

Lecture 10:

# LIFE SKILL: DEBUGGING

---

CSC111: Introduction to CS through Programming

R. Jordan Crouser

Assistant Professor of Computer Science

Smith College

# Announcements



posted to #questions  
cross-posted to #general

 **Annabel Yim** Yesterday at 9:59 AM  
Hi, Jordan. For those that have mountain day and don't have lab today, what do we do with our lab assignment?

4 replies

 **jordan** 22 hours ago  
This lab will be converted to an extra credit point for anyone who completes it ~~in the usual timeframe~~ before Friday at 5pm.  
(edited)

 1

**Happy Mountain Day!**

# 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 coompuuter sciieence! (or 2. II loovee coompuuter sciieence!" )
3. I lov...ence!
4. I Love Computer Science!
5. !ecneics retupmoc evol I



# Discussion

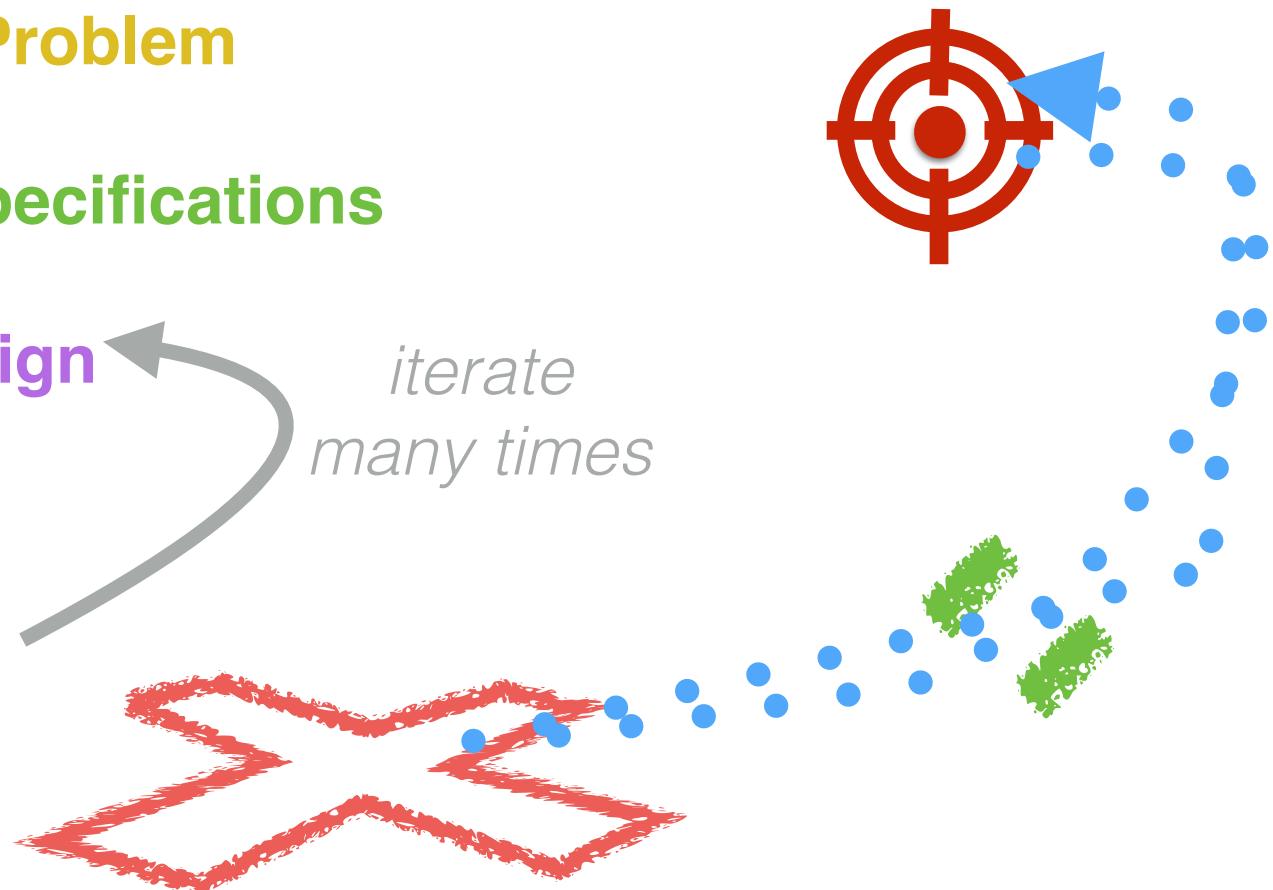
What is your plan  
for getting **unstuck**?



# 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 Antran started  
1000 " stopped - antran ✓ { 1.2700 9.037 847 025  
1300 (032) MP - MC 1.982 1.47000 9.037 846 995 connect  
                      (033) PRO 2 2.130 476415 4.615 925059 (-2)  
                      connect 2.130 676415  
Relays 6-2 in 033 failed special speed test  
in relay " 10.00 test.

