Scala Basics

Scala

Apache Spark is the current trend in big data industry.

Spark code actually can be written in 4 different languages:

Scala - gives the best performance with Spark.

also Spark is itself written in Scala.

whenever a new feature release happens its available in Scala first.

Python - there is Python process which interacts with your jvm.

Java - bulky code (lot of lines)

R (for machine learning)

Cloudera Spark-shell (terminal)

Windows or mac (Scala eclipse ide)

in Cloudera the version of Spark is 1.6 and Scala version is 2.10

Scala basics

1. var vs val

int a = 5;
val a: Int = 5

val is like a constant, we cannot change the value.

var is like a variable and can be changed.

2. Type inference means that Scala compiler will infer the type.

3. Types in Scala

4. s interpolation, f interpolation, raw interpolation

5. String comparision

in Java to compare 2 strings we have to use equals method.

also in Java == is for reference comparison.

but in case of your Scala == can be used for string comparison.

6. If else

7. MATCH CASE (SWITCH IN Java)

8. for loop

in Scala:

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

in other languages:

```
for (int i=1 ; i<=10 ; i++) {

}
```

9. while loop

10. do while

11. in a block the last statement is the return statement.

12 . Scala supports

object oriented style

functional programming style

```
def squareIt(x: Int) = x*x
public int squareIt(int x) {
return x*x
}
```

Anonymous function

A function without any name.

```
x => x * x * x
```

when we have a single time use.

Scala collections

- Array
- List
- Tuple
- Range
- Set
- Map

Array

Array can be referenced by index. and it is 0 based.

arrays are mutable collection

searching based on index is very fast.

adding a new element is tricky and is inefficient operation.

List

Internally holds the elements in a singly linked list.

searching is not very efficient.

but adding a new element specially at the starting is efficient.

lot of system defined functions are available to use in the list.

Tuple

you can treat a tuple like a record in your database table.

it can hold elements of different datatypes.

it is very commonly used in Spark.

and it is 1 based index.

x._1

if we have a tuple of 2 elements then this can be considered as a special case.

the first element can be treated like a key

and second element can be treated like a value

Range

we can specify a range of values

example:

```
val x = 1 to 10

val y = 1 until 10
```

Set

a set holds only unique values.

it cannot hold duplicates.

order is not maintained.

Map

a map is nothing but a collection of key value pairs.

with the help of key we can search the value.

Map holds unique keys.

How to create a class and then create an instance of class

```
class person {
    data
    +
    behaviour
}
```

we are having a person class

each person might have some

name
age
salary

student class

52 students..

than 52 instances of student class have to be created..

1 Student class

```
class Student {

    N_eyes = 2  //class level functionality. (“static”)
```

in case of Java we used to have a “static” keyword to have such a class level functionality.

but in case of Scala we do not have “static” keyword.

```
    name
    age
    grade
    getAge
    getGrade
}
```

in a batch 52 students..

then 52 instances will be there for this Student class.

```
object Person {
    val N_EYES = 2
    def canFly: Boolean = false
}
```

to get class level functionality we use object in Scala.

when we use object then we get a singleton reference with the same name.

this basically gives you a singleton design pattern without any extra code.

how to create a class

how to create an instance of a class

class parameters and promoting it to field by putting var or val before it.

class level functionality using object

there is nothing like static in Java

design pattern where in a single file we have

object and class with the same name

this design pattern is called as companion.

object is used for class level functionality

class is used for instance level functionality.

in Scala the singleton design pattern is achieved using object

object Person

1. What is inheritance

child class inheriting from parent class

multiple inheritance is not possible

a child class cannot extends from 2 parent classes.

2. access modifiers in Scala

private
protected
no modifier (public)

3. abstract class

Abstract class can contain unimplemented methods and undefined values.

The whole purpose to create an abstract class is to implement it later by inheriting it in child class.

instantiation of abstract class is not possible.

