Lecture 12:

LIFE SKILLS 2 & 3:
DEBUGGING & DOCUMENTATION

CSC111: Introduction to CS through Programming

R. Jordan Crouser

Assistant Professor of Computer Science

Smith College

Overview

- ✓Monday: Loops
  - ✓`for…in` (looping through items in a list)
  - ✓the `range()` function (getting a list of numbers)
  - ✓`while` (looping until something happens)
- ✓Lab: Loops
- Wednesday: Life Skill #2/3: Debugging & Documentation
- Friday: Dictionaries and Sets

Assignment 3 (what's hard so far?)

> In this assignment, you will write a python program that **simulates a Magic 8 Ball** (a toy first popularized in the 1950s, produced by Mattel).
>
> The user interface of your Magic 8 Ball should first ask the user to input **a question**. For example:
>
> ```
> Your question: Will lunch at Chase/Duckett be good today?
> ```
>
> Your program will then output a randomly selected answer, e.g.
>
> ```
> Ask again later.
> ```

Discussion

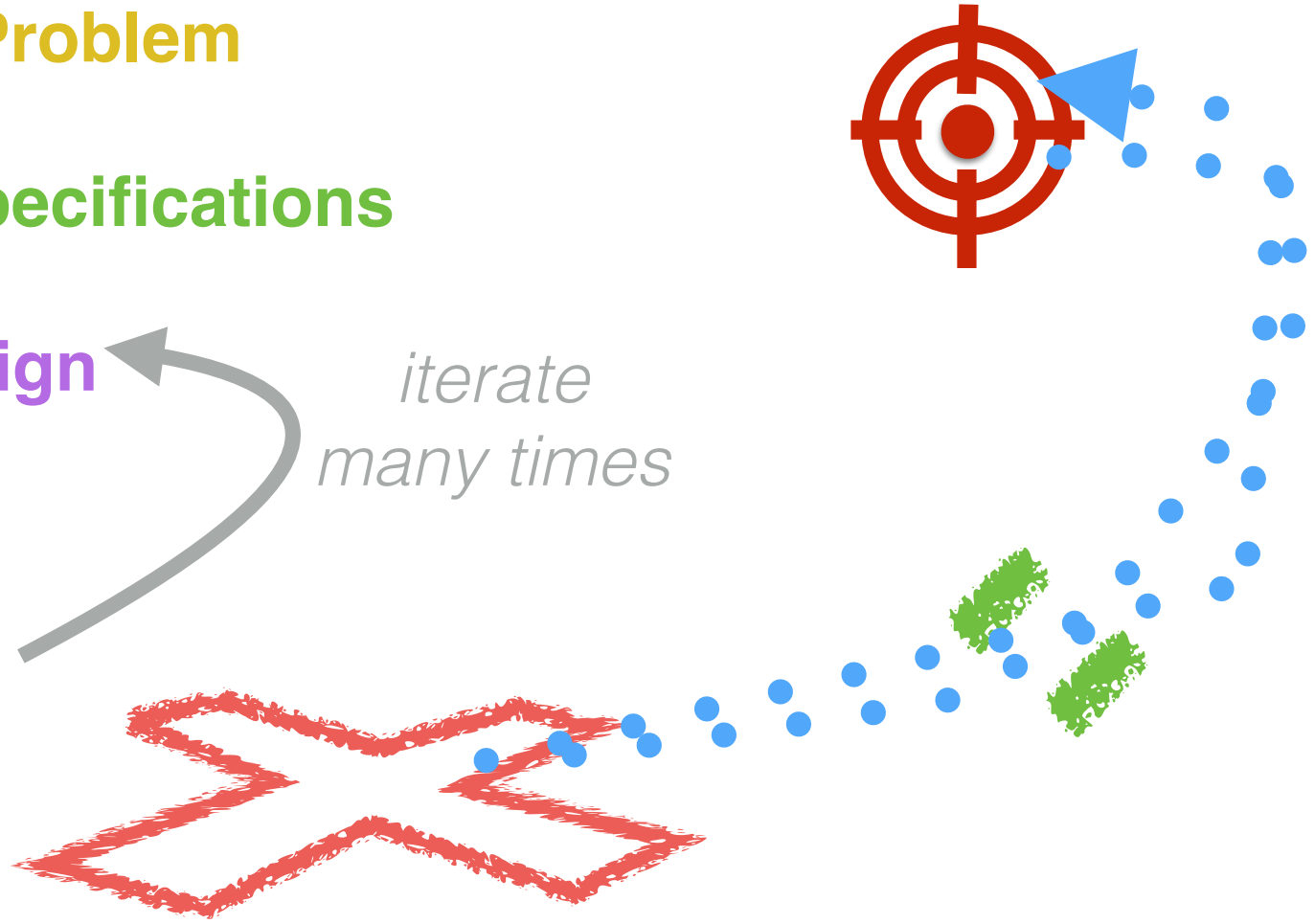
What is your plan

for getting **unstuck**?

RECAP: the programming process

- Analyze the **Problem**
- Determine **Specifications**
- *Refine the* ~~Create a~~ **Design**
- **Implement**
- Test & Debug

iterate many times



Fun history: the term “debug”

9/9

0800 Arctan started
1000 " stopped - arctan ✓ { 1.2700 9.037 847 025
9.037 846 795 correct
13" tape (032) MP - MC ~~1.9826447000~~ ~~2.13047641515~~) 4.615925059(-2)
(033) PRO 2 2.13047641515
correct 2.13067641515
Relays 6-2 in 033 failed special speed test
In relay " 10.00 test.
Relays changed
1100 Started Cosine Tape (Sine check)
1525 Started Multy Adder Test.

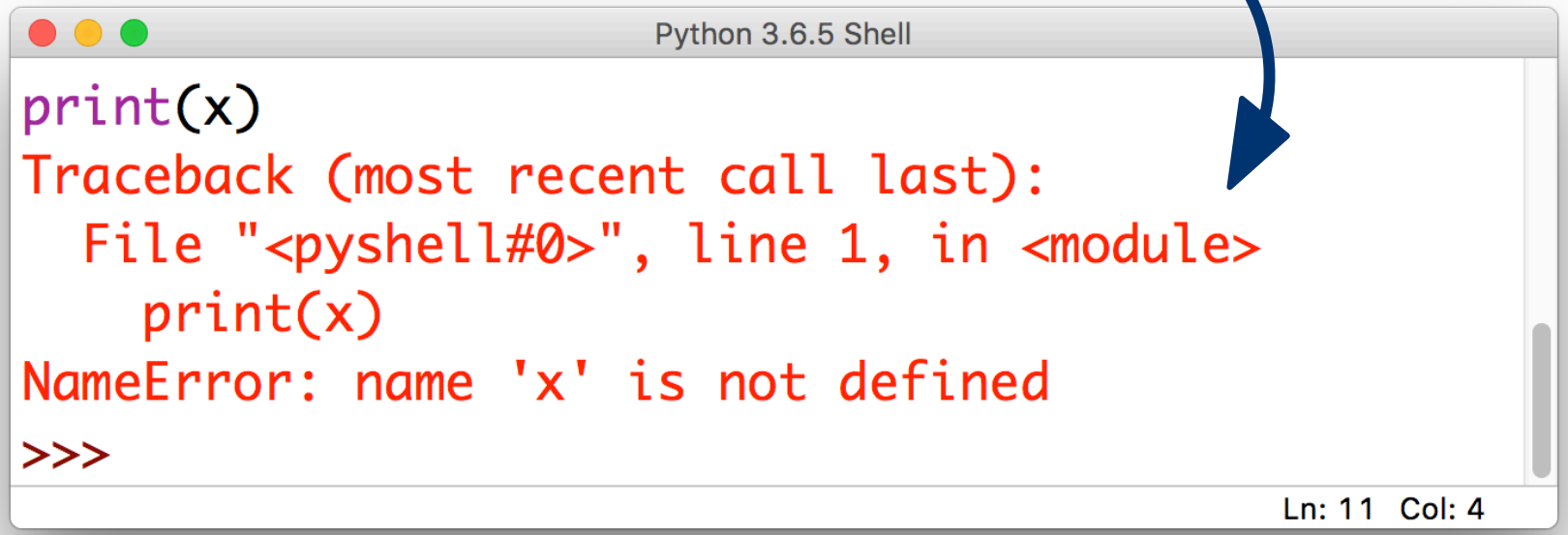
1545 Relay #70 Panel F
(moth) in relay.
First actual case of bug being found.
~~1700~~ 1630 Arctangent started.
1700 closed down.

