APS106

UNIVERSITY OF TORONTO

debugging.

Week 2 | Lecture 3 (3.2)

While waiting for class to start:

Download and open the Jupyter Notebook (.ipynb) for Lecture 2.3.2

You may also use this lecture's JupyterHub link instead (although opening it locally is encouraged).

Upcoming (Today!):

- Reflection 2 released Friday @ 11 AM
- Lab 3 released Friday @ 11 AM
- Lab 2 **deadline** this Friday @ 11 PM
- PRA (Lab) on Friday @ 2PM this week (ONLINE)

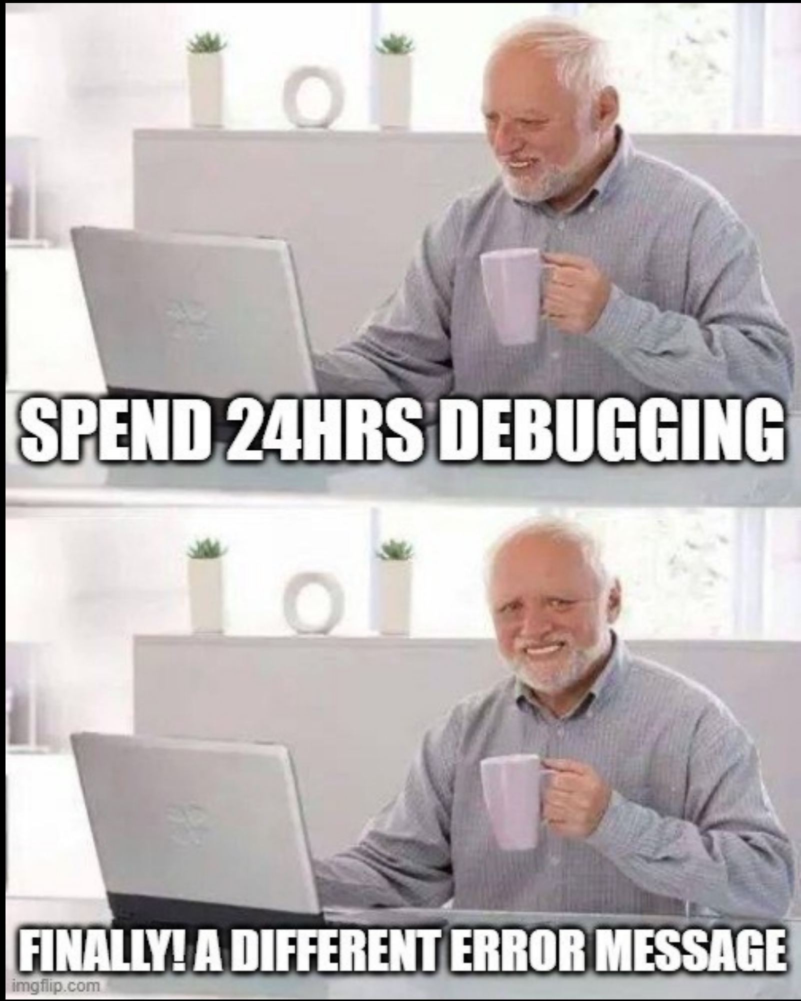
if nothing else, write #cleancode

Today's Content

- Lecture 3.2.1
  - More While Loops
- **Lecture 3.2.2**
  - **Debugging**

Error Reduction vs Debugging

- It is pretty much impossible to write code without errors.
  - *<u>Error Reduction</u>*: techniques we can use to reduce the number and severity of errors.
    - Write Readable Code
    - Comment comment comment!
    - Test test test!
  - *<u>Debugging</u>*: techniques for identifying and correcting errors



Readability Tips (#cleancode)

`>>> canda = cat + panda`

- Use whitespace to separate variables and operators
  - `>>> canda=cat+panda`
- Be consistent with spacing, too much whitespace can be bad
  - `>>> canda =                          cat      +panda`
- Pick variable names that are easy to read and interpret
  - `>>> canda = nom + nomnomnomnomnom`
- Be consistent with naming schemes
  - `>>> Canda = CAT + _panda42`

Write readable code.

- Use whitespace to separate variables and operators.

Bad

```
x=(1+3/2-4)*2-3**2
```

Good

```
x = (1 + 3 / 2 - 4) * 2 - 3**2
```

Write readable code.

- Be consistent with spacing, too much whitespace can be a bad thing.

