Programming in Python, without messing it up

Toon Verstraelen

Center for Molecular Modeling (CMM), Ghent University, Belgium

Universidad de Concepción, Chile
January, 2019

UNIVERSITEIT
GENT

FACULTY
OF SCIENCES

CENTER FOR
MOLECULAR MODELING

You know Python. Great!

Now the real fun begins...

- ¿Reliable software?
- ¿Managing complexity?
- ¿Sustainable software development?

~~¿How to debug?~~
¿How to avoid bugs?

Oh no! I wrote this code when I was 25.
This is going to be a long night!
imgflip.com

Coding style

- = rules for readability
- In general, adhere to PEP8
  https://www.python.org/dev/peps/pep-0008/
- Tools:
  - `pip install --user` **`pycodestyle`**

    Checks a subset of PEP8 rules
  - `pip install --user` **`pydocstyle`**

    Checks a subset of PEP257 (docstrings)
  - `pip install --user` **`pylint`**

    Checks subset of PEP8 and other things.
  - Cardboardlint => our wrapper for many checkers

Write transparent code

- **Single line of code = self-explaining**
  - Give variables, functions, … **sensible names**.
  - **Not too much** stuff in one line. No crazy one-liners.
- **Comments explain code** (implementation)
  - **English**, please.
  - Comment on **groups of lines**, rarely individual lines.
- **Docstrings explain usage of code** (API)
  - **Document** a function, class, module, …
  - **Describe** parameters, return values, exceptions & behavior

```
def fire_in_the_disco(msg):
    """Contributed by https://pythondev.slack.com/team/staticmethod
    This code was written for obfuscation contest.
    """
    reconstitute(msg,wwpd)
    try:
        f=type((lambda:(lambda:None for n in range(len((((((),(((),())))))))))))
().next())
        u=(lambda:type((lambda:(lambda:None for n in
range(len(zip(((((((((())))))))))))).func_code))()
        n=f(u(int(wwpd[4][1]),int(wwpd[7][1]),int(wwpd[6][1]),int(wwpd[9]
[1]),wwpd[2][1],
            (None,wwpd[10][1],wwpd[13][1],wwpd[11][1],wwpd[15][1]),(wwpd[20]
[1],wwpd[21][1]),
            (wwpd[16][1],wwpd[17][1],wwpd[18][1],wwpd[11][1],wwpd[19]
[1]),wwpd[22][1],wwpd[25][1],int(wwpd[4][1]),wwpd[0][1]),
            {wwpd[27][1]:__builtins__,wwpd[28][1]:wwpd[29][1]})
        c=partial(n, [x for x in map(lambda i:n(i),range(int(0xbeef)))])
        FIGHT = f(u(int(wwpd[4][1]),int(wwpd[4][1]),int(wwpd[5]
[1]),int(wwpd[9][1]),wwpd[3][1],
                (None, wwpd[23][1]), (wwpd[14][1],wwpd[24][1]),(wwpd[12]
[1],),wwpd[22][1],wwpd[26][1],int(wwpd[8][1]),wwpd[1][1]),
                {wwpd[14][1]:c,wwpd[24][1]:urlopen,wwpd[27]
[1]:__builtins__,wwpd[28][1]:wwpd[29][1]})
        FIGHT(msg)
    except:
        pass
```

```python
def compute_surface_polygon(x, y):
    """Compute the surface area of a 2D polygon.

    Parameters
    ----------
    x : np.array
        X-coordinates of the polygon's corners.
    y : np.array
        Y-coordinates of the polygon's corners.

    Returns
    -------
    area : type of x and y
        The surface area of the polygon.

    """
    # Shoelace algorithm, Meister, 1769
    if len(x) != len(y):
        raise TypeError("Arguments x and y must have the same length.")
    if len(x) <= 2:
        return 0.0
    else:
        return abs( x[-1]*y[0] + np.dot(x[:-1], y[1:])
                   -x[0]*y[-1] - np.dot(x[1:], y[:-1]))/2
```

$$A = \frac{1}{2}\Big| x_N y_1 - x_1 y_N + \sum_{i=1}^{N-1} x_i y_{i+1} - x_{i+1} y_i \Big|$$

