Arrays in Python

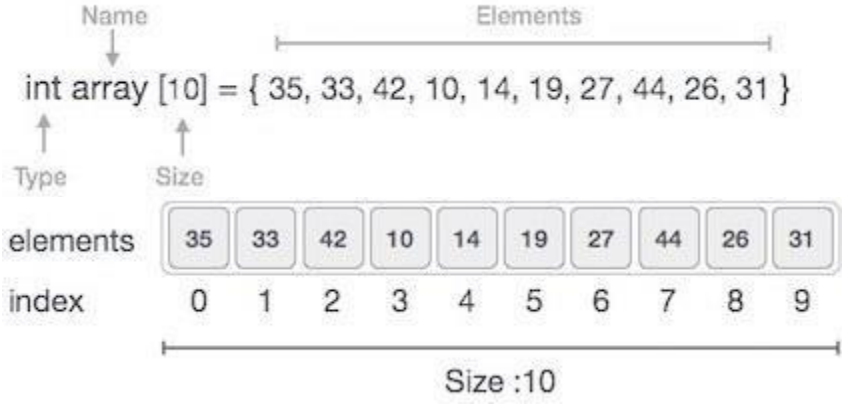
Array is a container which can hold a fix number of items and these items should be of the same type.

Most of the data structures make use of arrays to implement their algorithms. Following are the important terms to understand the concept of Array are as follows –

- **Element** − Each item stored in an array is called an element.
- **Index** − Each location of an element in an array has a numerical index, which is used to identify the element.

Array Representation

Arrays can be declared in various ways in different languages. Below is an illustration.



As per the above illustration, following are the important points to be considered −

- Index starts with 0.
- Array length is 10, which means it can store 10 elements.

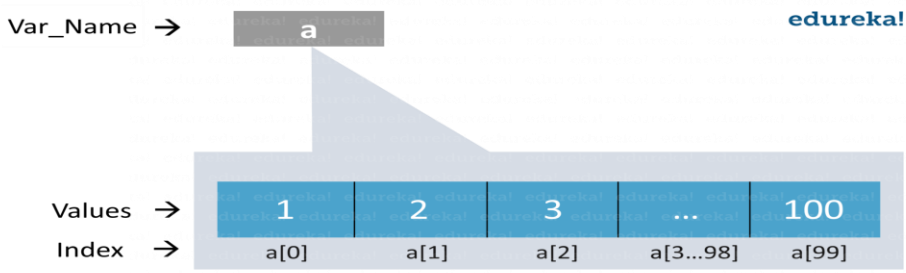
- Each element can be accessed via its index. For example, we can fetch an element at index 6 as 9.

Content

Why use Arrays in Python?

✓ A combination of Arrays, together with Python could ***save you a lot of time***.

✓ arrays help you **reduce the overall size** of your code, while Python helps you get rid of problematic syntax, unlike other languages. ***For example*****:** If you had to store integers from 1-100, you won't be able to remember 100 variable names explicitly, therefore, you can save them easily using an array.



What is an Array in Python?

An array is basically a *data structure* which can hold more than one value at a time. It is a collection or ordered series of elements of the same type.

Is Python list same as an Array?

Python Arrays and lists store values in a similar way. But there is a key difference between the two i.e. the values that they store.

- ✓ ***A list can store any type of values such as integers, strings, etc.***
- ✓ ***An arrays, on the other hand, stores single data type values.***
- ✓ ***Therefore, you can have an array of integers, an array of strings, etc.***

Creating an Array in Python:

Arrays in Python can be created after importing the array module as follows:

→ import array as arr

- ✓ The array(data type, value list) function takes two parameters, the first being the data type of the value to be stored and the second is the value list.
- ✓ The data type can be anything such as int, float, double, etc.
- ✓ Please make a note that arr is the alias name and is for ease of use.

✓ You can import without alias as well. There is another way to import the array module which is –

→ ***from array import ****

This means you want to import all functions from the array module.

The following syntax is used to create an array.

Syntax:

```
1. a=arr.array(data type, value list)          #when you import using arr alias
```

OR

```
1  a=array(data type, value list)              #when you import using *
```

Example: a=arr.array(‘d’ , [1.1 , 2.1 ,3.1])

Here, the first parameter is ‘d’ which is a data type i.e. float and the values are specified as the next parameter.

Note:

All values specified are of the type float. We cannot specify the values of different data types to a single array.

The following table shows you the various data types and their codes.

Typecode	Value

b	Represents signed integer of size 1 byte
B	Represents unsigned integer of size 1 byte
c	Represents character of size 1 byte
i	Represents signed integer of size 2 bytes
I	Represents unsigned integer of size 2 bytes
f	Represents floating point of size 4 bytes
d	Represents floating point of size 8 bytes

Type code	Python Data Type	Byte size
i	int	2
I	int	2
u	unicode character	2
h	int	2
H	int	2
l	int	4
L	int	4
f	float	4

d	float	8

Accessing array elements in Python:

To access array elements, you need to specify the index values. Indexing starts at 0 and not from 1. Hence, the index number is always 1 less than the length of the array.

