Week 4: debugging | testing

NRSC 7657 Workshop in Advanced Programming for Neuroscientists

course business

- Projects: started?

Debugging


FAST DELIVERY
PREMIUM
BEST PRICE 100% GUARANTEED
DISCOUNT 15%
6 STAGES OF DEBUGGING
1 THAT CAN'T HAPPEN
2 THAT DOESN'T HAPPEN ON MY MACHINE
3 THAT SHOULDN'T HAPPEN
4 WHY DOES THAT HAPPEN?
5 OH I SEE
6 HOW DID THAT EVER WORK?
OUR GLOBAL BRAND SINCE 2010
Baxboxshirt.com
shirtnation.net
leesilk.com
utrustshop.com
moteefe.com
teechip.com
Kybershop.com
Buy our products with 100% guaranteed, high quality items, fast shipping via tracking code in your email.
THANK YOU VERY MUCH !

Debugging

- The first step in debugging: `print("something")`

Debugging

Types of bugs: syntax errors

- Tracebacks

```
a = 2
print(2 + a)
b = a + c
```

```
4
```

```
----------------
NameError
<ipython-input-1-
      1 a = 2
      2 print(2 +
----> 3 b = a + c

NameError: name '
```

```
[116]: rez = loadmat('rez.mat')
```

```
---------------------------------------------------------------------------
NotImplementedError                       Traceback (most recent call last)
<ipython-input-116-852d7d6a9435> in <module>
----> 1 rez = loadmat('rez.mat')

~/opt/anaconda3/envs/NRSC7657/lib/python3.8/site-packages/scipy/io/matlab/mio.py in loadmat(file_name, mdict,
appendmat, **kwargs)
    223     variable_names = kwargs.pop('variable_names', None)
    224     with _open_file_context(file_name, appendmat) as f:
--> 225         MR, _ = mat_reader_factory(f, **kwargs)
    226         matfile_dict = MR.get_variables(variable_names)
    227

~/opt/anaconda3/envs/NRSC7657/lib/python3.8/site-packages/scipy/io/matlab/mio.py in mat_reader_factory(file_n
ame, appendmat, **kwargs)
     78         return MatFile5Reader(byte_stream, **kwargs), file_opened
     79     elif mjv == 2:
---> 80         raise NotImplementedError('Please use HDF reader for matlab v7.3 files')
     81     else:
     82         raise TypeError('Did not recognize version %s' % mjv)

NotImplementedError: Please use HDF reader for matlab v7.3 files
```

Debugging

Types of bugs: semantic errors

- No `Traceback`, but…something didn’t work like you thought it would.

Debugging

Types of bugs: syntax and semantic errors

- The first step in debugging: `print(“something”)`

Debugging

Types of bugs: you can fix both with print statements

- The first step in debugging: `print("something")`
  - Aside: you may see `print "something"` sometimes in older code, this is a relic of python 2. Change it to `print("something")`
    - https://docs.python.org/3/library/2to3.html

Debugging
Python standard debugger

import pdb

Go through this with `djd` in a minute

```
import pdb

x = 3
y = 4
pdb.set_trace()

total = x + y
pdb.set_trace()
```

We have inserted a few breakpoints in this program. The program will pause at each breakpoint (**pdb.set_trace()**). To view a variables contents simply type the variable name:

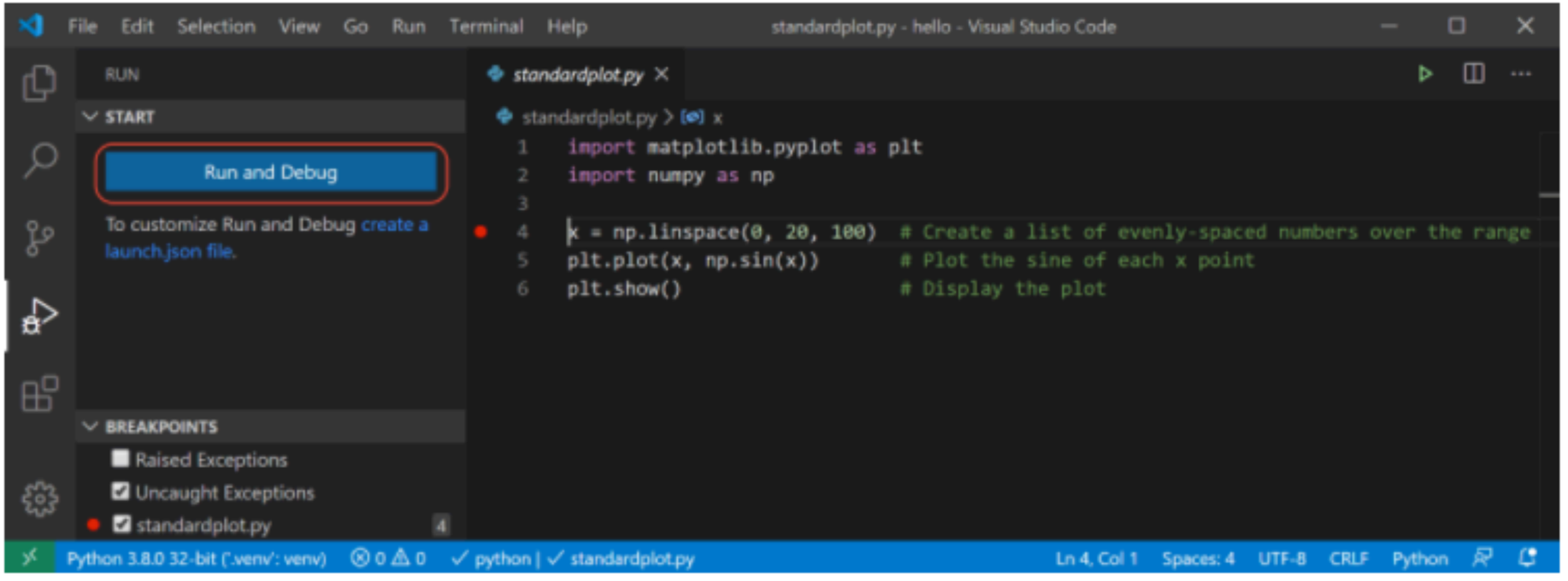
```
$ python3 program.py
(Pdb) x
3
(Pdb) y
4
(Pdb) total
*** NameError: name 'total' is not defined
(Pdb)
```

