

Intro to Coding with Python— Documentation and Debugging

Dr. Ab Mosca (they/them)

Plan for Today

- Documenting code
- Tracing code
- Debugging

Big Picture

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

The point

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

... but how?



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?

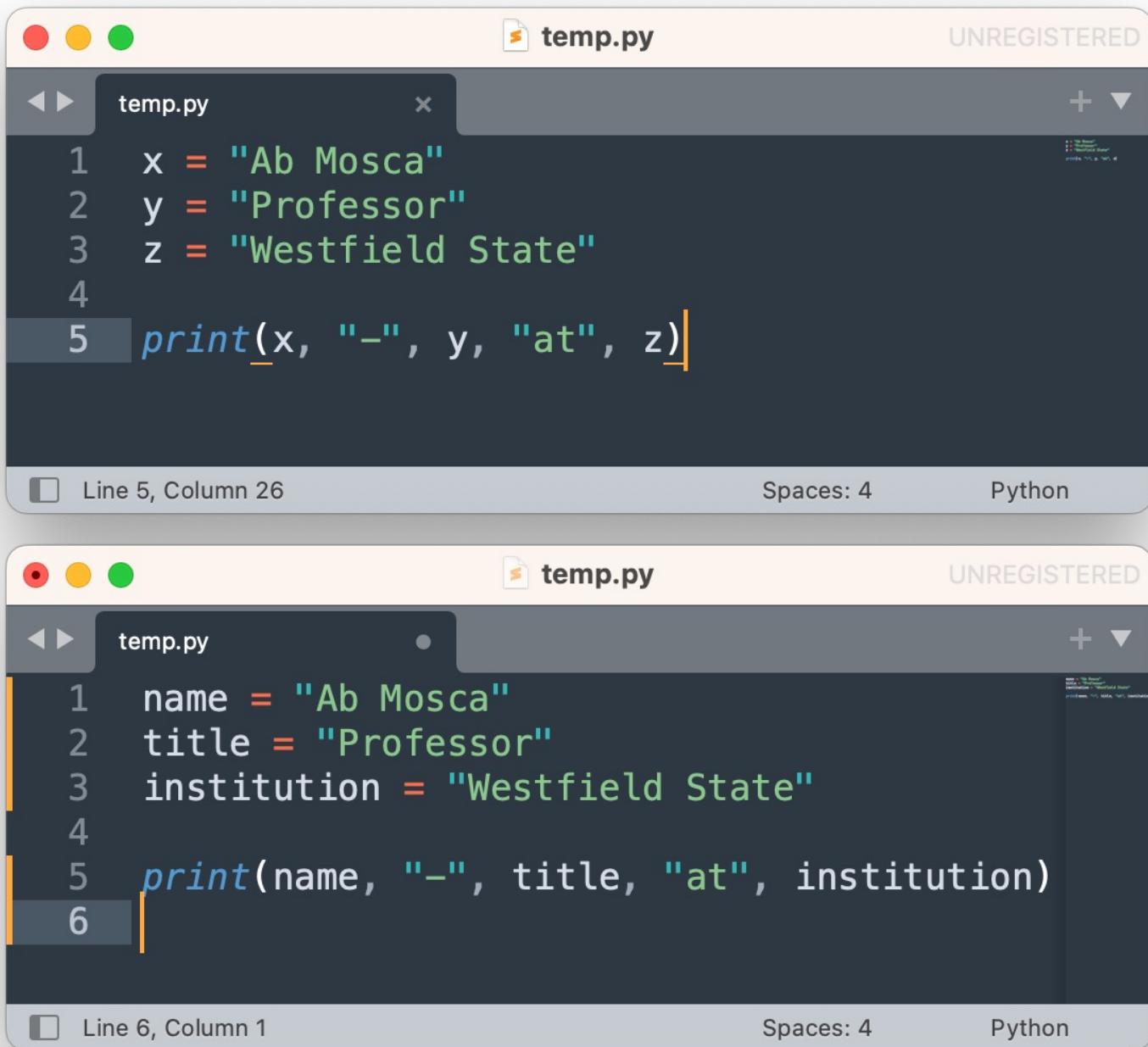
Step 1: meaningful nouns for variables



```
temp.py
1 x = "Ab Mosca"
2 y = "Professor"
3 z = "Westfield State"
4
5 print(x, "-", y, "at", z)
```

Line 5, Column 26 Spaces: 4 Python

Step 1: meaningful nouns for variables



The image shows two screenshots of a Python code editor, likely PyCharm, displaying the file `temp.py`. Both screenshots show the same code with different variable names.

Top Screenshot:

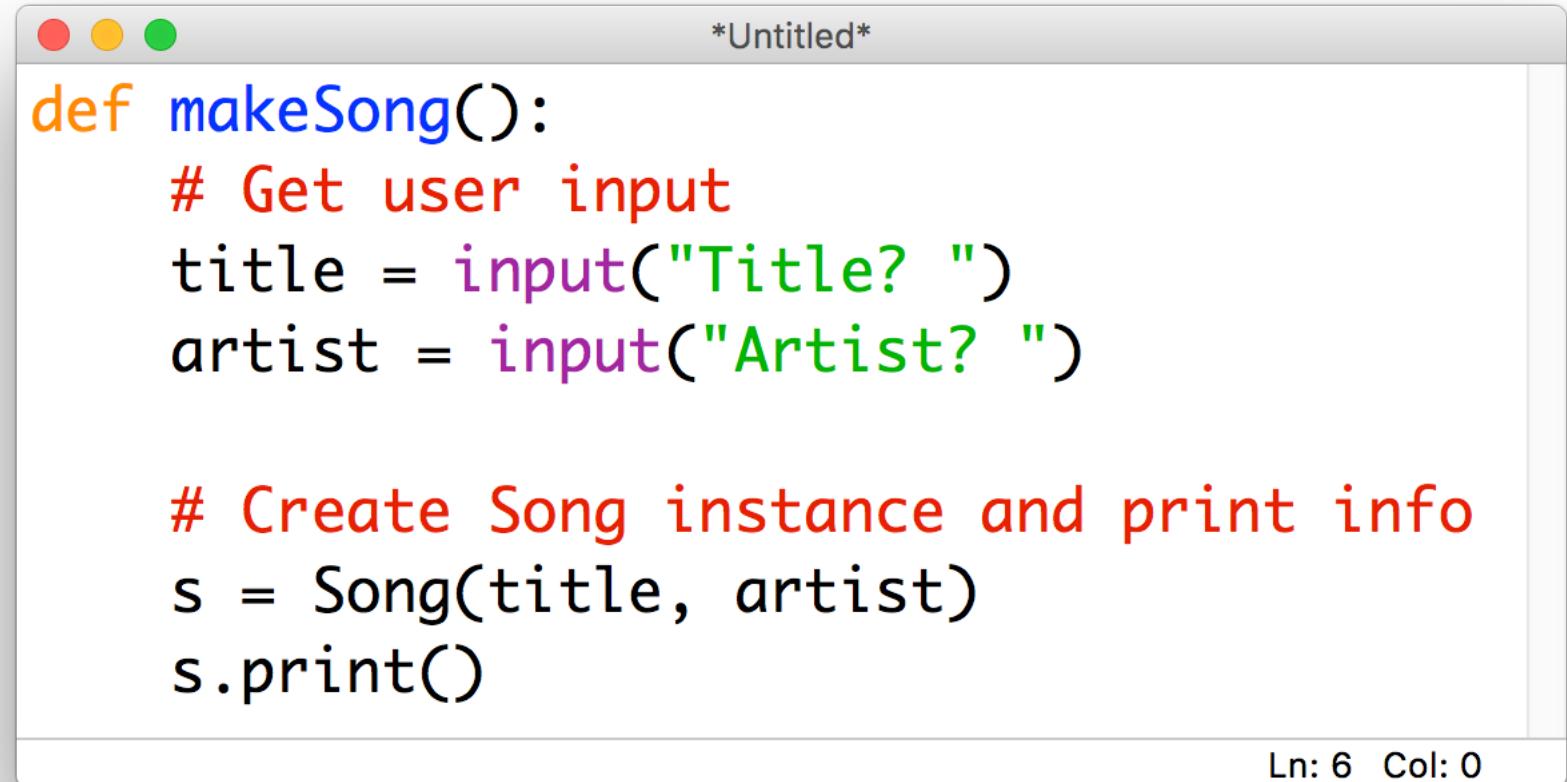
```
1 x = "Ab Mosca"
2 y = "Professor"
3 z = "Westfield State"
4
5 print(x, "-", y, "at", z)
```

Bottom Screenshot:

```
1 name = "Ab Mosca"
2 title = "Professor"
3 institution = "Westfield State"
4
5 print(name, "-", title, "at", institution)
6
```

The code in both screenshots prints the same string: "Ab Mosca - Professor at Westfield State". The variables have been renamed from `x`, `y`, and `z` to more descriptive nouns: `name`, `title`, and `institution`.

