CS162 Week 1

Kyle Dewey

Overview

- Basic Introduction
- CS Accounts
- Scala survival guide

Office Hour

- Tuesdays from 11 AM - 12 PM in Phelps 1413 (the TA office)
- Also by appointment

Google Group

- We have a Google group (162w13)
- 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

CS Account

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

Scala

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 `case`s

`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 `NullPossible<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

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

Taking a Step Back...

- When do we write loops?
  - Transform data
  - Scan data
  - Aggregate data
- Higher-order functions allow us to abstract away much of this

map

- Applies a given function to each element of a sequence
- Returns a new sequence that holds the results

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

Compiling/Running Code

- Use `scalac` to compile code
  - Alternatively use `fsc`, the fast Scala compiler
- Use `scala` to run the code
- `scala`, `scalac`, and `fsc` 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
- Will need most everything shown here
- Hint hint useful APIs:
  - `Seq.mkString`
  - `Seq.reverse`
  - `Seq.head`
  - `Seq.tail`