CS 1112: Introduction To Programming

Style & Debugging Python Code

Dr. Nada Basit // basit[*at*]Virginia[*dot*]edu

Spring 2024

Friendly Reminders

- Your ***safety*** and ***comfort*** is important!
  - If you choose to wear a mask you are welcome to do so
  - *We will interpret wearing a mask as being considerate and caring of others in the classroom (not that you are sick), and realize that some may choose to mask to remain distanced*
- Be an ***active*** participant in your learning!
  You're welcome and ***encouraged*** to ask questions during class!
- If you feel ***unwell***, or think you are, please stay home
  - *We will work with you!*
  - Get some rest ☺
  - View the recorded lectures – *please allow 24-48 hours to post*
  - *Contact us!*

Announcements / Reminders

- No new quiz this weekend!
  - Next quiz (Quiz 5, on Loops & Strings) will be due 3/18
- No programming assignment due next Wednesday!
  - PA04 (Loops) will be due 3/20
- No classes on 3/4, 3/6, 3/8 (Spring Break 😎)
- No office hours 3/2 → 3/10 (Spring Break 😎) ~ *both Prof and TA OHs*
- **Exam 1** next Wednesday, February 28 (Review session with TAs: TBD)
  - Closed-book, closed-notes, closed-PyCharm
  - Students with SDAC accommodations book through SDAC on 2/18

Debugging

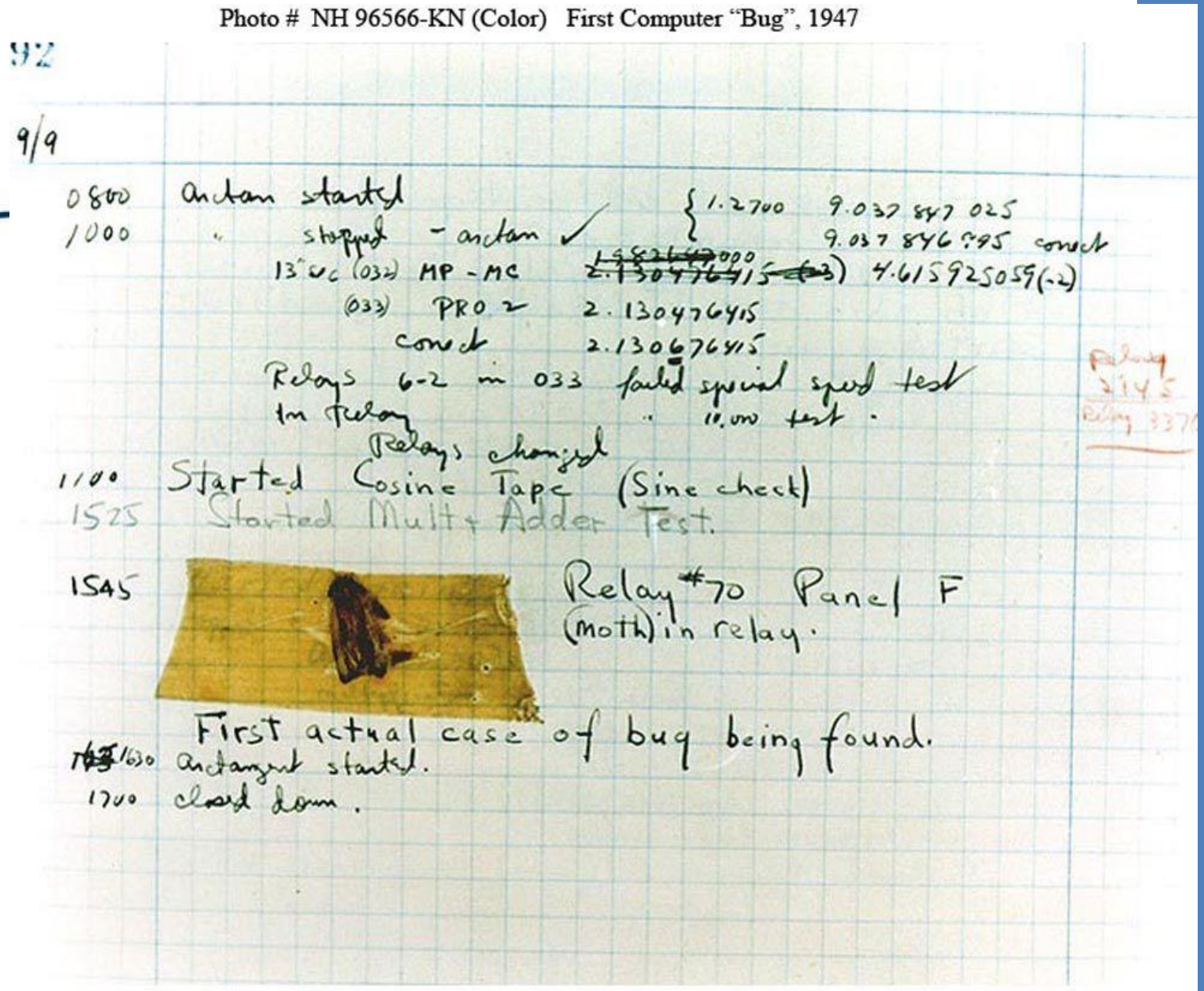
Take note of these strategies to improve your experience while debugging. Your debugging skills WILL improve with experience ☺

