Scala

Session: 2

Invention of Scala :

Why Scala or a new language was invented?

Scala wanted to do build one step further on Java.

Invention of Scala :

It was developed keeping below points in mind, they are:

- Retain platform independence :

  It wanted to retain the platform independency, JVM dependency and automatic memory management.

- Java's Verbosity:

  We have to get rid of Java's verbosity.

Invention of Scala :

- Object oriented paradigm:

To get rid of object oriented paradigm. As it cannot be used to model everything.

Ex: it is very hard to capture straight in an object because we are moving in concurrency and parallelism, when two threads are trying to access same variable you will get all sorts of bugs, concurrency issues, so they tried to introduce another programming paradigm called functional programming into the Scala arsenal. They doŶ't force prograŵŵer to thiŶk everythiŶg iŶ terŵs of objects.

Invention of Scala :

•JVM:

As Scala is based on JVM whatever the libraries, APS, interfaces written in Java you can perfectly execute them in Scala programme. For that include Jar in class path, you can call all Java functions, classes interfaces in Scala.

In other words code written in Java is interoperable to the code written in Scala. This is the huge advantage that Scala has.

Scala: Features

What are the features I get as a developer?

Scala: Features

- Most of the developer's time is spent in maintaining, reading and testing code then time for writing, you will spend only 20-30% time in writing code, maintaining and reading and testing will take most of your time in software development life cycle.

- You have to optimize for that 70% of your time, so any language which is easy to read code easy test and maintain is a better choice. The Scala code is more readable than any language.

Scala test case: Example

Let us see an example for Scala test case:

Just Released - ScalaTest and Scalactic 2.2.1!

```
import collection.mutable.Stack
import org.scalatest._

class ExampleSpec extends FlatSpec with Matchers {

  "A Stack" should "pop values in last-in-first-out order" in {
    val stack = new Stack[Int]
    stack.push(1)
    stack.push(2)
    stack.pop() should be (2)
    stack.pop() should be (1)
  }

  it should "throw NoSuchElementException if an empty stack is popped" in {
    val emptyStack = new Stack[Int]
    a [NoSuchElementException] should be thrownBy {
      emptyStack.pop()
    }
  }
}
```

- It reads like normal plain English, test cases for Stack and specification then create test stack which will push and then pop which should be "1".
- You can even add your own test case. In this image there is a test case like "throw NoSuch element Exception if an empty stack is popped" they create an empty stack for this and try to pop an expect a NoSuch element exception here, when you try to run this test case it prints the formatted way what has been done.
- It prints the specifications also.

Scala: Importance

- To summarise the importance of Scala to a developer are:
- It reduces the time in you spend in maintaining, designing and developing the code.
- It makes your code more readable.
- It is easier to test your code.
- Most of the projects are going towards the Agile model, in which you release software as frequently as possible you have to test then and there. So you have to write a lot of unit test cases for whatever code you write.
- Whatever code you write it must be followed by a test case, so it is as good as failed code.

Compiler and Interpreter file:

What is the difference between compiler and interpreter file?

Compiler and Interpreter file:

What are the advantages of interpreted languages?
It is easier and rapid to prototype.

