327: Object-oriented programming

Lecture 11

10/6/2021

Professor Barron

Schedule

Homework 2 due:	October 5th
Homework 3 due:	October 15th
Midterm (in class):	October 18th
Homework 4 due:	November 4th
Homework 5 due:	November 19th
Homework 6 due:	December 10th
Final exam:	December 21th

HW Stats and Regrade requests

- Median: 92
- Mean: 89.18
- Std dev: 13.26

Regrade request procedure

- Regrade requests open at 5pm today on Gradescope
  - Available for 1 week
  - To stay organized, they will be ignored over email or Ed
  - You may come to office hours (with me or Yuyang) to discuss a regrade request, but you should put it into gradescope first
- Must be specific about particular cases and the points you believe you should have earned
  - refer to point breakdown in the assignment and points per test to make your case
  - requests will be declined if they ask for more credit without explaining where or why
- It is possible to lose points from a regrade request if we manually go through it in more detail and find problems that were not accounted for.
  - unlikely for the autograder cases, but could apply to manually graded UML diagrams

Regrade request procedure

- Carefully review failed autograder test cases and UML diagram criteria
- -5 points per formatting issue
  - Can be combined into categories. For example, two typos and an extra \n in the prompt could count as one issue, but would be separate from improperly formatted amounts when listing transactions.
- The most common issues revolve around “double jeopardy”
  - In general you shouldn’t lose points repeatedly for the same mistake unless it directly prevented another part of your program from working
  - For example, a mistake in displaying the currently selected account could cause many tests to fail, but you could go through and count the points that should be given back for tests that ***solely*** due to that issue

Unittest framework

- Test class extends unittest.TestCase
- Each test is a method in that class with a name starting with test_
- Call assertion methods
- Run test from module or use python –m unittest to run all tests

```
1   import unittest
2
3   def average(seq):
4       return sum(seq) / len(seq)
5
6   class TestAverage(unittest.TestCase):
7       def test_zero(self):
8           self.assertRaises(ZeroDivisionError, average, [])
9
10      def test_with_zero(self):
11          with self.assertRaises(ZeroDivisionError):
12              average([])
13
14  if __name__ == "__main__":
15      unittest.main()
```

Unittest framework

- setUp method
  - run automatically before each test
  - create necessary objects or data
- tearDown method
  - run automatically after each test
  - clean up anything that isn’t simply replaced or garbage collected
  - for example, test that reads/writes files

Unittest framework

- Organizing tests
- Group tests into classes based on shared setUp/tearDown

```
ourprog/
    ourprog/
        __init__.py
        db.py
        gui.py
        rules.py
        test/
            __init__.py
            test_db.py
            test_gui.py
            test_rules.py
    setup.py
```

Unittest framework

- setUp and tearDown run before and after every test
- Can also run some code once before all the tests and once at the end

```
5   class TestValidInputs(unittest.TestCase):
6
7       @classmethod
8       def setUpClass(cls):
9           print("setUpClass")
10
11      @classmethod
12      def tearDownClass(cls):
13          print("tearDownClass")
```

Skipping some tests

- Code being tested is not yet implemented
- The test depends on the OS you are using
- Label these tests so they don’t distract from the ones we care about
  - expectedFailure()
  - skip(reason)
  - skipIf(condition, reason)
  - skipUnless(condition, reason)

Pytest

- Another popular testing framework
- Compatible with unittest
- More detailed output
- Does not require object-oriented tests
- Runs any functions named test_*

Unit test mocking

- Monkey patching lets us change methods at runtime
- Unit testing is probably the most common use case
- Fake the behavior of some code to make tests simpler and more independent
  - Interacting with a database
  - Fetching a file from the internet
  - Faking user input
  - Using fixed timestamps
  - Replacing randomness with consistent data