Experiment No 10

NAME: Amitesh Sawarkar SAPID:60009210105 BATCH:D12

Aim: - Implement Scala programs to demonstrate Functions Without Any Argument and Return Type, Function to accept another Function as an Argument, Function accepting list and an anonymous Function as argument.

Scala Function

Scala supports functional programming approach. It provides rich set of built-in functions and allows you to create user defined functions also. In Scala, functions are first class values. You can store function value, pass function as an argument and return function as a value from other function. You can create function by using **def** keyword. You must mention return type of parameters while defining function and return type of a function is optional. If you don't specify return type of a function, default return type is Unit.

Scala Function Declaration Syntax

```
def functionName (parameters: typeofparameters): returntypeoffunction = {
// statements to be executed
}
```

You can create function with or without = (equal) operator. If you use it, function will return value. If you don't use it, your function will not return anything and will work like subroutine.

Scala Function Example without using = Operator

The function defined below is also known as non-parameterized function.

```
object MainObject {
  def main(args: Array[String]) {
     functionExample()          // Calling function
   }
   def functionExample () {        // Defining a function
       println ("This is a simple function")
   }
}
```

Scala Function Example with = Operator

```
object MainObject {
  def main (args: Array[String]) {
     var result = functionExample ()        // Calling function
     println(result)
   }
   def functionExample () = {       // Defining a function
       var a = 10
       a
   }
}
```

Scala Parameterized Function Example

when using parameterized function, you must mention type of parameters explicitly otherwise compiler throws an error and your code fails to compile.

```
object MainObject {
  def main (args: Array[String]) = {
     functionExample (10,20)
   }
   def functionExample (a: Int, b: Int) = {
```

```
        var c = a+b
        println(c)
    }
}
```

Scala Parameterized Function Example

when using parameterized function, you must mention type of parameters explicitly otherwise compiler throws an error and your code fails to compile.

```
object MainObject {
  def main (args: Array[String]) = {
      var b: Int=0
    b=functionExample (10,20)
    print(b)
   }
   def functionExample (a: Int, b: Int): Int= {
        var c = a+b
        println(c)
        return c

   }
}
```

Scala Recursion Function

In the program given below, we are multiplying two numbers by using recursive function. In Scala, you can create recursive functions also. Be careful while using recursive function. There must be a base condition to terminate program safely.

```
object MainObject {
  def main (args: Array[String]) = {
      var result = functionExample (15,2)
      println(result)
   }
   def functionExample (a: Int, b: Int): Int = {
      if (b == 0)         // Base condition
       0
      else
       a+functionExample (a, b-1)
   }
}
```

Function Parameter with Default Value

Scala provides a feature to assign default values to function parameters. It helps in the scenario when you don't pass value during function calling. It uses default values of parameters.

Scala Function Parameter example with default value

```
object MainObject {
  def main(args: Array[String]) = {
      var result1 = functionExample(15,2)     // Calling with two values
```

```
    var result2 = functionExample(15)   // Calling with one value
    var result3 = functionExample()     // Calling without any value
    println(result1+"\n"+result2+"\n"+result3)
  }
  def functionExample (a:Int = 0, b:Int = 0):Int = {   // Parameters with default values as 0
    a+b
  }
}
```

Scala Function Named Parameter Example

In Scala function, you can specify the names of parameters during calling the function. In the given example, you can notice that parameter names are passing during calling. You can pass named parameters in any order and can also pass values only.

```
object MainObject {
  def main(args: Array[String]) = {
    var result1 = functionExample(a = 15, b = 2) // Parameters names are passed during call

    var result2 = functionExample(b = 15, a = 2)// Parameters order have changed during ca
ll
    var result3 = functionExample(15,2) // Only values are passed during call
    println(result1+"\n"+result2+"\n"+result3)
  }
  def functionExample(a:Int, b:Int):Int = {
    a+b
  }
}
```

Scala Higher Order Functions

Higher order function is a function that either takes a function as argument or returns a function. In other words, we can say a function which works with function is called higher order function.

Higher order function allows you to create lambda function or anonymous function etc.

Scala Example: Passing a Function as Parameter in a Function

```
object MainObject {
  def main(args: Array[String]) = {
   functionExample(25, multiplyBy2)              // Passing a function as parameter
  }
  def functionExample(a:Int, f:Int=>AnyVal):Unit = {
    println(f(a))                          // Calling that function
  }
  def multiplyBy2(a:Int):Int = {
    a*2
  }
}
```

Scala Anonymous (lambda) Function

Anonymous function is a function that has no name but works as a function. It is good to create an anonymous function when you don't want to reuse it latter.

You can create anonymous function either by using => (rocket) or _ (underscore) wild card in scala.

Scala Anonymous function Example

```
object MainObject {
  def main(args: Array[String]) = {
    var result1 = (a:Int, b:Int) => a+b       // Anonymous function by using => (rocket)
    var result2 = (_:Int)+(_:Int)             // Anonymous function by using _ (underscore) wild
card
    println(result1(10,10))
    println(result2(10,10))
   }
}
```

Scala Function Currying

In scala, method may have multiple parameter lists. When a method is called with a fewer number of parameter lists, then this will yield a function taking the missing parameter lists as its arguments.

In other words it is a technique of transforming a function that takes multiple arguments into a function that takes a single argument.