Bad

```
x=      (1+3    /    2-4) *2-3** 2
```

Good

```
x = (1 + 3 / 2 - 4) * 2 - 3**2
```

Variable Names and Conventions

- The rules for legal Python names:
  - Names must start with a letter or _ (underscore)
  - Names must contain only letters, digits, and _
- In most situations, the convention is to use pothole_case
  - Lowercase letters with words separated by _ to improve readability
- Try to add meaning where possible!
  - Ex: gas_mileage and cost_per_litre instead of nomnom and nomnomnom
  - Save yourself when debugging & put your TAs in a good mood when marking

Write readable code.

- Pick variable names that are easy to read and interpret.

Bad

```
na = 20*12/2
fah = 100*9/5+32
```

Good

```
normalized_area = 20 * 12 / 2
degrees_fahrenheit = 100 * 9 / 5 + 32
```

Write readable code.

- Try to be consistent with your naming schemes, for variables, functions, etc.

Bad

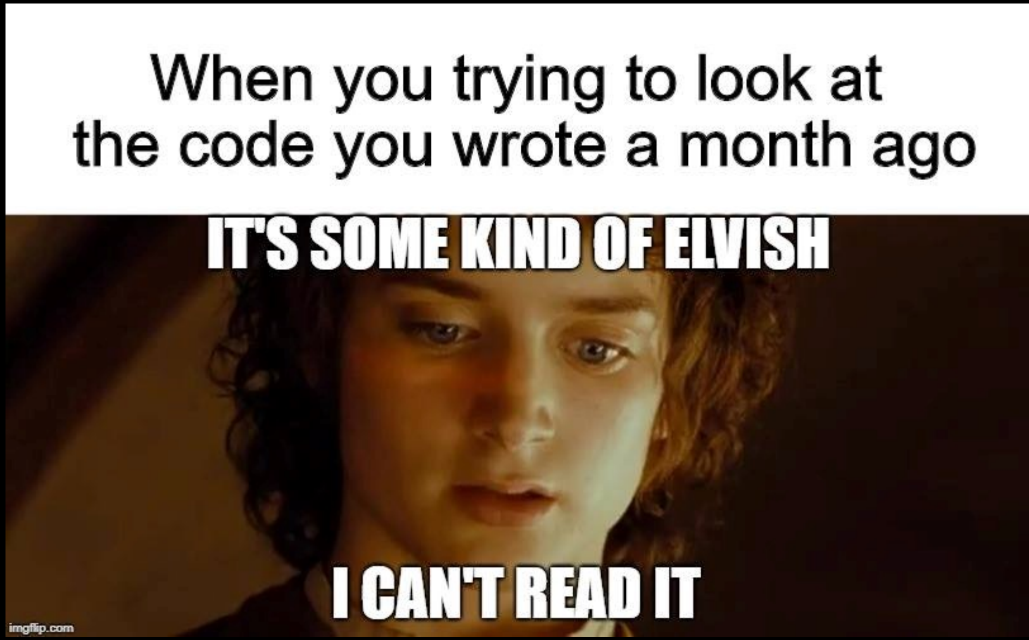
```
NormalizedArea = 20 * 12 / 2
degrees_fahrenheit = 100 * 9 / 5 + 32
```

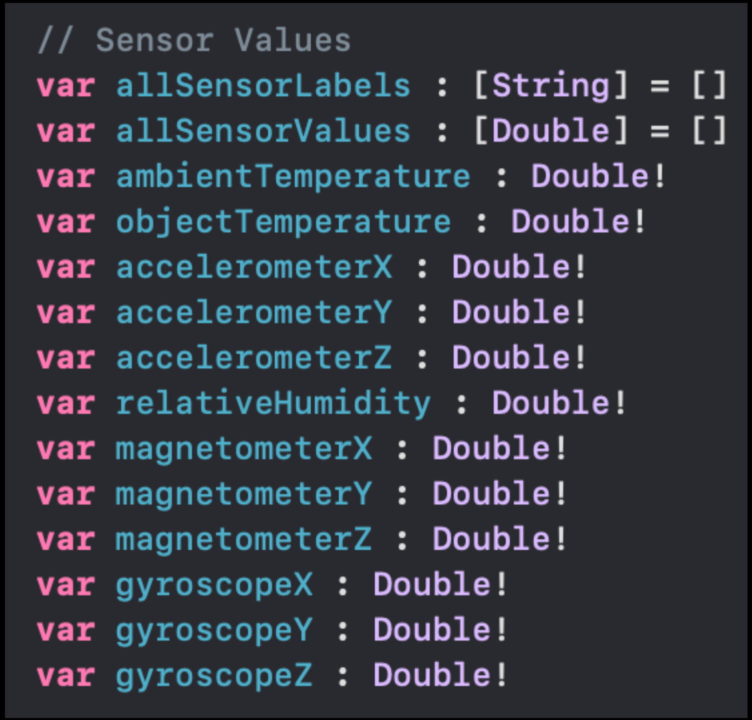
Good

```
normalized_area = 20 * 12 / 2
degrees_fahrenheit = 100 * 9 / 5 + 32
```

