



---

**Department of Science & Technology**

**Course: - PYTHON PROGRAMMING**  
**Course Code- BCAL503**

---

**Unit IV: Strings & Tuples**

**Strings** – string length, string traversal, string slices and string comparison with examples, strings are immutable, find function, string module List –list values, accessing elements, list traversal, list length, list membership, list and for loop, list operations with examples

**Tuples**-Mutability, tuple assignment, tuple as return values Dictionaries- dictionary operations, dictionary methods, aliasing and copying, counting letters using dictionaries.

**Python string length | len()**

**Syntax: len(string)**

**Return: It returns an integer which is the length of the string.**

**Python len() Example**

**len() methods with string in Python.**

```
string = "All students of final year rocks"  
print(len(string))
```

**OUTPUT**

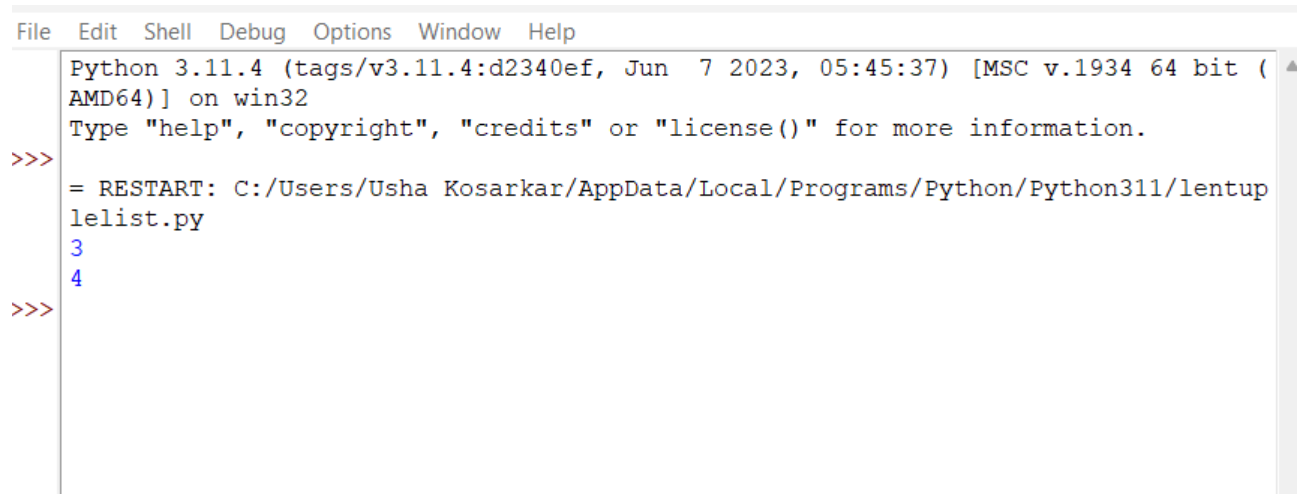
32

**Example 1: Len() function with tuples and string**  
**Here we are counting length of the tuples and list.**

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

```
# with tuple  
tup = (1,2,3)  
print(len(tup))
```

```
# with list  
l = [1,2,3,4]  
print(len(l))
```



```
File Edit Shell Debug Options Window Help
Python 3.11.4 (tags/v3.11.4:d2340ef, Jun 7 2023, 05:45:37) [MSC v.1934 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/Usha Kosarkar/AppData/Local/Programs/Python/Python311/lentuplelist.py
3
4
>>>
```

**Example : Python len() with dictionaries and sets**

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

```
dic = {'a':1, 'b': 2}  
print(len(dic))
```

```
s = { 1, 2, 3, 4}  
print(len(s))
```

```
>>>
= RESTART: C:/Users/Usha Kosarkar/AppData/Local/Programs/Python/Python311/lendic
tionaryset.py
2
4
>>>
```

## Iterate over string using for loop

Iterating over the string is simple using for loop and in operator i.e.

```
sampleStr = "Hello!!"
print("**** Iterate over string using for loop****")
for elem in sampleStr:
    print(elem)
```

```
>>>
= RESTART: C:/Users/Usha Kosarkar/AppData/Local/Programs/Python/Python311/iterat
ionaccess.py
**** Iterate over string using for loop****
H
e
l
l
o
!
!
>>>
```

## String Slicing in Python

Python slicing is about obtaining a sub-string from the given string by slicing it respectively from start to end.

