Array

- Arrays are fundamental part of most programming languages. It is the collection of elements of a single data type, eg. array of int, array of string.
- An array is a collection of items stored at contiguous memory locations.
- The memory location of the first element of the array (generally denoted by the name of the array).
- Array can be handled in Python by a module named **array**.
- If you create arrays using the **array** module, all elements of the array must be of the same type.
-

Creating a Array

- Array in Python can be created by importing array module. **array(*data_type, value_list*)** is used to create an array with data type and value list specified in its arguments.
-

ar.py - C:/Users/sony/ar.py (3.9.7)

File Edit Format Run Options Window Help

```
# Python program to demonstrate
# Creation of Array

# importing "array" for array creations
import array as arr

# creating an array with integer type
a = arr.array('i', [1, 2, 3])

# printing original array
print ("The new created array is : ", end =" ")
for i in range (0, 3):
        print (a[i], end =" ")
print()

# creating an array with float type
b = arr.array('d', [82.5, 33.2, 53.3])

# printing original array
print ("The new created array is : ", end =" ")
for i in range (0, 3):
        print (b[i], end =" ")
```

IDLE Shell 3.9.7

File Edit Shell Debug Options Window Help

```
Python 3.9.7 (tags/v3.9.7:1016ef3, Aug 30 2021, 20:19:38) [MSC v.1929 64 bit (AM
D64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: C:/Users/sony/ar.py ==========
The new created array is :  1 2 3
The new created array is :  82.5 33.2 53.3
>>>
```