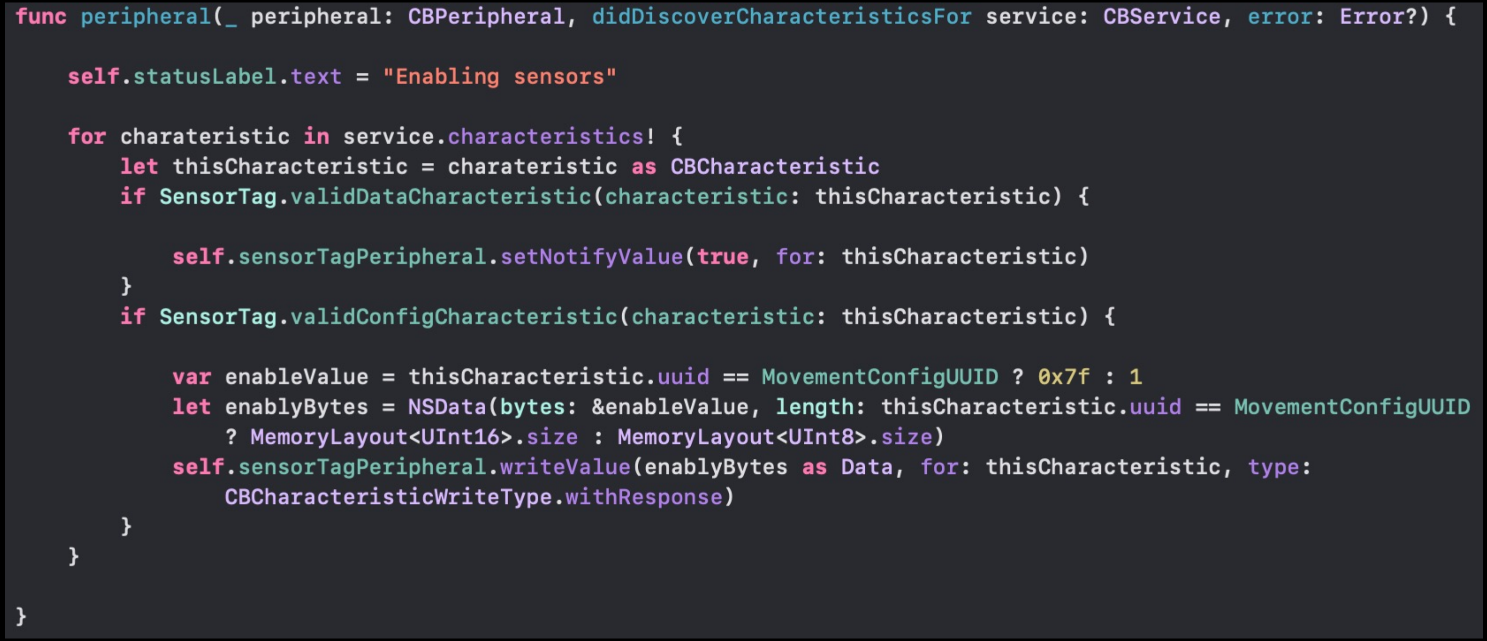
Comments

- Comments are to help you, and anyone else who is reading/using your code, to remember or understand the purpose of a given variable or function in a program.
- A comment begins with the number sign (#) and goes until the end of the line.
- Python ignores any lines that start with the (#) character





```
// Sensor Values
var allSensorLabels : [String] = []
var allSensorValues : [Double] = []
var ambientTemperature : Double!
var objectTemperature : Double!
var accelerometerX : Double!
var accelerometerY : Double!
var accelerometerZ : Double!
var relativeHumidity : Double!
var magnetometerX : Double!
var magnetometerY : Double!
var magnetometerZ : Double!
var gyroscopeX : Double!
var gyroscopeY : Double!
var gyroscopeZ : Double!
```



```
func peripheral(_ peripheral: CBPeripheral, didDiscoverCharacteristicsFor service: CBService, error: Error?) {

    self.statusLabel.text = "Enabling sensors"

    for charateristic in service.characteristics! {
        let thisCharacteristic = charateristic as CBCharacteristic
        if SensorTag.validDataCharacteristic(characteristic: thisCharacteristic) {

            self.sensorTagPeripheral.setNotifyValue(true, for: thisCharacteristic)
        }
        if SensorTag.validConfigCharacteristic(characteristic: thisCharacteristic) {

            var enableValue = thisCharacteristic.uuid == MovementConfigUUID ? 0x7f : 1
            let enablyBytes = NSData(bytes: &enableValue, length: thisCharacteristic.uuid == MovementConfigUUID
                ? MemoryLayout<UInt16>.size : MemoryLayout<UInt8>.size)
            self.sensorTagPeripheral.writeValue(enablyBytes as Data, for: thisCharacteristic, type:
                CBCharacteristicWriteType.withResponse)
        }
    }

}
```

Warning! This is not Python! It is an example from one of my iOS apps I had to come back to after a few years. Comments are (//) in Swift instead of (#) in Python

Comment often.

Bad

```
fahrenheit = 212
Celsius = (fahrenheit - 32) * 5 / 9
base = 20
height = 12
area = base * height / 2
```

Comment often.

Good

```
# Convert degrees Fahrenheit to Celsius
fahrenheit = 212
Celsius = (fahrenheit - 32) * 5 / 9

# Calculate the area of a triangle
base = 20
height = 12
area = base * height / 2
```

Comments

Space

Testing!

- The more lines of code you write, the more likely it is that you will make a mistake and the harder it will be to find the mistake
  - “like finding a needle in a haystack”
- Test your code as you write it
  - Requires you understanding what specific output an input will provide
- ”Modular code”
