Lecture 10:

LIFE SKILL: DEBUGGING

CSC111: Introduction to CS through Programming

R. Jordan Crouser

Assistant Professor of Computer Science

Smith College

Announcements

Overview

- ✓Strings
  - ✓operations on strings
  - ✓accessing individual letters
  - ✓handy methods
- ✓The `main()` function
- ✓(Bonus) Lab: Pretty Printing
- Life skill #2: debugging

Assignment 3 (what's hard so far?)

In this assignment, you will write a python program that manipulates a user-entered string.

The user interface of your calculator should first ask the user to input **a sentence**. For example:

```
Enter a sentence: I love computer science!
```

Your program will then output the following:

```
0. I love computer science!
1a. I LOVE COMPUTER SCIENCE!
1b. i love computer science!
2. I loovee coompuuteer sciieencee! (or 2. II loovee coompuuteer sciieencee!")
3. I lov...ence!
4. I Love Computer Science!
5. !ecneics retupmoc evol I
```

Discussion

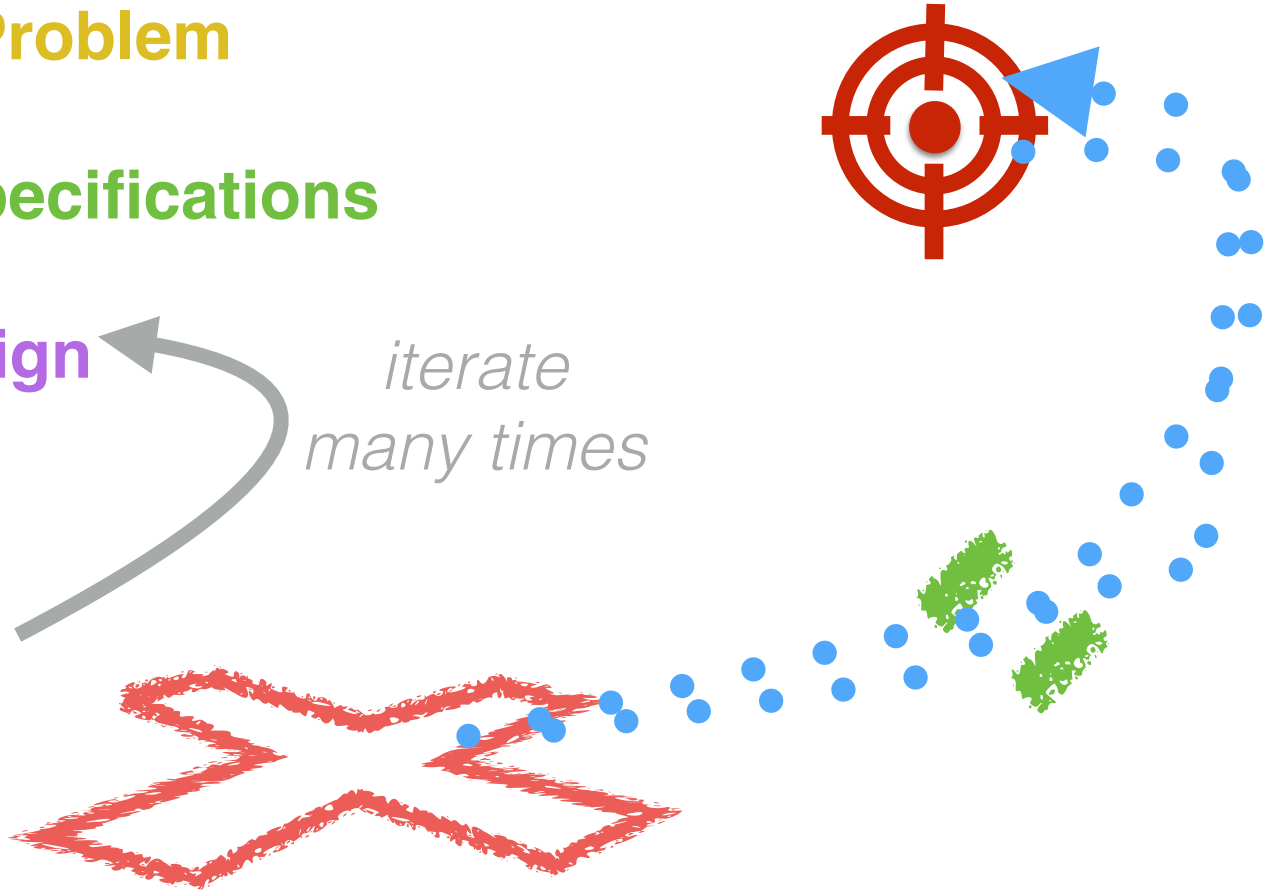
What is your plan

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

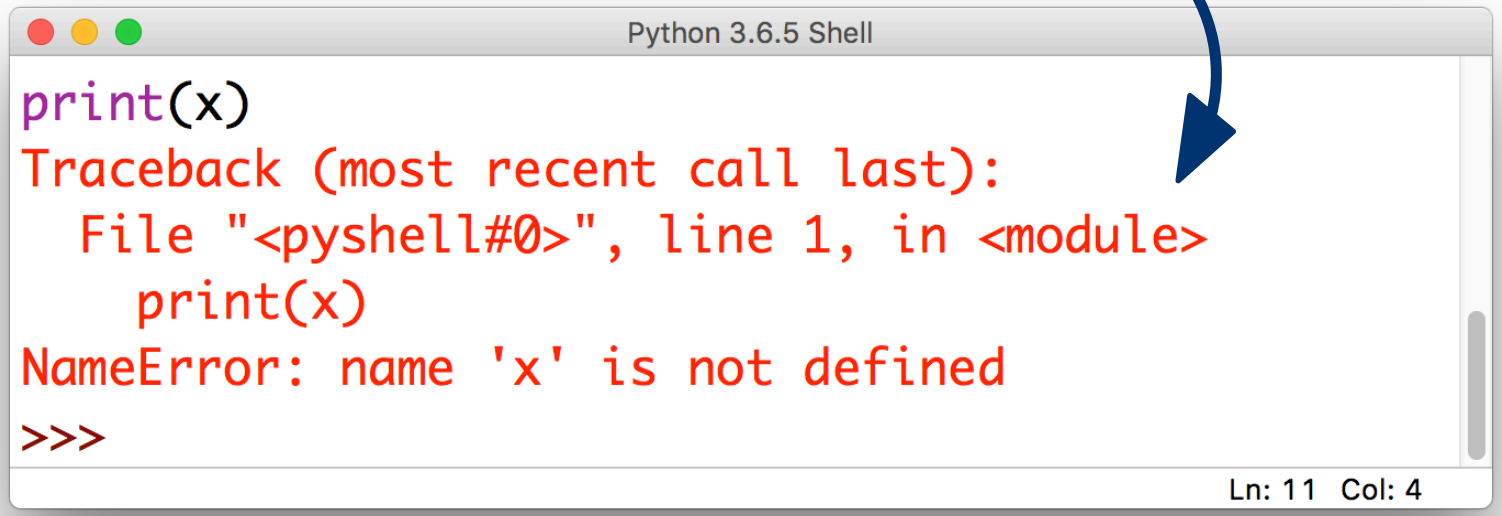
0800 Arctan started
1000 " stopped - arctan ✓ { 1.2700 9.037 847 025
9.037 846 ?95 correct
13"uc (032) MP - MC ~~1.9821447000~~ ~~2.13047764715 (033)~~ 4.615925059(-2)
(033) PRO 2 2.13047 6415
correct 2.13067 6415
Relays 6-2 in 033 failed special speed test
In relay " 11.00 test.
Relays changed
1100 Started Cosine Tape (Sine check)
1525 Started Mult + Adder Test.
1545  Relay #70 Panel F
(moth) in relay.
First actual case of bug being found.
1700 closed down.