Step 2: lots of comments



```
def makeSong():
    # Get user input
    title = input("Title? ")
    artist = input("Artist? ")

    # Create Song instance and print info
    s = Song(title, artist)
    s.print()
```

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 running lines
 - 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)|
```

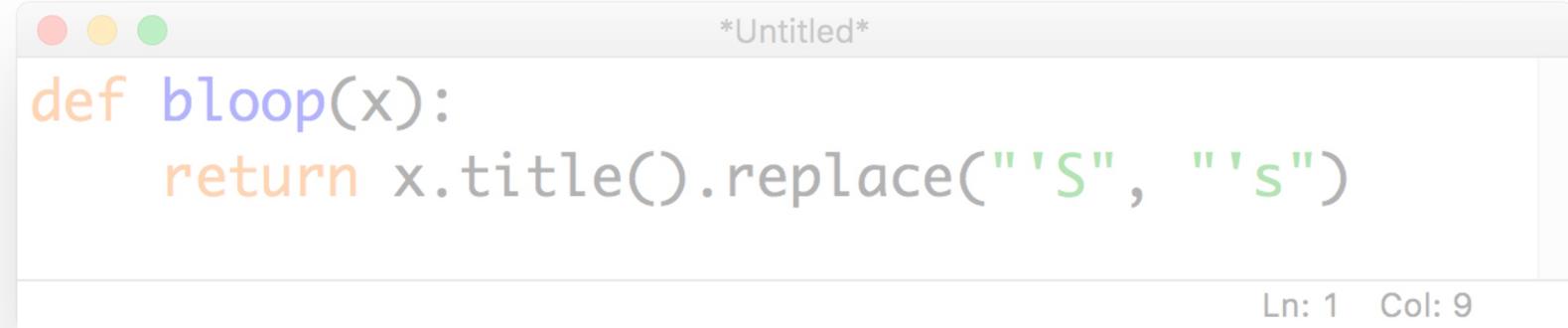
Ln: 12 Col: 16

Step 3*:
describe the
action for
functions

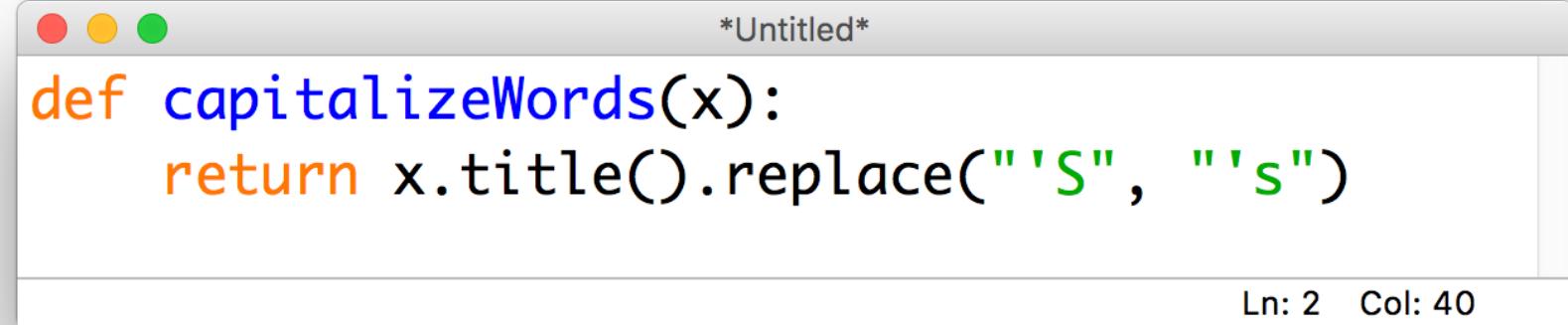


```
*Untitled*
def bloop(x):
    return x.title().replace("S", "s")
Ln: 1 Col: 9
```

Step 3*:
describe the
action for
functions

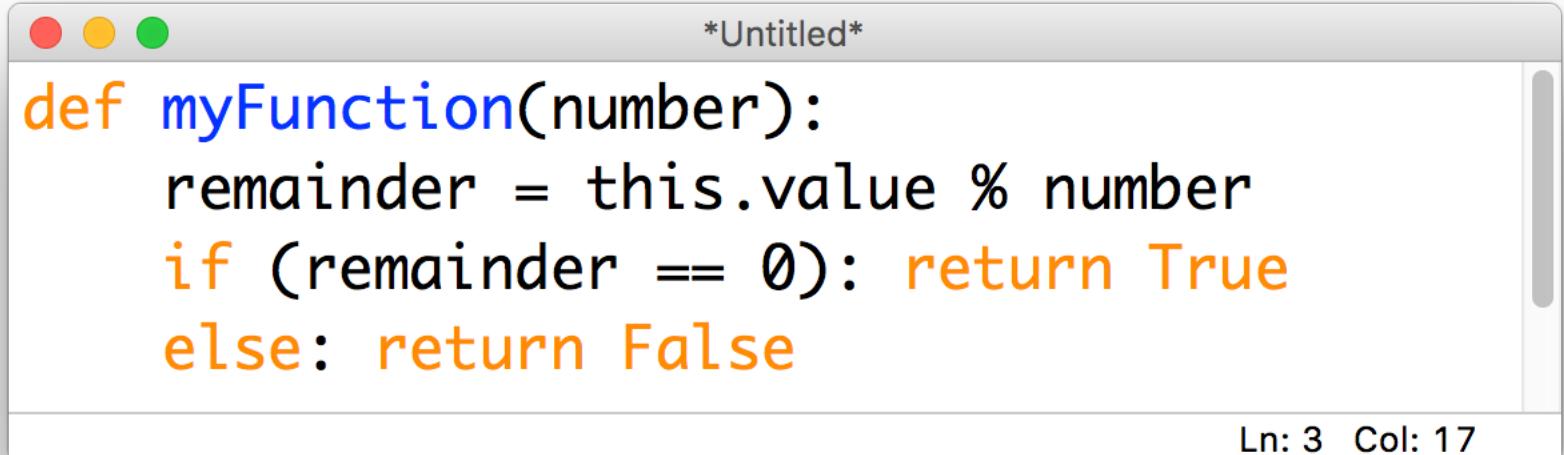


```
*Untitled*
def bloop(x):
    return x.title().replace("S", "s")
Ln: 1 Col: 9
```