Unit tests

- = function to validate another function
- Runs fast, easy to start
- Write many!
- Think of corner cases
- Coverage analysis = check if code is tested

```python
def check_single(x, y, area):
    np.testing.assert_almost_equal(compute_area_polygon(x, y), area)

def check_variants(x, y, area):
    x = np.asarray(x)
    y = np.asarray(y)
    check_single(x, y, area)
    check_single(x[::-1], y[::-1], area)
    check_single(x + 0.3, y - 0.5478, area)
    check_single(-2*x, 0.8*y, 1.6*area)
    xp = np.cos(0.3)*x - np.sin(0.3)*y
    yp = np.sin(0.3)*x + np.cos(0.3)*y
    check_single(xp, yp, area)

def test_compute_area_polygon():
    # Simple geometries
    check_single([0, 0, 1, 1], [0, 1, 1, 0], 1.0)
    check_single([0.0, 0.0, 2.0], [0.0, 1.0, 1.0], 1.0)
    check_single([-0.5, 2.5, 1.0, 0.0], [0.0, 0.0, 0.5, 0.5], 1.0)

    # Corner cases: flat, coinciding points, too short vectors
    check_single([0.0, 2.0, -1.0], [0.0, 2.0, -1.0], 0.0)
    check_single([0.0, 0.0, 2.0, 2.0], [0.0, 1.0, 1.0, 1.0], 1.0)
    check_single([], [], 0.0)
    check_single([1], [2], 0.0)
    check_single([2.0, 1.0], [0.0, 0.0], 0.0)
```

Live demo

```
# Plain, without coverage
nosetests -v meister.py

# With coverage analysis
nosetests -v meister.py \
    --with-coverage \
    --cover-html \
    --cover-package=meister
```

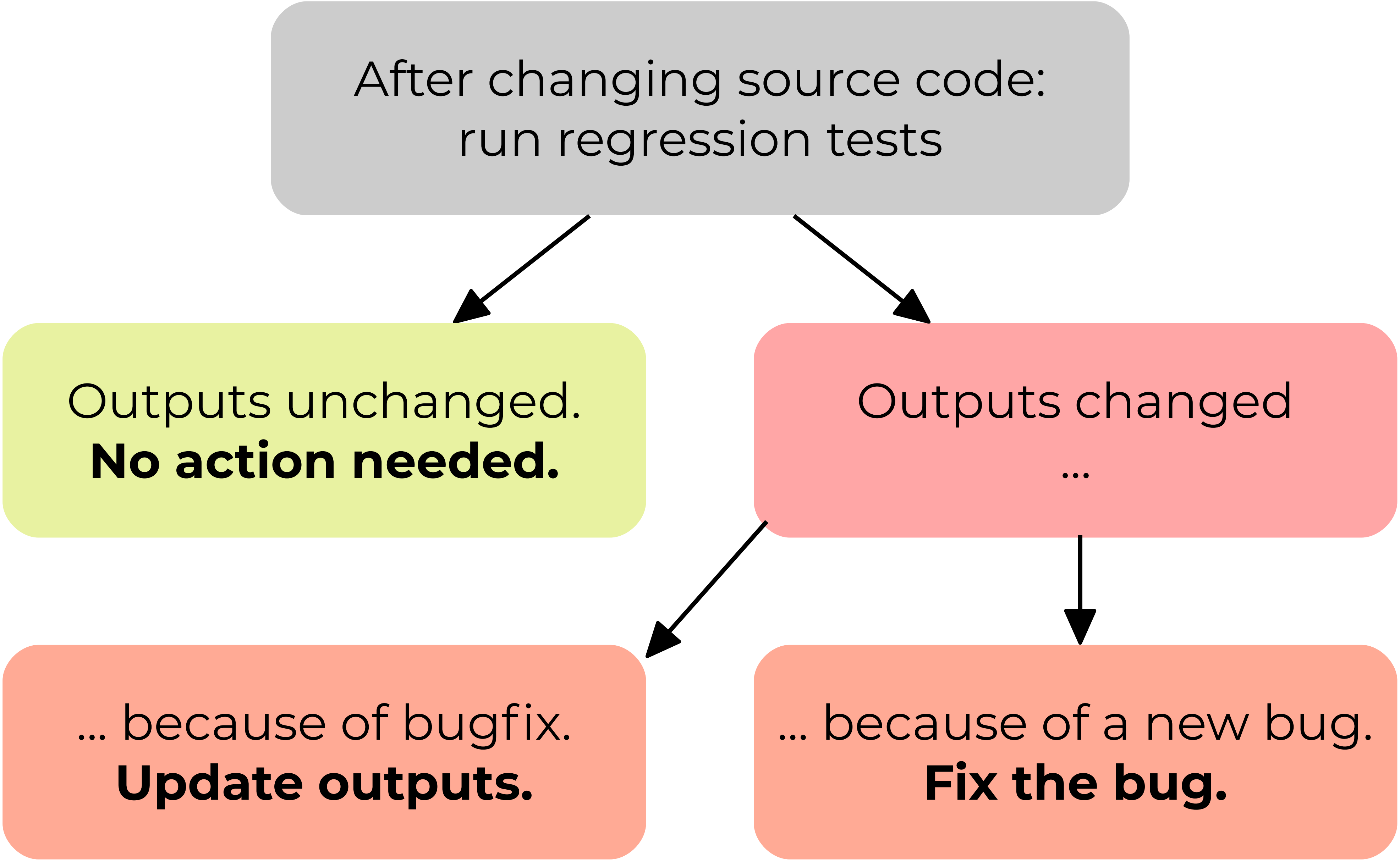
Regression tests

- = tests for entire program
- Slower than unit tests
- Test whether program changes behavior.

