

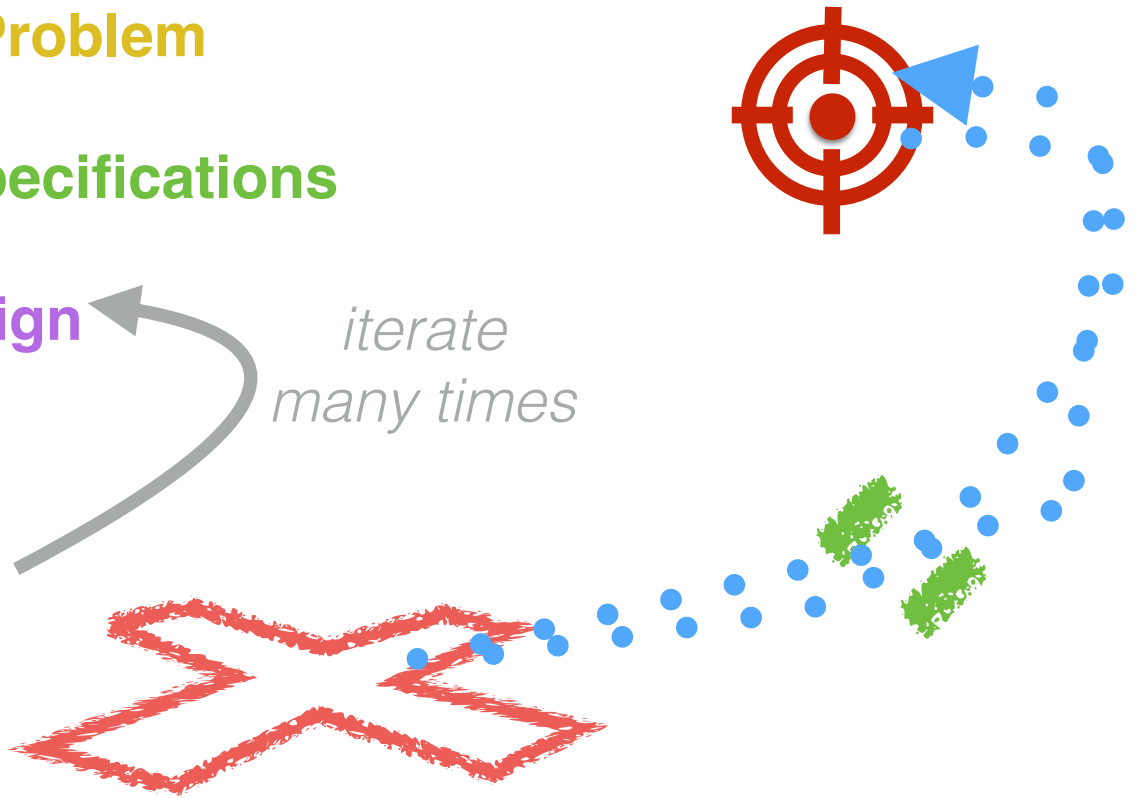
In-class activity: Debugging

Dr. Ab Mosca (they/them)

Slides based off slides courtesy of Jordan Crouser (<https://jcrouser.github.io/>)

