Python Arrays

In this lecture, you'll learn about arrays in Python. More specifically, you will learn to create arrays, modify them, access elements and so on with the help of examples.

Table of Contents

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`.

However, in Python, there is no native array data structure. So, we use Python lists instead of an array.

Note: If you want to create real arrays in Python, you need to use NumPy's array data structure. For mathematical problems, NumPy Array is more efficient.

Unlike arrays, a single list can store elements of any data type and does everything an array does. We can store an integer, a float and a string inside the same list. So, it is more flexible to work with.

`[10, 20, 30, 40, 50 ]` is an example of what an array would look like in Python, but it is actually a list.

List basics

A list in Python is just an ordered collection of items which can be of any type.

By comparison an array is an ordered collection of items of a single type –

so in principle a list is more flexible than an array but it is this flexibility that makes things slightly harder when you want to work with a regular structure.

A list is also a dynamic mutable type and this means you can add and delete elements from the list at any time.

- **To define a list you simply write a comma separated list of items in square brackets:**

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

This looks like an array because you can use "slicing" notation to pick out an individual element - indexes start from 0. For example

```
print myList[2]
```

will display the third element, i.e. the value 3 in this case. Similarly to change the third element you can assign directly to it:

```
myList[2]=100
```

The slicing notation looks like array indexing but it is a lot more flexible. For example

```
myList[2:5]
```

is a sublist from the third element to the fifth i.e. from myList[2] to myList[4]. notice that the final element specified i.e. [5] is not included in the slice.

Also notice that you can leave out either of the start and end indexes and they will be assumed to have their maximum possible value. For example

```
myList[5:]
```

is the list from List[5] to the end of the list and , Finally is it worth knowing that the list you assign to a slice doesn't have to be the same size as the slice - it simply replaces it even if it is a different size.

Create an Array

We can create a Python array with comma separated elements between `square brackets[]`.

Example 1: How to create an array in Python?

We can make an integer array and store it to `arr`.

```
arr = [10, 20, 30, 40, 50]
```

Access elements of an Array

We can access individual elements of an array using index inside `square brackets []`.

Array Index

Index is the position of element in an array. In Python, arrays are zero-indexed. This means, the element's position starts with 0 instead of 1.

Example 2: Accessing elements of array using indexing

```
arr = [10, 20, 30, 40, 50]
print(arr[0])
print(arr[1])
print(arr[2])
```

When we run the above program, the output will be:

```
10
20
30
```

Here, the first element of arr is `arr[0]`, second is `arr[1]`, third is `arr[2]`, and so on.

Negative Indexing

Python programming supports negative indexing of arrays, something that is not available in arrays in most programming languages.

This means the index value of -1 gives the last element, and -2 gives the second to last element of an array.

Example 3: Accessing elements of array using negative indexing

```
arr = [10, 20, 30, 40, 50]
print(arr[-1])
print(arr[-2])
```

When we run the above program, the output will be:

```
50
40
```

Find length of an Array

Python arrays are just lists, so finding the length of an array is equivalent to finding length of a list in Python.

Example 4: Find length of an array using len()

```
brands = ["Coke", "Apple", "Google", "Microsoft", "Toyota"]
num_brands = len(brands)
print(num_brands)
```

When we run the above program, the output will be:

```
5
```

As seen from the above example, the `len` function gives the length of array `brands` which is 5.

Add an element to an Array

To add a new element to an array, we use `append()` method in Python.

Example 5: Adding an element in an array using append()

```
add = ['a', 'b', 'c']
add.append('d')
print(add)
```

When we run the above program, the output will be

```
['a', 'b', 'c', 'd']
```

Here, we used `append()` method to add `'d'`.

Remove elements from an Array

Python's list implementation of array allows us to delete any elements from an array using `del` operator.

Similarly, we can also use `remove()` and `pop()` methods to remove elements in an array.