Debugging

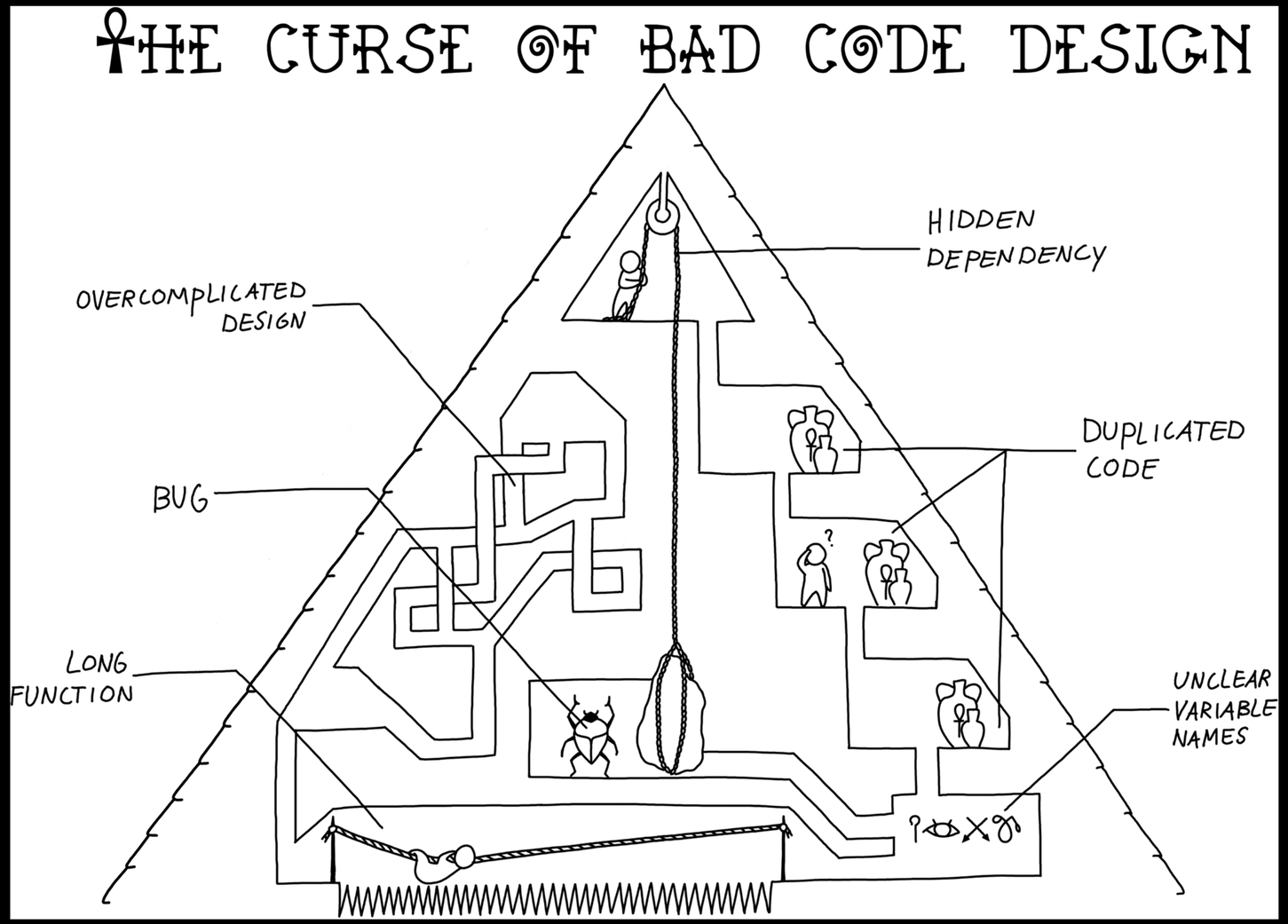
VSCode Run and Debug mode, for scripts

Basic debugging

The simplest way to begin debugging a Python file is to use the **Run** view and click the **Run and Debug** button. When no configuration has been previously set, you will be presented with a list of debugging options. Select the appropriate option to quickly begin debugging your code.