### How String slicing in Python works

For understanding slicing we will use different methods, here we will cover 2 methods of string slicing, one using the in-build slice() method and another using the [:] array slice. String slicing in Python is about obtaining a sub-string from the given string by slicing it respectively from start to end.

### Python slicing can be done in two ways:

Using a slice() method

Using the array slicing [:: ] method

Index tracker for positive and negative index: String indexing and slicing in python. Here, the Negative comes into consideration when tracking the string in reverse.

0	1	2	3	4	5	6
A	S	T	R	I	N	G

-7	-6	-5	-4	-3	-2	-1
A	S	T	R	I	N	G

### 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.

```
# Python program to demonstrate
# string slicing
```

```
# String slicing
String = 'ASTRING'
```

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

```
print("String slicing")
print(String[s1])
print(String[s2])
print(String[s3])
```

```
>>> = RESTART: C:/Users/Usha Kosarkar/AppData/Local/Programs/Python/Python311/slice
python.py
String slicing
AST
SR
GITA
>>>
```

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

In Python, indexing syntax can be used as a substitute for the slice object. This is an easy and convenient way to slice a string using list slicing and Array slicing both syntax-wise and execution-wise. A start, end, and step have the same mechanism as the slice() constructor.

Below we will see string slicing in Python with examples.

### Syntax

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

## Example 1:

In this example, we will see slicing in python list the index start from 0 indexes and ending with a 2 index(stops at  $3-1=2$  ).

```
# Python program to demonstrate
# string slicing

# String slicing
String = 'PYTHONPROGRAMMING'

# Using indexing sequence
print(String[:3])
```

```
>>> = RESTART: C:/Users/Usha Kosarkar/AppData/Local/Programs/Python/Python311/Slice.py
>>> PYT
```

## Example 2:

In this example, we will see the example of starting from 1 index and ending with a 5 index(stops at  $3-1=2$  ), and the skipping step is 2. It is a good example of Python slicing string by character.

```
# Python program to demonstrate
# string slicing

# String slicing
String = 'PYTHONPROGRAMMING'

# Using indexing sequence
print(String[1:5:2])
```

```
>>> = RESTART: C:/Users/Usha Kosarkar/AppData/Local/Programs/Python/Python311/len.py
>>> YH
```

### Example 3:

In this example, we will see the example of starting from -1 indexes and ending with a -12 index(stops at 3-1=2 )and the skipping step is -2.

```
# Python program to demonstrate
# string slicing

# String slicing
String = 'PYTHONPROGRAMMING'

# Using indexing sequence
print(String[-1:-12:-2])
```

```
>>> ===== RESTART: C:/Users/Usha_Kosarkar/AppData/Local/Programs/Python/Python311/len.py =====
>>> GIMROP
>>>
```

### Example 4:

In this example, the whole string is printed in reverse order.

```
# Python program to demonstrate
# string slicing

# String slicing
String = 'PYTHONPROGRAMMING'

# Prints string in reverse
print(String[::-1])
```

## Python String Comparison

Here we will see multiple methods to compare two strings in Python. However, let us start with the first method in Python.

Method-1: Using the == operator

You can use the == operator to check if two strings are equal in Python.

```
string1 = "PYTHONPROGRAMMING"
string2 = "PYTHONPROGRAMMING"

# Compare string1 and string2 for equality
if string1 == string2:
    print("The strings are equal")
else:
    print("The strings are not equal")
```

```
>>>
=====
The strings are equal
>>> .
```

The code defines two string variables string1 and string2, both with the value “PYTHONPROGRAMMING”. It then checks if the two strings are equal using the == operator.