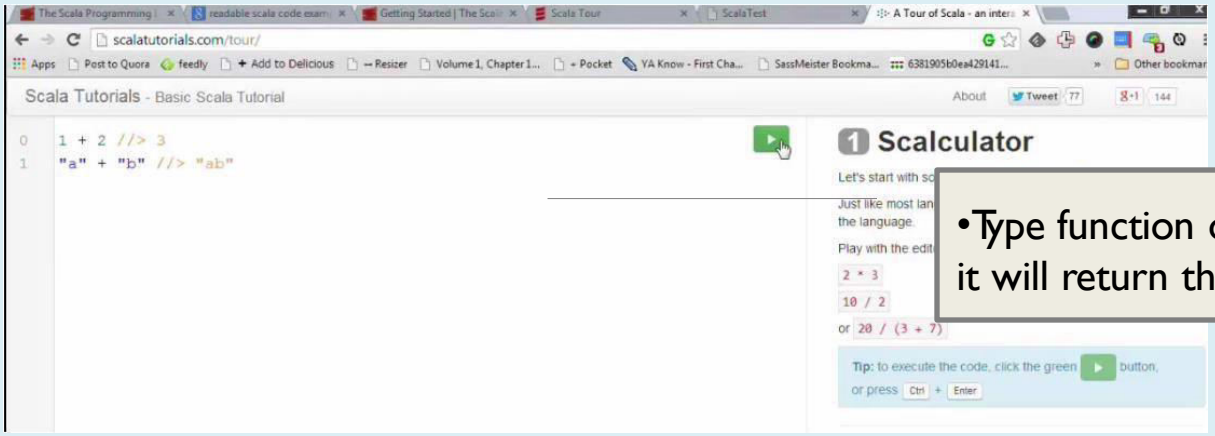
•If you take the equivalent, I can write the variable cases in Python in 60 hrs, for the same complexity it will take one week in Java. As the time taken to compile, pack into Jar and deploy it will add up.

•Scala has got both compiler and interpreter modes. 'Scalac" is a compiler and scala is an executable which runs the interpreted. Scala has got REPL version, which means "read evaluate print look".

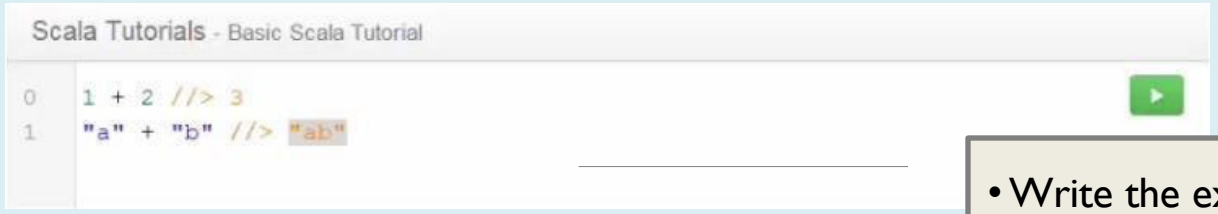
•Type function or expression in REPL, it will return the expression.

Scala test case: Example

See the below image:



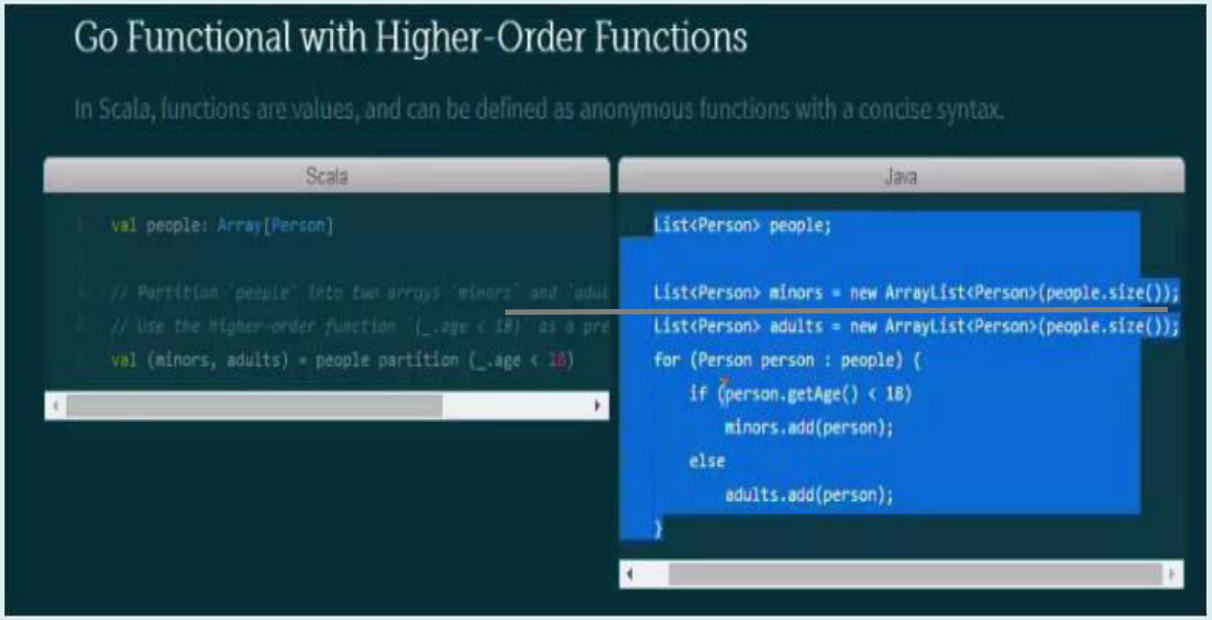
- Type function or expression in REPL, it will return the expression.