Debugging
Code architecture



PLOS COMPUTATIONAL BIOLOGY

OPEN ACCESS

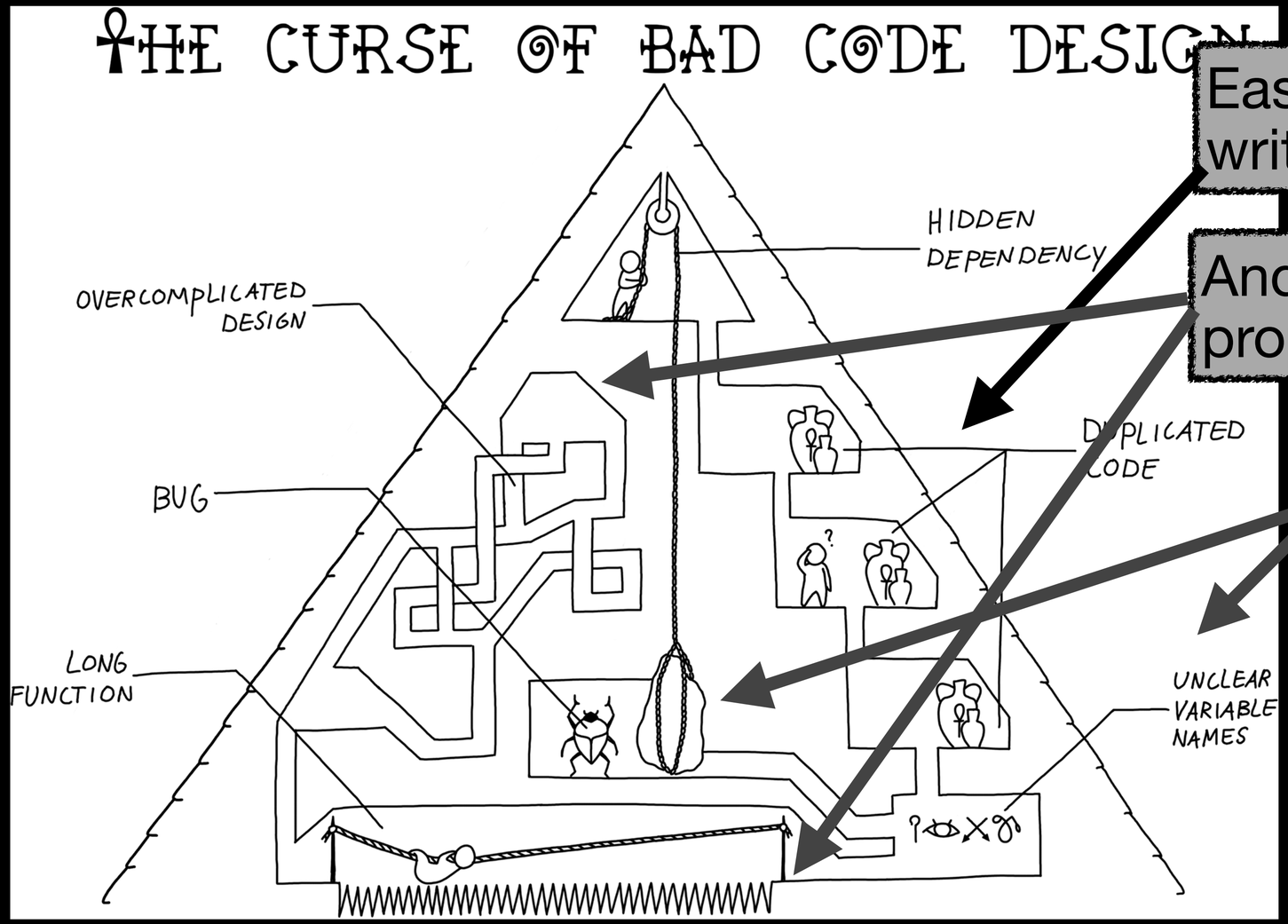
EDITORIAL

Ten simple rules for quick and dirty scientific programming

Gabriel Balaban, Ivar Grytten, Knut Dagestad Rand, Lonneke Scheffer, Geir Kjetil Sandve

Published: March 11, 2021 • https://doi.org/10.1371/journal.pcbi.1008549

Debugging
Code architecture



Ease of debugging is a reason to write functions.

And to keep functions bite-size and properly scoped.

…and understandable

PLOS COMPUTATIONAL BIOLOGY

OPEN ACCESS

EDITORIAL

Ten simple rules for quick and dirty scientific programming

Gabriel Balaban, Ivar Grytten, Knut Dagestad Rand, Lonneke Scheffer, Geir Kjetil Sandve

Published: March 11, 2021 • https://doi.org/10.1371/journal.pcbi.1008549

Testing

- You have written useful code. :tada:
- You (or someone else) wants to write some more code.
  - How do you make sure you (or they) don’t break it?

—> Testing

Other answers:

- Pull requests
- don’t share your code

Testing
What is it?

- A set of functions that check to make sure other functions are behaving as expected.
- Especially good for **semantic errors**. (The code seems like it is running but…)
- Most effective when a codebase is modular

Testing

Manual vs. automated testing

- We will mostly been discussing **unit testing** - do individual units (e.g., a function or a .py with a set of functions) pass tests.
- Can test any unit manually - pass the units (i.e functions) data, and see if it what comes out makes sense. jupyter notebooks are great!
- There are also many mature tools for automated testing - you set up the tests, and these packages run them
- Other forms of testing integration testing, performance testing, …

Testing

Automation packages

- `unittest`
- `nose, nose2`
- `pytest`
- Behavior-driven development: `Lettuce, Radish, behave`
  - Testing but with what each test does in English so you can understand what is happening. For *very * complicated codebases