Pairs of input and output
for every feature of your program

test1.in	test1.out
test2.in	test2.out
test3.in	test3.out
test4.in	test4.out
...	...

Regression tests



After changing source code:
run regression tests

Outputs unchanged.
No action needed.

Outputs changed
...

... because of bugfix.
Update outputs.

... because of a new bug.
Fix the bug.

Hands-on: the Kabsch algorithm

1. Write the function signature & docstring.
2. Write one unit test.
3. Implement the Kabsch algorithm.
4. Write more unit tests.
5. Perform coverage analysis.
6. Corner cases?
7. Review your neighbour’s code.

~~¿How to write complex software?~~
¿How to hide complexity?

Use (built-in) packages

Ideal for reducing code:

- **argparse**
  command-line argument parser
- **collections** (namedtuple)
  beyond lists and dicts
- **glob** & **fnmatch**
  UNIX-style pattern matching: `"foo*_???.txt"`
- **JSON**
  simple data representation, very widely used.
- **YAML**
  JSON generalization,
  better suited for humans

Live demo

```python
from collections import namedtuple
Point = namedtuple('Point', ['x', 'y'])
p = Point(11, y=22)
p[0] + p[1]
x, y = p
x, y
p.x + p.y
p
p._replace(x=100)

Point = namedtuple('Point', ['x', 'y'], verbose=True)
```

Use (built-in) packages

Scientific computing, besides the usual (NumPy, SciPy, MatplotLib, pandas):

- **H5Py:** binary cross-platform array file format
- **Cython:** Python C++ interface
- **Scikit-learn:** (old-school) machine learning & statistics
- **RDkit:** cheminformatics
- **Sympy:** symbolic calculus
- **Dask:** parallel workflows
- **AutoGrad:** algorithmic differentiation
- **Numba:** just-in-time compiler for Python
- **Mars:** parallel Numpy

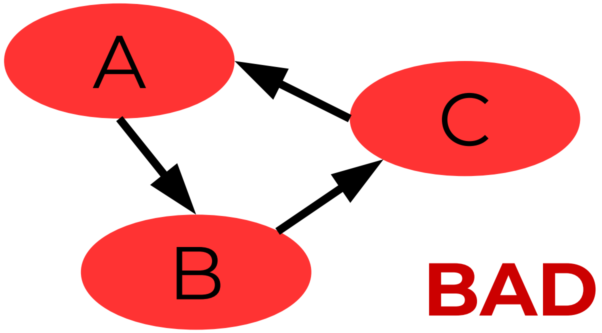
Split code into modules

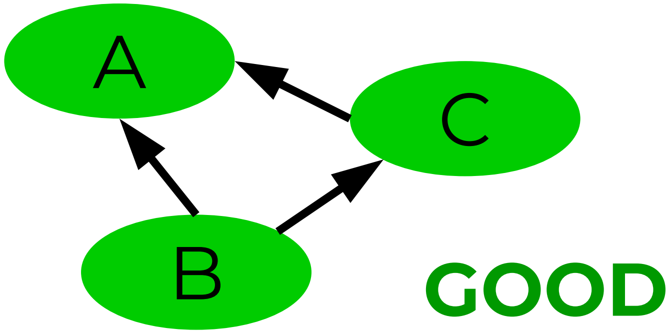
```python
# foo.py
def add(a, b):
    return a+b
```

```python
# bar.py
import foo
print(foo.add(1, 2))

from foo import add
print(add(1, 2))

from foo import *
```

Make *modular* modules

1. **No cyclic dependencies** between modules.
   You use a module ⇒ module does not use you.





2. Modules should have a **minimal API**.
3. Modules should have a well-defined **purpose**, which can be **summarized in 1 sentence**.

