Announcements

- AI Quiz on Autolab available now.
  - Due Mon Feb 6 @ 11:59 PM
  - Submit as many times as you want
  - To pass the class, your final submission must indicate that you have satisfied the requirement (1.0 out of 1.0 score)
  - If you don’t have access to CSE-250 on Autolab, let course staff know.

- Recitations and Office Hours start next week
- JOIN THE PIAZZA!
Why Scala?

- Strongly Typed Language
  - The compiler helps you make sure you mean what you say.

- JVM-based, Compiled Language
  - Run anywhere, but also see the impacts of data layout.

- Interactive REPL Interpreter
  - It’s easy to test things out quickly (more on this later).

- Well Thought-Out Container Library
  - Clearly separates data structure role and implementation.
Environment

- IntelliJ
  - Ubuntu Linux
  - MacOS
  - Windows

- Emacs/Vim + SBT
  - Ubuntu Linux
  - MacOS
  - Windows / WSL

Labs will come with an IntelliJ workspace and an SBT build.sbt file
Coding Style is Important!

- Indentation
- Names
- Comments
- Consistency
- Braces
- Return values

```scala
def doThings() = {
  val someString = 42
  val xyz = for (i <- 1 to 5) yield i
  val QQ = xyz.map(_ + someString)
  // This is a for loop.
  for (q <- QQ) println(q)
  // this is also a for loop
  for (i <- 0 until 14) println(i)
}
```
Coding Style is Important!

**Indentation**

```scala
val someString = 42
val xyz = for (i <- 1 to 5) yield i
val QQ = xyz.map(_ + someString)
```

**Names**

```scala
val QQ = xyz.map(_ + someString)
```

**Comments**

```scala
// This is a for loop.
```

**Consistency**

```scala
for (q <- QQ) println(q)
```

**Braces**

```scala
for (i <- 0 until 14) println(i)
```

**Return values**
Coding Style is Important!

- Indentation
- Names
- Comments
- Consistency
- Braces
- Return values

```scala
def doThings() = {
  val someString = 42
  val xyz = for (i <- 1 to 5) yield i
  val QQ = xyz.map(_ + someString)
  // This is a for loop.
  for (q <- QQ) println(q)
  // this is also a for loop
  for (i <- 0 until 14) println(i)
}
```
Coding Style is Important!

Indentation
Names

Comments

Consistency
Braces
Return values

def doThings() = {
    val someString = 42
    val xyz = for (i <- 1 to 5) yield i
    val QQ = xyz.map(_+someString)
    // This is a for loop.
    for (q <- QQ) println(q)
    // this is also a for loop
    for (i <- 0 until 14) println(i)
}
Coding Style is Important!

**Indentation**

**Names**

**Comments**

**Consistency**

**Braces**

**Return values**

```scala
def doThings() = {
  val someString = 42
  val xyz = for (i <- 1 to 5) yield i
  val QQ = xyz.map(_+someString)
  // This is a for loop.
  for (q <- QQ) println(q)
  // this is also a for loop
  for (i <- 0 until 14) println(i)
  5
}
```
Coding Style is Important!

- Indentation
- Names
- Comments
- Consistency
- **Braces**
- Return values

```scala
def doThings() = {
  val someString = 42
  val xyz = for (i <- 1 to 5) yield i
  val QQ = xyz.map(_ + someString)
  // This is a for loop.
  for (q <- QQ) println(q)
  // this is also a for loop
  for (i <- 0 until 14) println(i)
}
```
Coding Style is Important!

- **Indentation**
- **Names**
- **Comments**
- **Consistency**
- **Braces**

**Return values**

```scala
def doThings() = {
  val someString = 42
  val xyz = for (i <- 1 to 5) yield i
  val QQ = xyz.map(_+someString)
  // This is a for loop.
  for (q <- QQ) println(q)
  // this is also a for loop
  for (i <- 0 until 14) println(i)
}
```
Coding Style is Important!