What *is* a "bug"?

- One person's definition:
- A flaw in a software program, or
- Programs that allows unintended behavior to occur, producing incorrect results or a crash.
- There are plenty of definitions
  - Vulnerabilities (cybersecurity)
  - Hardware bugs
- Can run from minor to very bad
  - The more sensitive/complex the software, the worse impact a bug can have
- Remember: existence of bugs doesn't mean you are a bad programmer!
  - If someone tells you that they can program *without* any bugs… don't believe them! ☺

What *is* a "bug"?

- On September 9, 1947, computer scientist **Grace Hopper** reported the world's first computer bug—a moth trapped in her computer at Harvard University.



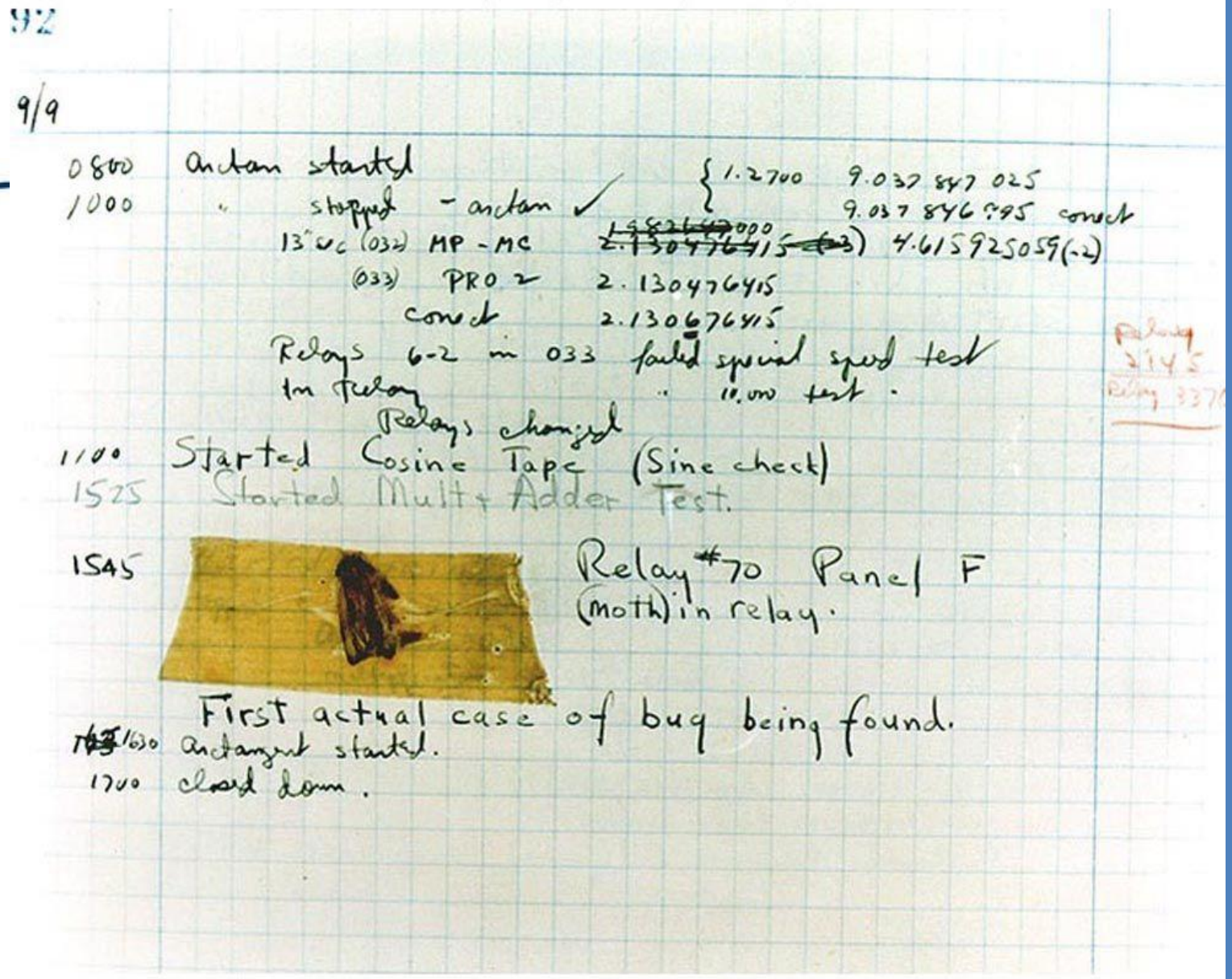
PHOTOGRAPH COURTESY NAVAL SURFACE WARFARE CENTER, DAHLGREN, VIRGINIA

What *is* a "bug"?

- We use the word "bug" to refer to:
  - Any error in the code (… likely not a moth!)
  - Anything your code did that didn't match your intention

- Sometimes debugging can take up more time in programming than actually writing the code!
  - *Therefore: Develop techniques to help!*

Photo # NH 96566-KN (Color) First Computer "Bug", 1947



PHOTOGRAPH COURTESY NAVAL SURFACE WARFARE CENTER, DAHLGREN, VIRGINIA

Errors/bugs

Here are 3 broad categories of errors that you could have in your program:

–**Syntax error:** code violates the rules of the programming language, and thus cannot be run

–**Runtime error:** code runs, but enters an illegal state or tries to do something impossible (such as int divided by zero).

–**Logical error:** this causes your program to operate incorrectly, but not crash. That is, syntax is correct, the code doesn't crash, but the output is incorrect. What you are *trying* to do isn't what happens.

Errors/bugs

Here are 3 broad categories of errors that you could have in your program:

–**Syntax error:** code violates the rules of the programming language, and thus cannot be run

–**Runtime error:** code runs, but enters an illegal state or tries to do something impossible (such as int divided by zero).

–**Logical error:** this causes your program to operate incorrectly, but not crash. That is, syntax is correct, the code doesn't crash, but the output is incorrect. What you are *trying* to do isn't what happens.

