Testing

- Idea: should think about how you'll know when your code is correct
- really the thinking should happen -before- you code
- think about the different scenarios that can come up
- think about possible inputs and outputs -- what results should you expect

Testing Categories

- Black box testing: tests functionality without knowing internal workings/structure
- White box testing: testing knowing complete design and implementation details
- Grey box testing: testing with limited/incomplete knowledge of internal/development details

Types of Testing

- Unit Testing:
  - testing from a programmer's perspective
  - testing that the each part of the code does the right things
- Functional Testing:
  - testing from a user's perspective
  - testing that the product functions as users expect
  - integration testing: making sure it interacts appropriately with dependent modules

Unit Testing

- Testing each small piece of code individually
- Means you can easily see if you've broken anything
- Not just about 1 unit test per method/function
  - about making sure that you fully test your method
- Edge cases: things that only come up some of the time
  - what happens if something is empty?
  - going out of bounds?
  - ...?

unittest

- built-in python library for unit testing
- Create a class for testing:
  - inherits from `unittest.TestCase`
  - may have more than one depending on scope of project
  - may be related to one another through inheritance
  - class contains methods representing individual, isolated tests
    - each test method name must begin with `test`
    - inside test use assertions that will cause it to fail if not true

assertions

- built-in methods in the class (you don't have to define them)
- For example:
  - `assertEqual(a,b)` and `assertNotEqual(a,b)`
  - `assertTrue(x)` and `assertFalse(x)`
  - `assertIsNone(x)` and `assertIsNotNone(x)`
  - `assertIn(a,b)` and `assertNotIn(a,b)`
  - `assertIsInstance(a,b)` and `assertNotIsInstace(a,b)`

unittest + exceptions

- special assertions that look for exceptions being raised
- `assertRaises(exceptionname, callable, -args, --kwargs)`

```
def foo(x,y):
    if y == 0:
        raise ValueError
    return x/0

class MyTests(unittest.TestCase):
    def test1(self):
        self.assertRaises(ValueError, foo, 5)
```

unittest + exceptions (cont.)

- can also use context
- `with self.assertRaises(exceptionname):`

```
def foo(x,y):
    if y == 0:
        raise ValueError
    return x/0

class MyTests(unittest.TestCase):
    def test1(self):
        with self.assertRaises(ValueError):
            foo(5,0)
```