LECTURE 8: ARRAYS

BY LINA HAMMAD & AHMAD BARGHASH

In this lecture, we will learn how to use arrays to store multiple values under one variable name and access them by an index.

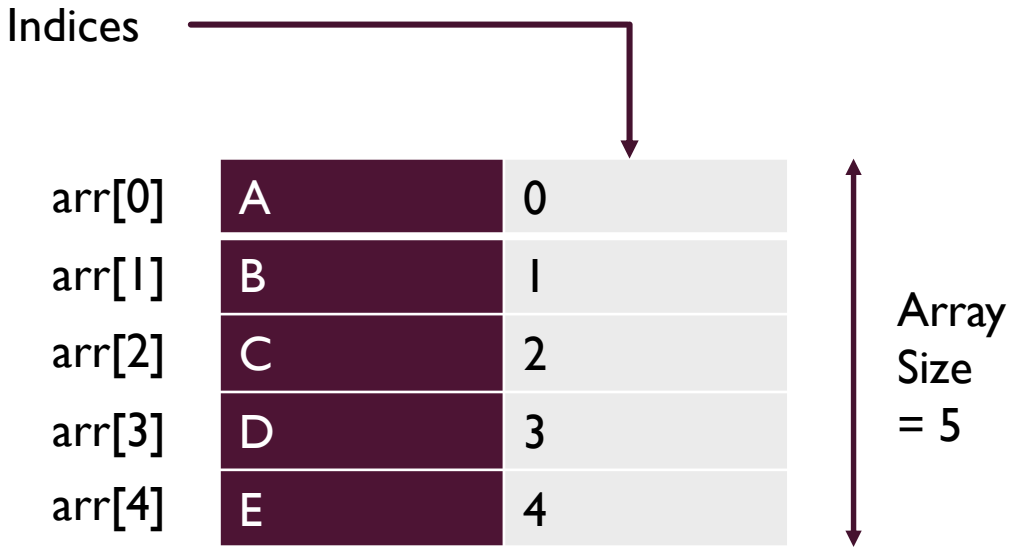
ARRAYS

- Array is one of the most fundamental data structure in practically all programming languages. The idea behind an array is to store multiple items of the same (or different such as in kotlin) data-type, such as an *integer* or *string* under a single variable name.
- Arrays are used to organize data in programming so that a related set of values can be easily sorted or searched. Here are some basic properties of arrays:
  - They are stored in contiguous memory locations.
  - They can be accessed programmatically through their indexes (array[1], array[0], etc.)
  - They are mutable.
  - Their size is fixed.

KOTLIN ARRAY

- An array is a container that holds data (values) of one single type. For example, you can create an array that can hold 100 values of Int type.
- In Kotlin, arrays are not a native data type, but a mutable collection of similar items which are represented by the Array class.
- Arrays in Kotlin are able to store multiple values of same or different data types.
- In Kotlin, arrays are represented by the Array class. The class has get and set functions, size property, and a few other useful member functions.

KOTLIN ARRAY REPRESENTATION



KOTLIN ARRAY DECLARATION

- There are **two** ways to define and declare an array in Kotlin and fill it with initial values:
  1. Using the arrayOf() function.
  2. Using the Array constructor.
  3. Using built-in functions for primitive types

KOTLIN ARRAY DECLARATION USING ARRAYOF()

- We can use the library function arrayOf() to create an array by passing the values of the elements to the function as the following example:

```
val num1 = arrayOf(1, 2, 3, 4)   //implicit type declaration
val num2 = arrayOf<Int>(1, 2, 3) //explicit type declaration
```

- Array that holds multiple different data types:

```
var arr = arrayOf(10, "BeginnersBook", 10.99, 'A')
```

- Array that can **only** hold integers:

```
var arr = arrayOf(1, 22, 55)
```

- Array that can **only** hold strings:

```
var arr2 = arrayOf("ab", "bc", "cd")
```

KOTLIN PROGRAM OF CREATING ARRAY USING ARRAYOF() AND ARRAYOF<INT> FUNCTIONS

```
fun main()
{
    // declaring an array using arrayOf()
    val arrayname = arrayOf(1, 2, 3, 4, 5)
    for (i in 0..arrayname.size-1)
    {
        print(" "+arrayname[i])
    }
    println()
    // declaring an array using arrayOf<Int>
    val arrayname2 = arrayOf<Int>(10, 20, 30, 40, 50)
    for (i in 0..arrayname2.size-1)
    {
        print(" "+arrayname2[i])
    }
}
```