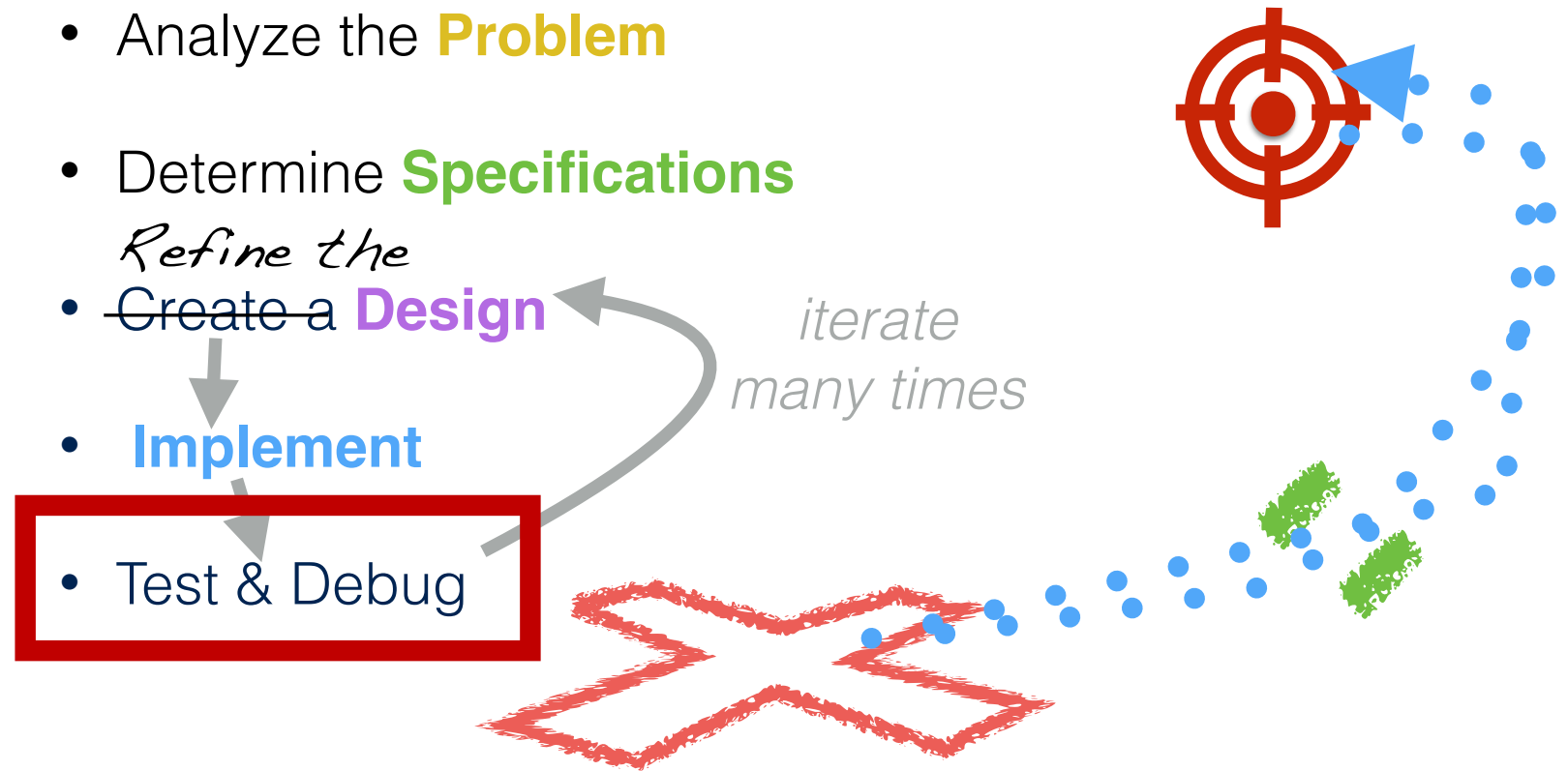
RECAP: the programming process

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



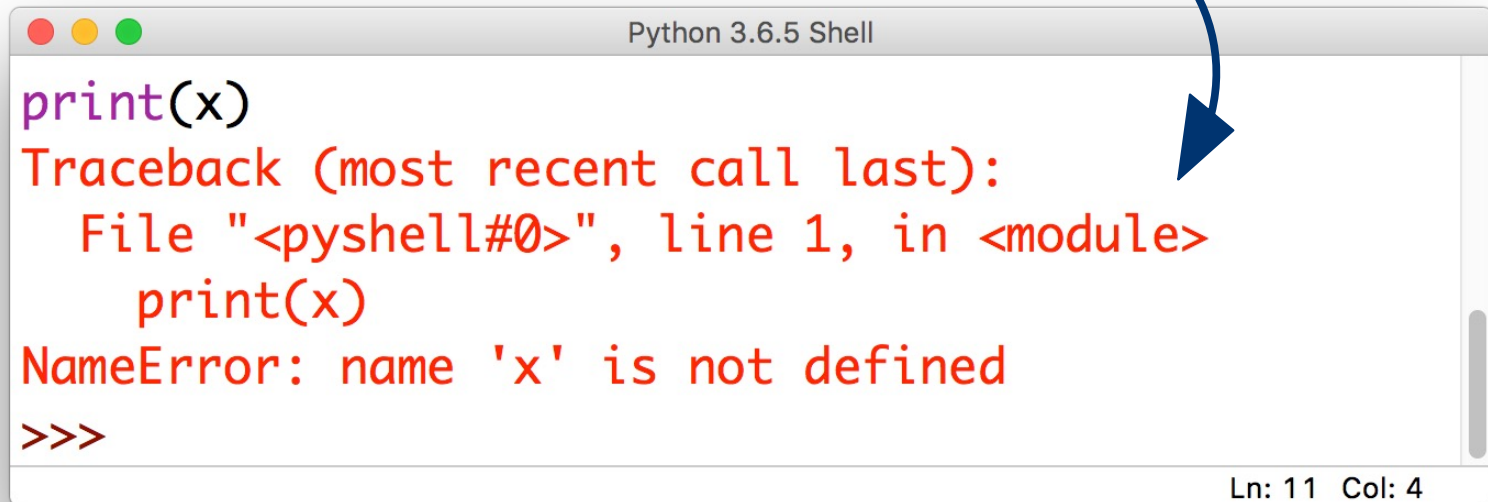
RECAP: the programming process

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



Some
problems are
obvious

this is called
an **Exception**

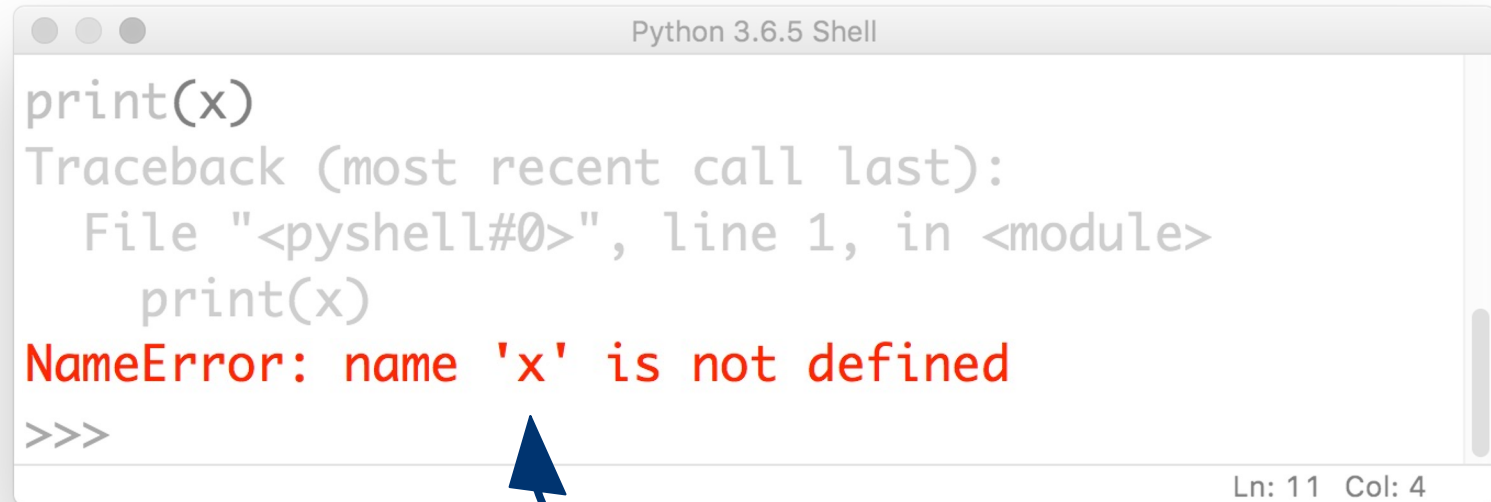


```
Python 3.6.5 Shell
print(x)
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    print(x)
NameError: name 'x' is not defined
>>>
```

Ln: 11 Col: 4

A blue arrow points from the word "Exception" to the "NameError" message in the terminal window.

Some
problems are
obvious

A screenshot of a Python 3.6.5 Shell window. The window has a title bar with three window control buttons (red, yellow, green) on the left and the text "Python 3.6.5 Shell" on the right. The main content area shows the following text: "print(x)" on the first line, "Traceback (most recent call last):" on the second line, "File "<pyshell#0>", line 1, in <module>" on the third line, "print(x)" on the fourth line, and "NameError: name 'x' is not defined" on the fifth line. The error message is highlighted in red. Below the error message is the prompt ">>>". At the bottom right of the window, the status bar shows "Ln: 11 Col: 4".

