Building Tests for your Python Program

Help Center

Informal testing using testing templates

In this class, we will emphasize the importance of testing your programs. In IIPP, we provided some simple informal testing templates that consisted of a collection of test data coupled with several calls to a function or method. Both the expected output and the computed output from these calls were printed to the console.

For example, this testing template from IIPP helped students determine whether their format function for their Stopwatch mini-project was implemented correctly. Typically, using this template requires users to paste their code into the template and manually compare the computed vs. expected results in the console. In situations where the user is not particularly experienced with testing, this manual approach can be beneficial to the programmer.

Improved testing with the `poc_simpletest` module

As our programs grow more complicated, manual comparison of test results can become tedious and error-prone. To facilitate testing in this class, we have built a small Python module called `simpletest` that automates the creation and reporting of testing results. The source for this module is available here. The module consists of a single class `TestSuite` that includes three methods:

- `__init__(self)` which creates an empty TestSuite object.
- `run_test(self, computed, expected, message = "")` runs a test comparing `computed` to `expected`. If they differ, an error message with header `message` is printed to the console.
- `report_results(self)` reports a summary of the result of running the various tests in the test suite.

An example

Using `poc_simpletest`, it's possible to create fairly complete test suites for your programs. This test suite is designed to test this format function for the stopwatch project. Note that the test suite is imported into the solution file in line 5 and run in line 21. Inside the test suite, the test code is encapsulated inside a single test function `run_suite()` that includes the function being tested as a parameter. This design allows us to import the test suite using a single `import` statement and then run the test suite using a single call to `run_suite()`. One can turn off testing by simply commenting out the call to `run_suite()`. (For information on how importing works, please this page.)

Unit testing using the `unittest` module

Python supports quite sophisticated testing using the `unittest` module described here. We will leave exploration of the substantial capabilities of `unittest` for later.

Created Wed 16 Apr 2014 3:22 PM PDT

Last Modified Sat 21 Feb 2015 3:12 PM PST