5. Python

And some debugging tips

Logistics ⚙️

- **Project Guidelines Posted! (iCollege -> Content -> Project Resources)**
  - You have a week to choose one of the posted ideas or propose your own.
  - I'm expecting custom proposals to take some time to consider/negotiate which is why I'm trying to start us early
  - Posted ideas are first-come, first-serve!
- Career Stuff:
  - Reminder on Meta Mock Interviews (link in Discord #announcements)!
  - Virtual Career Fair October 12th! Sign up on Handshake!
  - Who What Wednesdays are happening every Wednesday for a couple more weeks, I think?
- Next Career Day in class is planned for Oct 7th. Any topic suggestions?
  - My plan is to do a "different roles in tech" deep dive
- No one is coming to office hours?

Recap: Pull Requests

Why is a Pull Request called a Pull Request?

- What’s the “Pull”?
- What’s the “Request”?

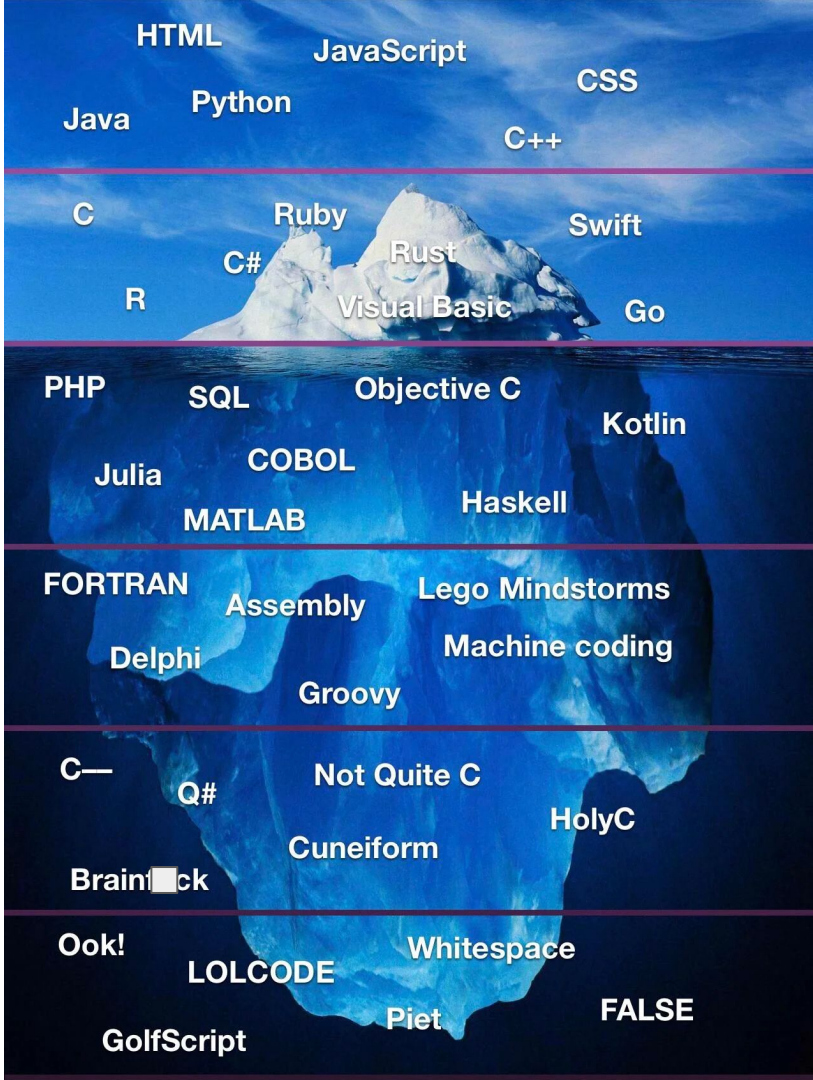
Recap: Pull Requests

- What was the difference between lecture 3 and lecture 4?
- Why do we care?