  - Test in small chunks or “modules”
  - Put a test input into the beginning where you know what the output is and see what you get!

Golden Rule: Never spend more than 15 minutes programming without testing

Test, test, test.

- Don't try writing this all in one shot.
- How many times do you read over an essay? An email? A text? A TWEET?

```
exam_one = int(input("Input exam grade one: "))

exam_two = input("Input exam grade two: "))

exam_3 = str(input("Input exam grade three: "))

sum = exam_one + exam_two + exam_3

avg = sum / 3

if avg >= 90:
    letter_grade = "A"
elif avg >= 80 and avg < 90
    letter_grade = "B"
elif avg > 69 and avg < 80:
    letter_grade = "C'
elif avg <= 69 and avg >= 65:
    letter_grade = "D"
elif:
    letter_grade = "F"

print("Exam 1: " + str(exam_one))
print("Exam 2: " + str(exam_two))
print("Exam 3: " + str(exam_3))
print("Average: " + str(avg))
print("Grade: " + letter_grade)

if letter-grade is "F":
    print "Student is failing."
else:
    print "Student is passing."
```

Test, test, test.

- Instead, write a smaller section with a clear purpose.
- Test it.
- Move on.

✓

```
exam_one = int(input("Input exam grade one: "))

exam_two = input("Input exam grade two: "))

exam_3 = str(input("Input exam grade three: "))

sum = exam_one + exam_two + exam_3

avg = sum / 3

if avg >= 90:
    letter_grade = "A"
elif avg >= 80 and avg < 90
    letter_grade = "B"
elif avg > 69 and avg < 80:
    letter_grade = "C'
elif avg <= 69 and avg >= 65:
    letter_grade = "D"
elif:
    letter_grade = "F"

print("Exam 1: " + str(exam_one))
print("Exam 2: " + str(exam_two))
print("Exam 3: " + str(exam_3))
print("Average: " + str(avg))
print("Grade: " + letter_grade)

if letter-grade is "F":
    print "Student is failing."
else:
    print "Student is passing."
```

Test, test, test.

- Instead, write a smaller section with a clear purpose.
- Test it.
- Move on.

✓

```
exam_one = int(input("Input exam grade one: "))

exam_two = input("Input exam grade two: "))

exam_3 = str(input("Input exam grade three: "))

sum = exam_one + exam_two + exam_3

avg = sum / 3

if avg >= 90:
    letter_grade = "A"
elif avg >= 80 and avg < 90
    letter_grade = "B"
elif avg > 69 and avg < 80:
    letter_grade = "C'
elif avg <= 69 and avg >= 65:
    letter_grade = "D"
elif:
    letter_grade = "F"

print("Exam 1: " + str(exam_one))
print("Exam 2: " + str(exam_two))
print("Exam 3: " + str(exam_3))
print("Average: " + str(avg))
print("Grade: " + letter_grade)

if letter-grade is "F":
    print "Student is failing."
else:
    print "Student is passing."
```

✓

Test, test, test.

