SCALA ARRAYS

Scala provides a data structure, the **array**, which stores a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.

Instead of declaring individual variables, such as number0, number1, ..., and number99, you declare one array variable such as numbers and use numbers[0], numbers[1], and ..., numbers[99] to represent individual variables. This tutorial introduces how to declare array variables, create arrays, and process arrays using indexed variables. The index of the first element of an array is the number zero and the index of the last element is the total number of elements minus one.

Declaring Array Variables:

To use an array in a program, you must declare a variable to reference the array and you must specify the type of array the variable can reference. Here is the syntax for declaring an array variable:

```
var z:Array[String] = new Array[String](3)

or

var z = new Array[String](3)
```

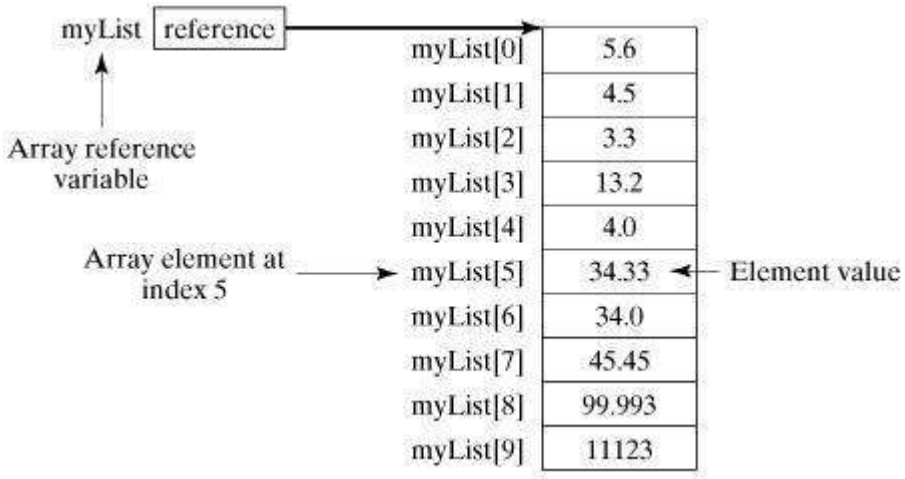
Here, z is declared as an array of Strings that may hold up to three elements. You can assign values to individual elements or get access to individual elements, it can be done by using commands like the following:

```
z(0) = "Zara"; z(1) = "Nuha"; z(4/2) = "Ayan"
```

Here, the last example shows that in general the index can be any expression that yields a whole number. There is one more way of defining an array:

```
var z = Array("Zara", "Nuha", "Ayan")
```

Following picture represents array myList. Here, myList holds ten double values and the indices are from 0 to 9.



Processing Arrays:

When processing array elements, we often use either for loop because all of the elements in an array are of the same type and the size of the array is known. Here is a complete example of showing how to create, initialize and process arrays:

```
object Test {
   def main(args: Array[String]) {
      var myList = Array(1.9, 2.9, 3.4, 3.5)

      // Print all the array elements
      for ( x <- myList ) {
         println( x )
      }

      // Summing all elements
      var total = 0.0;
      for ( i <- 0 to (myList.length - 1)) {
         total += myList(i);
      }
      println("Total is " + total);

      // Finding the largest element
      var max = myList(0);
      for ( i <- 1 to (myList.length - 1) ) {
         if (myList(i) > max) max = myList(i);
      }
      println("Max is " + max);

   }
}
```

When the above code is compiled and executed, it produces the following result:

```
C:/>scalac Test.scala
C:/>scala Test
1.9
2.9
3.4
3.5
Total is 11.7
Max is 3.5

C:/>
```

Multi-Dimensional Arrays:

There are many situations where you would need to define and use multi-dimensional arrays *i. e.* , *arrayswhoseelementsarearrays*. For example, matrices and tables are examples of structures that can be realized as two-dimensional arrays.