Data types for in creating an array

i - integer

b - boolean

u - unsigned integer

f - float

c - complex float

m - timedelta

M - datetime

O - object

S - string

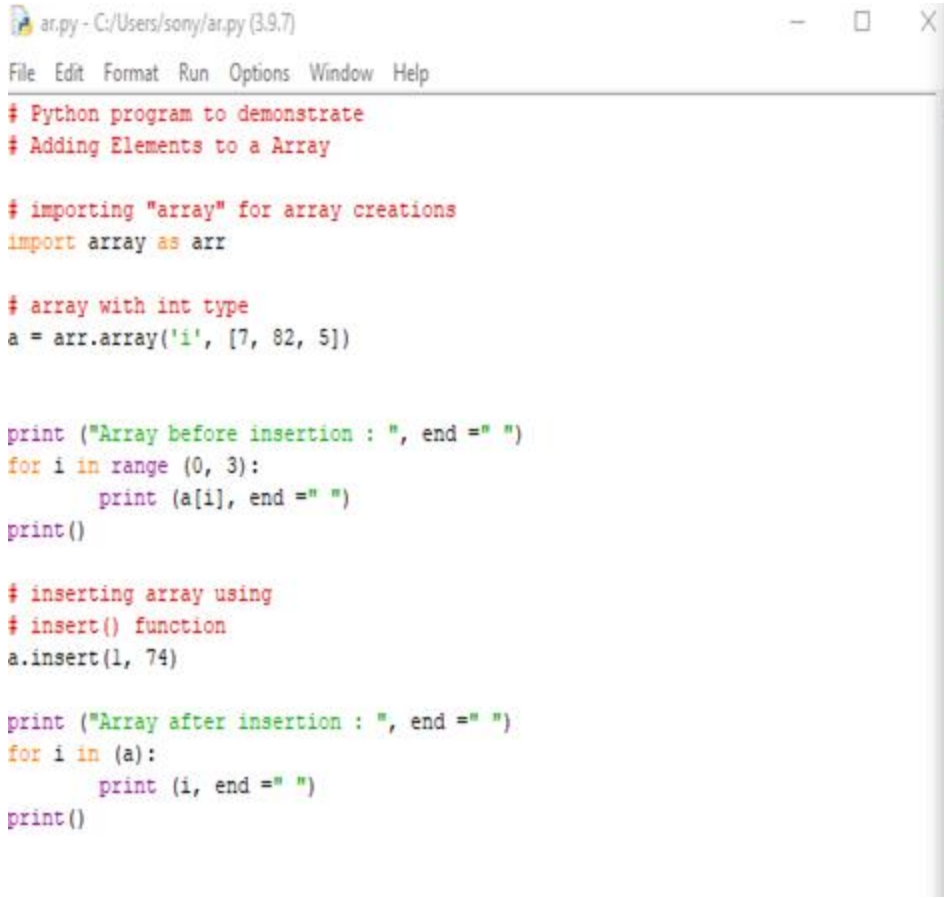
U - unicode string

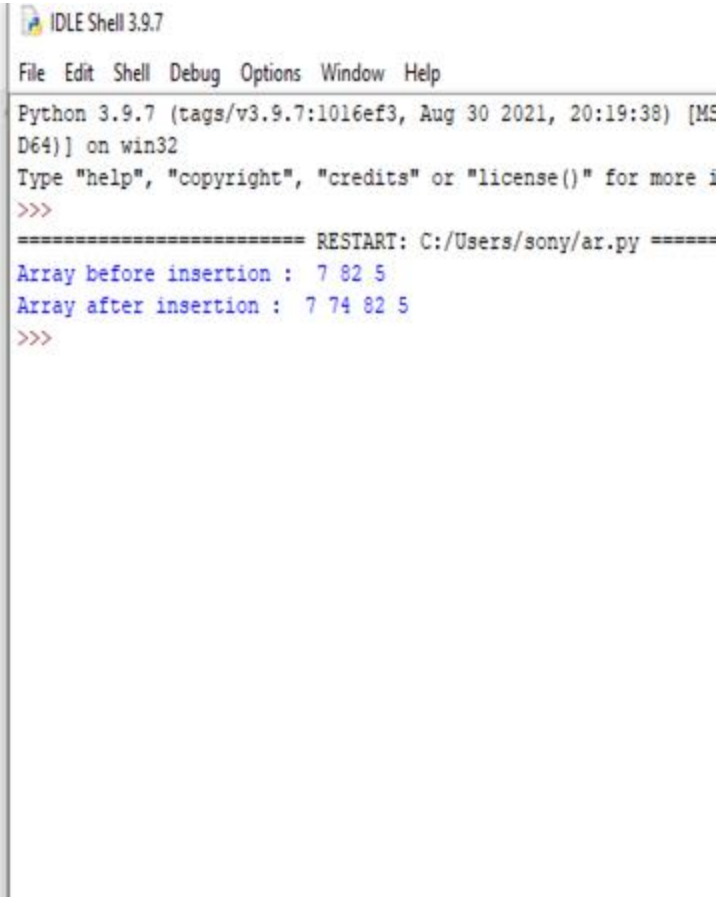
V - fixed chunk of memory for other type (void)

➢

Adding Elements to a Array

- Elements can be added to the Array by using built-in insert() function.
- Insert is used to insert one or more data elements into an array.
- Based on the requirement, a new element can be added at the beginning, end, or any given index of array.
-


ar.py - C:/Users/sony/ar.py (3.9.7)
File Edit Format Run Options Window Help
# Python program to demonstrate
# Adding Elements to a Array
# importing "array" for array creations
import array as arr
# array with int type
a = arr.array('i', [7, 82, 5])
print ("Array before insertion : ", end =" ")
for i in range (0, 3):
print (a[i], end =" ")
print()
# inserting array using
# insert() function
a.insert(1, 74)
print ("Array after insertion : ", end =" ")
for i in (a):
print (i, end =" ")
print()


IDLE Shell 3.9.7
File Edit Shell Debug Options Window Help
Python 3.9.7 (tags/v3.9.7:1016ef3, Aug 30 2021, 20:19:38) [MS
D64)] on win32
Type "help", "copyright", "credits" or "license()" for more i
>>>
========== RESTART: C:/Users/sony/ar.py ======
Array before insertion : 7 82 5
Array after insertion : 7 74 82 5
>>>

Accessing elements from the Array

- In order to access the array items refer to the index number.
- Use the index operator [ ] to access an item in a array.
- The index must be an integer.

ar.py - C:/Users/sony/ar.py (3.9.7)

File Edit Format Run Options Window Help

```
# Python program to demonstrate
# accessing of element from list

# importing array module
import array as arr

# array with int type
a = arr.array('i', [1, 2, 3, 4, 5, 6])

# accessing element of array
print("Access element is: ", a[0])

# accessing element of array
print("Access element is: ", a[3])

# array with float type
b = arr.array('d', [2.5, 3.2, 3.3])

# accessing element of array
print("Access element is: ", b[1])

# accessing element of array
print("Access element is: ", b[2])
```

IDLE Shell 3.9.7

File Edit Shell Debug Options Window Help

```
Python 3.9.7 (tags/v3.9.7:1016ef3, Aug 30 2021, 20:19:38) [MSC v.1929 64 bit (AM
D64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: C:/Users/sony/ar.py ==========
Access element is:  1
Access element is:  4
Access element is:  3.2
Access element is:  3.3
>>>
```

Removing Elements from the Array

- pop() function can also be used to remove
- by default it removes only the last element of the array.
- to remove element from a specific position of the array, index of the element is passed as an argument to the pop() method.