- Write the expression and click the play button, you can get the result.

Verbosity of Java: Example

See the below image:



Under Scala, there is code for splitting down an array on specific criteria into two lists, which we cannot see in Java, because I have to wrap in a class and wrap inside the system. Main functions and then run it.

About Scala:

Scala is
- oo
- Functional
- Strongly typed
- Runs on JVM
- Compatible with Java libraries

Scala: type system

What is a type system?

- It is an abstraction of whatever data you represent.
- Ex: you have a variable "i" for number, it can be any number, and type system is something which gives meaning to it. At compile time if you have strong type system it can do the validation while compiling and you can get rid of errors.

Type system classifications: Scala

There are two possibilities in static type:
•Strongly and weakly type.
Ex: In python, if you have string and do the "int" of that string, it will covert that integer. This is an example to dynamically weekly typed language.
•Scala is a strongly type static file system.
Strongly typed is a good option as it avoids error because it will not compile at the first place.
Scala has got the sternest typed system on the JVM platform.
•Scala by its automatic type inference, in which the Scala compiler automatically infers the type of system based on the context.

Language Features: Scala

Language features of Scala are:

- Type Inference
- Higher order functions
- Option Types
- Pattern Matching
- Collections.

Language Features: Scala

Data Type	Description
Byte	8 bit signed value. Range from -128 to 127
Short	16 bit signed value. Range -32768 to 32767
Int	32 bit signed value. Range -2147483648 to 2147483647
Long	64 bit signed value. -9223372036854775808 to 9223372036854775807
Float	32 bit IEEE 754 single-precision float
Double	64 bit IEEE 754 double-precision float
Char	16 bit unsigned Unicode character. Range from U+0000 to U+FFFF
String	A sequence of Chars
Boolean	Either the literal true or the literal false
Unit	Corresponds to no value

Variable Declaration

Variable Declaration
Scala has the different syntax for the declaration of variables and they can be defined as value, i.e., constant or a variable. Following is the syntax to define a variable using **var** keyword:

var myVar:String=“Spark“

Here, myVar is declared using the keyword var. This means that it is a variable that can change value and this is called mutable variable. Following is the syntax to define a variable using **val** keyword:

val myVal:String=“Spark"

Scala : Type Inference

What is a Type Inference?

•Type inference: we can infer the type of the variable based on the context, need not be explicitly declared it, which saves the lot of typing work.

1. saptak@sandbox:~ (ssh)

```
Spark context available as sc.

scala> var c = 9
c: Int = 9

scala> println(c.getClass)
int

scala>

scala> val d = 9.9
d: Double = 9.9

scala> println(d.getClass)
double

scala>

scala> val e = "Hello"
e: String = Hello

scala> println(e.getClass)
class java.lang.String

scala>
```