```
*Untitled*
def capitalizeWords(x):
    return x.title().replace("S", "s")
Ln: 2 Col: 40
```

Step 3*:
describe the
action for
functions

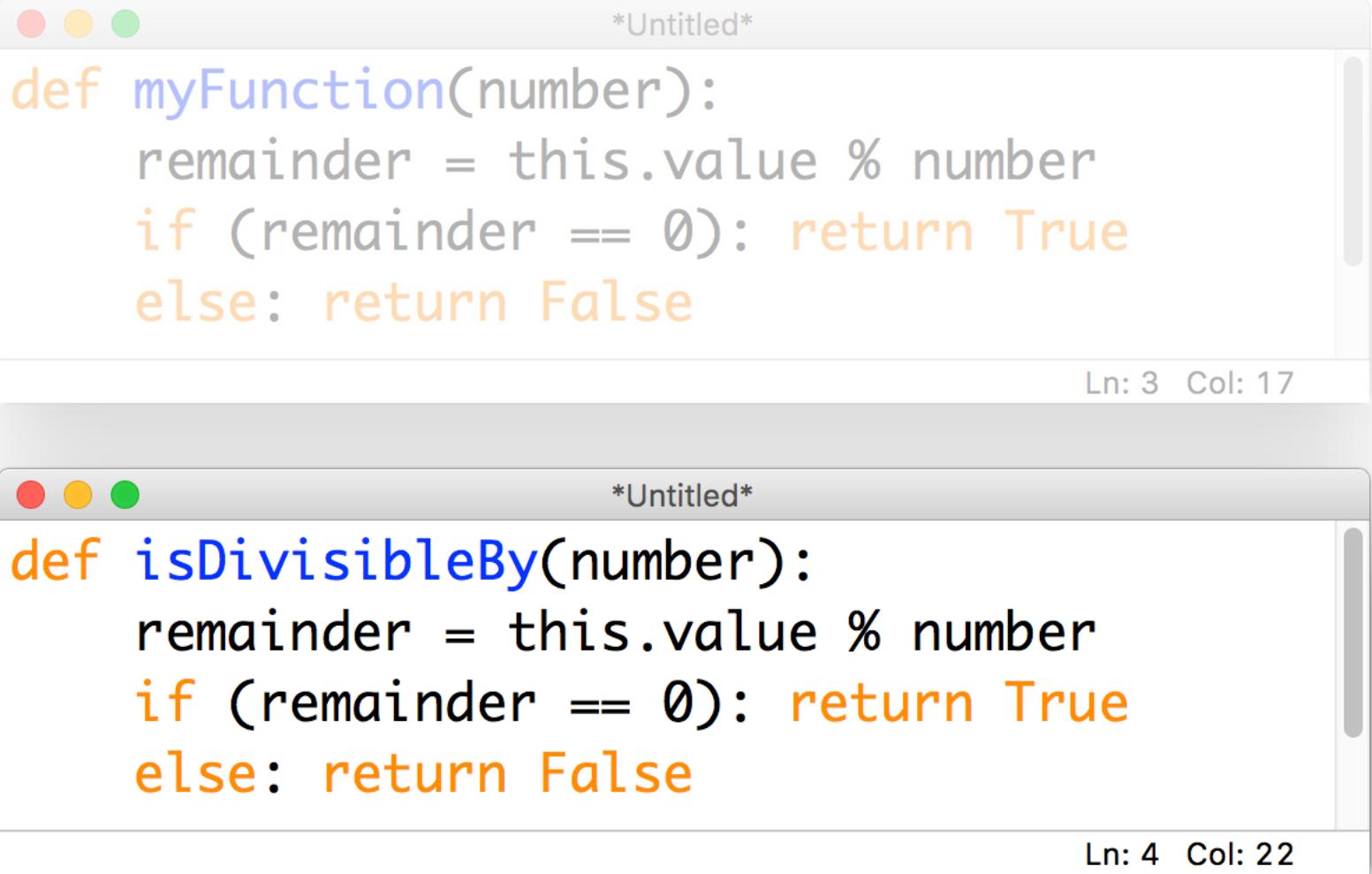


```
*Untitled*
```

```
def myFunction(number):
    remainder = this.value % number
    if (remainder == 0): return True
    else: return False
```

Ln: 3 Col: 17

Step 3*: describe the action for functions



The image shows two code editor windows side-by-side. Both windows have a title bar with three colored dots (red, yellow, green) and the text "*Untitled*". The top window contains the following Python code:

```
def myFunction(number):
    remainder = this.value % number
    if (remainder == 0): return True
    else: return False
```

In the bottom right corner of this window, the text "Ln: 3 Col: 17" is visible.

The bottom window contains the same Python code:

```
def isDivisibleBy(number):
    remainder = this.value % number
    if (remainder == 0): return True
    else: return False
```

In the bottom right corner of this window, the text "Ln: 4 Col: 22" is visible.

Step 4*: docstrings

```
*Untitled*
```

```
def makeSong():
    """ Creates and prints a Song instance
        from user input"""
    # Get user input
    title = input("Title? ")
    artist = input("Artist? ")

