**Python - Arrays**

Arrays in Python

Unlike other programming languages like C++ or Java, Python does not have built-in support for arrays. However, Python has several data types like lists and tuples (especially lists) that are often used as arrays but, items stored in these types of sequences need not be of the same type.

In addition, we can create and manipulate arrays the using the **array** module. Before proceeding further, let's understand arrays in general.

AD

What are arrays?

An **array** is a container which can hold a fix number of items and these items should be of the same type. Each item stored in an array is called an **element** and they can be of any type including integers, floats, strings, etc.

These elements are stored at contiguous memory location. Each location of an element in an array has a numerical **index** starting from 0. These indices are used to identify and access the elements.

Learn Python in-depth with real-world projects through our [Python certification course](https://www.tutorialspoint.com/certification/complete-python-prime-pack/index.asp?utm_source=tutorialspoint&utm_medium=python_tutorial_3p&utm_campaign=internal). Enroll and become a certified expert to boost your career.

AD

Array Representation

Arrays are represented as a collection of multiple containers where each container stores one element. These containers are indexed from '0' to 'n-1', where n is the size of that particular array.

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.

AD

Creating Array in Python

To create an array in Python, import the array module and use its **array() function**. We can create an array of three basic types namely integer, float and Unicode characters using this function.

The array() function accepts typecode and initializer as a parameter value and returns an object of array class.

Syntax

The syntax for creating an array in Python is −

# importing

import array as array\_name

# creating array

obj = array\_name.array(typecode[, initializer])

Where,

* **typecode** − The typecode character used to speccify the type of elements in the array.
* **initializer** − It is an optional value from which array is initialized. It must be a list, a bytes-like object, or iterable elements of the appropriate type.

Example

The following example shows how to create an array in Python using the array module.

Open Compiler

import array as arr

# creating an array with integer type

a = arr.array('i', [1, 2, 3])

print (type(a), a)

# creating an array with char type

a = arr.array('u', 'BAT')

print (type(a), a)

# creating an array with float type

a = arr.array('d', [1.1, 2.2, 3.3])

print (type(a), a)

It will produce the following **output** −

<class 'array.array'> array('i', [1, 2, 3])

<class 'array.array'> array('u', 'BAT')

<class 'array.array'> array('d', [1.1, 2.2, 3.3])

Python array type is decided by a single character Typecode argument. The type codes and the intended data type of array is listed below −

|  |  |  |
| --- | --- | --- |
| **typecode** | **Python data type** | **Byte size** |
| 'b' | signed integer | 1 |
| 'B' | unsigned integer | 1 |
| 'u' | Unicode character | 2 |
| 'h' | signed integer | 2 |
| 'H' | unsigned integer | 2 |
| 'i' | signed integer | 2 |
| 'I' | unsigned integer | 2 |
| 'l' | signed integer | 4 |
| 'L' | unsigned integer | 4 |
| 'q' | signed integer | 8 |
| 'Q' | unsigned integer | 8 |
| 'f' | floating point | 4 |
| 'd' | floating point | 8 |

AD

Basic Operations on Python Arrays

Following are the basic operations supported by an array −

* **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.

AD

Accessing Array Element

We can access each element of an array using the index of the element.

Example

The below code shows how to access elements of an array.

Open Compiler

from array import \*

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

print (array1[0])

print (array1[2])

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

10

30

AD

Insertion Operation

In insertion operation, we 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.

Open Compiler

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.

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.

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

Open Compiler

from array import \*

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

array1.remove(40)

for x in array1:

print(x)

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 operation on an array to find an array element based on its value or its index.

Example

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

Open Compiler

from array import \*

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

print (array1.index(40))

When we compile and execute the above program, it will display the index of the searched element. If the value is not present in the array, it will return an error.

3

Update Operation

Update operation refers to updating an existing element from the array at a given index. Here, we simply reassign a new value to the desired index we want to update.

Example

In this example, we are updating the value of array element at index 2.

Open Compiler

from array import \*

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

array1[2] = 80

for x in array1:

print(x)

On executing the above program, it produces the following result which shows the new value at the index position 2.

10

20

80

40