Removing Elements from the Array

ar.py - C:/Users/sony/ar.py (3.9.7)

File Edit Format Run Options Window Help

```
# Python program to demonstrate
# Removal of elements in a Array

# importing "array" for array operations
import array

# initializing array with array values
# initializes array with signed integers
arr = array.array('i', [1, 2, 3, 1, 5])

# printing original array
print ("The new created array is : ", end ="")
for i in range (0, 5):
        print (arr[i], end =" ")

print ("\r")

# using pop() to remove element at 2nd position
print ("The popped element is : ", end ="")
print (arr.pop(2))

# printing array after popping
print ("The array after popping is : ", end ="")
for i in range (0, 4):
        print (arr[i], end =" ")

print("\r")
```

IDLE Shell 3.9.7

File Edit Shell Debug Options Window Help

```
Python 3.9.7 (tags/v3.9.7:1016ef3, Aug 30 2021, 20:19:38) [MSC v.1929 64 bit (A
D64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: C:/Users/sony/ar.py ==========
Access element is:  1
Access element is:  4
Access element is:  3.2
Access element is:  3.3
>>>
========== RESTART: C:/Users/sony/ar.py ==========
The new created array is : 1 2 3 1 5
The popped element is : 3
The array after popping is : 1 2 1 5
>>>
```

Slicing of a Array

- There are multiple ways to print the whole array with all the elements, but to print a specific range of elements from the array, we use Slice operation.
- Slice operation is performed on array with the use of colon(:). To print elements from beginning to a range use [:Index].
- To print elements from end use [:-Index].
- print elements from specific Index till the end use [Index:].
- print elements within a range, use [Start Index:End Index]
- print whole List with the use of slicing operation, use [:].
- print whole array in reverse order, use [::-1].
-

Slicing of a Array

ar.py - C:/Users/sony/ar.py (3.9.7)

File Edit Format Run Options Window Help

```
# importing array module
import array as arr

# creating
l = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

a = arr.array('i', l)
print("Initial Array: ")
for i in (a):
        print(i, end =" ")

# Print elements of a range
# using Slice operation
Sliced_array = a[3:8]
print("\nSlicing elements in a range 3-8: ")
print(Sliced_array)

# Print elements from a
# pre-defined point to end
Sliced_array = a[5:]
print("\nElements sliced from 5th "
        "element till the end: ")
print(Sliced_array)

# Printing elements from
# beginning till end
Sliced_array = a[:]
print("\nPrinting all elements using slice operation: ")
print(Sliced_array)
```

IDLE Shell 3.9.7

File Edit Shell Debug Options Window Help

```
Python 3.9.7 (tags/v3.9.7:1016ef3, Aug 30 2021, 20:19:38) [MSC v.1929 64 bit (AM
D64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: C:/Users/sony/ar.py ==========
Initial Array:
1 2 3 4 5 6 7 8 9 10
Slicing elements in a range 3-8:
array('i', [4, 5, 6, 7, 8])

Elements sliced from 5th element till the end:
array('i', [6, 7, 8, 9, 10])

Printing all elements using slice operation:
array('i', [1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
>>>
```

Searching element in a Array

- In order to search an element in the array we use a python in-built index() method.
- This function returns the index of the first occurrence of value mentioned in arguments.

ar.py - C:/Users/sony/ar.py (3.9.7)

File Edit Format Run Options Window Help

```
# importing array module
import array

# initializing array with array values
# initializes array with signed integers
arr = array.array('i', [1, 2, 3, 1, 2, 5])

# printing original array
print ("The new created array is : ", end ="")
for i in range (0, 6):
        print (arr[i], end =" ")

print ("\r")

# using index() to print index of 1st occurrenece of 2
print ("The index of 1st occurrence of 2 is : ", end ="")
print (arr.index(2))

# using index() to print index of 1st occurrenece of 1
print ("The index of 1st occurrence of 1 is : ", end ="")
print (arr.index(1))
```

IDLE Shell 3.9.7

File Edit Shell Debug Options Window Help

```
Python 3.9.7 (tags/v3.9.7:1016ef3, Aug 30 2021, 20:19:38) [MSC v.1929 64 bit
D64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: C:/Users/sony/ar.py ==========
The new created array is : 1 2 3 1 2 5
The index of 1st occurrence of 2 is : 1
The index of 1st occurrence of 1 is : 0
>>>
```

ar.py - C:/Users/sony/ar.py (3.9.7)

File Edit Format Run Options Window Help

```
# importing array module
import array

# initializing array with array values
# initializes array with signed integers
arr = array.array('i', [1, 2, 3, 1, 2, 5])

# printing original array
print ("Array before updation : ", end ="")
for i in range (0, 6):
        print (arr[i], end =" ")

print ("\r")

# updating a element in a array
arr[2] = 6
print("Array after updation : ", end ="")
for i in range (0, 6):
        print (arr[i], end =" ")
print()

# updating a element in a array
arr[4] = 8
print("Array after updation : ", end ="")
for i in range (0, 6):
        print (arr[i], end =" ")
```

IDLE Shell 3.9.7

File Edit Shell Debug Options Window Help

```
Python 3.9.7 (tags/v3.9.7:1016ef3, Aug 30 2021, 20:19:38) [MSC v.1929 64 bit
D64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: C:/Users/sony/ar.py ==========
Array before updation : 1 2 3 1 2 5
Array after updation : 1 2 6 1 2 5
Array after updation : 1 2 6 1 8 5
>>>
```

Lists

- ist is an ordered sequence which is mutable
- list can have elements of different data types, such as integer, float, string,
- A list is very useful to group together elements of mixed data types.
- Elements of a list are enclosed in square brackets and are separated by comma.