4. trait

Multiple traits may be inherited by the same class

traits are behavior

traits does not have constructor parameter

both abstract class and traits can have implemented as well as unimplemented methods.

in Java we have interfaces which are totally unimplemented.

but in traits we can have implemented as well as unimplemented methods.

multiple inheritance is possible in Scala with the help of traits.

Case Classes

case class are special kind of classes where you need to write less code.

```
class Person {
//instance level properties

}

object Person {
//class level properties

}
```

In Java

```
public static void main(String args[]) {

}
```

1. App

App in Scala is a helper class which already have the main method.

so we can extend our class from App and we can skip writing the main method

2. Default args, named args, variable args

3. Nil, null, None, Nothing, Option, Unit

=> Null is a trait in Scala.
There exists only one instance of Null, and that is null

we should restrict the use of null as its not preferred.
because it can lead to null pointer exceptions.

=> Nil is an empty list

=> Nothing is a trait in Scala. There are no instances of Nothing.

Nothing means that there was an error or exception and nothing was returned.

Option

consider you are writing a function and you run into the situation where you dont have a useful value to return. then what to do?

returning null is not preffered because it can lead to null pointer exceptions.

Scala has a built in solution to this problem.

make sure we do not use null in our code.

rather use Option.

Option return some or none.

Unit

its like a void in Java to some extent.

()

// Nothing means that there was an error and nothing was returned.

// Unit means that there are side effects.

how to deal with nulls in your Scala code?

we should not use nulls in our code.

we should use Option

we can handle none using getOrElse

App

default args, named args and variable length args.

Nil, Null, None, Nothing, Option, Unit in Scala

Option (some and none)

getOrElse

1. yield

vector is kind of a mix of Array and a List

Array provides you indexed support.

List gives you immutability

vector provides indexed based searching and immutablity.

with the help of yield we can get a new list from a for loop.

2. if guard

3. pattern guard

case statements can be combined with if expressions to provide extra logic during pattern matching.

if you use if inside your case statement it is called as pattern guard.

4. for comprehension

```
for(i <- 1 to 10) println (i*i)

(1 to 10).foreach(i => println(i*i))


for(i <- 1 to 10) yield i*i

(1 to 10).map(i => i*i)


for (i <- 1 to 10 ; j <- ‘a’ to ‘c’) yield i*2 +” “+j
```

in Java == is meant for reference comparison

to check the natural equality we use equals method in Java.

a = new String(“Sumit”)
b = new String(“Sumit”)

a == b
the above will return false in Java because both a and b are different references.

a.equals(b) in Java it gives true.

in Scala == is meant for content comparison
== is same as equals.
== is internally implemented as equals.
eq to compare the references.

in Java == is an operator.
in Scala there are no operator everything is a method

yield

if guards

pattern guards

for comprehension

vector (immutability + indexed searched)

difference between == in Java and Scala

1. strict val vs lazy val

lazy val is evaluated during the first use.

2. default Scala packages

by default, three packages are implicitly imported for you:

Java.lang._

Scala._

Scala.Predef._

3. Scala apply

apply serves the purpose of closing the gap between object oriented and function paradign in Scala.

we will be able to call an object like a function

```
object Person {
      def apply(name: String, age: Int) = {
      s”$name is $age years old”
      }
}

Person.apply(“Sumit”,30)

case class Person(name: String, age: Int)

Person(“Sumit”,30)
```

4. Diamond problem in Scala and how it is solved.

A problem which comes with multiple inheritance.

```
class A {
      func1
}

class B{
      func1
}

class C extends A,B {
      func1()
}
```

class A class B

class C

How exactly this is handled in Scala.

multiple inheritance is possible using traits in Scala.

trait is to some extent like an interface in Java.

traits provide more flexibility than interfaces.

A trait can have both implemented and unimplemented things.

but interface has only unimplemented things.