1100 Started Cosine Tape (Sine check)  
1525 Started Multi+ Adder Test.  
Relays changed

1545 Relay #70 Panel F  
(moth) in relay.

1630 Antran started.  
1700 closed down.



RDML Grace M. Hopper  
b.1906 – d.1992

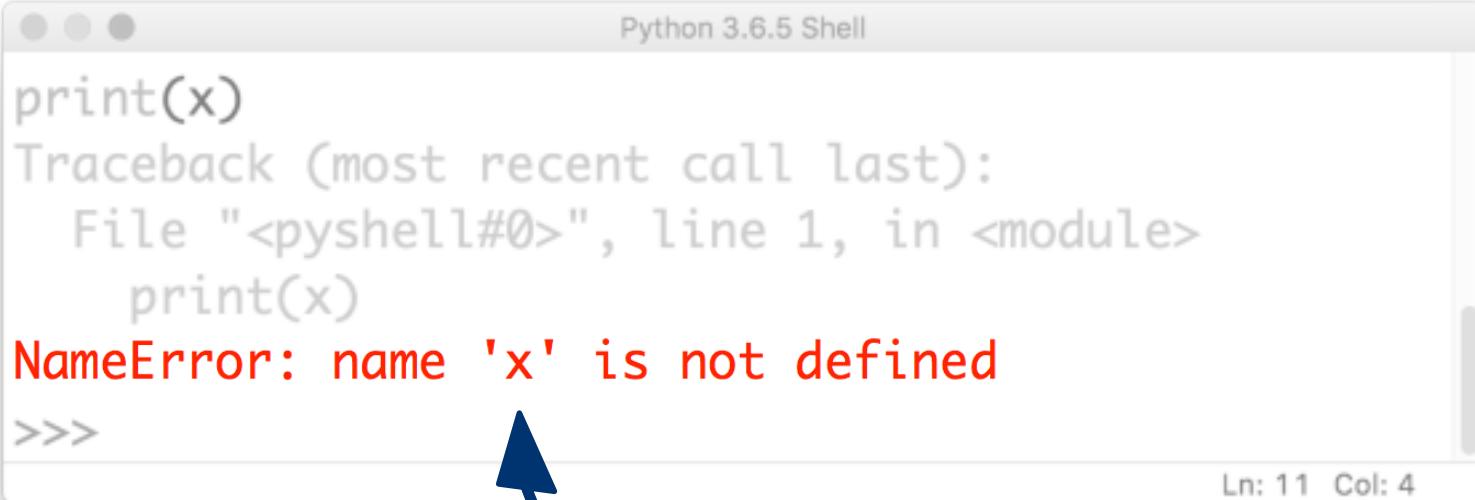
# Some problems are obvious

this is called  
an **Exception**



```
Python 3.6.5 Shell
print(x)
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    print(x)
NameError: name 'x' is not defined
>>>
Ln: 11 Col: 4
```

# Some problems are obvious



A screenshot of a Python 3.6.5 Shell window. The title bar says "Python 3.6.5 Shell". The code input area shows "print(x)". A red error message "NameError: name 'x' is not defined" is displayed below it. The cursor is at the start of the line "print(x)". The status bar at the bottom right shows "Ln: 11 Col: 4".

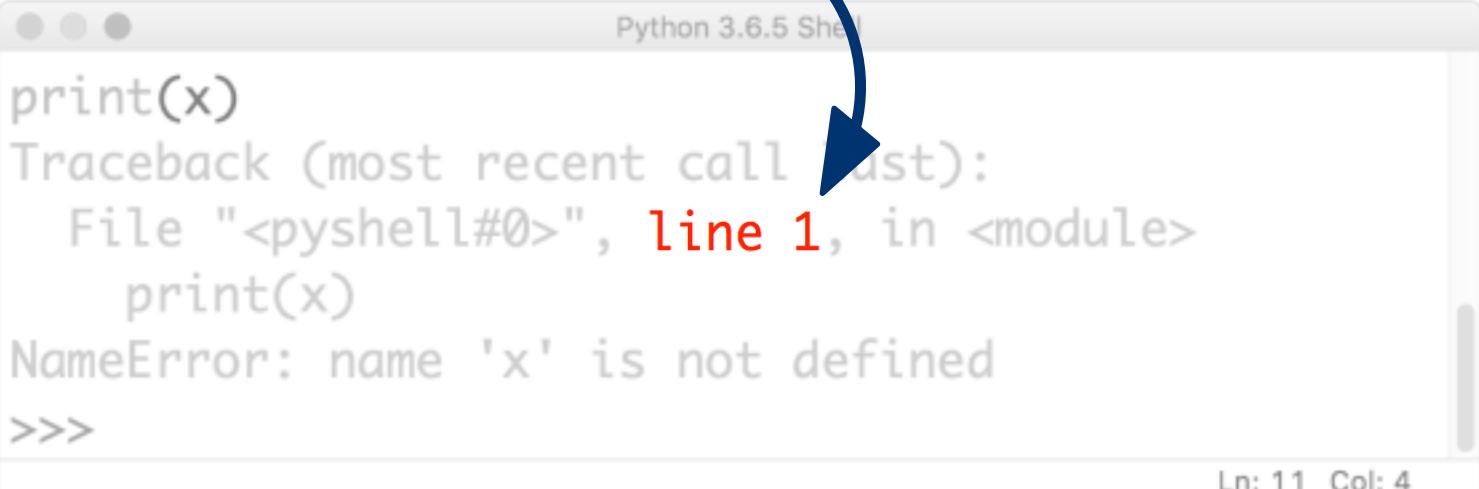
```
print(x)
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    print(x)
NameError: name 'x' is not defined
>>>
```



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!)