Scala does not directly support multi-dimensional arrays and provides various methods to process arrays in any dimension. Following is the example of defining a two-dimensional array:

```
var myMatrix = ofDim[Int](3,3)
```

This is an array that has three elements each being an array of integers that has three elements. The code that follows shows how one can process a multi-dimensional array:

```
import Array._

object Test {
   def main(args: Array[String]) {
      var myMatrix = ofDim[Int](3,3)

      // build a matrix
      for (i <- 0 to 2) {
         for ( j <- 0 to 2) {
            myMatrix(i)(j) = j;
         }
      }

      // Print two dimensional array
```

```
      for (i <- 0 to 2) {
         for ( j <- 0 to 2) {
            print(" " + myMatrix(i)(j));
         }
         println();
      }

   }
}
```

When the above code is compiled and executed, it produces the following result:

```
C:/>scalac Test.scala
C:/>scala Test
0 1 2
0 1 2
0 1 2

C:/>
```

Concatenate Arrays:

Following is the example which makes use of concat method to concatenate two arrays. You can pass more than one array as arguments to concat method.

```
import Array._

object Test {
   def main(args: Array[String]) {
      var myList1 = Array(1.9, 2.9, 3.4, 3.5)
      var myList2 = Array(8.9, 7.9, 0.4, 1.5)

      var myList3 =  concat( myList1, myList2)

      // Print all the array elements
      for ( x <- myList3 ) {
         println( x )
      }
   }
}
```

When the above code is compiled and executed, it produces the following result:

```
C:/>scalac Test.scala
C:/>scala Test
1.9
2.9
3.4
3.5
8.9
7.9
0.4
1.5

C:/>
```

Create Array with Range:

Following is the example, which makes use of range method to generate an array containing a sequence of increasing integers in a given range. You can use final argument as step to create the sequence; if you do not use final argument, then step would be assumed as 1.

```
import Array._

object Test {
   def main(args: Array[String]) {
```

```
      var myList1 = range(10, 20, 2)
      var myList2 = range(10,20)

      // Print all the array elements
      for ( x <- myList1 ) {
         print( " " + x )
      }
      println()
      for ( x <- myList2 ) {
         print( " " + x )
      }
   }
}
```

When the above code is compiled and executed, it produces the following result:

```
C:/>scalac Test.scala
C:/>scala Test
10 12 14 16 18
10 11 12 13 14 15 16 17 18 19

C:/>
```

Scala Arrays Methods:

Following are the important methods, which you can use while playing with array. As shown above, you would have to import **Array._** package before using any of the mentioned methods. For a complete list of methods available, please check official documentation of Scala.

SN	Methods with Description
1	**def apply***x*: *T*, *xs*: *T* ∗ **: Array[T]** Creates an array of T objects, where T can be Unit, Double, Float, Long, Int, Char, Short, Byte, Boolean.
2	**def concat[T]***xss*: *Array*[*T*] ∗ **: Array[T]** Concatenates all arrays into a single array.
3	**def copy***src*: *AnyRef*, *srcPos*: *Int*, *dest*: *AnyRef*, *destPos*: *Int*, *length*: *Int*: **Unit** Copy one array to another. Equivalent to Java's System.arraycopy *src*, *srcPos*, *dest*, *destPos*, *length*.
4	**def empty[T]: Array[T]** Returns an array of length 0
5	**def iterate[T]***start*: *T*, *len*: *Intf*: (*T* **=> T): Array[T]** Returns an array containing repeated applications of a function to a start value.
6	**def fill[T]***n*: *Intelem*:=> *T*: **Array[T]** Returns an array that contains the results of some element computation a number of

times.

7

def fill[T]_n1_**:** _Int_**,** _n2_**:** _Intelem_**:=>** _T_**: Array[Array[T]]**

Returns a two-dimensional array that contains the results of some element computation a number of times.

8

def iterate[T]_start_**:** _T_**,** _len_**:** _Intf_**: (**_T_ **=> T): Array[T]**

Returns an array containing repeated applications of a function to a start value.

9

def ofDim[T]_n1_**:** _Int_**: Array[T]**

Creates array with given dimensions.

10

def ofDim[T]_n1_**:** _Int_**,** _n2_**:** _Int_**: Array[Array[T]]**

Creates a 2-dimensional array

11

def ofDim[T]_n1_**:** _Int_**,** _n2_**:** _Int_**,** _n3_**:** _Int_**: Array[Array[Array[T]]]**

Creates a 3-dimensional array

12

def range_start_**:** _Int_**,** _end_**:** _Int_**,** _step_**:** _Int_**: Array[Int]**

Returns an array containing equally spaced values in some integer interval.

13

def range_start_**:** _Int_**,** _end_**:** _Int_**: Array[Int]**

Returns an array containing a sequence of increasing integers in a range.

14

def tabulate[T]_n_**:** _Intf_**: (**_Int_**=> T): Array[T]**

Returns an array containing values of a given function over a range of integer values starting from 0.

15

def tabulate[T]_n1_**:** _Int_**,** _n2_**:** _Intf_**: (**_Int_**,** _Int_ **=> T): Array[Array[T]]**

Returns a two-dimensional array containing values of a given function over ranges of integer values starting from 0.