Hardest to locate

Errors/bugs

- Syntax Errors:
  - `def func()`
    - Without a colon at the end
- Runtime Errors:
  - Opening a file that doesn't exist
  - Other issues that cause problems when the code is actually run. Such as:

```
my_str = "hello"
my_str[0] = "H"
```

- Logical Errors:
  - Doesn't crash the program, but doesn't produce desired output or outcome
  - For example: incorrect loop bounds, incorrect operator (you added instead of subtracted)
  - HARDEST to debug!!

Quick & Fun Survey Questions

Get to know your peers! ☺

Likes Debugging vs Doesn't Like Debugging ☺

A Method for Finding Bugs

1. **Know** something broke
   a. Know what to expect should happen
      i. Easy: we expected no error
      ii. Difficult: we expected the program to do a certain thing
   b. Know what it actually did
      i. Easy: we had an error
      ii. Difficult: the program did the wrong thing
2. **Localize** the bug
   a. Find some line where the bug hasn't happened yet
      i. You can always start with the first line
   b. Find some line after the bug has occurred
      i. If you have an error, the bug must have occurred before that error
   c. Between those two lines is the bad line
3. **Print** out the value of variables, looking for the unexpected.

Common Debugging Techniques

- Print statements
  - In loops: [iteration] [value]
  - Variables, generally: see "live" program state
  - Breaking up chunks of code with these print statements (formally, "unit testing")
- "Don't go on autopilot". Read and interact with your code to see how it's working on a closer level.
  - Read what is there literally, not what *should* be there
  - "Rubber duck" debugging – explaining your code to another person, an animal, or an inanimate object
  - Pencil and paper, hand-trace program flow and variables

Common Debugging Techniques

- Using a debugger (e.g. the PyCharm debugger)
- Test cases: check intended output as an “answer key” against actual output
  - Try correct input that should produce a correct output
  - Try incorrect input that should produce an incorrect output
  - Try input that should produce an exception
  - Don’t forget edge cases, boundary cases, one-offs, strange cases…


Learn From Your TAs!
TAs, what is your favorite debugging strategy or technique?