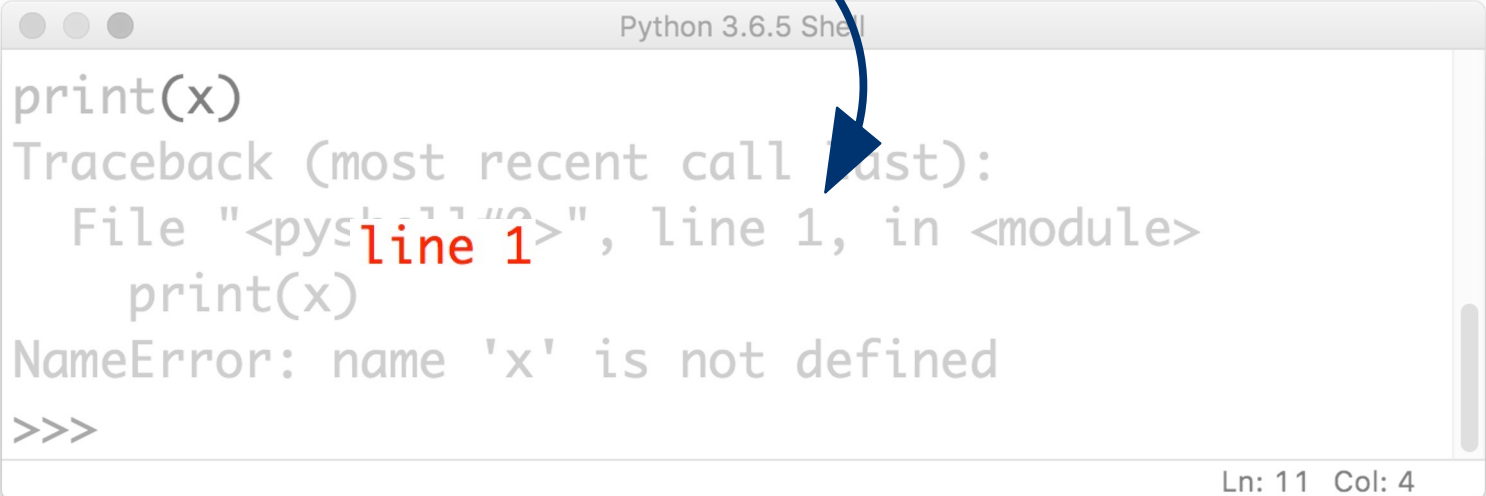
```
Python 3.6.5 Shell

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

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!)

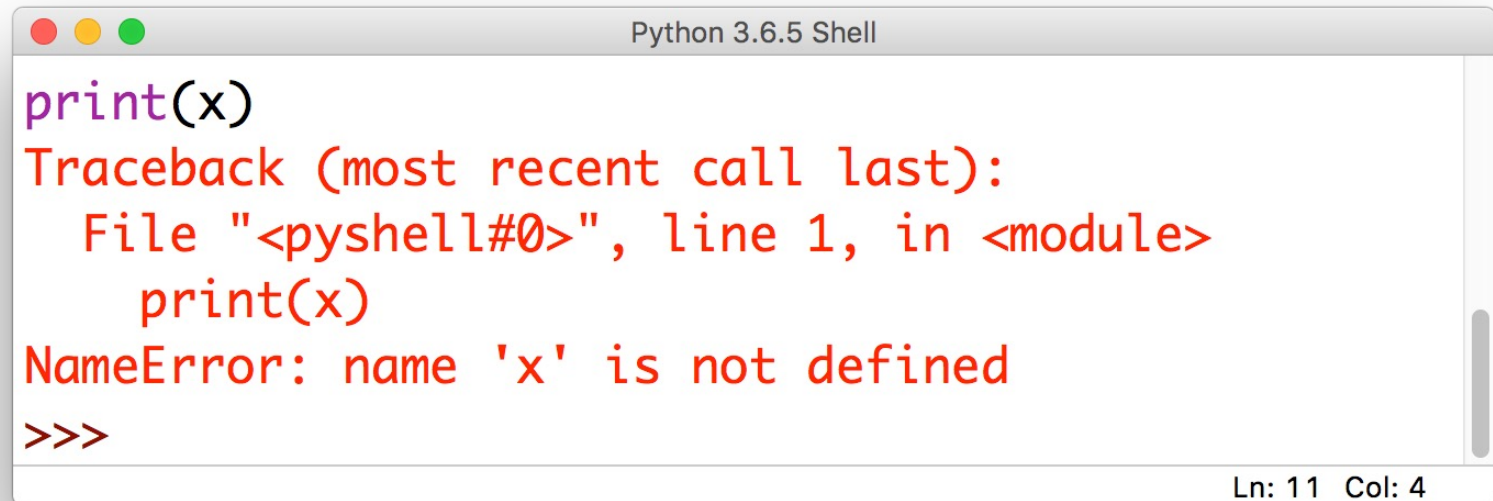


```
Python 3.6.5 Shell
print(x)
Traceback (most recent call last):
  File "<ipython>", line 1, in <module>
    print(x)
NameError: name 'x' is not defined
>>>
```

The image shows a terminal window titled "Python 3.6.5 Shell". It contains the command `print(x)` which has resulted in a `NameError: name 'x' is not defined`. A traceback is shown above the error, indicating the error occurred in the file `<ipython>` at line 1. A blue arrow points from the text "(but be careful!)" to the word "line" in the traceback, highlighting the location of the error.

Common Exceptions

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

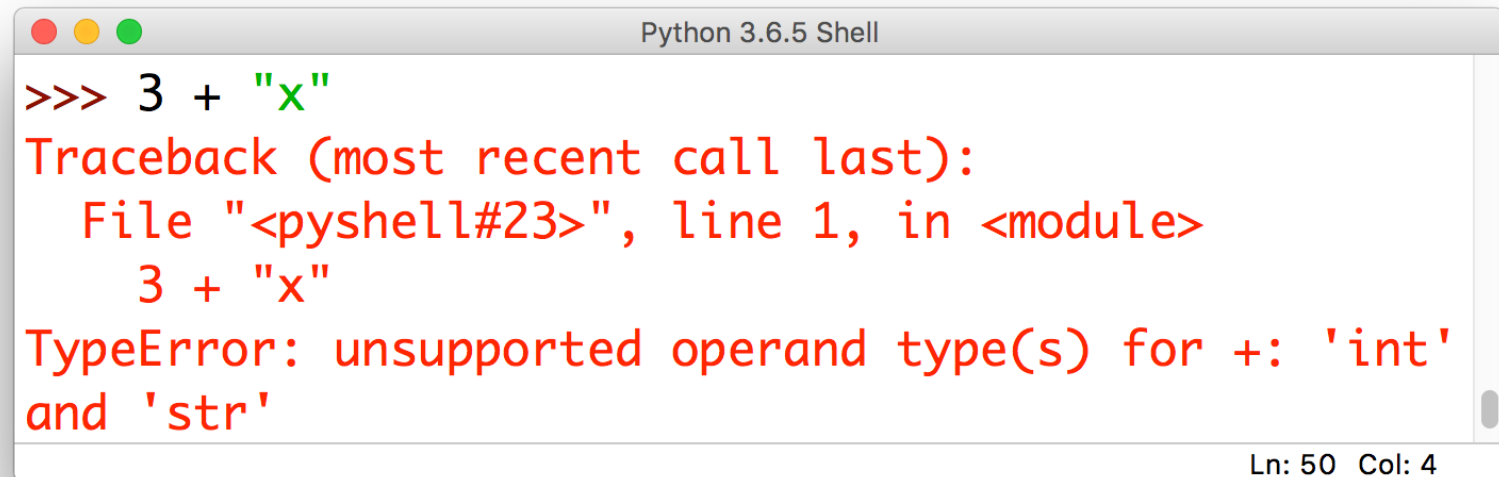
A screenshot of a Python 3.6.5 Shell window. The window has a title bar with three colored buttons (red, yellow, green) and the text "Python 3.6.5 Shell". The main area contains the following text:

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

 The text is color-coded: `print(x)` is purple, the traceback text is red, and the prompt `>>>` is black. A vertical scrollbar is visible on the right side of the text area. At the bottom right of the window, the text "Ln: 11 Col: 4" is displayed.

Common Exceptions

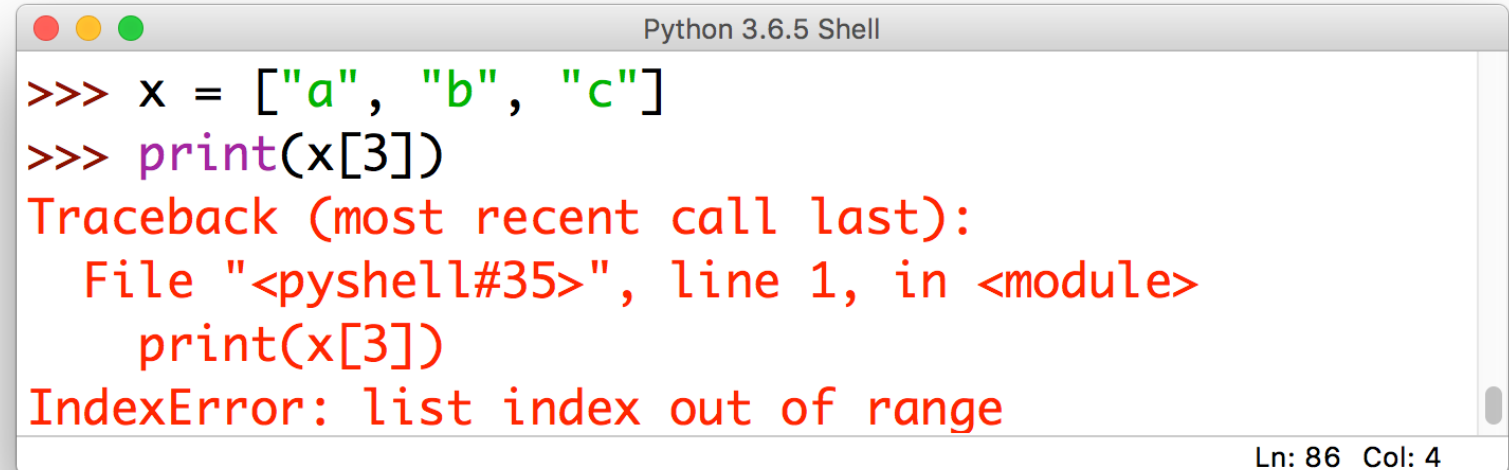
- **TypeError:** raised when you try to perform an operation on an object that's not the right type (i.e. a `string` instead of a number)

A screenshot of a Python 3.6.5 Shell window. The window has a title bar with three colored buttons (red, yellow, green) and the text "Python 3.6.5 Shell". The main area contains the following text: a prompt ">>>" followed by the code "3 + 'x'", a red "Traceback (most recent call last):", a red "File "<pyshell#23>", line 1, in <module>", a red "3 + 'x'", and a red "TypeError: unsupported operand type(s) for +: 'int' and 'str'". The bottom right corner shows "Ln: 50 Col: 4".

