CS 220 Final Project
All About Scala

Jake Ballinger & Katie Beisler

Honor Code: We pledge that all work is ours, unless otherwise cited.

December 9, 2015

1. Background

Scala (pronounced SKAH-lak, exactly like the word "stairs" in Italian) is a portmonteau of the words "scalable" and "language". It was designed to rectify the shortcomings of Java (its verboseness, lack of functional programming, and rigidness in defining functions) and maintain the elegance and readability of Python while the keeping the portability of Java [4].

```
/*************************************
* helloworld.scala
*
* Jake Ballinger
* CMPSC 220
* Fall 2015
* Dr. Roos
* Honor Code: Although it's difficult to write a unique
*       "hello world" program, I have tried my best.
*
* Description: Says "Hello World!"
* Compile: scalac helloworld.scala
* Run: scala helloworld
*************************************/

// An object is equivalent to a class in Java
object helloworld {
  // methods are declared as "def", just like in Python
  def main(args: Array[String]) {
    // Removes the annoying System.out from Java
    println("Hello, world!") // note the lack of semicolons
  } // main
} // HelloWorld
```

2. Importance

2.1 Where is Scala used?

Scala is the language of what is called *Fast Data*. In 2013 the Hadoop community realized that a replacement for MapReduce was needed and they embraced Apache Spark, which is written in Scala, as the replacement for fast data analytics. Twitter wrote Scalding in Scala on top of the Hadoop API. Finagle, an extensible RPC system for the JVM used to construct high-concurrency servers, was also developed by Twitter and written in Scala. Likewise, Akka was written in Scala to be a fact, concurrent framework for building distributed applications. Other projects written in Scala include ADAM, a genomics processing engine, and Lichess, an application that enables thousands of concurrent chess games using just one server [5].

2.2 Why is it worth studying?

Since Scala is so widely used in data processing—to the point where its usage is almost reaching Python and R levels—it pays to learn this language that is so similar to Java, except with extra functional elements.

Twitter developers cited their decision to program in Scala because it is fast, supports long-running processes, has advanced features, is fun, and has a very active and accessible community of developers, including the people who build the language [3].

3. Comparison to Similar Languages

Scala is most commonly compared to Java, probably because Scala was written to rectify all of Java's shortcomings. Scala is frequently cited as more fun than Java (although this may be due to Scala's functional nature) and not quite as verbose. Additionally, Scala are Java are both compiled, strongly-typed, and highly portable. Additionally, they are both used in Apache Spark, which is where one of the main comparisons comes in.

Given its functional nature, it is inevitable that Scala is also compared with Python. Python is also object-oriented and capable of both imperative and declarative programming, but Scala runs on the JVM, which is faster than the Python interpreter (although both are as nearly ubiquitious). So the difference between Scala and Python is similar to that of Java and Python—the ecosystems are different and the coding process differs. Python is better for rapid prototyping and fast, readable code, and while Scala isn't far behind, Scala works much better for scalable deployments [2].

For example, here is a code in Python that takes input from a .txt file, count the frequency of the letters, sorts them by frequency, and truncates the string after it finds the "_" character.

```
# unjumble.py
#
# Author: Jake Ballinger
# Date: 01 December 2015
# Honor Code: All work is mine.
#
# Compile/Run: python unjumble.py
```

```
# Output: binoculars

# Variables
d = {} # The dictionary in which we read the long string
x = [] # The list we use to grab the keys of d

def decode():
  # Counts the frequency of characters occuring in the long
     ↪ string
   file = "jumble.txt" # Read in the file
   for line in open(file, 'r'): # Parse through the file line by
      ↪ line
     for i in range(0, len(line)): # For each character, do the
        ↪ following:
       if line[i] not in d.keys(): # If not there, add to the
          ↪ dictionary
         d[line[i]] = 1
       else: # Else, increment the count (frequency) of that
          ↪ character
         d[line[i]] += 1
   sort() # Call the sort method
# decode() end

def sort():
  # Sort the values in the dictionary by size
  x = [i for i in sorted(d, key=d.get, reverse=True)] # Creates
     ↪ a new list of the keys ordered by value
  # This essentially sorts a representation of the dictionary.
     ↪ Since dictionaries are orderless, this is cool!
  # Now get rid of everything after '_' and print out the
     ↪ resulting word.
   print("".join(x[:x.index('_')]))
# sort() end

# Here, solve the problem
decode()
```