Syntax:

Array_name[index value]

Example:

```
1    a=arr. array(‘d', [1.1 , 2.1 ,3.1] )
2    a[1]
```

Output –

2.1

The output returned is the value, present at the second place in our array which is 2.1.

Example

The below code creates an array named **array1**.

```
from array import *

array1 = array('i', [10,20,30,40,50])

for x in array1:
  print(x)
```

Output

When we compile and execute the above program, it produces the following result –

```
10
20
30
40
50
```

Accessing Array Element

We can access each element of an array using the index of the element. The below code shows how to access an array element.

Example

```
from array import *

array1 = array('i', [10,20,30,40,50])

print (array1[0])

print (array1[2])
```

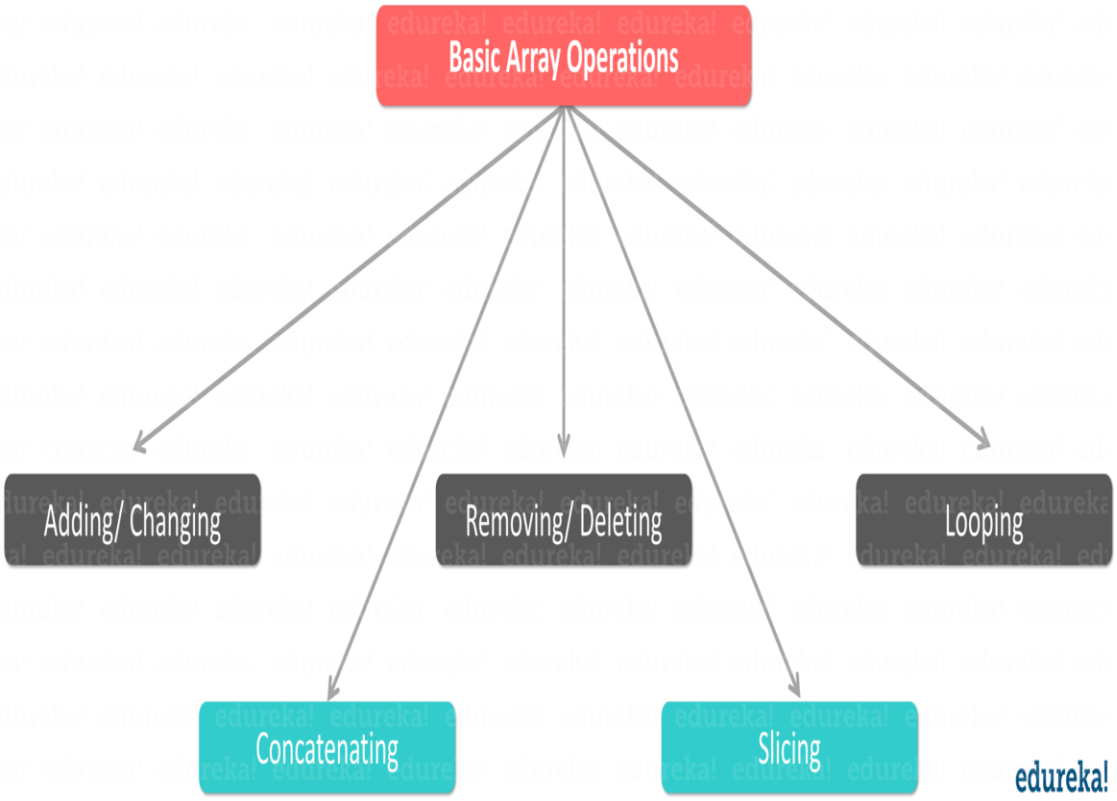
Output

When we compile and execute the above program, it produces the following result, which shows the element is inserted at index position 1.

```
10
30
```

Basic array operations:

There are many operations that can be performed on arrays which are as follows –



Basic Operations

The basic operations supported by an array are as stated below −

- **Traverse** − print all the array elements one by one.
- **Insertion** − Adds an element at the given index.
- **Deletion** − Deletes an element at the given index.
- **Search** − Searches an element using the given index or by the value.
- **Update** − Updates an element at the given index.

Insertion Operation

Insert operation is 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.

Example

Here, we add a data element at the middle of the array using the python in-built insert() method.

```
from array import *

array1 = array('i', [10,20,30,40,50])

array1.insert(1,60)

for x in array1:
   print(x)
```

When we compile and execute the above program, it produces the following result which shows the element is inserted at index position 1.

Output

```
10
60
20
30
40
50
```

Deletion Operation

Deletion refers to removing an existing element from the array and re-organizing all elements of an array.

Example

Here, we remove a data element at the middle of the array using the python in-built remove() method.

```
from array import *

array1 = array('i', [10,20,30,40,50])

array1.remove(40)

for x in array1:
   print(x)
```

Output

When we compile and execute the above program, it produces the following result which shows the element is removed form the array.

```
10
20
30
50
```

Search Operation

You can perform a search for an array element based on its value or its index.

