Software Development for Python in 30 minutes

Gabriel Pratt
gpratt@ucsd.edu
@gabepratt

Agenda

- Case Study
- Topics
  - Version Control
  - Best Practices
  - Testing
  - Standards
  - Profiling
  - Optimization
  - Debugging
  - Deployment

READ

Best Practices for Scientific Computing

Greg Wilson, D. A. Aruliah, C. Titus Brown, Neil P. Chue Hong, Matt Davis, Richard T. Guy, Steven H. D. Haddock, Katy Huff, Ian M. Mitchell, Mark Plumbley, Ben Waugh, Ethan P. White, Paul Wilson

WATCH

Pycon talks
http://pyvideo.org

CLIPper
(CLIP-seq peak calling algorithm)

- Inherited from biology grad student
- ~3000 lines of code
  - ~1000 lines in one function

Unix Development Practices

- Small programs that do one thing well
- When possible read from stdin output to stdout
- Use relative paths

Development Environment

- Prototype / Rapid iteration
  - IPython Notebook
- Heavyweight Development
  - Eclipse + pydev
    - Includes integrated version control, refactoring capabilities, automated testing and much more

Version Control

- Git
  - Distributed, better, easier to use than SVN
- Mercurial
- Bitbucket and github are good, free places to get offsite version control (and backups)

Testing

- Unit testing
  - Tests individual functions in a vacuum
  - Should be short, <1 sec
  - Nose
    - Python unit testing framework
    - Eclipse has continuous testing, every save you automatically test everything

```
class Test(unittest.TestCase):
    def test_check_for_index(self):
        some test...
```

python setup.py nosetests

- Integration Testing
  - Tests the entire program on a few medium sized datasets
  - Useful to make sure your program didn't just generally break
    - I cheat here and write them into my nosetests, better ideas?

Anatomy of a Test

```
def test_build_geneinfo(self):

    """

    Performs unit testing on build_geneinfo

    """

    #checks error mode
    self.assertRaises(TypeError, build_geneinfo, None)

    self.assertRaises(IOError, build_geneinfo, "foo")

    #checks working mode
    geneinfo = build_geneinfo(
                clipper.data_file("test.AS.STRUCTURE_genes.BED.gz"))


    true_values = {
    "ENSG00000232113" : ["chr1",  "ENSG00000232113",  173604911,      173606273, "+"],
    "ENSG00000228150" : ["chr1",  "ENSG00000228150",  10002980,       10010032,  "+"],
    "ENSG00000223883" : ["chr1",  "ENSG00000223883",  69521580,       69650686,  "+"],
    "ENSG00000135750" : ["chr1",  "ENSG00000135750",  233749749,      233808258, "+"],
    "ENSG00000227280" : ["chr1",  "ENSG00000227280",  145373053,      145375554 ,"-"],
    }


    self.assertDictEqual(geneinfo, true_values)
```

Standards

- Function / Method sizes
  - Keep them small, if you need more than 3 sentences to describe what a function is doing you are doing to much
  - This should be enforced by good unit testing practices
- Comments
  - Comment intent, not function.
    - Good: Sets a p-value cutoff based on...
    - Bad: Loops 10 times, prints result
  - doc strings
    - What the function does
    - Inputs and types of inputs
    - Outputs and types of outputs
  - Sphinx
- Pylint
  - Very strict standard checker. Quantifiable metrics for code quality

```
def build_transcript_data_gtf(gtf_file, pre_mrna):

    """

    Generates GTF file to use when calling genes
    Returns the longest gene from a group of transcripts to call peaks
    on

    gtf_file - bedtool from a standard gtf file
    pre_mrna - boolean flag to use pre_mrna instead of mrna

    """
```

