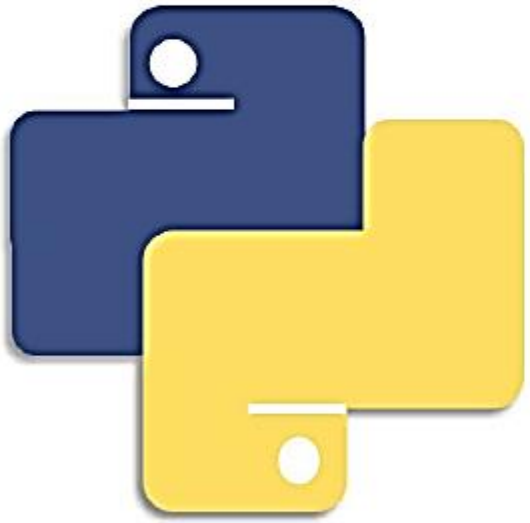
Python

By

Dr. rabih ali sbera

Department of Computer Engineering Techniques-2 nd

2024-2025



Python Programming

Arrays

Arrays

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.

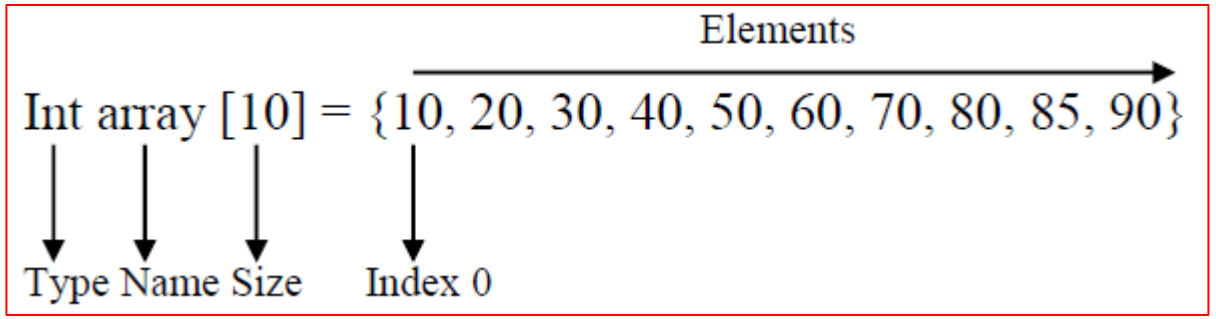
Following are the important terms to understand the concept of Array.

Element− Each item stored in an array is called an element.

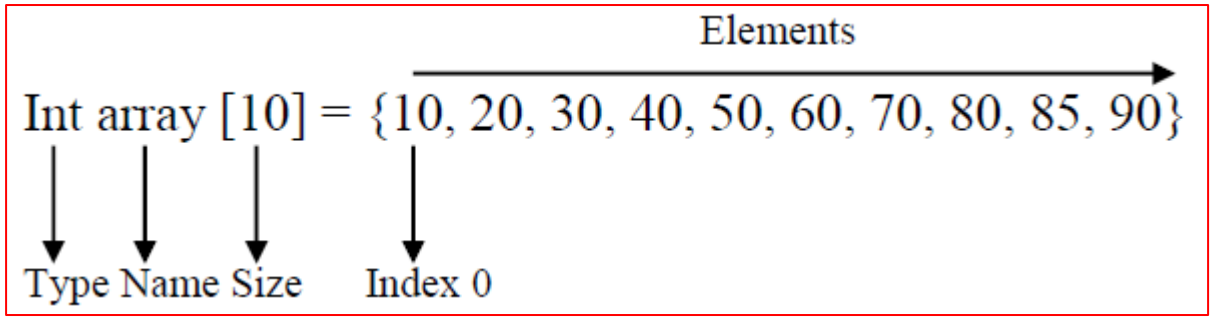
Index − Each location of an element in an array has a numerical index, which is used to identify the element.

Array Representation

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



Arrays



- 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 70

Arrays

- **Basic Operations**

➢ 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.