RDML Grace M. Hopper

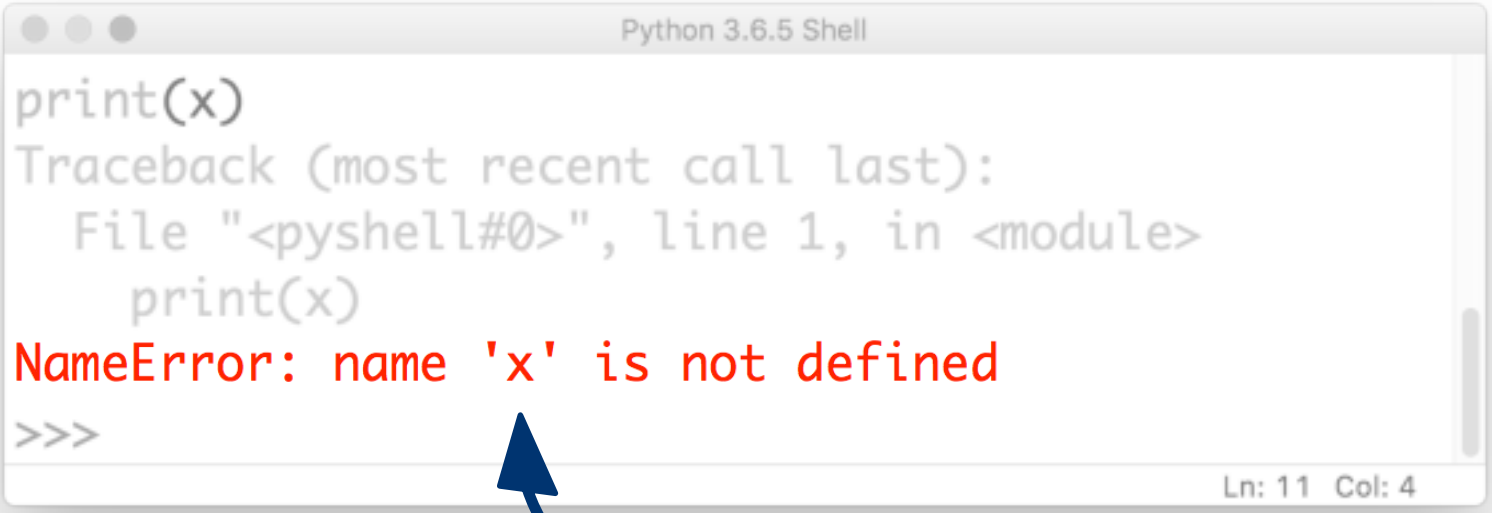
b.1906 – d.1992

Some problems are obvious

this is called
an **`Exception`**



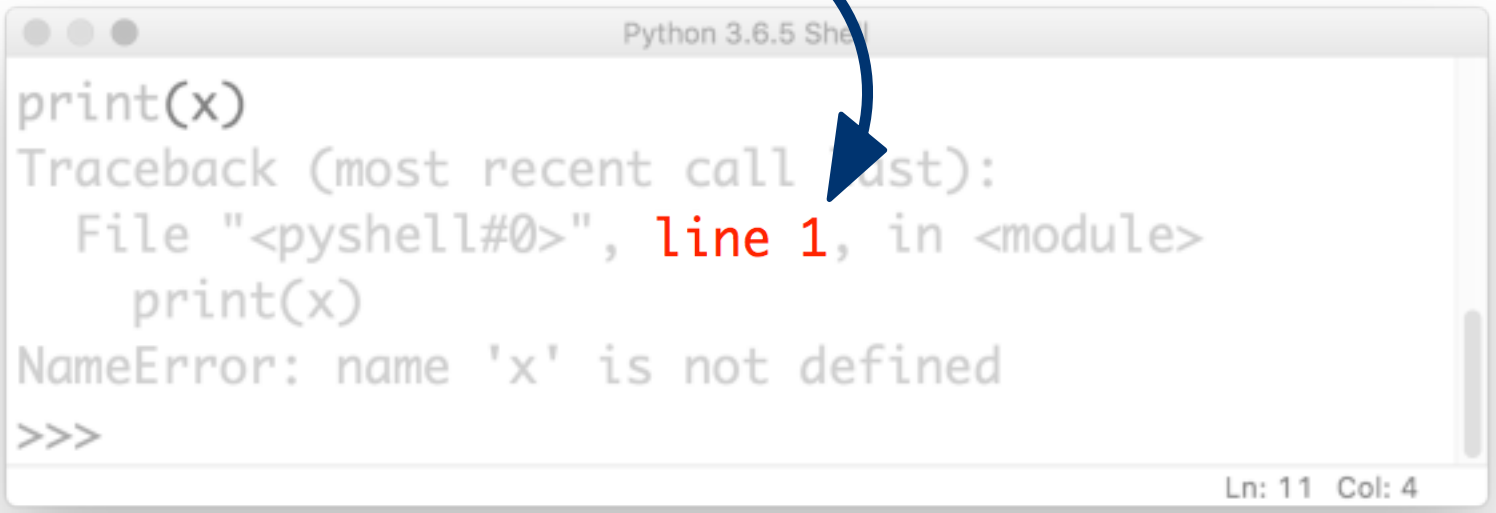
Some problems are obvious



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

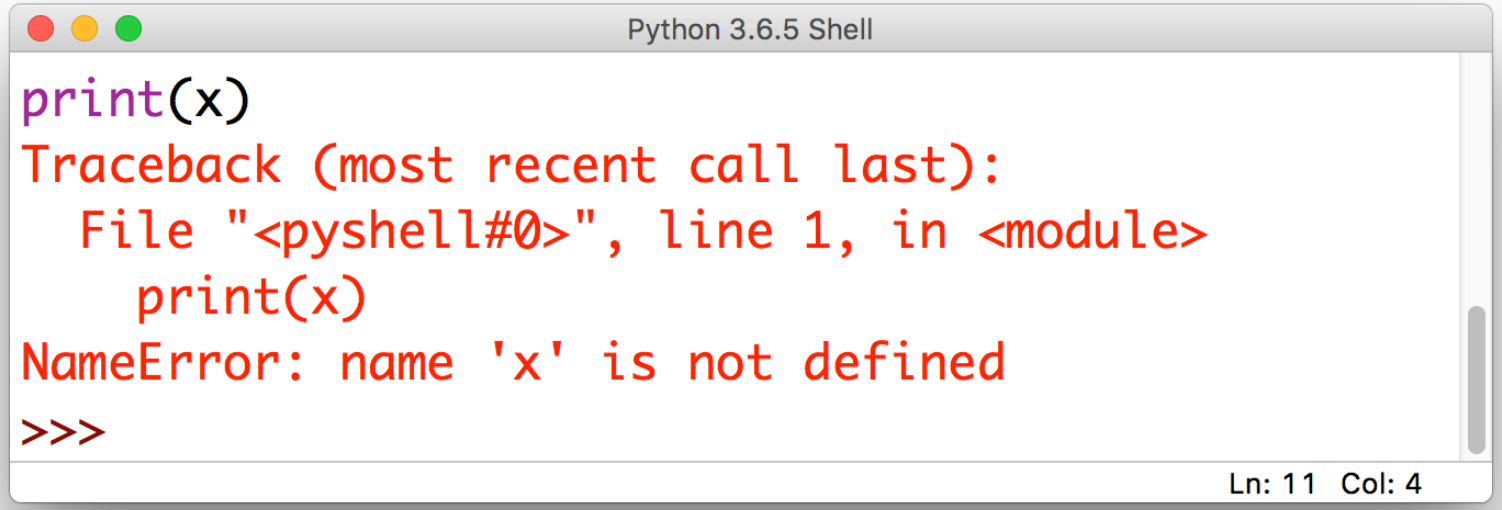
