Unit Testing and Errors
HW2: Pig

• A simple dice game between a human and a computer player

• Mostly testing functions

• Also test error checking
  – Your code is not allowed to crash for any inputs

• Due Tuesday, September 19, 11:59pm
Software Failures
Amusing Software Failures

Greetings from M. Gandhi, ruler and king of the Indians...
Our words are backed with NUCLEAR WEAPONS!
HeartBleed

SERVER, ARE YOU STILL THERE? IF SO, REPLY "HAT" (500 LETTERS).

User Meg wants these 500 letters: HAT.

Lucas requests the "missed connections" page. Eve (administrator) wants to set server's master key to "14835038534". Isabel wants pages about snakes but not too long. User Karen wants to change account password to "CalleR4t" User Isabel requests pages.
Mars Climate Orbiter
NATS System Failure

Nats: computer failure behind London airport chaos was unprecedented

British airspace controller says inability of operators to see all flight plan data prompted restrictions on departures and landings
Therac-25

● Radiation Therapy Machine developed by Atomic Energy of Canada Limited

● 6 cases of massive radiation overdose
  – 1000s of times the normal dosage
  – Painful electric shock feeling
  – 3 Deaths as a result
Error rate

• Let’s say you code so well you only generate 1 error every 1000 lines
  – Think about your HW1 error rate
  – This is a pretty good error rate

• Facebook has about 60 million lines of code
  – 60,000,000 lines / 1000 line per error = 60,000 errors
  – Is that good enough?
  – You tell me...

• Of course, most programs are a lot smaller than Facebook
  – The space shuttle’s primary software, for example, is only 400,000 lines
Testing

• Unit Testing
  – Test over a small section of code
  – Very “traceable”
  – Limited to finding “easier” errors
  – We’re talking about this today

• Integration Testing
  – Test the interaction between two systems
  – Tells you which pieces are working together well
  – We aren’t talking about this today
Testing

• Acceptance Testing
  – Is the application doing what the customer wants?
  – Often, acceptance testing is the final step in a software contract before initial delivery

• Regression Testing
  – Testing older portions of code
  – If a newer update breaks an older section, you can trace which update broke it
Why Test Driven Development

• Because you will catch bugs early
• Because it gives you a set of regression tests
• Because it ultimately will save you time

• “In examining conventionally scheduled projects, I have found that few allowed one-half of the project schedule for testing, but that most indeed spend half of the actual schedule for that purpose.”

-- Frederick P. Brooks, Jr.
Division of Labor

• Programs are written as a large collection of small functions
• A function should either do computation or do input/output, not both.
  – Following this rule makes computational functions reusable in other contexts
  – More important for this talk, following this rule makes testing computational functions easier
  – If a function under test requests input from the user, the function can’t be tested with a “single click”
  – If a function under test produces output, the output must be examined for correctness
Structure of test file

1. Import unittest
   - The `import` statement allows you to use functions in a given module
   - Seen math module so far
2. Import the program you want to test
3. Write test methods

```python
import unittest
from name_of_module import *
class NameOfClass(unittest.TestCase):
    # More about classes later
    # Think of it as a collection of functions
    # Test methods go here!

unittest.main()
```
Structure of test methods

• Each test has a name beginning with test_ and has one parameter named self

• Inside the test function is just normal Python code
  – Avoid input or output
  – I/O in tests will just slow down testing and make it more difficult
  – For the same reason, the code being tested should also be free of I/O

• Here are the three most common tests you can use:
  – `self.assertTrue(boolean_expression_that_should_be_true)`
  – `self assertFalse(boolean_expression_that_should_be_false)`
  – `self.assertEqual(first_expression, second_expression)`
  – Of these, `self.assertEqual` gives you more information when it fails, because it tells you the value of the two expressions

• IDLE example: TestStartFunctions.py
Be careful with floating points

• We want to see if two floating points are “equal enough”
  – This is that whole “epsilon” nonsense I talked about

• You can’t trust floating point computations
  – E.g., 1.3 * 1.5 = 1.95
  – It is 1.9500000000000002 in python!
  – The two are “equal” but are not ==
Testing Strategy

• Pick a method that doesn't depend on other, untested methods
  – Write and test one feature at a time
  – May write test before code (get conceptual input/output understanding)
  – If test fails, change just enough code to make it pass (unless you spotted bigger issues)
  – Refactor the code to make it cleaner
  – This strategy called **Test Driven Development**

• Test edge cases!
  – Largest/smallest possible inputs
  – Negative numbers, zero

• **Remember**: you are not trying to **prove** that your code is correct – you are trying to **find and expose flaws**, so that the code may be fixed
“I’ve finished the coding but I haven’t tested it yet”

• I hear this from students all the time.
  – This translates to: “I haven’t finished yet, but I also don’t realize how screwed I am”

• Writing coding takes 90% of the time
  – The other 90% is spent testing

• Writing unit tests is mandatory in this course!
  – It is annoying but you can’t get away without it in big programs
  – How many unit tests you write is sometimes a matter of judgment
  – **Rule**: Test every case you can think of that might possibly go wrong