```
Python 3.6.5 Shell
>>> 3 + "x"
Traceback (most recent call last):
  File "<pyshell#23>", line 1, in <module>
    3 + "x"
TypeError: unsupported operand type(s) for +: 'int'
and 'str'
Ln: 50 Col: 4
```

Common Exceptions

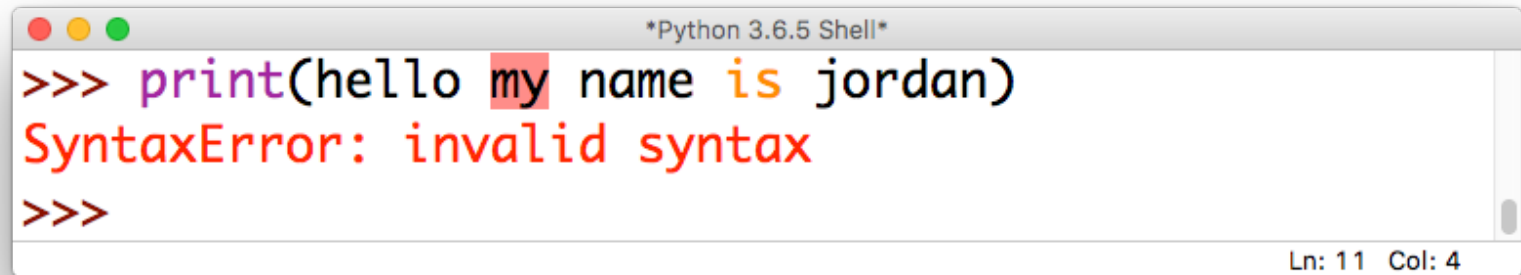
- **IndexError**: raised when you try to use an index that's out of bounds

A screenshot of a Python 3.6.5 Shell window. The window has a title bar with three colored buttons (red, yellow, green) and the text "Python 3.6.5 Shell". The main area contains the following text: a prompt ">>>" followed by "x = ['a', 'b', 'c']" on the next line, another prompt ">>>" followed by "print(x[3])" on the next line. Below this, the text "Traceback (most recent call last):" is shown, followed by "File "<pyshell#35>", line 1, in <module>" and "print(x[3])" on the next line. The final line is "IndexError: list index out of range". At the bottom right of the window, it says "Ln: 86 Col: 4".

```
Python 3.6.5 Shell
>>> 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

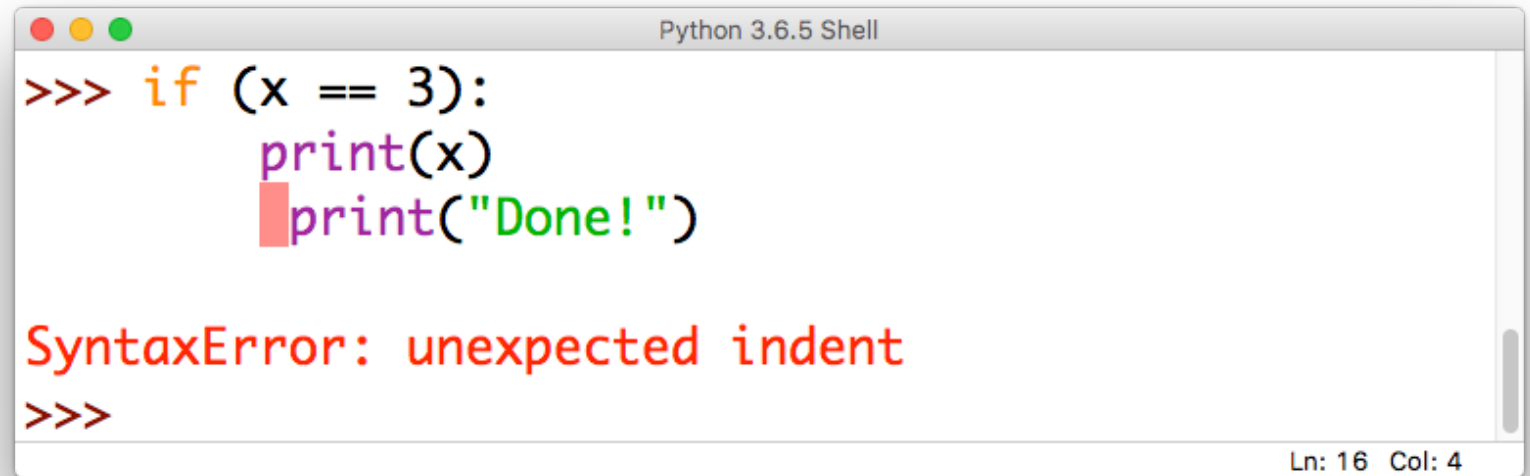
- **SyntaxError**: raised when you try to run a command that isn't a valid Python statement