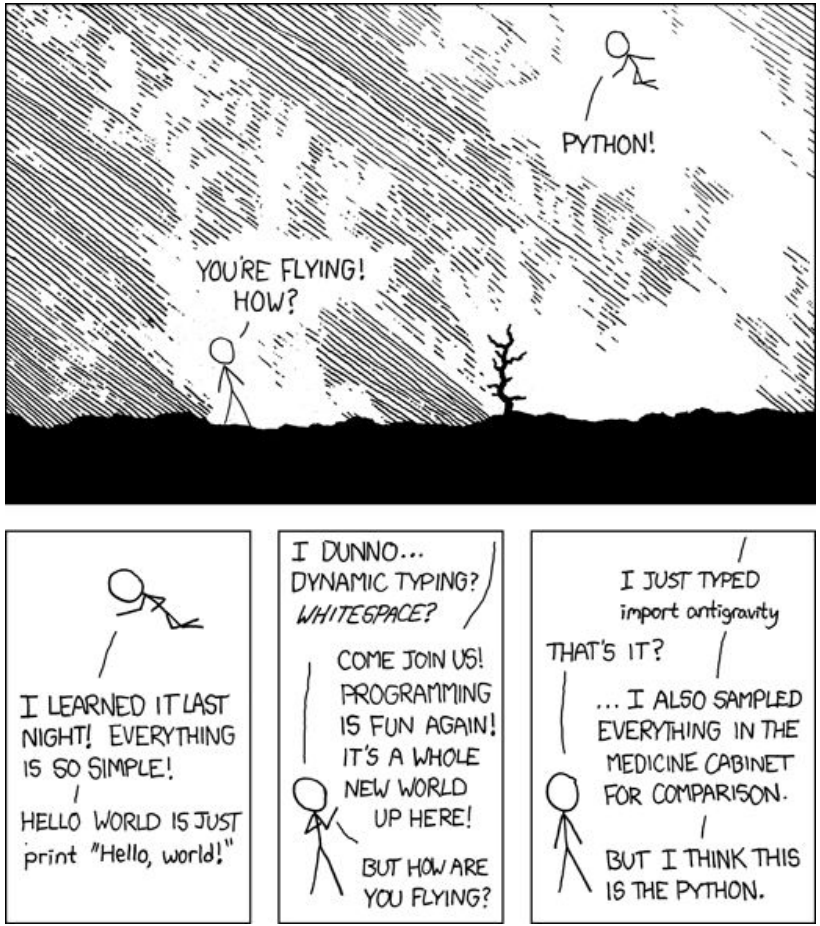
Learning New Languages

How do we do it?

- **Write a bunch of code!**
  - You'll naturally Google stuff you need as it comes up
- **Get someone to warn you about big ideas!**
  - It would be kinda goofy to try to write React without knowing about components.



Python 🐍


PYTHON!
YOU'RE FLYING! HOW?
I LEARNED IT LAST NIGHT! EVERYTHING IS SO SIMPLE!
HELLO WORLD IS JUST print "Hello, world!"
I DUNNO... DYNAMIC TYPING? WHITESPACE?
COME JOIN US! PROGRAMMING IS FUN AGAIN! IT'S A WHOLE NEW WORLD UP HERE!
BUT HOW ARE YOU FLYING?
I JUST TYPED import antigravity
THAT'S IT?
... I ALSO SAMPLED EVERYTHING IN THE MEDICINE CABINET FOR COMPARISON.
BUT I THINK THIS IS THE PYTHON.

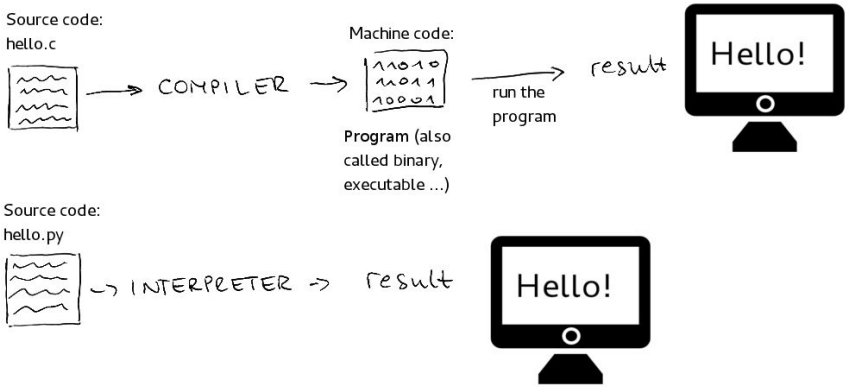
Python is EASIER to write and HARDER to run.

(Another way of saying this: Python is easier on developers and harder on users).

(Your mileage may vary though)

Python vs Java: Big Ideas

- Interpreted vs. Compiled
  - Python is interpreted, Java is compiled*
  - Trade-offs of speed & flexibility
- Static vs. Dynamic Typing
  - Java is statically typed, Python is dynamically typed. That means Python doesn't make you explicitly declare variable types
  - Dynamic typing seems pretty sweet but it's tricky in large codebases. It's harder to read someone else's code when you don't know the type of all variables.