Some problems are obvious

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



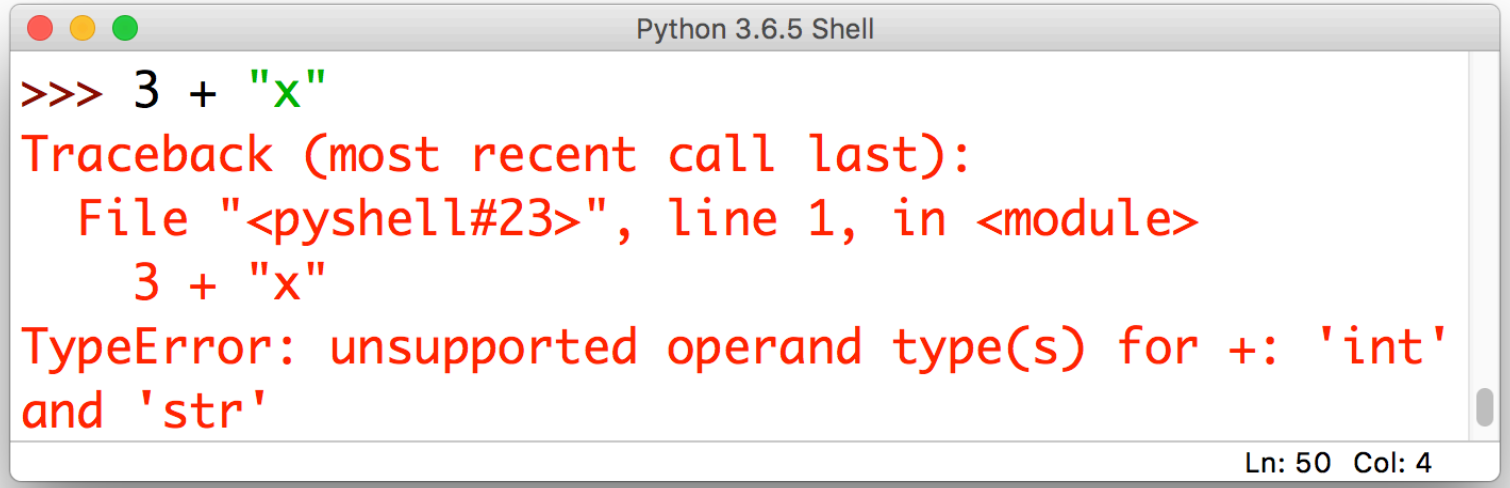
Common **`Exceptions`**

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



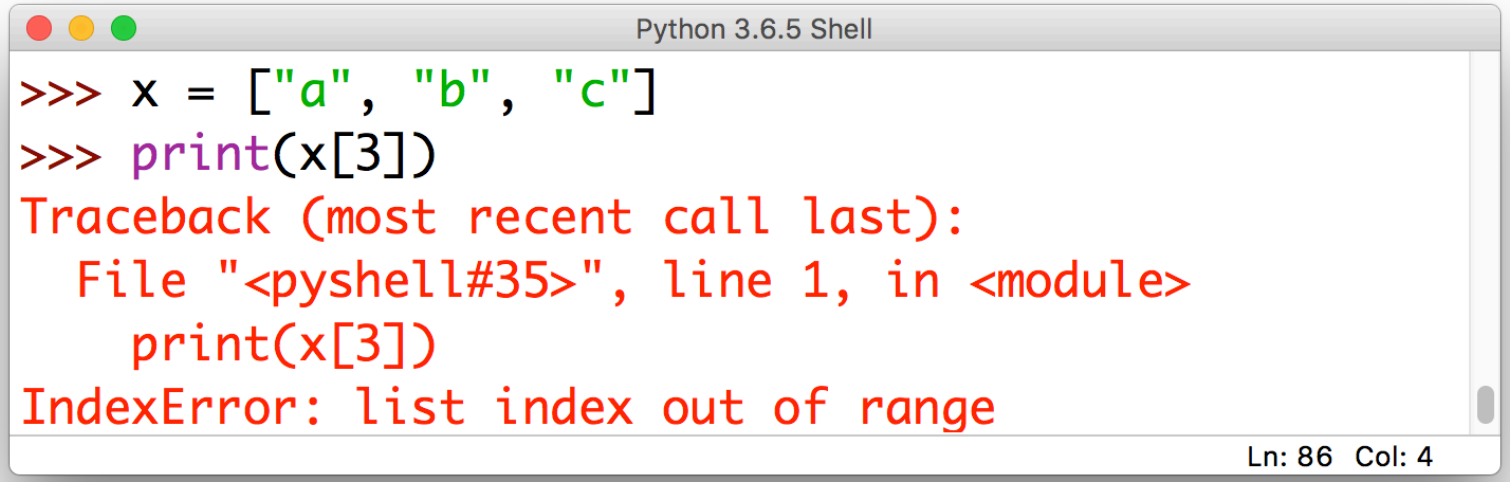
Common **`Exceptions`**

- **`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)