Idiomatic Python

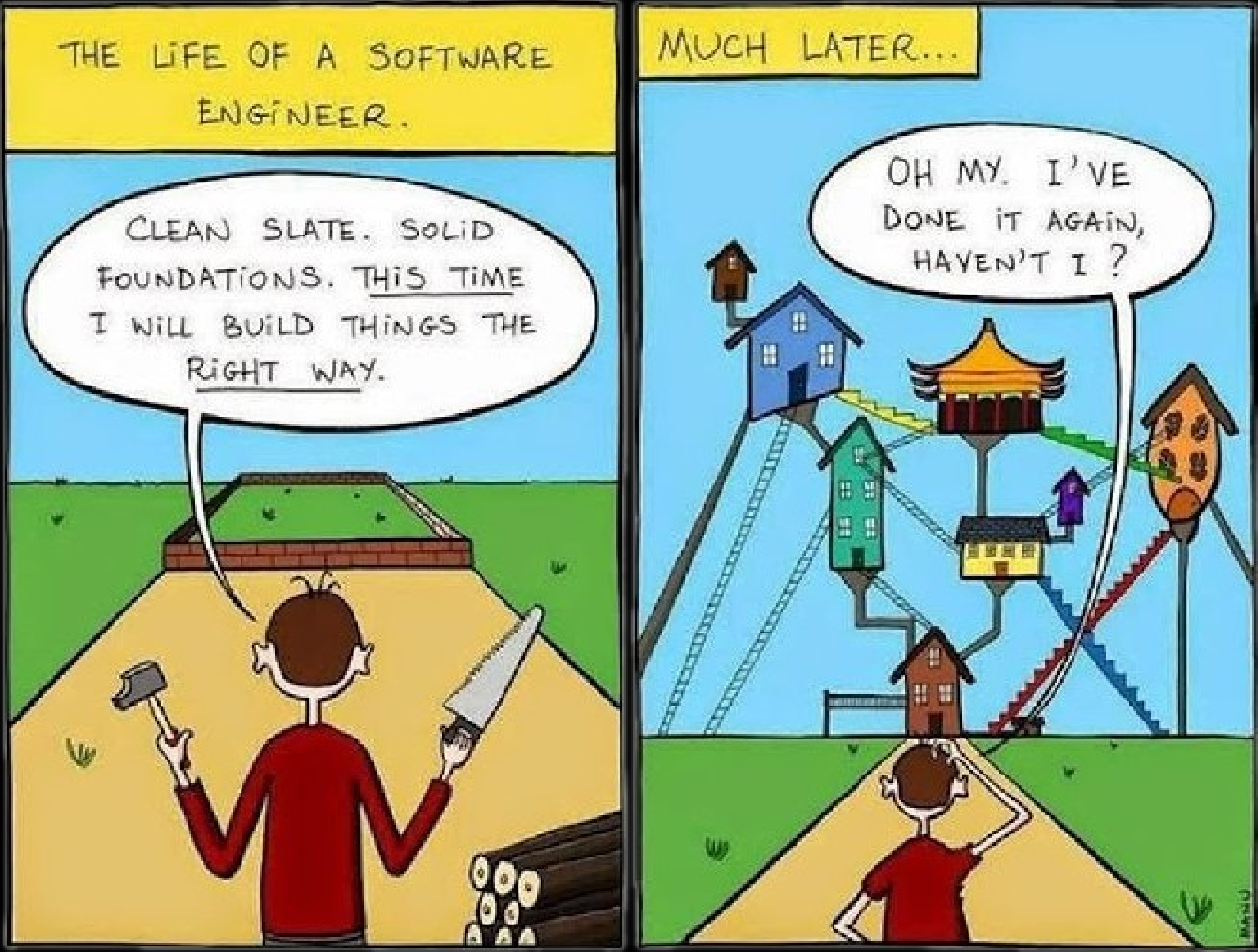
```
# Pythonic code, use context manager ("with") and enumerate:
with open("somefile.txt") as fh:
    for counter, line in enumerate(fh):
        print(counter, " ", line[: -1])

# C++ish code:
fh = None
try:
    fh = open("somefile.txt")
    counter = 0
    line = fh.readline()
    while len(line) > 0:
        print(counter, " ", line[:-1])
        counter += 1
        line = fh.readline()
finally:
    if fh is not None:
        fh.close()
```

See also:
https://docs.python.org/3.0/howto/doanddont.html

Object-oriented programming (OOP)

Before going into detail:

- OOP is sometimes over-rated. (Java)
- OOP does not solve all your problems.
- Keep it simple.
- Python does not support all OOP concepts. Hooray!


THE LIFE OF A SOFTWARE ENGINEER.
CLEAN SLATE. SOLID FOUNDATIONS. THIS TIME I WILL BUILD THINGS THE RIGHT WAY.
MUCH LATER...
OH MY. I'VE DONE IT AGAIN, HAVEN'T I ?
MANU

- Next to built-in types (int, list, str, ...), you can define more general "**objects**" with **attributes** and a **behavior**.