RDML Grace M. Hopper

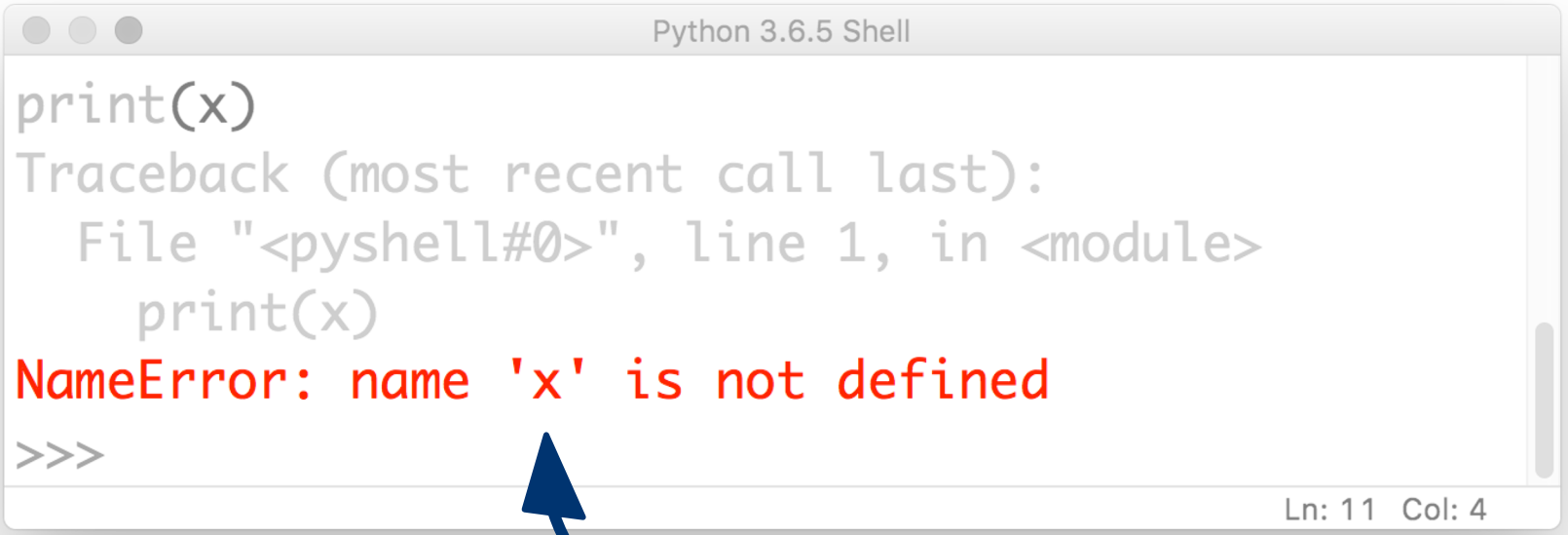
b.1906 – d.1992

Some problems are obvious

this is called
an **`Exception`**



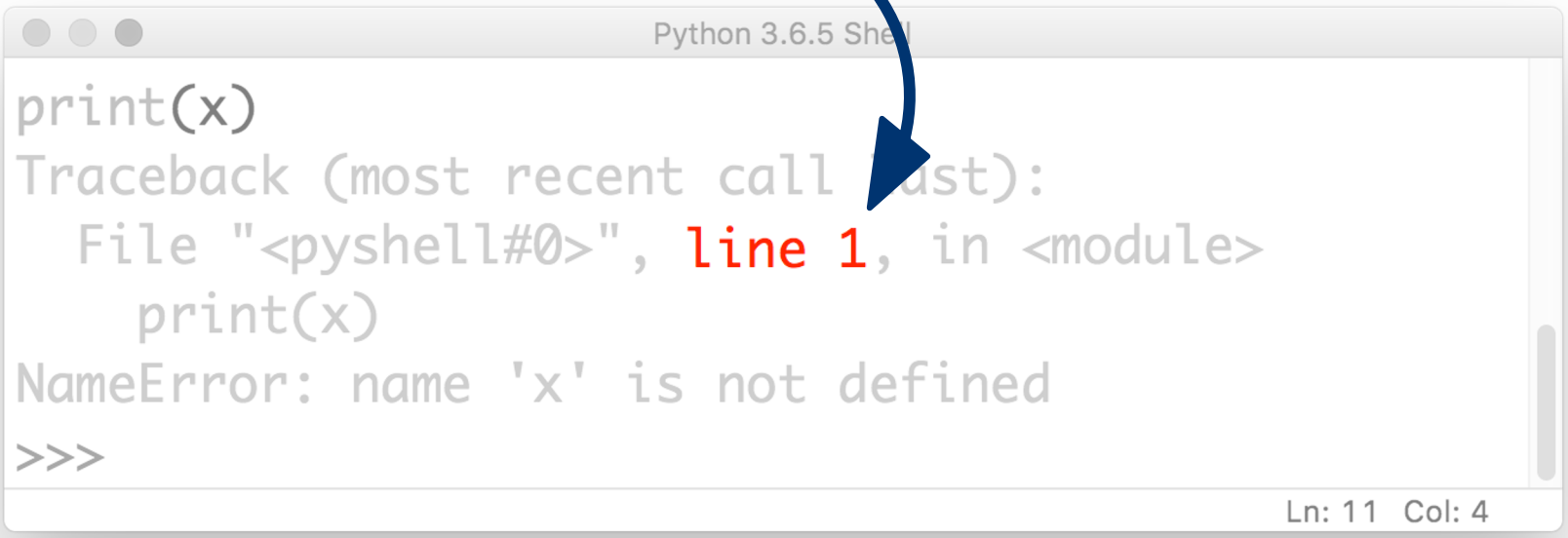
Some problems are obvious



the kind of error gives you
a **clue** about what the problem is

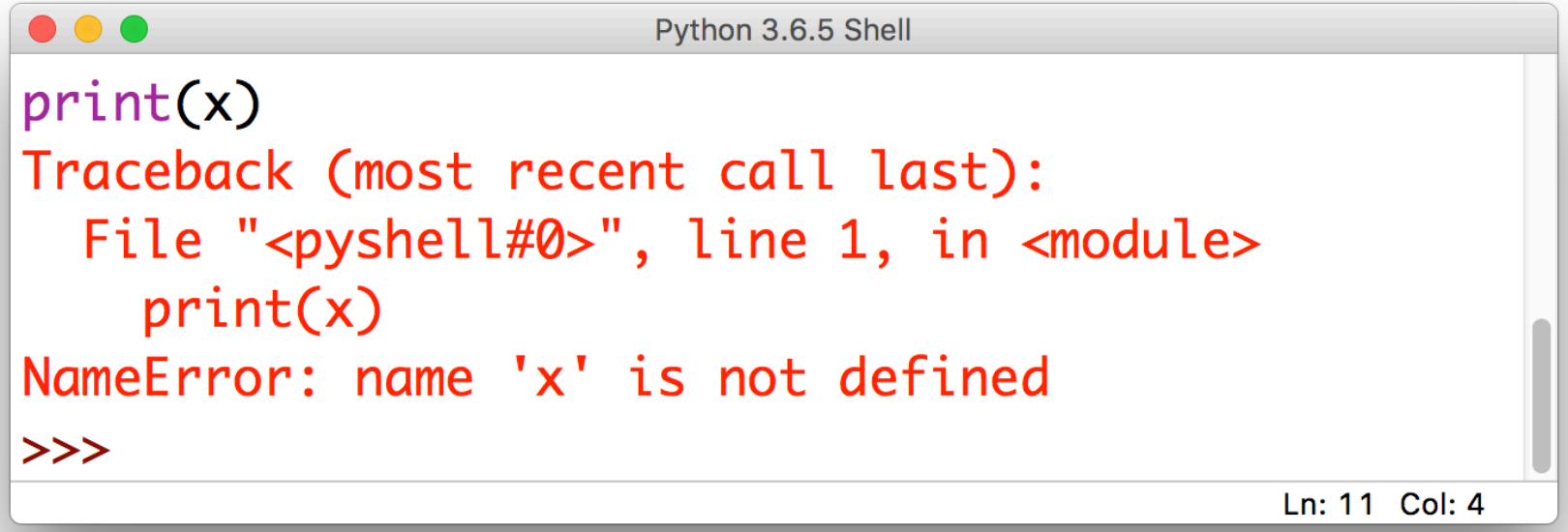
Some problems are obvious

it also tells you **where** the problem is
(but be careful!)