**Indentation** - indent bracketed code uniformly

**Names** - give variables *semantically* meaningful names

**Useful comments** - convey the “why” not the “what”

**Consistency** - *many* ways to express concepts, pick one and be consistent

**Braces** - like indentation, braces are not required, but can help avoid bugs

**Return values** - clearly indicate them
Coding Style is Important!

Indentation - indent bracketed code uniformly.

Names - give variables semantically meaningful names.

Useful comments - convey the "why" not the "what".

Consistency - many ways to express concepts, pick one and be consistent.

Braces - like indentation, braces are not required, but can help avoid bugs.

Return values - clearly indicate them.

This isn’t just for our benefit…you will be reading your code more than anyone! Help yourself out!
Some Best Practices

- **Never** start with code!
  - Plan out what you are trying to do
  - Think about the bigger picture first
- Figure out what you have. How is it structured?
  - Draw (on real paper) diagrams
  - Construct examples
- What do you want to get, and how should that be structured?
  - Same as above
- How do you get from one to the other?
  - Connect the diagrams
  - Pseudocode!!! (break the big problem down into smaller ones)
What if you get stuck?

- Explain *exactly* what you have tried
  - Which test cases fail? How do they fail? Have you written your own?
  - What other things have you tried which don’t work?
- Explain what you are trying to accomplish and why
  - Context matters
  - Sometimes figuring out the what and the why can already uncover misunderstandings
- Follow coding style guidelines! It will be easier to help you.
- WRITE TESTS!!!
Still stuck?

- **Guarantee**: If you bring us (mostly working) pseudocode, the TAs and I will help you translate it to Scala.

- Translation Challenges:
  - Syntax (e.g., “I don’t know how to break out of a for loop”)
    - Ask on Piazza, Office Hours, Recitation; We will help you!
  - Semantics (e.g., “I don’t know how to insert into a linked list”)
    - Ask, but we’ll ask you to be more precise

- Oftentimes questions about syntax are actually asking about semantics.
Still stuck?

- **Guarantee**: If you bring us (mostly working) pseudocode, the TAs and I will help you translate it to Scala.

- **Translation Challenges**:
  - Syntax (e.g., "I don't know how to break out of a for loop"): Ask on Piazza, Office Hours, Recitation; We will help you!
  - Semantics (e.g., "I don't know how to insert into a linked list"): Ask, but we'll ask you to be more precise.

- Oftentimes questions about syntax are actually asking about semantics.

- Ultimately, you aren’t here to learn Scala. You are here to learn about data structures.

- If Scala is tripping you up, we want to help.
Now...onto some Scala
<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>Binary value</td>
<td>true, false</td>
</tr>
<tr>
<td>Char</td>
<td>16-bit unsigned integer</td>
<td>‘x’, ‘y’</td>
</tr>
<tr>
<td>Byte</td>
<td>8-bit signed integer</td>
<td>42.toByte</td>
</tr>
<tr>
<td>Short</td>
<td>16-bit signed integer</td>
<td>42.toShort</td>
</tr>
<tr>
<td>Int</td>
<td>32-bit signed integer</td>
<td>42</td>
</tr>
<tr>
<td>Long</td>
<td>64-bit signed integer</td>
<td>42l</td>
</tr>
<tr>
<td>Float</td>
<td>Single-precision floating-point number</td>
<td>42.0f</td>
</tr>
<tr>
<td>Double</td>
<td>Double-precision floating-point number</td>
<td>42.0</td>
</tr>
<tr>
<td>Unit</td>
<td>No value</td>
<td>()</td>
</tr>
</tbody>
</table>
Every Expression has a Type

Optionally, you can annotate anything with “: type”

- Variables (declare the type)
- Functions (declare the return type)
- Parenthesized arithmetic

Anything you don’t annotate, Scala will try to infer

```scala
val cost: Float = (7 / 2.0).toFloat
val income = 15 + 10.2 * 9.3f
```
Every Expression has a Type