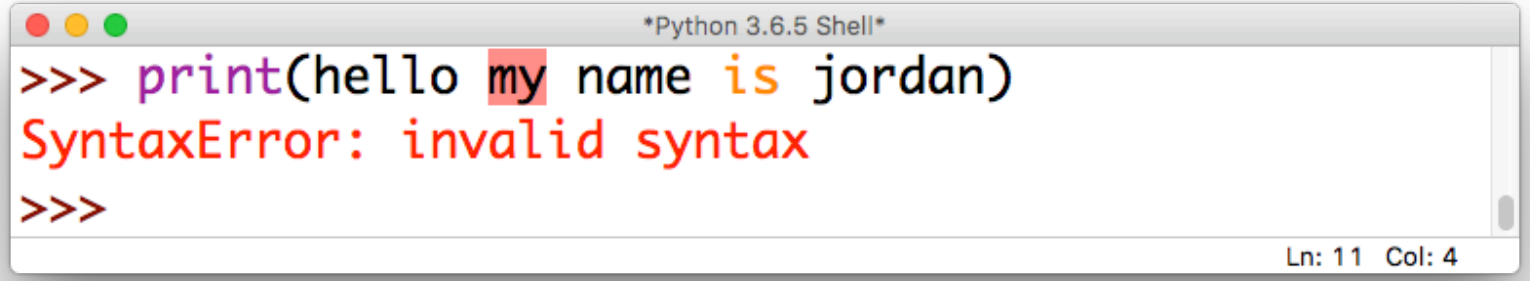
Common **`Exceptions`**

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



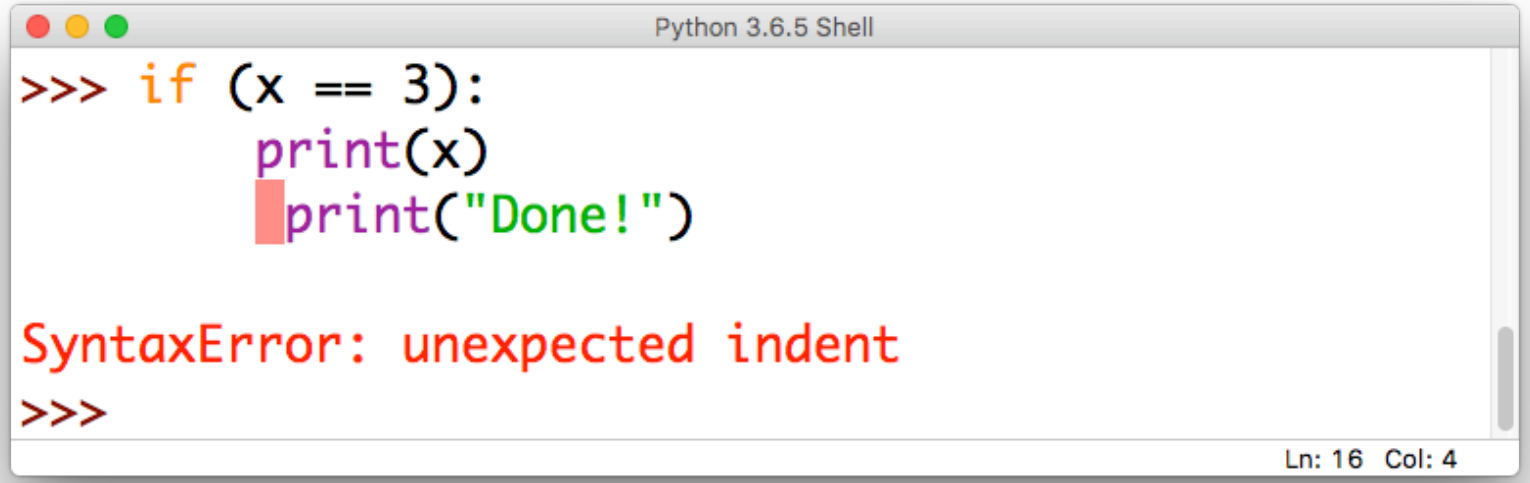
Common **`Exceptions`**

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



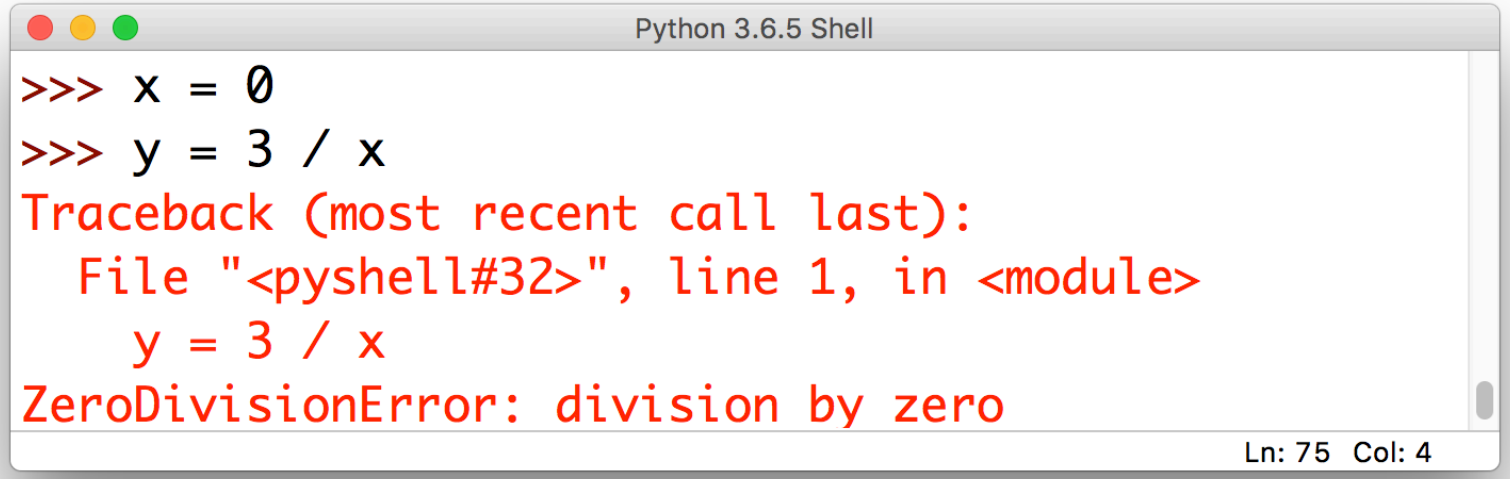