A screenshot of a Python 3.6.5 Shell window. The window has a title bar with three colored buttons (red, yellow, green) and the text '*Python 3.6.5 Shell*'. The main area shows a Python prompt '>>>' followed by the code 'print(hello my name is jordan)'. The word 'my' is highlighted with a red background. Below the code, the text 'SyntaxError: invalid syntax' is displayed in red. Another prompt '>>>' is shown on the next line. The bottom right corner of the window shows 'Ln: 11 Col: 4'.

```
*Python 3.6.5 Shell*
>>> print(hello my name is jordan)
SyntaxError: invalid syntax
>>>
```

Common Exceptions

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



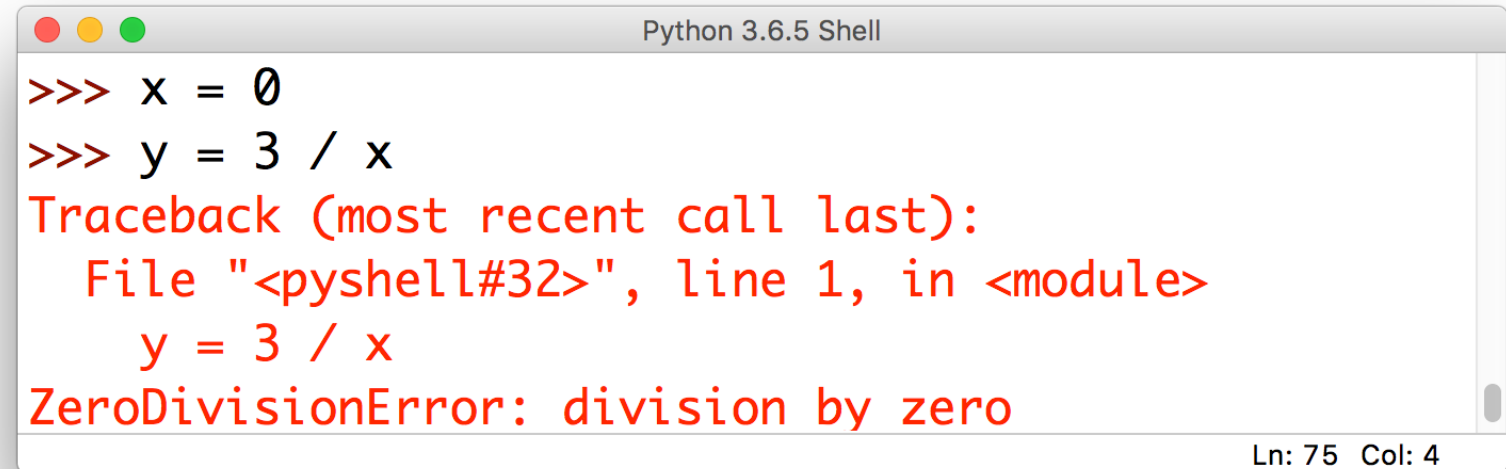
```
Python 3.6.5 Shell
>>> if (x == 3):
      print(x)
      print("Done!")

SyntaxError: unexpected indent
>>>
```

Ln: 16 Col: 4

Common Exceptions

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

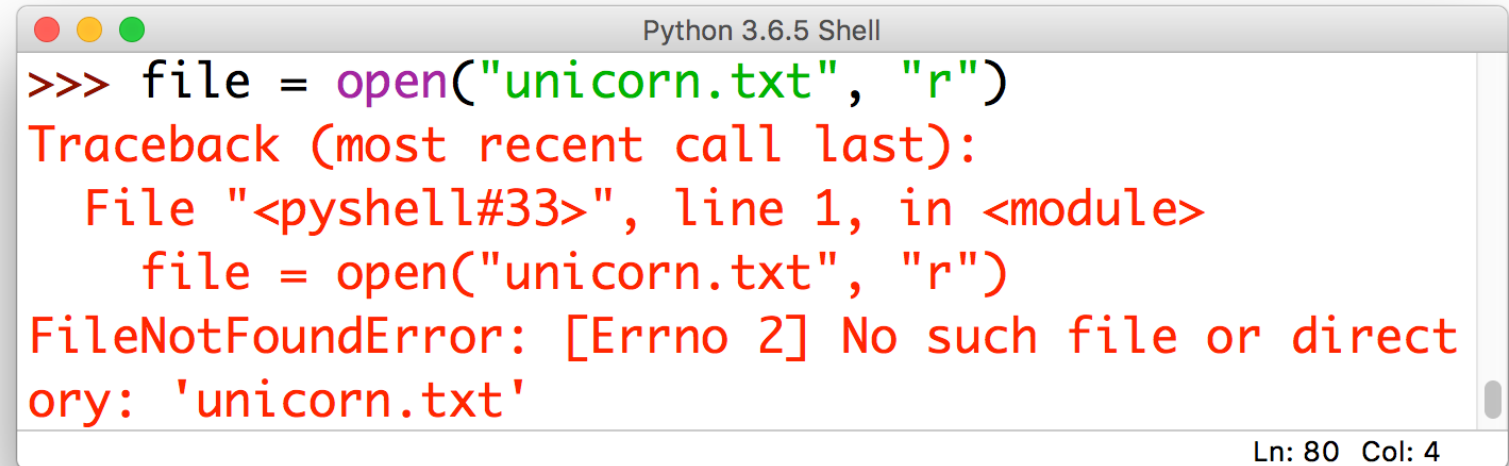


```
Python 3.6.5 Shell
>>> x = 0
>>> y = 3 / x
Traceback (most recent call last):
  File "<pyshell#32>", line 1, in <module>
    y = 3 / x
ZeroDivisionError: division by zero
Ln: 75 Col: 4
```

The screenshot shows a terminal window titled "Python 3.6.5 Shell". It contains two lines of Python code: `>>> x = 0` and `>>> y = 3 / x`. After the second line, a red traceback message is displayed: `Traceback (most recent call last):`, `File "<pyshell#32>", line 1, in <module>`, `y = 3 / x`, and `ZeroDivisionError: division by zero`. The status bar at the bottom right indicates "Ln: 75 Col: 4".

