

CS 162 Week 1

Kyle Dewey

Overview

- Basic Introduction
- CS Accounts
- Scala survival guide

Office Hour

- Choose an hour from within:
 - Tuesday/Thursday 11 AM - 1 PM
 - Friday 11 AM - 4 PM
- Also available by appointment

Google Group

- We have a Google group (162w14)
- Feel free to discuss, even post test cases
- Pretty much anything CS162-related that doesn't involve sharing code

Communication Policy

- Assume I'll take 24 hours to respond to any email
- I'm usually a lot faster than that
- Google group is *usually* a better resource
 - I can still answer it
 - Other people can see it
 - Someone else may respond faster

Friday, January 10, 14

-This is mostly so I'm not swamped with emails right before a project. It gets ugly.
-Certain things are obviously directed at me: what is my grade for this? However, much project help is better fit for the Google group

CS Account

- You will need a CS account
- One can be created at:
<https://accounts.engr.ucsb.edu/create/>

Collaboration

- You may discuss **ideas**
 - I.e. check if the list is sorted
- You may exchange **test cases**
 - I.e. this test fails if your code does not check if the list is sorted
- All these can be freely posted to the Google group, too

Collaboration

- You may **not exchange or discuss code**
 - I.e. this code checks if a list is sorted
 - I.e. you need a function that uses three variables that returns...
- We automatically determine similarity scores for code via a proven reliable method

Protect Your Code

- Do **not** host your code on a public repository
- Execute `chmod 600 filename` for each file in your project (`chmod 600 *.scala` usually does the trick)
- Otherwise, people **can** and **have** taken other people's code
- In general, it is not possible to determine who stole from who

Discussion / Lecture Importance

- These are not mandatory, but it is nearly impossible to do the assignments without them
- “nearly impossible” means “expect to take several hours studying the lecture and discussion notes before you can implement it”
- If you cannot make lecture/discussion, arrange for someone else to take notes for you

Assignment Difficulty

- The assignments are intended to be very difficult and take multiple sittings to do
- It is ill-advised to do them alone
- It is even more ill-advised to do them the night before

Friday, January 10, 14

–Your entire grade is based on 7 assignments, so the difficulty needs to come from somewhere

–Start early!

Assignment Difficulty

- Most of the difficulty will be in figuring out exactly what needs to be done and how to do it
- The coding is accidental
- Historically, shorter complete solutions tend to pass more tests

Scala

Scala Discussion

- It's not possible to cover everything in these slides in-depth in one section
- This is intended as a strong foundation for the class assignments
- Examples are on my website
(`scala_examples.zip`)

What?

- A non-Java language that runs on the Java Virtual Machine (JVM)
- Essentially a “better Java”
- Better suited for object-oriented programming and functional programming

Why

- Less boilerplate
- More expressive (read: less code)
 - Think more, type less
- Clarity

Properties and Idioms

- Everything is an object (unlike Java)
- Emphasis on immutable state
 - In other words, avoid reassignment

Variable Declaration

- Two forms: `val` and `var`
 - `val` creates a runtime constant, much like `final` in Java
 - `var` creates a typical mutable variable (HIGHLY discouraged and will typically negatively impact grade)

Method Definition

- Uses the `def` reserved word
- Everything is `public` by default
- The result of the last expression in the function is what is returned - no need for `return` (which should be avoided)

Type Inferencer

- Can automatically determine the type of
 - Variables
 - Function return values
 - Anonymous function parameters
- Not completely foolproof, but usually excellent

Higher-Order Functions

- Functions can take other functions as parameters, or even return functions
- Functions (well, closures) can be created on the fly
- Note: this is strictly more powerful than function pointers
- For the JavaScript people: think callbacks

Classes

- Created with the `class` reserved word
- Defaults to `public` access
- Constructors are **not** typical

Traits

- Created with the `trait` reserved word
- Like a mixin in Ruby
- Think Java interfaces, but they can have methods defined on them
- More powerful than that, but not relevant to this course

object

- Used in much the same way as `static` is in Java
- Defines both a class and a single instance of that class (and **only** a single instance)
- Automated implementation of the Singleton design pattern
- Keeps everything consistently an object

`equals`, `==`, and `eq`

- As with Java, if you want to compare **value** equality, you must extend `equals`
- Case classes automatically do this for you
- However, instead of saying `x.equals(y)`, merely say `x == y`
- If you want **reference** equality, say:
`x eq y`

Case Classes

- Behave just like classes, but a number of things are automatically generated for you
- Including `hashCode`, `equals`, and `getters`
- Typically used for pattern matching

Pattern Matching

- Used extensively in Scala
- Like a super-powerful `if`
- Used with the `match` reserved word, followed by a series of `cases`

null

- In general, `null` is an excellent wonderful/
terrible feature
- Often poorly documented whether or not
`null` is possible
- Checking for impossible cases
- Not checking for possible cases

Option

- A solution: encode `null` as part of a type
- For some type, say `Object`, if `null` is possible say we have a `Nullable<Object>`
- Scala has this, known as `Option`
- In general, if `null` is possible, use `Option`

Tuples

- For when you want to return more than one thing
- Can be created by putting datums in parenthesis
- Can pattern match on them

Sequence Processing Functions

AKA: Why `while` is rare and `for` isn't `for`

Looping

- Scala has a `while` loop, but its use is highly discouraged (again, point loss)
- It's not actually needed
- General functional programming style is recursion, but this is usually overkill

foreach

- Applies a given function to each element of a Seq

map

- Like `foreach`, in that it applies a given function to each element of a sequence
- However, it also returns a new sequence that holds the return values of each of the function calls

filter

- Takes a predicate, i.e. a function that returns true or false
- Applies the predicate to each item in a list
- A new list is returned that contains all the items for which the predicate was true

foldLeft

- Extremely flexible, but sometimes unwieldy
- Takes a base element
- Takes a function that takes a current result and a current list element
- The function will manipulate result with respect to the current element

flatMap

- Like `map`, but made especially for functions that return `Seqs`
- Will internally “flatten” all of the inner `Seqs` into a single `Seq`
- More on this later in the course

`for` Comprehensions

- Much like Python's list comprehensions
- Internally translated into a series of `foreach`, `flatMap`, `map`, and `filter` operations

Compiling/Running Code

- Use `scalac` to compile code
- Use `scala` to run the code
- `scala` and `scalac` are all on CSIL

Running the REPL

- Just type `scala` at the command line
- Pretty nifty to quickly check to see what an expression does

Development

- If you want an IDE, IntelliJ IDEA has been recommended
- Personally, I use `emacs` and the `scala-mode` plugin (needs to be downloaded)

Assignment 1

- Due Tuesday, January 14
- Will need most everything shown here
- Hint hint useful APIs:
 - `Seq.mkString`
 - `Seq.reverse`
 - `Seq.head`
 - `Seq.tail`