APS106

UNIVERSITY OF TORONTO

The Programming Process.

Week 1 | Lecture 3 (1.3)

if nothing else, write #cleancode

Today's Content

Lecture 2.1

- The Programming Process

Lecture 2.2

- Functions, input & output, importing modules
- Chapters 3

Let's Code!

Convert gas mileage from American to Canadian

- In the old days (and still in the United States), the mileage of a gas-powered car was measured in miles per gallon.
- Now for places that use the metric system, we prefer to measure "mileage" as "fuel consumption" in litres per hundred kilometres.
- Write code to do the conversion to metric given a value in miles per gallon.

Open your notebook

Click Link (Lecture 1):
5. Let's Code!

Recap: What is Programming?

- A way of telling a computer what to do.
- A computer can't infer (…yet).
  - Need to tell a computer every single step it needs to do in a language it can understand.
  - How would you request an egg for breakfast to a chef and to a computer/robot?

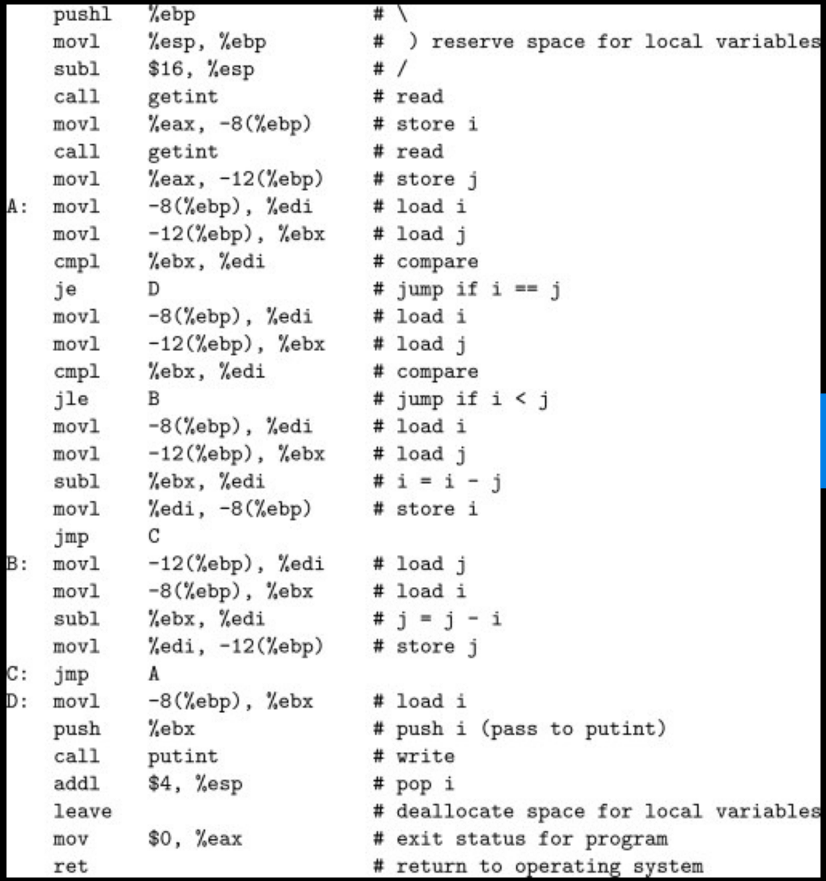
We need to tell it what to do CORRECTLY

- **To a Chef**
  1. Sunny-side up, please!

