The Scala Programming

Bite-sized introductions to the most frequently used features of Scala.

Agenda

- Introduction
- Getting Started
- Concepts with Hands-on

What is Scala?

- Modern multi-paradigm programming language. (2003)
  - Imperative
  - Object Oriented
  - Functional
- Concise
- Elegant and
- Type-safe

Prog_1

Scala is object-oriented

- Pure OO
- Every value is an object
- Types and behaviour of objects -> Classes & Traits

Prog_2

Scala is functional

- Every function is a value.
- Supports
  - higher-order functions
  - Nested functions
  - Currying

- Functional Programs follow two main principles
  - Referential Transparency for functions.
  - Values have immutable state.

Prog_3

Scala is statically typed

- Scala type system supports:
  - generic classes
  - inner classes
  - abstract type members as object members
  - compound types
  - explicitly typed self references
  - implicit parameters and conversions
  - polymorphic methods

Scala is extensible

- New language constructs can be added easily in form of libraries.
- Can be done without using meta-programming facilities, like -
  - Implicit classes
  - String interpolation

Scala interoperates

- Interoperates with JRE
- Smooth interaction with JAVA

Prog_4

Getting Started

- How to start with Scala ?
  - Outside Morgan Env -
    - Quickest way - https://scalafiddle.io/
    - https://www.scala-lang.org/ -> Download
  - In Morgan Env -
    - In Unix Env - /ms/dist/ossjava/PROJ/scala/
    - Eclipse/IntelliJ IDE (with Scala Plugin).

Baby Steps

- Entry point
  - Two approaches for main method.
    - Extend App
    - Define Main method in an Object.
- Find Java signatures for Scala Compiled Classes.
  - javap

Variables

- //Variable Declaration :
- **val a: Int = 50** **//> a : Int = 50**

- //is same as
- **val a1 = 50** **//> a1 : Int = 50**

- //Compile Error : Reassignment to Val
- //a=70

Variables

- **var b: Int = 50 //> b : Int = 50**

- **val msg: java.lang.String = "Hello" //> msg : String = Hello**
- //Is same as
- **val msg1: String = "Hello" //> msg1 : String = Hello**
- //Is same as
- **val msg2: String = "Hello" //> msg2 : String = Hello**

Variables ...Contd

- //Every Value is an object :
- a - 5 //> res0: Int = 45
- //Is Same as
- a.-(5) //> res1: Int = 45

- println(5) //> 5
- //Is same as
- print(5 + "\n") //> 5
- //Is same as
- println({
- **val a = 5**
- a
- }) //> 5

Variables ...Contd

```
val multiLine =
  "This is the next line."                    //> multiLine  : String = This is the next line.

 //Compiler Error
 //val multiLine2 = "
 //        This is the next line."

 val multiLine2 = """
  This is a multiline String.
  Starts and ends at different lines.
  """                                    //> multiLine2  : String = "
                                        //|    This is a multiline String.
                                        //|    Starts and ends at different lines.
                                        //|    "
 println(multiLine2)                          //>
                                        //|    This is a multiline String.
                                        //|    Starts and ends at different lines.
                                        //|
 val multiLine3 = """
  |This is a multiline String.
  |Starts and ends at different lines.
  """.stripMargin('|')                        //> multiLine3  : String = "
                                        //| This is a multiline String.
                                        //| Starts and ends at different lines.
                                        //|    "

 println(multiLine3)                          //>
                                        //| This is a multiline String.
                                        //| Starts and ends at different lines.
                                        //|
```

Methods / Functions

```
//Method Declaration
def addOne(x: Int): Int = {
  return (x + 1)
}                                        //> addOne: (x: Int)Int
//Is same as
def addOne_v2(x: Int) = x + 1            //> addOne_v2: (x: Int)Int

//Calling Method
addOne(50)                               //> res2: Int = 51
addOne_v2(50)                              //> res3: Int = 51

//Method which does not take any argument neither return any.
def greet(): Unit = println("Hello, world!")    //> greet: ()Unit
//Is same as
def greet_v2() = println("Hello, world!")       //> greet_v2: ()Unit

greet()                                  //> Hello, world!
greet_v2()                                 //> Hello, world!
```

Scripts

```
//Run as :
$ scala a.scala
Hello, world, from a script!
```

File : a.scala

```
println("Hello, world, from a script!")
```

Scripts ... contd

```
$ scala echoargs.scala Friends
Hello, Friends!
```

File : echoargs.scala

```scala
println("Hello, " + args(0) + "!")
```

Looping

```
$ scala echoargs1.scala Friends
Planet Arguments

Friends
Planet
Arguments
```

