Lecture 9: Debugging and Testing

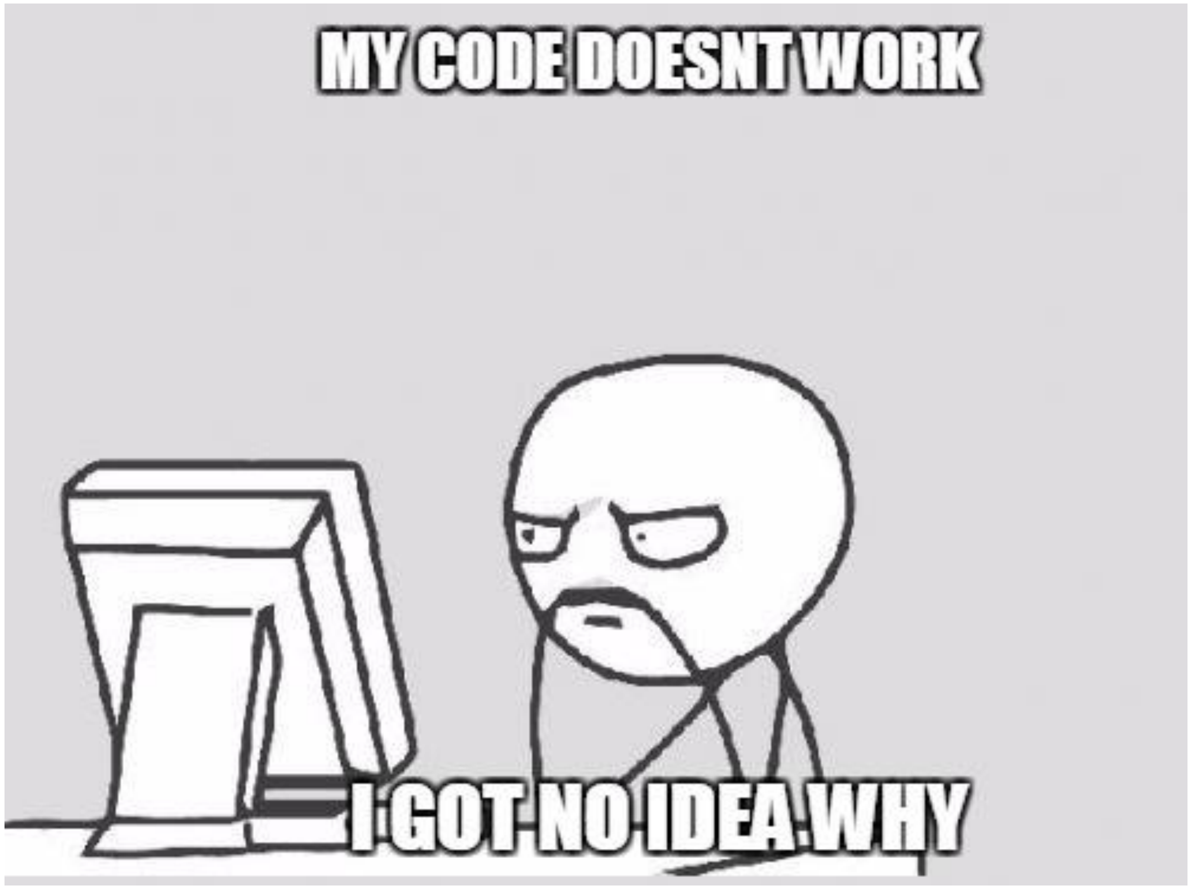
CS 50

February 17, 2026

Reminders

- Midterm is this Thursday (Feb. 19), in class
  - pen and paper exam
  - 75 minutes
  - covers everything through lists
  - one page of notes (double-sided) allowed

  - examples and exercises from class make good study materials
  - section 1 also has exercises posted on their page (“practice problems”)
  - textbook also has good practice problems (online, linked from website)