- *Example*

```
 list1 = [2,4,6,8,10,12]
>>> print(list1)
[2, 4, 6, 8, 10, 12]
#list is the list of mixed data types
>>> list2= [100,23.5,'Hello']
>>> print(list2)
[100, 23.5, 'Hello']
```

Lists

- **Accessing Elements in a List** :
- The elements of a list are accessed in the same way as characters are accessed in a string

```
#initializes a list list1
>>> list1 = [2,4,6,8,10,12]
>>> list1[0] #return first element of list1
2
>>> list1[3] #return fourth element of list1
8
#return error as index is out of range
>>> list1[15]
IndexError: list index out of range
```

Lists

- **Accessing Elements in a List** :
- #an expression resulting in an integer index

```
>>> list1[1+4]
12
>>> list1[-1] #return first element from right
12
#length of the list list1 is assigned to n
>>> n = len(list1)
>>> print(n)
6
#return the last element of the list1
>>> list1[n-1]
12
#return the first element of list1
>>> list1[-n]
2
```

Lists

- **Lists are Mutable** :
- In Python, lists are mutable. It means that the contents of the list can be changed after it has been created.
- #List list1 of colors
  >>> list1 = ['Red','Green','Blue','Orange']
  #change/override the fourth element of list1
  >>> list1[3] = 'Black'
  >>> list1 #print the modified list list1
  ['Red', 'Green', 'Blue', 'Black']

Lists

- **LIst operatIons** :

  1. **Concatenation**

  to join two or more lists using concatenation operator depicted by the symbol +.

- #list1 is list of first five odd integers
  >>> list1 = [1,3,5,7,9]
  #list2 is list of first five even integers
  >>> list2 = [2,4,6,8,10]
  #elements of list1 followed by list2
  >>> list1 + list2
  [1, 3, 5, 7, 9, 2, 4, 6, 8, 10]
  The concatenation
  operator '+' requires that the operands should be of list type only. If we try to concatenate a list with elements of some other data type, TypeError occurs.
  >>> list1 = [1,2,3]
  >>> str1 = "abc"
  >>> list1 + str1
  TypeError: can only concatenate list (not
  "str") to list

Lists

- **LIst operatIons** :

  2. **Repetition**

  Replicate a list using repetition operator depicted by symbol *.

- >>> list1 = ['Hello']
  #elements of list1 repeated 4 times
  >>> list1 * 4
  ['Hello', 'Hello', 'Hello', 'Hello']

Lists

- **LIst operatIons** :

  3. **Membership**

  The membership operators **in** checks if the element is present in the list and returns True, else returns False.

- >>> list1 = ['Red','Green','Blue']
  >>> 'Green' in list1
  True
  >>> 'Cyan' in list1
  False
  The **not in** operator returns True if the element is not present in the list, else it returns False.
  >>> list1 = ['Red','Green','Blue']
  >>> 'Cyan' not in list1
  True
  >>> 'Green' not in list1
  False

Lists

- **LIst operatIons** :

  4. **Slicing**

```
>>> list1 =['Red','Green','Blue','Cyan',
'Magenta','Yellow','Black']
>>> list1[2:6]
['Blue', 'Cyan', 'Magenta', 'Yellow']
```

-

```
#list1 is truncated to the end of the list
>>> list1[2:20] #second index is out of range
['Blue', 'Cyan', 'Magenta', 'Yellow',
'Black']
```

Lists

- **List operations** :

  4. **Slicing**

  >>> list1[7:2] #first index > second index

  [ ] #results in an empty list

- #return sublist from index 0 to 4
- >>> list1[:5] #first index missing
- ['Red','Green','Blue','Cyan','Magenta']
-

Lists

- **TraversIng a LIst** :
- ***(A) List Traversal Using for Loop:***
  >>> list1 = ['Red','Green','Blue','Yellow', 'Black']
- >>> for item in list1:
  print(item)