File : echoargs1.scala

```
var i = 0
  while (i < args.length) {
    println(args(i))
    i += 1
  }
```

Looping ...contd

```
for (i <- args) {
     println(i)
}
//Is same as
 {
   def itemHandler(s: String):
Unit = { println(s) }
   args.foreach(item =>
itemHandler(item))
  }
 //Is same as
  {
   def itemHandler(s: String):
Unit = { println(s) }
   args.foreach(itemHandler(_))
  }
```

```
//Is same as
  {
   def itemHandler(s: String):
Unit = { println(s) }
   args.foreach(itemHandler)
  }
 //Is same as
  {
   args.foreach(item =>
println(item))
  }
 //Is same as
  {
   args.foreach(println)
  }
```

Looping ...contd

```
//All Even numbers
 for (i <- 1 to 6) if (i % 2 == 0) {
   println(i)
 }                              //> 2
                                //| 4
                                //| 6
 //Is same as
 for (i <- 1 to 6 if (i % 2 == 0)) {
   println(i)                      //> 2
                                //| 4
                                //| 6
 }
 //Is Same as
 for (i <- 1 to 6) {
   if (i % 2 == 0)
     println(i)                    //> 2
                                //| 4
                                //| 6
 }

 //For yield
 val a1 = for (i <- 1 to 6 if (i % 2 == 0)) yield (i)
                                  //> a1  :
scala.collection.immutable.IndexedSeq[Int] = Vector(2,
4, 6)
 println(a1)                      //> Vector(2, 4, 6)
 //Is same as
 val a2 = for (i <- 1 to 6) yield { if (i % 2 == 0) i }
                                  //> a2  :
scala.collection.immutable.IndexedSeq[AnyVal] =
Vector((), 2, (), 4, (
                                  //| ), 6)
 println(a2)                      //> Vector((), 2, (), 4, (),
6)
```

Looping ...contd

```
//Nested loops
scala>   for (i <- 1 to 3; j <- 4 to 5)
     | println(s"i=$i, j=$j")
i=1, j=4
i=1, j=5
i=2, j=4
i=2, j=5
i=3, j=4
i=3, j=5
```

Conditions

```
 //Conditions

var isEven = false                              //> isEven  : Boolean = false
  if (a % 2 == 0)
    isEven = true
  println(isEven)                               //> true
  //Is Same as

  val isEven_v2 =
    if (a % 2 == 0)
      true
    else
      false                                 //> isEven_v2  : Boolean = true

  println(isEven_v2)                               //> true
```

Currying & Partially Applied functions

```
def n_divides_m(n: Int)(m: Int): Boolean = n % m == 0
n_divides_m(4)(2)

 def is_even(n: Int) = n_divides_m(n)(2)

 is_even(5)
 is_even(6)

 def is_odd(n: Int) = !n_divides_m(n)(2)

 is_odd(5)
 is_odd(6)
```

Tail Recursion

```
def factorial(n: Int): Int = if (n < 1) 1 else n * factorial(n - 1)

factorial(30)

def factorial_v2(n: Int): BigInt = if (n < 1) BigInt(1) else (n * factorial_v2(n - 1))

//factorial_v2(50000)

def factorial_v3(n: Int): BigInt = {

  @tailrec
  def factorial_v4(acc: BigInt, n: Int): BigInt = if (n < 1) acc else (factorial_v4(acc * n, n - 1))

  factorial_v4(1, n)
}

println(factorial_v3(50000).toString.length)
println(factorial_v3(50000).toString.substring(0, 500))


val r = factorial_v3(50000) / factorial_v3(49999)                    //> r  : scala.math.BigInt = 50000
```

Arrays (Mutable)

```
val greetStrings = new Array[String](3)        //> greetStrings  : Array[String] = Array(null, null, null)
 greetStrings(0) = "Hello"
 greetStrings                                  //> res10: Array[String] = Array(Hello, null, null)
 //Same as
 greetStrings.update(0, "Hello")
 greetStrings                                  //> res11: Array[String] = Array(Hello, null, null)

 greetStrings(0)                                //> res12: String = Hello
 //Is same as
 greetStrings.apply(0)                            //> res13: String = Hello

val numNames2 = Array.apply("zero", "one", "two")
```