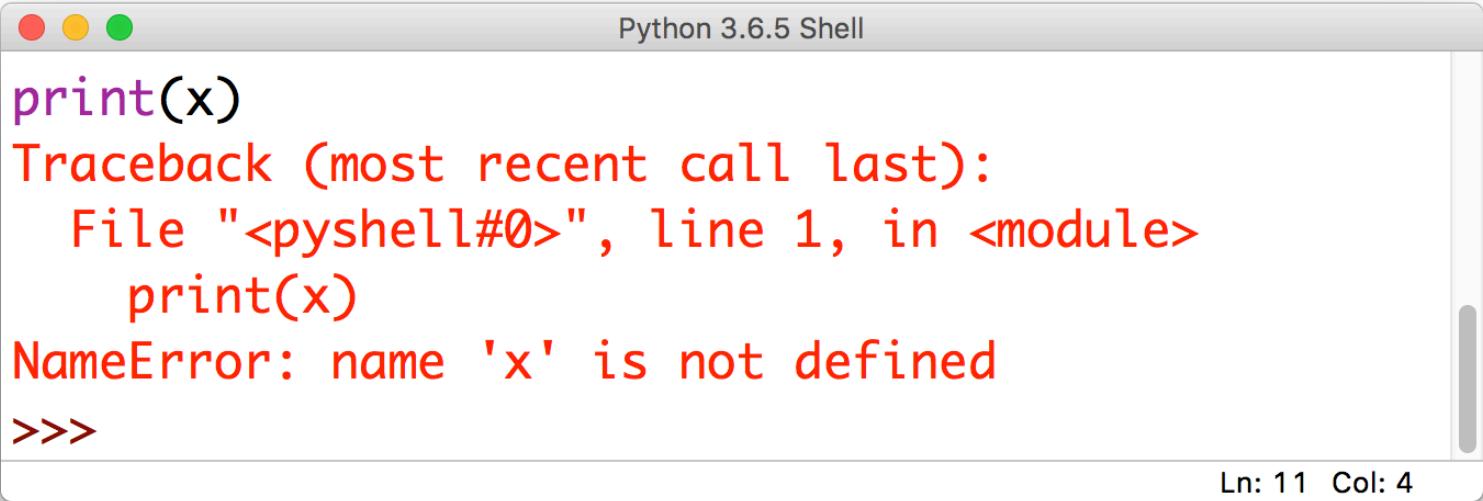


```
Python 3.6.5 Shell
print(x)
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    print(x)
NameError: name 'x' is not defined
>>>
```

Ln: 11 Col: 4

# Common Exceptions

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



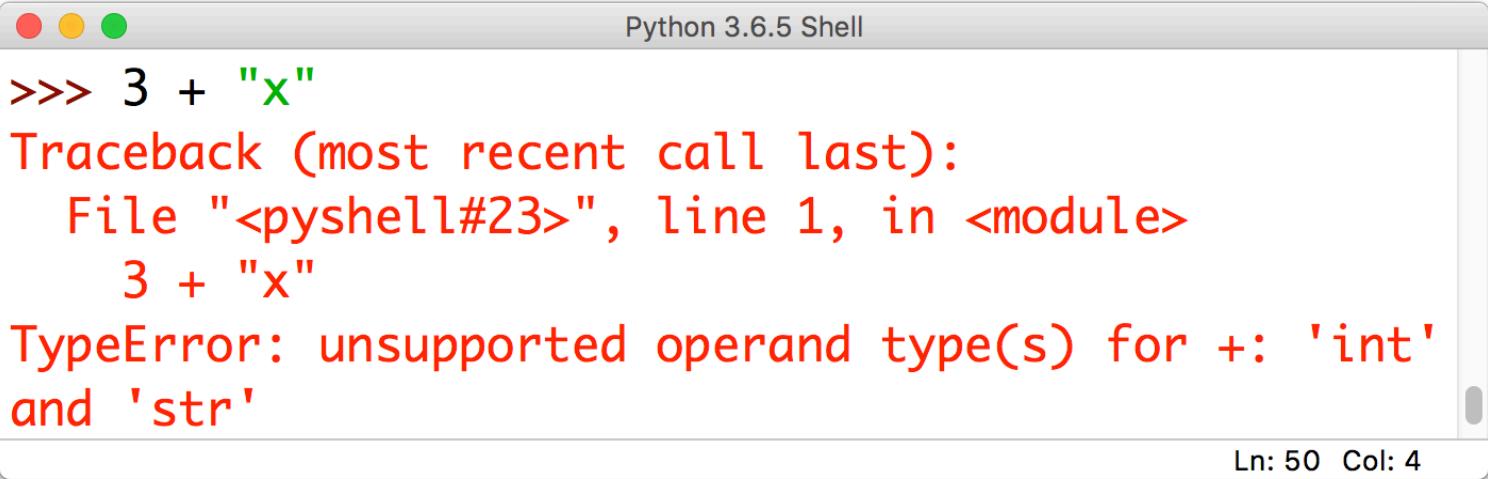
The screenshot shows a Python 3.6.5 Shell window. The title bar reads "Python 3.6.5 Shell". The main area contains the following text:

```
print(x)
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    print(x)
NameError: name 'x' is not defined
>>>
```

In the bottom right corner of the shell window, there is a status bar with the text "Ln: 11 Col: 4".

