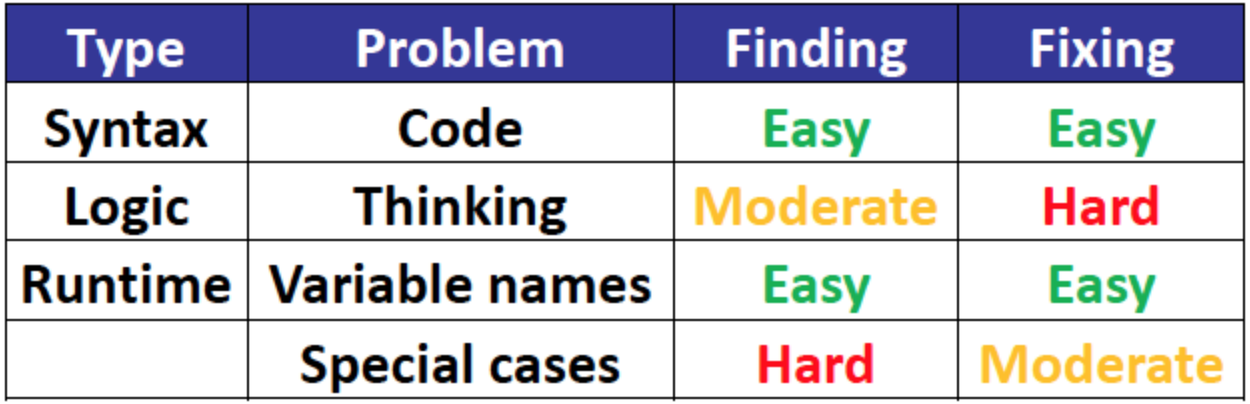
Python Debugging and Testing

www.cs.uoregon.edu/Classes/15U/cis122

Welcome Back!

A Quick Recap

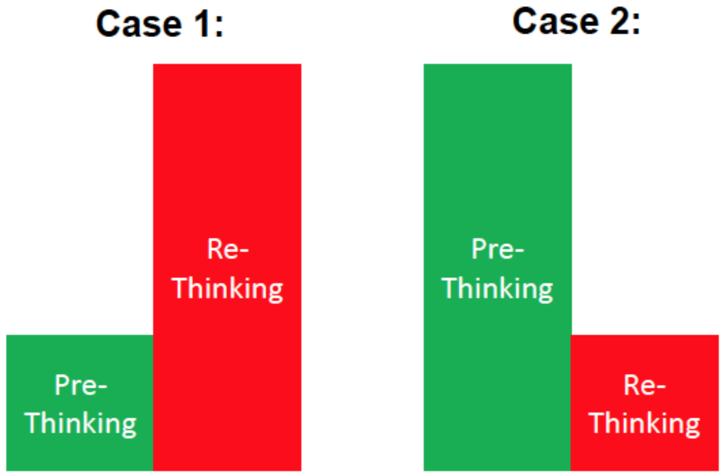


Type	Problem	Finding	Fixing
Syntax	Code	Easy	Easy
Logic	Thinking	Moderate	Hard
Runtime	Variable names	Easy	Easy
	Special cases	Hard	Moderate

Prethinking v/s Rethinking

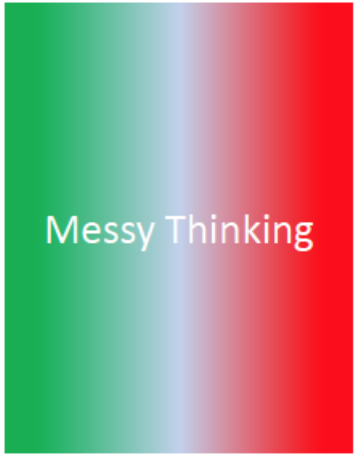
You basically have two options when programming:

- you can spend time thinking *before* coding, or
- you can spend time rethinking *after* coding (i.e. debugging)



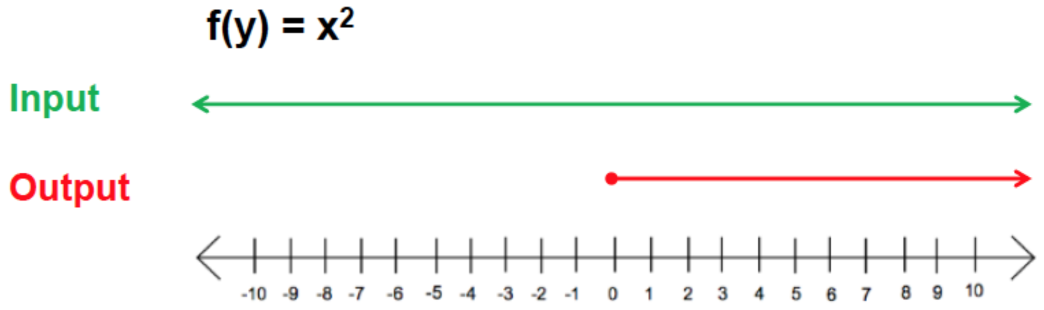
The 3rd Option

The third option is slightly more messy. This is when you do double the work.