Common Exceptions

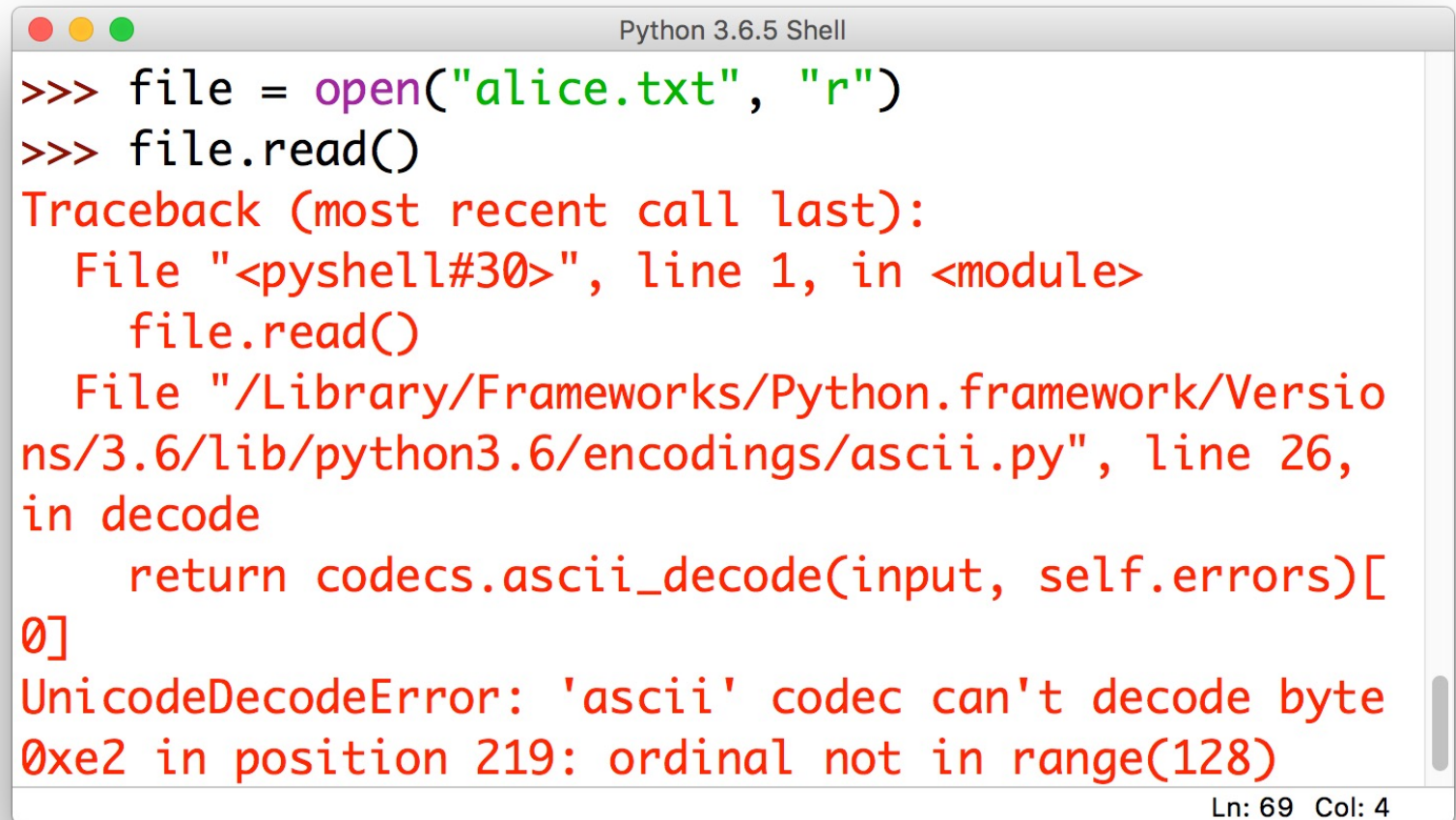
- **FileNotFoundError**: raised when Python can't find the thing you're referring to (a file)

A screenshot of a Python 3.6.5 Shell window. The window has a title bar with three colored buttons (red, yellow, green) and the text "Python 3.6.5 Shell". The main area contains the following text: a prompt ">>>" followed by the code "file = open('unicorn.txt', 'r')", a red "Traceback (most recent call last):" message, a red line "File '<pyshell#33>', line 1, in <module>", a red line "file = open('unicorn.txt', 'r')", and a red "FileNotFoundError: [Errno 2] No such file or directory: 'unicorn.txt'" message. At the bottom right, it says "Ln: 80 Col: 4".

```
Python 3.6.5 Shell
>>> file = open("unicorn.txt", "r")
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'
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*)

A screenshot of a Python 3.6.5 Shell window. The window has a title bar with three colored buttons (red, yellow, green) and the text "Python 3.6.5 Shell". The main area contains the following text:

```
>>> 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)
```

At the bottom right of the window, it says "Ln: 69 Col: 4".

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?

(Hint: we looked at an example last week)

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 == x:  
    print("Okay!")  
else:  
    print("Invalid choice.")  
Ln: 6 Col: 28
```

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

There is no
reason to be
concerned.

Logical Error

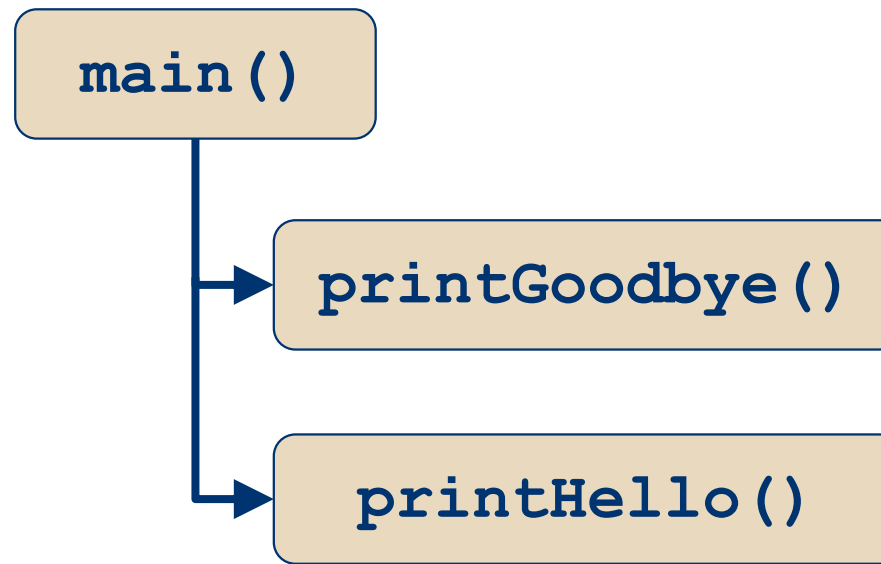
If an animal is
green, it must
be a frog.