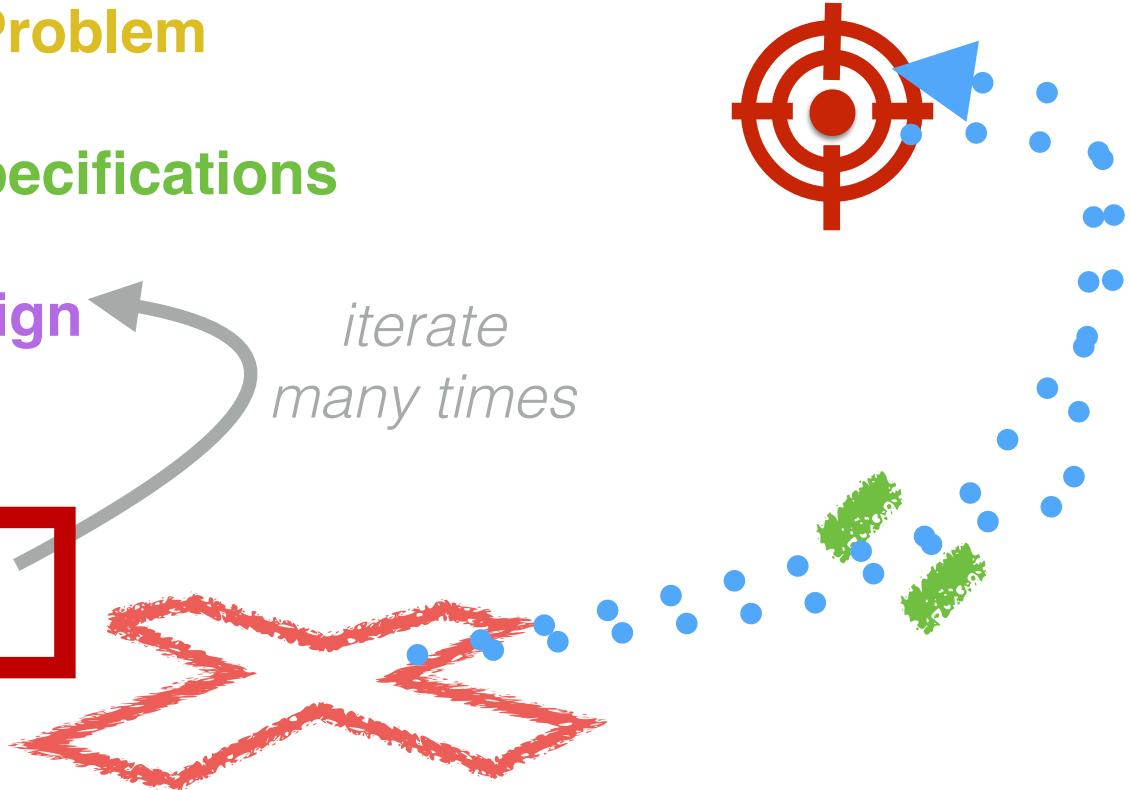
    # Create Song instance and print info
    s = Song(title, artist)
    s.print()
```

Ln: 5 Col: 0

Debugging

RECAP: the programming process

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



Fun history: the term “debug”

9/9

- 0800 Antran started
1000 " stopped - antran ✓
1300 (032) MP-MC { 1.2700 9.037 847 025
 2.1304767675 (-3) 9.037 846 995 const
 (033) PRO 2 2.130476415
 const 2.130676415
Relays 6-2 in 033 failed special sped test
in relay " 10.00 test.
Relays changed
1100 Started Cosine Tape (Sine check)
1525 Started Multi Adder Test.
1545 Relay #70 Panel F
(moth) in relay.
1630 First actual case of bug being found.
Antran started.
1700 closed down.

RDML Grace M. Hopper
b.1906 – d.1992



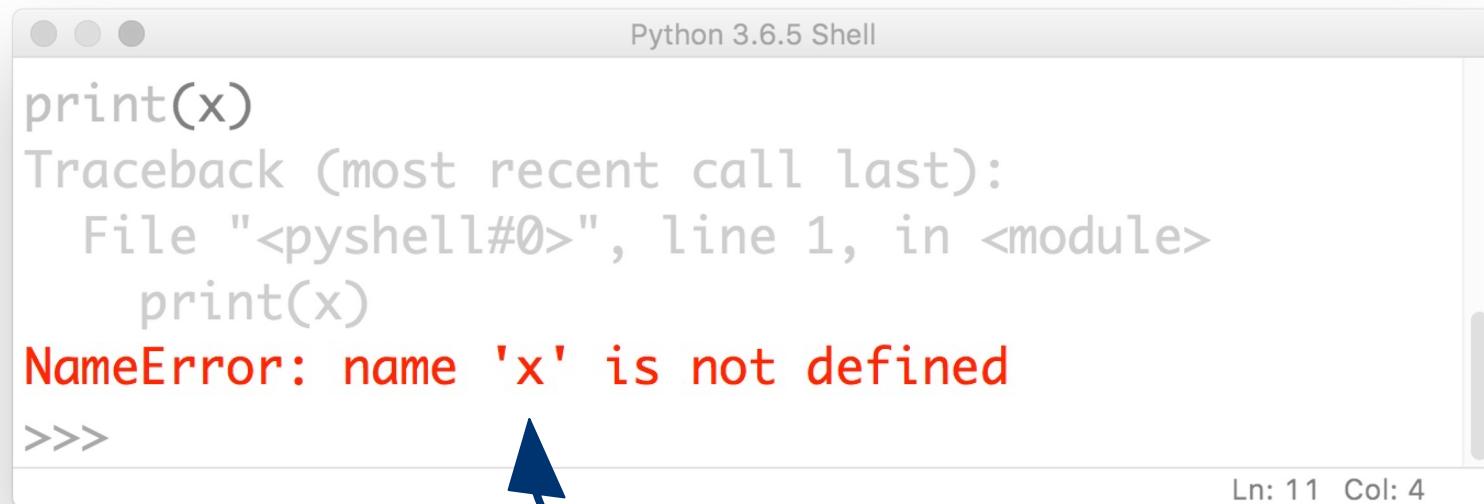
Some
problems are
obvious

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 the 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 command prompt ">>> " is at the bottom left, and "Ln: 11 Col: 4" is at the bottom right. A blue arrow points from the text "this kind of error gives you a **clue** about what the problem is" to the word "NameError" in the error message.

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

this 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!)

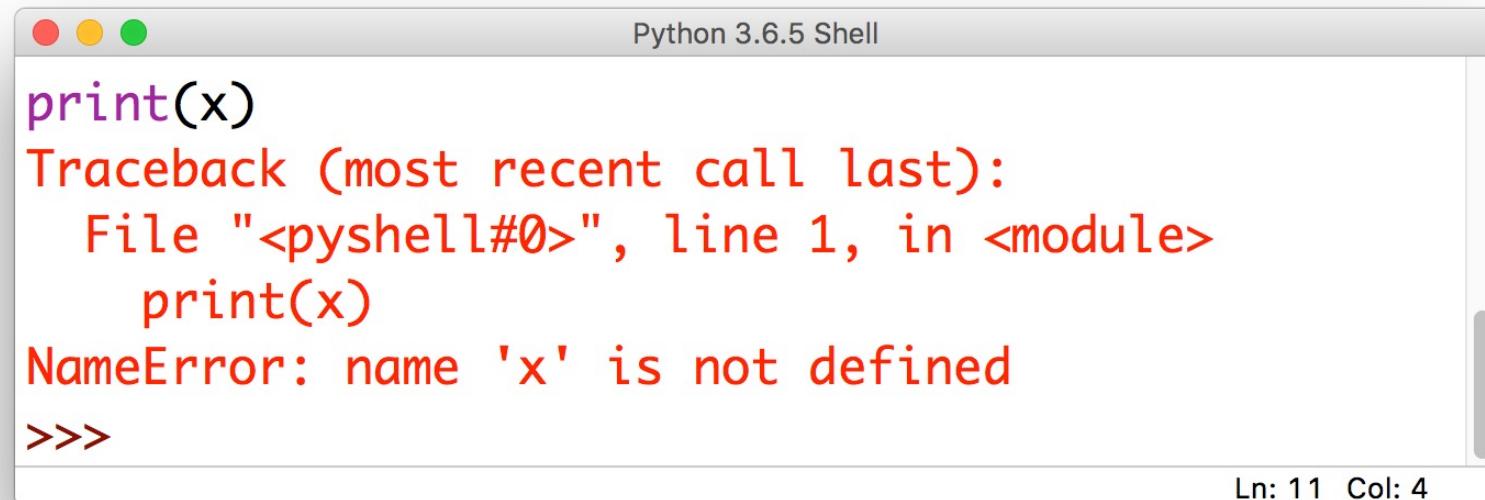


```
Python 3.6.5 Shell
print(x)
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    print(x)
NameError: name 'x' is not defined
>>>
```

Ln: 11 Col: 4

Common Exceptions

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



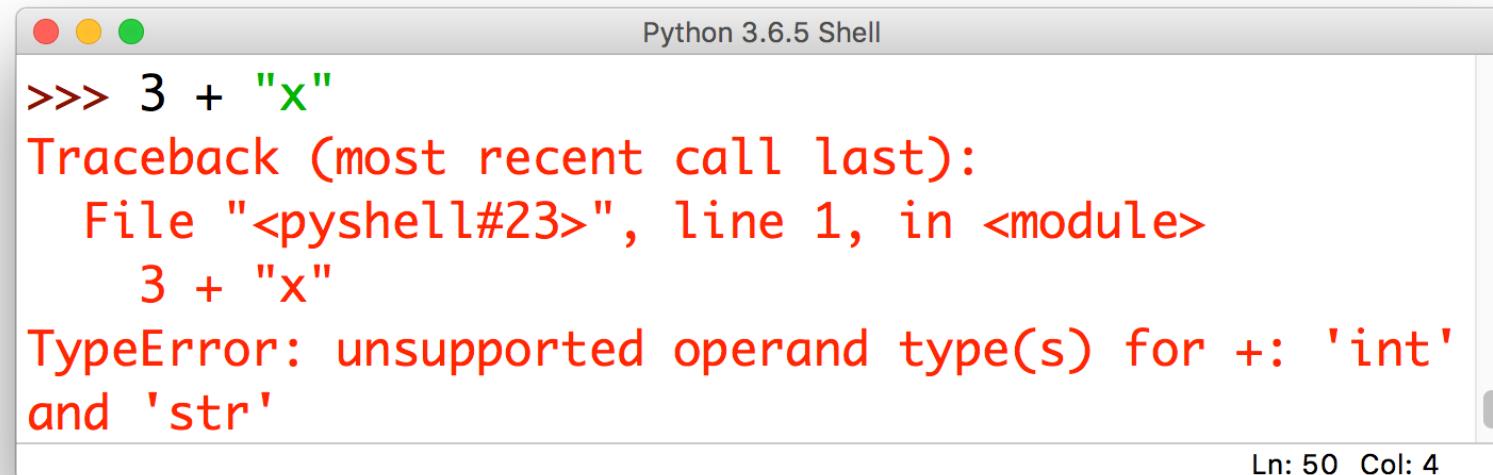
A screenshot of a Python 3.6.5 Shell window. The title bar says "Python 3.6.5 Shell". The code input area shows "print(x)". The output area displays a traceback and an error message: "Traceback (most recent call last): File "<pyshell#0>", line 1, in <module> print(x) NameError: name 'x' is not defined". The bottom status bar indicates "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
>>>
```

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)