Input and Solution Spaces

Functions (in the mathematical sense) map inputs to outputs (or solutions).



Similarly..

Computer science functions also map inputs to solutions but we often have to give more thought to carefully evaluating a variety of inputs and how they might map to unexpected values or errors.

```
def foo(x):
        y = x ** 3
        y /= x - y
        return y
```

what values might be an issue?

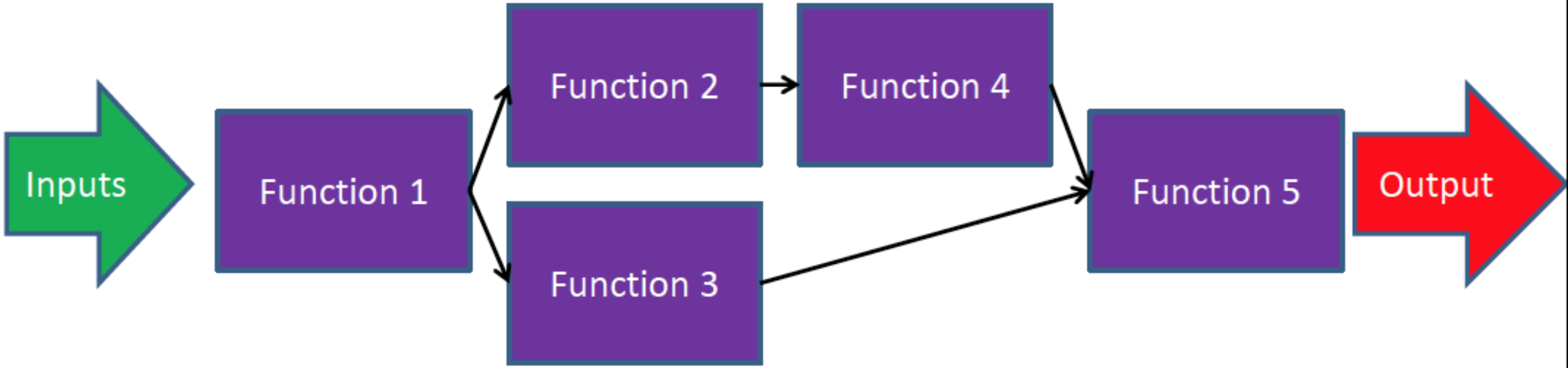
Can we test exhaustively?

Test Cases

If we can't test exhaustively (and we can't) what can we do?

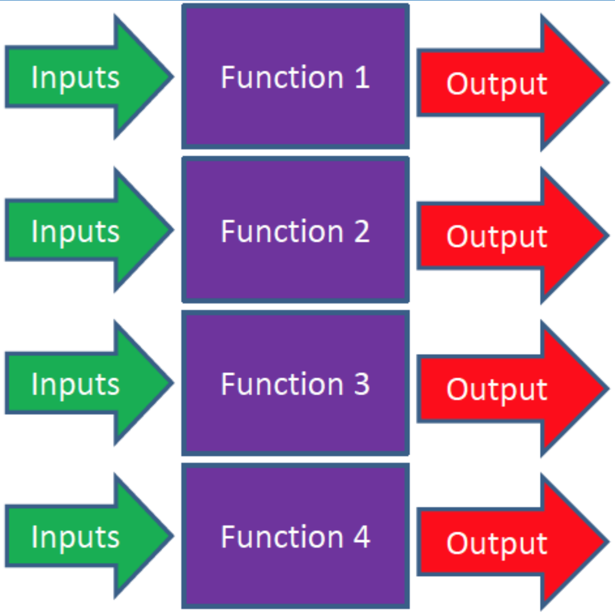
Choose a good assortment of test cases. We'll discuss this more tomorrow.

Integrated Testing



with *integrated* testing we test the entire run of the program, or at least a major section of it, potentially testing dozens or hundreds of individual function calls at one time.

Unit Testing



With Unit Testing, we check each function/module individually.

But, Which one do we use?

The answer is almost always **both**.

Unit testing lets you isolate problems *within* specific functions that integrated testing might notice but have trouble pinning down ("somewhere my program went bad but I don't know if it was in function 2, 4, or 5"). Unit testing is also good for testing for edge or corner cases that may cause problems.

Integrated testing lets you find problems *between* functions (i.e. mismatches of outputs to inputs) which unit testing is oblivious to. Example: function 1 returns a string as an output which is used as the input of function 2, but function 2 was written expecting an int input. Both functions work in isolation but when tested as an integrated whole they break down.