Java

```
int x = 0;
```

Python

```
x = 0
```

Python vs Java: Syntax

```java
class Solution
{
    public int fib(int N)
    {
        if(N <= 1)
            return N;

        int a = 0, b = 1;

        while(N-- > 1)
        {
            int sum = a + b;
            a = b;
            b = sum;
        }
        return b;
    }
}
```

Python vs Java: Syntax

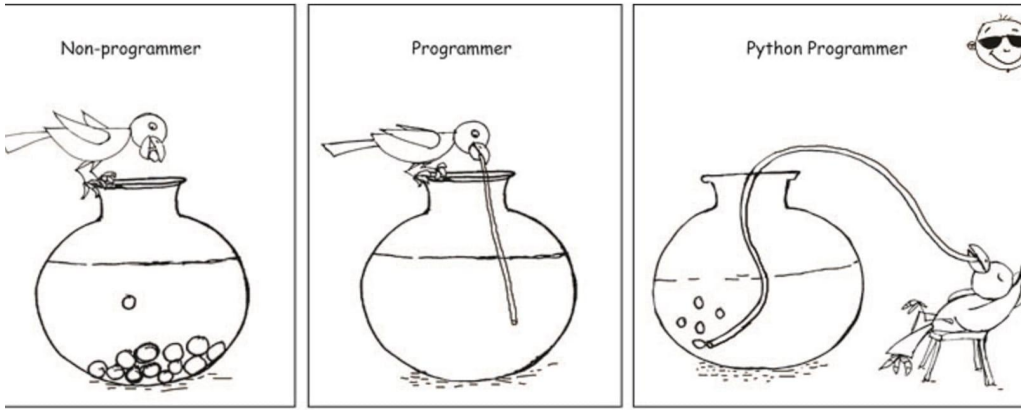
```java
class Solution
{
    public int fib(int N)
    {
        if(N <= 1)
            return N;

        int a = 0, b = 1;

        while(N-- > 1)
        {
            int sum = a + b;
            a = b;
            b = sum;
        }
        return b;
    }
}
```

```python
class Solution:
    def fib(N):
        if N <= 1:
            return N

        a = 0
        b = 1  # could also do a = 0; b = 1

        while N > 1:
            total = a + b
            a = b
            b = total
            N = N - 1  # could also do N -= 1

        return b
```

Python vs Java: Syntax

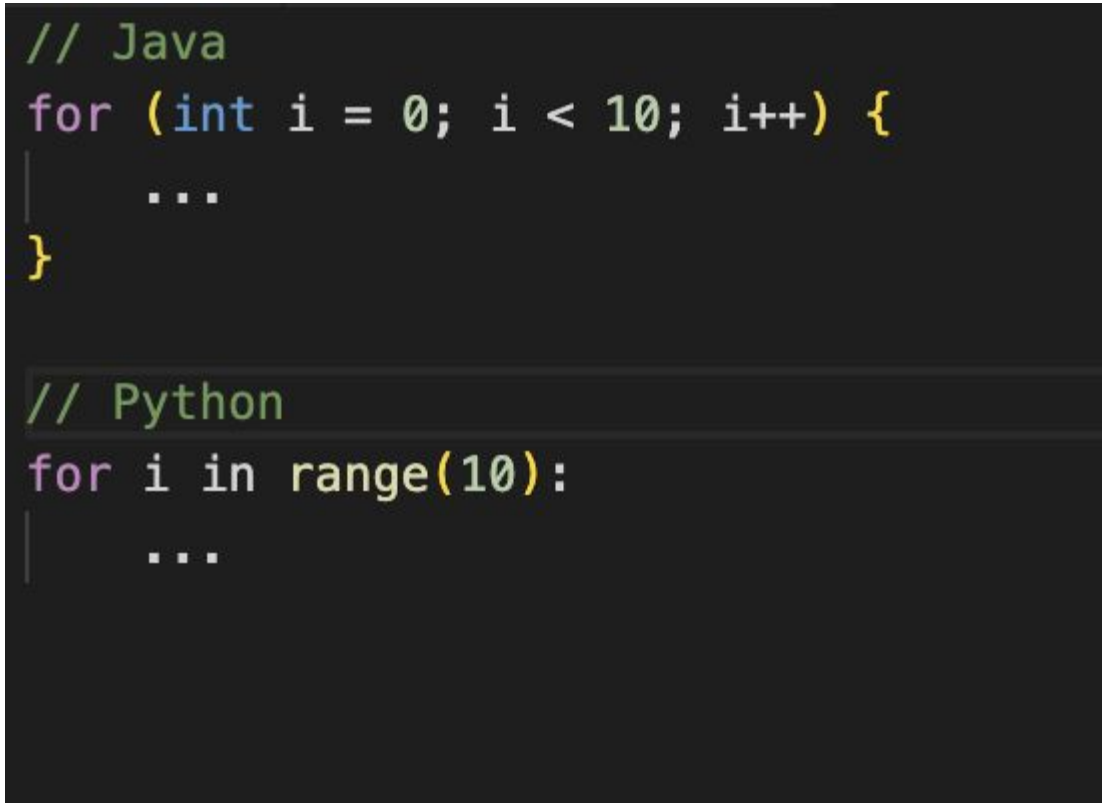
- Colons + indents instead of curly braces
- Use “def” to start functions, “#” to start comments
- “++” and “--” not supported, better to be explicit
  - Stylistically, Python strives to be **readable.**.
- Different syntax for common data structures

```
1  my_set = set()
2  my_set.add("foo")
3  "foo" in my_set  # True
4
5  my_dict = {}
6  my_dict["key"] = "value"
7  my_dict["key"]  # "value"
8
9  my_list = []
10 my_list.append(1)
11 my_list[0]  # 1
12 my_list[0] = 2
13 my_list  # [2]
```