Pylint

```
Raw metrics
-----------

+----------+-------+------+---------+-----------+
|type      |number |%     |previous |difference |
+==========+=======+======+=========+===========+
|code      |2526   |55.12 |NC       |NC         |
+----------+-------+------+---------+-----------+
|docstring |1144   |24.96 |NC       |NC         |
+----------+-------+------+---------+-----------+
|comment   |400    |8.73  |NC       |NC         |
+----------+-------+------+---------+-----------+
|empty     |513    |11.19 |NC       |NC         |
+----------+-------+------+---------+-----------+
```

```
+---------+-------+----------+----------+-----------+---------+
|type     |number |old number |difference |%documented |%badname |
+=========+=======+==========+==========+===========+=========+
|module   |16     |NC        |NC        |50.00       |25.00    |
+---------+-------+----------+----------+-----------+---------+
|class    |9      |NC        |NC        |44.44       |0.00     |
+---------+-------+----------+----------+-----------+---------+
|method   |93     |NC        |NC        |59.14       |25.81    |
+---------+-------+----------+----------+-----------+---------+
|function |65     |NC        |NC        |93.85       |10.77    |
+---------+-------+----------+----------+-----------+---------+
```

```
144 def get_acceptable_species():
145
146     """
147
148     Finds all species in data directory
149
150     """
151
152     acceptable_species = set([])
153     for fn in os.listdir(clipper.data_dir()):
154         fn = fn.split(".")[0]
155
156         if fn == "__init__":
157             continue
158
159         acceptable_species.add(fn)
160
161     return acceptable_species
162
```

Optimization

“Premature optimization is the root of all evil” – Donald Kanooth

The two rules of optimization:

1. Don't
2. Don't yet (experts only)

-Michael Jackson

Profiling

- Your slowdown is never where you think
- cProfile
  - Built in python profiler
  - Abuse it if you really need to get speedups

    (I've dropped running time for 400x in some cases)