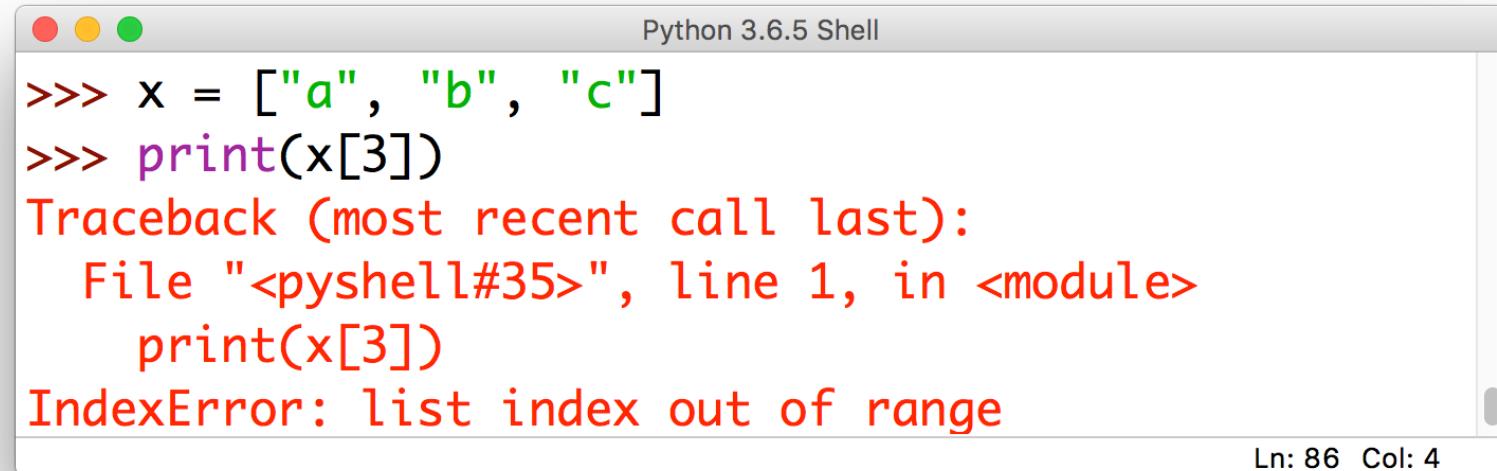


A screenshot of a Python 3.6.5 Shell window. The title bar says "Python 3.6.5 Shell". The code input is ">>> 3 + \"x\"". The output shows a traceback and the resulting `TypeError`:
Traceback (most recent call last):
 File "<pyshell#23>", line 1, in <module>
 3 + "x"
TypeError: unsupported operand type(s) for +: 'int'
and 'str'

Ln: 50 Col: 4

Common Exceptions

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



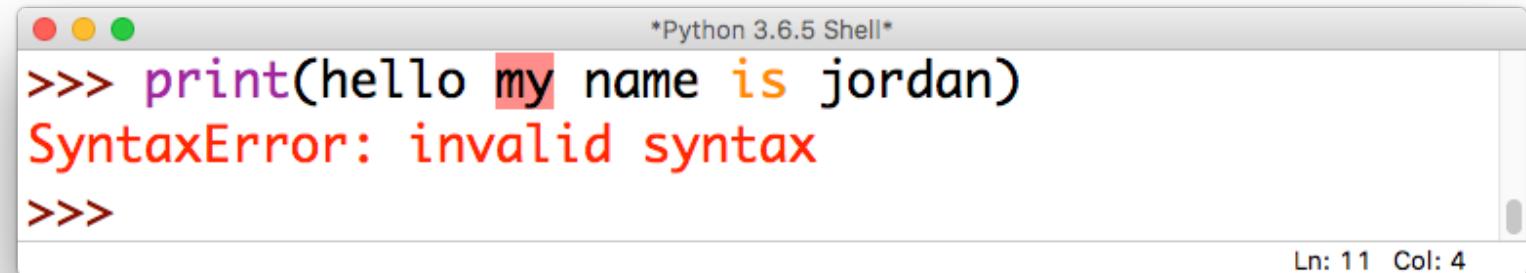
A screenshot of a Python 3.6.5 Shell window. The title bar says "Python 3.6.5 Shell". The code entered was:

```
>>> 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
```

The output shows a traceback and the resulting **IndexError: list index out of range** exception. The status bar at the bottom right indicates "Ln: 86 Col: 4".

Common Exceptions

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



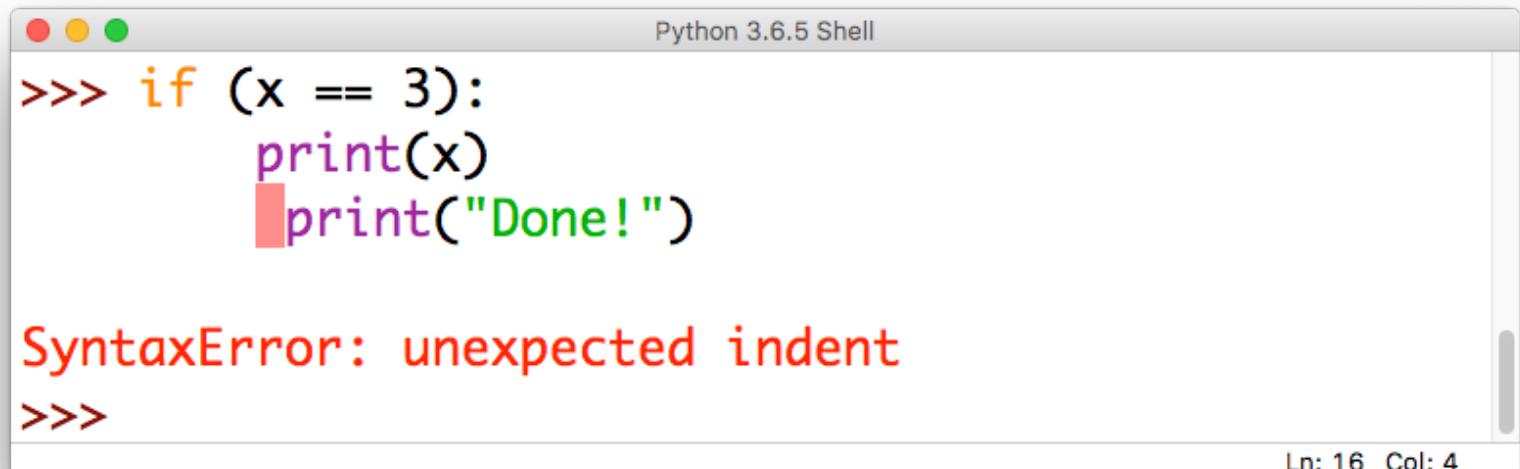
A screenshot of a Mac OS X window titled "*Python 3.6.5 Shell*". The window contains the following text:

```
>>> print(hello my name is jordan)
SyntaxError: invalid syntax
>>>
```