➢ Array is created in Python by importing array module to the python program. Then the

➢ array is declared as shown below.

```
from array import *
arrayName=array(typecode, [initializers])
```

Arrays

Type code are the codes that are used to define the type of value the array will hold. Some common type codes used are:

Typecode	Value
b	Represents signed integer of size 1 byte/td>
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

Arrays

Creating an array

```
from array import *
array1 = array('i', [10,20,30,40,50])
for x in array1:
 print(x)
```

```
10
20
30
40
50
```

Accessing Array Element

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

```
from array import *
array1 = array('i', [10,20,30,40,50])
print (array1[0])
print (array1[2])
```

```
10
30
```

Array methods:

Python has a set of built-in methods that you can use on lists/arrays

Method	Description
append()	Adds an element at the end of the list
clear()	Removes all the elements from the list
copy()	Returns a copy of the list
count()	Returns the number of elements with the specified value
extend()	Add the elements of a list (or any iterable), to the end of the current list
index()	Returns the index of the first element with the specified value
insert()	Adds an element at the specified position
pop()	Removes the element at the specified position
remove()	Removes the first item with the specified value
reverse()	Reverses the order of the list
sort()	Sorts the list

Arrays

```
college=["mrcet","it","cse"]
college.append("autonomous")
college
```

```
['mrcet', 'it', 'cse', 'autonomous']
```

```
college.pop()
college
```

```
['mrcet', 'it', 'cse']
```

```
college.pop(2)
college
```

```
['mrcet', 'it']
```

```
college.remove("it")
college
```

```
['mrcet']
```

```
college=["mrcet","it","cse"]
college.reverse()
college
```

```
['cse', 'it', 'mrcet']
```

```
new=[5,4,2]
new.sort()
new
```

```
[2, 4, 5]
```

```
new=[5,4,2,4,4,4,4,3,2]
new.count(4)
```

```
5
```

```
new=[5,4,2,4,4,4,4,3,2]
b=[4,8]
new.extend(b)
new
```

```
[5, 4, 2, 4, 4, 4, 4, 3, 2, 4, 8]
```

```
new=[5,4,2,4,4,4,4,3,2]
b=["R","s"]
new.extend(b)
new
```

```
[5, 4, 2, 4, 4, 4, 4, 3, 2, 'R', 's']
```

```
new=[5,4,2,4,4,4,4,3,2]
new.index(4)
```

```
1
```

```
new=[5,4,2,4,4,4,4,3,2]
new.index(2)
```

```
2
```

```
new=[5,4,2,4,4,4,4,3,2]
b=["R","s"]
new.extend(b)
new.clear()
new
```

```
[]
```

List

It is a general purpose most widely used in data structures

List is a collection which is ordered and changeable and allows duplicate members.

(Grow and shrink as needed, sequence type, sortable).

To use a list, you must declare it first. Do this using square brackets and separate values with commas.

```
list1=[1,2,3,'A','B',7,8,[10,11]]
print(list1)
```

```
[1, 2, 3, 'A', 'B', 7, 8, [10, 11]]
```

```
x=list()
x
```

```
[]
```

```
tuple1=(1,2,3,4)
x=list(tuple1)
x
```

```
[1, 2, 3, 4]
```

List

- **List operations:**
- These operations include indexing, slicing, adding, multiplying, and checking for membership
- Basic List Operations:
- Lists respond to the + and * operators much like strings; they mean concatenation and
- repetition here too, except that the result is a new list, not a string.

Python Expression	Results	Description
len([1, 2, 3])	3	Length
[1, 2, 3] + [4, 5, 6]	[1, 2, 3, 4, 5, 6]	Concatenation
['Hi!'] * 4	['Hi!', 'Hi!', 'Hi!', 'Hi!']	Repetition
3 in [1, 2, 3]	True	Membership
for x in [1, 2, 3]: print x,	1 2 3	Iteration

List

- L = ['mrcet', 'college', 'MRCET!']

