Best Practices for Software Development

Test-Driven Development

Use assertions throughout your program to check that values are as you expect them to be at different steps.

Example:

```
Assert x > 0.0, 'Data should only be positive'
```

Test the output of each function often

Know What it's Supposed to Do

Two levels of "knowing":

1. Know the syntax of the programming language
   What is this line of code supposed to do?
2. Know the science of what your program is doing
   Do the results make sense?

Know What it's Supposed to Do

1. Test with simplified data or simplified case
    a. Test it on something you can solve or calculate using pen and paper to make sure it works correctly
    b. Unit testing - Test the output of each function
2. Compare to an oracle (or benchmark)
    a. Compare to trusted results of an experiment or another code that does the same thing
3. Check the science at different stages
    a. What values are supposed to be conserved? If they are conserved in real life, they should be conserved in the code.
4. Visualize it often
    a. Do the results look correct base on my knowledge of the subject?
    b. My personal use: Use matplotlib to automatically make plots while running a simulation.

Make it Fail

- Make it fail **every time**
- Make it fail **fast**
- Know how to make it fail, so you know how to avoid it.

We can only debug something if it fails.

Make One Change at a Time

- If you change several things at one time, it can be hard to trace back an error. *What line of the 10 I just changed caused this?*
- Rerun tests after each function

Keep Track of Your Changes

- Documentation
  - Docstrings for functions
  - READMEs
- Version Control (git and github)
  - Descriptive commit messages
  - Commit often
- Comment, comment, comment!
  - Someone should be able to understand your code through the comments without needing extensive knowledge of the language syntax.
  - Make notes in the code about what chunks of code are supposed to do

Your biggest collaborator is you 6 months ago and they don't answer emails!

Human Readability

- Use descriptive variable, function, and file names
- Make use of whitespace in the file
  - Examples:

```
x=(2+1)/3.9 #this is a comment
x = (2 + 1) / 3.9        # this is a comment

myarray=numpy.array([[1,1,4,1],[2,1,3,3],[3,4,2,5]])
myarray = numpy.array([ [1, 1, 4, 1],
                        [2, 1, 3, 3],
                        [3, 4, 2, 5] ])
```

- Follow a style guide (example: PEP8 Style Guide - available online)

Modularity

- Make functions that can be used many times in different cases (modularity)
- Keep functions concise
- Make many functions
- Test each function or chunk of code

Tip: Make a written outline for what you want your program to do (ie, different steps). Each bullet point can be made into a function.

Ask for help!

- Learning how to debug a code is a learned skill
- If you struggle for more than 10 min, ask someone for help
  - Different Google search terms
  - Fresh set of eyes
- Rubber Duck Method
  - Explain your code out loud to someone as if they were a rubber duck
  - Makes you think deeper
  - Sometimes saying things out loud make you realize things

Despite common belief, programming is actually very collaborative!