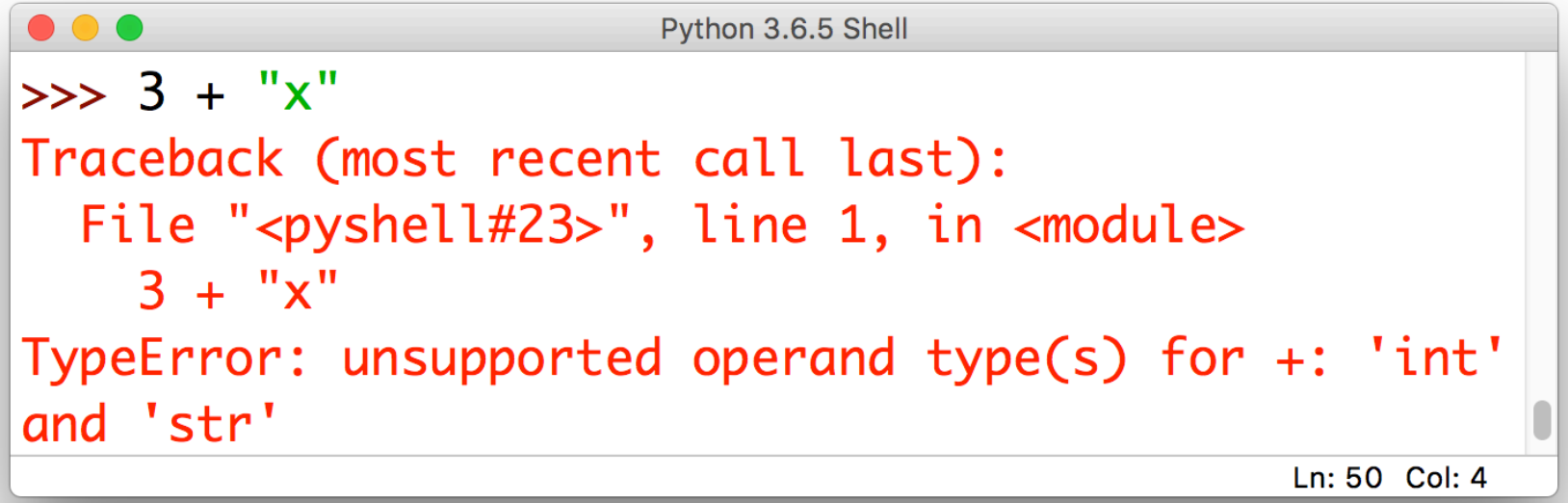
Common **`Exceptions`**

- **`NameError`**: raised when Python can't find the thing you're referring to (a variable or a function)



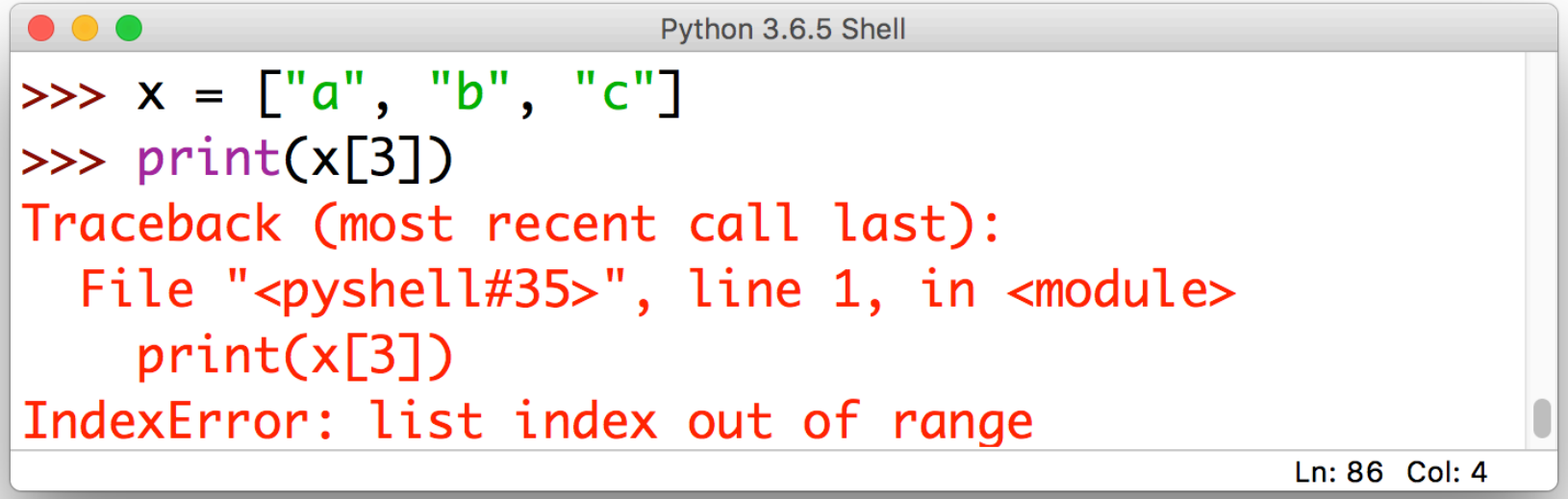
Common **`Exceptions`**

- **`TypeError`**: raised when you try to perform an operation on an object that's not the right type (i.e. a `string` instead of a number)