Python Expression	Results	Description
L[2]	MRCET	Offsets start at zero
L[-2]	college	Negative: count from the right
L[1:]	['college', 'MRCET!']	Slicing fetches sections

```
list1=[1,2,3,4,5,6,7,8,9,10]
list1[1:]
```

```
[2, 3, 4, 5, 6, 7, 8, 9, 10]
```

```
list1[:1]
```

```
[1]
```

```
list1[2:5]
```

```
[3, 4, 5]
```

```
list1[:6]
```

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

```
list1[1:2:4]
```

```
[2]
```

```
list1[1:8:2]
```

```
[2, 4, 6, 8]
```

List

```
list1=[4,2,7,4,5,6,9,8,11,10]
```

```
list1[1:8:3]
```

```
[2, 5, 8]
```

```
list1[2:8:3]
```

```
[7, 6]
```

```
list1[1:2:1]
```

```
[2]
```

```
list1[1:3:4]
```

```
[2]
```

```
list1[1:9:2]
```

```
[2, 4, 6, 8]
```

```
list1[2:9:2]
```

```
[7, 5, 9, 11]
```

List

List methods:

- The list data type has some more methods. Here are all of the methods of list objects:

❖ Del()

❖ Append()

❖ Extend()

❖ Insert()

❖ Pop()

❖ Remove()

❖ Reverse()

❖ Sort()

❖ **Delete:** Delete a list or an item from a list

List

```
>>> x=[5,3,8,6]
>>> del(x[1])          #deletes the index position 1 in a list
>>> x
[5, 8, 6]
-----------
>>> del(x)
>>> x                  # complete list gets deleted
```

Append: Append an item to a list

```
>>> x=[1,5,8,4]
>>> x.append(10)
>>> x
[1, 5, 8, 4, 10]
```

List

Extend: Append a sequence to a list.

```
>>> x=[1,2,3,4]
>>> y=[3,6,9,1]
>>> x.extend(y)
>>> x
[1, 2, 3, 4, 3, 6, 9, 1]
```

Insert: To add an item at the specified index, use the insert () method:

```
>>> x=[1,2,4,6,7]
>>> x.insert(2,10)   #insert(index no, item to be inserted)
>>> x
[1, 2, 10, 4, 6, 7]
```

List

```
>>> x.insert(4,['a',11])

>>> x

[1, 2, 10, 4, ['a', 11], 6, 7]
```

Pop: The pop() method removes the specified index, (or the last item if index is not specified) or simply pops the last item of list and returns the item.

```
>>> x=[1, 2, 10, 4, 6, 7]

>>> x.pop()

7

>>> x

[1, 2, 10, 4, 6]

---------------------------------

>>> x=[1, 2, 10, 4, 6]

>>> x.pop(2)

10

>>> x

[1, 2, 4, 6]
```

List

Remove: The **remove()** method removes the specified item from a given list.

```
>>> x=[1,33,2,10,4,6]
>>> x.remove(33)
>>> x
[1, 2, 10, 4, 6]
>>> x.remove(4)
>>> x
[1, 2, 10, 6]
```

Reverse: Reverse the order of a given list.

```
>>> x=[1,2,3,4,5,6,7]
>>> x.reverse()
>>> x
[7, 6, 5, 4, 3, 2, 1]
```

Sort: Sorts the elements in ascending order

```
>>> x=[7, 6, 5, 4, 3, 2, 1]
>>> x.sort()
>>> x
[1, 2, 3, 4, 5, 6, 7]
```

```
>>> x=[10,1,5,3,8,7]
>>> x.sort()
>>> x
[1, 3, 5, 7, 8, 10]
```

List loop

- Loops are control structures used to repeat a given section of code a certain number of times or until a particular condition is met.
- **Method #1: For loop**
- #list of items

```
list = ['M','R','C','E','T']
i= 1
#Iterating over the list
for item in list:
 print ('college ',i,' is ',item)
 i = i+1
```

```
college  1  is  M
college  2  is  R
college  3  is  C
college  4  is  E
college  5  is  T
```

Method #2: For loop and range()
In case we want to use the traditional for loop which iterates from number x to number y.

```
# Python3 code to iterate over a list
list = [1, 3, 5, 7, 9]
# getting length of list
length = len(list)
# Iterating the index
# same as 'for i in range(len(list))'
for i in range(length):
 print(list[i])
```

```
1
3
5
7
9
```

List loop

Method #3: using while loop

```
# Python3 code to iterate over a list
list = [1, 3, 5, 7, 9]
# Getting length of list
length = len(list)
i = 0
# Iterating using while loop
while i < length:
 print(list[i])
 i += 1
```

```
1
3
5
7
9
```