# Common Exceptions

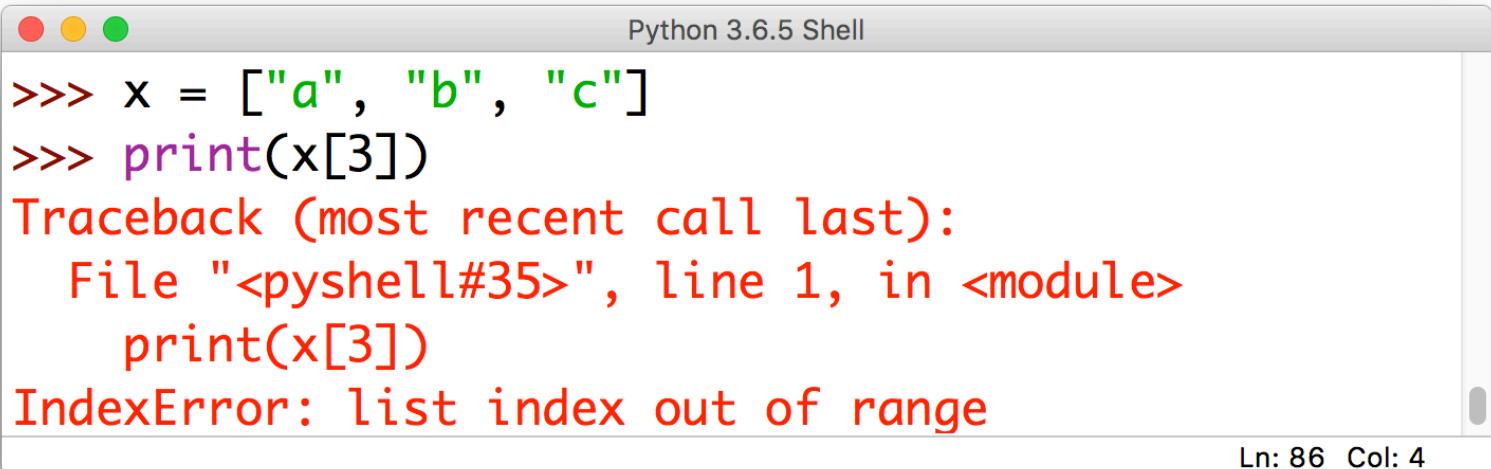
- **TypeError**: raised when you try to perform an operation on an object that's not the right type (i.e. a **string** instead of a **number**)



The screenshot shows a Python 3.6.5 Shell window. The user has typed `>>> 3 + "x"`. The shell then displays a red error message: `Traceback (most recent call last):  
 File "<pyshell#23>", line 1, in <module>  
 3 + "x"  
TypeError: unsupported operand type(s) for +: 'int'  
and 'str'`. In the bottom right corner of the window, it says `Ln: 50 Col: 4`.

# Common Exceptions

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

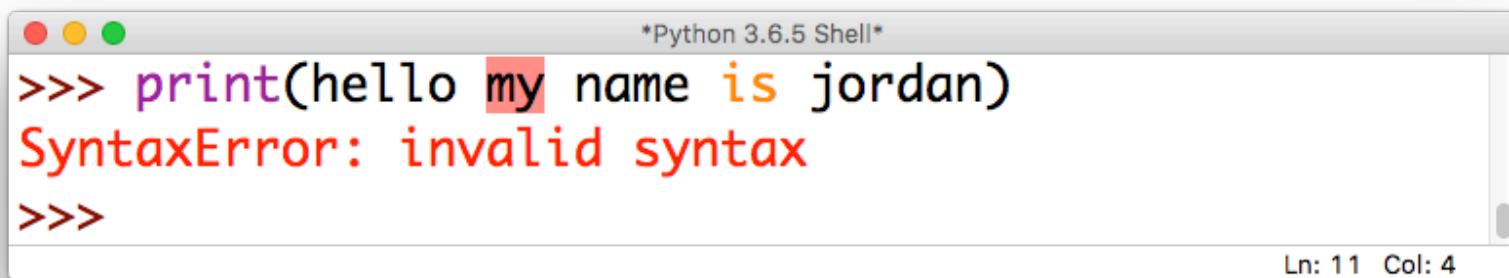


A screenshot of a Mac OS X window titled "Python 3.6.5 Shell". Inside the window, Python code is being run. The code defines a list `x` with three elements: "a", "b", and "c". When the user tries to print the fourth element `x[3]`, a traceback is generated. The traceback shows the file name as "<pyshell#35>" and the line number as "line 1, in <module>". The error message "print(x[3])" is also visible. At the bottom of the window, the status bar displays "Ln: 86 Col: 4".

```
>>> x = ["a", "b", "c"]
>>> print(x[3])
Traceback (most recent call last):
  File "<pyshell#35>", line 1, in <module>
    print(x[3])
IndexError: list index out of range
Ln: 86 Col: 4
```

