Intro to Coding with Python– Documentation and Debugging

Dr. Ab Mosca (they/them)

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

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

```
x = "Ab Mosca"
y = "Professor"
z = "Westfield State"

print(x, "-", y, "at", z)
```

Line 5, Column 26 Spaces: 4 Python

Step 1: meaningful **nouns** for variables

temp.py

```
x = "Ab Mosca"
y = "Professor"
z = "Westfield State"

print(x, "-", y, "at", z)
```

Line 5, Column 26 Spaces: 4 Python

temp.py

```
name = "Ab Mosca"
title = "Professor"
institution = "Westfield State"

print(name, "-", title, "at", institution)
```

Line 6, Column 1 Spaces: 4 Python

Step 2: lots of comments

Untitled

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

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

Ln: 6 Col: 0

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

Untitled

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

Ln: 3 Col: 17

Untitled

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

Ln: 4 Col: 22

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()
```

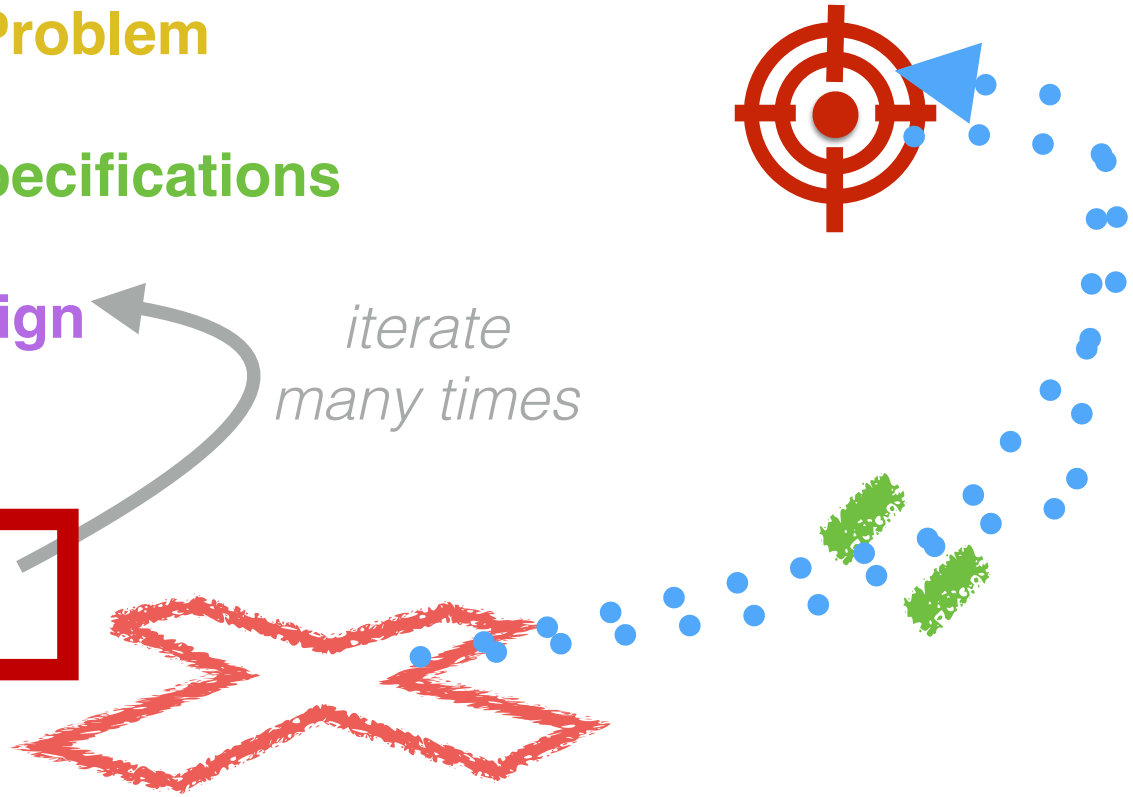
Ln: 5 Col: 0

Debugging

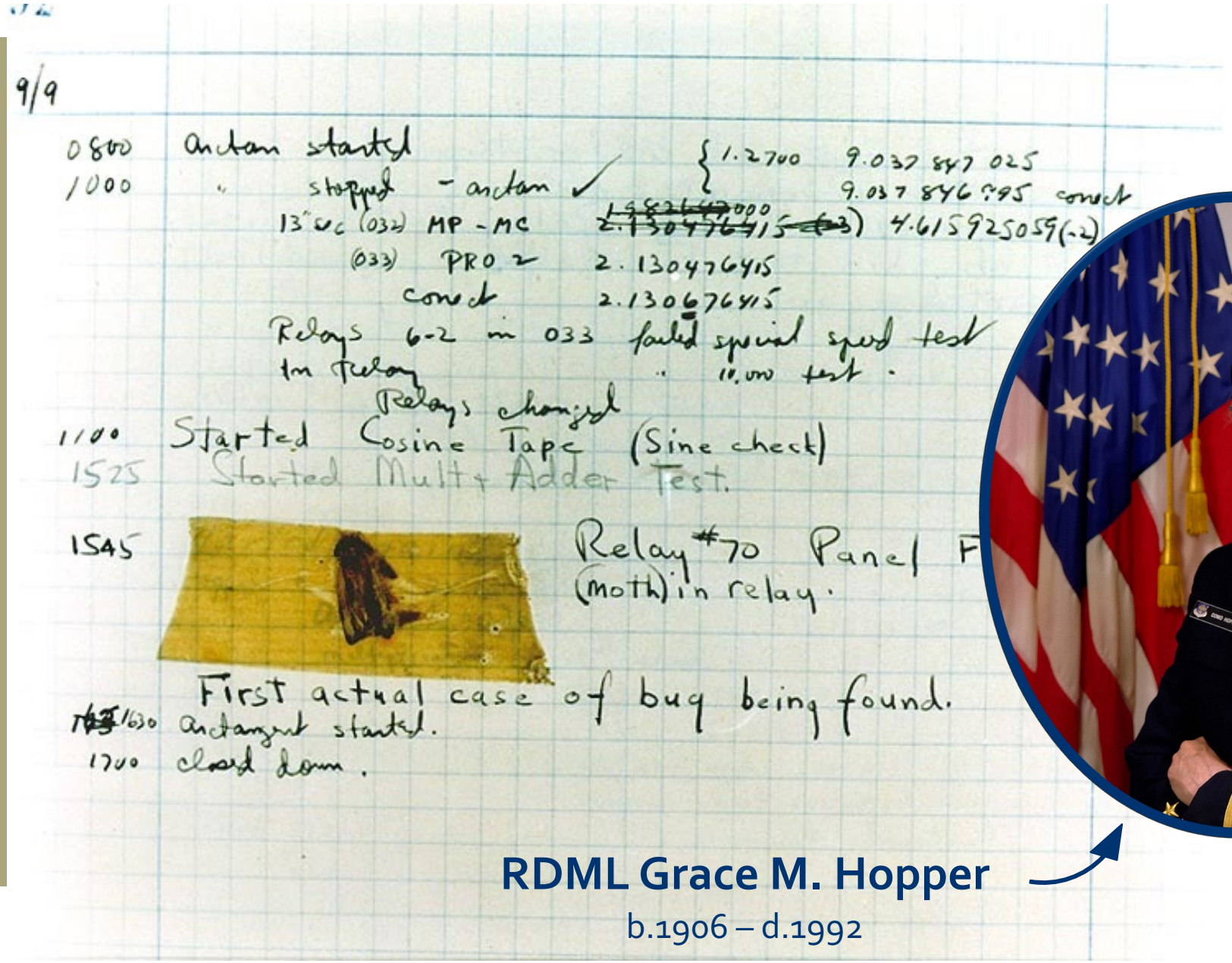
RECAP: the programming process

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

iterate many times



Fun history: the term "debug"



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

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

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)

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**