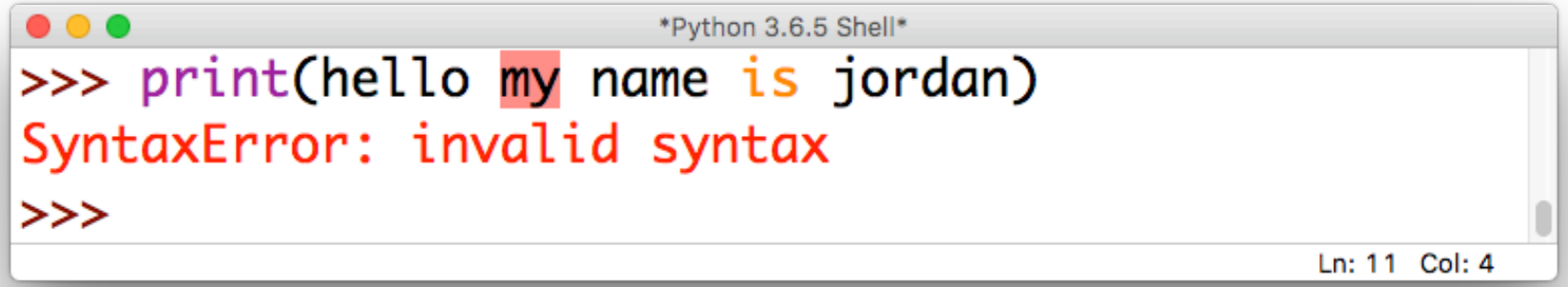
Common **`Exceptions`**

- **`IndexError`**: raised when you try to use an index that's out of bounds



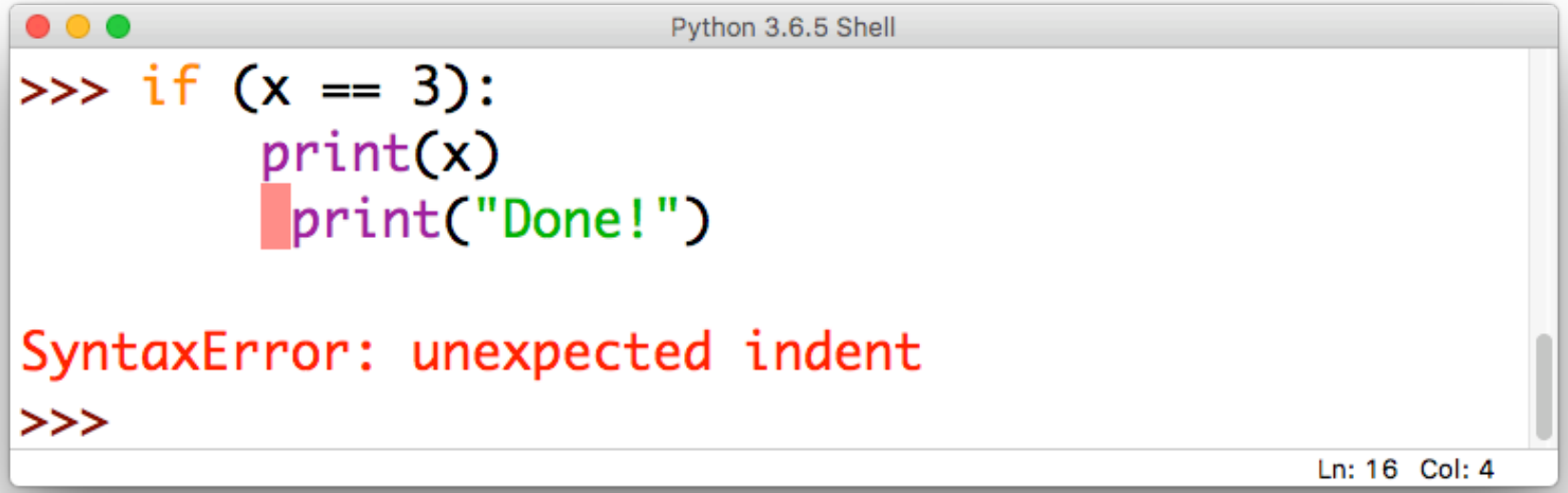
Common **`Exceptions`**

- **`SyntaxError`**: raised when you try to run a command that isn't a valid Python statement



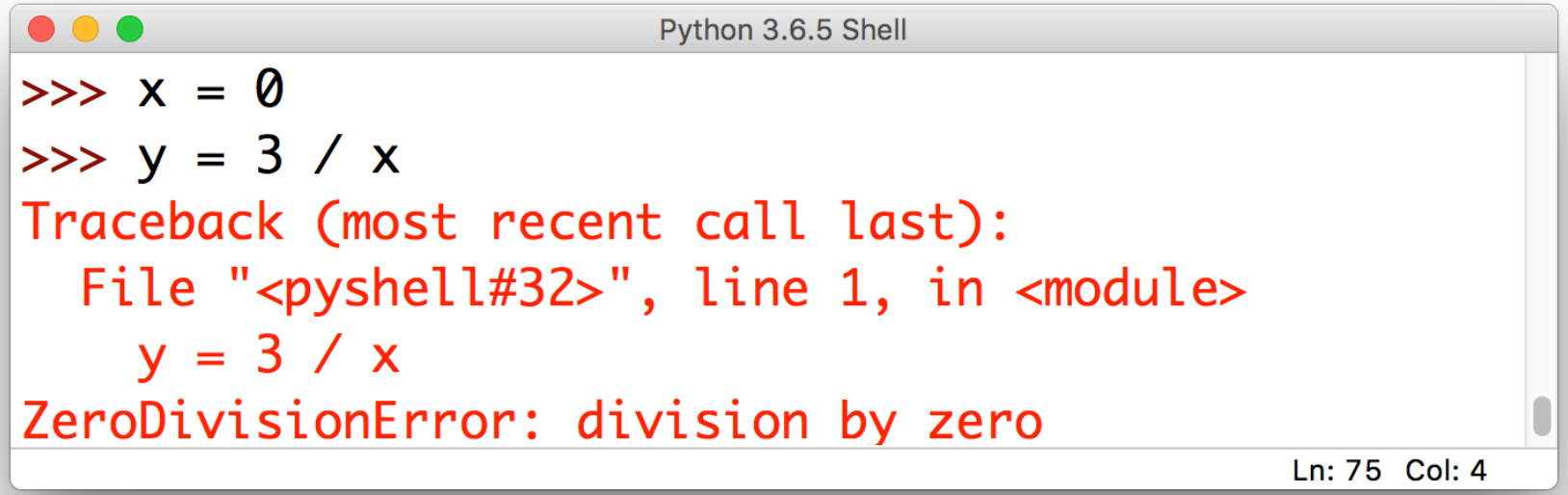
Common **`Exceptions`**

- **`SyntaxError`**: also raised if your indentation is messed up (this is a special kind of `SyntaxError` called an `IndentationError`)



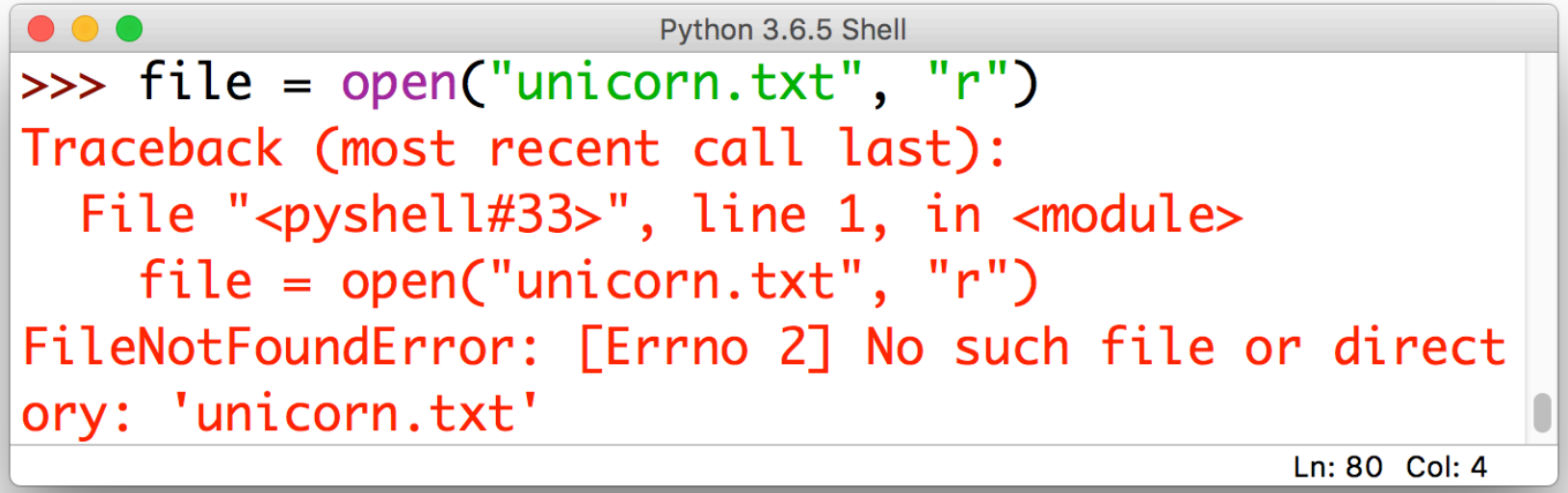
Common **`Exceptions`**

- **`ZeroDivisionError`**: raised when you try to divide by zero (or do modular arithmetic with zero)



Common **`Exceptions`**

- **`FileNotFoundError`**: raised when Python can't find the thing you're referring to (a file)



Common **`Exceptions`**

- **`UnicodeDecodeError`**: raised when you try to read a file that has weird characters in it (most common culprit: *apostrophe* vs. the *single quote*)

Python 3.6.5 Shell

```
>>> file = open("alice.txt", "r")
>>> file.read()
Traceback (most recent call last):
  File "<pyshell#30>", line 1, in <module>
    file.read()
  File "/Library/Frameworks/Python.framework/Versio
ns/3.6/lib/python3.6/encodings/ascii.py", line 26,
in decode
    return codecs.ascii_decode(input, self.errors)[
0]
UnicodeDecodeError: 'ascii' codec can't decode byte
0xe2 in position 219: ordinal not in range(128)
```

Less common **`Exceptions`**

Did your program throw an **`Exception`** not listed here?

