Testing & Efficiency

CS 339R (Python) — Chapter 11

Spring 2011
Unit Testing

• What’s a Unit?

• Whatever you think it is:
  • module
  • class
  • Something that has a cohesive existence
What’s a Unit Test?

- What a programmer does to verify two things:
  - “I understand the requirements”
  - “My code meets those requirements”

- Avoids hand-waving and other Nasty Things
  - “All my tests pass”

- Should be done many times during development
  - After each logical code change
What Else is a Unit Test?

• Code
  • in its own module
  • you import the module that you want to test

• A collection of boolean statements
  • let the computer do the testing!

• Some people write tests before the code to be tested
  • “Test-First Development”
The unittest Module

- Defines the **TestCase** class
- You *inherit* from it
- And provide *test methods*
  - names start with "**test**"
  - test methods take no arguments
  - contain calls to **TestCase**’s *boolean* methods

- See `rattest.py`
unittest Methods

- `assert_(cond)`
- `assertAlmostEqual(arg1, arg2, places=7)`
- `assertEqual(arg1, arg2)`
- `assertNotAlmostEqual(arg1, arg2, places=7)`
- `assertNotEqual(arg1, arg2)`
- `assertRaises(excType, f, *args, **kwargs)` (Exception of excType is expected)
- `fail()`
- `failIf(cond)`
- `setUp()` (Runs before each test method)
- `tearDown()` (Runs after each test method)
Profiling

- A way of finding performance bottlenecks in code
- It is always better to profile than to optimize prematurely
  - It’s hard to tell where the problem really is
- Modules: `profile`, `cProfile` (faster)
- Usually call from the command line:
  - `python -m cProfile perfect.py`
Timing Code

- **time** module
  - see `perfecttest2.py, timesample.py` and page 191

- **timeit** module
  - executes a statement 1 million times
  - see `timeit_sample.py`, and page 191–192
Speeding Up Your Code

1) Start with biggest bottleneck revealed from profiling
2) Use a better algorithm
3) Use built-in types when possible (i.e., don’t add needless layers; see p. 195)
4) Use __slots__ when possible
5) Comprehensions are faster than loops
6) Avoid . by using from <module> import ...
7) Use C Extensions
8) Use exceptions for the exceptional (instead of return codes; p. 196-197)
9) Use exceptions only for the exceptional (p. 197)
C Extensions

* You can import C code as a Python module

* A tedious process!
  * include `Python.h`
  * Use specific types for inter-language communication
  * Build and install a shared library for it

* Other options:
  * boost.python, Cython
Using Cython

- Translates a Python module to a C extension!
- Download from cython.org
  - run `python setup.py install`
- Write a python module to translate to C (`foo.py`, say)
- Run `cython foo.py` (creates `foo.c`)
- Then use the `distutils` standard C extension `setup.py`
  - places an importable module in the import search path
# isperfect.py: Test a number for "perfectness"

def isperfect(testnum):
    factors = [n for n in range(1,testnum/2+1) if testnum % n == 0]
    return sum(factors) == testnum

$ Cython isperfect.py$
$ ls isperfect.*$
isperfect.c isperfect.py
$
Install the Extension

$ cat setup.py
from distutils.core import setup, Extension
setup(name='isperfect',ext_modules=[Extension('isperfect',sources=['isperfect.c'])])

$ python setup.py install
running install
running build
running build_ext
building 'isperfect' extension
creating build/temp.macosx-10.6-intel-2.7
gcc-4.2 -fno-strict-aliasing -fno-common -dynamic -arch i386 -arch x86_64 -g -O2 -DNDEBUG -g -O3 -I/Library/Frameworks/Python.framework/Versions/2.7/include/python2.7 -c isperfect.c -o build/temp.macosx-10.6-intel-2.7/isperfect.o
creating build/lib.macosx-10.6-intel-2.7
 gcc-4.2 -arch i386 -arch x86_64 -isysroot / -g -bundle -undefined dynamic_lookup -arch i386 -arch x86_64 -isysroot / -g build/temp.macosx-10.6-intel-2.7/isperfect.o -o build/lib.macosx-10.6-intel-2.7/isperfect.so
running install_lib
copying build/lib.macosx-10.6-intel-2.7/isherfect.so -> /Library/Frameworks/Python.framework/Versions/2.7/lib/python2.7/site-packages
running install_egg_info
Writing /Library/Frameworks/Python.framework/Versions/2.7/lib/python2.7/site-packages/isherfect-0.0.0-py2.7.egg-info
$
Use the Extension

Charles-Allisons-MacBook-Pro-2:339R chuck$ python perfecttest.py
True
Charles-Allisons-MacBook-Pro-2:339R chuck$ python perfecttest2.py
True
1.413469
Charles-Allisons-MacBook-Pro-2:339R chuck$ python perfecttest2.py
True
1.826655
Charles-Allisons-MacBook-Pro-2:339R chuck$ python perfecttest.py
True
1.409206
Charles-Allisons-MacBook-Pro-2:339R chuck$ python perfecttest2.py
True
1.829295

30% improvement.