Scala Functions

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

}
```

In the above syntax, = (equal) operator is looking strange but don't worry scala has defined it as:

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 functions don?t use return statement. Return type infers by compiler from the last expression or statement present in the function.

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")
  }
}
```

Output:

```
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
  }
}
```

Output:

```
10
```

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)
  }
}
```

Output:

```
30
```

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)
  }
}
```

Output:

30

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.

Let's see an example.

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
  }
}
```

Output:

17

15

0

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.

Let's see an example.

```
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

      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

   }

}
```

Output:

```
17

17

17
```