=> Scala does not support inheritance from multiple classes, but a user can extend multiple traits in a single class.

linearization rule come into play to decide the call hierarchy.

grandChild -> traitC -> traitB -> traitA

5. Why Scala over Python or Java

Scala we can write a very consise code.

moreover Spark is written in Scala.

so if any new feature is released in Spark then first it comes in Scala and then rolled to other languages.

Scala gives you the best performance.

1. what is type Safe in Scala?

type safety means that the compiler will validate types while compiling, and throw and error if you try to assign the wrong type to a variable.

2. what is the difference between statically typed vs dynamically typed language?

A language is statically typed if the type of a variable is known at compile time rather than runtime.

Java, C, Scala

so we need to give the datatype when we are using.

In dynamically typed language the type is checked at runtime.
example:
Javascript, Python etc..

Static typing usually results in a better performance. because compiler knows datatypes in advance and has scope to do optimizations.

Static type we do not run into errors at runtime.

3. Exception handling in Scala

Exception is a abnormal condition which arise because of your code.

Error is a abnormal condition because of system issues.

```
try{
      val b = 5/0
}

catch{
      case e: Exception => println(“please give a denominator other than 0”)
}
```

4. monad

A monad is not a class or a trait. it is just a concept.

A monad is an object that wraps another object in Scala.

The output of a calculation at any step is the input to other calculations.

```
val list1 = List(1, 2, 3, 4)
val list2 = List(5, 6, 7, 8)
```

```
list1.flatMap { x => list2.map {
      y => x + y
}
}
```

```
List(List((1+5),(1+6),(1+7),(1+8)),
List((2+5),(2+6),(2+7),(2+8))

List(List(6,7,8,9),List(7,8,9,10),List(),List())

List(6,7,8,9,7,8,9,10..)
```

5. Streams in Scala

Streams are lazy lists in Scala.

6. ofDim

is used to create a multi dimentional array

```
val myarr = Array.ofDim[Int](2,2)
```

```
myarr(0)(0) = 2
myarr(0)(1) = 7
myarr(1)(0) = 3
myarr(1)(1) = 4

for (i <- 0 to 1 ; j <- 0 until 2) {
      println(myarr(i)(j))
}
```

1. Design pattern

Design pattern is a general reusable solution to a commonly occuring problem in software design.

Is is a template for how to solve a problem that can be used in many different situations.

1. Factory design pattern

The main aim of a factory design pattern is that, it separates instance creation logic from the client.

we implement instance creation logic in a factory class without exposing the logic to the client.

separates instance creation logic from the client.

it is used when we have a super class with multiple sub classes and based on input, we need to return one of the sub-class.

benefits of a factory design pattern:

1. offers loose coupling between object creation and the client.
2. clear separation of responsibilities.
3. easy to change object creation logic without effecting client program.

2. Singleton design pattern

The singleton pattern restricts the instantiation of a class to one object, and provides a global point of access to it.

```
object Student {

    //class level functionality

}
```

3. Lazy initialization

it's a technique that initializes a value on its first access.

Lazy initialization allows to defer (or avoid) some expensive computation.

```
lazy val x = {
    println("computing x")
    42
}
```

2. Difference between Array and ArrayBuffer in Scala.

both Array and ArrayBuffer are mutable.

ArrayBuffer is resizable but Array isn't.

if you append an element to an ArrayBuffer, it gets larger.

if you try to append an element in Array it will internally create a new Array.

3. how to remove duplicates from an array or List

Google REVIEWS

5 Star Google Rated Big Data Course

LEARN FROM THE EXPERT

9108179578

Call for more details

Follow US

Trainer	Mr. Sumit Mittal
LinkedIn	https://www.linkedin.com/in/bigdatabysumit/
Website	https://trendytech.in/courses/big-data-online-training/
Phone	9108179578
Email	trendytech.sumit@gmail.com
Youtube	TrendyTech
Twitter	@BigdataBySumit
Instagram	bigdatabysumit
Facebook	https://www.facebook.com/trendytech.in/