Testing
Conventions

- Conventions for python testing: `test_*.py` or `*_test.py`

- These can be in folders of your module, or in a test folder of there own

- Within these, functions named `test_*():` that have an `assert`

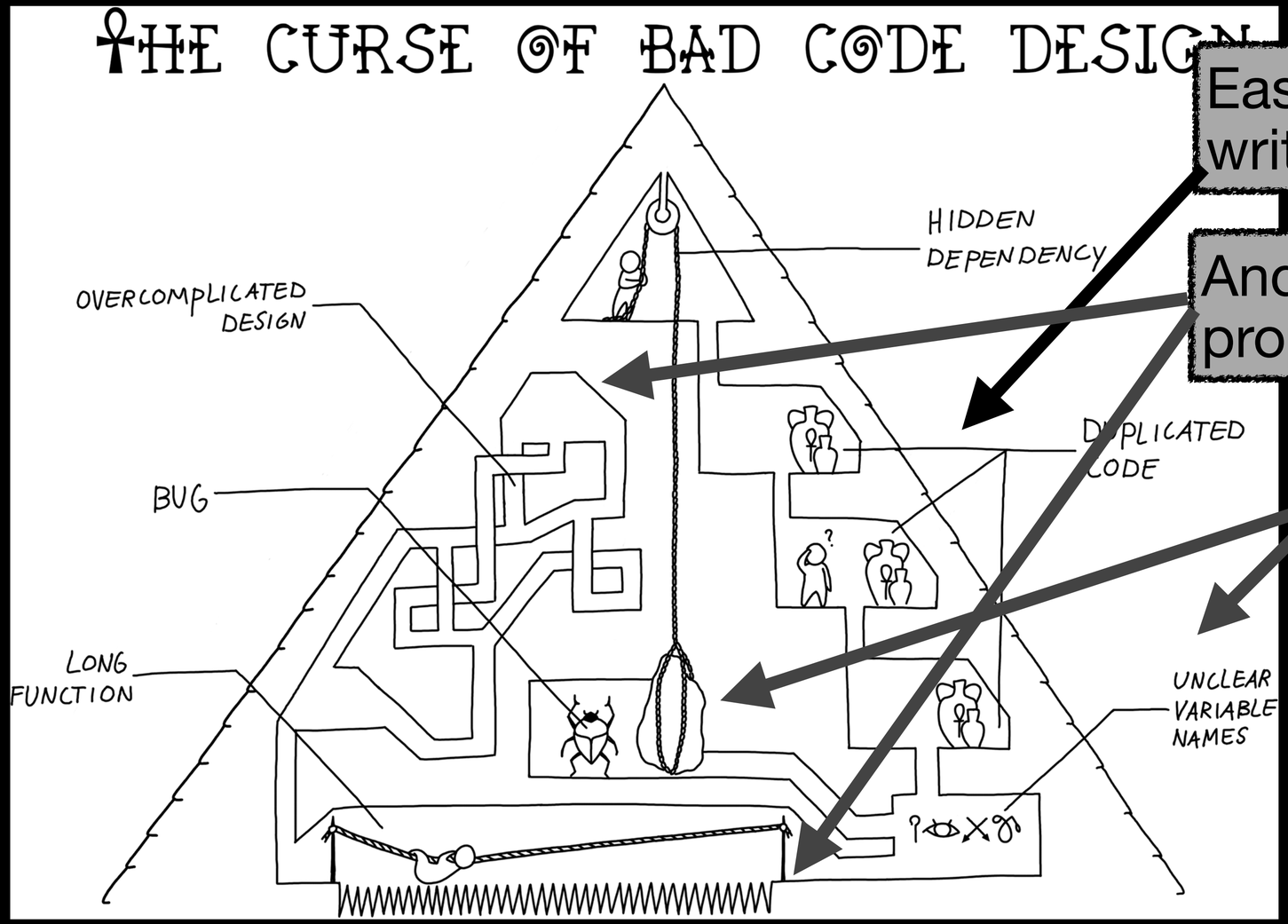
Testing
Example

`patsy`

Testing
Example

```
djd
```

Debugging
Code architecture



Ease of debugging is a reason to write functions.

And to keep functions bite-size and properly scoped.

…and understandable

PLOS COMPUTATIONAL BIOLOGY

OPEN ACCESS

EDITORIAL

Ten simple rules for quick and dirty scientific programming

Gabriel Balaban, Ivar Grytten, Knut Dagestad Rand, Lonneke Scheffer, Geir Kjetil Sandve

Published: March 11, 2021 • https://doi.org/10.1371/journal.pcbi.1008549

Testing

Can also be for data integrity

- Write tests that traverse data structures (like folder with experiments in them, or images or DataFrames) and make sure all the parts are there and shaped the expected way with `assert`

Testing

Kind of like version control

- You are going to test units no matter what - either manually, with automated tests, or with fancy behavior-drive
- In science, manual tests will be 95%-100% of the time
- and you might consider for the rest, the really important bits, setting up some automated tests
  - Manually testing is **critical**. Your code has to work.
  - Automated is a “nice-to-have” that will make your code better.