# Common Exceptions

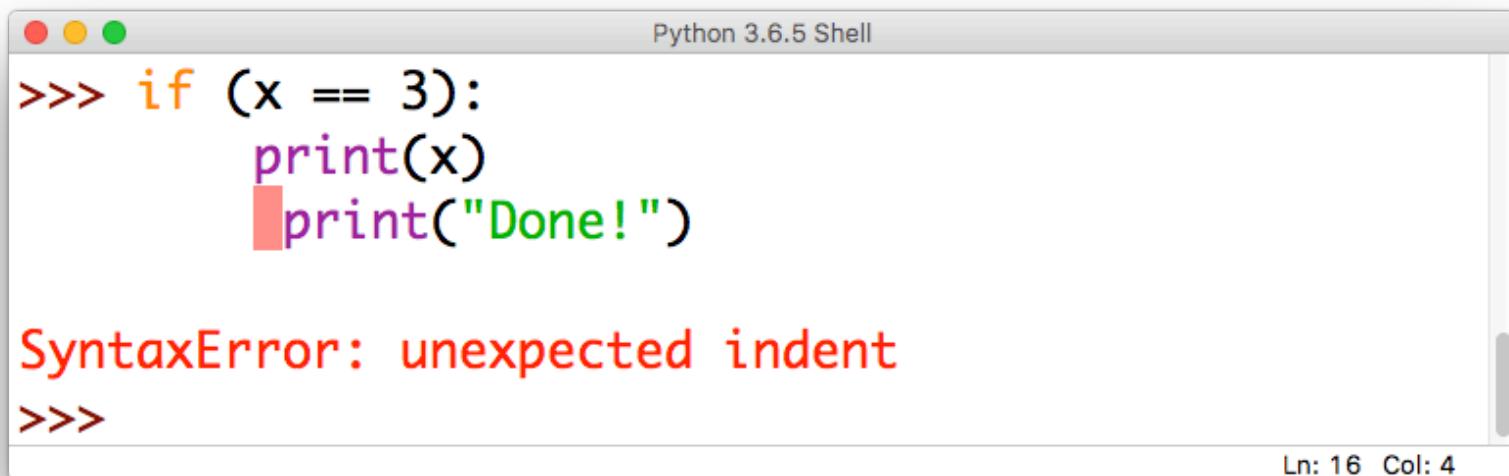
- **SyntaxError:** raised when you try to run a command that isn't a valid Python statement



A screenshot of a Python 3.6.5 Shell window. The title bar says "\*Python 3.6.5 Shell\*". The console area shows the following text:  
>>> print(hello my name is jordan)  
SyntaxError: invalid syntax  
>>>  
In the bottom right corner, it says "Ln: 11 Col: 4".

# Common Exceptions

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



The screenshot shows a Python 3.6.5 Shell window. The code entered is:

```
>>> if (x == 3):
    print(x)
    print("Done!")
```

The output is:

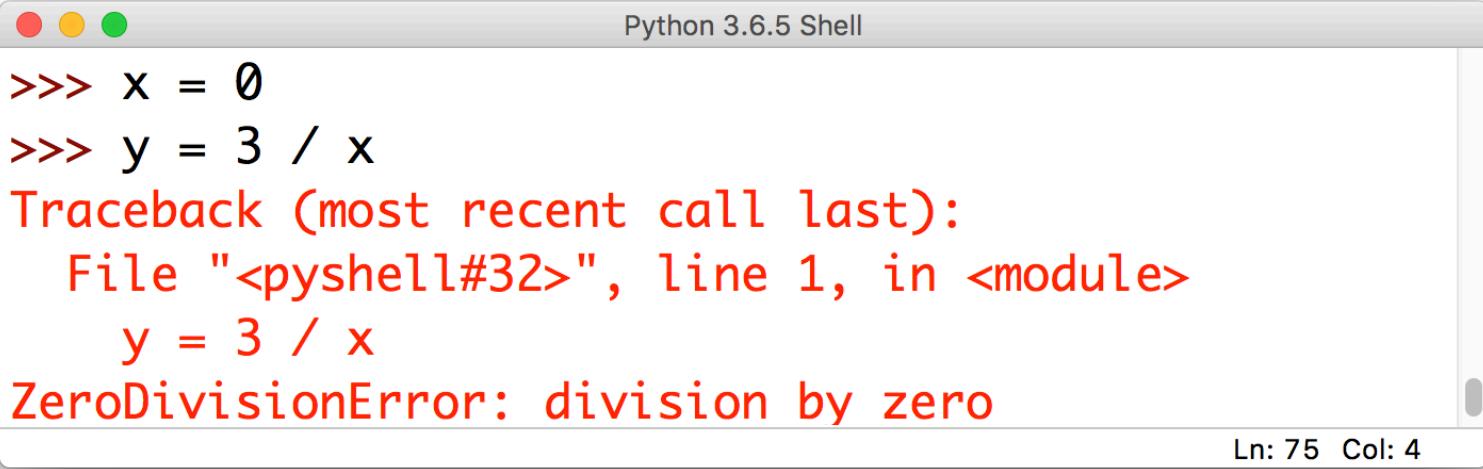
SyntaxError: unexpected indent

The shell prompt >>> is visible at the bottom.