If the two strings are equal, the program will print “The strings are equal”, otherwise it will print “The strings are not equal”. Since the two strings are indeed equal, the output will be “The strings are equal”.

Method-2: Using the is operator

You can use the is an operator to check if two strings are the same object in memory.

```
string1 = "Python Programming"
string2 = " PYTHON PROGRAMMING"

# Check if string1 and string2 refer to the same string in memory
if string1 is string2:
    print("The strings are the same string")
else:
```



```
print("The strings are not the same string")
```

```
>>>
= RESTART: C:/Users/Usha Kosarkar/AppData/Local/Programs/Python/Python311/COMPAR
E.py
The strings are not the same string
>>>
```

## Python String find() Method

A string is a collection of characters that Python defines as contained in one two or three quotation marks. It is a line of Unicode characters with set lengths. This means that after a string has been created you can not change its value.

In Python, you can show text numbers symbols and other forms of data using strings. They are frequently used to store and modify textual material read and write files manage user input, and produce output in a certain format.

Concatenation slicing and indexing are just a few of the operations Python strings can do. Texts may also be changed and edited using various built-in string techniques, such as character replacement, case conversions to uppercase or lowercase, or substring.

## find() Function in String

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:

The syntax of the find() method is as follows:

```
string.find(substring, start, end)
```

## Parameters:

Where:

- 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 to 0)

- End is the index at which the search should end (optional, defaults to the end of the String)

Return Type:

The find() method returns the index of the first occurrence of that substring in the given range. In case the substring is not found, it returns -1.

Some Examples:

Here's an example of using the find() method to search for a substring within a string:

Example 1:

Program to find a substring in a string:

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

Example 2:

Finding a substring in a string with start and end indices:

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

## String module List –list values, accessing elements

**You access the list items by referring to the index number:**

**Print the second item of the list:**

```
thislist = ["apple", "banana", "cherry"]
print(thislist[1])
```

**Negative Indexing**

**Negative indexing means beginning from the end, -1 refers to the last item, -2 refers to the second last item etc.**

```
thislist = ["apple", "banana", "cherry"]  
print(thislist[-1])
```

**Return the third, fourth, and fifth item:**

```
thislist =  
["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]  
print(thislist[2:5])
```

**Note: The search will start at index 2 (included) and end at index 5 (not included).**

**This example returns the items from the beginning to "orange":**

```
thislist =  
["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]  
print(thislist[:4])
```

**This example returns the items from "cherry" and to the end:**

```
thislist =  
["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]  
print(thislist[2:])
```

## LIST MEMBERSHP

```
str1 = "Hello world"  
list1 = [10, 20, 30, 40, 50]  
# Check 'w' (capital exists in the str1 or not  
if 'w' in str1:  
    print("Yes! w found in ", str1)  
else:  
    print("No! w does not found in ", str1)  
# check 'X' (capital) exists in the str1 or not  
if 'X' not in str1:  
    print("yes! X does not exist in ", str1)  
else:  
    print("No! X exists in ", str1)  
# check 30 exists in the list1 or not  
if 30 in list1:  
    print("Yes! 30 found in ", list1)
```

```
else:
    print("No! 30 does not found in ", list1)
# check 90 exists in the list1 or not
if 90 not in list1:
    print("Yes! 90 does not exist in ", list1)
else:
    print("No! 90 exists in ", list1)
```

## LIST OPERATION

**Some of the most widely used list operations in Python include the following:**

### 1. append()

The append() method adds elements at the end of the list. This method can only add a single element at a time. You can use the append() method inside a loop to add multiple elements.

```
myList = [1, 2, 3, 'EduCBA', 'makes learning fun!']
myList.append(4)
myList.append(5)
myList.append(6)
for i in range(7, 9):
    myList.append(i)
print(myList)
```