The word "hello" is highlighted in red, and the error message "SyntaxError: invalid syntax" is also in red. In the bottom right corner of the window, 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!")
```

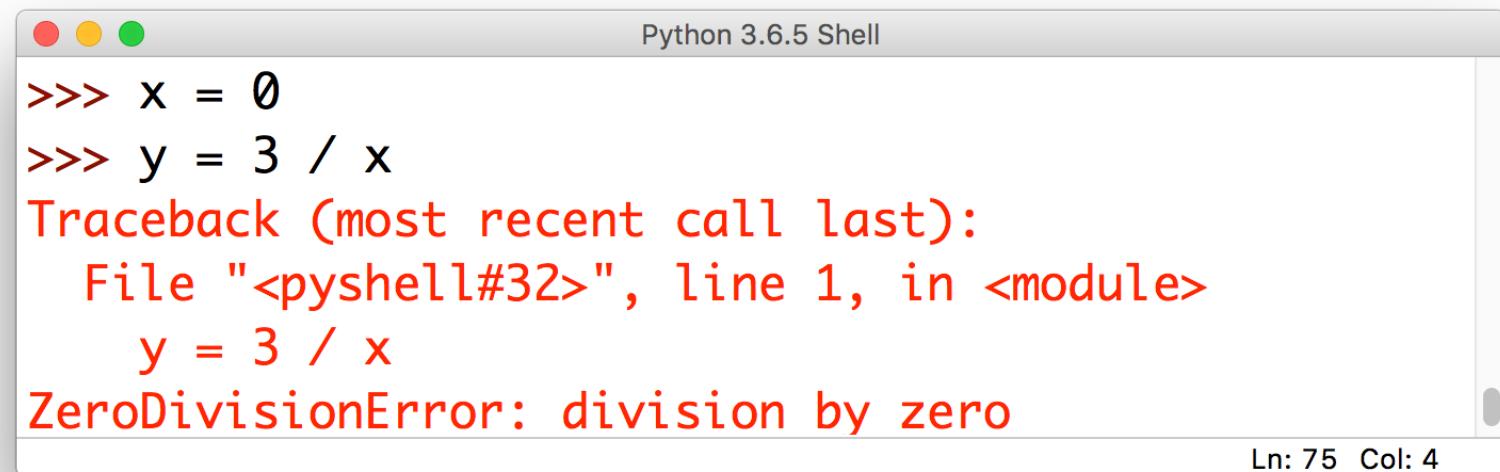
The output is:

SyntaxError: unexpected indent

Ln: 16 Col: 4

Common Exceptions

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



A screenshot of the Python 3.6.5 Shell window. The title bar says "Python 3.6.5 Shell". The code input area shows:

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

The output area shows a red traceback:

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

At the bottom right of the window, it says "Ln: 75 Col: 4".

Less common **Exceptions**

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

Look it up at:

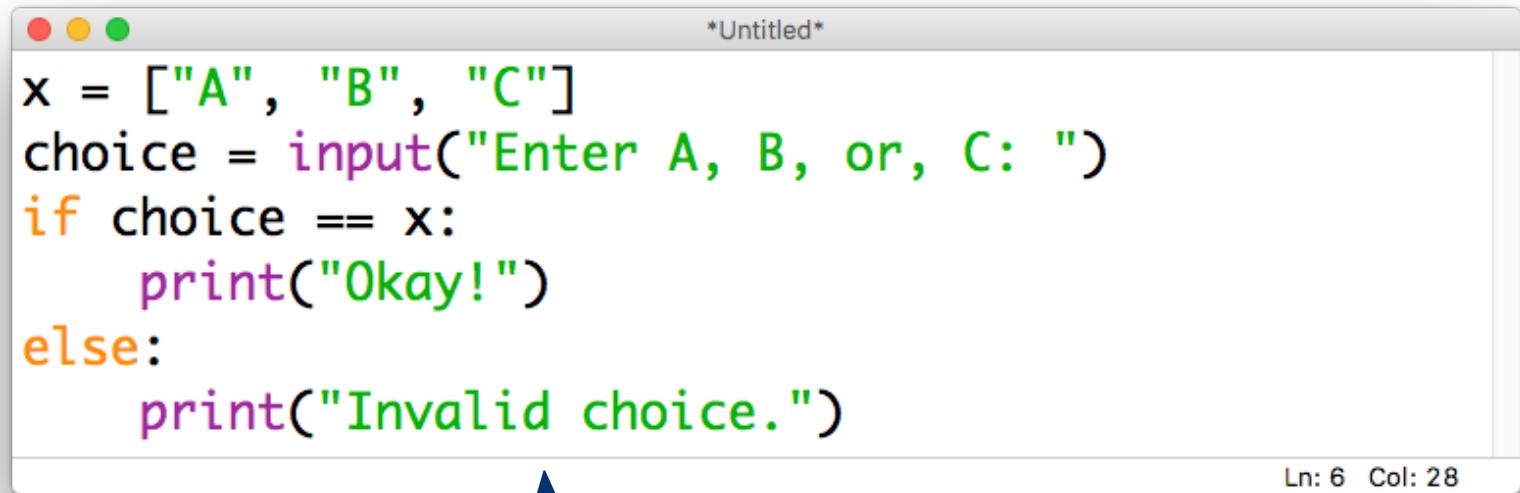
<https://docs.python.org/3/library/exceptions.html>

Exceptions
= relatively
easy to fix

Why would I say that?
What's the alternative?

Logical errors

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



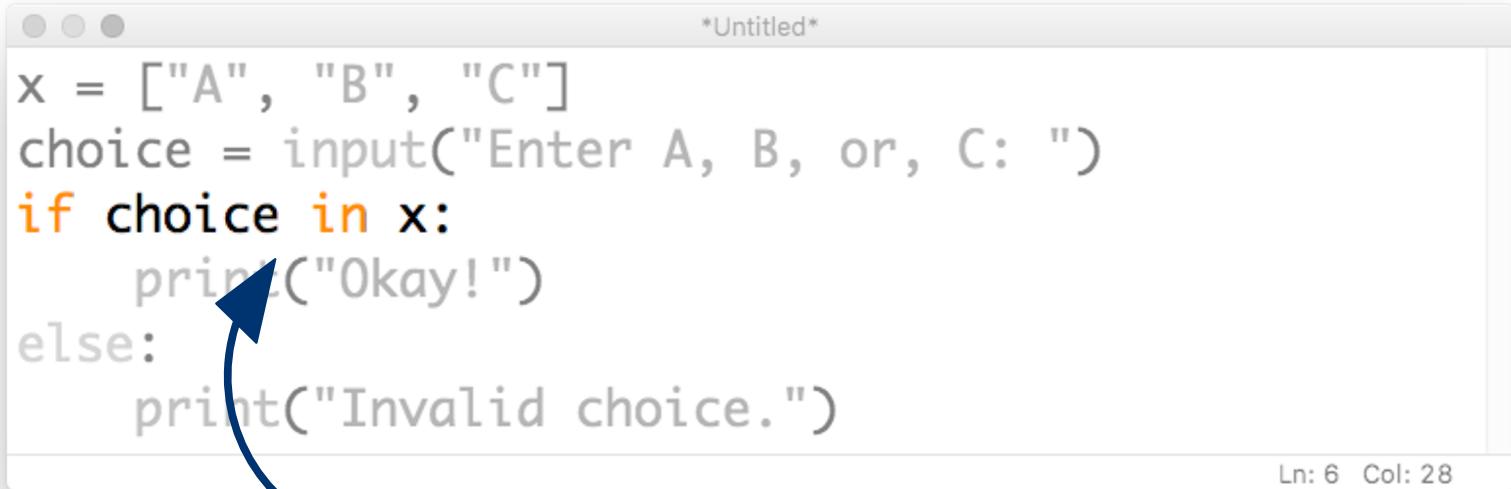
A screenshot of a Mac OS X-style window titled "Untitled". The window contains Python code. A blue curved arrow points from the text "(just not what we wanted)" to the line "if choice == x:". The status bar at the bottom right shows "Ln: 6 Col: 28".

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

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.



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 in x:
    print("Okay!")
else:
    print("Invalid choice.")
```

A blue curved arrow points from the text "what we were" down to the word "in" in the if statement, indicating a logical error in the reasoning behind the code.

Ln: 6 Col: 28

what we were
actually going for

An analogy

Syntactic Error

Their is no
reason to be
concerned.

Logical Error

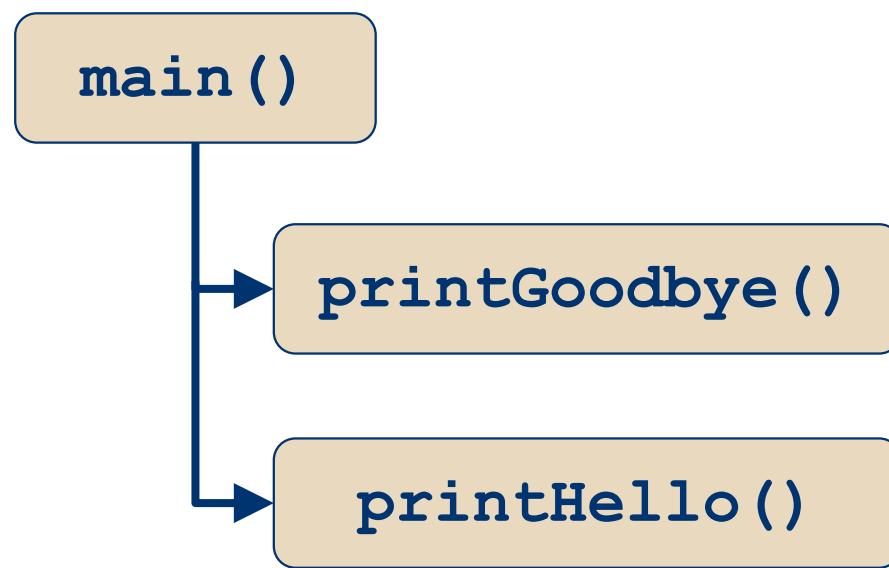
If an animal is
green, it must
be a frog.