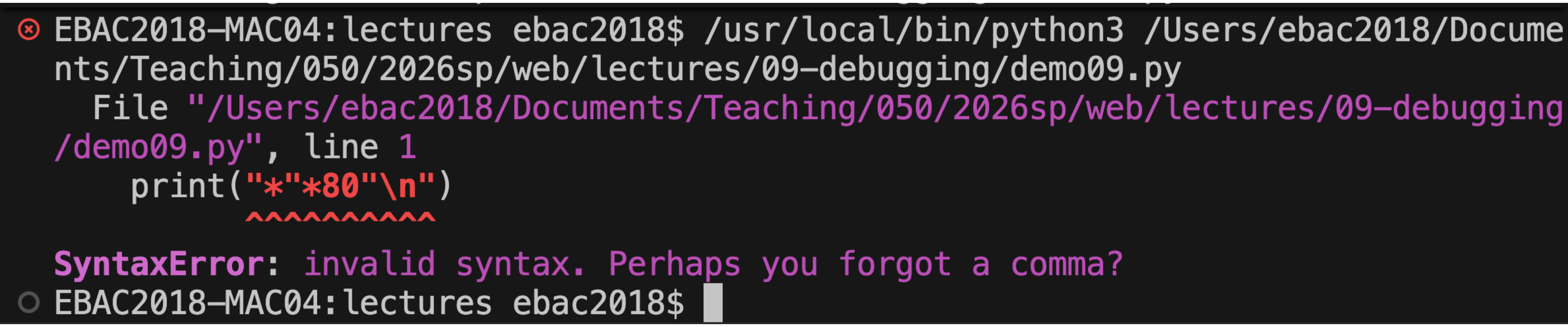

MY CODE DOESNT WORK
I GOT NO IDEA WHY

Common Types of Errors

- **Syntax Errors:** there is something wrong with the structure of the program, and Python doesn't understand it
- **Runtime Errors:** something goes wrong while the program is running
- **Semantic Errors:** the program runs, but it doesn't do what you want it to do

Handling Syntax Errors

1. Find the bug



```
⊗ EBAC2018-MAC04:lectures ebac2018$ /usr/local/bin/python3 /Users/ebac2018/Docume
  nts/Teaching/050/2026sp/web/lectures/09-debugging/demo09.py
    File "/Users/ebac2018/Documents/Teaching/050/2026sp/web/lectures/09-debugging
  /demo09.py", line 1
      print("*"*80"\n")
            ^^^^^^^^^^

  SyntaxError: invalid syntax. Perhaps you forgot a comma?
○ EBAC2018-MAC04:lectures ebac2018$ █
```

2. Do you see the problem?
   1. If yes, fix it!
   2. If no, try running through the list of common syntax errors
   3. If still no, check your class notes, discuss the problem abstractly with a friend ("what's the right syntax for…"), or ask a TA/instructor (it's ok to get help!)

Common Syntax Errors

- Misspelling a variable name or a function name
- Missing quotation marks around a string
- Mismatched parentheses or quotation marks
- Missing a colon at the end of an if/while/for statement
- Using = instead of ==
- Using a Python keyword as a variable name

Make sure you remembered to save your file
after making your changes!

Example 1: Syntax Errors

```
in = int(input("Pick a number\n"))                ← SyntaxError
if in = 13:                                       ← SyntaxError
    print("I am also fond of the number 13!")
elif in > 13:
    print("I am fond of the number 13, which is "
          + str(in-13) + " less than " + str(in)  ← SyntaxError
else                                              ← SyntaxError
    print("I am fond of the number 13, which is "
          + str(13-in) + " more than " + str(in)  ← SyntaxError
in2 = input("Do you like tea?)                    ← SyntaxError
while in2 != "yes" and != "no":                   ← SyntaxError
    in2 = input("Please answer yes or no. Do you like tea?")
if in2 == "yes":
    print("Great!")
else:
    print("That's too bad.")
print("Bye!)                                      ← SyntaxError
```

Can you find the
the **mistake**?

1 2 3 4 5 6 7 8 9

Handling Runtime Errors: Program Hangs

- You are probably in an infinite loop!
- Add print statements to figure out how far you got
- Add print statements to find line(s) that repeat over and over

- Your program might also just be waiting for an input

Handling Runtime Errors: Exceptions

- NameError: Python doesn't recognize a (variable) name
  - Find the bug!
  - Did you forget quotation marks around a string?
  - Did you misspell a variable name? Make a typo?
  - Is the variable you are trying to use in scope? Use before define?

Scope

```
def good_choice(num):
    b = (num == pomona_fav)
    return b

def check_num():
    pomona_fav = 47
    number = int(input())
    if good_choice(number):
      print("yay")
    else:
      print("boo")

hmc_fav = 42
check_num()
```

Storing a value in a variable:

1. If there is a variable with that name in the current function's stack frame, store the value in that variable
2. Otherwise create a new variable in the current function's stack frame and store the value there

Using a variable:

1. Check for a local variable with that name. If it exists, use the value stored in that variable
2. If not, check for a global variable with that name. If it exists, use the value stored in that variable
3. Otherwise get a NameError

Exercise 1: Variable Scope

```
def f2(s7):
    s7 = s7 + 'd' + s5
    print("f2:", s7)

def f1(s4):
    s1 = 'c’
    s5 = s2 + s4
    print("f1:", s1)
    print("f1:", s5)
    return s1+s5

s1 = 'a'
s2 = 'b'
print(s1)
s3 = f1(s1)
print(s1, s2, s3)
print(s5)
f2(s2)
```

Handling Runtime Errors: Exceptions

- NameError: Python doesn't recognize a (variable) name
  - Find the bug!
  - Did you forget quotation marks around a string?
  - Did you misspell a variable name? Make a typo?
  - Is the variable you are trying to use in scope? Use before define?

