

Unit Testing

Live Examples

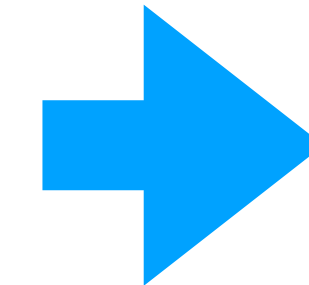
Lecture Task

- This is Lecture Task 4 from the Pale Blue Dot project -
 - In the `pbd.PaleBlueDot` object, write a method named "bigCities" which:
 - Takes three Strings and an Int as parameters representing:
 - The name of a file containing city data. ex. "data/cities.csv"
 - A two character country code. You may assume that the input is in all lowercase letters
 - A region code in the same format as they appear in the cities file
 - The minimum population of a city
 - Returns a List of all the city names in the given country/region with at least the minimum population.
 - In `tests.LectureTask4`, complete a test suite to test the functionality of this method
 - Note: All the Strings will be in the same format as they appear in the cities file. You don't not have to do anything with upper/lower case for this objective

Testing Doubles

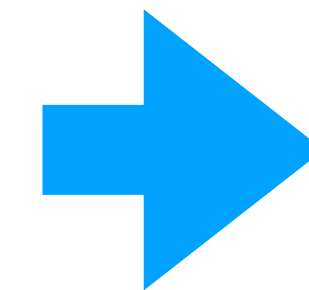
- **Never** use `==` to compare two Doubles
- Check if the difference between the Doubles is less than a *small* value
 - Small enough to not interfere with your logic
 - Large enough to ignore truncation errors

```
val b: Double = 0.1
val c: Double = b * 3
val expected: Double = 0.3
assert(c == expected)
```



Fails

```
val epsilon: Double = 0.00000001
val b: Double = 0.1
val c: Double = b * 3
val expected: Double = 0.3
assert(Math.abs(c - 0.3) < epsilon)
```

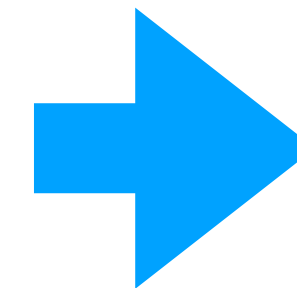


Passes

Testing Maps

- Use `==` to compare two Maps
- Scala will check if each key exists in both maps and map to the same value in both
- Order does not matter in key-value data structures

```
val map1: Map[Int, Int] = Map(  
  1 -> 15,  
  2 -> 20,  
  3 -> 25  
)  
  
val map2: Map[Int, Int] = Map(  
  2 -> 20,  
  3 -> 25,  
  1 -> 15  
)  
  
assert(map1 == map2)
```



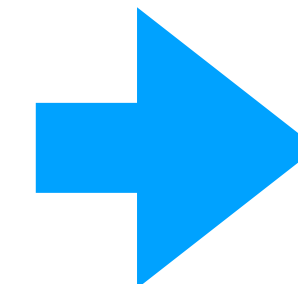
Passes

Testing Lists

- Use `==` to compare two Lists
- **Order matters in lists!**
 - Scala will check if both lists contain the same elements **in the same order**
- If you only care about the values, not the order, **sort** both lists before comparing

```
val list1: List[Int] = List(1, 2, 3)
val list2: List[Int] = List(2, 3, 1)

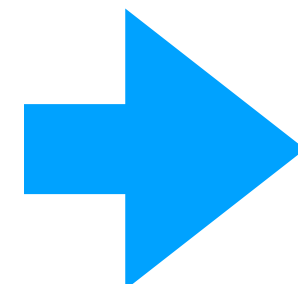
assert(list1 == list2)
```



Fails

```
val list1: List[Int] = List(1, 2, 3)
val list2: List[Int] = List(2, 3, 1)

assert(list1.sorted == list2.sorted)
```



Passes

Example

- **Task:** Write a method named `isAnagram` that takes two `Strings` and returns a `Boolean` that is `true` if the `Strings` are anagrams of each other, ignoring case and spaces, and `false` otherwise

`isAnagram("Astronomer", "Moon Starer") == true`

- **Testing:** Write a test suite to test this functionality
- Live Walkthrough

Recap of isAnagram

- When comparing case-insensitive Strings
 - Call toLowerCase on each String
- Using a Map to store test cases can keep your testing code organized
- How much testing is enough?
 - At a minimum, complete the AutoLab portion of the assignment
 - More testing will often be required to ensure correctness
- Your testing should be thorough enough that you are confident that your code is correct if it passes all your testing
 - Convince yourself that your testing is complete

Example

- **Task:** Write a method that takes a String and returns a List of all the anagrams of the input
- **Testing:** Write a test suite to test this functionality
- Live Walkthrough

Recap of Anagrams

- Comparing Lists
 - Can use ==
 - Elements and order must match
 - Can sort the Lists if the order is not important
- It will not always be easy to know that a method is correct
 - My method should be very difficult for you to read at this point in your career
- How will you be confident that my code is correct on all inputs?
 - Thorough unit testing!
- How will you be confident that code you write is correct on all inputs?
 - Thorough unit testing!

Lecture Task

- This is Lecture Task 4 from the Pale Blue Dot project -
 - In the `pbd.PaleBlueDot` object, write a method named "bigCities" which:
 - Takes three Strings and an Int as parameters representing:
 - The name of a file containing city data. ex. "data/cities.csv"
 - A two character country code. You may assume that the input is in all lowercase letters
 - A region code in the same format as they appear in the cities file
 - The minimum population of a city
 - Returns a List of all the city names in the given country/region with at least the minimum population.
 - In `tests.LectureTask4`, complete a test suite to test the functionality of this method
 - Note: All the Strings will be in the same format as they appear in the cities file. You don't not have to do anything with upper/lower case for this objective