Language Features: Scala

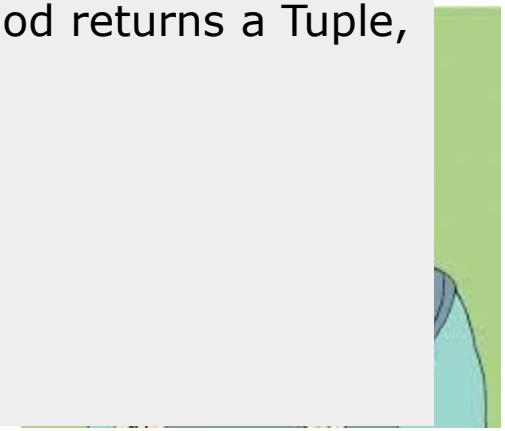
Multiple assignments:

Scala supports multiple assignments. If a code block or method returns a Tuple, the Tuple can be assigned to a val variable.

```
val(myVar1:Int, myVar2:String)=Pair(40,"Foo")
```

And the type inferencer gets it right:

```
val(myVar1, myVar2)=Pair(40,"Foo")
```



Language Features: Scala

The if...else Statement:

An if statement can be followed by an optional *else* statement, which executes when the Boolean expression is false.

Syntax:

The syntax of a if...else is:

```
if(Boolean_expression){
//Executes when the Boolean expression is true
}else{
//Executes when the Boolean expression is false
}

var x =30;

if( x <20){
println("This is if statement");
}else{
println("This is else statement");
}
```

Language Features: Scala

The if...else if...else Statement:

Syntax:

The syntax of a if...else if...else is:

```
if(Boolean_expression1){
//Executes when the Boolean expression 1 is true
}elseif(Boolean_expression2){
//Executes when the Boolean expression 2 is true
}elseif(Boolean_expression3){
//Executes when the Boolean expression 3 is true
}else{
//Executes when the none of the above condition is true.
}
```

Creating Strings:

The most direct way to create a string is to write:

```
var greeting ="Hello world!";
Or
var greeting:String="Hello world!";
```

String class is immutable, so that once it is created a String object cannot be changed. If there is a necessity to make a lot of modifications to Strings of characters then you should use String Builder

Creating Strings:

```
val S = "CS109 is nice"
S.contains("ice")
S.indexOf("ice")
S.indexOf("rain")
S.replace('i', '#')
S.split("\\s+")
S.toLowerCase
S.toUpperCase
S.substring(5)
S.substring(5,8)
S.reverse
```

Creating Format Strings:

```
var floatVar=12.456
var intVar=2000
var stringVar="Hello, Scala!"
var fs =printf("The value of the float variable is
"+
"%f, while the value of the integer "+
"variable is %d, and the string "+
"is %s",floatVar,intVar,stringVar)
println(fs)
```

StringBuilder

. In a string append, a new string must be created. A string cannot be changed. But with StringBuilder we add data to an internal buffer. This makes things faster.

```
val builder = StringBuilder.newBuilder
builder.append("cat ")
builder.append("bird")

// Convert StringBuilder to a string.
val result = builder.toString()
println(result)
```

StringBuilder

Insert: With insert, we place a string beginning at an index. Later characters are moved to be after the new string.

Replace: This receives a start index, and an end index (not a length). We replace the first char with the arguments 0, 1

```
val builder = StringBuilder.newBuilder
or
val builder = new StringBuilder
// Append initial data.
builder.append("I like cats")
println(builder)
// Insert this string at index 1.
builder.insert(1, " do not")
println(builder)
// Replace first character with a new string.
builder.replace(0, 1, "You")
println(builder)
b1.append("I like spark")
b1.replace(2,6,"love")
```

Declaring Array Variables:

To use an array in a program, you must declare a variable to reference the array and you must specify the type of array the variable can reference. Here is the syntax for declaring an array variable:

```
val fruits = new Array[String](3)
fruits(0) = "Apple"
fruits(1) = "Banana"
fruits(2) = "Orange"
```

Processing Arrays:

```
var myList=Array(1,2,3,4,5,6)

// Print all the array elements
for( x <-myList){
println( x )
}
// Summing all elements
var total =0.0;
for(i<-0 to (myList.length-1)){
total+=myList(i);
}
println("Total is "+ total);

// Finding the largest element
var max =myList(0);
for(i<-1 to (myList.length-1)){
if(myList(i)> max) max =myList(i);
}
println("Max is "+ max);
```

Scala: Higher order functions

What are Higher order functions?