Discussion

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

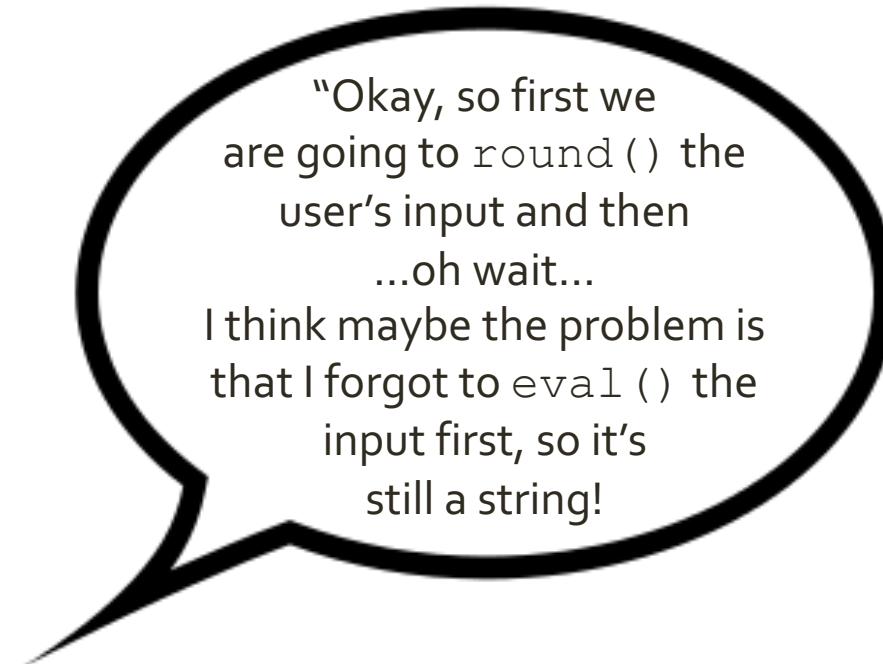
Step 1: map out the code

- 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:



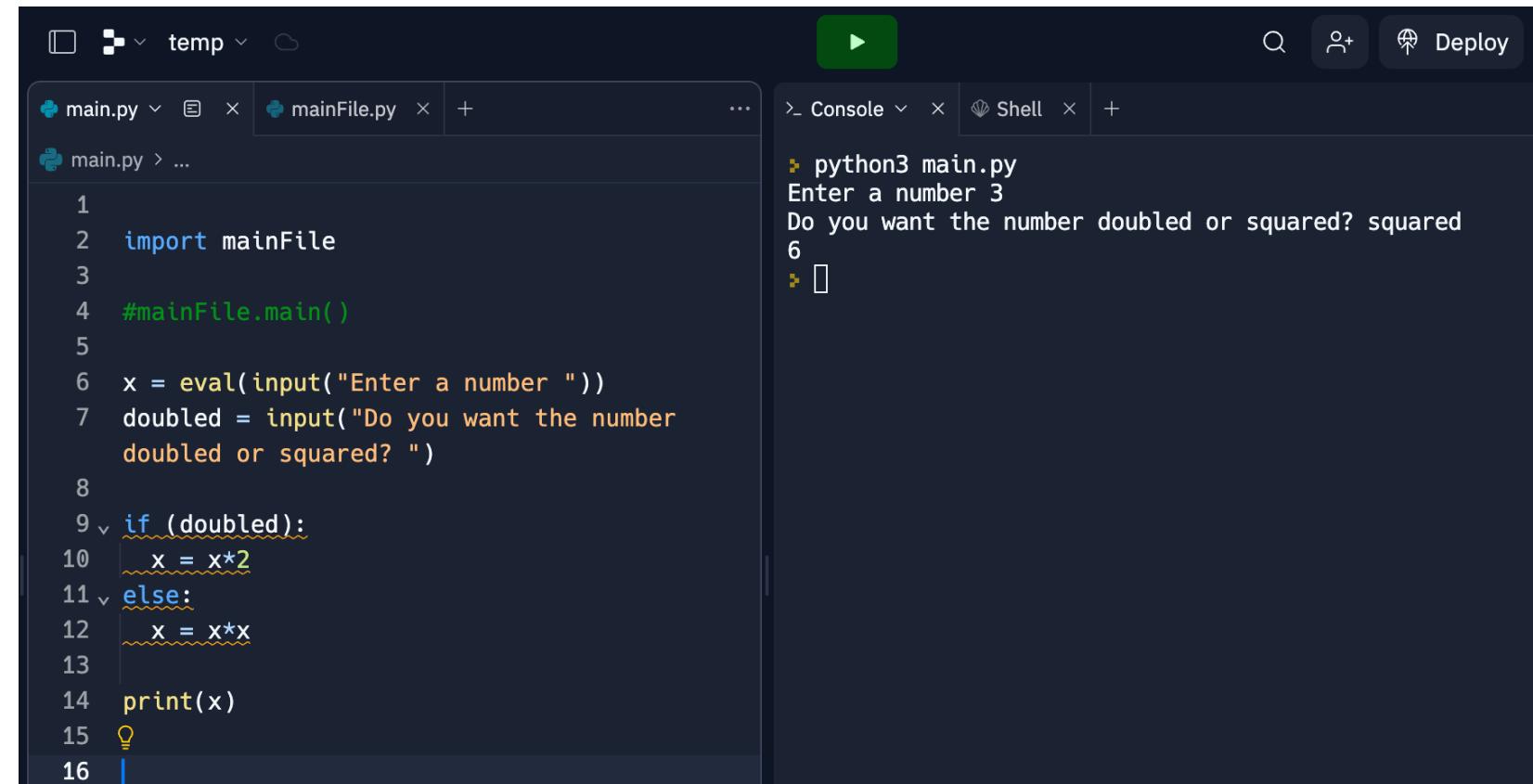
Step 2: “rubber ducking”

- 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?
- Add **print()** statements to leave a “trail” on the console



