3. Python Arrays

Arrays are a fundamental data structure, and an important part of most programming languages. In Python, they are containers which are able to store more than one item at the same time.
Specifically, they are an ordered collection of elements with every value being of the same data type. That is the most important thing to remember about Python arrays - the fact that they can only hold a sequence of multiple items that are of the same type.

3.1 Difference between Python Lists and Python Arrays

No	List	Array
1	List is used to collect items that usually consist of elements of multiple data types.	An array is also a vital component that collects several items of the same data type.
2	List cannot manage arithmetic operations.	Array can manage arithmetic operations.
3	It consists of elements that belong to the different data types.	It consists of elements that belong to the same data type.
4	When it comes to flexibility, the list is perfect as it allows easy modification of data.	When it comes to flexibility, the array is not suitable as it does not allow easy modification of data.
5	It consumes a larger memory.	It consumes less memory than a list.
6	In a list, the complete list can be accessed without any specific looping.	In an array, a loop is mandatory to access the components of the array.
7	It favors a shorter sequence of data.	It favors a longer sequence of data.

3.2 When to Use Python Arrays

Lists are built into the Python programming language, whereas arrays aren't. Arrays are not a built-in data structure, and therefore need to be imported via the `array module` in order to be used. They are also more compact and take up less memory and space which makes them more size efficient compared to lists, and are useful when you want to work with homogeneous data.
If you want to perform mathematical calculations, then you should use NumPy arrays by importing the NumPy package. Besides that, you should just use Python

arrays when you really need to, as lists work in a similar way and are more flexible to work with.

3.3 Use Arrays in Python

In order to create Python arrays, you'll first have to import the **array module** which contains all the necessary functions.

There are three ways you can import the **array module**:

1. By using **import array** at the top of the file. This includes the module **array**. You would then go on to create an array using **array.array().**

```
import array

#how you would create an array
array.array()
```

2. Instead of having to type **array.array()** all the time, you could use **import array as arr** at the top of the file, instead of **import array** alone. You would then create an array by typing **arr.array()**. The **arr** acts as an alias name, with the array constructor then immediately following it.

```
import array as arr

#how you would create an array
arr.array()
```

3. Lastly, you could also use **from array import ***, with ***** importing all the functionalities available. You would then create an array by writing the **array()** constructor alone.

```
from array import *

#how you would create an array
array()
```

3.4 Define Arrays in Python

Once you've imported the **array module**, you can then go on to define a Python array.
The general syntax for creating an array looks like this:

```
variable_name = array(typecode,[elements])
```

Below is a typecode table, with the different typecodes that can be used with the different data types when defining Python arrays:

pe code	C Type	Python Type	Minimum size in bytes
'b'	signed char	int	1
'B'	unsigned char	int	1
'u'	wchar_t	Unicode character	2
'h'	signed short	int	2
'H'	unsigned short	int	2
'i'	signed int	int	2
'I'	unsigned int	int	2
'l'	signed long	int	4

'L'	unsigned long	int	4
'q'	signed long long	int	8
'Q'	unsigned long long	int	8
'f'	float	float	4
'd'	double	float	8

Outline: Steps to create array:

- First we included the array module, in this case with **import array as arr** .
- Then, we created a **numbers** array.
- We used **arr.array()** because of **import array as arr** .
- Inside the **array()** constructor, we first included **i**, for signed integer. Signed integer means that the array can include positive *and* negative values. Unsigned integer, with **H** for example, would mean that no negative values are allowed.
- Lastly, we included the values to be stored in the array in square brackets.

 Keep in mind that if you tried to include values that were not of **i** typecode, meaning they were not integer values, you would get an error:

```
from array import *

#an array of floating point values
numbers = array('i',[10.0,20.0,30.0])

print(numbers)
```

```
muthanna@maxo79:~/Desktop$ python3 array1.py
Traceback (most recent call last):
  File "/home/muthanna/Desktop/array1.py", line 4, in <module>
    numbers = array('i',[10.0,20.0,30.0])
              ^^^^^^^^^^^^^^^^^^^^^^^^^^
TypeError: 'float' object cannot be interpreted as an integer
```

In the example above, I tried to include a floating point number in the array. I got an error because this is meant to be an integer array only.

see the correct

```
from array import *

#an array of floating point values
numbers = array('d',[10.0,20.0,30.0])

print(numbers)
```

```
muthanna@maxo79:~/Desktop$ python3 array2.py
array('d', [10.0, 20.0, 30.0])
```

3.5 Find the Length of an Array in Python

To find out the exact number of elements contained in an array, use the built-in **len()** method. It will return the integer number that is equal to the total number of elements in the array you specify.

```
import array as arr

numbers = arr.array('i',[10,20,30])

print(len(numbers))
```

```
muthanna@maxo79:~/Desktop$ python3 array3.py
3
```

3.6 Array Indexing and How to Access Individual Items in an Array in Python

Each item in an array has a specific address. Individual items are accessed by referencing their *index number*.
Indexing in Python, and in all programming languages and computing in general, starts at 0. It is important to remember that counting starts at 0 and **not** at 1.
To access an element, you first write the name of the array followed by square brackets. Inside the square brackets you include the item's index number.

The general syntax would look something like this:

```
array_name[index_value_of_item]
```

Here is how you would access each individual element in an array:

```
import array as arr

numbers = arr.array('i',[10,20,30])

print(numbers[0]) # gets the 1st element
print(numbers[1]) # gets the 2nd element
print(numbers[2]) # gets the 3rd element
```

```
muthanna@maxo79:~/Desktop$ python3 array4.py
10
20
30
```

Remember that the index value of the last element of an array is always one less than the length of the array. Where n is the length of the array, n - 1 will be the index value of the last item.
Note that you can also access each individual element using negative indexing.