•In C++ can you pass a string as the parameter to the function? Can you have list of strings? Yes, but can u have a list of functions? Can u pass a function as the parameter to another function in Java also?

–No you cannot do.

•Functions are not treated as a variable are treated in C++, Java. Higher order function is when you treat a function just like a variable. This is possible in Scala. This advantages the developer to express complex business logic using higher order functions.

Scala: Higher order functions

Let us see an example:

•For Ex: if you take the builder designed pattern, you need not have so much of code in higher order function, you can directly pass the business logic as the function to your builder object. It will generate an object instance for you. You can achieve it using a single function. Most designed patterns are invented because there are no higher order functions in those programming languages.

Scala: Option types

What is the type of null?

Whenever we write a function for example "if this is present in list, we return that object, if it is not present we return null". What exactly you try to imply when you return null is not found.

Scala: Option types

How your programming language would look if there are no null at all? How do you handle such cases?

Ex: you try to find an employee and that employee is not found I should enter the subsequent blocks of code at all, but if someone forgets to check for null then you get null pointer exceptional.

Scala: Option types

How do you remove null but at same time be safe?

Scala can do this; it has option type which holds the value, when there is a value if there is no value it doesn't hold anything at all which is not null.

•Option type is generic construct which can hold anything.

•If option type"T" is declared for person and I do p=find employee, it returns if P is found and go to next line of code, if p is not found it will not go to the next line of code.I didn't do null check and not equated anything to null.

• Find employee function looks like:

```
findEmployee(list of persons, p) {
  for each(person in persons) {
    if(person is p) {
      return p;
    }
  }
}

optional[Person] p = findEmployee(list of persons, p1)
var h = p
h
```

Scala: Pattern matching, Collection

Pattern matching:
•It makes your code clearer and concise.

Collection:
•Scala has got good collection library then what is there in Java. Scala also supports Generics; it encourages programming generics as generics are all about reusing the code and making it more testable.

Scala: User interference

In scala we declare the variable as below, where variable name is number and int is the type of variable.

•There are no primitive types in Scala, so in type inference you need not have to include type information after the variable name. Like name=joe in codes, the scala compiler automatically figures it is a string, so you need not have type it is a string.

• Type inference is applicable to both static and dynamic types.

Scala: Higher order functions

See the below function:

Type inference

```
scala> class Person(val name: String, val age: Int) {
     |   override def toString = s"$name ($age)"
     | }
defined class Person

scala> def underagePeopleNames(persons: List[Person]) = {
     |   for (person <- persons; if person.age < 18)
     |     yield person.name
     | }
underagePeopleNames: (persons: List[Person])List[String]
```

Here there is declaration with person, class with attributes name and age, why type is declared is type inference work only when you are assigning something to a variable.

•If it is not given the compiler will get confused with what name means? We can know it can only be a string but compiler cannot, compiler can work only when you are assigning something and not declaring it. So here we are defining a class called "person" and "under age people names" , it gives a list of names of persons who age is less than 18, it should return persons names but you have not declared anywhere that you are returning a list of names.

Scala: Higher order functions

See the below function:

Type inference

```
scala> class Person(val name: String, val age: Int) {
     |   override def toString = s"$name ($age)"
     | }
defined class Person

scala> def underagePeopleNames(persons: List[Person]) = {
     |   for (person <- persons; if person.age < 18)
     |     yield person.name
     | }
underagePeopleNames: (persons: List[Person])List[String]
```

•Scala interpreter automatically infers it as the list of strings only. As you yield person.name, it iterates through persons then it should be string only.