Example

Here, we search a data element using the python in-built index() method.

```
from array import *

array1 = array('i', [10,20,30,40,50])

print (array1.index(40))
```

Output

When we compile and execute the above program, it produces the following result which shows the index of the element. If the value is not present in the array then th eprogram returns an error.

```
3
```

Update Operation

Update operation refers to updating an existing element from the array at a given index.

Example

Here, we simply reassign a new value to the desired index we want to update.

```
from array import *

array1 = array('i', [10,20,30,40,50])

array1[2] = 80

for x in array1:
   print(x)
```

Output

When we compile and execute the above program, it produces the following result which shows the new value at the index position 2.

```
10
20
80
40
50
```

Finding the Length of an Array

Length of an array is the number of elements that are actually present in an array. You can make use of **len()** function to achieve this. The len() function returns an integer value that is equal to the number of elements present in that array.

Syntax:

→ len(array_name)

Example:

```
1    a=arr.array('d', [1.1 , 2.1 ,3.1] )
2    len(a)
```

Output – 3

This returns a value of 3 which is equal to the number of array elements.

Adding/ Changing elements of an Array:

We can add value to an array by using the **append ()**, **extend ()** and the **insert (i,x)** functions.

The append() function is used when we need to add a single element at the end of the array.

Example:

```
1    a=arr.array('d', [1.1 , 2.1 ,3.1] )
2    a.append(3.4)
3    print(a)
```

Output –

array(‘d’, [1.1, 2.1, 3.1, 3.4])

The resultant array is the actual array with the new value added at the end of it. To add more than one element, you can use the extend() function. This function takes a list of elements as its parameter. The contents of this list are the elements to be added to the array.

Example:

```
1    a=arr.array('d', [1.1 , 2.1 ,3.1] )
2    a.extend([4.5,6.3,6.8])
3    print(a)
```

Output –

array(‘d’, [1.1, 2.1, 3.1, 4.5, 6.3, 6.8])

The resulting array will contain all the 3 new elements added to the end of the array.

However, when you need to add a specific element at a particular position in the array, the insert(i,x) function can be used. This function inserts the element at the respective index in the array. It takes 2 parameters where the first parameter is the index where the element needs to be inserted and the second is the value.

Example:

```
1    a=arr.array('d', [1.1 , 2.1 ,3.1] )
2    a.insert(2,3.8)
3    print(a)
```

Output –

array(‘d’, [1.1, 2.1, 3.8, 3.1])

The resulting array contains the value 3.8 at the 3rd position in the array.

Arrays can be merged as well by performing array concatenation.

Array Concatenation:

Any two arrays can be concatenated using the + symbol.

Example:

```
1  a=arr.array('d',[1.1 , 2.1 ,3.1,2.6,7.8])
2  b=arr.array('d',[3.7,8.6])
3  c=arr.array('d')
4  c=a+b
5  print("Array c = ",c)
```

Output –

Array c= array(‘d’, [1.1, 2.1, 3.1, 2.6, 7.8, 3.7, 8.6])

The resulting array c contains concatenated elements of arrays a and b.

Now, let us see how you can remove or delete items from an array.

Removing/ Deleting elements of an array:

Array elements can be removed using **pop()** or **remove()** method. The difference between these two functions is that the former returns the deleted value whereas the latter does not.

The pop() function takes either no parameter or the index value as its parameter. When no parameter is given, this function pops() the last element and returns it. When you explicitly supply the index value, the pop() function pops the required elements and returns it.

Example:

```
1  a=arr.array('d', [1.1, 2.2, 3.8, 3.1, 3.7, 1.2, 4.6])
2  print(a.pop())
3  print(a.pop(3))
```

Output –

4.6

3.1

The first pop() function removes the last value 4.6 and returns the same while the second one pops the value at the 4th position which is 3.1 and returns the same.

The remove() function, on the other hand, is used to remove the value where we do not need the removed value to be returned. This function takes the element value itself as the parameter. If you give the index value in the parameter slot, it will throw an error.

Example:

```
1  a=arr.array('d',[1.1 , 2.1 ,3.1])
2  a.remove(1.1)
3  print(a)
```

Output –

array(‘d’, [2.1,3.1])

The output is an array containing all elements except 1.1.

When you want a specific range of values from an array, you can slice the array to return the same, as follows.

Looping through an array:

Using the for loop, we can loop through an array.

Example:

```
1  a=arr.array('d', [1.1, 2.2, 3.8, 3.1, 3.7, 1.2, 4.6])
2  print("All values")
3  for x in a:
4  print(x)
5  print("specific values")
6  for x in a[1:3]:
7  print(x)
```

Output –

All values

1.1

2.2

3.8

3.1

3.7

1.2

4.6

specific values

2.2

3.8

The above output shows the result using for loop. When we use for loop without any specific parameters, the result contains all the elements of the array given one at a time. In the second for loop, the result contains only the elements that are specified using the index values. Please note that the result does not contain the value at index number 3.