Now here is the same thing in Scala:

```
/*****************************************************
* unjumble.scala
*
* Jake Ballinger
* CMPSC 220
* Dr. Roos
* Fall 2015
* 03 December 2015
```

```
* Honor Code: I pulled my own teeth with this one. All work is
   ↪ mine.
*
* Compile: scalac unjumble.scala
* Run: scala unjumble (must have jumble.txt in the directory!)
**********************************************/

import scala.io.Source
import scala.collection.mutable.Map

object unjumble {
  // Global Variables
  var m:scala.collection.mutable.Map[Char, Int] = scala.
     ↪ collection.mutable.Map()

  def decode() {
        var line: String = ""
        var lineCh: Array[Char] = Array()
        for(line <- Source.fromFile("jumble.txt").getLines()) {
          lineCh = line.toCharArray()
                for (i <- 0 until lineCh.length) {
                  if (m.contains(line(i))) {
                        // Add things to the map
                       m.put(line(i), m(line(i)) + 1)
                } else
                  // Add the key, value pair
                  m += (line(i) -> 1)
                } // for
        } // for
  // Almost there! Need to figure out a way to truncate the list
     ↪  after the '_'
  line = m.toSeq.sortBy(-_._2).map(_._1).mkString
  line = line.slice(0, line.indexOf('_'))
  println(line)

  } // decode

  def main(args: Array[String]) {
        decode()
  } // main
} // unjumble
```

We see that Python code takes only 15 lines while the Scala code takes 19, although the Python is much easier to read. Now, imagine this problem in Java! It would take much more code to complete.

Haskell is purely functional, but Scala can hold its own in a functional programming

environment. While Haskell takes the cake in all things syntactical, and while Haskell's type inference just *works*, Scala's object-oriented style allows for a certain flexibility in programming that is simply not possible in Haskell. Unlike Haskell's lazy evaluation, Scala use a strict evaluation [6].

4. Classification

Scala is a compiled language with a mix of functional and declarative styles. It is also object-oriented like Java [4]. Like most modern languages, Scala is lexically scoped [1]. It also has a strong static typing system [4].

5. Features of Scala

5.1 Object-Oriented

Scala is an object-oriented programming language, as demonstrated by the PointTest program:

```
/*******************************************************
* PointTest.scala
*
* Jake Ballinger
* CMPSC 220
* Fall 2015
* Dr. Roos
* 29 November 2015
* Honor Code: This is my work.
*
* Description: Creates a simple 3D point object in Scala.
* Compile: scalac PointTest.scala
* Run: scala PointTest
*******************************************************/
object PointTest {
  def main(args: Array[String]) {
    // Create Point A and move it (10, 10, 10)
    // Instantiating a new instance of a class is like Java
    val A = new Point(0, 1, 2)
    println("Point A: " + A)
    // The dot operator in Scala works the same as eveywhere
    A.move(10, 10, 10)
    println("Point A after a shift of 10, 10, 10: " + A)

    // Do the same thing for Point B
    val B = new Point(-1, 7, 2)
    println("Point B: " + B)
    B.move(12, 5, -9)
    println("Point B after a shift of -12, 5, -9: " + B)
```

```
    // Create a dummy point at the origin
    val C = new Point(0, 0, 0)
    // Now subtract and assign to D
    val D = C.sub(A, B)
    println("Now, give the parametric equation of the line
      ↪ between these two points.")
    println("Step 1")
    println(A + " - " + B + " = " + D)
    println("Step 2")
    println("(x, y, z) = " + A + " + t" + D)
  } // main
} // PointTest

// When declaring a class, use "class"
// Note the different syntax in parameters: name: type, ...
class Point(xi: Int, yi: Int, zi: Int) {
  var x: Int = xi
  var y: Int = yi
  var z: Int = zi

  // Define a method by the keyword "def"
  def move (dx: Int, dy: Int, dz: Int) {
    x += dx
    y += dy
    z += dz
  } // move

  // Sub: takes two points, subtracts them, and assigns them to
    ↪ a new point
  // Note the return type is after the parameter list
  def sub (A: Point, B: Point): Point = {
    var a: Point = A
    var b: Point = B
    var cx: Int = A.x - B.x
    var cy: Int = A.y - B.y
    var cz: Int = A.z - B.z
    return new Point(cx, cy, cz)

  } // sub
  // Override the toString() method
  override def toString(): String = "(" + x + ", " + y + ", " +
    ↪ z + ")"
} // Point
```