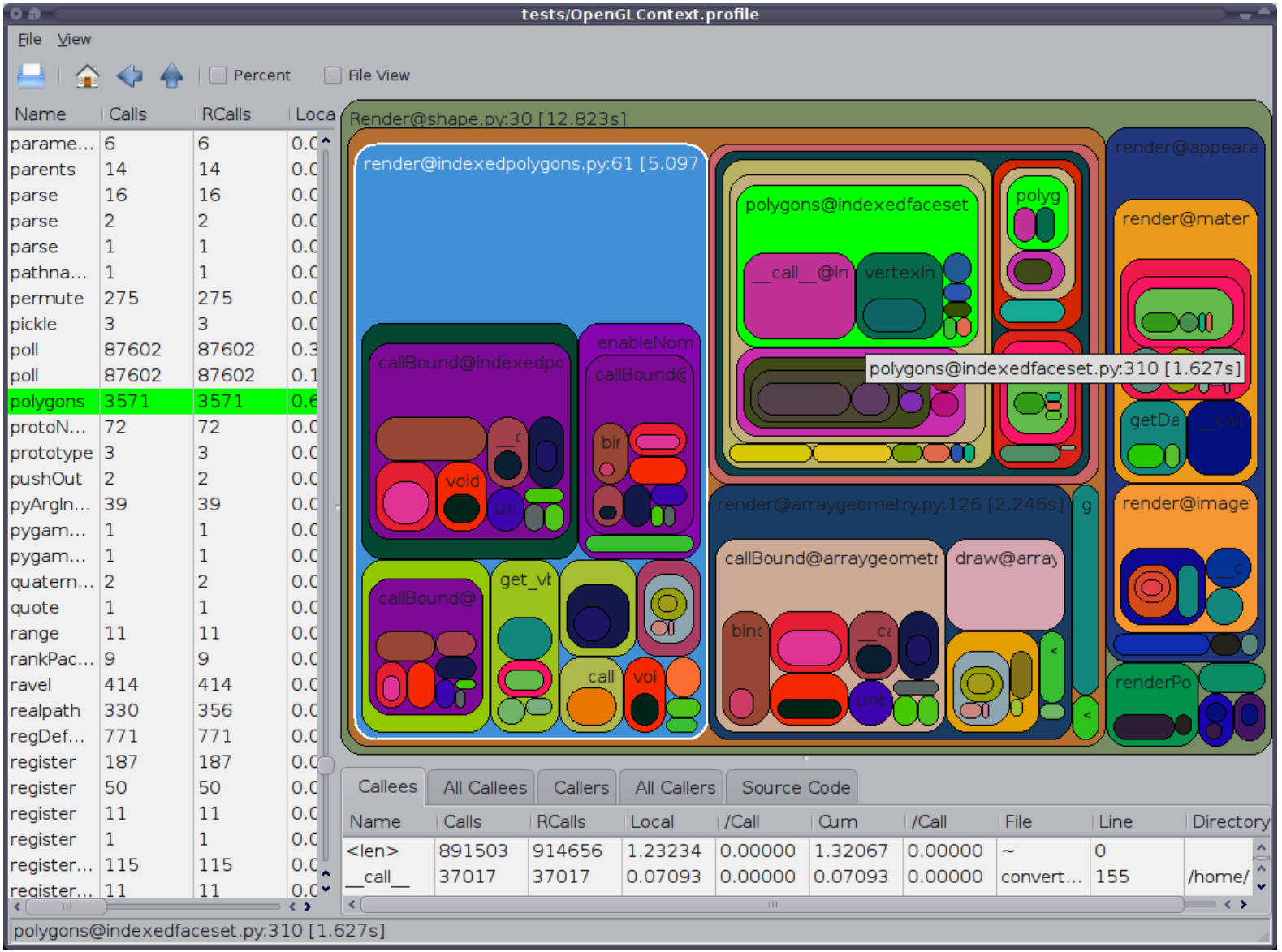
How to

python -m cProfile -o prof.txt program <args>

```
In [6]: import pstats

In [7]: p = pstats.Stats("prof.txt")

In [8]: p.sort_stats('cumulative').print_stats(10)
Wed Apr 10 10:21:33 2013    prof.txt

         2668708 function calls (2614879 primitive calls) in 32.886 CPU seconds

   Ordered by: cumulative time
   List reduced from 3657 to 10 due to restriction <10>

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
      2/1    0.005    0.003   32.713   32.713 build/lib.linux-i686-2.7/clipper/src/peakfinder.py:1(<module>)
        1    0.000    0.000   30.299   30.299 build/lib.linux-i686-2.7/clipper/src/peakfinder.py:571(call_main)
        1    0.056    0.056   30.294   30.294 build/lib.linux-i686-2.7/clipper/src/peakfinder.py:478(main)
        3    0.000    0.000   25.818    8.606 /usr/local/lib/python2.7/dist-packages/pybedtools-0.6.2-py2.7-linux-i686.egg/pybedtools/bedtool.py
ecorated)
        2    0.000    0.000   25.806   12.903 /usr/local/lib/python2.7/dist-packages/pybedtools-0.6.2-py2.7-linux-i686.egg/pybedtools/bedtool.py
saveas)
        2    0.000    0.000   25.796   12.898 /usr/local/lib/python2.7/dist-packages/pybedtools-0.6.2-py2.7-linux-i686.egg/pybedtools/bedtool.py
_collapse)
  51008/5   18.062    0.000   25.796    5.159 cbedtools.pyx:605(__next__)
        2    0.477    0.239   25.793   12.897 /usr/local/lib/python2.7/dist-packages/pybedtools-0.6.2-py2.7-linux-i686.egg/pybedtools/bedtool.py
genexpr>)
        1    0.456    0.456    4.377    4.377 build/lib.linux-i686-2.7/clipper/src/peakfinder.py:249(build_transcript_data)
   102005    1.859    0.000    3.252    0.000 cbedtools.pyx:495(create_interval_from_list)
```

Visualization

RunSnakeRun - http://www.vrplumber.com/programming/runsnakerun/



Speedups

- Hard: C extension functions
  - Write a C function that can interface with python
- Easy: Cython
  - Compile your python into C
- Medium: Ctypes
  - Statically define types in python, compile down to C

Watch for memory leaks!

Debugging

- PDB
  - Ipython integrated debugger
    - %pdb
    - %run myfunction.py
    - Will catch any crash and automagically drop you into pdb, no more print statements all over your code (well less)
- GDB
  - Used for C debugging
  - Use for debugging python extension functions

Deployment

- Setup.py
  - Easy to install your python program locally
- Pypi
  - Repository python packages (how easy_install works)
  - You can put your package up as well!

Setup.py

```
setup(
    name = "clipper",
    long_description = long_description,
    version = "0.1.1",
    packages = find_packages(),
    ext_modules = [peaks],
    package_data = {
        '' : ['*.lengths', '*.gz', '*.bam', '*.bai']
        },

    install_requires = ['setuptools',
                        'pysam >= 0.6',
                        'numpy >= 1.5.1 ',
                        'scipy >= 0.11.0',
                        'matplotlib >= 1.1.0',
                        'pybedtools >= 0.5',
                        'scikit-learn >= 0.13.0',
                        ],

    setup_requires = ["setuptools_git >= 0.3",],

    entry_points = {
                    'console_scripts': [
                                        'clipper = clipper.src.peakfinder:call_main',],
                    },
)
```

```
python setup.py upload
```