The output is:

```
1 2 3 4 5

10 20 30 40 50
```

KOTLIN ARRAY DECLARATION USING CONSTRUCTOR

- Since Array is a class in Kotlin, we can also use the Array constructor to create an array. The constructor takes **two** parameters:
  1. The size of the array, and
  2. A function which accepts the **index** of a given element and returns the **initial value** of that element.

```
val arr = Array(3, {i-> i*1})
```

KOTLIN PROGRAM OF CREATING ARRAY USING CONSTRUCTOR

```
fun main()
{
    val arrayname = Array(5, { i -> (3.14*i*i) })
    for (i in 0..arrayname.size-1)
    {
        println(arrayname[i])
    }
}
```

The output is:

```
0.0
3.14
12.56
28.259999999999998
50.24
```

BUILT-IN FACTORY METHODS

- Kotlin also has some **built-in** factory methods to create arrays of primitive data types, such as byteArray, intArray, shortArray, etc. These classes do not extend the Array class; however, they implement the same methods and properties.
- For example, the factory method to create an integer array is:

```
val num = intArrayOf(1, 2, 3, 4)
```

- Other factory methods available for creating arrays:
  - byteArrayOf()
  - charArrayOf()
  - shortArrayOf()
  - longArrayOf()

ACCESSING AND MODIFYING ARRAYS

- There two ways to access and modify arrays:
  1. Using get() and set() methods.
  2. Using the index operator [ ].

SUMMARY

- Get():
  - In the previous example we have used the following statement to access the value of **4th** element of the array.

```
arr[3]
```

  - We can rewrite the same statement using get function like this:

```
arr.get(3)
```

- Set():
  - In the previous example we have used the following statement to modify the value of **4th** element in the array.

```
arr[3] = 100
```

  - We can rewrite the same statement using get function like this:

```
arr.set(3, 100)
```

ACCESS ARRAY ELEMENTS USING SET AND GET FUNCTIONS

- As you know, an array in Kotlin is basically a class. Therefore, we can access the data of a class object via its member functions. The get() and set() functions are said to be member functions.
- The get() method takes a single parameter, the index of the element and returns the value of the item at that index.
- Syntax:

```
val x = arr.get(0)
```

- The set() method takes 2 parameters: the index of the element and the value to be inserted.
- Syntax:

```
arr.set(1, 3)
```

ACCESS ARRAY ELEMENTS USING [ ] OPERATOR

- The [ ] operator can be used to access and modify arrays.
- To access an array element, the syntax would be:

```
val x = arr[1]
```

- This will assign the value of the second element in num to x.
- To modify an array element, the syntax would be:

```
arr[2] = 5
```

- This will change the value of the third element in the num array to 5.
- Note: Internally, the index operator or [ ] is actually an overloaded operator (see operator overloading) and only stands for calls to the get() and set() member functions.

ACCESS ARRAY ELEMENTS IN KOTLIN

- In the following example, we have an array arr that has multiple elements of different data types, we are displaying the 4th element of the array using the index (arr[3]). Since array is mutable, we can change the value of an element, which we have shown in the example below.

```
fun main(){
    //Array that holds multiple different data types
    var arr = arrayOf(10, "BeginnersBook", 10.99, 'A')

    //Displaying 4th element
    println(arr[3])

    //modifying 4th element
    arr[3] = 100

    //Displaying 4th element again
    println(arr[3])
}
```

Mutable

The output is:

```
A
100
```

BASIC ARRAY EXAMPLE

```
fun main()
{   // declare an array using arrayOf()
    val num = arrayOf(1, 2, 3, 4, 5)

    num.set(0, 10)  // set the first element to 10
    num.set(1, 6)   // set the second element to 6

    println(num.get(0)) // print the first element using get()
    println(num[1]) // print the second element using []
}
```

The output is:

```
10

6
```

FUNCTIONS READY TO USE

```
fun main(args : Array<String>){
    var arr = arrayOf(1, 2, "kotlin", "is","useful")

    println("Size of Array arr is: ${arr.size}")

    println("The available indices are: ${arr.indices}")

    println("Array contains is: ${arr.contains("is")}")

    println("Array contains fun: ${arr.contains("fun")}")

    println("First Element: ${arr.first()}")

    println("Last Element: ${arr.last()}")

    println("Index of kotlin: ${arr.indexOf("kotlin")}")

}
```

The output is:

```
Size of Array arr is: 5
The available indices are: 0..4
Array contains is: true
Array contains fun: false
First Element: 1
Last Element: useful
Index of kotlin: 2
```

CHECK THE ELEMENT IN AN ARRAY

- We can also check whether an element exists in array or not using the contains().

```
fun main(args : Array<String>){

    var arr = arrayOf(1, 22, "CPS")

    println("Array contains CPS: ${arr.contains("CPS")}")
    println("Array contains APS: ${arr.contains("APS")}")

}
```

