ISMRM 27TH ANNUAL MEETING & EXHIBITION
Palais des congrès de Montréal · Montréal, QC, Canada · 11–16 May 2019

Secret Session: Master coding in your research environment

Code Structure and Syntax

Renat Sibgatulin[1] and Saskia Bollmann[2]

[1]Department of Radiology, University Hospital Jena, Jena, Germany
[2]Athinoula A. Martinos Center for Biomedical Imaging, Massachusetts General Hospital, Boston, MA, USA
[2]Department of Radiology, Harvard Medical School, Boston, MA, USA
[2]Centre for Advanced Imaging, The University of Queensland, Brisbane, Australia

THE UNIVERSITY OF QUEENSLAND
AUSTRALIA

What constitutes good scientific code?

- Best practices for scientific computing (Wilson et al., 2014, PLOS Biology)
- Good enough practices in scientific computing (Wilson et al., 2017, PLOS CB)

correct

reliable

maintainable

readable

???

https://www.software.ac.uk/blog/2016-10-07-what-makes-good-code-good-digital-social-research-view

Readability

- You and future-you, your PI, the reviewer of your paper and the grad student taking over your project can understand what's happening
- Note: follows the same rules as scientific writing (clear, simple and short)
- sections: definitions, data read-in, preprocessing, …
- meaningful function and variable names: `fieldStrength_T`, `compute_gradient_moment`
- linear and explicit

Habits for good scientific code

+ choose a scientific programming language

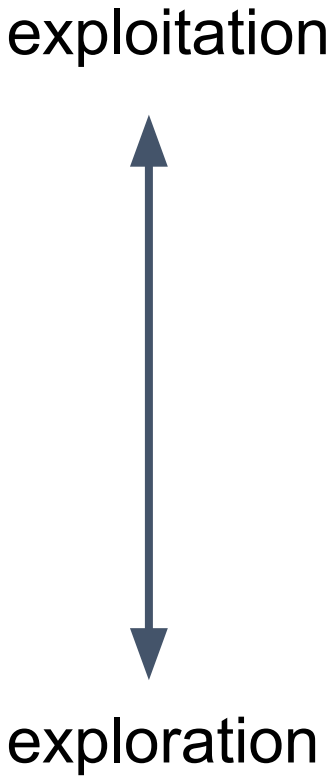
- Refactor (rewrite) your code
  - remove the copy and paste bits
  - split your code in small functional units (functions)
  - less code, increased readability, increased functionality
- Use toolboxes & the latest core functions
  - the 20 minutes invested in the tutorial will give you hours in functionality
  - write good (and tested) open source code :P
- Contribute
  - adapt to the coding principles/style
  - good learning experience

VectorStock®
VectorStock.com/13438794

+ keep it simple

Why is it so hard? OR The purpose of scientific code

- analyse my data / do stuff
- find the solution for a problem (or, more often, find the problem)
  - spaghetti code
  - unnecessary parameters/functions
  - cryptic comments
  - loops/if/else
- educate
  - great tool to understand a method
- it's science, dude!
  - reproducibility challenge, re-inventing the wheel, non-linearity



Procedural vs. OOP

- Scenario: need to solve an optimization problem (a nasty one)
- Found an algorithm. Goes like:

$$\arg\min_{\mathbf{x},\mathbf{y}} f(\mathbf{x}) + g(\mathbf{y}) \quad \text{s. t.} \quad A\mathbf{x} + B\mathbf{y} = \mathbf{c}$$

$$\mathbf{x}^{j+i} = \arg\min_{\mathbf{x}} f(\mathbf{x}) + \frac{\rho}{2}\|A\mathbf{x} - (\mathbf{c} - B\mathbf{y}^j - \mathbf{u}^j)\|_2^2$$

$$\mathbf{y}^{j+i} = \arg\min_{\mathbf{y}} g(\mathbf{y}) + \frac{\rho}{2}\|B\mathbf{y} - (\mathbf{c} - A\mathbf{x}^j - \mathbf{u}^j)\|_2^2$$

$$\mathbf{u}^{j+1} = \mathbf{u}^j + A\mathbf{x}^{j+1} + B\mathbf{y}^{j+1} - \mathbf{c}$$

Procedural vs. OOP

- General algorithm
  + pen & paper
  + your problem
  = tailored solution

\+ Concise

\+ Doesn't get faster

– Hard to generalize

```
for iter_num in range(iter_num_max):
    # update x, the minimizer
    GHdyu = np.conj(Gf) * fftn(dy - du, axes=(0,1))
    DHyu  = np.conj(Df) * fftn( y -  u, axes=(0,1))

    x_prev = x
    x = ifftn( (DHys + rho*GHdyu.sum(axis=-1)) / \
              (np.finfo(float).eps + DHDf + rho*GHGf) ).real

    x_update = 100 * np.linalg.norm(x-x_prev) / np.linalg.norm(x)
    print(f'Iter: {iter_num}. Update: {x_update:.3g}%')

    if x_update < tol_update:
        break

    # update y, the gradient varible
    xf = fftn(x)
    dx = ifftn(Gf * xf[...,None], axes=(0,1)).real

    dy = np.maximum(abs(dx + du) - regweight * (alpha/rho), 0) \
       * np.sign(dx + du)
    y = Wy + (ifftn(Df*xf)+u).real / (weight + rho)

    # update u, the dual variable
    du += dx - dy
    u += ifftn(Df*xf).real - y
```

Procedural vs. OOP

- General algorithm
  = base class
- Inheritance
  + pen & paper
  + your problem
  = tailored solution

\+ Easy to generalize
\+ Easier to extend

– ~~This will never work~~ 👉
Takes extra thought

```
class OptimizerBase:
```

```
class TVDenoise(OptimizerBase):
    def __init__(self, data):
        xshape = data.shape
        # specifics of the problem
        super().__init__(xshape, yshape, ushape)

    def solve(self):
        # left alone

    def xstep(self):
        # specific implementation...
```

```
def ustep(self):
    return self.const_A(self.x) + self.const_B(self.y) - self.const_c()
def const_A(self, x):
    return X
def const_B(self, y):
    return -y
def const_c(self):
    return 0.0
```

Procedural vs. OOP

- Should you? Depends

- Code to do one job / publish one paper -> procedural
- See it extendable, possibly by others -> OOP