- **`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)

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

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

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

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

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.

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

Their is no reason to be concerned.

Logical Error

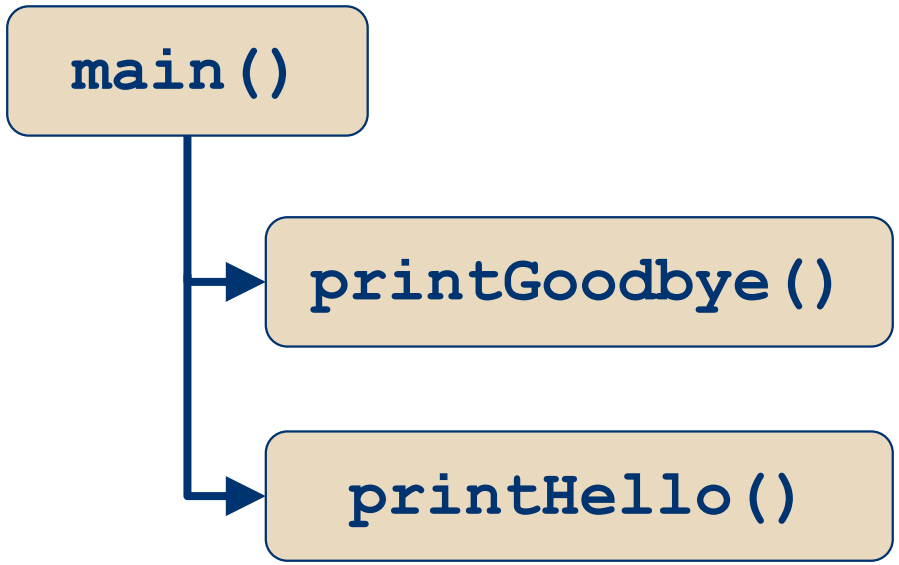