Scala Function Currying Example

```
object MainObject {
   def add(a:Int)(b:Int) = {
      a+b
   }
   def main(args: Array[String]) = {
      var result = add(10)(10)
      println("10 + 10 = "+result)
      var addIt = add(10)_
      var result2 = addIt(3)
      println("10 + 3 = "+result2)
   }
}
```

Scala Nested Functions Example

```
object MainObject {
   def add(a:Int, b:Int, c:Int) = {
      def add2(x:Int,y:Int) = {
         x+y
      }
      add2(a,add2(b,c))
   }
   def main(args: Array[String]) = {
      var result = add(10,10,10)
      println(result)
   }
}
```

Scala Function with Variable Length Parameters

In scala, you can define function of variable length parameters. It allows you to pass any number of arguments at the time of calling the function.

Scala Example: Function with Variable Length Parameters

```
def add(args: Int*) = {
```

```
    var sum = 0;
    for(a <- args) sum+=a
    sum
}
var sum = add(1,2,3,4,5,6,7,8,9);
println(sum);
```

Programming Exercise to performed in this session: -

1. Write a Scala program to check whether a given positive number is a multiple of 3 or a multiple of 7

```scala
import scala.io.StdIn

object MyClass {
  def main(args: Array[String]): Unit = {
    println("Enter a positive number:")
    var n = StdIn.readInt()

    if (n >= 0) {
      if ((n % 3 == 0) || (n % 7 == 0)) {
        println(s"$n is a multiple of 3 or 7")
      } else {
        println(s"$n is not a multiple of 3 or 7")
      }
    } else {
      println("Enter a positive number")
    }
  }
}
```

RESULT

```
Enter a positive number:
21
21 is a multiple of 3 or 7
```

2. Write a Scala program to find sum of square of the given list.

```scala
import scala.io.StdIn

object SumOfSquares extends App {
  def sumOfSquares(numbers: List[Int]): Int = {
    var sum = 0
    for (n <- numbers) {
      sum += n * n
    }
    sum
  }

  println("Enter numbers separated by spaces:")
  val inputString: String = scala.io.StdIn.readLine()

  val numbers: List[Int] = inputString.split("\\s+").map(_.toInt).toList

  if (numbers.nonEmpty) {
    val result: Int = sumOfSquares(numbers)
    println(s"The sum of squares is: $result")
  } else {
    println("Invalid input. Please enter valid integer numbers.")
  }
}
```

Stdin Inputs

1 2 3 4 5

Interactive

CommandLine Arguments

Execute

RESULT

```
Enter numbers separated by spaces:
The sum of squares is: 55
```

3. Write a Scala program to calculate the total cost for a customer who is buying 10 Glazed donuts. You can assume that the price of each Glazed donut item is at $2.50.

```scala
object DonutCostCalculator extends App {
  val glazedDonutPrice: Double = 2.50
  val numberOfDonuts: Int = 10

  val totalCost: Double = glazedDonutPrice * numberOfDonuts
  println(s"Total cost for $numberOfDonuts Glazed donuts: $${totalCost.formatted("%.2f")}")
}
```

```
Total cost for 10 Glazed donuts: $25.00

warning: 1 deprecation (since 2.12.16); re-run
```

4. Write a Scala program to compute the sum of the two given integer values. If the two values are the same, then return triples their sum.

```scala
import scala.io.StdIn

object SumOrTripleSum extends App {

  println("Enter the first integer:")
  val a: Int = StdIn.readInt()
  println("Enter the second integer:")
  val b: Int = StdIn.readInt()

  if (a == b) {
     println(3 * (a + b))
   } else {
     print(a + b)
   }

}
```

RESULT

```
Enter the first integer:
2
Enter the second integer:
8
10
```

```
Enter the first integer:
10
Enter the second integer:
10
60
```

5. Write a recursive function to get the nth Fibonacci number. The first two Fibonacci numbers are 0 and 1. The nth number is always the sum of the previous two—the sequence begins 0, 1, 1, 2, 3, 5.
   def fib (n: Int): Int

```scala
object Fibonacci extends App {
  def fib(n: Int): Int = {
    if (n <= 1) {
      n
    } else {
      fib(n - 1) + fib(n - 2)
    }
  }

  val n: Int = scala.io.StdIn.readInt()
  val result: Int = fib(n)

  println(s"The $n-th Fibonacci number is: $result")
}
```

RESULT

```
4
The 4-th Fibonacci number is: 3
```

6. Write a function to find the values of following series: -Value=$a+a^2/2! +a^3/3! +a^4/4!.....a^n/n!$

```scala
object SeriesCalculator extends App {
  def calculateSeries(a: Double, n: Int): Double = {
    def factorial(num: Int): Double = {
      if (num <= 1) 1.0
      else num * factorial(num - 1)
    }
    (1 to n).map(i => Math.pow(a, i) / factorial(i)).sum
  }

  val a: Double = 2.0
  val n: Int = 5
  val result: Double = calculateSeries(a, n)

  println(s"The sum of the series is: $result")
}
```

RESULT

```
The sum of the series is: 6.266666666666667
```

7. Write a function to find multiplication of first 10 no's using function with variable length parameters

Experiment No 10

```scala
object MultiplicationCalculator extends App {
  def multiplyNumbers(numbers: Int*): Long = {
    val first10Numbers = numbers.take(10)

    first10Numbers.reduceLeft(_ * _)
  }
  val result: Long = multiplyNumbers(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
  println(s"The multiplication of the first 10 numbers is: $result")
}

```

RESULT

```
The multiplication of the first 10 numbers is: 3628800
```