads with zeros, output: 00042
```

```
00042
```

[73]:
```
pi = 3.1415926535
print(f"Pi is approximately {pi:010.3f}")  # Zero-padded, width 10, 3 decimal␣
 ↪places

number = 12345.6789
print(f"{number:,.2f}")  # Adds commas, two decimal places
```

```
Pi is approximately 000003.142
12,345.68
```

[82]:
```
import math
variable = 10
print(f"Using Numeric {variable = }")
print(f"This prints without formatting {variable}")
print(f"This prints with formatting {variable:d}")
print(f"This prints also with formatting {variable:n}")
print(f"This prints with spacing {variable:10d}\n")
variable = math.pi
print(f"Using Numeric {variable = }")
print(f"This prints without formatting {variable}")
print(f"This prints with formatting {variable:f}")
print(f"This prints with spacing {variable:20f}")
```

```
Using Numeric variable = 10
This prints without formatting 10
This prints with formatting 10
This prints also with formatting 10
This prints with spacing         10

Using Numeric variable = 3.141592653589793
This prints without formatting 3.141592653589793
```

```
This prints with formatting 3.141593
This prints with spacing             3.141593
```

14.1

The format() method can still be used, but f-strings are faster and the preferred way to format strings.

[78]:
```
# Combining string & numbers using format method
item1 = 40
item2 = 55
item3 = 77
res = "Cost of item1 , item2 and item3 are {} , {} and {}"
print(res.format(item1,item2,item3))
```

```
Cost of item1 , item2 and item3 are 40 , 55 and 77
```

[80]:
```
item1 = 40
item2 = 55
item3 = 77
res = "Cost of item3 , item2 and item1 are {2} , {1} and {0}"
print(res.format(item1,item2,item3))
```

```
Cost of item3 , item2 and item1 are 77 , 55 and 40
```

[79]:
```
quantity = 3
itemno = 567
price = 49
myorder = "I want {} pieces of item number {} for {:.2f} dollars."
print(myorder.format(quantity, itemno, price))
```

```
I want 3 pieces of item number 567 for 49.00 dollars.
```

[81]:
```
#named indexes
myorder = "I have a {carname}, it is a {model}."
print(myorder.format(carname = "Ford", model = "Mustang"))
```

```
I have a Ford, it is a Mustang.
```

15 center()

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.

[6]:
```
str2 = " WELCOME EVERYONE "
str2 = str2.center(100)
print(str2)
```

```
                                                  WELCOME EVERYONE
```

[7]:
```
str2 = " WELCOME EVERYONE "
print(len(str2))
str2 = str2.center(50)
print(str2)
```

```
18
                  WELCOME EVERYONE
```

[9]:
```
str2 = " WELCOME EVERYONE "
print(len(str2))
print(str2)
```

```
18
 WELCOME EVERYONE
```

[12]:
```
str2 = " WELCOME EVERYONE "
print(len(str2))
str2 = str2.center(25)
print(str2)
```

```
18
     WELCOME EVERYONE
```

[14]:
```
str2 = " WELCOME EVERYONE "
print(len(str2))
str2 = str2.center(15)
'''Here, in the output, the new string is unchanged, because the original string␣
 ↪length (18)
is more than the length value provided (15).
Thus, the new string returned is unchanged.
'''
print(str2)
```

```
18
 WELCOME EVERYONE
```

[4]:
```
str2 = " WELCOME EVERYONE "
print(len(str2))
str2 = str2.center(50,'*')
print(str2)
```

```
18
*************** WELCOME EVERYONE ***************
```

15.1 rjust() and ljust()

String rjust() The string rjust() method returns a new string of given length after substituting a given character in left side of original string.

The string ljust() method returns a new string of given length after substituting a given character in right side of original string.

[31]:
```
str2 = " WELCOME EVERYONE "
str2 = str2.rjust(50) # Right align the string
print(str2)
```

```
                                  WELCOME EVERYONE
```

[32]:
```
str2 = " WELCOME EVERYONE "
str2 = str2.rjust(200) # Right align the string
print(str2)
```

```
                                                                                                                  WELCOME EVERYONE
```

[16]:
```
str2 = " WELCOME EVERYONE "
str2 = str2.rjust(50,"-") # Right align the string using a specific character
print(str2)
```

