Testing

# Lesson Notes

## How do we know if our code is correct?

* So far we have been looking at variables in the debugger
* What happens when we make changes to a program?
* What happens when our program is larger?
* There has to be a better way!

## Test code

* Test code is code that verifies other code
  + Do our programs produce the correct result?
* For example, our school example:
  + We already wrote a function to count students
  + We can write another function to make sure our function produces the right answer
* For example:
  + function testCountingOfStudents() {
  + var result1 = countStudentsInSchool(testSchool1);
  + if (result1 != 100) {
  + return false;
  + }
  + var result2 = countStudentsInSchool(testSchool2);
  + if (result2 != 200) {
  + return false;
  + }
  + var result3 = countStudentsInSchool(testSchool3);
  + if (result3 != 200) {
  + return false;
  + }
  + return true;
  + }
* In the above example, the test function returns true or false.
* We have picked a couple of tests schools
* If there is a problem, we still have to step through the debugger to find the problem case.
* There is a better way!

## Assert

* The word “assert” is short for “assertion”
  + Definition: <http://www.google.com/#q=define+assertion&fp=1>
* An assert is a function that expects its parameter to be true
  + If the parameter is not true, the assert function does something that makes it easy to debug the problem
* So:
  + function assert(condition) {
  + if (!condition) {
  + // something!
  + }
  + }

## The debugger statement

* Javascript has a debugger statement that will cause the program to stop in the debugger if the debugger is running
* Let’s add that to our assert function:
  + function assert(condition) {
  + if (!condition) {
  + debugger;
  + }
  + }
* Show example:
  + We can step past the debugger, back to the place that called the assert function
  + We now know what the problem is, and we can debug it.
* Set next statement
  + To move back to the line that caused the problem
  + Execute it again to see the problem
  + Avoids having to rerun all the tests

## Some advantages to testing

* We can test each part of our program separately
  + **Unit testing**
* When we assemble the parts, they have a better chance of working
  + **Integration testing**
* When we make changes to our program, we can quickly see if something was broken
  + **Regression testing**

## Pitfalls with testing

* Your test may have a bug
  + It is just software after all.
* You may not have picked the right cases to test.
  + *“Program testing can be a very effective way to show the presence of bugs, but is hopelessly inadequate for showing their absence.”   
    – Edsger W. Dijkstra,* [*The Humble Programmer*](http://www.cs.utexas.edu/~EWD/transcriptions/EWD03xx/EWD340.html)

## Program verification

* We can also use mathematical logic to verify our programs
  + Result is a formal proof that our algorithm should be correct
  + Can be difficult time consuming, especially for larger programs
  + Still doesn’t guarantee that there aren’t bugs
  + “Beware of bugs in the above code; I have only proved it correct, not tried it.”
    - Donald Knuth, American Computer Scientist
    - <http://www-cs-faculty.stanford.edu/~knuth/faq.html>
* This approach tends to be used less often than testing

## Exercise

* Add tests to one of your existing programs
  + Add the assert function
  + Write test cases for your program
  + Introduce a bug into your algorithm
  + Use the debugger to find the bug
  + Fix the bug
  + Use the debugger to verify that the bug is gone