5.2 Functions are Objects

In Scala, a function is treated as an object [4]. This makes it possible to pass in a function as a parameter to a method, and it allows for currying, as demonstrated below:

```
/************************************
* curry.scala
*
* Jake Ballinger
* 30 November 2015
* CMPSC 220 Final Project
* Dr. Roos
* Fall 2015
* Honor Code: All work is mine
*
* Description: A simple demonstration
*       of currying in scala.
*
* Compile: scalac curry.scala
* Run: scala curry
************************************/
object curry {
  def main(args: Array[String]) {
    // Show how the sum function works
    println("sum(1, 2, 3) = " + sum(1, 2, 3))

    // Curry: reassign the sum function to the addSix
    // variable, where the first parameter is defined
    val addSix = sum(6, _:Int, _:Int)

    // Now print and show the currying
    println("addSix(2, 3) = " + addSix(2, 3))
  } // main

  // The sum method. Takes three Ints as input
  // and returns an Int.
  def sum(a: Int, b: Int, c: Int): Int = {
    return a + b + c
  } // sum
} // curry
```

5.3 Simple Syntax

Scala allows for much simpler syntax for writing prgrams that need to be, well, simple. The following program is a calculation of all the prime numbers up to 100. While it is slightly inefficient, it gets the job done in five lines of code.

```
/************************************
* primes.scala
*
* Katie Beisler
* CMPSC 220
* Fall 2015
* Dr. Roos
* Honor Code: I tried to make a prime's calculator. Ta Da
*
* Description: Prints out prime numbers until 100
* Compile: scalac primes.scala
* Run: scala primes
************************************/

//Prints out primes up to 100...very inefficiently....
object primes {
  def main(args: Array[String]) //defining array
        {
         def isPrime(number: Int) = (2 until number) forall (
            ↪ number % _ != 0) //prime calculation
         for (p <- 1 to 100 if isPrime(p))
                println(p)
        }//isPrime method in main
}//primes
```

5.4 Nested Functions

In Scala it is possible to nest function definitions. Nested methods can use and even update everything visible in their scope (including local variables or arguments of enclosing methods) without running inefficiently.

```
/************************************
* recurse.scala
*
* Katie Beisler
* CMPSC 220
* Fall 2015
* Dr. Roos
* Honor Code: I tried to make recursion program
*
* Description: sorts recursively
* Compile: scalac Recurse.scala
* Run: scala Recurse
************************************/

object Recurse {
```

```
def sort(a: Array[Int]) { //initial sort

  def swap(i: Int, j: Int) { //nested function
    val t = a(i); a(i) = a(j); a(j) = t
  }

  def sort1(l: Int, r: Int) { //pivot defined
    val pivot = a((l + r) / 2)
    var i = l //no semi-colons yay!!!
    var j = r
    while (i <= j) { //pivot
      while (a(i) < pivot) i += 1
      while (a(j) > pivot) j -= 1
      if (i <= j) {
        swap(i, j)
        i += 1
        j -= 1
      }
    }
    if (l < j) sort1(l, j)
    if (j < r) sort1(i, r)
  }

  if (a.length > 0) //similar to java syntax, interesting
    sort1(0, a.length - 1)
}

def println(ar: Array[Int]) { //what to print
  def print1 = {
    def iter(i: Int): String =
      ar(i) + (if (i < ar.length-1) "," + iter(i+1) else "")
    if (ar.length == 0) "" else iter(0)
  }
  Console.println("[" + print1 + "]")
}

def main(args: Array[String]) { //array input
  val ar = Array(6, 2, 8, 5, 1, 4, 6, 7, 3, 4, 1, 10, 13, 4,
    ↪ 23) //can fill this infinitely, size does not need to
    ↪ be defined
  println(ar)
  sort(ar)
  println(ar)
}
```

```
}
```

References

[1] Debasish Ghosh. Why I like Scala's Lexically Scoped Open Classes. http://debasishg.blogspot.com/2008/02/why-i-like-scalas-lexically-scoped-open.html. Accessed: 07 December 2015.

[2] Nate Hemingway. Which one should I learn? Python or Scala? https://www.quora.com/Which-one-should-I-learn-Python-or-Scala. Accessed: 08 December 2015.

[3] Glenn Kelman. Engineer-to-Engineer Talk: How and Why Twitter Uses Scala. https://www.redfin.com/devblog/2010/05/how_and_why_twitter_uses_scala.html. Accessed: 07 December 2015.

[4] Martin Odersky. MS Windows NT kernel description. http://www.scala-lang.org/what-is-scala.html. Accessed: 2015-12-06.

[5] Laura Masteron. Eight hot technologies that were built in Scala. https://www.typesafe.com/blog/eight-hot-technologies-that-were-built-in-scala. Accessed: 07 December 2015.

[6] Jesper Nordenberg. . http://jnordenberg.blogspot.com/2012/05/my-take-on-haskell-vs-scala.html. Accessed: 08 December 2015.