Ln: 16 Col: 4

# Common Exceptions

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



A screenshot of a Mac OS X window titled "Python 3.6.5 Shell". The window contains a command-line interface. The user has entered the following code:

```
>>> x = 0
>>> y = 3 / x
```

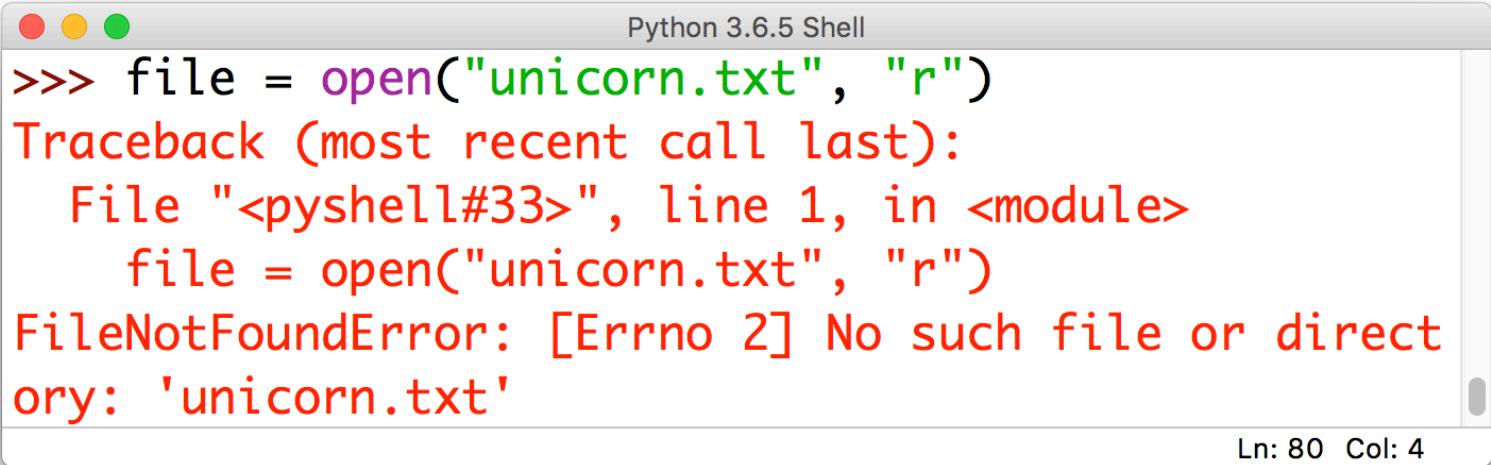
When the user attempts to execute the second line, a traceback is printed in red text:

```
Traceback (most recent call last):
  File "<pyshell#32>", line 1, in <module>
    y = 3 / x
ZeroDivisionError: division by zero
```

In the bottom right corner of the window, there is a status bar with the text "Ln: 75 Col: 4".

# Common Exceptions

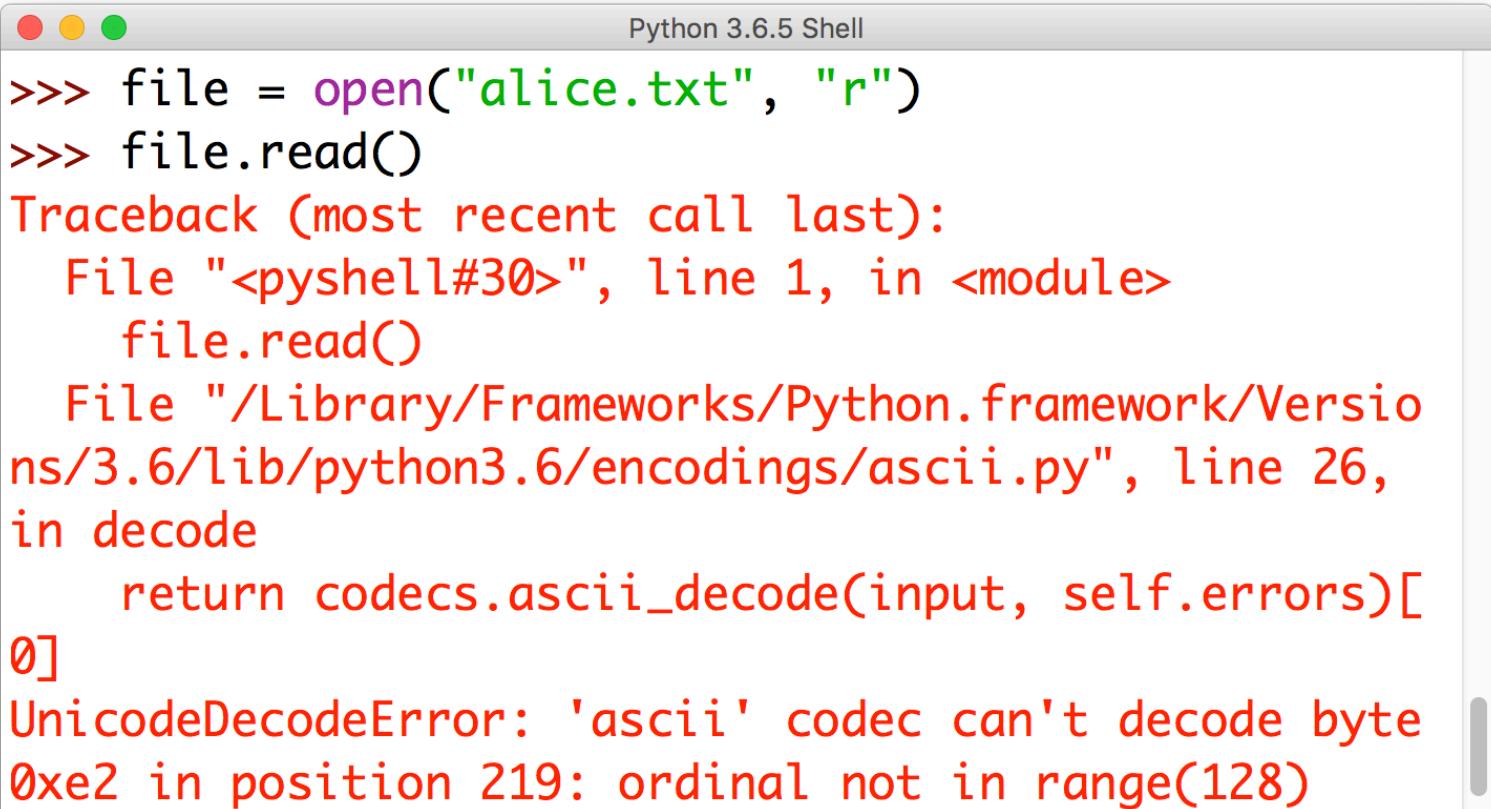
- **FileNotFoundException**: raised when Python can't find the thing you're referring to (a file)