```
-------------------------------- WELCOME EVERYONE
```

[33]:
```
str2 = " WELCOME EVERYONE "
str2 = str2.rjust(150,"-") # Right align the string using a specific character
print(str2)
```

```
------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------- WELCOME EVERYONE
```

[29]:
```
str2 = "WELCOME EVERYONE "
print(len(str2))
str2 = str2.ljust(200) # left align the string
print(str2)
```

```
17
WELCOME EVERYONE
```

[17]:
```
str2 = " WELCOME EVERYONE "
str2 = str2.ljust(50,"-") # left align the string using a specific character
print(str2)
```

```
 WELCOME EVERYONE --------------------------------
```

[27]:
```
str2 = "               WELCOME EVERYONE "
str2 = str2.ljust(150,"-") # left align the string using a specific character
print(str2)
```

```
                WELCOME EVERYONE ----------------------------------------------
------------------------------------------------------------------------
```

[28]:
```
str2 = "               WELCOME EVERYONE "
str2 = str2.ljust(150) # left align the string
print(str2)
```

```
                WELCOME EVERYONE
```

15.2 Python String find()

method returns the lowest index or first occurrence of the substring if it is found in a given string.

[34]:
```
str4 = "one two three four five six seven"
loc = str4.find("five") # Find the location of word 'five' in the string "str4"
print(loc)
```

```
19
```

[43]:
```
str5= 'find me if you can'
print(len(str5))
print(str5.find('me'))
print(str5.find('if'))
print(str5.find('a'))
print(str5.find('f'))
print(str5.find('k'))
```

```
18
5
8
16
0
-1
```

[2]:
```
str5= 'findmeifyoucan'
print(str5.find('if'))
```

```
6
```

[40]:
```
str5= 'find me if you can'
print(str5.index('if'))
```

```
8
```

[44]:
```
text = "Hello, world!"

print(text.find("world"))   # Output: 7
print(text.index("world"))  # Output: 7

print(text.find("Python"))  # Output: -1
```

```
print(text.index("Python")) # Raises ValueError
```

```
7
7
-1
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-44-6fd3a17c6308> in <module>
      5
      6 print(text.find("Python"))  # Output: -1
----> 7 print(text.index("Python")) # Raises ValueError

ValueError: substring not found
```

15.3

isdecimal() method supports only Decimal Numbers. isdigit() method supports Decimals, Subscripts, Superscripts. isnumeric() method supports Digits, Vulgar Fractions, Subscripts, Superscripts, Roman Numerals, Currency Numerators.

[47]:
```
mystr6 = '123456789'
print(mystr6.isalpha()) # returns True if all the characters in the text are␣
 ↪letters
print(mystr6.isalnum()) # returns True if a string contains only letters or␣
 ↪numbers (alphanumeric)
print(mystr6.isdecimal()) # returns True if all the characters are decimals (0-9)
print(mystr6.isnumeric()) # returns True if all the characters are numeric (0-9)
print(mystr6.isdigit())
```

```
False
True
True
True
True
```

[48]:
```
mystr6 = 'abcde'
print(mystr6.isalpha())
print(mystr6.isalnum())
print(mystr6.isdecimal())
print(mystr6.isnumeric())
print(mystr6.isdigit())
```

```
True
True
False
False
False
```

[49]:
```
mystr6 = 'abcde123'
print(mystr6.isalpha())
print(mystr6.isalnum())
print(mystr6.isdecimal())
print(mystr6.isnumeric())
print(mystr6.isdigit())
```

```
False
True
False
False
False
```

[51]:
```
str1=' 110'.isdigit() #if a string has a space
str1
```

[51]: False

[52]:
```
str1=' 11.0'.isdigit()
str1
```

[52]: False

[57]:
```
a=input("Enter a number:" ) #check introducing a space while inputting also
print(a.isdigit())
print(a.strip().isdigit())
```

```
Enter a number:5
True
True
```

[58]:
```
mystr7 = 'ABCDEF'
print(mystr7.isupper()) # Returns True if all the characters are in upper case
print(mystr7.islower()) # Returns True if all the characters are in lower case
```

```
True
False
```

[4]:
```
mystr8 = 'Abcdef'
print(mystr8.isupper()) # Returns True if all the characters are in upper case
print(mystr8.islower()) # Returns True if all the characters are in lower case
```