Common **`Exceptions`**

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



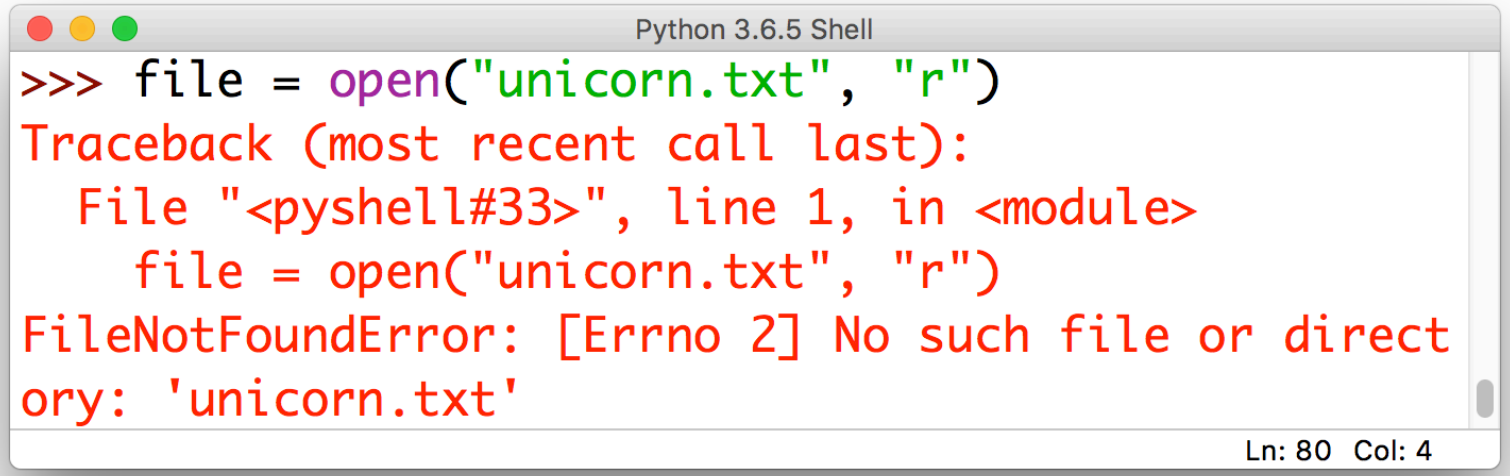
Common **`Exceptions`**

- **`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

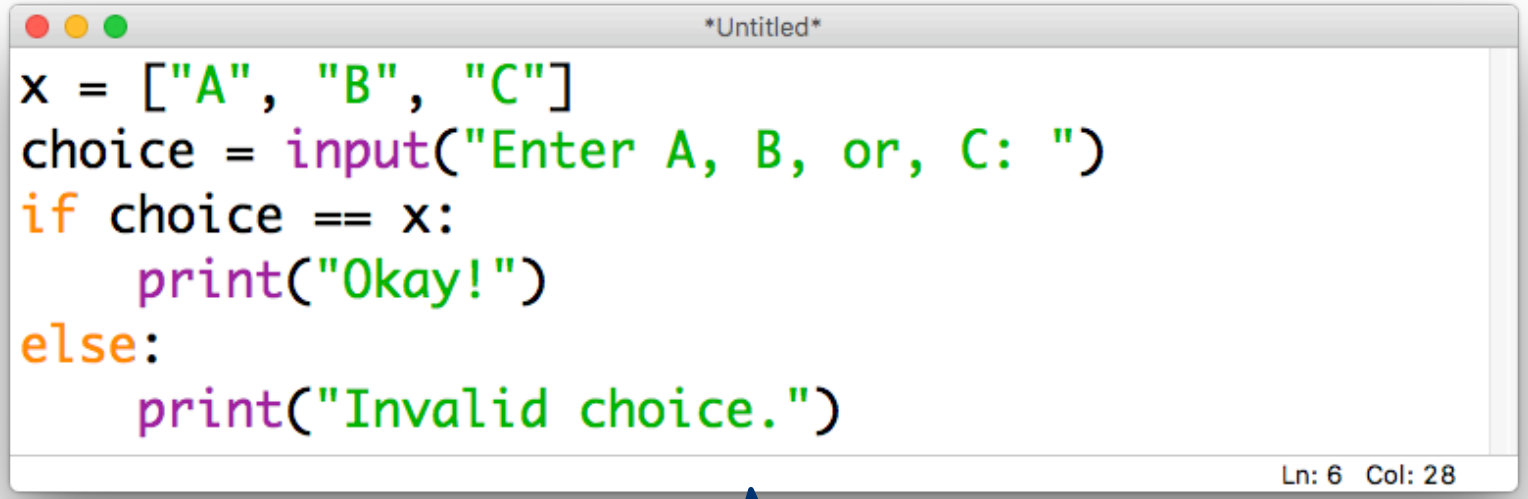
`Exceptions` = relatively easy to fix

Why would I say that?