Style in Python Programming

Essential for code ***readability***, ***consistency***, and ***clarity***

(Which in turns leads to **easier** *code maintenance* and debugging)

Human Language

- How do we expect to read human language?

- How do formatting and style impact our ability to **read**?
- How do formatting and style affect our ability to **catch errors**?

Comparing English Sentences

I like using Linux a lot.

I originally learned Linux for a scientific computing class. I started with Ubuntu Linux and ran that on my Windows laptop with the Windows Subsystem for Linux (WSL) tool.

Today, I use Linux all the time. It allows you to customize whatever you want however you want, and as a result I enjoy using it much more than Windows.

i like using linux a lot i originally learned linux for a scientific computing class i started with ubuntu linux and ran that on my windows laptop with the windows subsystem for linux (wsl) tool today i use linux all the time it allows you to customize whatever you want however you want and as a result i enjoy using it much more than windows

ilikeusinglinuxalotioriginallylearnedlinuxforascientificcomputingclassistartedwithubuntulinuxandranthatonmywindowslaptopwiththewindowssubsystemforlinux(wsl)tooltodayiuselinuxallthetimeitallowsyoutocustomizewhateveryouwanthoweveryouwantandasaresultienjoyusingitmuchmorethanwindows

Example thanks to Paul Karhnak, previous TA

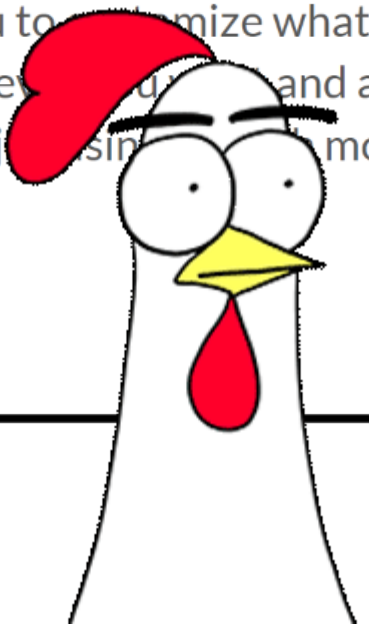
Comparing English Sentences

I like using Linux a lot.

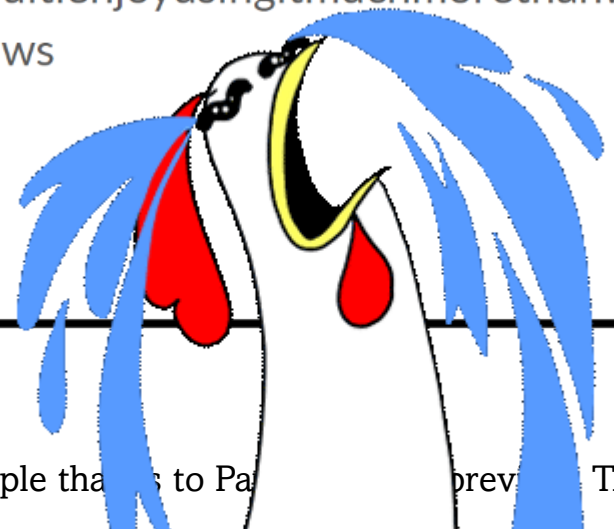
I originally learned Linux for a scientific computing class. I started with Ubuntu Linux and ran that on my Windows laptop with the Windows Subsystem for Linux (WSL) tool.

Today, I use Linux all the time. It allows you to customize whatever you want however you want and as a result I enjoy using it much more than Windows.

i like using linux a lot i originally learned linux for a scientific computing class i started with ubuntu linux and ran that on my windows laptop with the windows subsystem for linux (wsl) tool today i use linux all the time it allows you to customize whatever you want however you want and as a result i enjoy using it much more than windows



ilikeusinglinuxalotioriginallylearnedlinuxforascientificcomputingclassistartedwithubuntulinuxandranthatonmywindowslaptopwiththewindowssubsystemforlinux(wsl)tooltodayiuselinuxallthetimeitallowsyoutocustomizewhatevryouwanthoweveryouwantandasaresultienjoyusingitmuchmorethanwindows



Example tha[illegible]s to Pa[illegible]prev[illegible] TA

Activity: What's wrong with this text?