Python vs. Java: For Loops

- Python for loops look a little different
- They can only be written as “**for** *variable* **in** *iterable*”
- This means we need constructs like “range()”

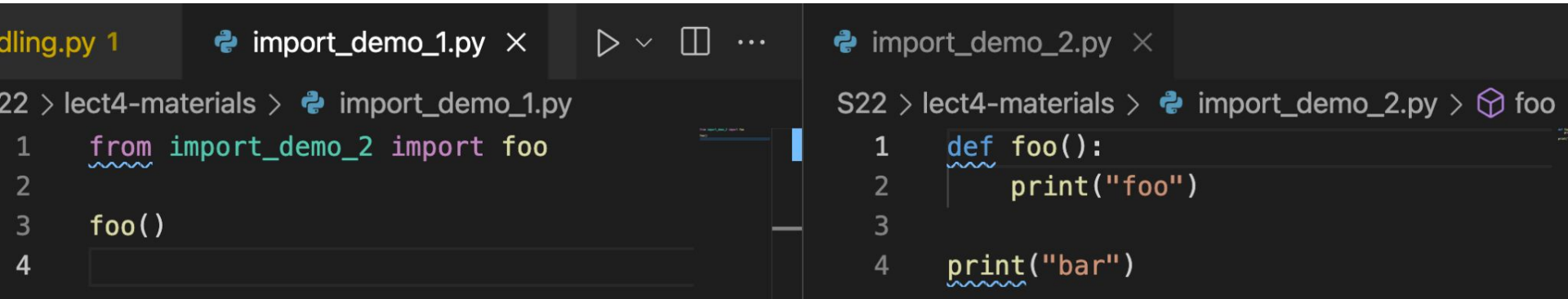


Python tips: imports

- Two ways of importing: specific functions or entire module
  - “import library” vs. “from library import function”
  - When you import a file in EITHER way, the Python interpreter executes the **entire file**