Look it up at:

https://docs.python.org/3/library/exceptions.html

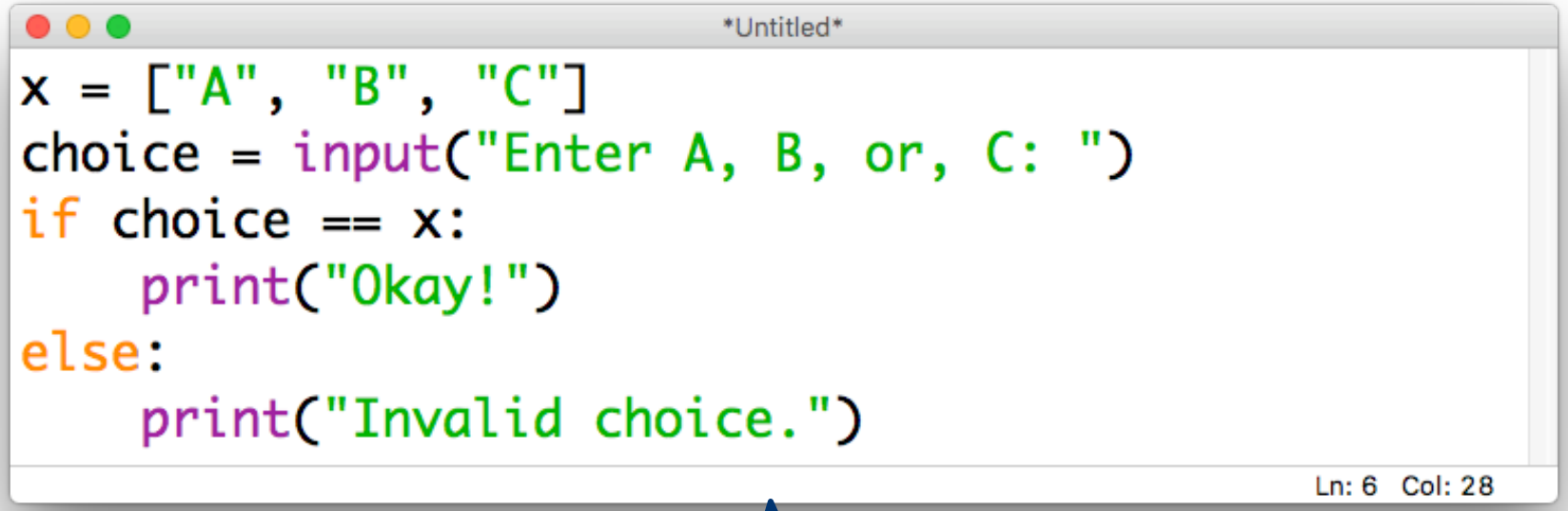
`Exceptions` = relatively easy to fix

Why would I say that?

What's the alternative?

Logical errors

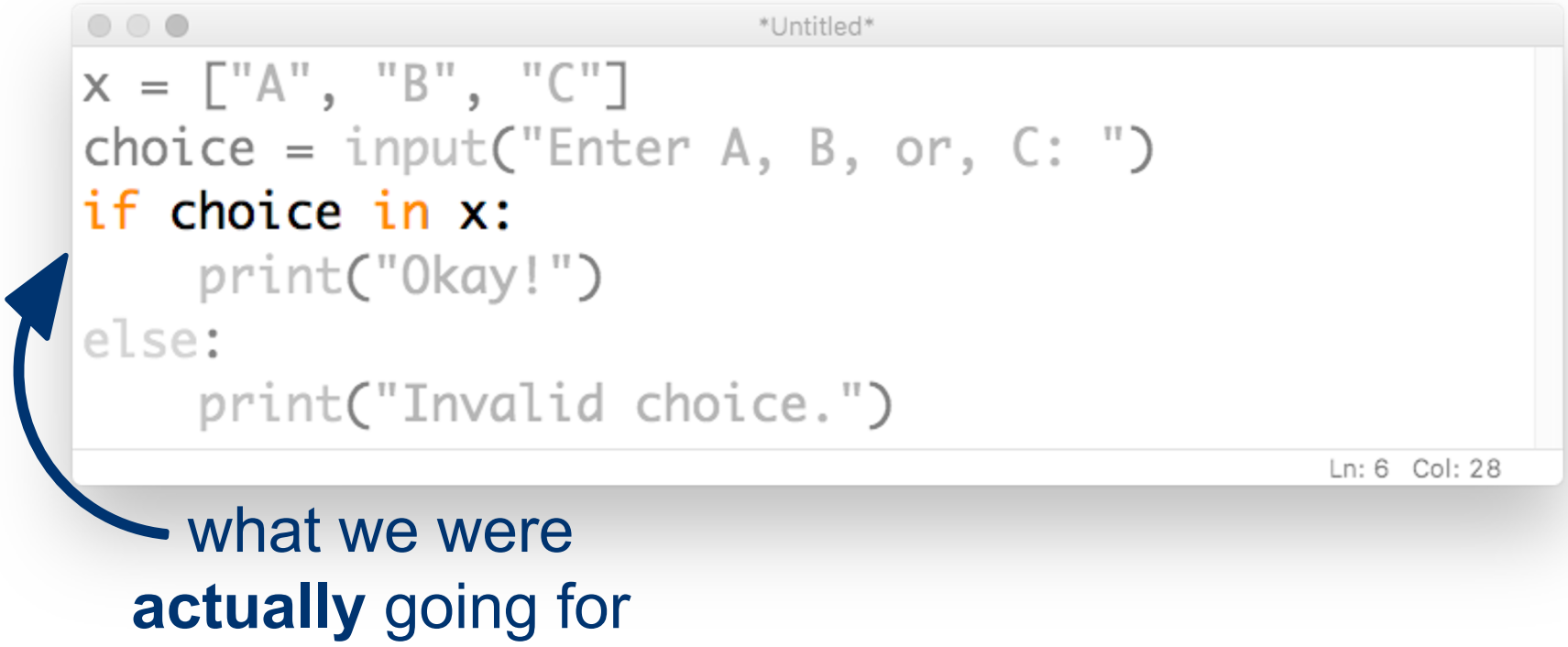
- Mistakes in the **reasoning** behind the code (though the statements are valid and there are no `Exceptions`), e.g.



perfectly **valid**
(just not what we wanted)

Logical errors

- Mistakes in the **reasoning** behind the code (though the statements are valid and there are no `Exceptions`), e.g.



An analogy

Syntactic Error

Their is no reason to be concerned.

Logical Error

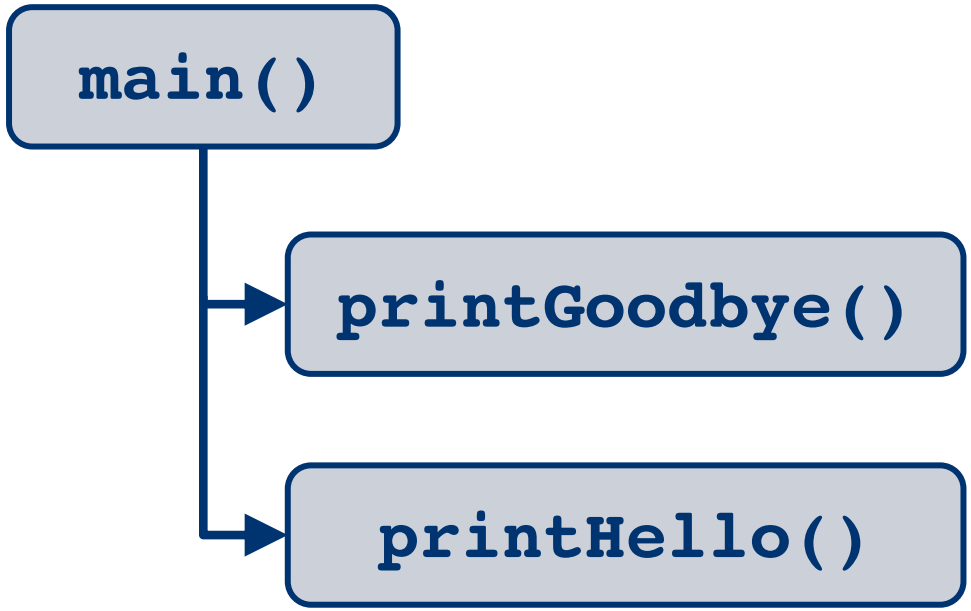
If an animal is green, it must be a frog.

Discussion

How do you find and fix **logical** errors?