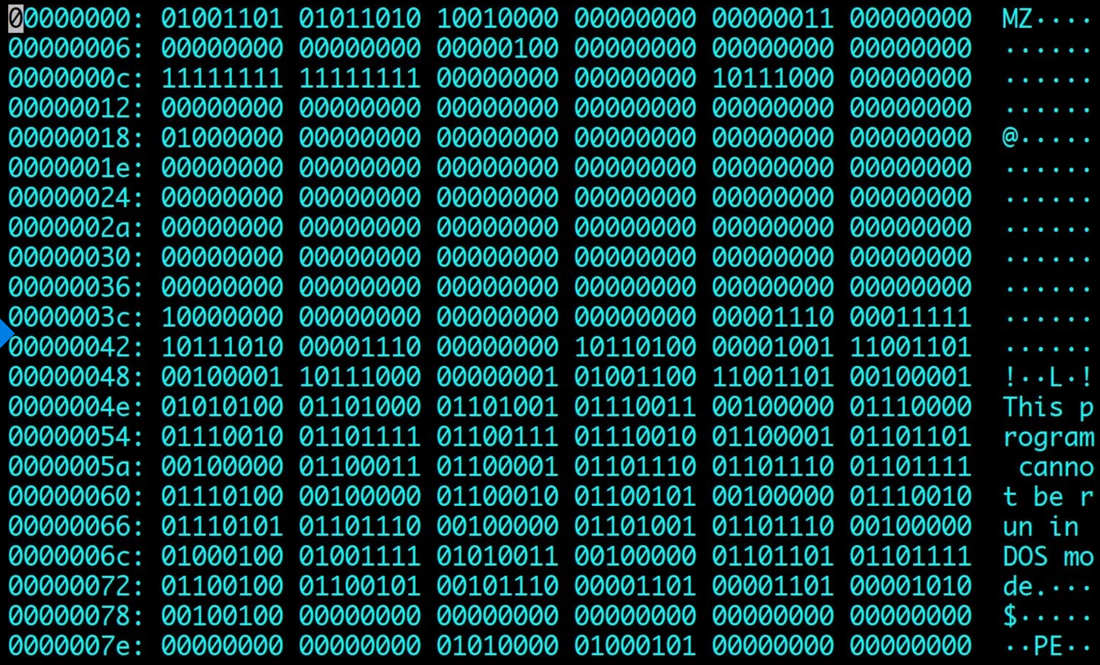
- **To a Computer**
  1. "Turn on stove"
  2. "Take out pan"
  3. "Take one egg out of fridge"
  4. "Crack egg"
  5. "Pour egg into pan"
  6. "Wait 5 minutes"

Recap: The power of programming languages

-
  ```
  if x > 10:
      print("x is greater than 10")
  ```



```
    pushl   %ebp                #  \
    movl    %esp, %ebp          #   ) reserve space for local variables
    subl    $16, %esp           #  /
    call    getint              # read
    movl    %eax, -8(%ebp)      # store i
    call    getint              # read
    movl    %eax, -12(%ebp)     # store j
A:  movl    -8(%ebp), %edi      # load i
    movl    -12(%ebp), %ebx     # load j
    cmpl    %ebx, %edi          # compare
    je      D                   # jump if i == j
    movl    -8(%ebp), %edi      # load i
    movl    -12(%ebp), %ebx     # load j
    cmpl    %ebx, %edi          # compare
    jle     B                   # jump if i < j
    movl    -8(%ebp), %edi      # load i
    movl    -12(%ebp), %ebx     # load j
    subl    %ebx, %edi          # i = i - j
    movl    %edi, -8(%ebp)      # store i
    jmp     C
B:  movl    -12(%ebp), %edi     # load j
    movl    -8(%ebp), %ebx      # load i
    subl    %ebx, %edi          # j = j - i
    movl    %edi, -12(%ebp)     # store j
C:  jmp     A
D:  movl    -8(%ebp), %ebx      # load i
    push    %ebx                # push i (pass to putint)
    call    putint              # write
    addl    $4, %esp            # pop i
    leave                       # deallocate space for local variables
    mov     $0, %eax            # exit status for program
    ret                         # return to operating system
```



```
00000000: 01001101 01011010 10010000 00000000 00000011 00000000  MZ....
00000006: 00000000 00000000 00000100 00000000 00000000 00000000  ......
0000000c: 11111111 11111111 00000000 00000000 10111000 00000000  ......
00000012: 00000000 00000000 00000000 00000000 00000000 00000000  ......
00000018: 01000000 00000000 00000000 00000000 00000000 00000000  @.....
0000001e: 00000000 00000000 00000000 00000000 00000000 00000000  ......
00000024: 00000000 00000000 00000000 00000000 00000000 00000000  ......
0000002a: 00000000 00000000 00000000 00000000 00000000 00000000  ......
00000030: 00000000 00000000 00000000 00000000 00000000 00000000  ......
00000036: 00000000 00000000 00000000 00000000 00000000 00000000  ......
0000003c: 10000000 00000000 00000000 00000000 00001110 00011111  ......
00000042: 10111010 00001110 00000000 10110100 00001001 11001101  ......
00000048: 00100001 10111000 00000001 01001100 11001101 00100001  !..L.!
0000004e: 01010100 01101000 01101001 01110011 00100000 01110000  This p
00000054: 01110010 01101111 01100111 01110010 01100001 01101101  rogram
0000005a: 00100000 01100011 01100001 01101110 01101110 01101111   canno
00000060: 01110100 00100000 01100010 01100101 00100000 01110010  t be r
00000066: 01110101 01101110 00100000 01101001 01101110 00100000  un in
0000006c: 01000100 01001111 01010011 00100000 01101101 01101111  DOS mo
00000072: 01100100 01100101 00101110 00001101 00001101 00001010  de....
00000078: 00100100 00000000 00000000 00000000 00000000 00000000  $.....
0000007e: 00000000 00000000 01010000 01000101 00000000 00000000  ..PE..
```