```
False
False
```

[59]:
```
mystr8 = 'abcdef'
print(mystr8.upper().isupper())
```

```
True
```

[69]:
```
mystr8 = 'ABCDEF'
print(mystr8.lower().islower())
```

```
True
```

[61]:
```
str6 = "one two three four one two two three five five six one ten eight ten␣
 ↪nine"
loc = str6.rfind("one") # last occurrence of word 'one' in string "str6"
print(loc)
```

```
51
```

[62]:
```
loc = str6.rindex("one") # last occurrence of word 'one' in string "str6"
print(loc)
```

```
51
```

[63]:
```
loc = str6.rindex("ten") # last occurrence of word 'one' in string "str6"
print(loc)
```

```
65
```

[65]:
```
txt = " abc def ghi  "
txt.rstrip()
```

[65]:
```
' abc def ghi'
```

[9]:
```
txt = " abc def ghi  "
a=txt.rstrip()
print('original:',txt)
a
```

```
original:  abc def ghi
```

[9]:
```
' abc def ghi'
```

[67]:
```
txt = "  abc def ghi   "
txt.lstrip()
```

[67]:
```
'abc def ghi   '
```

[10]:
```
txt = "  abc def ghi   "
a=txt.lstrip()
print('original:',txt)
a
```

```
original:   abc def ghi
```

[10]:
```
'abc def ghi   '
```

[11]:
```
txt = "  abc def ghi   "
a=txt.strip()
print('original:',txt)
a
```

```
original:   abc def ghi
```

[11]:
```
'abc def ghi'
```

[15]:
```
txt = "&#abc def ghi+++"
a=txt.lstrip("&#")
print('original:',txt)
a
```

```
original: &#abc def ghi+++
```

[15]:
```
'abc def ghi+++'
```

[16]:
```
txt = "&#abc def ghi+++"
a=txt.rstrip("+")
print('original:',txt)
a
```

```
original: &#abc def ghi+++
```

[16]:
```
'&#abc def ghi'
```

[1]:
```
txt = "&#abc def ghi+++&&"
a=txt.rstrip("+")
print('original:',txt)
a
```

```
original: &#abc def ghi+++&&
```

[1]:
```
'&#abc def ghi+++&&'
```

[2]:
```
txt = "&#abc def ghi+++&&"
a=txt.rstrip("&")
print('original:',txt)
a
```

```
original: &#abc def ghi+++&&
```

[2]:
```
'&#abc def ghi+++'
```

[3]:
```
txt = "&#abc def ghi+++&&"
a=txt.rstrip("+&")
print('original:',txt)
a
```

```
original: &#abc def ghi+++&&
```

[3]:
```
'&#abc def ghi'
```

[5]:
```
txt = "&#abc def ghi+++&&"
a=txt.rstrip("ghi")
print('original:',txt)
a
```

```
original: &#abc def ghi+++&&
```

[5]:
```
'&#abc def ghi+++&&'
```

15.4 replace()

[1]:
```
str2='Hi, fine'
str2.replace('fine','hine')
str2  #original string is not replaced
```

[1]:
```
'Hi, fine'
```

[2]:
```
str2='Hi, fine'
x=str2.replace('fine','hine')
x
```

[2]:
```
'Hi, hine'
```

[3]:
```
str2='Hi, fine'
x=str2.replace('Hi','Hello')
x
```

[3]:
```
'Hello, fine'
```

15.5 title(),swapcase(),istitle()

title() method returns a new string after converting the first letter of every word to upper-case(capital) and keeping the rest of the letters lowercase.

string.title() Parameters title() doesn't accept any parameter.

swapcase(): string.swapcase() The swapcase() method does not take any parameter.

[5]:
```
x='honesty is the best policy'
y=x.title()
print(x)
print(y)
```

```
honesty is the best policy
Honesty Is The Best Policy
```

[10]:
```
# title() method considers any non-alphabet as a word boundary
x = "He's smarter &stronger."
```

```
expected_string = "He's Smarter &stronger"
# after ‘(apostrophe) and & string title() method considers the start of a new␣
 ↪word
print("Expected:", expected_string, "\n Actual:", x.title())
```

```
Expected: He's Smarter &stronger
 Actual: He'S Smarter &Stronger.
```

[12]:
```
x="hello-how-are-you"
y=x.title()
print(x)
print(y)
```