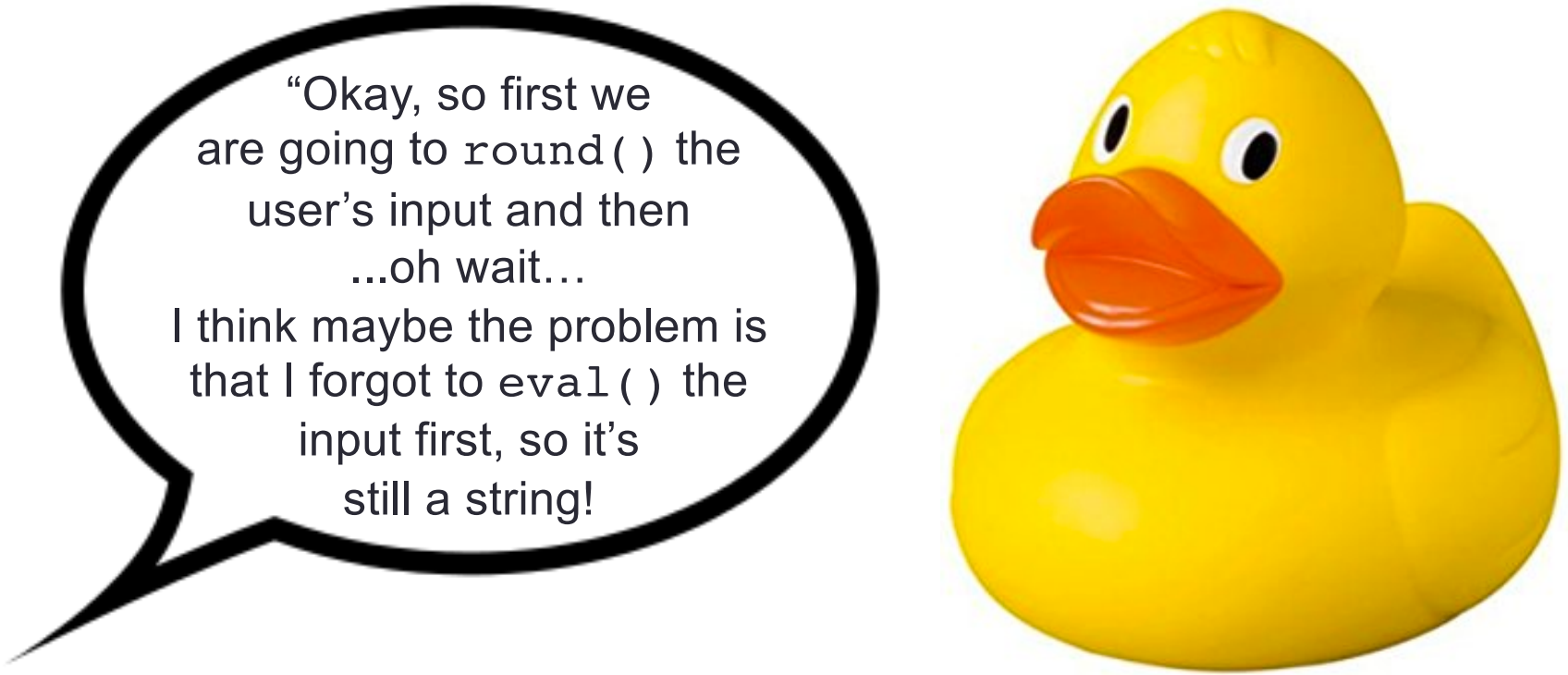
Step 1: map out the code

- It is impossible to debug code that you **don't understand**
  (and it's possible to not understand code even if **you wrote it**!)
- It's often helpful to map out how the code fits together:



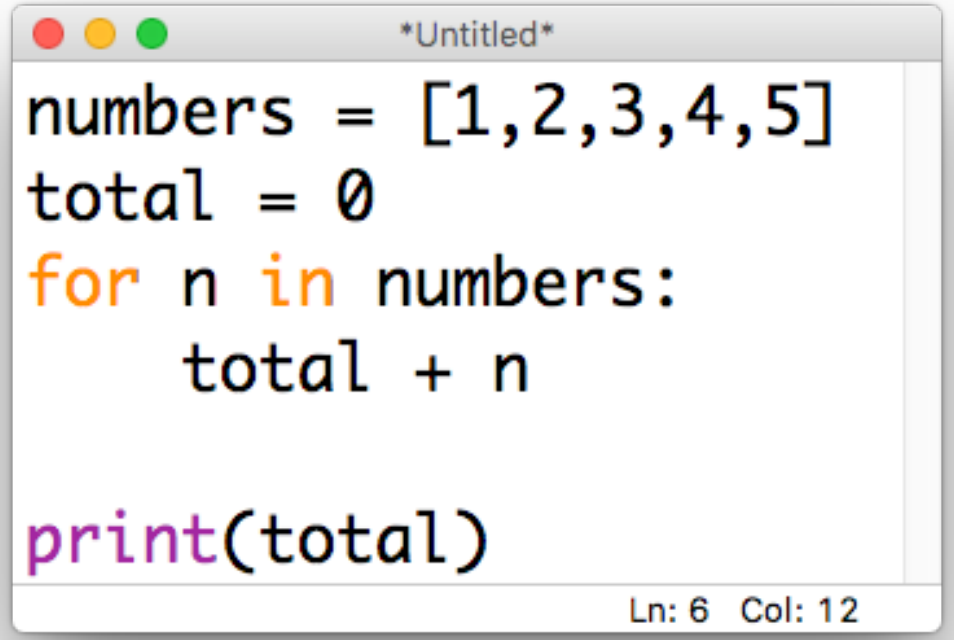
Step 2: “rubber ducking”

- Still stuck? Try explaining it to someone else
  (or historically, to a rubber duckie)
- This is the debugging equivalent of **pair programming**