Excercise

```
import turtle

def line_and_turn(length, angle):
        turtle.fd(length)
        turtle.rt(angle)

def poly(sides, size):
        for i in range(sides):
                line_and_turn(size, 360/sides)

def turtHouse():
        poly(4, 100)
        poly(3, 100)
```

how do we test this?

So in keeping with a theme: randomness

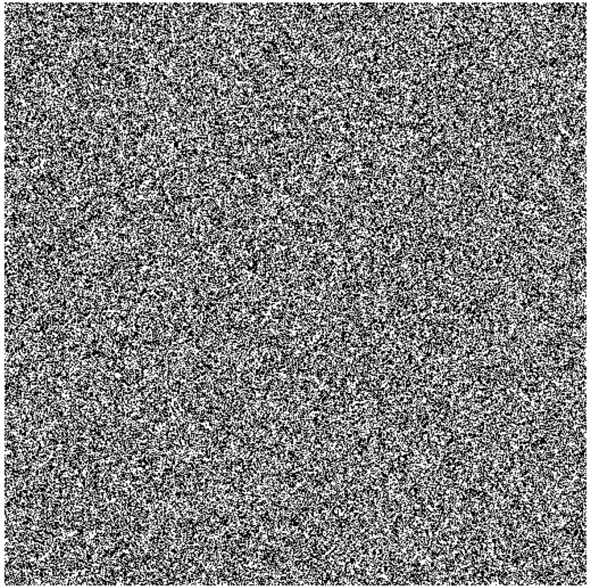
Random Module

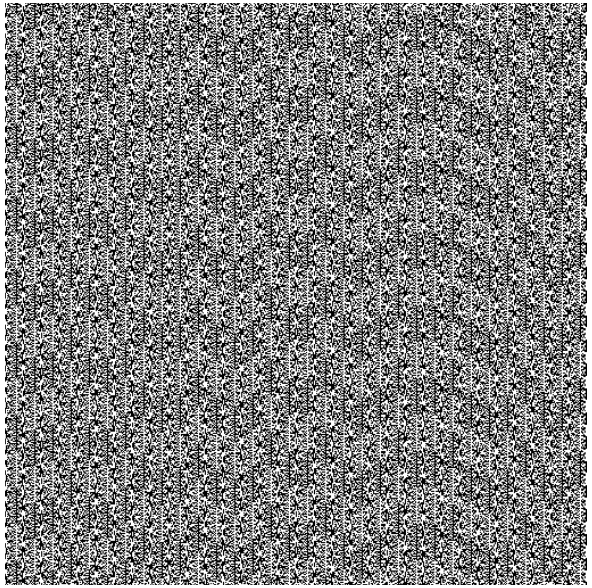
Python's random module generates pseudorandom numbers for us (and has other functionality).

```
import random
print( random.randint(0,10) )
print( random.randint(0,10) )
print( random.randint(0,10) )
```

generates a random int between 0 and 10 inclusive of both (unusual in python).

Pseudorandomness





images from: http://boallen.com/random-numbers.html

The difference

Pseudorandom numbers are sequences mathematically generated to approximate a random distribution of numbers, but they are actually deterministic based on a specific starting seed.

The seed is a number or vector used for initialization of the process. Enter the same seed you get the same sequence of numbers.

Ever play a computer game where if you take a turn some stuff happens, but if you save first and then take the turn multiple times the same random events happen every time? Welcome to pseudorandom.

Seeds

Speaking of seeds

```
 import random
 random.seed(1)
 print( random.randint(0,10) )
 random.seed(1)
 print( random.randint(0,10) )
```

Not exactly random.

seed() initializes the seed used by the pseudorandom number generator. If called with no argument it uses the current system time.

randrange()

randrange() works very similar to randint() except the end is exclusive (i.e. below a 10 cannot be generated)

```
 import random
print( random.randrange(0,10) )
 print( random.randrange(0,10) )
 print( random.randrange(0,10) )
```

generates a random int between 0 and 10 inclusive of 0 but not 10. Let's test that…

(printing is sloooooooooooooow)

choice

choice selects one member of a sequence pseudorandomly

```
import random
 str1 = "abcd"
 print( random.choice(str1) )
 print( random.choice(str1) )
 print( random.choice(str1) )
```

each object has *essentially* the same chance of being picked.

sample

similar to choice but can select multiple times from the sequence returning the result as a list

```
import random
str1 = "abcd"
print( random.sample(str1, 3) )
```

each object has *essentially* the same chance of being picked.

sample/choice+range

remember range() gave us a sequence of number?

```
import random print( random.choice( range(1000000000000) ) )
```

(you could do the same thing with randint() but I believe this method is faster)

Generate your own random strings

Remember the string constants?

string.ascii_letters

string.digits

string.punctuation

We can use these now to generate random strings!