My favorite programing language is easily C. C has relatively explict syntax and you do not have to make many gueses about your code is running on actual hardwre. C is low-level, which I like, but I recognize some do not. I have writtn C for a scientific computing clss, computer science classes (CS 2130 and CS 3130), and an embeded systems class. Truly, it all comes back to C, and I would mind writing C for the rest of my life.

my favorite programing language is easily c c has relatively explict syntax and you do not have to make many gueses about your code is running on actual hardwre c is low-level which i like but i recognize some do not i have writtn c for a scientific computing clss computer science classes cs 2130 and cs 3130 and an embeded systems class truly it all comes back to c and i would mind writing c for the rest of my life

myfavoriteprograminglanguageiseasilycchasrelativelyexplictsyntaxandyoudonothavetomakemanyguesesaboutyourcodeisrunningonactualhardwrecislowlevelwhichilikebutirecognizethatsomedonotihavewrittncforascientificcomputingclasscomputerscienceclassescs2130andcs3130andanembededsystemclasstrulyitallcomesbacktocandiwouldmindwritingcfortherestofmylife

Example thanks to Paul Karhnak, previous TA

Activity: What's wrong with this text?

My favorite **programing** language is easily C. C has relatively **explict** syntax and you do not have to make many **gueses** about **your** code is running on actual **hardwre**. C is low-level, which I like, but I recognize some do not. I have **writtn** C for a scientific computing **clss**, computer science classes (CS 2130 and CS 3130), and an **embeded** systems class. Truly, it all comes back to C, and I **would mind** writing C for the rest of my life.

my favorite **programing** language is easily c c has relatively **explict** syntax and you do not have to make many **gueses** about **your** code is running on actual **hardwre** c is low-level which i like but i recognize some do not i have **writtn** c for a scientific computing **clss** computer science classes cs 2130 and cs 3130 and an **embeded** systems class truly it all comes back to c and i **would mind** writing c for the rest of my life

myfavorite**programing**languageiseasil ycchasrelatively**explict**syntaxandyoud onothavetomakemany**gueses**about**yo ur**codeisrunningonactual**hardwre**cislo wlevelwhichilikebutirecognizethatso medonotihavewrittncforascientificco mputing**clss**computerscienceclassescs 2130andcs3130andan**embeded**syste mclasstrulyitallcomesbacktocandi**wou ldmind**writingcfortherestofmylife

It is much easier to diagnose and fix (debug) the errors on the left-hand side than in the middle box and on the right. (Here: Syntax errors=misspellings)

Example thanks to Paul Karhnak, previous TA

Which Would You Rather Debug?

```
def is_ar():
    r = input()
    a = 4 * 3.14 * r * r
    print(a)
```

```
def interactive_sphere_area():
    radius = input("Please \
enter a radius: ")
    area = \
(4)*(3.14159)*(radius ** 2)
    print("The area of your \
sphere is:", area)
```

Style – PEP8

- Using good style makes code more readable
- **PEP8** is a general set of style guidelines/suggestions
  - https://www.python.org/dev/peps/pep-0008/
- These are recommendations for style. We will try to use these when we can
- We also have our document on "aesthetics" which covers many similar style guidelines and programming conventions

Style – Docstring comments

- Consistent style – similar information captured for each function
  - Making it easier to understand and read

```
def func(first_param, second_param):
    '''
    This is a Python docstring for a function.
    First line - Description of the function - what it does
    :param first_param: what is it's data type,
        what does this parameter represent in the context of the function
    :param second_param: what is it's data type,
        what does this parameter represent in the context of the function
    :return: what is the data type of the item being returned,
        what does the item returned represent
    '''
```

Indentation consistency around if-statements

Syntax of Python If-Else:

```
if (condition):
    # Executes this block if
    # condition is true
else:
    # Executes this block if
    # condition is false
```

Syntax:

```
if (condition1):
   # Executes when condition1 is true
   if (condition2):
      # Executes when condition2 is true
   # if Block is end here
# if Block is end here
```

Indentation consistency around loops

```
for var in iterable:
    # statements
```

```
while expression:
    statement(s)
```

Code Layout – spaces and indentation

```
# Correct:

# Aligned with opening delimiter.
foo = long_function_name(var_one, var_two,
                         var_three, var_four)

# Add 4 spaces (an extra level of indentation) to distinguish arguments from the rest.
def long_function_name(
        var_one, var_two, var_three,
        var_four):
    print(var_one)

# Hanging indents should add a level.
foo = long_function_name(
    var_one, var_two,
    var_three, var_four)
```

```
# Wrong:

# Arguments on first line forbidden when not using vertical alignment.
foo = long_function_name(var_one, var_two,
    var_three, var_four)

# Further indentation required as indentation is not distinguishable.
def long_function_name(
    var_one, var_two, var_three,
    var_four):
    print(var_one)
```