- TypeError: Python can't perform that operation/function on that type
  - Find the bug!
  - Are the types that the error reports the type you expected?
  - Add a print statement on the previous line and print out all the variables/values on that line. Are they what you expect?

- ValueError: Python can't perform that operation/function on that value
  - Find the bug!
  - Add a print statement on the previous line and print out all the variables/values on that line. Are they what you expect?

When your code runs…

Belvin's Law 3
YESSS!
If the code works the first time, that just means that the bug is more carefully hidden.
Gee, thanks for that.
© 2017 The Code Zone
Visit angriestprogrammer.com for your dose of cynicism

Example 2: Debugging buggy code

- Simple password checking (does it contain at least 8 chars?)

```
def example2(): # warning: this code contains a bug!!!
  valid = False
  char_count = 0

  while not valid:
    password = input("Please enter your password:\n")
    # count number of chars
    for c in password:
      char_count += 1

    # provide feedback
    if char_count < 8:
      print("Password should contain at least 8 characters")
    else:
      print("Password " + password + " is a valid password")
      valid = True
```

Handling Semantic Errors

- Add print statement in the appropriate places
- Print out the value of variables that you want to keep track of
- Compare the print out result with your expected result

Testing

- Try running your function with different values, called test cases, and make sure it returns the right value

- Branch Testing (white-box testing)
  - make sure that every line of code is executed by at least once
  - for conditionals, try include a test case that makes the condition evaluate to True and a test case that makes the condition evaluate to False
  - for loops, try to include test cases that make the program go through the loop 0 times, 1 time, and lots of times

- Corner-Case Testing (black-box testing)
  - include the "weird" values in your test cases
  - e.g., for ints, include negative numbers and zero, as well as positive
  - e.g., for strings, include the empty string

Testing in Python

- Create a new file called <program_name>_tester.py

- Import the functions you want to test

```
from demo09 import sum_even
```

- Using assert statements to test program behavior

```
assert <condition>
```

Example 3: Writing Test Cases

demo09.py

```
def sum_even(start, end):
  """
  Computes the sum of the even
  numbers between <start> and <end>
  (inclusive). Warning: buggy code!!

  :param start: (int) start of range
  :param end: (int) end of range

  :return: (int) sum of evens
  """

  for i in range(start, end):
    if i % 2 == 0:
      sum = i
```

demo09_tester.py

```
from demo09 import sum_even


def test_sum_even():
  assert type(sum_even(1,5)) == int
  assert sum_even(1,5) == 6
  assert sum_even(1,6) == 12
  assert sum_even(2,5) == 6
  assert sum_even(2,6) == 12
  assert sum_even(1,1) == 0
  assert sum_even(2,2) == 2



test_sum_even()
```

Exercise 2: Writing Test Cases

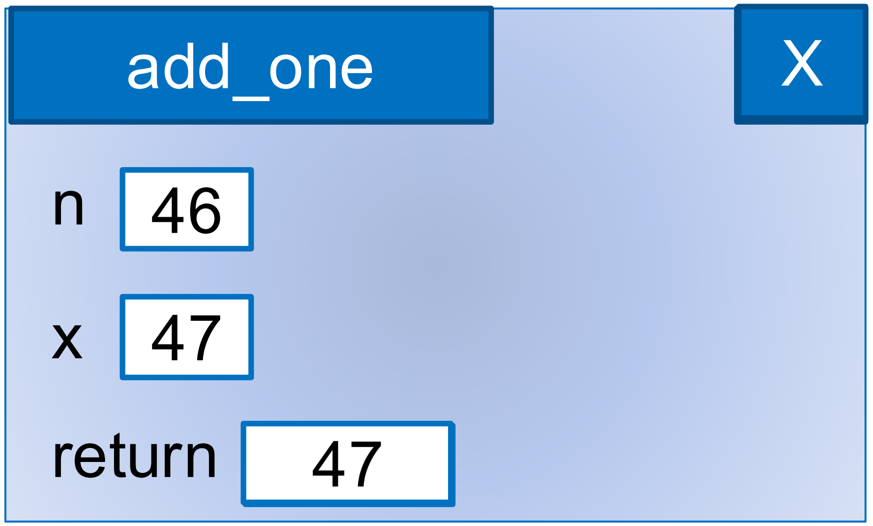
- Write a set of test cases for the following function

```
def square_neg2(lst):
    """
    Creates a new list of numbers where each element is
    squared if the original number is negative and
    otherwise is the same.

    :param lst (list): A list of numbers (ints or floats)

    :return (list): A new list similar to the original but
                    with the negatives squared
    """

  new_list = []

  for number in lst:
    if number < 0: # if negative
        new_list.append(number**2)
    else:
        new_list.append(number)

  return new_list
```