- Instead, write a smaller section with a clear purpose.
- Test it.
- Move on.

✓ ✓ ✓

```
exam_one = int(input("Input exam grade one: "))

exam_two = input("Input exam grade two: "))

exam_3 = str(input("Input exam grade three: "))

sum = exam_one + exam_two + exam_3

avg = sum / 3

if avg >= 90:
    letter_grade = "A"
elif avg >= 80 and avg < 90
    letter_grade = "B"
elif avg > 69 and avg < 80:
    letter_grade = "C'
elif avg <= 69 and avg >= 65:
    letter_grade = "D"
elif:
    letter_grade = "F"

print("Exam 1: " + str(exam_one))
print("Exam 2: " + str(exam_two))
print("Exam 3: " + str(exam_3))
print("Average: " + str(avg))
print("Grade: " + letter_grade)

if letter-grade is "F":
    print "Student is failing."
else:
    print "Student is passing."
```

How to Debug

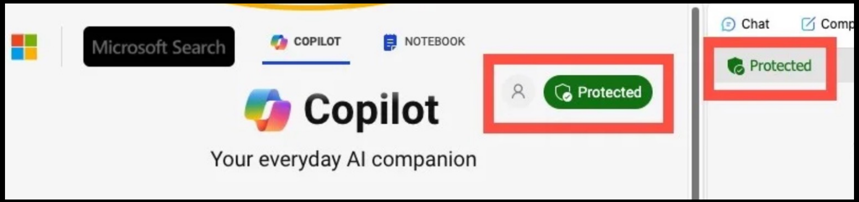
- **Run the Code "By Hand“**
  - You should develop the skill to run it in your head (or on paper). This is often a first step. You need to know the expected output to know if something is wrong!
- **Check the Python Error Output**
  - When Python encounters an error, it will print some output that can help track down the error in you code.
- **Add `print()` Statements**
  - you can often figure out what you are misunderstanding by giving yourself some evidence.
  - If you can see the values of the variables, you can then compare them against what you think they should be.
- **Use a Debugger**
  - Using an IDE (like **PyCharm**) you will see that there is an integrated debugger which allows you to do all sorts of things:
    - Look at the values of the variables.
    - Step through the code instruction by instruction.

Use print to Debug!.

```
def cats_and_dogs(cats, dogs):
    print("function called")
    if cats > 0 and dogs > 0:
        print("and")
        if cats != 0 and dogs - 1 > 0:
            print("cats")
        elif cats - 1 > 0 and dogs != 0:
            print("dogs")
    elif cats > 0 or dogs > 0:
        print("or")
        if cats !=0 or dogs - 1 > 0:
            print("cats")
        elif cats - 1 > 0 or dogs != 0:
            print("dogs")
    else:
        print("else")
        if cats != 0 and dogs - 1 > 0:
            print("cats")
        elif cats - 1 > 0 or dogs != 0:
            print("dogs")
    print("end")
```

On Using ChatGPT, Co-Pilot, and LLMS

- Be honest with yourself - what skills do you want to gain?
  - Dependency or Independency?
- Giving away your intellectual property
  - Or other people's (also known as theft!)
- [Access IP protected version of Microsoft Co-Pilot (with U of T license)]



- Advantages of new tech does not come free, and 'cost' is not money
- Invention of the ski lift drastically increased injuries, without any changes to the hill. **"Learn on the way up."**

On Using ChatGPT, Co-Pilot, and LLMS

- All research supports these tools helping experts, not beginners
- Play with it for a few hours with something you are an expert in – then you will see the limitations!
- The syntax on ChatGPT is solid, but is that all you need to solve coding problems?
- Our ‘currency’ is originality and new ideas – the **opposite** of ChatGPT
  - The most average (and therefore worst) cover letters of all time

UNIVERSITY OF TORONTO

Let's Practice Debugging

Open your notebook

Click Link:

1. Breakout Session 1

Lecture Recap

- Unfortunately, if you are going to program, you are going to spend a lot of time finding your own mistakes.
- Write small pieces of code and test.
- Work on simulating the code in your head - run the code "by hand".
- Well located `print()` statements can really help understanding the code and finding the bug.
- If you need big guns, looking into learning how to use debugger might be a good idea.

APS106

UNIVERSITY OF TORONTO

debugging.

Week 2 | Lecture 3 (3.2)

if nothing else, write #cleancode