Scala Intro

Instructor: Edwin Guo

About Scala

High-level language for the JVM

- Object oriented + functional programming

Statically typed

- Comparable in speed to Java*
- Type inference saves us from having to write explicit types most of the time

Interoperates with Java

- Can use any Java class (inherit from, etc.)
- Can be called from Java code

"If I were to pick a language to use today other than Java, it would be Scala" by James Gosling

"I can honestly say if someone had shown me the Programming in Scala book by by Martin Odersky, Lex Spoon & Bill Venners back in 2003 I'd probably have never created Groovy." by James Strachan.

Best way to Learn Scala

Interactive scala shell (type scala in your terminal env)
Supports importing libraries, tab completing, and all of the constructs in the language

Quick Taste of Scala

Declaring variables:

```
var x: Int = 7
var x = 7        => type inferred
val y = “hi”   =>  read-only, immutable when declare as val
```

Functions:

```
def square(x: Int): Int = x*x
def square(x: Int): Int = {
    x*x}
def announce(text: String) {
   println(text)  }
```

Main Scala:

```
object HelloWorld {

  def main(args: Array[String]) {

   println("Hello, world!”)
}
}
```

Java equivalent:

```
int x = 7;

final String y = “hi”;
```

Java equivalent:

```
int square(int x) {
  return x*x;
}
void announce(String text) {
    System.out.println(text); }
```

Main Java:

```
public class Test {

  static void main(String[] args){

      System.out.println("Hello World");

  }
}
```

Continue…

Processing collections with functional programming

```
val lst = List(1, 2, 3)
lst.foreach(x => println(x))          // prints 1, 2, 3
lst.foreach(println)                  // same as above

lst.map(x => x + 2)                   // returns a new List(3, 4, 5)
lst.map(_ + 2)                        // same as above

lst.filter(x => x % 2 == 1)           // returns a new List(1, 3)
lst.filter(_ % 2 == 1)                // same as above

lst.reduce((x, y) => x + y)           // => 6
lst.reduce(_ + _)                     // same as above

def addTwo (num: Int) = num + 2       // define a function
lst.map(x => addTwo(x))               // increase every element by two
lst.map(addTwo _)                     // same as above
```

Functional methods on collections

Method on Seq[T]	Explanation
map(f: T => U): Seq[U]	Each element is result of f
flatMap(f: T => Seq[U]): Seq[U]	One to many map
filter(f: T => Boolean): Seq[T]	Keep elements passing f
exists(f: T => Boolean): Boolean	True if one element passes f
forall(f: T => Boolean): Boolean	True if all elements pass
reduce(f: (T, T) => T): T	Merge elements using f
groupBy(f: T => K): Map[K, List[T]]	Group elements by f
sortBy(f: T => K): Seq[T]	Sort elements

Scala Basics Terms

Object: An entity that has state and behavior is known as an object. For example: table, person, car etc.

Class: A class can be defined as a blueprint or a template for creating different objects which defines its properties and behavior.

Method: It is a behavior of a class. A class can contain one or more than one method. For example: deposit can be considered a method of bank class.

Closure: Closure is any function that closes over the environment in which it's defined. A closure returns value depends on the value of one or more variables which is declared outside this closure.

Traits: Traits are used to define object types by specifying the signature of the supported methods. It is like interface in java.

Things to note about Scala

- It is case sensitive
- File are end with .scala
- Like Java, entry point is main() function
- Identifier name can’t start with numbers => def 123name = “exception" throws error: Invalid literal number

Variable declaration

- var vs val

```
scala> val v1 = "cloud"
v1: String = cloud
scala> v1 = "data"
<console>:12: error: reassignment to val
       v1 = "data"
          ^
scala> var v1 = "cloud"
v1: String = cloud
scala> v1 = "data"
v1: String = data
scala> v1
res1: String = data
```

Operations on variable

```
scala>var v1 = 5
scala>var v2 = 2
scala> v1 + v2
res2: Int = 10
scala> val a = 2
scala> val b = 2
b: Int = 2
scala> a == b
res4: Boolean = true
```

Complete list: https://www.tutorialspoint.com/scala/scala_operators.htm

If else

```
scala> if (v1 == "weclouddata") println("Right here") else println("Hello?")
Right here
```