Code Tracing

- Execute the program line by line by hand

num = add_one(46)

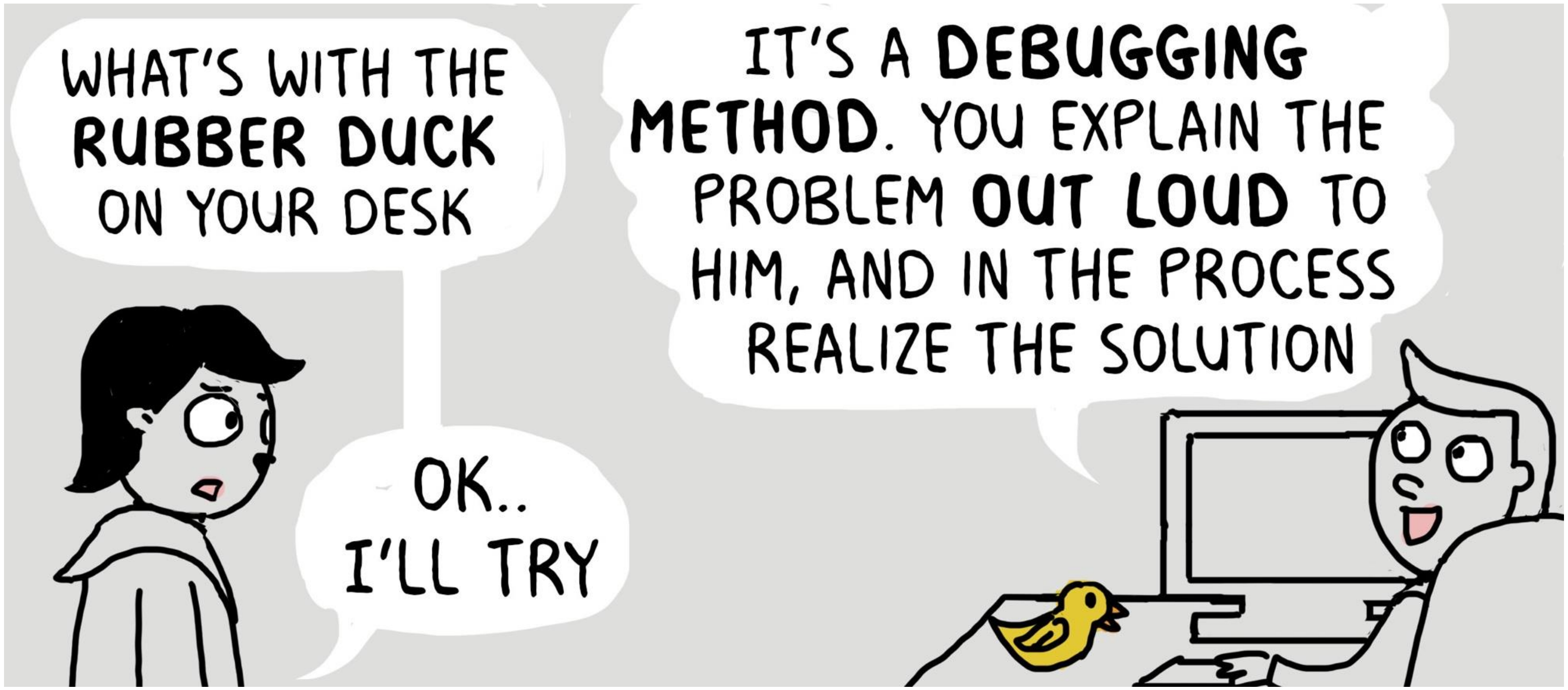


- If you get the right answer by hand, add print statements to determine where your code starts doing something different