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”

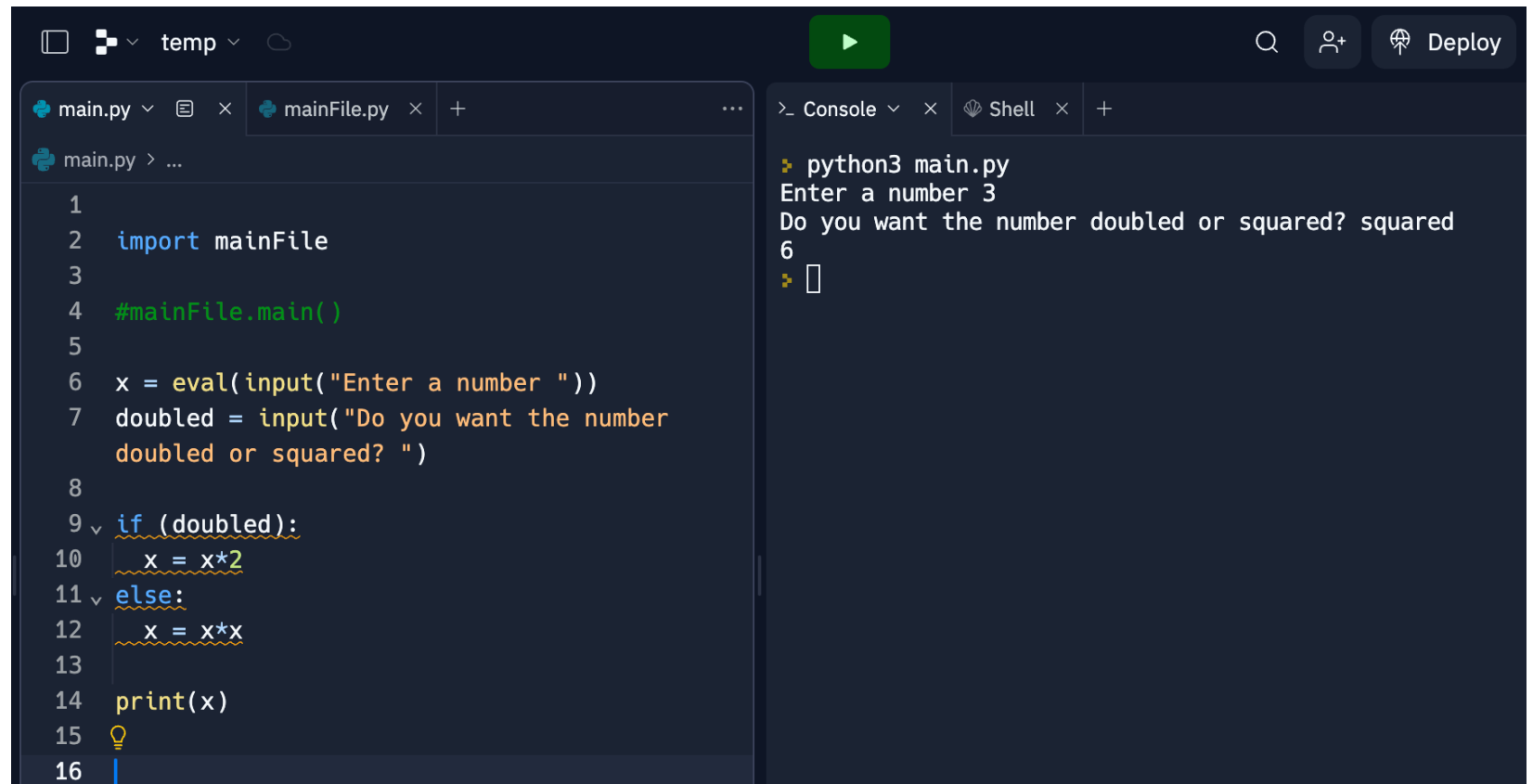
- Still stuck? Try explaining it to someone else (or historically, to a rubber duckie)
- This is the debugging equivalent of **pair programming**

“Okay, so first we are going to `round()` the user’s input and then ...oh wait... I think maybe the problem is that I forgot to `eval()` the input first, so it’s still a string!



Step 3: add `print()` statements

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



The screenshot shows a code editor with two tabs: `main.py` and `mainFile.py`. The `main.py` tab is active, displaying the following Python code:

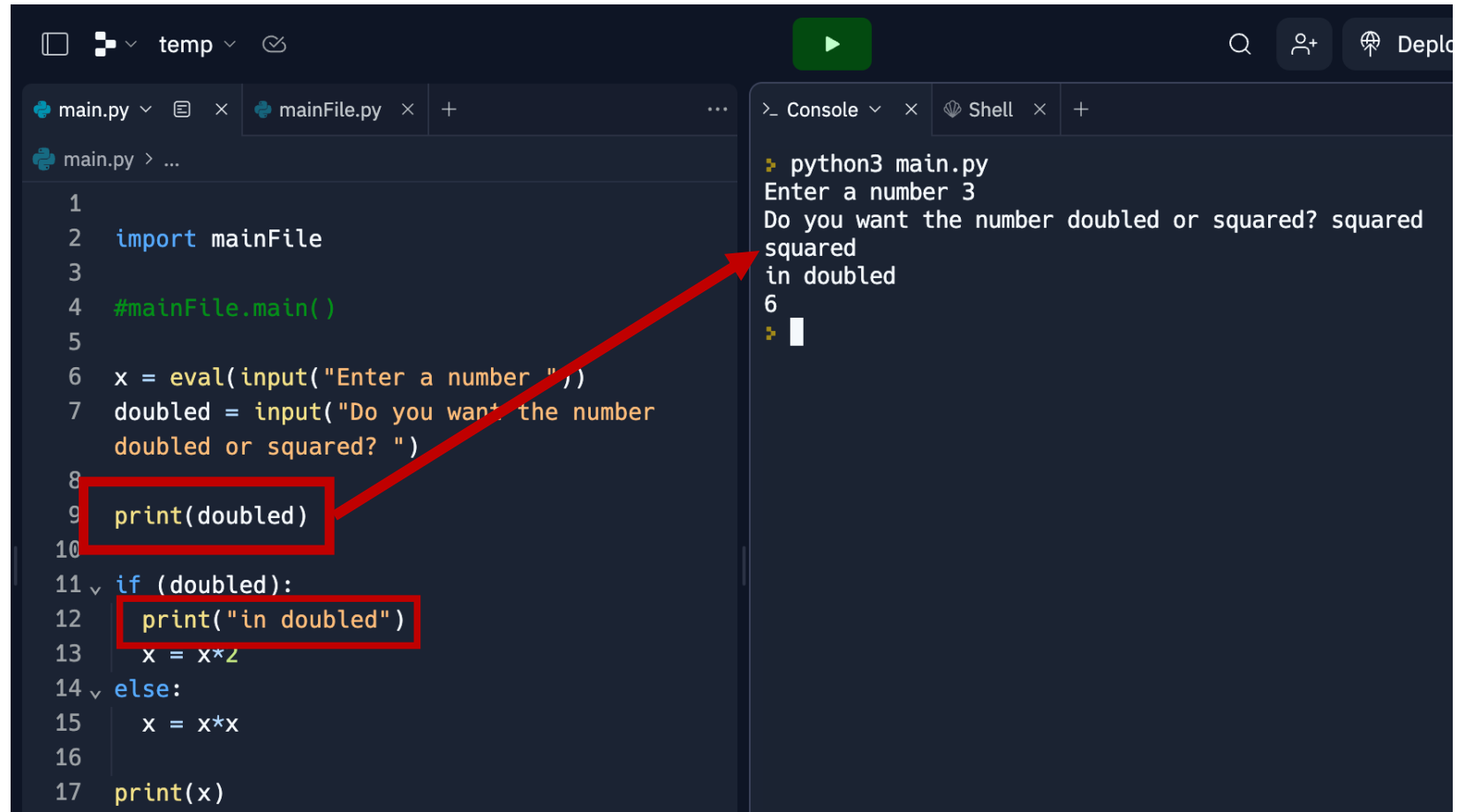
```
1
2 import mainFile
3
4 #mainFile.main()
5
6 x = eval(input("Enter a number "))
7 doubled = input("Do you want the number
8 doubled or squared? ")
9
10 if (doubled):
11     x = x*2
12 else:
13     x = x*x
14
15 print(x)
16
```