Iteration

```
scala> for( a <- 1 to 3){
     |    println( "Currently in loop " + a)
     | }
Currently in loop 1
Currently in loop 2
Currently in loop 3
```

Declare function

```
scala> def multipleByTwo (input: Int) = input * 2
multipleByTwo: (input: Int)Int

scala> multipleByTwo(5)
res16: Int = 10
```

Popular Data structure

- Arrays

  scala> val names = Array("Jack", "Lucy", "Mike")
  names: Array[String] = Array(Jack, Lucy, Mike)

  scala> names(0)
  res17: String = Jack

  scala> names(1)
  res18: String = Lucy

- Lists

- scala> val lst = List[Int](95,96,97)
- lst: List[Int] = List(95, 96, 97)

- scala> lst(1)
- res19: Int = 96

- scala> lst(2)
- res20: Int = 97

- scala> List(3,4,5) ++ List(6,7,8)
- res40: List[Int] = List(3, 4, 5, 6, 7, 8)

- scala> List(3,4,5) :+ 7
- res41: List[Int] = List(3, 4, 5, 7)

Continue...

- Sets

```
scala> val sts = List(3, 5, 5, 5, 10, 12).toSet
sts: scala.collection.immutable.Set[Int] = Set(3, 5, 10, 12)
scala> val sts = Set(3, 5, 5, 5, 10, 12)
sts: scala.collection.immutable.Set[Int] = Set(3, 5, 10, 12)
scala> sts + 13
res42: scala.collection.immutable.Set[Int] = Set(5, 10, 13, 12, 3)
scala> sts - 5
res43: scala.collection.immutable.Set[Int] = Set(3, 10, 12)
```

Tuple

- val t1 = Tuple1(“Toronto”)
- val t2 = (“Ottawa”, “London”)

- To access:
- t1._1
- t2._2, t2._1

Popular Data structure

- Maps

```
scala> val kv = Map("name" -> "WeCloudData", "class" -> "Data Engineer Basic")
kv: scala.collection.immutable.Map[String,String] = Map(name -> WeCloudData, class -> Data Engineer Basic)
scala> kv("empty")
java.util.NoSuchElementException: key not found: empty
  at scala.collection.immutable.Map$Map2.apply(Map.scala:135)
  ... 28 elided

scala> kv("name")
res49: String = WeCloudData
```

- Option

```
scala> kv.get("empty")
res52: Option[String] = None

scala> kv.get("name")
res53: Option[String] = Some(WeCloudData)

scala> res53.isEmpty
res54: Boolean = false

scala> res52.isEmpty
res55: Boolean = true
```

Writing & Running a program

- Create a scala file and write your hello world:

```
object HelloWorld {
  def main(args: Array[String]) {
  println("Hello, world!”)}}
```

Run scala HelloWorld.scala in your terminal env

=> Hello, world!

Case Class

```
scala> case class Account(name: String, number: Long, balance: Double){
def printBalance() = {println(balance)};
def updateBalance(num: Double) = balance + num}
defined class Account

scala> Account("Edwin",  352279230, 945.23).updateBalance(2.5)
res2: Double = 947.73

scala> val acc = Account("Edwin",  352279230, 945.23)
acc: Account = Account(Edwin,352279230,945.23)

scala> acc.updateBalance(2.5)
res3: Double = 947.73

scala> acc
res4: Account = Account(Edwin,352279230,945.23)
```

Continue....

```
trait MovingObject { def returnNumWheels: Int}

case class Bike(numberOfWheels: Int, withEngine:Boolean) extends MovingObject
{ override def returnNumWheels() =
   numberOfWheels}
defined class Bike

case class Car(numberOfWheels: Int, withEngine:Boolean) extends MovingObject
{ override def returnNumWheels() = numberOfWheels;
def startEngine() =
 if (withEngine)
     println("Starting engnie!!!")
 else
     println("There is no engine...")}
```

Pattern matching

```
case class Player(name: String, score: Int)

def printMessage(player: Player) = player match {
  case Player(_, score) if score >= 60 => println("MVP")
  case Player(_, score) if score <60 && score > 10 => println("Great job!")
  case Player(player, _) => println("You can do Better!")
}
```