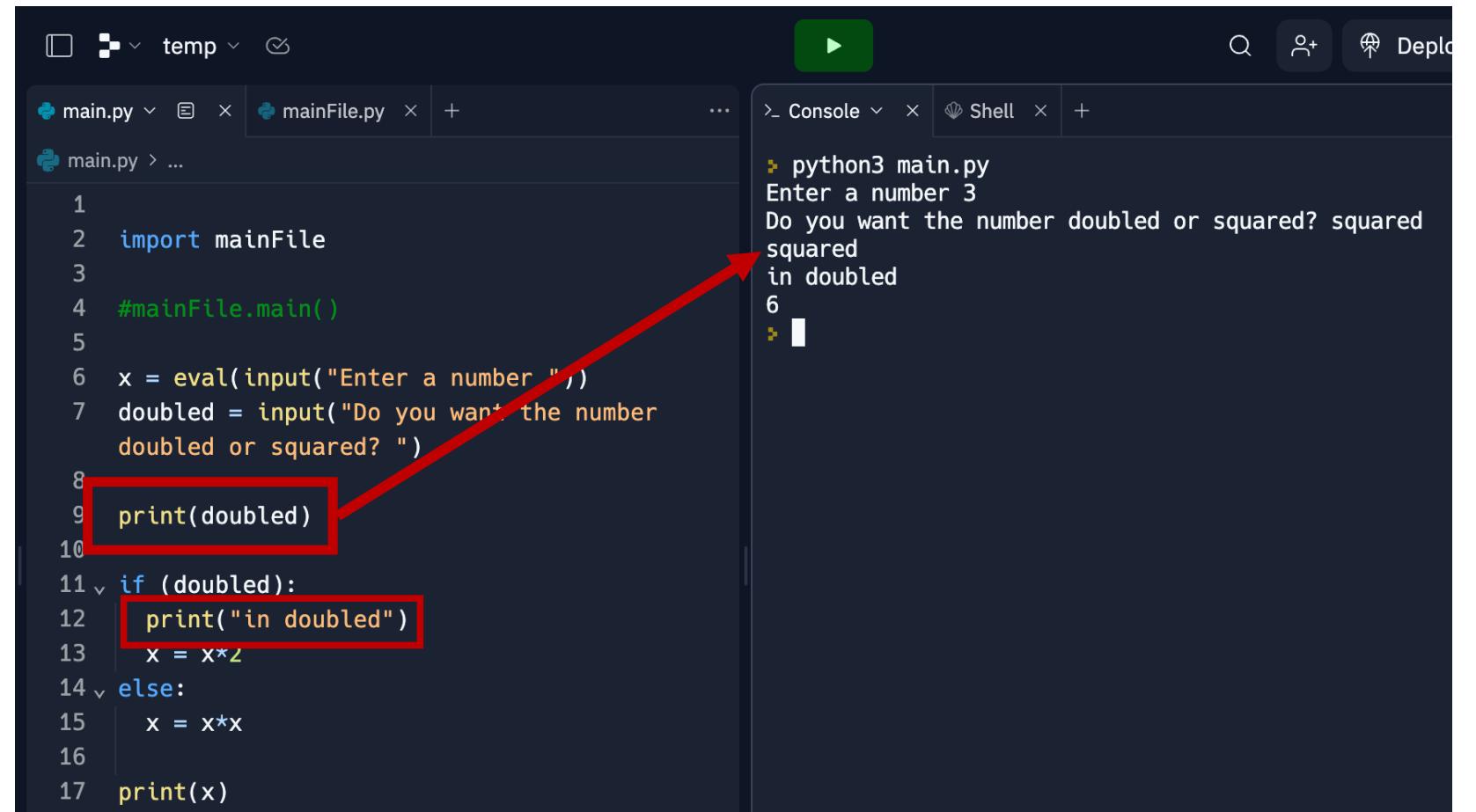
The screenshot shows a Python development environment with two tabs open: `main.py` and `mainFile.py`. The `main.py` tab contains the following code:

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

The `mainFile.py` tab is currently active. In the `Console` tab, the command `python3 main.py` is run, followed by user input `Enter a number 3`, and the question `Do you want the number doubled or squared? squared`. The user then types `6` and presses enter.

Step 3: add **print()** statements

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



The image shows a code editor interface with two tabs: `main.py` and `mainFile.py`. The `main.py` tab contains the following 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
doubled or squared? ")
8
9 print(doubled)
10
11 if (doubled):
12     print("in doubled")
13     x = x*x
14 else:
15     x = x*x
16
17 print(x)
```

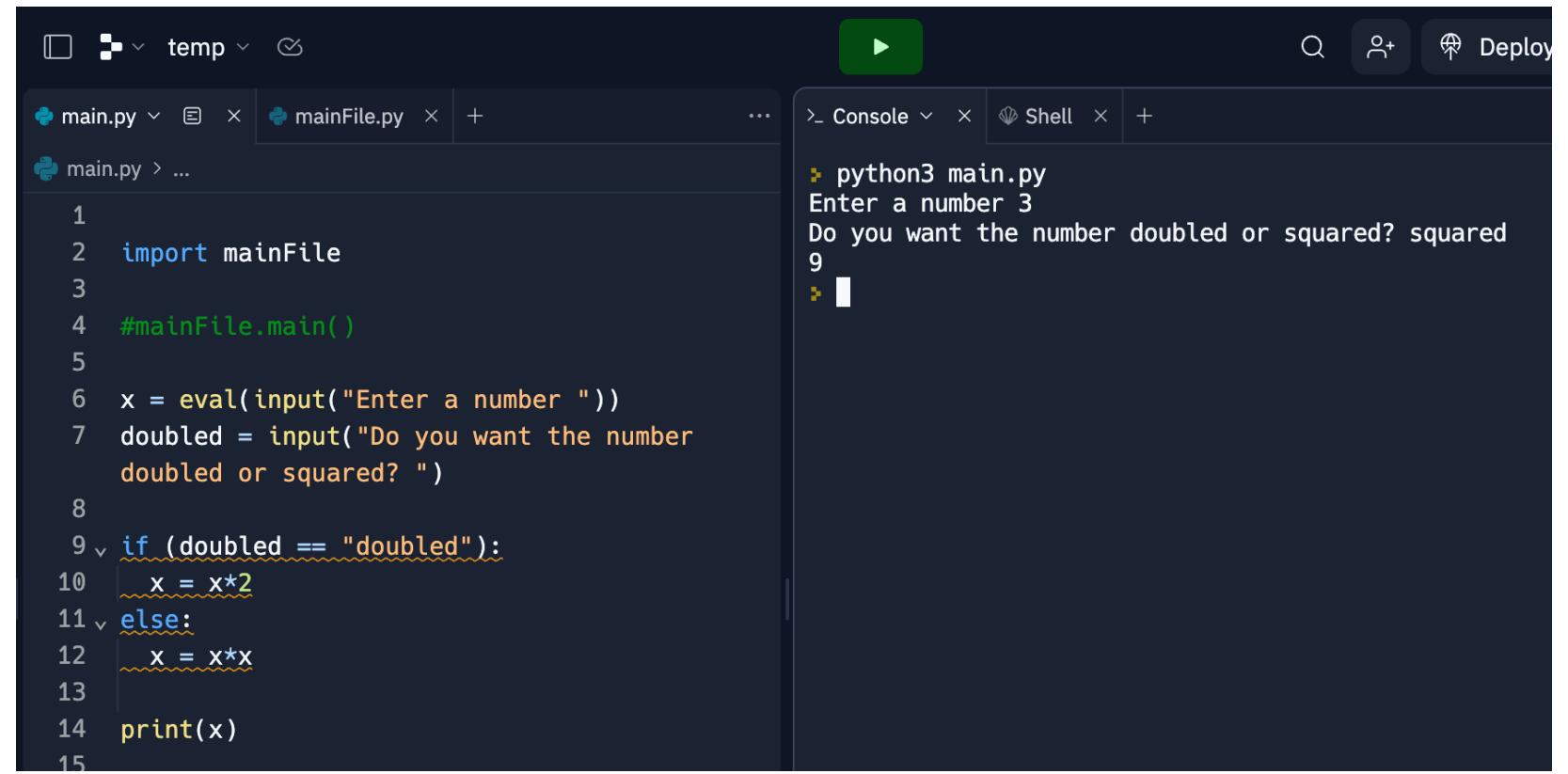
A red box highlights the line `print(doubled)`, and another red box highlights the line `print("in doubled")`. A red arrow points from the `print(doubled)` line to the corresponding output in the terminal window.

The terminal window on the right shows the command `python3 main.py` being run, followed by user input and the program's response:

```
> 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 image shows a code editor interface with two tabs: 'main.py' and 'mainFile.py'. The 'main.py' tab contains the following 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
doubled or squared? ")
8
9 if (doubled == "doubled"):
10     x = x*2
11 else:
12     x = x*x
13
14 print(x)
15
```

The 'mainFile.py' tab is currently not visible. To the right of the editor is a terminal window titled 'Console' showing the execution of the script:

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

Takeaways

- There are **lots** of other techniques for both dealing with and **preventing** bugs
- 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!

Activity: “code detective”

ic03

```
challenge2.py - /Users/jcrouser/Google Drive/Teaching/Course  
zz = []  
p1go = True  
w = False  
  
def su():  
    for x in range(10):  
        zz.append(str(x + 1))  
  
def pb():  
    print(' ' + zz[0])  
    print('---+---')
```