```
hello-how-are-you
Hello-How-Are-You
```

[13]:
```
x="hai 4g#h%s /c\d=e"
x.title()
#In output \\D to show that it's part of the string rather than an escape␣
 ↪sequence.
```

[13]:
```
'Hai 4G#H%S /C\\D=E'
```

[77]:
```
x="hello-how-are-you"
y=x.title()
print(y)
print(y.istitle())
print(x.istitle())
```

```
Hello-How-Are-You
True
False
```

[6]:
```
x='honESTY is THE best policy'
y=x.swapcase()
print(x)
print(y)
```

```
honESTY is THE best policy
HONesty IS the BEST POLICY
```

[14]:
```
x = "He's smarter &stronger."
y=x.swapcase()
print(x)
print(y)
```

```
He's smarter &stronger.
hE'S SMARTER &STRONGER.
```

[15]:
```
x = "hai 4g#h%s /c\d=e"
y=x.swapcase()
print(x)
print(y)
```

```
hai 4g#h%s /c\d=e
HAI 4G#H%S /C\D=E
```

15.6 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.

[46]:
```
x="welcome to the city"
y=x.capitalize()
print(x)
print(y)
```

```
welcome to the city
Welcome to the city
```

[47]:
```
x="$welcome to the city"
y=x.capitalize()
print(x)
print(y)
```

```
$welcome to the city
$welcome to the city
```

[48]:
```
x="\nwelcome to the city"
y=x.capitalize()
print(x)
print(y)
```

```

welcome to the city

welcome to the city
```

[49]:
```
x="&welcome to the city"
y=x.capitalize()
print(x)
print(y)
```

```
&welcome to the city
&welcome to the city
```

[50]:
```
x="+welcome to the city"
y=x.capitalize()
print(x)
```

```
print(y)
```

```
+welcome to the city
+welcome to the city
```

15.7 zfill() method

returns a copy of the string with '0' characters padded to the left side 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 left side of the given string.

[16]:
```
x="Welcome to the new world"
print(len(x))
print(x.zfill(30))
```

```
24
000000Welcome to the new world
```

[23]:
```
#only + and - will change the output

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

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

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

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

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

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

text = "--anything%(&%(%)*^"
print(len(text))
print(text.zfill(25))
```

```
00006041
+0006041
-0006041
```

```
000*6041
000/6041
000&6041
19
-000000-anything%(&%(%)*^
```

15.8 if not

if not in string values in Python is a common and efficient way to check for empty strings.

[28]:
```
str1=""
if not str1:
    print("Empty string")
```

```
Empty string
```

[29]:
```
str1="hello, hai"
if str1:
    print("string is available")
```

```
string is available
```

15.9 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 (!”#$%&’()*+, -./:;?@[]ˆ_‘{ | }~) Space ( )

Syntax:

string.isprintable()

Parameters:

isprintable() does not take any parameters

[31]:
```
str1="Hi"
print(str1.isprintable())
```

```
True
```

[32]:
```
str1=""
print(str1.isprintable())
```

```
True
```

[33]:
```
str1="   "
print(str1.isprintable())
```

```
True
```

[34]:
```
str1="\n\t\f"
print(str1.isprintable())
```

```
False
```

[35]:
```
str1="Hello \n hi"
print(str1.isprintable())
```

```
False
```

[39]:
```
txt = "Hello! Are you #1?"

x = txt.isprintable()

print(x)
```

```
True
```

15.10 count the non-printable characters in a string

[38]:
```
str1="Hello\nhii\tcall\nname"
newstring=""
count=0
for a in str1:
    if (a.isprintable()) == False:
            count+= 1
            newstring+=' '
    else:
            newstring+= a
print(count)
print(newstring)
```

```
3
Hello hii call name
```

15.11 isspace()

method returns ‘‘True’’ if all characters in the string are whitespace characters, Otherwise, It r

Syntax: string.isspace()

[40]:
```
str1="Hello"
print(str1.isspace())
```

```
False
```

[41]:
```
str1=""
print(str1.isspace())
```

```
False
```

[42]:
```
str1="   "
print(str1.isspace())
```

```
True
```

[43]:
```
str1="Hello\n"
print(str1.isspace())
```

```
False
```

[44]:
```
str1="\nHello"
print(str1.isspace())
```

```
False
```

[45]:
```
str1="\n\f"
print(str1.isspace())
```

```
True
```

[51]:
```
str1="Hello Hi"
print(str1.isspace())
```

```
False
```

15.12 join()

is an inbuilt string function used to join elements of a sequence separated by a string separator. This function joins elements of a sequence and makes it a string.

string.join(iterable)

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.

[52]:
```
str1="Hello"
print("&".join(str1))
```