Example 6: Removing elements of an array using del, remove() and pop()

```
colors = ["violet", "indigo", "blue", "green", "yellow", "orange", "red"]
del color[4]
colors.remove("blue")
colors.pop(3)
print(color)
```

When we run the above program, the output will be

```
['violet', 'indigo', 'green', 'red']
```

Modify elements of an Array

We can change values of elements within an array using indexing and assignment operator (=). We select the position of any element using indexing and use assignment operator to provide a new value for the element.

Example 7: Modifying elements of an array using Indexing

```
fruits = ["Apple", "Banana", "Mango", "Grapes", "Orange"]
fruits[1] = "Pineapple"
fruits[-1] = "Guava"
print(fruits)
```

When we run the above program, the output will be:

```
['Apple', 'Pineapple', 'Mango', 'Grapes', 'Guava']
```

Python operators to modify elements in an Array

In Python arrays, operators like `+` , `*` can also be used to modify elements.

We can use `+` operator to concatenate (combine) two arrays.

Example 8: Concatenating two arrays using + operator

```
concat = [1, 2, 3]
concat + [4,5,6]
print(concat)
```

When we run the above program. the output will be:

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

Similarly, we can use `*` operator to repeat the elements multiple times.

Example 8: Repeating elements in array using * operator

```
repeat = ["a"]
repeat = repeat * 5
print(repeat)
```

When we run the above program, the output will be

```
['a', 'a', 'a', 'a', 'a'
```

Slicing an Array

Python has a slicing feature which allows to access pieces of an array. We, basically, slice an array using a given range (eg. 2nd to 5th position), giving us elements we require. This is done by using indexes separated by a colon `[x : y]`.

We can use negative indexing with slicing too.

Example 9: Slicing an array using Indexing

```
fruits = ["Apple", "Banana", "Mango", "Grapes", "Orange"]
print(fruits[1:4])
print(fruits[ : 3])
print(fruits[-4:])
print(fruits[-3:-1])
```

When we run the above program, the output will be:

```
['Banana', 'Mango', 'Grapes']

['Apple', 'Banana', 'Mango']

['Banana', 'Mango', 'Grapes', 'Orange']

['Mango', 'Grapes']
```

Python Array Methods

Other array operations are also available in Python using list/array methods given as:

Methods	Functions
append()	to add element to the end of the list
extend()	to extend all elements of a list to the another list
insert()	to insert an element at the another index
remove()	to remove an element from the list
pop()	to remove elements return element at the given index
clear()	to remove all elements from the list
index()	to return the index of the first matched element
count()	to count of number of elements passed as an argument
sort()	to sort the elements in ascending order by default
reverse()	to reverse order element in a list
copy()	to return a copy of elements in a list

Multidimensional arrays

All arrays created above are single dimensional. We can also create a multidimensional array in Python.

A multidimensional array is an array within an array. This means an array holds different arrays inside it.

Example 10: Create a two-dimensional array using lists

```
multd = [[1,2], [3,4], [5,6], [7,8]]
print(multd[0])
print(multd[3])
print(multd[2][1])
print(multd[3][0])
```

When we run the above program, the output will be

```
[1, 2]
[7, 8]
6
7
```

Here, we have 4 elements and each elements hold another 2 sub-elements.

Basic array operations

So far so good, and it looks as if using a list is as easy as using an array.

The first thing that we tend to need to do is to scan through an array and examine values. For example, to find the maximum value (forgetting for a moment that there is a built-in max function) you could use:

```
m=0
for e in myList:
 if m<e:
  m=e
```

This uses the for..in construct to scan through each item in the list. This is a very useful way to access the elements of an array but it isn't the one that most programmers will be familiar with. In most cases arrays are accessed by index and you can do this in Python:

```
m=0
for i in range(len(myList)):
 if m<myList[i]:
  m=myList[i]
```

or you could use the non-indexed loop and the index method:

```
m=0
for e in myList:
 if m<e:
  m=e
mi=myList.index(m)
print mi
```