With negative indexing, the last element would have an index of -1, the second to last element would have an index of -2, and so on.
Here is how you would get each item in an array using that method:

```
import array as arr

numbers = arr.array('i',[10,20,30])

print(numbers[-1]) #gets last item
print(numbers[-2]) #gets second to last item
print(numbers[-3]) #gets first item
```

```
muthanna@maxo79:~/Desktop$ python3 array5.py
30
20
10
```

3.7 Loop through an Array in Python

If you want to print each value one by one, For this you can use a simple **for** loop:

```
import array as arr

numbers = arr.array('i',[10,20,30])

for number in numbers:
    print(number)
```

```
muthanna@maxo79:~/Desktop$ python3 array6.py
10
20
30
```

Or use the **range()** function, and pass the **len()** method as its parameter. This would give the same result as above:

```
import array as arr

values = arr.array('i',[10,20,30])

#prints each individual value in the array
for value in range(len(values)):
    print(values[value])
```

```
muthanna@maxo79:~/Desktop$ python3 array7.py
10
20
30
```

4.8 Slice an Array in Python

To access a specific range of values inside the array, use the slicing operator, which is a colon `:` .

When using the slicing operator and you only include one value, the counting starts from **0** by default. It gets the first item, and goes up to but not including the index number you specify.

```
import array as arr

#original array
numbers = arr.array('i',[10,20,30])

#get the values 10 and 20 only
print(numbers[:2])  #first to second position
```

```
muthanna@maxo79:~/Desktop$ python3 array8.py
array('i', [10, 20])
```

When you pass two numbers as arguments, you specify a range of numbers. In this case, the counting starts at the position of the first number in the range, and up to but not including the second one:

```
import array as arr

#original array
numbers = arr.array('i',[10,20,30])

#get the values 20 and 30 only
print(numbers[1:3]) #second to third position
```

```
muthanna@maxo79:~/Desktop$ python3 array9.py
array('i', [20, 30])
```

4.9 Methods For Performing Operations on Arrays in Python

Arrays are mutable, which means they are changeable. You can change the value of the different items, add new ones, or remove any you don't want in your program anymore.

1. **Change the Value of an Item in an Array:**

You can change the value of a specific element by speficying its position and assigning it a new value:

```
import array as arr

#original array
numbers = arr.array('i',[10,20,30])

#change the first element
#change it from having a value of 10 to having a value of 40
numbers[0] = 40

print(numbers)
```

```
muthanna@maxo79:~/Desktop$ python3 array10.py
array('i', [40, 20, 30])
```

2. Add a New Value to an Array:

To add one single value at the end of an array, use the **append()** method:

```
import array as arr

#original array
numbers = arr.array('i',[10,20,30])

#add the integer 40 to the end of numbers
numbers.append(40)

print(numbers)
```

```
muthanna@maxo79:~/Desktop$ python3 array11.py
array('i', [10, 20, 30, 40])
```

Be aware that the new item you add needs to be the same data type as the rest of the items in the array.

Look what happens when I try to add a float to an array of integers:

```
import array as arr

#original array
numbers = arr.array('i',[10,20,30])

#add the float 40.0 to the end of numbers
numbers.append(40.0)

print(numbers)
```

```
muthanna@maxo79:~/Desktop$ python3 array12.py
Traceback (most recent call last):
  File "/home/muthanna/Desktop/array12.py", line 7, in <module>
    numbers.append(40.0)
TypeError: 'float' object cannot be interpreted as an integer
```

3. Add more than one value to the end an array:

Use the **extend()** method, which takes an iterable (such as a list of items) as an argument. Again, make sure that the new items are all the same data type.

```
import array as arr

#original array
numbers = arr.array('i',[10,20,30])

#add the integers 40,50,60 to the end of numbers
#The numbers need to be enclosed in square brackets

numbers.extend([40,50,60])

print(numbers)
```

```
muthanna@maxo79:~/Desktop$ python3 array13.py
array('i', [10, 20, 30, 40, 50, 60])
```

4. Add an item in any position of an array:

Use the **insert()** method, to add an item at a specific position.

The **insert()** function takes two arguments: the index number of the position the new element will be inserted, and the value of the new element.

```
import array as arr

#original array
numbers = arr.array('i',[10,20,30])

#add the integer 40 in the first position
#remember indexing starts at 0

numbers.insert(0,40)

print(numbers)
```

```
muthanna@maxo79:~/Desktop$ python3 array14.py
array('i', [40, 10, 20, 30])
```

5. Remove a Value from an Array

To remove an element from an array, use the **remove()** method and include the value as an argument to the method.

```
import array as arr

#original array
numbers = arr.array('i',[10,20,30])

numbers.remove(10)

print(numbers)
```

```
muthanna@maxo79:~/Desktop$ python3 array15.py
array('i', [20, 30])
```

With **remove()**, only the first instance of the value you pass as an argument will be removed.
See what happens when there are more than one identical values:

```
import array as arr

#original array
numbers = arr.array('i',[10,20,30,10,20])

numbers.remove(10)

print(numbers)
```

```
muthanna@maxo79:~/Desktop$ python3 array16.py
array('i', [20, 30, 10, 20])
```

Only the first occurence of 10 is removed.

You can also use the **pop()** method, and specify the position of the element to be removed:

```
import array as arr

#original array
numbers = arr.array('i',[10,20,30,10,20])

#remove the first instance of 10
numbers.pop(0)

print(numbers)
```

```
muthanna@maxo79:~/Desktop$ python3 array17.py
array('i', [20, 30, 10, 20])
```