```
H&e&l&l&o
```

[55]:
```
str1='hello'
x='+'.join(str1)
print(str1)
print(x)
```

```
hello
h+e+l+l+o
```

[56]:
```
str1="Hello"
print("".join(str1))
```

```
Hello
```

[57]:
```
str1="Hello"
print(" ".join(str1))
```

```
H e l l o
```

[58]:
```
l1=['hi','come','root'] #for a list
print('#'.join(l1))
```

```
hi#come#root
```

[59]:
```
l1=['hi',1,'root']
print('#'.join(l1))
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-59-97860bc5952f> in <module>
      1 l1=['hi',1,'root']
----> 2 print('#'.join(l1))

TypeError: sequence item 1: expected str instance, int found
```

[60]:
```
t1=('hi','come','root') #for a tuple
print('#'.join(t1))
```

```
hi#come#root
```

[65]:
```
d1={'a':'hi','b':'hello','c':'well'}
print(', '.join(d1))
```

```
a, b, c
```

[66]:
```
s1={'hi','come','root'} #for a set
print('+'.join(s1))
```

```
hi+root+come
```

15.13 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).

string.splitlines([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 etc as boundaries for strings.

splitlines() splits on the following line boundaries:

Representation

Description

[2]:
```
# Python code to illustrate splitlines()
string = "Welcome everyone to\rthe beautiful \x1c world"

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

# A specified number is passed
print (string.splitlines(0))
'''When you pass 0 as an argument to splitlines(), it is treated as False
because 0 is equivalent to False in Python.
This means that line breaks are not included in the output.
'''
# True has been passed
print (string.splitlines(True))
```

```
['Welcome everyone to', 'the beautiful ', ' world']
['Welcome everyone to', 'the beautiful ', ' world']
['Welcome everyone to\r', 'the beautiful \x1c', ' world']
```

[71]:
```
string = "Welcome everyone to\rthe beautiful \x1c world"
li = string.splitlines()
print (li)
l = [len(element) for element in li]
print(l)
```

```
['Welcome everyone to', 'the beautiful ', ' world']
[19, 14, 6]
```

[72]:
```
string = "Welcome everyone to \r the beautiful \x1c world"  #check the space in␣
 ↪the output
li = string.splitlines()
print (li)
l = [len(element) for element in li]
print(l)
```

```
['Welcome everyone to ', ' the beautiful ', ' world']
[20, 15, 6]
```

[75]:
```
string = "Welcome everyone to\nthe beautiful\x1cworld"
li = string.splitlines()
print (li)
l = [len(element) for element in li]
print(l)
```

```
['Welcome everyone to', 'the beautiful', 'world']
[19, 13, 5]
```

15.14 translate

To replace or remove specific characters in a string based on a translation table. Requires a translation table created with str.maketrans() to specify character mappings.

[3]:
```
# Creating a translation table
#character mapping:maketrans(from_string, to_string)
translation_table = str.maketrans("aeiou", "12345")

# Original string
text = "hello world"

# Using the translate() method
translated_text = text.translate(translation_table)
print(translated_text)
```

```
h2ll4 w4rld
```

[4]:
```
# Creating a translation table
translation_table = str.maketrans("aeiou", "12345")

# Original string
text = "measure"

# Using the translate() method
translated_text = text.translate(translation_table)
print(translated_text)
```

```
m21s5r2
```

[6]:
```
# Create a translation table that removes specific characters
# character deletion:maketrans(from_string, to_string,delete_string)
'''
tr.maketrans("", "", "aeiou")

First parameter (""): This is an empty string, which means there are no
characters to map from. Since you are not replacing any characters,
it doesn't do anything.

Second parameter (""): This is also an empty string, which means there are
no characters to map to. It only matters if you are replacing characters.