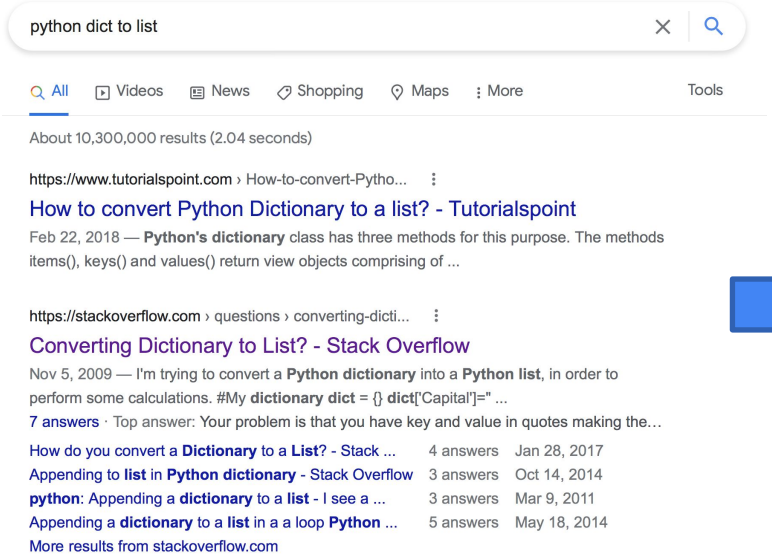
Python tips: imports

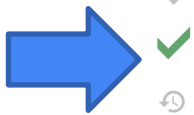
- Two ways of importing: specific functions or entire module
  - “import library” vs. “from library import function”
  - When you import a file in EITHER way, the Python interpreter executes the **entire file**
- **Poll: in the below, when import_demo_1 is run, what will the output be?**
  - Foo, then bar
  - Bar, then foo
  - Just foo



Got more Python questions?

- Documentation: https://docs.python.org/3/
- StackOverflow for common snippets:





Your problem is that you have `key` and `value` in quotes making them strings, i.e. you're setting `aKey` to contain the string `"key"` and not the value of the variable `key`. Also, you're not clearing out the `temp` list, so you're adding to it each time, instead of just having two items in it.

To fix your code, try something like:

```
for key, value in dict.iteritems():
    temp = [key,value]
    dictlist.append(temp)
```

You don't need to copy the loop variables `key` and `value` into another variable before using them so I dropped them out. Similarly, you don't need to use append to build up a list, you can just specify it between square brackets as shown above. And we could have done `dictlist.append([key,value])` if we wanted to be as brief as possible.

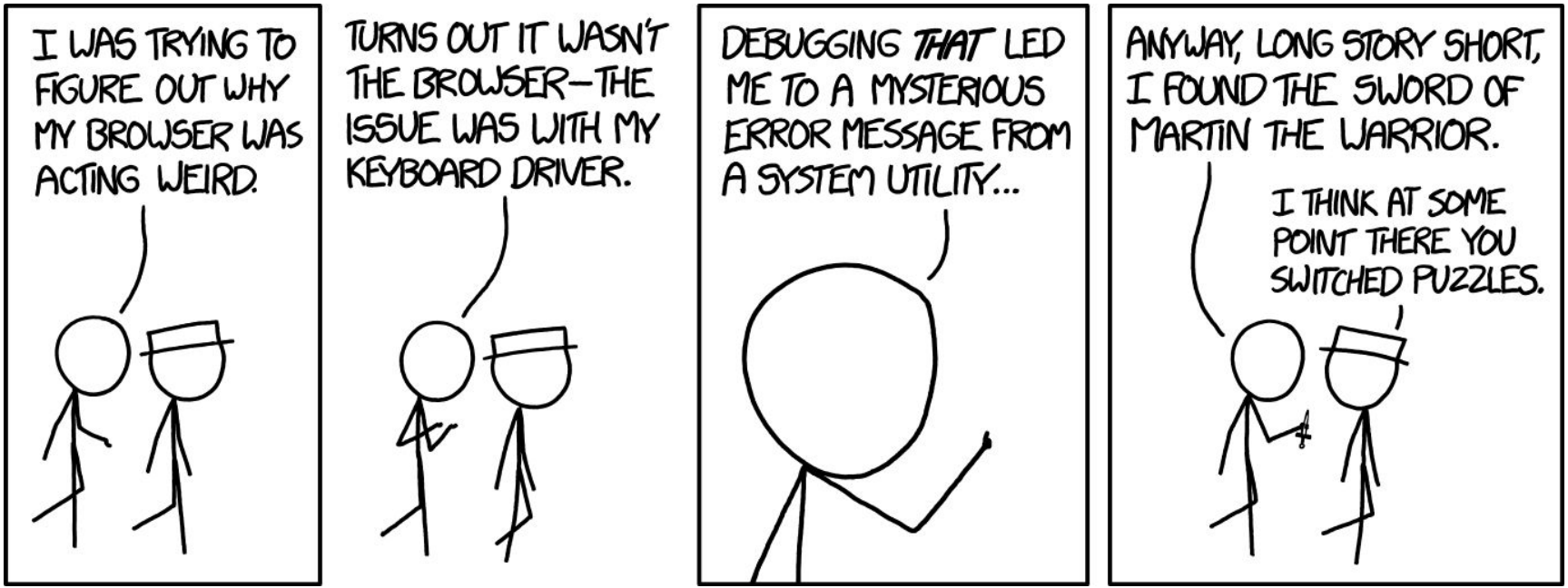
Or just use `dict.items()` as has been suggested.