Optionally, you can annotate anything with “: type”
● Variables (declare the type)
● Functions (declare the return type)
● Parenthesized arithmetic

Anything you don’t annotate, Scala will try to infer

```scala
val cost: Float = (7 / 2.0).toFloat // Float
val income = 15 + 10.2 * 9.3f // Double
def howCute(x: Int) = "Aw" + "w" * x // Int => String
```
Inconsistent Types

val indicator = if (x > 0) { "positive" * x } else { -1 }

What is the type of indicator?
A: String
B: Int
C: Any
D: AnyRef
Inconsistent Types

```scala
val indicator = if (x > 0) { "positive" * x }
else { -1 }
```

What is the type of `indicator`?
A: String  
B: Int  
C: Any  
D: AnyRef

Answer: C  
The if clause is a String (AnyRef)  
The else clause is an int (AnyVal)
Inconsistent Types

```scala
val indicator = if (x > 0) { "positive" * x }
               else { -1.toString }
```

Now the type of `indicator` is `String`
What is the return value of this horrific block of code?

def doThings() = {
    val someString = 42
    val xyz = for (i <- 1 to 5) yield i
    val QQ = xyz.map(_ + someString)
    // This is a for loop.
    for (q <- QQ) println(q)
    // this is also a for loop
    for (i <- 0 until 14) println(i)
    5
}
def doThings() = {
  val someString = 42
  val xyz = for (i <- 1 to 5) yield i
  val QQ = xyz.map(_+someString)
  // This is a for loop.
  for (q <- QQ) println(q)
  // this is also a for loop
  for (i <- 0 until 14) println(i)
}
Assignments using Blocks

val point = { val x = 10; val y = 20; (x,y) }

val name = {
    val first = “Eric”
    val last = “Mikida”
    first + “ “ + last
}
Assignments using Blocks

```scala
val point = { val x = 10; val y = 20; (x, y) }

Value of point: (10, 20)

val name = {
    val first = "Eric"
    val last = "Mikida"
    first + " " + last
}

Value of name: "Eric Mikida"
```
Assignments using Blocks

val point = { val x = 10; val y = 20; (x, y) }

(notice the semicolons for the single-line assignment)

val name = {
    val first = "Eric"
    val last = "Mikida"
    first + " " + last
}
## Mutable vs Immutable

<table>
<thead>
<tr>
<th>Mutable</th>
<th>Immutable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Can be changed</td>
<td>Cannot be changed</td>
</tr>
<tr>
<td><code>var</code> variable that can be</td>
<td><code>val</code> value that cannot be</td>
</tr>
<tr>
<td>reassigned</td>
<td>reassigned</td>
</tr>
</tbody>
</table>

*Mutable state is more flexible (can but updated), but it is harder to reason about!*
val set = mutable.Set(1,2,3)
set += 4
Will this work?

val set = mutable.Set(1,2,3)

set += 4

Yes!

After executing this code, set will point to a mutable set containing 1, 2, 3 and 4! The key here is “points to”.

set was assigned a reference that points to a mutable set
We did not change that reference (we followed the rules, set is immutable)
What we changed was the object being referenced
Scala Class Types

- **class**
  - Normal OOP type (instantiate with 'new')

- **object**
  - A ‘singleton’ class; Only one instance

- **trait**
  - A ‘mixin’ class; Can not be instantiated directly

- **case class**
  - Like class, but provides bonus features
Companion Objects

An object with the same name as a class (in the same file)

- Defines global (static) methods for that class
- Useful, for example, to avoid directly using 'new'

```scala
class Register(val x: Int) {
  def addValue(y: Int) = x + y
}
object Register {
  def apply(x: Int) = new Register(x)
}
val reg = Register(10) ← Creates a new register instance
```

*Syntactic sugar: In Scala* `foo(x)` *is the same as* `foo.apply(x)`