The image shows a screenshot of a Python 3.6.5 Shell window. The title bar reads "Python 3.6.5 Shell". The code input is: `>>> file = open("unicorn.txt", "r")`. The output shows a traceback and an error message: `Traceback (most recent call last):  
 File "<pyshell#33>", line 1, in <module>  
 file = open("unicorn.txt", "r")  
FileNotFoundError: [Errno 2] No such file or directory: 'unicorn.txt'`. In the bottom right corner of the shell window, there is a status bar with the text "Ln: 80 Col: 4".

# 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*)



The screenshot shows a window titled "Python 3.6.5 Shell". Inside, a Python session is running:

```
>>> 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/Versions/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

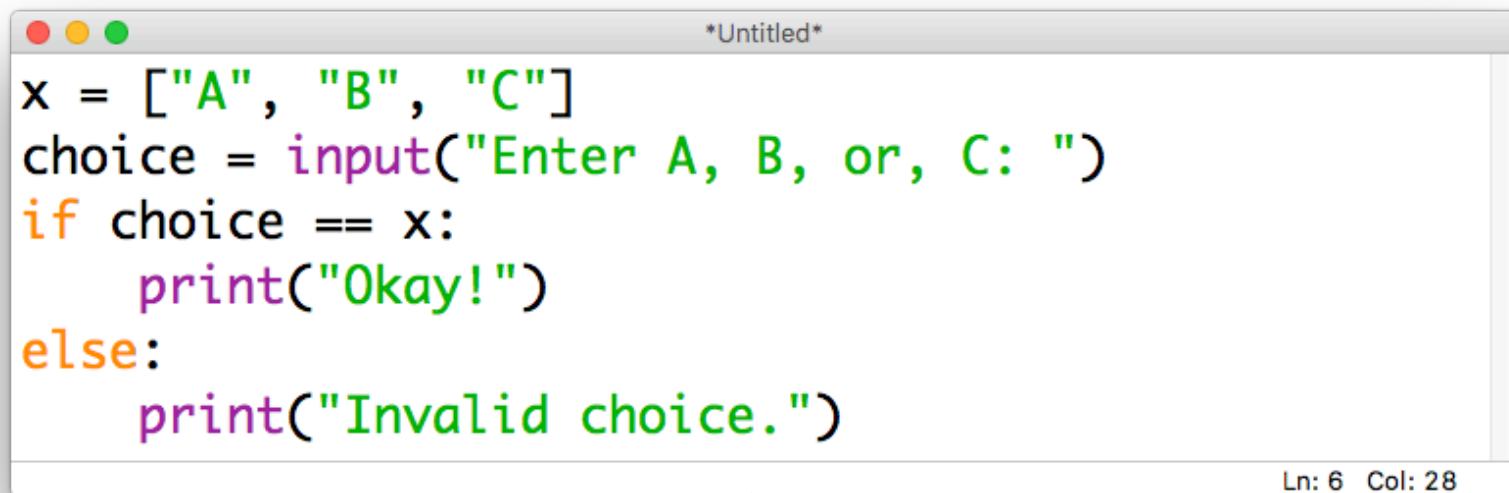
Why would I say that?

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.



The screenshot shows a code editor window titled "Untitled". The code is as follows:

```
x = ["A", "B", "C"]
choice = input("Enter A, B, or, C: ")
if choice == x:
    print("Okay!")
else:
    print("Invalid choice.")
```

The status bar at the bottom right indicates "Ln: 6 Col: 28". A blue arrow points from the text "perfectly valid" to the line "if choice == x:".

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.