Python Practice! 🐍

- I’m sending out a link to a Gist with a “Python obstacle course.”
- We’ll reconvene in 10 minutes to walk through solutions!
- If you can’t access Discord for some reason, here’s a link: https://gist.github.com/jomart-gsu/e5bdd1b835a1fc5e788a93e8bce92329

Break

Debugging


I WAS TRYING TO FIGURE OUT WHY MY BROWSER WAS ACTING WEIRD.
TURNS OUT IT WASN'T THE BROWSER—THE ISSUE WAS WITH MY KEYBOARD DRIVER.
DEBUGGING THAT LED ME TO A MYSTERIOUS ERROR MESSAGE FROM A SYSTEM UTILITY...
ANYWAY, LONG STORY SHORT, I FOUND THE SWORD OF MARTIN THE WARRIOR.
I THINK AT SOME POINT THERE YOU SWITCHED PUZZLES.

Mindset

Mindset

- For your first years as a computer science student, stack traces were scary and indecipherable. Most of us learned to ignore them and ask for help when stuff broke. **This was a good coping mechanism at the time!**
- However, we are now mature enough as programmers to get useful information out of our program output. When stuff breaks, you can probably figure out why! And if you can’t, Google probably can!
- **When to ask for help?** Stuck spinning wheels on same problem for more than an hour, not having new ideas about how to approach it, error message doesn’t make any sense.

Debugging

- **Debugging begins before your code runs.**
- Three major techniques...

Debugging

Bottom-up development (modularity)

- Think about how to write code in small, testable steps! If you're writing a program A that calls subroutines B and C, and B calls subroutine D, you want to make sure D is written and reliably tested before moving onto B and C. Then, if A breaks, you know the problem (probably) isn't D.

```
def complicated_process(x):
    y = do_something(x)  #  this can be tested!
    z = do_another_thing(y)  # so can this!
    result = do_one_final_thing(z)  # etc
    return result
```

Debugging

Failing fast (offensive coding)

- Kind of like a micro-version of bottom-up development. Your code should constantly be checking for weird stuff happening, and crashing itself when something's wrong. This comes up a little more when algorithmically complex things are happening.
- *When might this backfire?*

```
def complicated_process(x):
    y = do_something(x)  #  this can be tested!
    z = do_another_thing(y)  # so can this!
    result = do_one_final_thing(z)  # etc
    assert(result > 0)
    return result
```

```
def simple_process():
    x = 1
    assert(x == 1)
    return x
```

← Non-example!!!

Debugging

Logging

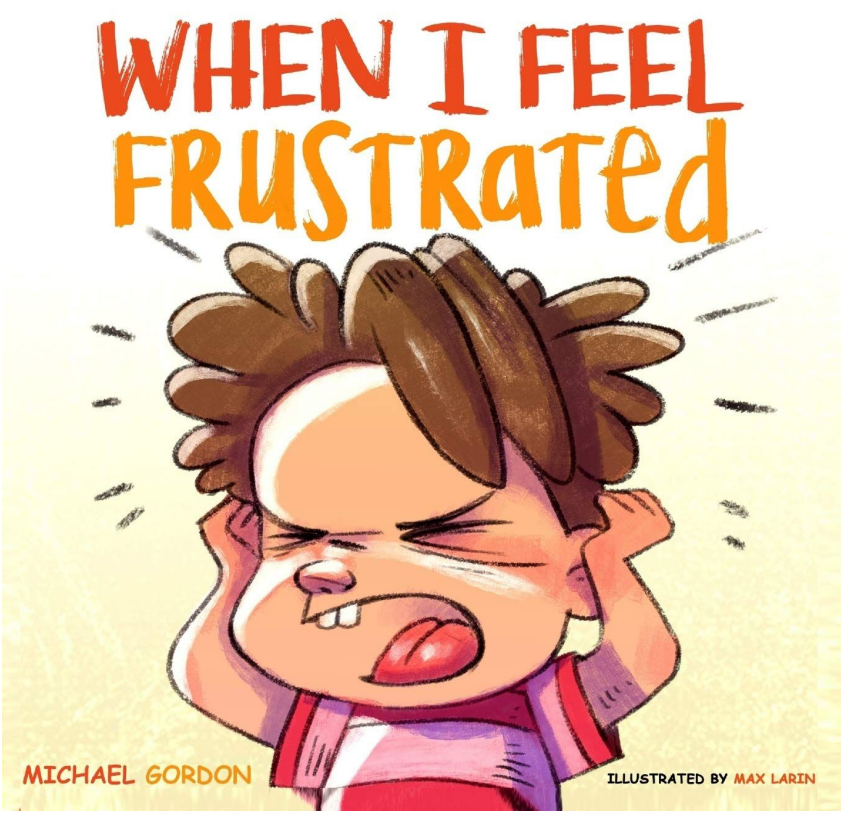
- If your program fails, you want to have as much information about what happened as possible. This is especially true for long-running programs or complicated systems where reproducing failures is costly or impossible.

```
def complicated_process(x):
    y = do_something(x)  #  this can be tested!
    print(f"y: {y}")
    z = do_another_thing(y)  # so can this!
    print(f"z: {z}")
    result = do_one_final_thing(z)  # etc
    assert(result > 0)
    return result
```

```
def simple_process():
    x = 1
    print("Got here!")
    assert(x == 1)
    print("x was equal to 1")
    return x
```