Live demo

- Classes can "**inherit**" from other classes, and add & override attributes & methods.

Live demo

"Polymorphism" justifies inheritance.

= Difference in behavior with the same API

Object-oriented programming (OOP)

Benefits

- Related elements (data and code) are also nearby in source.
- Higher-level programming, in terms of objects
- Polymorphism can reduce many "if" statements.

Limitations

- Methods are essentially unary operators.

Pitfalls

- Too many classes.
- Too complex inheritance diagrams. Use composition where possible.
- Too many methods.

Free functions

- = method “degraded” to a function.
  See https://www.youtube.com/watch?v=nWJHhtmWYcY
- Goal: keep classes simple & easy to understand
- When to write a free function?
  - Attributes are not modified (directly).
  - Algorithms that "work with" objects
  - Binary (or higher) operators.
  - When a class becomes too complicated.

Hands-on: polygon & regular polygon

1. Write polygon class and add features:

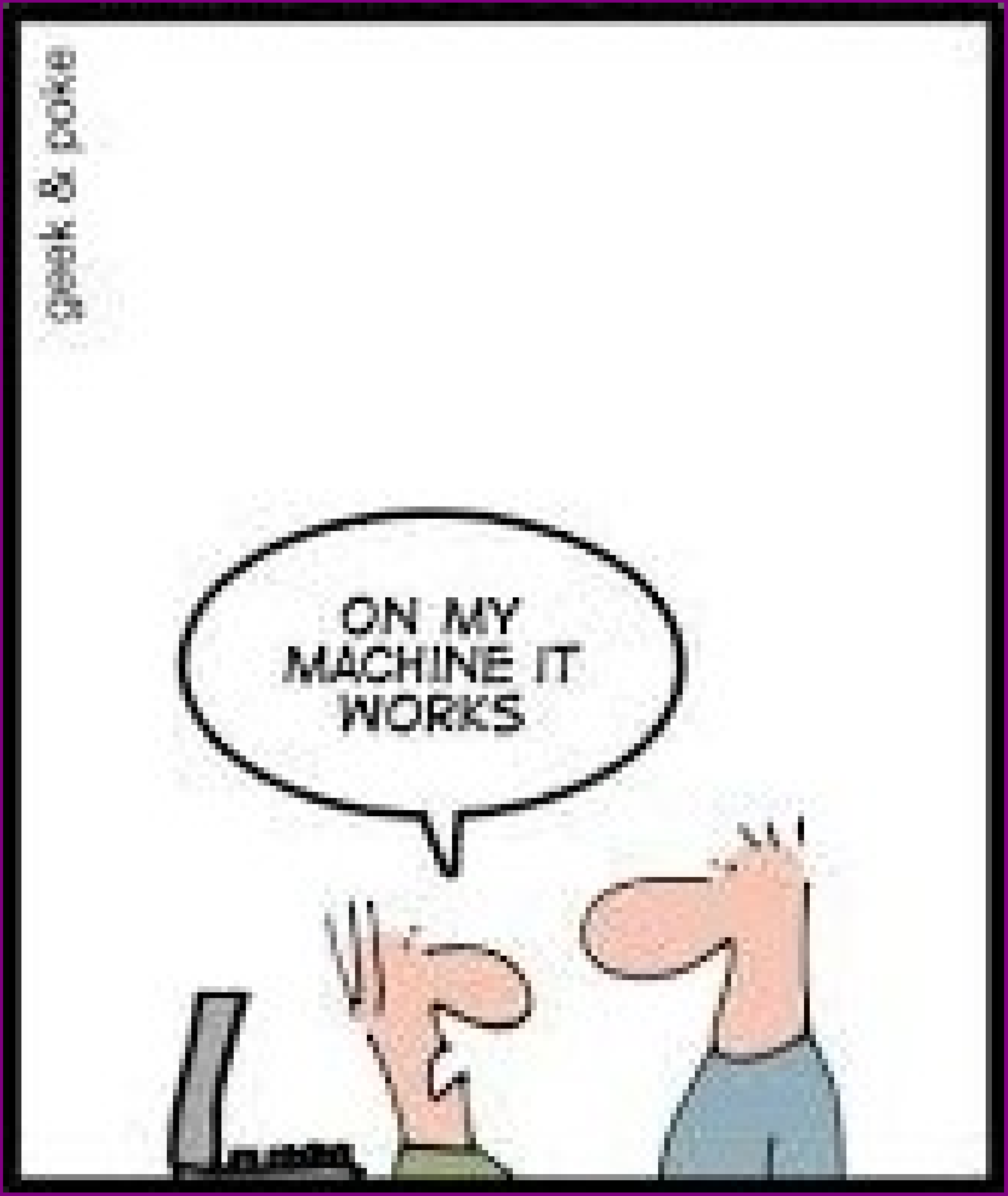
- compute_area and compute_perimeter
- rotate, scale and translate
- regular polygon

Select the “best” patterns: inheritance or composition, method or function.