Companion Object

```
scala> :paste
import scala.math._

case class Circle(radius: Double) {
  import Circle._
  def area: Double = calculateArea(radius)
}

object Circle {
  private def calculateArea(radius: Double): Double = Pi * pow(radius, 2.0)
}
```

```
val circle1 = new Circle(5.0)

circle1.area
```

```
import scala.math._
defined class Circle
defined object Circle
circle1: Circle = Circle(5.0)
res6: Double = 78.53981633974483
```

Tail Recursion

- def factorial(n: Int): Int =
  | if (n == 0) 1 else n * factorial(n - 1)

When you call factorial(4), it being evaluated as:

factorial(4)
if (4 == 0) 1 else 4 * factorial(4 - 1)
4 * factorial(3)
4 * (3 * factorial(2))
4 * (3 * (2 * factorial(1)))
4 * (3 * (2 * (1 * factorial(0))))
4 * (3 * (2 * (1 * 1)))
24

Continue…

- A tail reserve way to rewrite the factorial function:

```
import scala.annotation.tailrec
def factorial(n: Int) = {
  @tailrec
  def  exec(acc: Int, n: Int) = {
     If (n <= 1){
       acc
     } else {
       exec(acc * n, n-1)
     }
  }
  exec (1, n)
}
```

FoldLeft, reduce

- Range(1,5).toList.map(_ + 1)
- res28: List[Int] = List(2, 3, 4, 5)

- scala> Range(1,5).toList.foldLeft(List[Int]()){(a, b) => a :+ b + 1}
- res29: List[Int] = List(2, 3, 4, 5)

- scala> Range(1,5).toList.reduce(_ + _)
- res30: Int = 10

Higher Order Functions

- Function that take function as parameter or a function that return a function

```
def  numFunction(numFunc: (Int, Int) => Int, v1: Int, v2: Int) = numFunc(v1, v2)
numFunction: (numFunc: (Int, Int) => Int, v1: Int, v2: Int)Int

scala> def timeTwoVal(a: Int, b: Int) =a * b
timeTwoVal: (a: Int, b: Int)Int

scala> def addTwoVal(a: Int, b: Int) = a + b
addTwoVal: (a: Int, b: Int)Int

scala> numFunction(timeTwoVal, 3, 4)
res33: Int = 12

scala> numFunction(addTwoVal, 3, 4)
res34: Int = 7
```

Continue....

```
def returnFunc(a :Int, b: Int) = {
 a - b match {
  case v if v >= 0 => (v1 : Int, v2 : Int) => v1 + v2
  case _ =>  (v1 : Int, v2 : Int) => v1 * v2
}
}
```

```
scala> returnFunc(3, 4)(1, 2)
res36: Int = 2
```

```
scala> returnFunc(10, 3)(1, 2)
res37: Int = 3
```

Anonymous Functions

- scala> val functor = (x: Int) => x + 1
- functor: Int => Int = $$Lambda$1669/367602319@103c46e9

- scala> functor(5)
- res38: Int = 6

Exercise

- **Part1:**
- The goal of this first exercise is to review quickly the basics of Scala. In the file Part1.scala, a sequence of Person objects have been initialized.

- Complete the implementation of the method printStatus that should print the output:
- D is young [8]
- A is young [15]
- G is young [21]
- B is not that young [24]
- F is not that young [34]
- C is not that young [62]
- E is not that young [90]

- Complete the implementation of the method getCityStats that should return a Map of the city names with their occurrences:
- Map(
- "Quebec" -> 1
- "Granby" -> 1
- "Montreal" -> 4
- "Sherbrooke" -> 1
-)
- https://github.com/EdwinGuo/scala-exercises/blob/master/src/main/scala/Part1.scala

-

Implicit

- When the compiler finds an expression of the wrong type for the context, it will look for an implicit Function value of a type that will allow it to typecheck. So if an A is required and it finds a B, it will look for an implicit value of type B => A in scope (it also checks some other places like in the B and A companion objects, if they exist).

- ***Example***:
- **scala> import scala.language.implicitConversions**
- **import scala.language.implicitConversions**

- **scala> implicit def stringToInt(s: String) = s.length**
- **stringToInt: (s: String)Int**

- **scala> def printNum(len: Int) = println("The length is " + len.toString)**
- **printNum: (len: Int)Unit**

- **scala> printNum(“Weclouddata")**
- **The length is 11**

Future

- Future allow you to run your logic concurrently in a simple and manageable way.