  Output:
  Red
  Green
  Blue
  Yellow
  Black

-

Lists

- **TraversIng a LIst** :
- ***(B) List Traversal Using while Loop:***

```
>>> list1 = ['Red','Green','Blue','Yellow',
'Black']
>>> i = 0
>>> while i < len(list1):
print(list1[i])
i += 1
```

-

Lists

- List Methods and Built-In Functions

Method	Description	Example

len()	Returns the length of the list passed as the argument	>>> list1 = [10,20,30,40,50]
		>>> len(list1)
		5

list()	Creates an empty list if no argument is passed	>>> list1 = list() >>> list1

append()	**Appends a single element passed as an argument at the end of the list
	The single element can also be a list**
		**>>> list1 = [10,20,30,40]
		>>> list1.append(50)
		>>> list1
		[10, 20, 30, 40, 50]
		>>> list1 = [10,20,30,40]
		>>> list1.append([50,60])
		>>> list1
		[10, 20, 30, 40, [50, 60]]**

extend()	Appends each element of the list passed as argument to the end of the given list	>>> list1 = [10,20,30]
		>>> list2 = [40,50]
		>>> list1.extend(list2)
		>>> list1
		[10, 20, 30, 40, 50]

insert()	Inserts an element at a particular index in the list	>>> list1 = [10,20,30,40,50]
		>>> list1.insert(2,25)
		>>> list1
		[10, 20, 25, 30, 40, 50]
		>>> list1.insert(0,5)
		>>> list1
		[5, 10, 20, 25, 30, 40, 50]

count()	**Returns the number of times a given element appears in the list**	**>>> list1 = [10,20,30,10,40,10]** **>>> list1.count(10)** **3** **>>> list1.count(90)** **0**

index()	Returns index of the first occurrence of the element in the list. If the element is not present, ValueError is generated	>>> list1 = [10,20,30,20,40,10]
		>>> list1.index(20)
		1
		>>> list1.index(90)
		ValueError: 90 is not in list

remove()	Removes the given element from the list. If the element is present multiple times, only the first occurrence is removed. If the element is not present, then ValueError is generated	>>> list1 = [10,20,30,40,50,30]
		>>> list1.remove(30)
		>>> list1
		[10, 20, 40, 50, 30]
		>>> list1.remove(90)
		ValueError:list.remove(x):x not in list

reverse()	Reverses the order of elements in the given list	>>> list1 = [34,66,12,89,28,99]
		>>> list1.reverse()
		>>> list1
		[99, 28, 89, 12, 66, 34]
		>>> list1 = ['Tiger' ,'Zebra' ,
		'Lion' , 'Cat' ,'Elephant' ,'Dog']
		>>> list1.reverse()
		>>> list1
		['Dog', 'Elephant', 'Cat',
		'Lion', 'Zebra', 'Tiger']

sort()	Sorts the elements of the given list in-place	>>>list1 = ['Tiger','Zebra','Lion',
		'Cat', 'Elephant' ,'Dog']
		>>> list1.sort()
		>>> list1
		['Cat', 'Dog', 'Elephant', 'Lion',
		'Tiger', 'Zebra']
		>>> list1 = [34,66,12,89,28,99]
		>>> list1.sort(reverse = True)
		>>> list1
		[99,89,66,34,28,12]

sorted()	It takes a list as parameter and creates a
	new list consisting of the same elements
	arranged in sorted order
		>>> list1 = [23,45,11,67,85,56]
		>>> list2 = sorted(list1)
		>>> list1
		[23, 45, 11, 67, 85, 56]
		>>> list2
		[11, 23, 45, 56, 67, 85]

Lists

- **Nested LIsts**
  When a list appears as an element of another list, it is called a nested list.
- >>> list1 = [1,2,'a','c',[6,7,8],4,9]
  #fifth element of list is also a list
  >>> list1[4]
  [6, 7, 8]
  To access the element of the nested list of list1, we have to specify two indices list1[i][j]. The first index I will take us to the desired nested list and second index j will take us to the desired element in that nested list.
- >>> list1[4][1]
  7
  #index i gives the fifth element of list1
  #which is a list
  #index j gives the second element in the
  #nested list

Lists

- **copyIng LIsts**

Given a list, the simplest way to make a copy of the list
is to assign it to another list.

```
>>> list1 = [1,2,3]
>>> list2 = list1
>>> list1
[1, 2, 3]
>>> list2
[1, 2, 3]
```

The statement list2 = list1 does not create a new list. Rather, it just makes list1 and list2 refer to the same list object. Here list2 actually becomes an alias of list1. Therefore, any changes made to either of them will be reflected in the other list.

```
>>> list1.append(10)
>>> list1
[1, 2, 3, 10]
>>> list2
[1, 2, 3, 10]
```

Lists

- **copyIng LIsts**

  We can also create a copy or clone of the list as a distinct object by three methods. The first method uses slicing, the second method uses built-in function list() and the third method uses copy() function of python library copy.