The output is:

```
Array contains CPS: true

Array contains APS: false
```

KOTLIN ARRAY FIRST() AND LAST() FUNCTIONS

- We can easily find out the first and last element of an array using first() and last() function.

```
fun main(args : Array<String>){

    var arr = arrayOf(1, 22, "CPS")

    println("First Element: ${arr.first()}")
    println("Last Element: ${arr.last()}")

}
```

The output is:

```
First Element: 1

Last Element: CPS
```

FINDING THE INDEX OF AN ELEMENT IN AN ARRAY

- We can find out the index of a particular element in an array using indexOf() function.

```
fun main(args : Array<String>){
    var arr = arrayOf(1, 22, "CPS")

    println("Index of 22: ${arr.indexOf(22)}")
}
```

The output is:

```
Index of 22: 1
```

TRAVERSING ARRAYS

- One important property of an array is that it can be traversed programmatically, and each element in the array can be manipulated individually. Kotlin supports few powerful ways to traverse array.
- The simplest and most commonly used idiom when it comes to traversing an array is to use the for-loop.
- We can track the array indices using the indices and size functions
- You can track the array elements directly using "in" followed by the array name

```
for(i in arr.indices){
    println(arr[i])
}
```

```
for (index in 0..arr.size-1){
        println(arr[index])
```

```
for(element in arr){
    println(element)
```

TRAVERSING ARRAYS EXAMPLE 1

```
// Traversing an array
fun main()
{
    val arr = arrayOf(1, 2, 3, 4, 5)
    num.set(0, 10)
    num.set(1, 6)
    for (i in arr.indices)
    {
        println(arr[i])
    }
}
```

The output is:

```
10
6
3
4
5
```

TRAVERSING ARRAYS EXAMPLE 2

```
// Traversing an array
fun main()
{
    val arrayname = arrayOf<Int>(1, 2, 3, 4, 5)
    for (i in 0..arrayname.size-1)
    {
        println(arrayname[i])
    }
}
```

The output is:

```
1
2
3
4
5
```

ALL IN ONE SLIDE

```
fun main(){
    val name = arrayOf<String>("Ajay","Prakesh","Michel","John","Sumit")
    var myArray2 = arrayOf<Int>(1,10,4,6,15)
    var myArray3 = arrayOf(5,10,20,12,15)
    var myArray4 = arrayOf(1,10,4, "Ajay","Prakesh")
    var myArray5: IntArray = intArrayOf(5,10,20,12,15)

    for(element in name){
        println(element)
    }
    for(element in myArray2){
        println(element)
    }
    for(element in myArray3){
        println(element)
    }
    for(element in myArray4){
        println(element)
    }
    for(element in myArray5){
        println(element)
    }
}
```

What is the output ?!

KOTLIN ARRAY EXAMPLE

```
fun main(args: Array<String>){
    var myArray5: IntArray = intArrayOf(5,10,20,12,15)

    myArray5[6]=18 // ArrayIndexOutOfBoundsException
    for(element in myArray5){
        println(element)
    }
}
```

- This code will throw an *ArrayIndexOutOfBoundException*. This is because the index value is not present at which we tried to insert element. Due to this, array is called **fixed size** length.

FILL ARRAY FROM USER INPUT

```
fun main ()
{
    var ArrayToFill = Array<Int>(5){0}
    var index:Int = 0
    while( index < ArrayToFill.size)
    {
        println("Enter element no. $index")
        ArrayToFill[index] = readLine()!!.toInt()
        index++
    }
    println(index)
    for(index in 0..4)
    {
        println(ArrayToFill[index])
    }
}
```

The output is:

```
Enter element no. 0
1
Enter element no. 1
2
Enter element no. 2
3
Enter element no. 3
4
Enter element no. 4
5
5
1
2
3
4
5
```

ASSIGNMENT 1

- Write a program that declares and fills an array of size 5 from user input. Your program should calculate the following
  - how many numbers are divisible by 3
  - The average of integers in even positions/indexes

ASSIGNMENT 2

- Write a program that declares and fills an integer array then sort it using the following algorithm then print the sorted version

  Sorting Algorithm

  - Step 1: Set i to 0.
  - Step 2: Set j to i + 1.
  - Step 3: If a[i] > a[j], exchange their values.
  - Step 4: Set j to j + 1.If j < n, go to step 3.
  - Step 5: Set i to i + 1.If i < n – 1, go to step 2.
  - Step 6: a is now sorted in ascending order.