ARRAYS IN PYTHON

- An array is an object that stores a group of elements of same datatype.
- Array can store only one type of data.
- Arrays can increase or decrease their size dynamically.

Advantages of Arrays

- Arrays are similar to lists. The main difference is that arrays can store only one datatype whereas lists can store different types of elements.
- The size of the array is not fixed.
- Arrays can grow or shrink in memory dynamically.
- Methods that are useful to process the elements of array are available in **'array' module.**

Creating an Array

Syntax: arrayname = array (type code, [elements])

Type Code	C Type	Minimum size in bytes
'b'	Signed integer	1
'B'	Unsigned integer	1
'i'	Signed integer	2
'I'	Unsigned integer	2
'l'	Signed integer	4
'L'	Unsigned integer	4
'f'	Floating Point	4
'd'	Double precision floating point	8
'u'	Unicode character	2

Example:

```
a=array('i',[14, 4, 6, 9])

a1=array('d',[5.3,5.2,6.9])
```

Importing the Array Module

```
#First Way
import array
a=array.array('i',[4,9,12,18])
```

```
#Second Way
from array import *
a=array('i',[6,7,8,19])
```

Example:

```
import array as arr
a=arr.array('d',[5.6,1.2,7.5])
print(a)

#To create an integer type array
import array
a=array.array('i',[8,16,45,-8,63])
print('The array elements are:')
for element in a:
   print(element)
```

- It is also possible to create an array from another array.

Example:

```
a1=array('d',[2.5,9.6,8.5,2.2])
a2=array(a1.typecode,(a for a in a1)) #a2 have same type code and same elements
a3=array(a1.typecode,(a*2 for a in a1))

#Create one array from another array
from array import *
a1=array('i',[4,2,5,3])

a2=array(a1.typecode,(a*2 for a in a1))
print('The elements of a2 are:')
for i in a2:
   print(i)
```

Accessing Python Array Elements

- We use indices to access elements of an array.

Example:

```
import array as arr
a=arr.array('i',[2,4,6,8])
print("First element:",a[0])
print("Second element:",a[1])
print("Last element:",a[-1])
```

Indexing and Slicing on Arrays

Example:

```
import array as arr
number_list=[2,3,4,5,6,7,8,9 ,10]
number_array=arr.array('i',number_list)
print(number_array[2:5]) #3rd to 5th
print(number_array[:-5]) #beginning to 4th
print(number_array[5:]) #6th to end
print(number_array[:]) #beginning to end
```

- To find out the numbers of elements in an array we can use **len() function**.
- **Syntax of Slicing: arrayname[start:stop:stride]**
- Stride represents step size excluding the starting element.

Example:

```
# To retrieve the elements of an array
from array import *
x=array('i',[10,20,30,40,50])
n=len(x)

for i in range(n):
   print(x[i],end=' ')
```

Changing and Adding Elements

- Array are mutable; their elements can be changed in a similar way as lists.

Example:

```
import array as arr
numbers=arr.array('i',[1,2,3,5,7,10])

#chaging the first element
numbers[0]=0
print(numbers)

#changing 3rd to 5th element
numbers[2:5]=arr.array('i',[4,6,8])
print(numbers)
```

Processing the Arrays

- Methods are generally called as: **objectname.method()**
- We can add one item to the array using **append () method,** or add several items the **extend () method**.

Example:

```
import array as arr
numbers=arr.array('i',[1,2,3])
print(numbers)
numbers.append(4)
print(numbers)
numbers.extend([5,6,7]) #extend() appends iterable to the end of the array
print(numbers)
```

Example:

```
#Operations on Arrays

from array import *
arr=array('i',[10,20,30,40,50])
print('Original Array : ', arr)

#Use of append
arr.append(30)
arr.append(60)
print('After appending 30 and 60 : ', arr)

#insert 90 at position number 1 in arr
arr.insert(1,90)
print('After inserting 90 in 1st position : ', arr)

#remove an element from arr
arr.remove(20)
print('After removing 20 :',arr)

#remove the last element using pop() method
n=arr.pop()
print('Array after using pop() :',arr)
print('Popped Element is : ',n)

#finding position of element using index() method
n=arr.index(30)
```

```
print('First occurrence of element 30 is at :',n)

#Covert an array into a list using tolist() method
lst=arr.tolist()
print("List : ", lst)
print("Array : ", arr)

arr.remove(30)
print('After removing 30 :',arr)

arr.remove(30)
print('After removing 30 :',arr)
```

- We can also concatenate two arrays using **+ operator.**

Example:

```
import array as arr
odd=arr.array('i',[1,3,5])
print(odd)
even=arr.array('i',[2,4,6])
print(even)

numbers=arr.array('i') #creating empty array of integers
numbers=odd+even
print(numbers)
```

Removing Python Array Elements

- We can delete one or more items from an array using python's **del statement.**
- We can use the **remove() method** to remove the given item, and **pop() method** to remove an item at the given index.

Example 1:

```
import array as arr
number=arr.array('i',[1,2,3,4,5])

del number[2] #removing third element
print(number)

del number #deleting entire array
print(number) #Error array is not defined
```

Example 2:

```
import array as arr
number=arr.array('i',[10,11,12,13,14,15])

number.remove(12)
print(number)

print(number.pop(2))
print(number)
```

Python Lists Vs Arrays

- In python, we can treat lists as arrays. Hence, we cannot constrain the type of elements stored in a list.

```
#elements of different types
a=[1,3.5,"Hello"]
```

- If you create arrays using the **array module,** all elements of the array must be of the same numeric type.

```
import array as arr
#Error
a=arr.array('d',[1,3.5,"Hello"])
```

When to use Arrays?

- Just a thin wrapper on C arrays which provides space-efficient storage of basic C-style data types.
- Arrays can be faster and use less memory than lists.

Types of Arrays

1. Single Dimensional Arrays (1-D Array)
2. Multi-Dimensional Arrays

```
marks = array([[50,60,70,80,90],
               [55,65,75,85,95],
               [42,69,45,85,79]])
```

NOTE: Python does not support multi-dimensional arrays. We can construct multidimensional arrays using third party package like numpy (numerical Python).