If an animal is green, it must be a frog.

Discussion

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

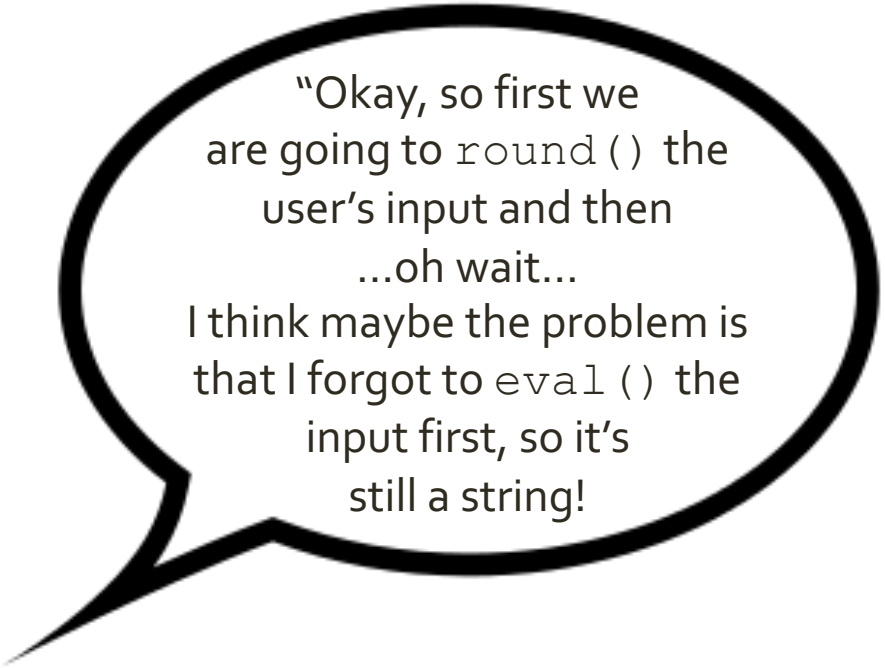
Step 1: map out the code

- 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:



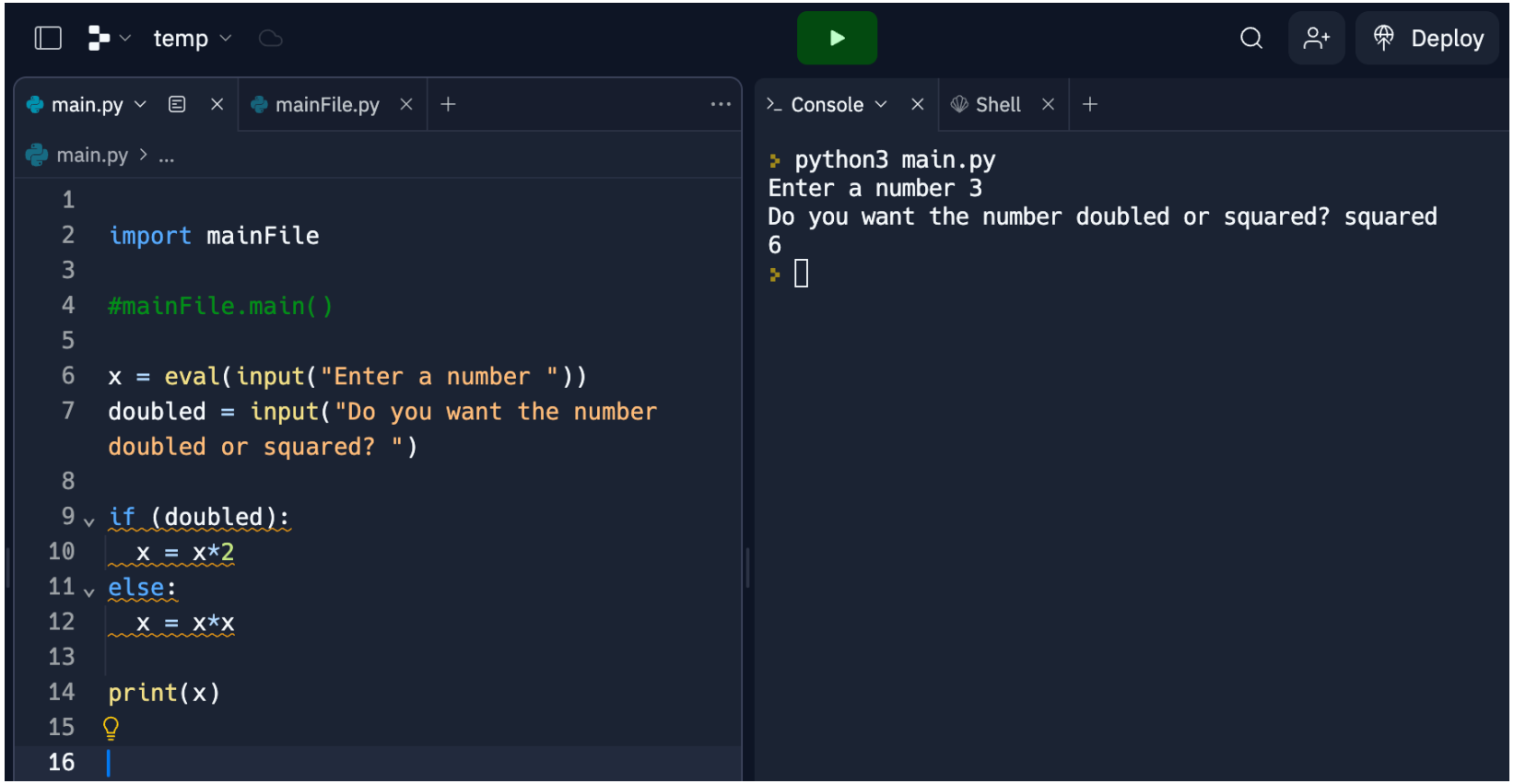
Step 2: "rubber ducking"

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



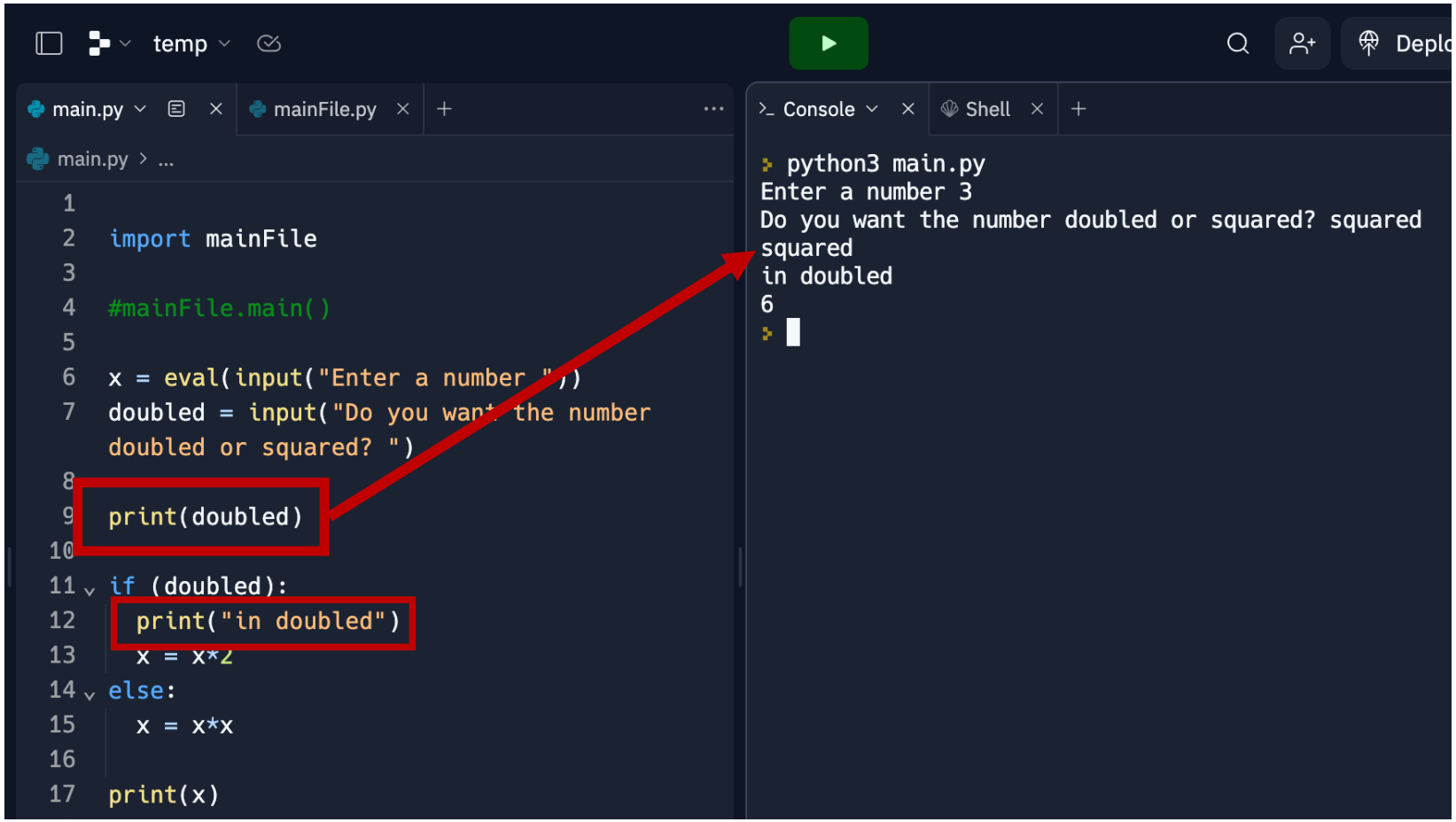
Step 3: add `print()` statements

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



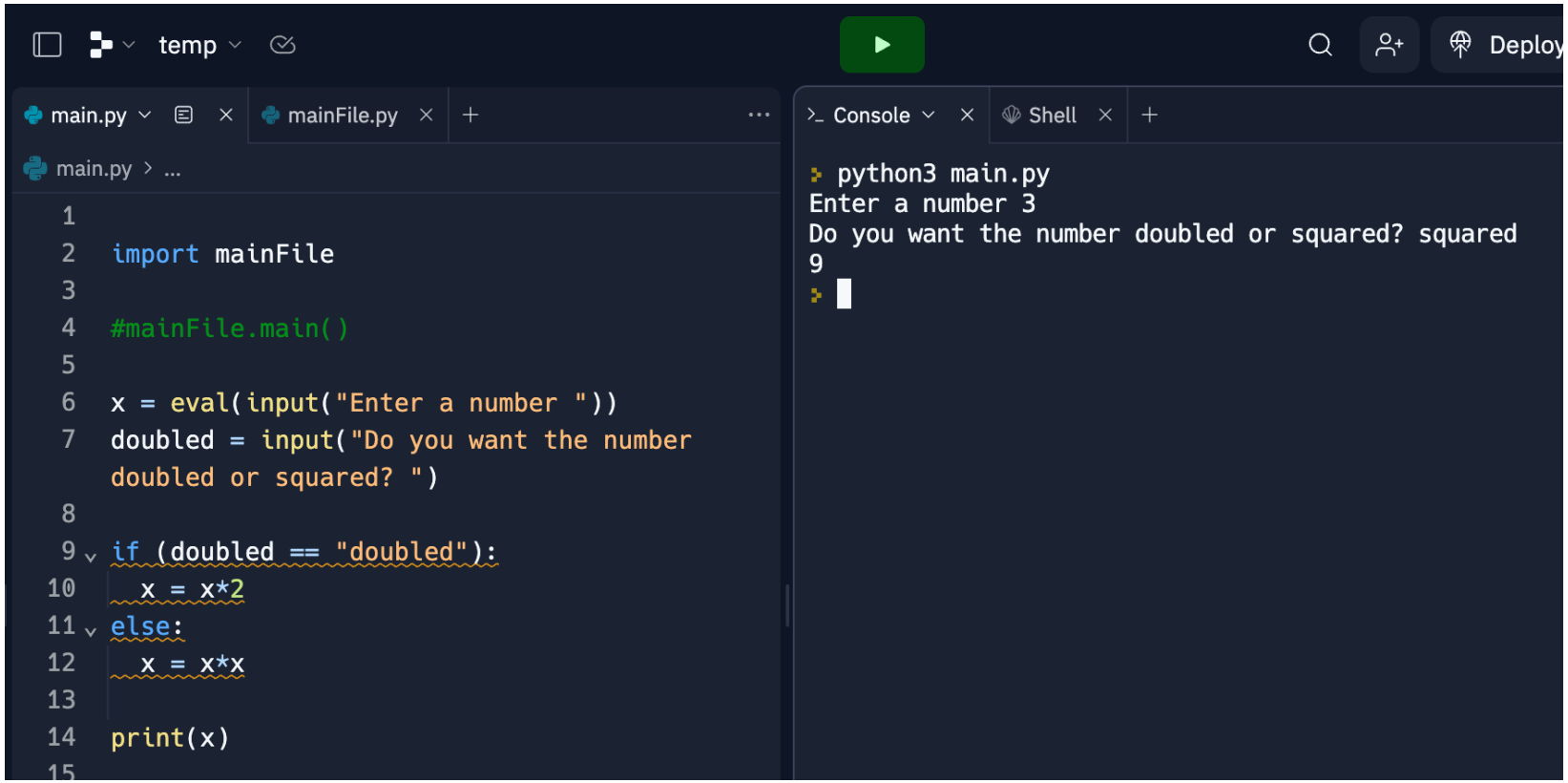
Step 3: add `print()` statements

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



Step 3: add `print()` statements

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



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!

ic03

Activity: “code detective”

challenge2.py - /Users/jcrouser/Google Drive/Teaching/Course ... /labs/challe...

```
zz = []
p1go = True
w = False

def su():
    for x in range(    ):
        zz.append(st    x + 1))

def pb():
    print( ' ' + zz[0]        zz
    print( '--+-+--')
```