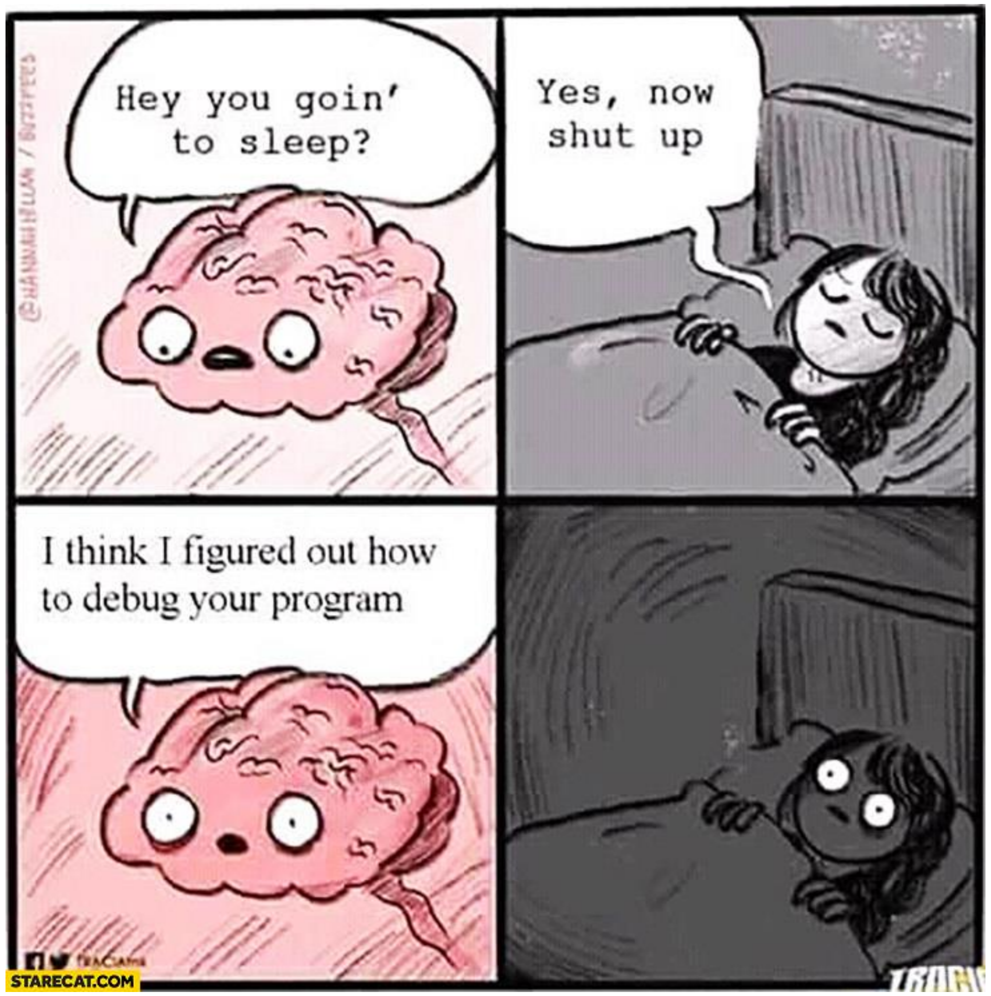
Exercise 3: Debugging Code

```
def example3(num1, num2): # WARNING: Buggy code!!!
    """
    Computes the sum of the even numbers between <start>
    and <end> (inclusive).
    :param start: (int) one end of range
    :param end: (int) other end of range
    :return: (int) sum of evens
    """
    sum_even = 0
    i = num1

    while i < num2:
        if i % 2 != 0: # if even
            sum_even = sum_even + i

        i = i + 1 # increment counter

        return sum_even # return sum of even numbers
```

Rubber-Duck Debugging


WHAT'S WITH THE **RUBBER DUCK** ON YOUR DESK
IT'S A **DEBUGGING METHOD**. YOU EXPLAIN THE PROBLEM **OUT LOUD** TO HIM, AND IN THE PROCESS REALIZE THE SOLUTION
OK.. I'LL TRY

Take a break


Hey you goin' to sleep?
Yes, now shut up
I think I figured out how to debug your program
STARECAT.COM

Debugging…

Debugging…

YOU HAVE DISHONORED ME FOR TOO LONG.
EVEN THOUGH FOR EVERY ONE OF YOU I STRIKE DOWN, ANOTHER TWO MAY TAKE HIS PLACE...
... I WILL KILL YOU ALL!
AAAAAAAAAAAAAAAA
DEBUGGING IS A STATE OF MIND.
WWW.INKTANK.COM
© 2000 BARRY T. SMITH