The console on the right shows the output of running `python3 main.py`:

```
> python3 main.py
Enter a number 3
Do you want the number doubled or squared? squared
6
```

Step 3: add
print()
statements

- Not sure exactly where things are going wrong?
- Add **print()** statements to leave a "trail" on the console

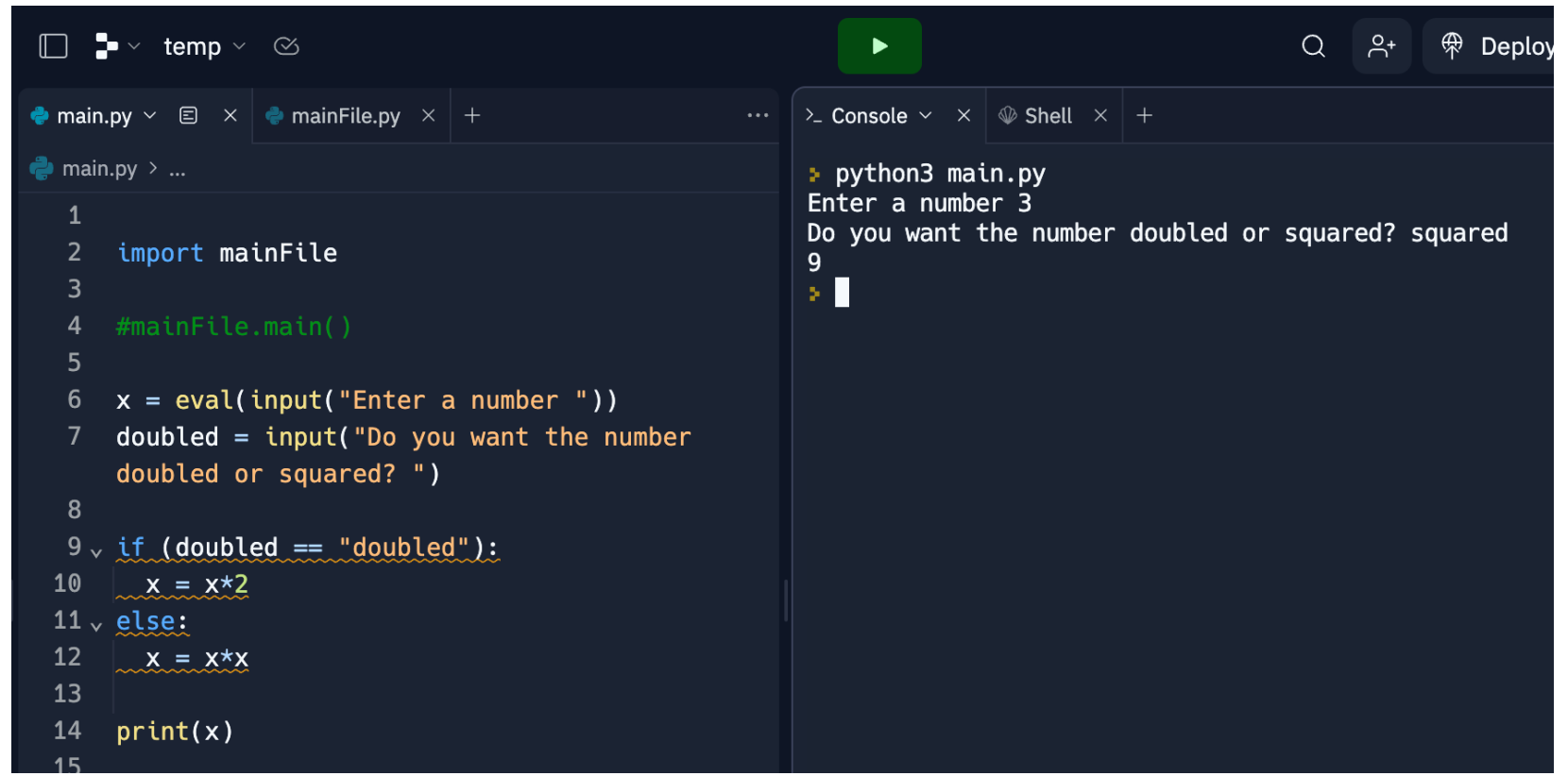


```
main.py > ...
1
2 import mainFile
3
4 #mainFile.main()
5
6 x = eval(input("Enter a number "))
7 doubled = input("Do you want the number
doubled or squared? ")
8
9 print(doubled)
10
11 if (doubled):
12     print("in doubled")
13     x = x*2
14 else:
15     x = x*x
16
17 print(x)
```

```
> python3 main.py
Enter a number 3
Do you want the number doubled or squared? squared
in doubled
6
>
```

Step 3: add `print()` statements

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



The screenshot shows a code editor with two tabs: `main.py` and `mainFile.py`. The `main.py` tab is active, displaying a Python script. The script imports `mainFile` and calls `mainFile.main()`. It then prompts the user to enter a number and whether they want the number doubled or squared. Based on the input, it either doubles the number or squares it and prints the result.

```
1
2 import mainFile
3
4 #mainFile.main()
5
6 x = eval(input("Enter a number "))
7 doubled = input("Do you want the number
8 doubled or squared? ")
9
10 if (doubled == "doubled"):
11     x = x*2
12 else:
13     x = x*x
14 print(x)
15
```

The console on the right shows the execution of `python3 main.py`. The user entered `3` for the number and `squared` for the operation. The console output shows the number `9` being printed.

```
> python3 main.py
Enter a number 3
Do you want the number doubled or squared? squared
9
```

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!

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.
# -----
```

Discussion

What did you find?