2. Minimization of perimeter/area ratio

- Implement function for ratio with 1 argument: x & y arrays concatenated.
  Add regularization term, 1e-6*(1−area)**2.
- Implement gradient with autograd.
- Minimize with scipy.optimize.fmin_lbfgs_b.

geek & poke
YESTERDAY IT WORKED


geek & poke
ON MY MACHINE IT WORKS

Even if you don't collaborate...

Long-term maintenance

≈

Collaboration with your future self

More than *avoiding bugs* & *hiding complexity*.

Semantic versioning

https://semver.org/

Given a version number MAJOR.MINOR.PATCH, increment the:

- **MAJOR** version when you make incompatible API changes,
- **MINOR** version when you add functionality in a backwards-compatible manner, and
- **PATCH** version when you make backwards-compatible bug fixes.

Version Control System (VCS)

= records history of all changes in source code

Why?

Collaboration

- Merging: combine changes from different persons.
- Review code before merging.
- When was a bug introduced (bisection)
- Blame people for their ugly code. :)

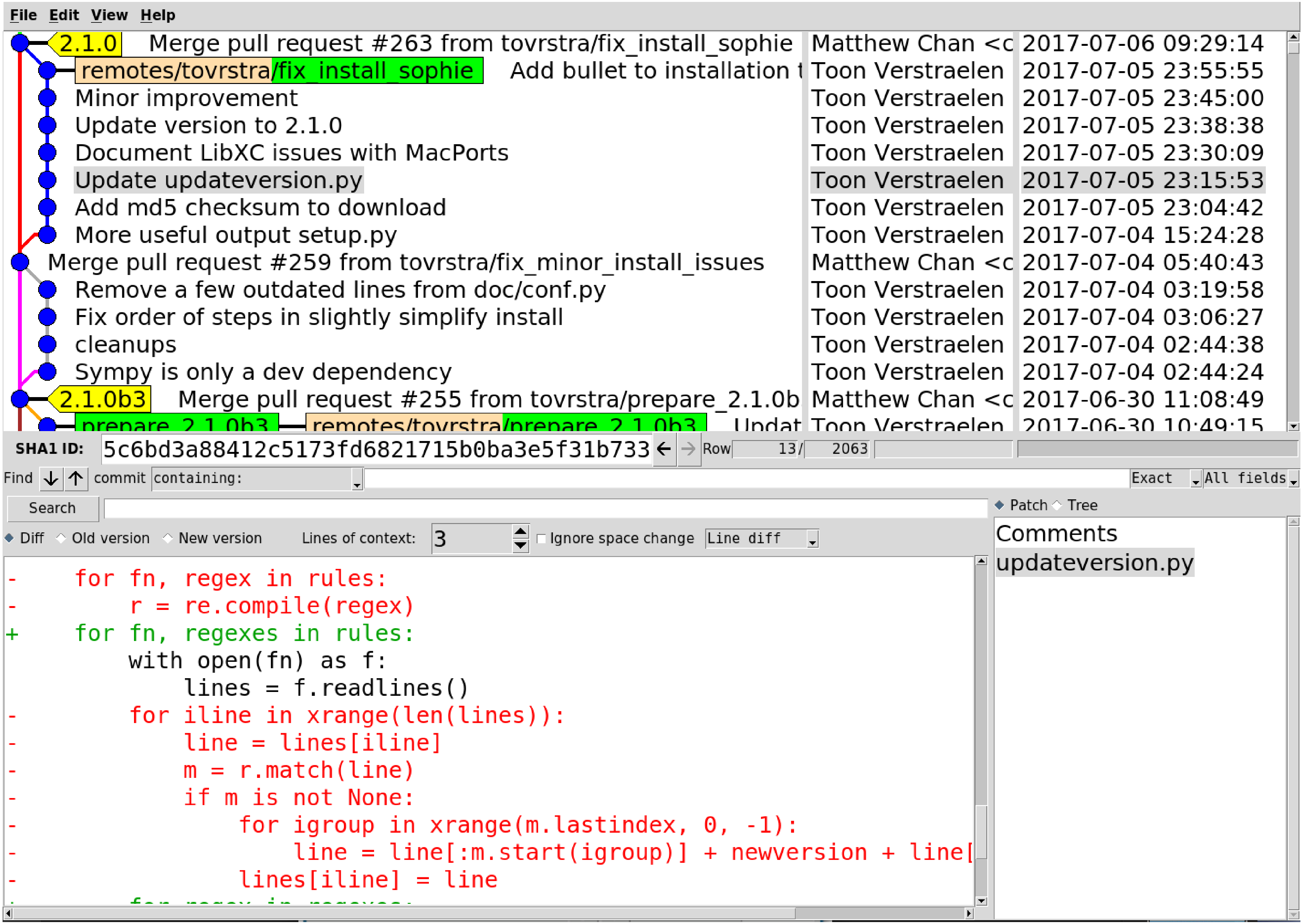
Access to all versions

Backup

A patch (file)

```
diff --git a/horton/grid/cext.pyx b/horton/grid/cext.pyx
index e4615275..47c607fc 100644
--- a/horton/grid/cext.pyx
+++ b/horton/grid/cext.pyx
@@ -55,7 +55,7 @@
     'PowerExtrapolation', 'PotentialExtrapolation', 'tridiagsym_solve',
     'CubicSpline', 'compute_cubic_spline_int_weights',
     # evaluate
-    'index_wrap', 'eval_spline_grid', 'eval_decomposition_grid',
+    'eval_spline_grid', 'eval_decomposition_grid',
     # ode2
     'hermite_overlap2', 'hermite_overlap3', 'hermite_node',
     'hermite_product2', 'build_ode2',
@@ -477,10 +477,6 @@



-def index_wrap(long i, long high):
-    return evaluate.index_wrap(i, high)
-
-
 def eval_spline_grid(CubicSpline spline not None,
                      np.ndarray[double, ndim=1] center not None,
                      np.ndarray[double, ndim=1] output not None,
```

Patch, Commit, Branch, Review, Merge, Release



2.1.0 Merge pull request #263 from tovrstra/fix_install_sophie	Matthew Chan <c	2017-07-06 09:29:14
remotes/tovrstra/fix_install_sophie Add bullet to installation t	Toon Verstraelen	2017-07-05 23:55:55