Lists (immutable)

```
val oneTwoThree = List(1, 2, 3)              //> oneTwoThree  : List[Int] = List(1, 2, 3)

 val oneTwo = List(1, 2)                      //> oneTwo  : List[Int] = List(1, 2)
 val threeFour = List(3, 4)                   //> threeFour  : List[Int] = List(3, 4)
 val oneTwoThreeFour = oneTwo ::: threeFour      //> oneTwoThreeFour  : List[Int] = List(1, 2, 3, 4)

 println(oneTwo + " and " + threeFour + " were not mutated.")
                                   //> List(1, 2) and List(3, 4) were not mutated.
 println("Thus, " + oneTwoThreeFour + " is a new list.")
                                   //> Thus, List(1, 2, 3, 4) is a new list.
 val twoThree = List(2, 3)                    //> twoThree  : List[Int] = List(2, 3)
 //Cons operator
 val oneTwoThree_v2 = 1 :: twoThree           //> oneTwoThree_v2  : List[Int] = List(1, 2, 3)
 println(oneTwoThree_v2)                      //> List(1, 2, 3)
```

Lists ...Contd.

What it is	**What it does**
`List()` or `Nil`	The empty `List`
`List("Cool", "tools", "rule")`	Creates a new `List[String]` with the three values `"Cool"`, `"tools"`, and `"rule"`
`val thrill = "Will" :: "fill" :: "until" :: Nil`	Creates a new `List[String]` with the three values `"Will"`, `"fill"`, and `"until"`
`List("a", "b") ::: List("c", "d")`	Concatenates two lists (returns a new `List[String]` with values `"a"`, `"b"`, `"c"`, and `"d"`)
`thrill(2)`	Returns the element at index 2 (zero based) of the `thrill` list (returns `"until"`)
`thrill.count(s => s.length == 4)`	Counts the number of string elements in `thrill` that have length 4 (returns 2)
`thrill.drop(2)`	Returns the `thrill` list without its first 2 elements (returns `List("until")`)

Lists …Contd.

What it is	What it does
`thrill.dropRight(2)`	Returns the `thrill` list without its rightmost 2 elements (returns `List("Will")`)
`thrill.exists(s => s == "until")`	Determines whether a string element exists in `thrill` that has the value `"until"` (returns `true`)
`thrill.filter(s => s.length == 4)`	Returns a list of all elements, in order, of the `thrill` list that have length 4 (returns `List("Will", "fill")`)
`thrill.forall(s => s.endsWith("l"))`	Indicates whether all elements in the `thrill` list end with the letter `"l"` (returns `true`)
`thrill.foreach(s => print(s))`	Executes the `print` statement on each of the strings in the `thrill` list (prints `"Willfilluntil"`)
`thrill.foreach(print)`	Same as the previous, but more concise (also prints `"Willfilluntil"`)

Lists ...Contd.

What it is	What it does
`thrill.head`	Returns the first element in the `thrill` list (returns `"Will"`)
`thrill.init`	Returns a list of all but the last element in the `thrill` list (returns `List("Will", "fill")`)
`thrill.isEmpty`	Indicates whether the `thrill` list is empty (returns `false`)
`thrill.last`	Returns the last element in the `thrill` list (returns `"until"`)
`thrill.length`	Returns the number of elements in the `thrill` list (returns 3)
`thrill.map(s => s + "y")`	Returns a list resulting from adding a `"y"` to each string element in the `thrill` list (returns `List("Willy", "filly", "u ntily")`)
`thrill.mkString(", ")`	Makes a string with the elements of the list (returns `"Will, fill, until"`)

Lists ...Contd.

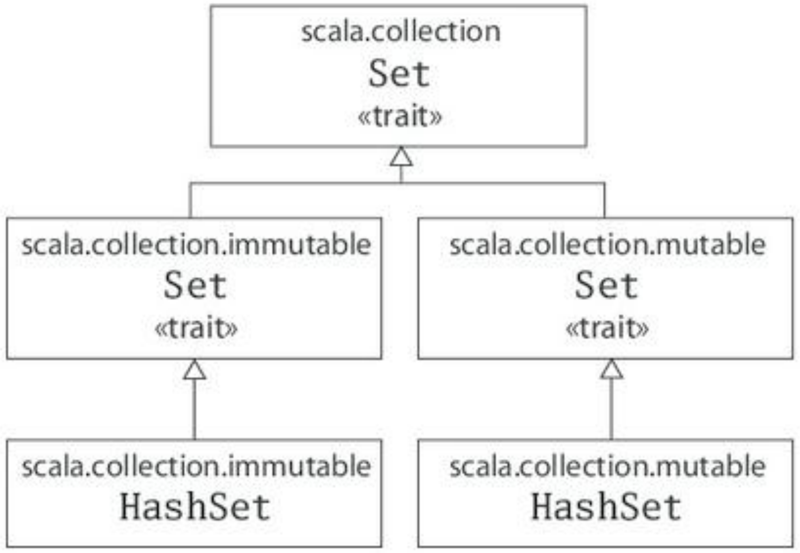
What it is	**What it does**
`thrill.filterNot(s => s.length == 4)`	Returns a list of all elements, in order, of the `thrill` list *except those*that have length 4 (returns `List("until")`)
`thrill.reverse`	Returns a list containing all elements of the `thrill` list in reverse order (returns `List("until", "fill", "Will")`)
`thrill.sort((s, t) => s.charAt(0).toLower < t.charAt(0).toLower)`	Returns a list containing all elements of the `thrill` list in alphabetical order of the first character lowercased (returns `List("fill", "until", "Will")`)
`thrill.tail`	Returns the `thrill` list minus its first element (returns `List("fill", "until")`)