- **Method 1**

  We can slice our original list and store it into a new variable as follows:
  newList = oldList[:]

```
>>> list1 = [1,2,3,4,5]
>>> list2 = list1[:]
>>> list2
[1, 2, 3, 4, 5]
```

Lists

- **copyIng LIsts**

Method 2

We can use the built-in function list() as follows:

newList = list(oldList)

```
>>> list1 = [10,20,30,40]
>>> list2 = list(list1)
>>> list2
[10, 20, 30, 40]
```

Lists

- **copyIng LIsts**

Method 3

We can use the copy () function as follows:

```
import copy #import the library copy
#use copy()function of library copy
newList = copy.copy(oldList)
>>> import copy
>>> list1 = [1,2,3,4,5]
>>> list2 = copy.copy(list1)
>>> list2
[1, 2, 3, 4, 5]
```

Lists

➢ List as argument to a Function

Whenever a list is passed as an argument to a function, we have to consider two scenarios:

A) Elements of the original list may be changed :

For example in the following program list list1 of numbers is passed as an argument to function increment(). This function increases every element of the list by 5.

Lists

- **List as argument to a Function**

```
#Function to increment the elements of the list passed as argument
def increment(list2):
for i in range(0,len(list2)):
#5 is added to individual elements in the list
list2[i] += 5
print('Reference of list Inside Function',id(list2))
#end of function
list1 = [10,20,30,40,50] #Create a list
print("Reference of list in Main",id(list1))
print("The list before the function call")
print(list1)
increment(list1) #list1 is passed as parameter to function
print("The list after the function call")
print(list1)
```

o/p :

- Reference of list in Main 70615968
  The list before the function call
  [10, 20, 30, 40, 50]
  Reference of list Inside Function 70615968 #The id remains same
  The list after the function call
  [15, 25, 35, 45, 55]

Lists

- **List as argument to a Function**

```
#Function to increment the elements of the list passed as argument
def increment(list2):
for i in range(0,len(list2)):
#5 is added to individual elements in the list
list2[i] += 5
print('Reference of list Inside Function',id(list2))
#end of function
list1 = [10,20,30,40,50] #Create a list
print("Reference of list in Main",id(list1))
print("The list before the function call")
print(list1)
increment(list1) #list1 is passed as parameter to function
print("The list after the function call")
print(list1)
```

o/p :

- Reference of list in Main 70615968
  The list before the function call
  [10, 20, 30, 40, 50]
  Reference of list Inside Function 70615968 #The id remains same
  The list after the function call
  [15, 25, 35, 45, 55]

Lists

- **LIst ManIpuLatIon :**
- **1.** Append an element

- myList = [22,4,16,38,13]
  element = int(input("Enter the element to be appended: "))
  myList.append(element)
- print("The element has been appended\n")

-

Lists

- **LIst ManIpuLatIon :**
- **1.** insert an element at desired position

```
myList = [22,4,16,38,13]
element = int(input("Enter the element to be inserted: "))
pos = int(input("Enter the position:"))
myList.insert(pos,element)
print("The element has been inserted\n")
```

-

Lists

- **LIst ManIpuLatIon :**
- **3.** append a list to the given list

  myList=[1,2,3,4,5,6]
- newList = eval(input("Enter the elements separated by commas"))
- myList.extend(list(newList))
- print("The list has been appended\n")
- print(myList)
-

Lists

- **LIst ManIpuLatIon :**
- **4.** modify an existing element

  myList=[1,2,3,4,5,6]
- i = int(input("Enter the position of the element to be modified: "))
  if i < len(myList):
  newElement = int(input("Enter the new element: "))
  oldElement = myList[i]
  myList[i] = newElement
  print("The element",oldElement,"has been modified\n")
  else:
  print("Position of the element is more than the length of list")

Lists

- **LIst ManIpuLatIon :**
- **5.** delete an existing element by position

```
myList=[1,2,3,4,5,6]
```

-
```
i = int(input("Enter the position of the element to be deleted: "))
if i < len(myList):
element = myList.pop(i)
print("The element",element,"has been deleted\n")
else:
print("\nPosition of the element is more than the length of list")
```

Lists

- **LIst ManIpuLatIon :**
- **6.** delete an existing element by value

  myList=[1,2,3,4,5,6]

- element = int(input("\nEnter the element to be deleted: "))
  if element in myList:
  myList.remove(element)
  print("\nThe element",element,"has been deleted\n")
  else:
  print("\nElement",element,"is not present in the list")

Lists

- **LIst ManIpuLatIon :**
- **7.** list in sorted order

  myList=[1,2,3,4,5,6]
- myList.sort()
  print("\nThe list has been sorted")

Lists

- **LIst ManIpuLatIon :**
- **8.** list in reverse sorted order

  myList=[1,2,3,4,5,6]
- myList.sort(reverse = True)
  print("\nThe list has been sorted in reverse order")

Lists

- Q. Write a user-defined function to check if a number is present in the list or not. If the number is present, return the position of the number. Print an appropriate message if the number is not present in the list.