← Non-example!!!

So Your Code Isn’t Working 😡



So Your Code Isn’t Working 🤬

- You wrote nice, testable modules; you added logging and assert statements at crucial points, and you still don’t know why your code isn’t doing what it’s supposed to. What now?

So Your Code Isn’t Working 😡

- You wrote nice, testable modules; you added logging and assert statements at crucial points, and you still don’t know why your code isn’t doing what it’s supposed to. What now?
- **Use the debugger**.

```
import pdb
pdb.set_trace()
```

(← old school way if you don’t have fancy IDE features)

Demo time!

Let’s debug a code snippet!

So Your Code Isn’t Working 🤬

- You wrote nice, testable modules; you added logging and assert statements at crucial points, and you still don’t know why your code isn’t Doing the Thing. What now?
- **Use the debugger**.
- **Talk to someone or something.** This is often called “rubber ducking” because it doesn’t really matter if you’re talking to a person or a rubber duck you keep at your desk - the process of describing out loud what’s happening so someone else could understand it can untangle your thinking.

So Your Code Isn’t Working 😴

- You wrote nice, testable modules; you added logging and assert statements at crucial points, and you still don’t know why your code isn’t Doing the Thing. What now?
- **Use the debugger**.
- **Talk to someone or something.** This is often called “rubber ducking” because it doesn’t really matter if you’re talking to a person or a rubber duck you keep at your desk - the process of describing out loud what’s happening so someone else could understand it can untangle your thinking.
- **Walk away (if time permits).** If you’re not having fresh realizations for an entire hour, it’s time to focus on something else for a while. Even if you ARE having fresh realizations, know when to call it quits.
    - Sleep is REALLY, REALLY good for your cognition and problem-solving :)

Checklist: What to do if you see an error message ✅

- Does the error message tell you what to do? Do that!

```
git commit -m "initial commit"
*** Please tell me who you are.  Run
    git config --global user.email "you@example.com"
    git config --global user.name "Your Name"
to set your account's default identity.

Omit --global to set the identity only in this repository.

fatal: empty ident name (for <me@DESKTOP-T5KF9QB.localdomain>) not allowed
```