```
*Untitled*
```

```
x = ["A", "B", "C"]
choice = input("Enter A, B, or, C: ")
if choice in x:
    print("Okay!")
else:
    print("Invalid choice.")
```

Ln: 6 Col: 28

what we were  
**actually** going for

# An analogy

## Syntactic Error

Their is no  
reason to be  
concerned.

## Logical Error

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

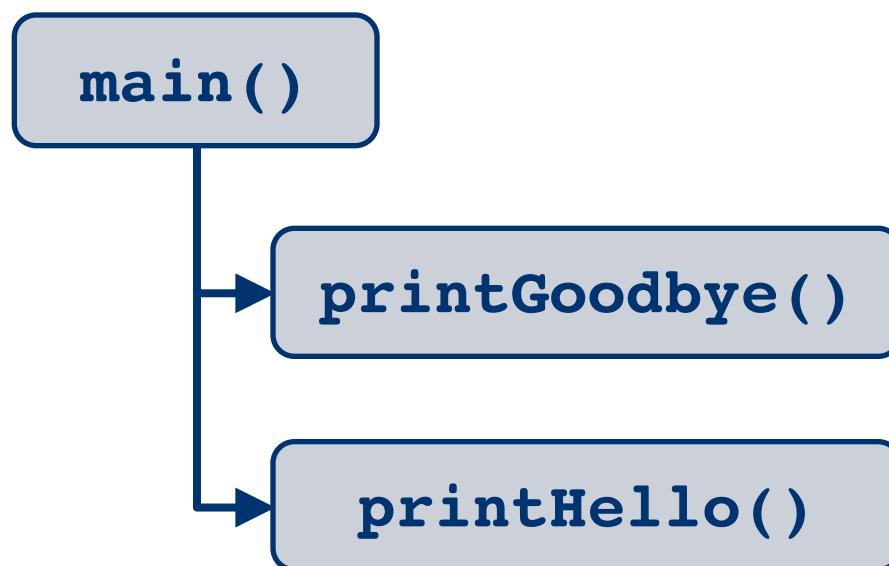
# Discussion

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



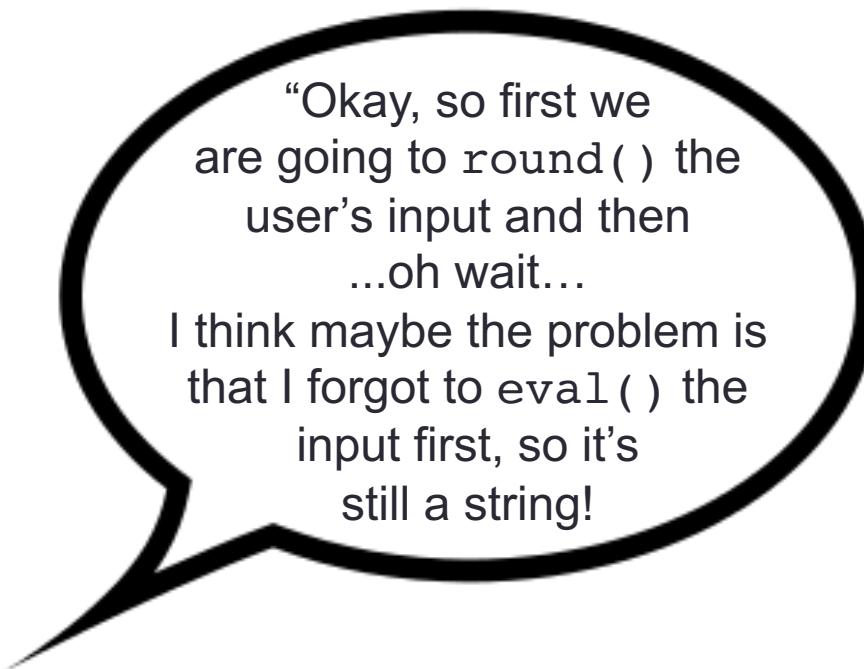
# Step 1: map out the code

- 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:



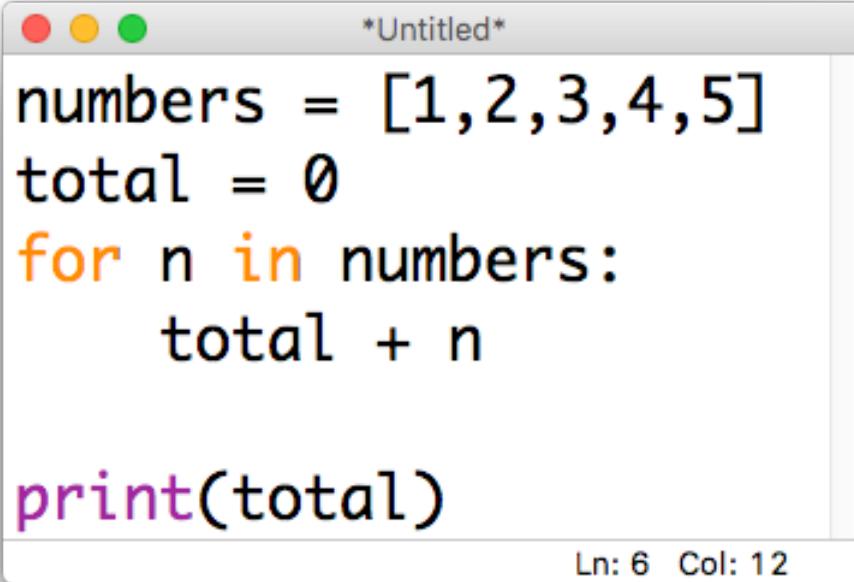
## Step 2: “rubber ducking”

- 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



The image shows a screenshot of a code editor window titled "\*Untitled\*". The code is written in Python and calculates the sum of a list of numbers. It starts by defining a list named 'numbers' containing integers from 1 to 5. A variable 'total' is initialized to 0. A for loop iterates over each number in the list, adding it to the 'total'. Finally, a 'print' statement is used to output the value of 'total'. The code is as follows:

```
numbers = [1,2,3,4,5]
total = 0
for n in numbers:
    total + n

print(total)
```

In the bottom right corner of the code editor, there is a status bar displaying "Ln: 6 Col: 12".

# 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