كــليـة الـتقنيــات الصحية والطبيـة

قســـم الانـظـمـة الـطبية الـذكـيـة

Intelligent Medical Systems Department

Lecture: (1)

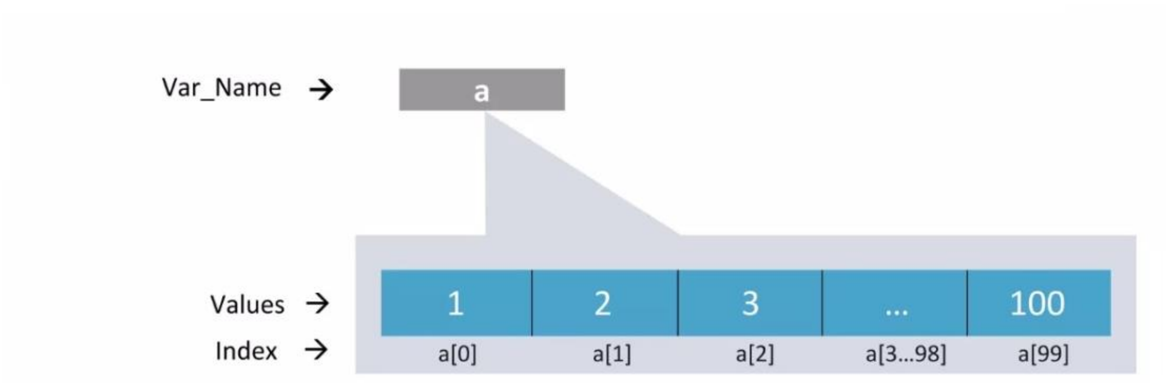
Arrays in Python

Subject: Data Structure Lab.
Class: Second
Lecturer: Asst. Prof. Mehdi Ebady Manaa
Asst. Lec. Sajjad Ibrahim Ismael

Why use Arrays in Python?

A combination of Arrays, together with Python could save you a lot of time. As mentioned earlier, 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.



Now that you are aware of the importance of arrays in Python, let's study more about it in detail.

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.

Example:

```
a=array.array('d',[1.2,1.3,2.3])
```

Now, there is always a question that comes up to our mind –

Is Python list same as an Array?

Python Arrays and lists are 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 intergers, 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 array

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 array 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:

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

OR

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

Example: a=array.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.

Type code	Python Data Type	C Type	Byte size
I	int	Signed int	2
I	int	Unsigned int	2
U	unicode character	Py_UNICODE	2
H	int	Signed short	2
H	int	Unsigned short	2
L	int	Signed long	4
L	int	Unsigned log	4
F	float	Float	4
D	float	Double	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.

Example:

```
import array as array

a=array.array( 'd', [1.1 , 2.1 ,3.1] )

print(a[1])
```

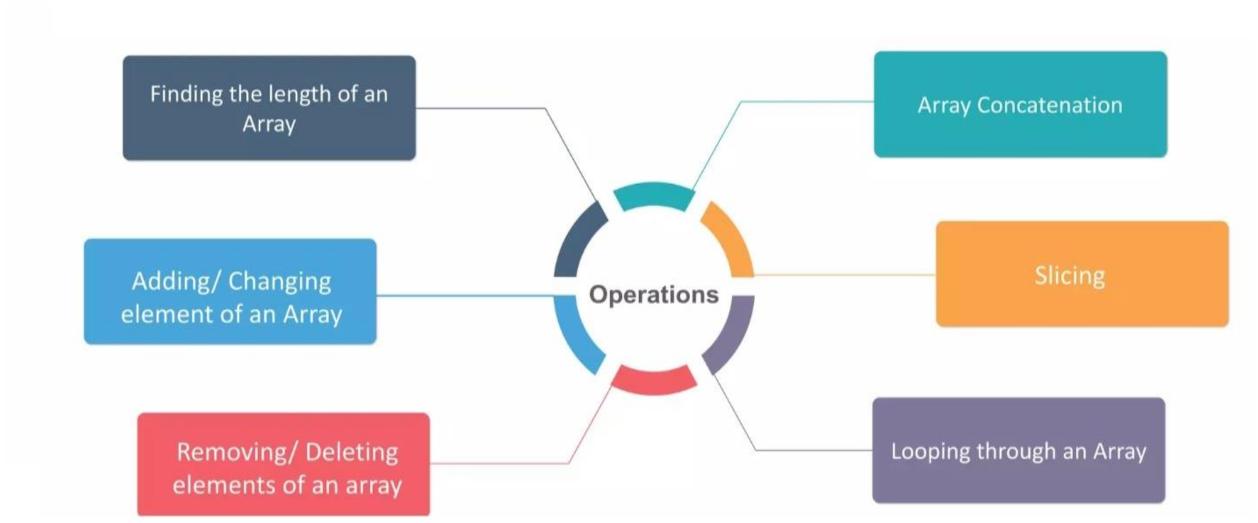
Output –

```
2.1
```

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

Basic array operations:

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


Finding the length of an Array
Array Concatenation
Adding/ Changing element of an Array
Operations
Slicing
Removing/ Deleting elements of an array
Looping through an Array

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:

```
import array as array

a=array.array('d', [1.1 , 2.1 ,3.1] )

print(len(a))
```

Output –

```
2.1
```

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:

```
import array as array

a=array.array('d', [1.1 , 2.1 ,3.1] )

a.append(3.4)

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:

```
import array as array

a=array.array('d', [1.1 , 2.1 ,3.1] )

a.extend([4.5,6.3,6.8])

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:

```
import array as array

a=array.array('d', [1.1 , 2.1 ,3.1] )

a.insert(2,3.8)

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.

```
import array as array

a=array.array('d',[1.1 , 2.1 ,3.1,2.6,7.8])

b=array.array('d',[3.7,8.6])

c=array.array('d')

c=a+b

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.

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.

```
import array as array
a=array.array('d', [1.1, 2.2, 3.8, 3.1, 3.7, 1.2, 4.6])
print(a.pop())
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.

```
import array as array
a=array.array('d',[1.1 , 2.1 ,3.1])
a.remove(1.1)
print(a)
```

Output –

```
array('d', [2.1,3.1])
```

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

Slicing an array:

An array can be sliced using the : symbol. This returns a range of elements that we have specified by the index numbers.

Example:

```
import array as array
a=array.array('d',[1.1 , 2.1 ,3.1,2.6,7.8])
print(a[0:3])
```

Output –

```
array(‘d’, [1.1, 2.1, 3.1])
```

The result will be elements present at 1st, 2nd and 3rd position in the array.

Looping through an array:

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

Example:

```
import array as array
a=array.array('d', [1.1, 2.2, 3.8, 3.1, 3.7, 1.2, 4.6])
print("All values")
for x in a:
  print(x)
print("specific values")
for x in a[1:3]:
  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.

Looping through an array using while loop

Example:

```
import array as array
a=array.array('d', [1.1, 2.2, 3.8, 3.1, 3.7])
b=0
while b<len(a) :
    print(a[b])
    b=b+1
```

Output –

```
1.1
2.2
3.8
3.1
3.7
```