# Lecture 4

## 0 Queries

## 1 Main Motivation and TDD

Make a file `funcs.py`. Make two functions:

```
def f(x): # f(x) = 7x^2 + 2x
    pass

def g(x, y): # g(x, y) = (x^2+y^2)/(3x)
    pass
```

Now I can write and run the tests without having actually written the functions! Let's write some tests!

```
import unittest
import funcs

class Tests(unittest.TestCase):
    def test_f1(self):
        self.assertEqual(funcs.f(0), 0)

    def test_f2(self):
        self.assertEqual(funcs.f(-2), 24)

    def test_f3(self):
        self.assertEqual(funcs.f(4), 120)

    def test_g1(self):
        self.assertAlmostEqual(funcs.g(1, 0), 0.33333333)

    def test_g2(self):
        self.assertAlmostEqual(funcs.g(2.3, 4.7), 3.968115942)

    def test_g3(self):
        self.assertAlmostEqual(funcs.g(-3, -1), -1.11111111)

if __name__ == '__main__':
    unittest.main()
```

At this point all the tests will fail. That's fine! They should! We never wrote them! But now we can write the functions little by little and we'll know we're done when all the tests pass! So now we write `f`. (Do this.) Before even running the tests, we may want to verify that the function kind of does what we expect, a sanity check of sorts. So let's add a **print** call. Okay, it looks like its working. Let's run the tests!

Uh oh! There's some garbage being printed in my testing output! That's our print! Let's put the prints in `main`!

## Mutation

Variables can change values! Think of `x` as a box that can only contain one value. (Draw this!)

```
>>> x = 5
>>> print(x)
5
>>> x = 10
>>> print(x)
10
>>> x = x + 1
>>> print(x)
11
>>> x += 1
>>> print(x)
12
>>> x -= 5
>>> print(x)
7
```

## 2 Scope

Draw out the variables in different scopes in `scope.py`. Also probably mention the decimal format specifier.

## 3 Tracing Code

Go through `weird.py` one line at a time noting the variables at each line. Put variables in parentheses if they go out of scope (and remove them entirely when they no longer exist). (Have code/tracing paper printed out!)

```
x: 8.5
y: hello

x: 5
y: hi
```

## 4 Loops

What if I want to write code that repeats? Let's say I want to print out the numbers from 10 $\searrow$ 1. I could do something like:

```
print(10)
print(9)
⋮
print(1)
```

But... that's a lot of typing. I don't want to do that... Instead, we can use loops!

```
counter = 10
while counter > 0:
    print(counter)
    # What would go wrong if I stopped here?
    counter -= 1
```