Code Layout – spaces and indentation

```
# No extra indentation.
if (this_is_one_thing and
    that_is_another_thing):
    do_something()
```

```
# Add some extra indentation on the conditional continuation line.
if (this_is_one_thing
        and that_is_another_thing):
    do_something()
```

Line Break Before or After a Binary Operator?

```
# Wrong:
# operators sit far away from their operands
income = (gross_wages +
          taxable_interest +
          (dividends - qualified_dividends) -
          ira_deduction -
          student_loan_interest)
```

```
# Correct:
# easy to match operators with operands
income = (gross_wages
          + taxable_interest
          + (dividends - qualified_dividends)
          - ira_deduction
          - student_loan_interest)
```

Whitespace in Expressions and Statements

Avoid extraneous whitespace in the following situations:

- Immediately inside parentheses, brackets or braces:

```
# Correct:
spam(ham[1], {eggs: 2})
```

```
# Wrong:
spam( ham[ 1 ], { eggs: 2 } )
```

- Between a trailing comma and a following close parenthesis:

```
# Correct:
foo = (0,)
```

```
# Wrong:
bar = (0, )
```

- Immediately before a comma, semicolon, or colon:

```
# Correct:
if x == 4: print(x, y); x, y = y, x
```

```
# Wrong:
if x == 4 : print(x , y) ; x , y = y , x
```

Whitespace in Expressions and Statements

- Immediately before the open parenthesis that starts the argument list of a function call:

```
# Correct:
spam(1)
```

```
# Wrong:
spam (1)
```

- Immediately before the open parenthesis that starts an indexing or slicing:

```
# Correct:
dct['key'] = lst[index]
```

```
# Wrong:
dct ['key'] = lst [index]
```

Whitespace in Expressions and Statements

- More than one space around an assignment (or other) operator to align it with another:

```
# Correct:
x = 1
y = 2
long_variable = 3
```

```
# Wrong:
x             = 1
y             = 2
long_variable = 3
```

Other Recommendations

- If operators with **different priorities** are used, consider adding whitespace around the operators with the lowest priority(ies). Use your own judgment; however, never use more than one space, and always have the same amount of whitespace on both sides of a binary operator:

```
# Correct:
i = i + 1
submitted += 1
x = x*2 - 1
hypot2 = x*x + y*y
c = (a+b) * (a-b)
```

```
# Wrong:
i=i+1
submitted +=1
x = x * 2 - 1
hypot2 = x * x + y * y
c = (a + b) * (a - b)
```

Comments – PEP8

- Recommendations on comments
- https://peps.python.org/pep-0008/#comments

```
# Write a function that takes two lists of integers
# and determines if the average of the first list
# is higher than the average of the second list

def first_list_is_bigger(list1, list2):
    '''
    Returns True if the average of the first list is
    bigger than the average of the second list
    '''
    avg1 = 0
    avg2 = 0
    total = 0

    for each in list1:
        total += each
    avg1 = total/len(list1)

    for each in list2:
        total += each
    avg2 = total/len(list1)

    if avg1 > avg2:
        return True
    else:
        return False

a = [2,4]     # List 1
b = [1,2,3]   # List 2
print(first_list_is_bigger(a,b))
```

In-Class "lab" Activity!

- In pairs or groups up to three work on the following activity.
- **Use debugging techniques to catch the error in this code!**
- *Discuss strategies with your classmates*

Remember to check-in with a TA before leaving class today!

Copy this code into PyCharm then add print statements or other things …

Reminder: CS Laptop Loaner Program

- This course requires students to have a **laptop**
- I realize that not everybody might have one (nor necessarily need one for their desired major / path…)
- If you do not have a laptop for any reason… ***not to worry!***
- The CS department's Systems staff has a notebook / laptop loaner program and will be able to loan you a notebook / laptop computer for the duration of the semester if you don't have one or if you cannot afford one.
  - Also available if your laptop is broken and under repair, we can arrange for you to receive a loaner laptop for a week or two until your own laptop is fixed

Interested? Link: https://www.cs.virginia.edu/wiki/doku.php?id=cs_laptop_loaner

I am happy to be your sponsor. Please let me know.