### 2. extend()

The extend() method adds more than one element at the end of the list. Although it can add more than one element, unlike append(), it adds them at the end of the list like

```
myList = [1, 2, 3, 'PYTHON', 'makes learning fun!']
myList.extend([4, 5, 6])
for i in range(7, 11):
    myList.append(i)
print(myList)
```

3. insert() method can add an element at a given position in the list. Thus, unlike append(), it can add elements at any position, but like append(), it can add only one element at a time. This method

```
myList = [1, 2, 3, 'PYTHON', 'makes learning fun!']
myList.insert(3, 4)
myList.insert(4, 5)
myList.insert(5, 6)
print(myList)
```

#### 4. remove()

The remove() method removes an element from the list. Only the first occurrence of the same element is removed in the case of multiple occurrences.

```
myList = [1, 2, 3, 'PYTHON', 'makes learning fun!']
myList.remove('makes learning fun!')
print(myList)
```

#### 5. pop()

The method pop() can remove an element from any position in the list. The parameter supplied to this method is the element in

```
myList = [1, 2, 3, 'PYTHON', 'makes learning fun!']
myList.pop(3)
print(myList)
```

6. The min() method returns the minimum value in the list. The max() method returns the maximum value in the list. Both methods accept only homogeneous lists, i.e., lists with similar elements.

```
myList = [1, 2, 3, 4, 5, 6, 7]
print(min(myList))
print(max(myList))
```

7. min() method returns the minimum value in the list. The max() method returns the maximum value in the list. Both methods accept only

```
myList = [1, 2, 3, 4, 5, 6, 7]
print(min(myList))
print(max(myList))
```

## TUPLE ASSIGNMENT AND TUPLE MUTABILITY

What is Tuple in Python?

Tuples are the sequences of values of different data types. It is

Ordered

Immutable

Allows duplicate values.

## Creating tuple in Python

The elements of a tuple are enclosed between circular brackets, (), with each element separated by a comma.

### 1. Creating an empty Python tuple:

To create an empty tuple we just need to assign the square brackets to the variable as shown below. We can use the type() function to check the data type.

#### Creating Python tuple with single element:

To create a tuple with a single element, we need to add a comma after the element. If we don't add the comma, it will be assumed as the data type of that element. The below code shows this.

Example of creating python tuple with a single element:

```
tup2=(4)
print("The type of", tup2,"is:", type(tup2))
tup3=(4,)
print("The type of", tup3,"is:", type(tup3))
```

#### Creating Python tuple with multiple elements:

We can create a tuple with multiple elements by separating the elements by comma and enclosing them between circular parentheses. We don't need to add a comma at the end like in the case of a single element.

It can also contain different values of different data types, including lists, tuples, etc. For example,

Example of creating tuple in Python with multiple element:

```
tup4 = ('a','e','i','o','u')
print("The type of", tup4,"is:", type(tup4))
tup5=(3,9.0,'a',"PythonGeeks",4.5)
print("The type of", tup5,"is:", type(tup5))
tup=(1,4,['a','b'],(4.5,8))
print("The type of", tup,"is:", type(tup))
```

## Accessing elements of tuples in Python

Since tuples are ordered, we can access the elements of a tuple using indexing. Similar to the lists, here the indexing starts from zero. That is, the first element has an index of 0.

### 1. Accessing single element in Python Tuple:

To access an element from a list we can write the tuple name followed by the index in square brackets. We can also give negative indexing, where the first element from the right has the index -1. And this reduces as we go to the left.

### 2. Accessing multiple elements in Python Tuples:

We can again use indexing to access multiple elements. These are the following cases encountered to get more than one element:

- a. To get the elements from index i to j, excluding the jth, we give “i:j” indexing.
- b. To get all the elements from the index i, we give “i:” indexing.
- c. To get all the elements before index i, excluding the ith, we give “:i” indexing.

This method is called slicing. The values of i and j can be positive or negative. For more understanding look at the below example.