What's the alternative?

Logical errors

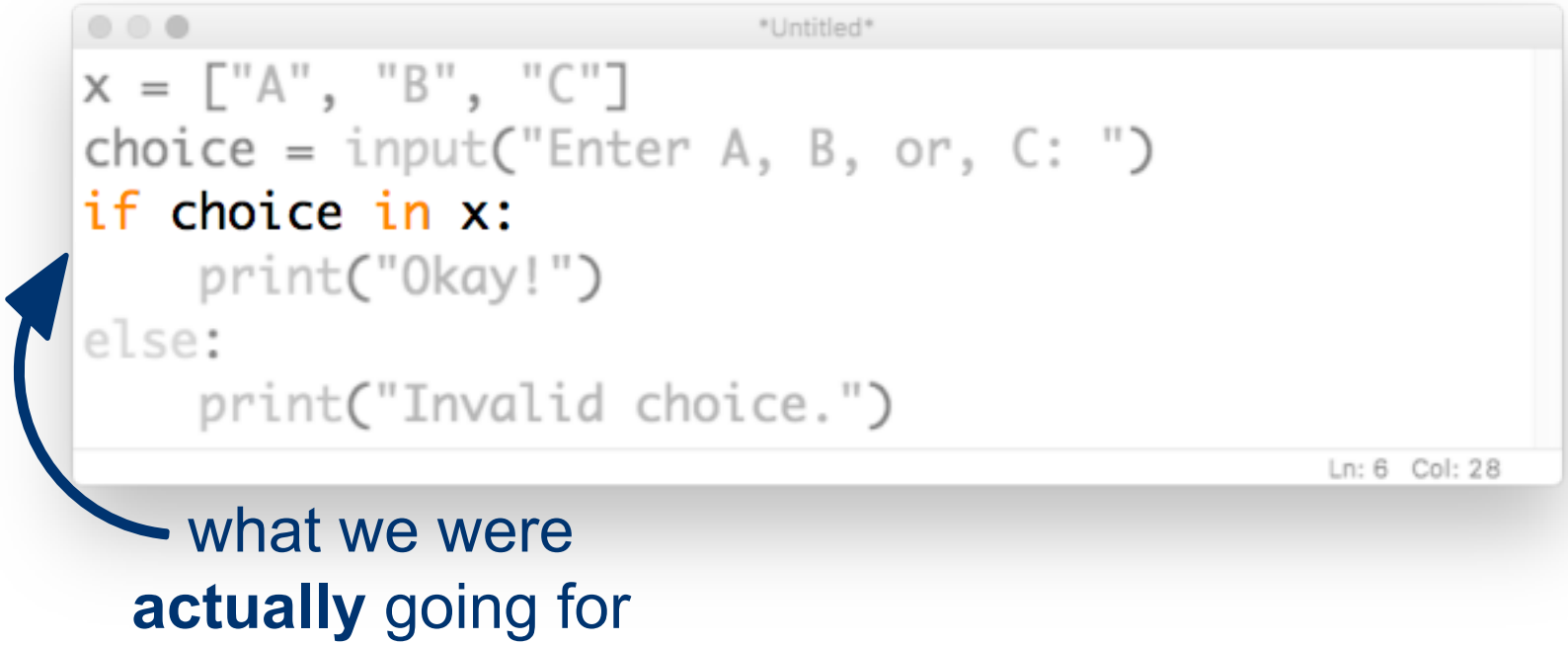
- Mistakes in the **reasoning** behind the code (though the statements are valid and there are no `Exceptions`), e.g.



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

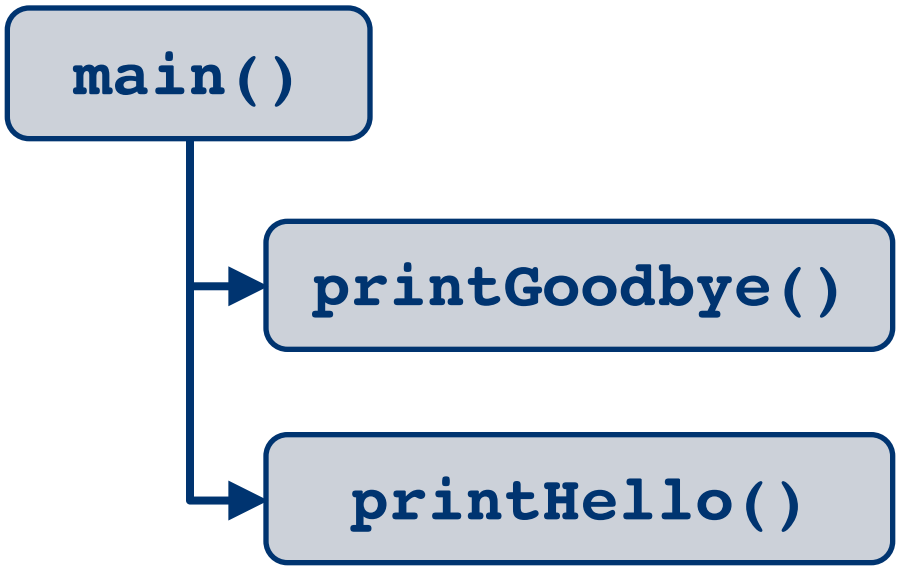
If an animal is green, it must be a frog.