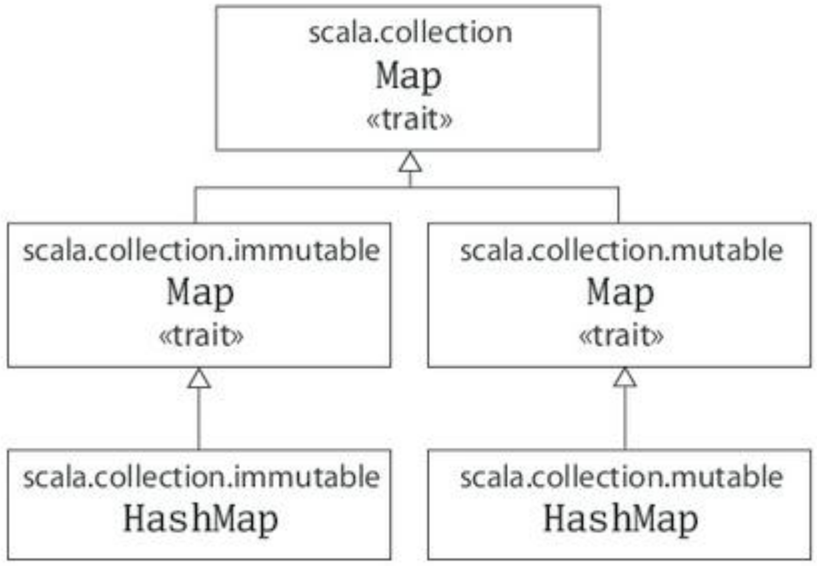
Tuples (immutable)

```
//Can contain heterogeneous data types.

 val pair = (99, "Luftballons")              //> pair  : (Int, String) = (99,Luftballons)
  println(pair._1)                         //> 99
  println(pair._2)                         //> Luftballons
```

Sets and Maps (Mutable as well immutable)





Sets and Maps

```
 var jetSet = Set("Boeing", "Airbus")
//> jetSet  : scala.collection.immutable.Set[String] = Set(Boeing, Airbus)

   jetSet += "Lear"

   jetSet
//> res0: scala.collection.immutable.Set[String] = Set(Boeing, Airbus, Lear)

   println(jetSet.contains("Cessna"))          //> false

 import scala.collection.mutable

  val movieSet = mutable.Set("DeadPool", "Transformer")
                                              //> movieSet  : scala.collection.mutable.Set[String] = Set(DeadPool, Transformer
                                              //| )
  movieSet += "Avenger"                              //> res1: practise.Practise2.movieSet.type = Set(Avenger, DeadPool,
Transformer)
                                              //|
  println(movieSet)                         //> Set(Avenger, DeadPool, Transformer)
```

Sets and Maps

```
import scala.collection.immutable.HashSet

 val hashSet = HashSet("Tomatoes", "Chilies")      //Factory Method from companion object.
 println(hashSet + "Coriander")
```

Sets and Maps

```
 import scala.collection.mutable

  val numberMap = mutable.Map[Int, String]()
//> numberMap  : scala.collection.mutable.Map[Int,String] = Map()
  numberMap += (1 -> "One")
//> res2: practise.Practise2.numberMap.type = Map(1 -> One)
  numberMap += (2 -> "Two")
//> res3: practise.Practise2.numberMap.type = Map(2 -> Two, 1 -> One)
  numberMap += (3 -> "Three")
//> res4: practise.Practise2.numberMap.type = Map(2 -> Two, 1 -> One, 3 -> Three
//| )
  println(numberMap(2))
//> Two
```