Opening Discussion

- Minute essay comments:
  - First element that satisfies condition.
  - Other Scala books.
  - Midterm issues.
  - Code on tests.
  - Why can you beat the top level in Super Smash Bros?
  - What is the best way to memorize a PL?
  - Why did I cut my hair?
  - Do I watch “Game of Thrones”? 
Putting things together moving from REPL to script.
Lots of ways to write things.
DNA sequencing and Dr. Hibbs.
Looking at a “real” program.
Knowing the difference between foldLeft and reduceLeft in “real life”.
Why use zip?

IcP Solutions
Let's Put These Into Action

- I want to spend some class time playing with these methods and seeing what we can do with them.
- A String is a collection so you can do these things with a String as well.
- String also has a method called split.
- BLS data
Variable Length Argument Lists

- You can make functions that don't specify exactly how many arguments they take.
- These are often called var-args.
- To do this, but a * after the type. It can only be the last argument in a list.
It is often helpful to call a var-args method passing a collection for the variable length arguments.

You can do this, but you have to tell Scala what you are doing.

Follow the collection with :_* to do this.

The : is like specifying a type.

The _ says you don't care about the exact type.

The * is like the * in var-args declarations.
I argue that immutable collections like Lists can be safer than mutable ones like Arrays.

One of the big reasons for this is aliasing.

An alias in programming is just like in normal life. It is a second name for something.

Variables are really references to objects.

If a second variable is assigned the same value as the first, they are aliases to that object.

Let's play with this and draw on the board.
When you pass arguments, you are really passing references.

So arguments in functions are aliases to the objects outside the function.

If the object is mutable, the function can change it.
There is another way to pass things in Scala called pass-by-name.

When you pass something by name, it isn't evaluated at the time it is passed. Instead it is turned into a function and that function is evaluated every time the variable is used.

The syntax is to put an => before a type, but not have an argument list before the arrow.
Fill and Tabulate

- There are two other ways of creating collections: fill and tabulate. Both are curried. Second argument to fill is by name, second argument to tabulate is a function.

- The fill method on Array or List takes a first argument of how many elements. After that is a by-name parameter that gives back the type you want in the array or list.

- Tabulate also takes a size first. After that is a function that takes the index.
Recursion is sufficient for making repetition, but in imperative languages it isn't the normal approach. Instead, people use loops.

The simplest loop is the while loop.

- `while(condition) statement`

The condition is evaluated first. If it is true the statement (possibly a block) executes.

This repeats until the condition is false.
do-while Loop

- The partner to the while loop is the do-while loop.
  - do {
    - statement
  } while(condition)
- This loop is post-check instead of the pre-check of the normal while loop.
- Always happens once.
- The while loop might never happen.
What questions do you have?