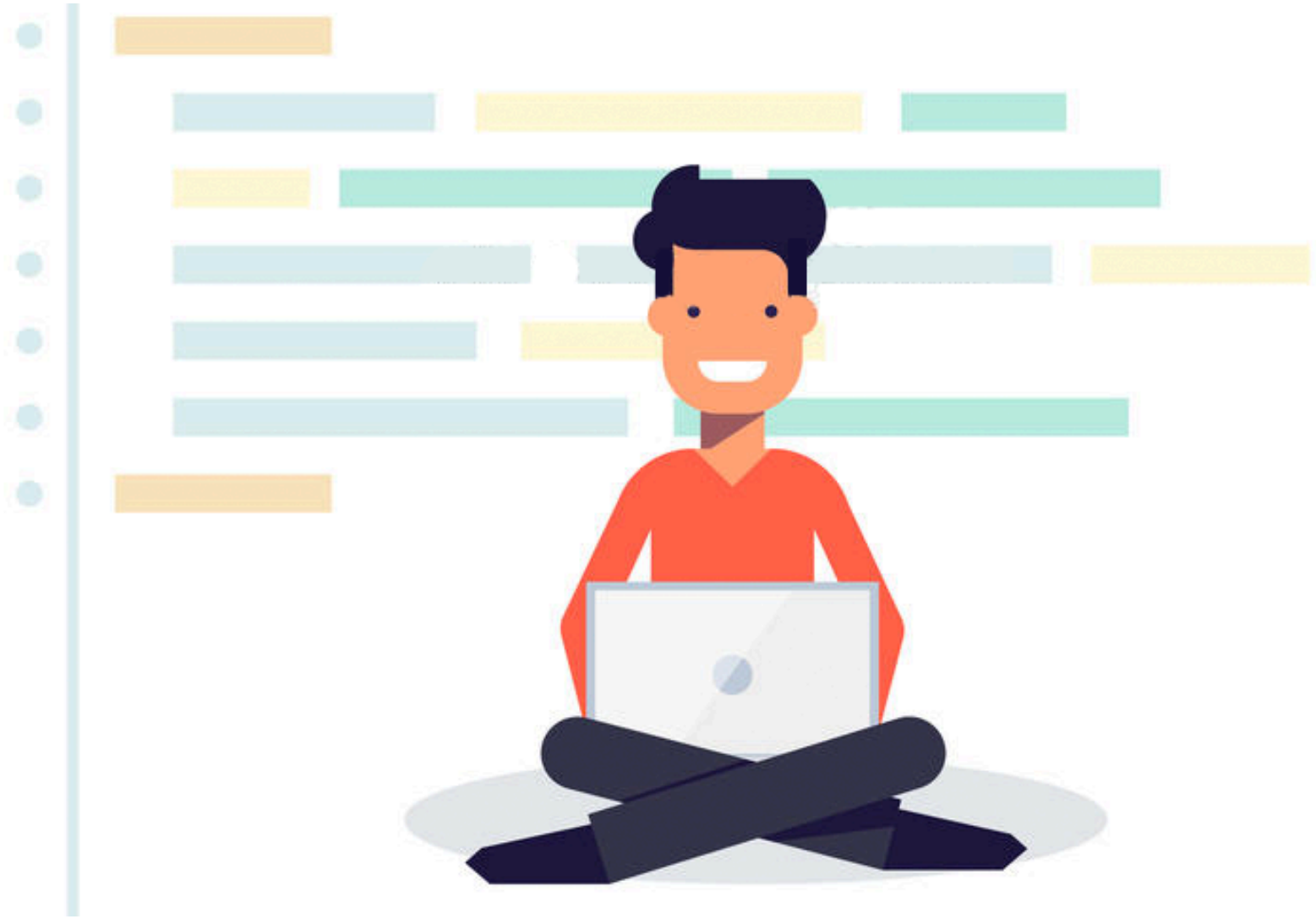


Step 3: add `print()` statements

- Not sure exactly where things are going wrong (esp. inside a loop)?
- Add `print()` statements to leave a “trail” on the console

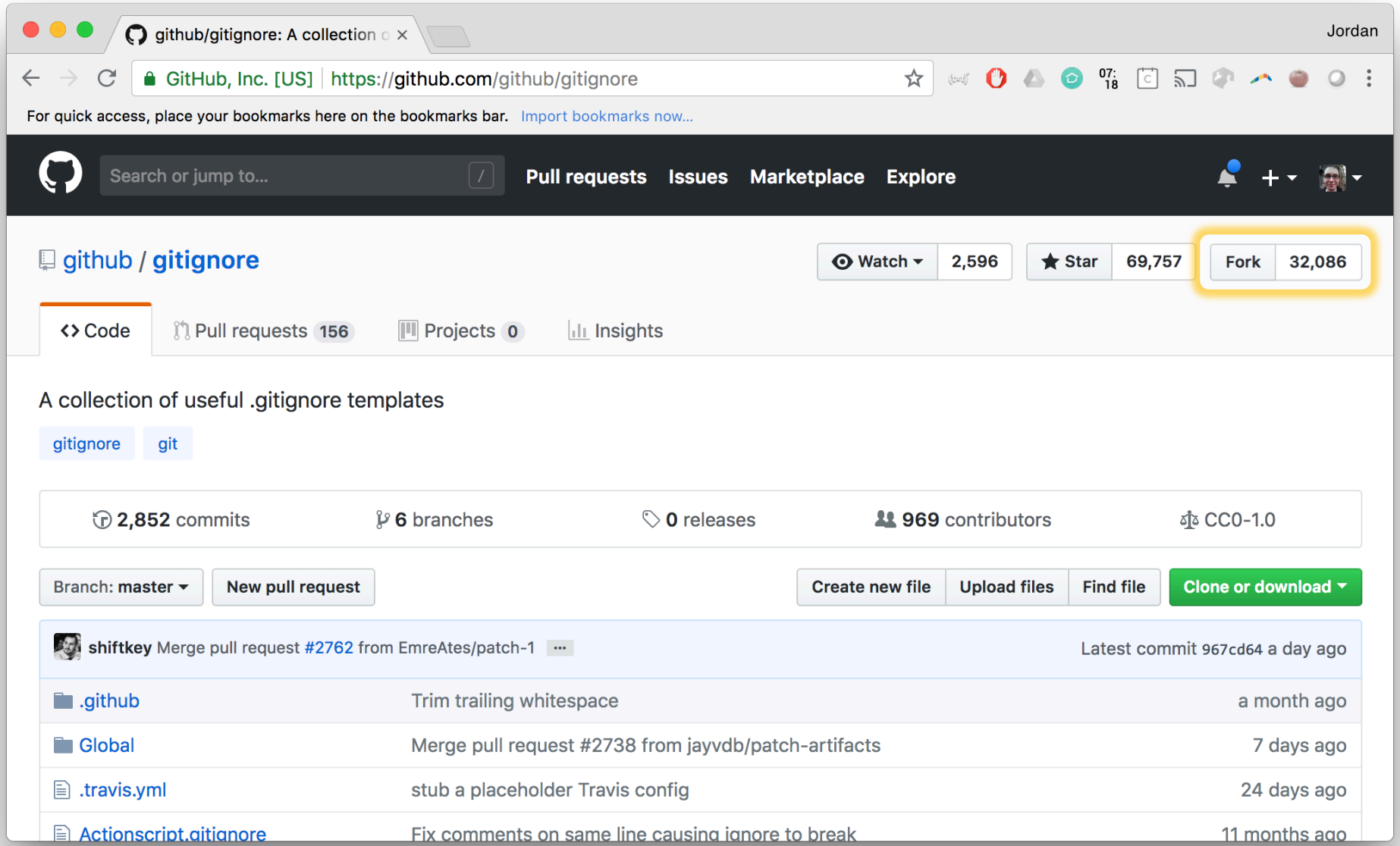


In the beginning…



Now…

Eventually…


github/gitignore: A collection
Jordan
GitHub, Inc. [US] https://github.com/github/gitignore
For quick access, place your bookmarks here on the bookmarks bar. Import bookmarks now...
Search or jump to...
Pull requests
Issues
Marketplace
Explore
github / gitignore
Watch 2,596
Star 69,757
Fork 32,086
Code
Pull requests 156
Projects 0
Insights
A collection of useful .gitignore templates
gitignore
git
2,852 commits
6 branches
0 releases
969 contributors
CC0-1.0
Branch: master
New pull request
Create new file
Upload files
Find file
Clone or download
shiftkey Merge pull request #2762 from EmreAtes/patch-1
Latest commit 967cd64 a day ago
.github
Trim trailing whitespace
a month ago
Global
Merge pull request #2738 from jayvdb/patch-artifacts
7 days ago
.travis.yml
stub a placeholder Travis config
24 days ago
Actionscript.gitignore
Fix comments on same line causing ignore to break
11 months ago

The point

- **Other people** need to be able to understand your code
- **Future you** needs to be able to understand your code
- **Document it**

… but how?

Step 4: meaningful **nouns** for variables

Untitled

```
x = "Jordan Crouser"
y = "Professor"
z = "Smith College"
print(x, "-", y, "at", z)
```