Minor improvement	Toon Verstraelen	2017-07-05 23:45:00
Update version to 2.1.0	Toon Verstraelen	2017-07-05 23:38:38
Document LibXC issues with MacPorts	Toon Verstraelen	2017-07-05 23:30:09
Update updateversion.py	Toon Verstraelen	2017-07-05 23:15:53
Add md5 checksum to download	Toon Verstraelen	2017-07-05 23:04:42
More useful output setup.py	Toon Verstraelen	2017-07-04 15:24:28
Merge pull request #259 from tovrstra/fix_minor_install_issues	Matthew Chan <c	2017-07-04 05:40:43
Remove a few outdated lines from doc/conf.py	Toon Verstraelen	2017-07-04 03:19:58
Fix order of steps in slightly simplify install	Toon Verstraelen	2017-07-04 03:06:27
cleanups	Toon Verstraelen	2017-07-04 02:44:38
Sympy is only a dev dependency	Toon Verstraelen	2017-07-04 02:44:24
2.1.0b3 Merge pull request #255 from tovrstra/prepare_2.1.0b	Matthew Chan <c	2017-06-30 11:08:49
prepare_2.1.0b3 remotes/tovrstra/prepare_2.1.0b3 Updat	Toon Verstraelen	2017-06-30 10:49:15

SHA1 ID: 5c6bd3a88412c5173fd6821715b0ba3e5f31b733 Row 13/ 2063

Find commit containing: Exact All fields

◆ Diff ◇ Old version ◇ New version Lines of context: 3 □ Ignore space change Line diff

```
-    for fn, regex in rules:
-        r = re.compile(regex)
+    for fn, regexes in rules:
         with open(fn) as f:
             lines = f.readlines()
-        for iline in xrange(len(lines)):
-            line = lines[iline]
-            m = r.match(line)
-            if m is not None:
-                for igroup in xrange(m.lastindex, 0, -1):
-                    line = line[:m.start(igroup)] + newversion + line[
-                lines[iline] = line
         for regex in regexes:
```

Comments
updateversion.py

Git & Github

Git = probably the best VCS software

https://git-scm.com/

- Steep learning curve, but worth it.
- Lots of online tutorials.

Github = Git hosting

https://github.com/

- Hosts git repositories
- Extra's: issue tracker, pull requests, web hosting

Continuous integration (CI)

= automatically analyze every commit on Github:

- Unit tests + coverage analysis
- Coding style (pylint, pycodestyle, ...)
- Test package build & install
- ...