Lists

- ```
  def linearSearch(num,list1):
  for i in range(0,len(list1)):
  if list1[i] == num:       #num is present
  return i                  #return the position
  return None               #num is not present in the list
  #end of function
  ```
- ```
  list1 = [ ]                    #Create an empty list
  print("How many numbers do you want to enter in the list: ")
  maximum = int(input())
  print("Enter a list of numbers: ")
  for i in range(0,maximum):
  n = int(input())
  list1.append(n)          #append numbers to the list
  num = int(input("Enter the number to be searched: "))
  result = linearSearch(num,list1)
  if result is None:
  print("Number",num,"is not present in the list")
  else:
  print("Number",num,"is present at",result + 1, "position")
  ```

Lists

Summary

- Lists are mutable sequences in Python, i.e., we can change the elements of the list.
- Elements of a list are put in square brackets separated by comma.
- A list within a list is called a nested list.
- Operator + concatenates one list to the end of other list.
- Operator * repeats a list by specified number of times.
- Membership operator in tells if an element is present in the list or not and not in does the opposite.
- Slicing is used to extract a part of the list.
- There are many list manipulation functions including: len(), list(), append(), extend(), insert(), count(), find(), remove(), pop(), reverse(), sort(), sorted(), min(), max(), sum().

Tuples and dictionaries

- A tuple is an ordered sequence of elements of different data types, such as integer, float, string.
- Elements of a tuple are enclosed in parenthesis (round brackets) and are separated by commas.
- >>> tuple1 = (1,2,3,4,5)
  >>> tuple1
  (1, 2, 3, 4, 5)
- >>> tuple2 =('Economics',87,'Accountancy',89.6)
  >>> tuple2
  ('Economics', 87, 'Accountancy', 89.6)

Accessing Elements in a Tuple

- Elements of a tuple can be accessed in the same way as a list or string using indexing and slicing.
- >>> tuple1 = (2,4,6,8,10,12)
  #initializes a tuple tuple1
  #returns the first element of tuple1
  >>> tuple1[0]
  2
  #returns fourth element of tuple1
  >>> tuple1[3]
  8
  #returns error as index is out of range
  >>> tuple1[15]
  IndexError: tuple index out of range
  #an expression resulting in an integer index
  >>> tuple1[1+4]
  12
  #returns first element from right
  >>> tuple1[-1]
  12

Accessing Elements in a Tuple

- Elements of a tuple can be accessed in the same way as a list or string using indexing and slicing.
- >>> tuple1 = (2,4,6,8,10,12)
  #initializes a tuple tuple1
  #returns the first element of tuple1
  >>> tuple1[0]
  2
  #returns fourth element of tuple1
  >>> tuple1[3]
  8
  #returns error as index is out of range
  >>> tuple1[15]
  IndexError: tuple index out of range
  #an expression resulting in an integer index
  >>> tuple1[1+4]
  12
  #returns first element from right
  >>> tuple1[-1]
  12

- Tuple is an immutable data type. It means that the elements of a tuple cannot be changed after it has been created.
- >>> tuple1 = (1,2,3,4,5)
  >>> tuple1[4] = 10
  TypeError: 'tuple' object does not support item assignment .
-

Accessing Elements in a Tuple

- Elements of a tuple can be accessed in the same way as a list or string using indexing and slicing.
- >>> tuple1 = (2,4,6,8,10,12)
  #initializes a tuple tuple1
  #returns the first element of tuple1
  >>> tuple1[0]
  2
  #returns fourth element of tuple1
  >>> tuple1[3]
  8
  #returns error as index is out of range
  >>> tuple1[15]
  IndexError: tuple index out of range
  #an expression resulting in an integer index
  >>> tuple1[1+4]
  12
  #returns first element from right
  >>> tuple1[-1]
  12

Accessing Elements in a Tuple

- Elements of a tuple can be accessed in the same way as a list or string using indexing and slicing.
- >>> tuple1 = (2,4,6,8,10,12)
  #initializes a tuple tuple1
  #returns the first element of tuple1
  >>> tuple1[0]
  2
  #returns fourth element of tuple1
  >>> tuple1[3]
  8
  #returns error as index is out of range
  >>> tuple1[15]
  IndexError: tuple index out of range
  #an expression resulting in an integer index
  >>> tuple1[1+4]
  12
  #returns first element from right
  >>> tuple1[-1]
  12

