Scala Programming

Scala provides best performance with Apache Spark. Any new features release in Spark happens in scala first.

val vs var

val - like a constant, re-assignment is not possible, cannot change the value.

var - variable, which allows re-assignment.

Type Inference

Scala compiler will infer the types of the variables when data type is not specified.

Data Types

```
val numberOne : Int = 5

val boo : Boolean = true

val d : Char = 'a'

val doublePi : Double = 3.1415

val floatPi : Float = 3.1415f  //f at the end for float

val e : Long = 123434534534534l   //l at the end for long number

val smallNumber : Byte = 127
```

Interpolations

S - Interpolation

Using s"$var" for interpolation

```
val name: String = "Riz"
println(s"Hello $name how are you?")
```

F - Interpolation

```
println(f"value of pi is $doublePi%.3f")
```

Raw - Interpolation

```
println(raw"hello how \n are you")
```

String Comparision

In Java, to comapre 2 strings we have to use equals method, and == is used for reference comparision.

But in case of Scala, == can be used for string comparition.

```
val x: String  = "sumit"
val y: String = "sumit"

val z: Boolean = x == y  //true in scala and false in Java.
```

Conditional Statements

```
if (1 > 3){             //false
    println("Hello")
}else{
    println("there")
}
```

Match Case Statement (Switch in Java)

```
val num = 1

num match {
    case 1 => println("One")
    case 2 => println("two")
    case 3 => println("three")
    case _ => println("something else")
}
```

Loops (for, while)

```
// for loop
for (x <- 1 to 10) {
    val squared = x*x
    println(squared)
}

//while loop
var i = 0
while (i <= 10) {
```

```
    println(i)
    i = i+1
}

//do-while loop => at least executes once guarantee
do{
    println(i)
    i = i+1
} while (i <= 10) {
    println(i)
}
```

Blocks of code

if statements are written in a same line, semicolons are required. The Last statement in the expression block is our return statement.

```
{
    val x = 10
    x + 20
    10
}  //return of this block is 10
```

Functional Programming using Scala

Defining a function

```
//Square Function
def squareIt(x: Int): Int = {
    x*x     // curly braces are optional for one line code block.
}

println(squareIt(4))

// Cube Function
def cubeIt(x:Int) = x*x*x

println(cubeIt(2))

//Transform Function
def transformInt(x:Int, f:Int => Int): Int = {
    f(x)
}

println(transformInt(2, cubeIt))

transformInt(2, x => x*x*x) //CubeIt function Using anonymous function

// Divide Function
```

```
def divideByTwo(x: Int) = {
    x/2
}
divideByTwo(4)

// Same function using transformInt
transformInt(4, x => x/2)

//Another example of func prog
transformInt(2, x => {val y = x*2; y*y})
```

Scala Collections

- Array

Array can be referenced by Index and it is 0 based.
Arrays are mutable collection.
Searching the array based on index is very fast.
Adding a new element is tricky and is inefficient operation

```
val a = Array(1,2,3,4,5)
println(a.mkString(","))    // o/p => "1,2,3,4,5"

//Iterating through the array
for (i <- a) println(a)

//Mutating the array
a(2) = 7
println(a.mkString(","))    // "1,2,7,4,5"
```

- List

Indexing starts from 0 like array.
Internally, list holds the elements in a single linked list.
Searching the list is not efficient
Adding a new element, especially at the starting is efficient.

```
val l = List(1,2,3,4,5)
println(l.head)     //> 1
println(l.tail)     //> 2,3,4,5

for (i <- l) println(i)     //> l: List[Int] = List(1, 2, 3, 4, 5)
l.reverse       //> List[Int] = List(5, 4, 3, 2, 1)

10 :: l     // Adds the element at the begining of the list. List[Int] = List(10,
1, 2, 3, 4, 5)
```

- Tuple

Indexing starts from 1.
Holds elements of different data types.
You can treat a tuple like a record in your donorbase table.

```
val t = ("Riyaz", 2500000, 29, true)
println(t._1)
println(t._2)

val y = (107, "sumit")      //Tuple of 2 elements, this can be treated as key-value pair

val z = 107 -> "sumit"      //Another way of writing a 2 element tuple, treated as key-value pair.
```

- Range

```
val rng = 1 to 10
for(i <- rng) println(i)

val rng2 = 1 until 10   //In Until, end range is not inclusive.
for(i <- rng2) println(i)
```

- Set

A set holds only unique values. It cannot hold duplicates.

```
val set1 = Set(1,1,1,1,3,3,3,3,4,4,4,45,5,5,5,2)
println(set1)       //>Set(5, 1, 2, 45, 3, 4)
(set1.min, set1.max, set1.sum)      //(Int, Int, Int) = (1,45,60)
```

- Map

A collection of key-value pairs like dictionary.
Keys cannot be repeated, latest key-value is displayed, first one is discarded.

```
val map1 = Map(1 -> "Sumit", 2 -> "Riyaz", 3 -> "Sushant")
// scala.collection.immutable.Map[Int,String] = Map(1 -> Sumit, 2 -> Riyaz, 3 -> Sushant)

map1.get(1)     //Option[String] = Some(Sumit)
```