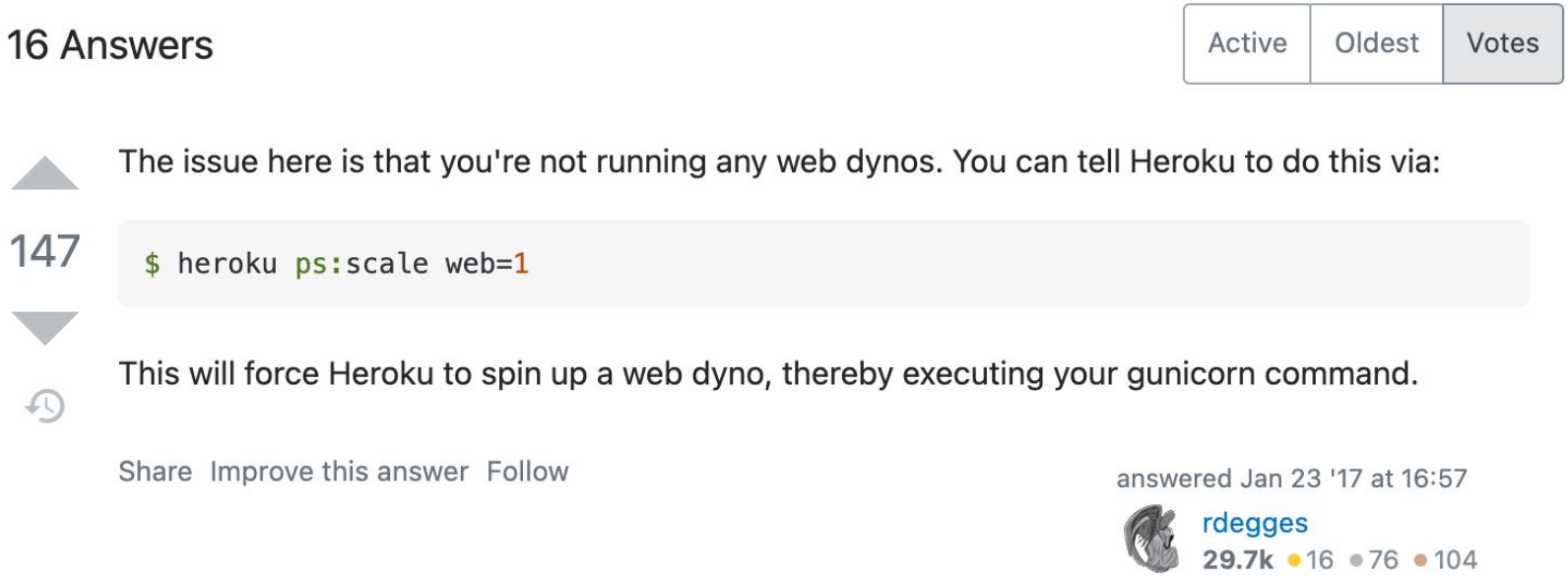
Checklist: What to do if you see an error message ✅

- Can you find which part is related to your code?

```
2022-01-19T16:22:50.446456+00:00 app[web.1]: response = self.full_dispatch_request()
2022-01-19T16:22:50.446459+00:00 app[web.1]: File "/app/.heroku/python/lib/python3.9/site-packages/flask/app.py", line 1518, in full_dispatch_request
2022-01-19T16:22:50.446459+00:00 app[web.1]: rv = self.handle_user_exception(e)
2022-01-19T16:22:50.446459+00:00 app[web.1]: File "/app/.heroku/python/lib/python3.9/site-packages/flask/app.py", line 1516, in full_dispatch_request
2022-01-19T16:22:50.446460+00:00 app[web.1]: rv = self.dispatch_request()
2022-01-19T16:22:50.446460+00:00 app[web.1]: File "/app/.heroku/python/lib/python3.9/site-packages/flask/app.py", line 1502, in dispatch_request
2022-01-19T16:22:50.446460+00:00 app[web.1]: return self.ensure_sync(self.view_functions[rule.endpoint])(**req.view_args)
2022-01-19T16:22:50.446461+00:00 app[web.1]: File "/app/app.py", line 23, in index
2022-01-19T16:22:50.446462+00:00 app[web.1]: (title, tagline, genre, poster_image) = get_movie_data(movie_id)
2022-01-19T16:22:50.446462+00:00 app[web.1]: File "/app/tmdb.py", line 24, in get_movie_data
2022-01-19T16:22:50.446462+00:00 app[web.1]: title = json_response["title"]
2022-01-19T16:22:50.446463+00:00 app[web.1]: KeyError: 'title'
2022-01-19T16:22:50.448896+00:00 heroku[router]: at=info method=GET path="/" host=rocky-hamlet-91877.herokuapp.com request_id=ac1ff50a-6747-46ca-9f2e-cd0493669a67 fwd="131.96.220.102" dyno=web.1 connect=0ms service=51ms status=500 bytes=15133 protocol=https
2022-01-19T16:22:50.636455+00:00 app[web.1]: 10.1.89.95 - - [19/Jan/2022 16:22:50] "GET /?__debugger__=yes&cmd=resource&f=style.css HTTP/1.1" 200 -
2022-01-19T16:22:50.640445+00:00 heroku[router]: at=info method=GET path="/?__debugger__=yes&cmd=resource&f=style.css" host=rocky-hamlet-91877.herokuapp.com request_id=e08d0c72-3511-439c-a02a-d00beebe73df fwd="131.96.220.102" dyno=web.1 connect=0ms service=3ms status=200 bytes=6859 protocol=https
```

Checklist: What to do if you see an error message ✅

- Can you Google the specific error?
  - Think back to the Heroku example!



Bye 👋 (+ Python Free Work Time)

- Before next class, do HW5.
- Next class, we'll talk web dev basics. That's going to be a pretty hefty demo, so if you want a head start, I'll be posting the recording of the same lecture from last year :)