Scala: Options

See the below function:

Options

```
def getUserFromDb(id: Int): User = // DB stuff

val userMap = Map("Joe" -> 1, "Bob" -> 2, "Kip" -> 3)

val bobsId: Option[Int] = userMap.get("Bob")
// Some(Int) OR None

val bob: Option[User] = bobsId.map(getUserFromDb)
```

•User Map.get("Bob") what this function tries to do is, ".get" is a function defined inside user map, it is of type Map, it tries to get the ID of person name "Bob".
• It is an optional "int", in scala there is no option of null so we have to mention option [int] it will return an integer or nothing.
• When you build something on an option type there should also be an option type.

Ex: If there is water, fetch water from jug, what if there is no jug at all.
You cannot build a known type on top of option type

Scala: Compile time error:

What is Compile time error?
Compile time error:
If you try to do a user and the compiler will point that you have built on top of option. The return text should also be an option.

Scala: Pattern matching

Pattern matching:

•It is a switch case in steroids. It is rudimentary use case and straight forward. Here option can hold nothing or something, if it holds something then print the age, if it holds nothing print "nothing" as inform.

Scala: Pattern matching

Scala supports recursion.
See the below function:

```
val names = List("joe","adom","smith","Sample")

for (name <- names) {
name match {
case "joe" => println("This is Joe")
case "adom" => println("This is Adom")
case default => println("None of the List")
}
}
```

Scala: Recursion

See the below function:

```
case class User(name:String,age:Int)
val users:List[User] =
User("joe",22)::User("Bob",43)::User("Kip",56)::Nil

def
getAge(name:String,users:List[User]):Option[Int] =
users match {
 case Nil => None
 case User(n,age)::tail if n == name => Some(age)
 case head :: tail => getAge(name,tail)
}

getAge("Bob",users)

res24: Option[Int] = Some(43)
```

There is user object which has name and age attributes, there is a list of users, and you define a function called get age, then we iterate through the whole list and do the pattern matching for each and every item in the list.
We check for edge cases like edge is less than or if it is end of the function etc. We also use the option type here.

Scala: Functions

Functions are defined with the **def** keyword. In Scala, the last expression in the function body is returned without the need of
the return keyword.

Syntax:

```
def functionName ([list of parameters]) : [return type] = {
   function body
   return [expr]
}
```

Scala: **Anonymous Functions**

Anonymous functions can be assigned as a var or val value. It can also be passed to and returned from other functions.

•Example:

```
scala> (x:Int) => x*2
res29: Int => Int = <function1>

scala> res29(5)
res30: Int = 10
```

Scala: **Partial Application**

You can partially apply a function with an underscore, which gives you another function. Scala uses the underscore to mean different things in different contexts, but you can usually think of it as an unnamed magical wildcard

•Example:

```
def adder(m: Int, n: Int) = m + n
val add2 = adder(2, _:Int)
add2(3)

res50: Int = 5
```

Scala: **Curried Functions**

See the below function:

```
def add(x:Int, y:Int) = x + y

add(1, 2)   // 3
add(7, 3)   // 10

def add(x:Int) = (y:Int) => x + y

add(1)(2)   // 3
add(7)(3)   // 10
```

In currying, it has function which accepts specific criteria for filter

Example: In the first sample, the add method takes two parameters and returns the result of adding the two. The second sample redefines the add method so that it takes only a single Int as a parameter and returns a functional (closure) as a result.

Scala: **Closures**

See the below code:

```
scala> val multiplier =(i:Int)=>i*10
multiplier: Int => Int = <function1>

scala> multiplier(50)
res37: Int = 500
```

A **closure** is a function, whose return value depends on the value of one or more variables declared outside this function. Consider the following piece of code with anonymous function.

Scala: **Procedures**

See the below code:

```
def printMe( ) : Unit = {
    println("Hello, Scala!")
  }
```

Scala has a special notation for a function that returns no value. If the function body is enclosed in braces *without a preceding = symbol*, then the return type is Unit. Such a function is called a *procedure*

Thank You

Email us – support@intellipaat.com

Visit us - https://intellipaat.com