Discussion

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

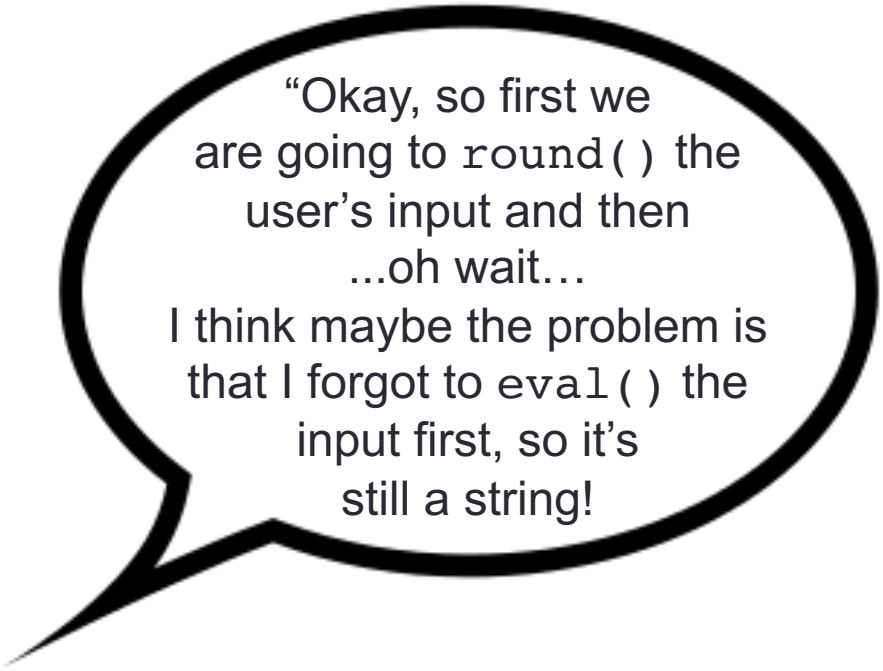
Step 1: map out the code

- 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:



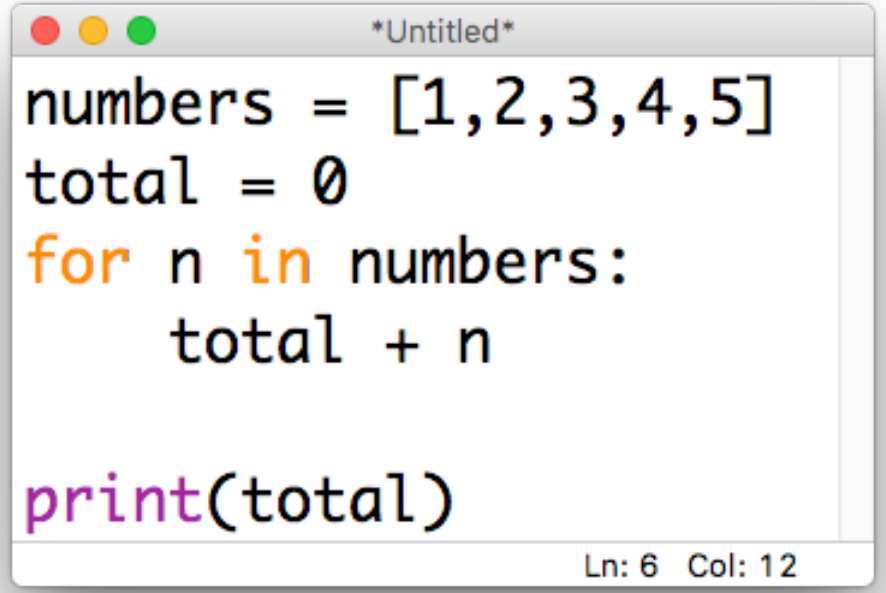
Step 2: “rubber ducking”

- 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



Takeaways

- This is a really quick crash course in **basic** debugging
- 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

connectFour-broken.py - /Users/jcrouser/Google Drive/Teaching/Course Material/CSC111/CSC111/labs-old/connectFour-broken.py (3.6.5)

```
# ------------------------------------------------
#       Names: <YOUR NAMES HERE>
#    Filename: connectFour-broken.py
#        Date: <TODAY'S DATE HERE>
#
# Description: This file contains a broken version
#              of Jordan's ConnectFour game.
#
#              There are 5 SYNTACTIC ERRORS (mistakes
#              that are not correct Python statements
#              and so cause the program to throw
#              Exceptions) as well as 5 LOGICAL ERRORS
#              (mistakes that are technically correct
#              Python statements, but which cause the
#              program not to behave the way we want).
#
#              Your job is to find (and correct!) each
#              of these mistakes using your new
#              DEBUGGING TECHNIQUES.
# ------------------------------------------------
```

Ln: 15 Col: 54

Discussion

What did you find?

Coming up next

- *A3: Copycat* is due **Sunday 11:55pm**
- **Mon 10/1: `for` loops and `while` loops**
- **Weds 10/3**: **`range()`** and other useful functions
- **Lab**: Old MacDonald
- **Fri 10/5**: Life skill #2: Documentation