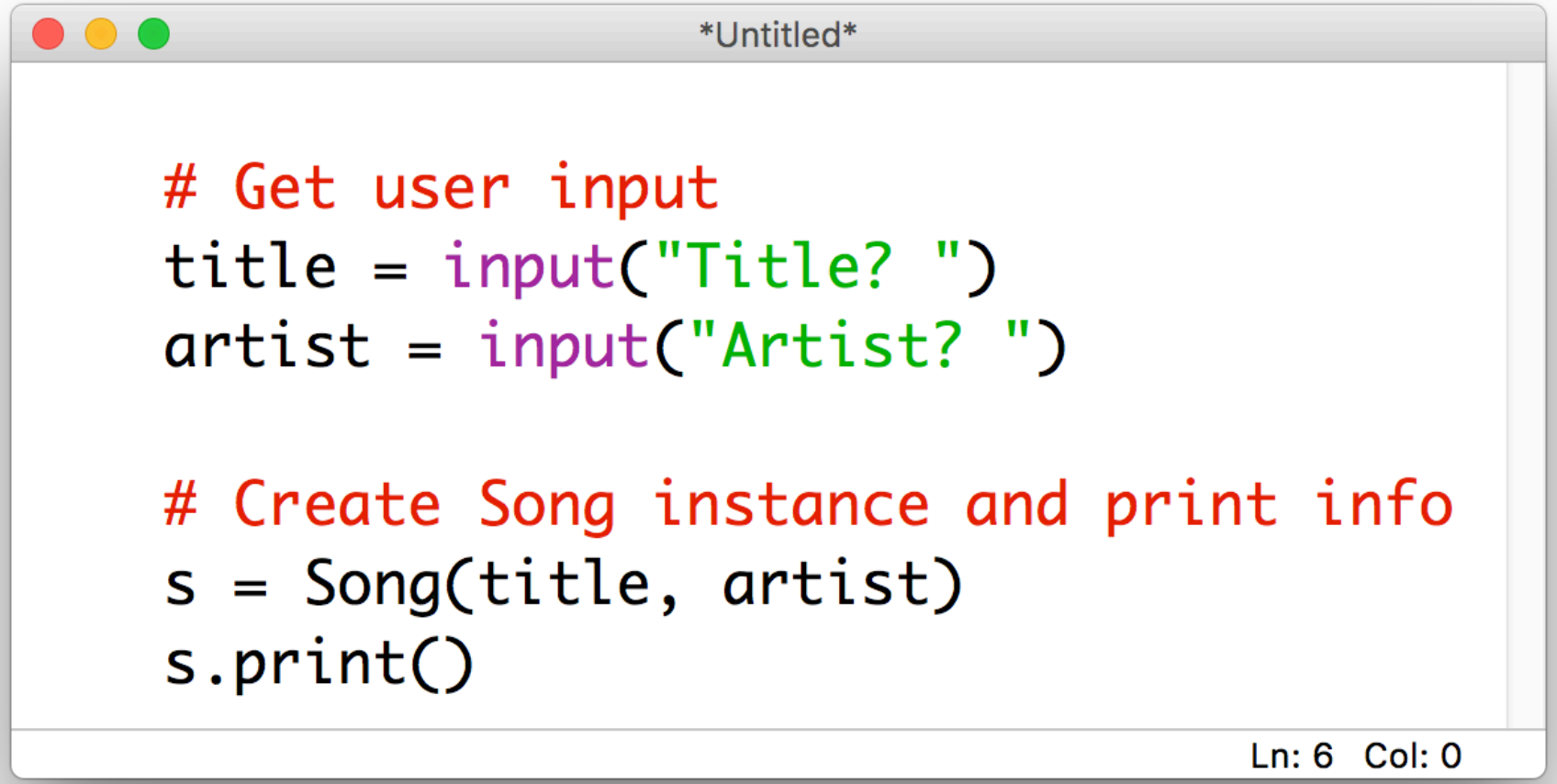
Ln: 1 Col: 1

Untitled

```
name = "Jordan Crouser"
title = "Professor"
institution = "Smith College"
print(name, "-", title, "at", institution)
```

Ln: 4 Col: 41

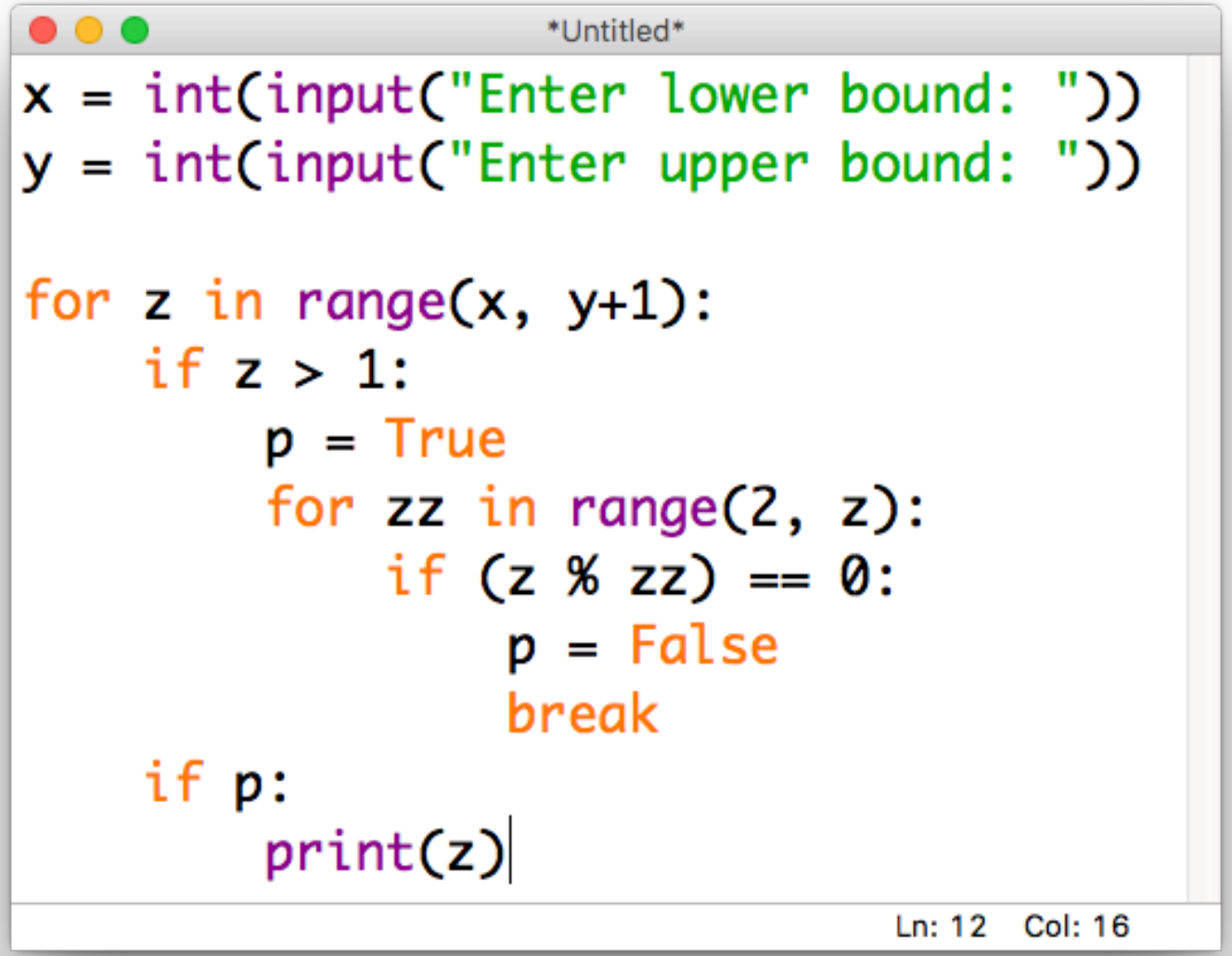
Step 5: lots of comments



A useful technique: code tracing

- **Given**: a very short, poorly-documented program
- **Your goal**: try to figure out what it's doing
- **Recommendations**:
  - walk through the program step-by-step ("**trace its execution**") using the **whiteboard or paper** instead of typing it into a repl
  - once you understand what's happening, then rewrite it using **informative variable names** and **comments**

Example


Untitled
x = int(input("Enter lower bound: "))
y = int(input("Enter upper bound: "))
for z in range(x, y+1):
if z > 1:
p = True
for zz in range(2, z):
if (z % zz) == 0:
p = False
break
if p:
print(z)
Ln: 12 Col: 16

Takeaways

- This is a crash course in **basic** debugging / documentation
- There are **lots** of other techniques for both dealing with and **preventing** bugs, but for now this will suffice
- The most important part is to understand:
  - what the code is **trying** to do
  - what the code is **actually** doing
- Tips:
  - change **one thing** at a time
  - **keep track** of what you change!

Demo

Your task

```
1  # ----------------------------------------------------
2  #       Names: <YOUR NAMES HERE>
3  #    Filename: connectFour-broken.py
4  #        Date: <TODAY'S DATE HERE>
5  #
6  # Description: This file contains a broken version
7  #              of Jordan's ConnectFour game.
8  #
9  #              There are 5 SYNTACTIC ERRORS (mistakes
10 #              that are not correct Python statements
11 #              and so cause the program to throw
12 #              Exceptions) as well as 5 LOGICAL ERRORS
13 #              (mistakes that are technically correct
14 #              Python statements, but which cause the
15 #              program not to behave the way we want).
16 #
17 #              Your job is to find (and correct!) each
18 #              of these mistakes using your new
19 #              DEBUGGING TECHNIQUES.
20 # ----------------------------------------------------
```

Discussion

What did you find?

Up next

- ✓Monday: Loops
  - ✓`for…in` (looping through items in a list)
  - ✓the `range()` function (getting a list of numbers)
  - ✓`while` (looping until something happens)
- ✓Lab: Loops
- ✓Wednesday: Life Skill #2/3: Debugging & Documentation
- A3: Magic 8 Ball due Thursday at 11:55pm EST
- Friday: Dictionaries and Sets