Third parameter ("aeiou"): This is the set of characters that will be
removed from the string when you use the translate() method.
'''
translation_table = str.maketrans("", "", "aeiou")
```

```
# Original string
text = "hello world"

# Use translate() to remove the vowels
translated_text = text.translate(translation_table)
print(translated_text)
```

```
hll wrld
```

[7]:
```
# Create a translation table to map English letters to Tamil letters
translation_table = str.maketrans("aeiou", "")

# Original English text
text = "hello world"

# Use translate() to replace English vowels with Tamil letters
translated_text = text.translate(translation_table)
print(translated_text)
```

```
hll wrld
```

[15]:
```
# Create a translation table to map English letters to Tamil letters
translation_table = str.maketrans("hello", "")

# Original English text
text = "marshal"

# Use translate() to replace English vowels with Tamil letters
translated_text = text.translate(translation_table)
print(translated_text)
```

```
marsa
```

[11]:
```
print(len(""))
```

```
7
```

[12]:
```
print(len(""))
```

```
5
```

15.15 Comaparison

[16]:
```
string1 = "apple"
string2 = "Apple"

print(string1 == string2)
print(string1 != string2)
```

```
print(string1  <  string2)
print(string1  >  string2)
print(string1  <=  string2)
print(string1  >=  string2)
```

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

[22]:
```
"""the ordinal value refers to the numerical representation of a character
in the Unicode standard. You can obtain the ordinal value of a
character using the ord() function"""
ord('A')
```

[22]: 65

[21]:
```
ord('a')
```

[21]: 97

[20]:
```
ord('')
```

[20]: 128512

15.16 Escape sequence

Common Escape Sequences ’ : Single quote " : Double quote \ : Backslash : Newline : Tab : Carriage return : Backspace : Form feed Bell/Alert sound : Vertical tab : Octal value (e.g., \101 represents ‘A’) : Hexadecimal value (e.g., 41 represents ‘A’)

[37]:
```
# Using single quote escape
single_quote = 'It\'s a beautiful day!'
print("Single quote escape:", single_quote)

# Using double quote escape
double_quote = "She said, \"Hello!\""
print("Double quote escape:", double_quote)

# Using backslash escape
backslash = "This is a backslash: \\"
print("Backslash escape:", backslash)

# Newline escape sequence
new_line = "First line\nSecond line"
```

```
print("Newline escape sequence:\n", new_line)

# Tab escape sequence
tab_space = "Name\tAge\tLocation"
print("Tab escape sequence:", tab_space)

# Carriage return escape sequence
carriage_return = "Hello, World!\rNew Text"
print(carriage_return)

# Backspace escape sequence
backspace = "Hello\bWorld!"
print("Backspace escape sequence:", backspace)


# Bell/Alert escape sequence
alert_sound = "This will trigger a bell sound\a"
print("Bell/Alert escape sequence:", alert_sound)

# Vertical tab escape sequence
vertical_tab = "Column1\vColumn2"
print("Vertical tab escape sequence:", vertical_tab)

# Using octal value escape sequence
octal_value = "\101\102\103"  # Represents 'ABC'
print("Octal value escape sequence:", octal_value)

# Using hexadecimal value escape sequence
hex_value = "\x41\x42\x43"  # Represents 'ABC'
print("Hexadecimal value escape sequence:", hex_value)
```

```
Single quote escape: It's a beautiful day!
Double quote escape: She said, "Hello!"
Backslash escape: This is a backslash: \
Newline escape sequence:
 First line
Second line
Tab escape sequence: Name       Age     Location
New Text
Backspace escape sequence: HelloWorld!
Bell/Alert escape sequence: This will trigger a bell sound
Vertical tab escape sequence: Column1 Column2
Octal value escape sequence: ABC
Hexadecimal value escape sequence: ABC
```

[28]:
```
print("Hello, World!\rNew Text")
```

```
New Text
```

[36]:
```
x = "Hello, World!\rNew Text"
print("return escape sequence:", x)
```

```
New Text
```

[33]:
```
backspace = "Hello\bWorld!"
print("Backspace escape sequence:", backspace)
```

```
Backspace escape sequence: HelloWorld!
```

[34]:
```
form_feed = "Line1\fLine2"
print("Form feed escape sequence:", form_feed)
```

```
Form feed escape sequence: Line1 Line2
```

[35]:
```
alert_sound = "This will trigger a bell sound\a"
print("Bell/Alert escape sequence:", alert_sound)
```

```
Bell/Alert escape sequence: This will trigger a bell sound
```

[ ]:
```

```