Very neat, involved setup, **use cookiecutter**.

https://travis-ci.org/

Example: https://github.com/theochem/grid/pull/4

Write user documentation

README.md

- Links to other documentation
- Quick install instructions
- Contact & License information

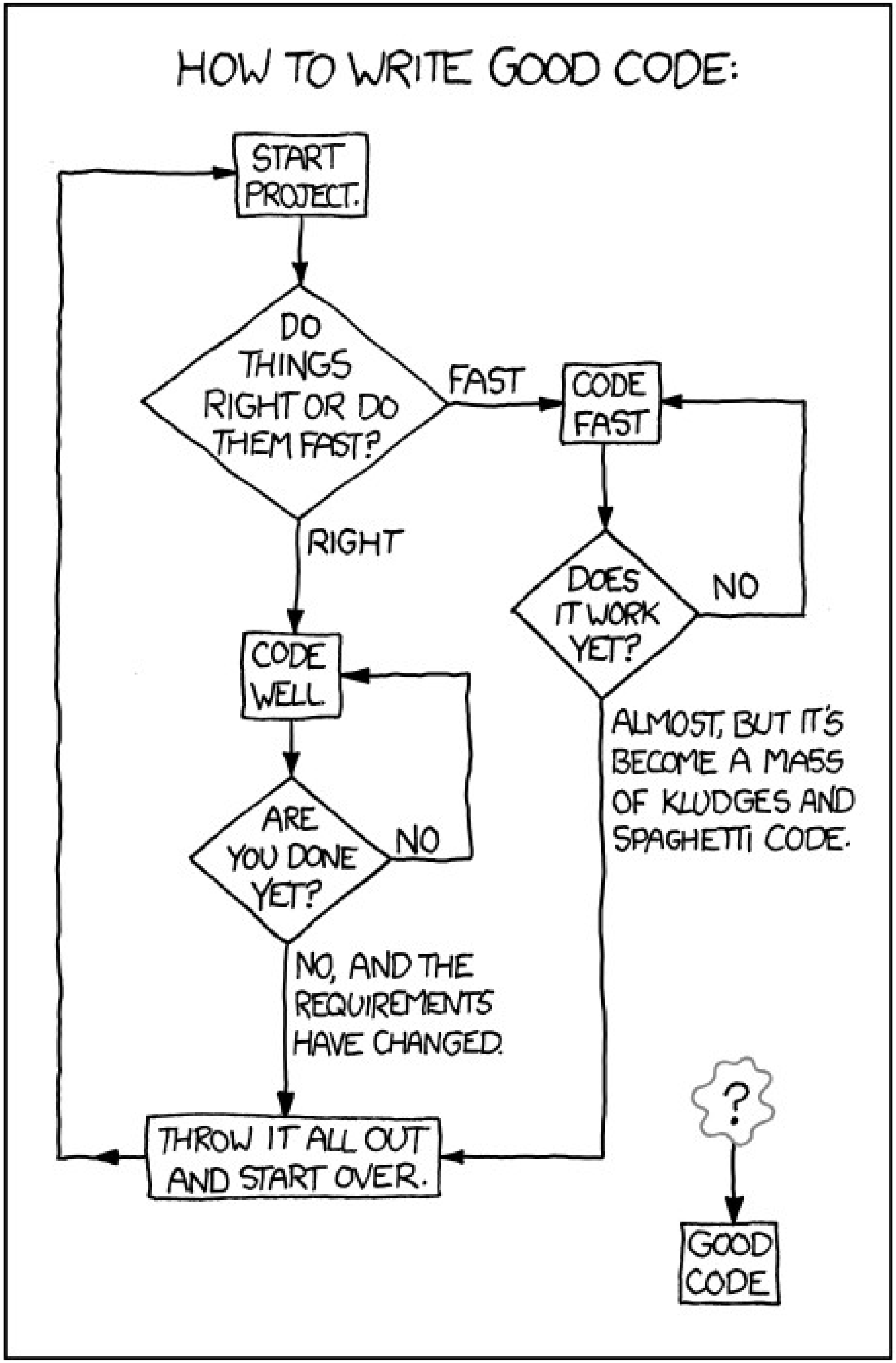
Website (use Sphinx; http://www.sphinx-doc.org)

- Background
- Tutorials
- API reference

Assignment: fix a simple bug

Fix Scipy documentation:

https://github.com/scipy/scipy/issues/7168


HOW TO WRITE GOOD CODE:
START PROJECT.
DO THINGS RIGHT OR DO THEM FAST?
FAST
CODE FAST
DOES IT WORK YET?
NO
ALMOST, BUT IT'S BECOME A MASS OF KLUDGES AND SPAGHETTI CODE.
RIGHT
CODE WELL.
ARE YOU DONE YET?
NO
NO, AND THE REQUIREMENTS HAVE CHANGED.
THROW IT ALL OUT AND START OVER.
?
GOOD CODE

SCRUM

Keep it simple.