- Example:
- import scala.util._
- import scala.concurrent.ExecutionContext.Implicits.global
- import scala.concurrent.Future

- val result = Future{3/2}

- result.value.get match {
- case Success(s) => println("success " + s) ;
- case Failure(_) => println("failure")}

Execise

1) Write a loop that swaps adjacent elements of an array of integers. For example, Array(1, 2, 3, 4, 5) becomes Array(2, 1, 4, 3, 5).
2) Given an array of integers, produce a new array that contains all positive values of the original array, in their original order, followed by all values that are zero or negative, in their original order.
3) How do you compute the average of an Array[Double]?
4) How do you rearrange the elements of an Array[Int] so that they appear in reverse sorted order? How do you do the same with an ArrayBuffer[Int]?
5) Write a code snippet that produces all values from an array with duplicates removed. (Hint: Look at Scaladoc.)
6) Rewrite the example at the end of Section 3.4, “Transforming Arrays,” on page 34 using the drop method
7) Set up a map of prices for a number of gizmos that you covet. Then produce

- Write a new Centigrade-to-Fahrenheit conversion (using the formula (x * 9/5) + 32), saving each step of the conversion into separate values. What do you expect the type of each value will be?
- Modify the Centigrade-to-Fahrenheit formula to return an integer instead of a floating-point number.
- Using the input value 2.7255, generate the string “You owe $2.73.” Is this doable with string interpolation?
- Is there a simpler way to write the following?

- val flag: Boolean = false
- val result: Boolean = (flag == false)
- Convert the number 128 to a Char, a String, a Double, and then back to an Int. Do you expect the original amount to be retained? Do you need any special conversion functions for this?
- Using the input string “Frank,123 Main,925-555-1943,95122” and regular expression matching, retrieve the telephone number. Can you convert each part of the telephone number to its own integer value? How would you store this in a tuple?

- Write a function that computes the area of a circle given its radius.
- Provide an alternate form of the function in exercise 1 that takes the radius as a String. What happens if your function is invoked with an empty String ?
- Write a recursive function that prints the values from 5 to 50 by fives, without using for or while loops. Can you make it tail-recursive?
- Write a function that takes a milliseconds value and returns a string describing the value in days, hours, minutes, and seconds. What's the optimal type for the input value?
- Write a function that calculates the first value raised to the exponent of the second value. Try writing this first using math.pow, then with your own calculation. Did you implement it with variables? Is there a solution available that only uses immutable data? Did you choose a numeric type that is large enough for your uses?
- Write a function that calculates the difference between a pair of 2D points (x and y) and returns the result as a point. Hint: this would be a good use for tuples (see Tuples).
- Write a function that takes a 2-sized tuple and returns it with the Int value (if included) in the first position. Hint: this would be a good use for type parameters and the isInstanceOf type operation.
- Write a function that takes a 3-sized tuple and returns a 6-sized tuple, with each original parameter followed by its String representation. For example, invoking the function with (true, 22.25, "yes") should return (true, "true", 22.5, "22.5", "yes", "yes"). Can you ensure that tuples of all possible types are compatible with your function? When you invoke this function, can you do so with explicit types not only in the function result but in the value that you use to store the result?

Advance Feature in Scala

Partial function

- cala> def addOne: PartialFunction[Int, Int] = {case input: Int => input + 1}
- addOne: PartialFunction[Int,Int]

- scala> addOne(3)
- res2: Int = 4

- scala> def addTwo: PartialFunction[Int, Int] = {case input: Int => input + 2}
- addTwo: PartialFunction[Int,Int]

- scala> def addThree: PartialFunction[Int, Int] = {case input: Int => input + 3}
- addThree: PartialFunction[Int,Int]

- scala> val addSix = addOne andThen addTwo andThen addThree
- addSix: PartialFunction[Int,Int] = <function1>

- scala> addSix(1)
- res4: Int = 7

Curry Function

- def aTimeb(a: Int)(b: Int) = a * b
- val timeThree = aTimeb(3) _
- timeThree(5)