7 Testing

Key terms: `assert unittest`

Reading: Mike's Python 3 Testing: An Intro to unittest[16]

Exercise: Use assertions and `time.sleep()` to confirm that `/proc/driver/rtc` keeps good time.

Validations are checkpoints that have to be passed for work to proceed. User input can be "wrong" in many situations and we generally have to stop work on impossible requests. `try` blocks are one major way to handle invalid input. The other simple test is the `assert` statement, which raises `AssertionError` if a given expression is `False`. Liberally applied, these can greatly increase confidence. Asserts are not unit tests. They only come up when execution reaches that part of the code, and then only when optimization mode is disabled, and perhaps not in release versions. The immutable built-in `bool` called `__debug__` controls whether the `assert` statements are executed.

```
# Validate that an argument was passed

import sys
assert len(sys.argv)>1
```

Our main topic here is the unit test, which determines whether some aspect of program state is correct. Using preordained rules, they help ensure that a program has not undergone regression (when a program feature ceases to work properly). Tests should not have any side effects, including calling other methods that print output.

We should test units as small as possible, so that the tests are not confounded by any other factors. We also want to test as broadly as possible, to increase coverage. One useful norm is writing unit tests to verify any bug fix on an ongoing basis. In Test Driven Development the tests must be written before the code. We may want to run our tests before and after checking in code. If any one test fails, all tests are aborted. We won't care about what fraction of tests are passing; rather, what matters is whether any one fails: whether the program is in a passing or a failing state.

The framework `unittest` defines the `TestCase` object. Its `main()` method calls the tests, which can be in separate class files, in alphabetical order. `unittest.main()` creates a `TestProgram` object which discovers and runs all the tests (by default, those objects whose names start with `test`). `assertEqual(a,b)` is similar to `assert a==b`. Other mechanisms include `assertTrue` and `assertFalse`. `assertRaises` detects if the correct `Exception` is raised in a given situation.

```
# Define and test an Exponent class:

import unittest

class Exponent():
 def square(x):
  return x**2

class test_exponent(unittest.TestCase):
 def test_square(self):
  self.assertEqual(Exponent.square(-6),36)
  self.assertAlmostEqual(Exponent.square(0),0.001,places=2)
  with self.assertRaises (TypeError):
   Exponent.square('j')

unittest.main(verbosity=2)
```

Next, we will study our programs' means of handling character data.

[16] `https://www.blog.pythonlibrary.org/2016/07/07/python-3-testing-an-intro-to-unittest/`