Tuple operations

- **Concatenation** :
- Python allows us to join tuples using concatenation operator depicted by symbol +.
- >>> tuple1 = (1,3,5,7,9)
  >>> tuple2 = (2,4,6,8,10)
  >>> tuple1 + tuple2

- **Repetition** :
- Repetition operation is depicted by the symbol *. It is used to repeat elements of a tuple. We can repeat the tuple elements. The repetition operator requires the first operand to be a tuple and the second operand to be an integer only.
- >>> tuple1 = ('Hello','World')
  >>> tuple1 * 3
  ('Hello', 'World', 'Hello', 'World', 'Hello', 'World')

Tuple operations

- **Membership** :
- The in operator checks if the element is present in the tuple and returns True, else it returns False.

```
>>> tuple1 = ('Red','Green','Blue')
>>> 'Green' in tuple1
True
```

The not in operator returns True if the element is not present in the tuple, else it returns False.

```
>>> tuple1 = ('Red','Green','Blue')
>>> 'Green' not in tuple1
False
```

Tuple operations

- **Slicing** :
- Like string and list, slicing can be applied to tuples also.

```
#tuple1 is a tuple
>>> tuple1 = (10,20,30,40,50,60,70,80)
#elements from index 2 to index 6
>>> tuple1[2:7]
(30, 40, 50, 60, 70)
#slice starts from zero index
>>> tuple1[:5]
(10, 20, 30, 40, 50)
#slice is till end of the tuple
>>> tuple1[2:]
(30, 40, 50, 60, 70, 80)
```

Introduction to dictionaries

- The data type *dictionary* fall under mapping. It is a mapping between a *set of keys* and a *set of values*. The key-value pair is called an *item*. A key is separated from its value by a colon(:) and consecutive items are separated by commas.
- Items in dictionaries are unordered, so we may not get back the data in the same order in which we had entered the data initially in the dictionary.
- **Creating a Dictionary** :
- To create a dictionary, the items entered are separated by commas and enclosed in curly braces. Each item is a key value pair, separated through colon (:).
- The keys in the dictionary must be unique and should be of any immutable data type, i.e., number, string or tuple. The values can be repeated and can be of any data type.

- >>> dict3 = {'Mohan':95,'Ram':89,'Suhel':92, 'Sangeeta':85}
  >>> dict3
  {'Mohan': 95, 'Ram': 89, 'Suhel': 92, 'Sangeeta': 85}

Accessing Items in a Dictionary

- We have already seen that the items of a sequence (string, list and tuple) are accessed using a technique called indexing. The items of a dictionary are accessed via the keys rather than via their relative positions or indices.
- >>> dict3 = {'Mohan':95,'Ram':89,'Suhel':92, 'Sangeeta':85}
  >>> dict3['Ram']
  89
  >>> dict3['Sangeeta']
  85
  #the key does not exist
  >>> dict3['Shyam']
  KeyError: 'Shyam'

Dictionaries are mutable

- Dictionaries are mutable which implies that the contents of the dictionary can be changed after it has been created.

- **Adding a new item** :
- >>> dict1 = {'Mohan':95,'Ram':89,'Suhel':92, 'Sangeeta':85}
- >>> dict1['Meena'] = 78
  >>> dict1
  {'Mohan': 95, 'Ram': 89, 'Suhel': 92, 'Sangeeta': 85, 'Meena': 78}
- **Modifying an Existing Item** :
- The existing dictionary can be modified by just overwriting the key-value pair.
- >>> dict1 = {'Mohan':95,'Ram':89,'Suhel':92,
  'Sangeeta':85}
  #Marks of Suhel changed to 93.5
  >>> dict1['Suhel'] = 93.5
  >>> dict1
  {'Mohan': 95, 'Ram': 89, 'Suhel': 93.5,
  'Sangeeta': 85}

Dictionary operations

- **Membership** :
- The membership operator in checks if the key is present in the dictionary and returns True, else it returns False.

```
>>> dict1 = {'Mohan':95,'Ram':89,'Suhel':92,
'Sangeeta':85}
>>> 'Suhel' in dict1
True
```

The not in operator returns True if the key is not present in the dictionary, else it returns False.

```
>>> dict1 = {'Mohan':95,'Ram':89,'Suhel':92,
'Sangeeta':85}
>>> 'Suhel' not in dict1
False
```

Traversing a dictionary

- We can access each item of the dictionary or traverse a dictionary using for loop.
  >>> dict1 ={'Mohan':95,'Ram':89,'Suhel':92, 'Sangeeta':85}
- >>> for key in dict1:
  print(key,':',dict1[key])

  Mohan: 95
  Ram: 89
  Suhel: 92
  Sangeeta: 85