Experiment-10

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) = {
```

```scala
        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.

```scala
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.

```scala
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

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

```scala
      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.

```scala
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 call
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

```scala
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

```scala
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

```scala
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

```scala
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

```scala
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

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

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.

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.

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

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

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

Q1)

```scala
object CheckMultiples {
  def isMultipleOf3Or7(number: Int): Boolean = {
    (number % 3 == 0) || (number % 7 == 0)
  }

  def checkNumber(number: Int): Unit = {
    if (isMultipleOf3Or7(number)) {
      println(s"$number is a multiple of 3 or 7.")
    } else {
      println(s"$number is not a multiple of 3 or 7.")
    }
  }

  def main(args: Array[String]): Unit = {
    val exampleNumber = 21
    checkNumber(exampleNumber)
  }
}
```

```
21 is a multiple of 3 or 7.
```

Q2)

```scala
object SumOfSquares {
  def main(args: Array[String]): Unit = {
    val numbers = List(1, 2, 3, 4, 5)
    val sumOfSquares = numbers.map(x => x * x).sum
    println(s"Sum of squares: $sumOfSquares")
  }
}
```

```
Sum of squares: 55
```

Q3)

```scala
object DonutCostCalculator {
  def main(args: Array[String]): Unit = {
    val glazedDonutPrice = 2.50
    val numberOfDonuts = 10
    val totalCost = glazedDonutPrice * numberOfDonuts

    println(s"Total cost for $numberOfDonuts Glazed donuts: $$${"%.2f".format(totalCost)}")
  }
}
```

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

Q4)

```scala
 1  object SumCalculator {
 2    def main(args: Array[String]): Unit = {
 3      val num1 = 5
 4      val num2 = 5
 5
 6      val result = if (num1 == num2) {
 7        3 * (num1 + num2)
 8      } else {
 9        num1 + num2
10      }
11
12      println(s"The result is: $result")
13    }
14  }
```

```
The result is: 30
```

Q5)

```scala
 1  object Fibonacci {
 2    def fib(n: Int): Int = {
 3      if (n <= 1) {
 4        n
 5      } else {
 6        fib(n - 1) + fib(n - 2)
 7      }
 8    }
 9
10    def main(args: Array[String]): Unit = {
11      val n = 6
12      val result = fib(n)
13      println(s"The $n-th Fibonacci number is: $result")
14    }
15  }
```

```
The 6-th Fibonacci number is: 8
```

Q6)

```scala
1  object SeriesCalculator {
2    def calculateSeries(a: Double, n: Int): Double = {
3      def factorial(num: Int): Int = {
4        if (num <= 1) 1
5        else num * factorial(num - 1)
6      }
7
8      def termValue(term: Int): Double = {
9        math.pow(a, term) / factorial(term)
10     }
11
12     (1 to n).map(termValue).sum
13   }
14
15   def main(args: Array[String]): Unit = {
16     val a = 2.0
17     val n = 5
18
19     val result = calculateSeries(a, n)
20     println(s"The value of the series is: $result")
21   }
22 }
```

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

Q7)

```scala
1  object MultiplicationCalculator {
2    def multiplyNumbers(numbers: Int*): BigInt = {
3      if (numbers.isEmpty) 0
4      else numbers.foldLeft(BigInt(1))(_ * _)
5    }
6
7    def main(args: Array[String]): Unit = {
8      val result = multiplyNumbers(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
9      println(s"The multiplication of the first 10 numbers is: $result")
10   }
11 }
```

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