Recap: Arithmetic Operators

Operator	Operation	Expression	English description	Result
`+`	addition	`11 + 56`	11 plus 56	`67`
`-`	subtraction	`23 - 52`	23 minus 52	`-29`
`*`	multiplication	`4 * 5`	4 multiplied by 5	`20`
`**`	exponentiation	`2 ** 5`	2 to the power of 5	`32`
`/`	division	`9 / 2`	9 divided by 2	`4.5`
`//`	integer division	`9 // 2`	9 divided by 2	`4`
`%`	modulo (remainder)	`9 % 2`	9 mod 2	`1`

Augmented Assignment Operations

Operator	Expression	Identical Expression	English description

+=	x = 7
	x += 2
		x = 7
		x = x + 2
			x refers to 9

-=	x = 7
	x -= 2
		x = 7
		x = x - 2
			x refers to 5

*=	x = 7
	x *= 2
		x = 7
		x = x * 2
			x refers to 14

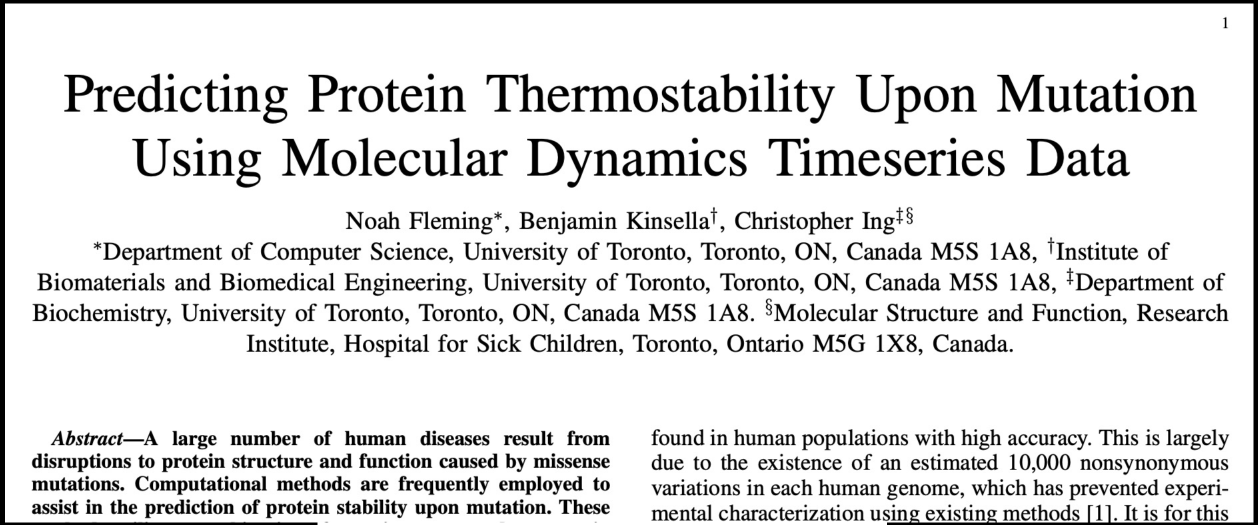
/=	x = 7
	x /= 2
		x = 7
		x = x / 2
			x refers to 3.5

//=	x = 7
	x //= 2
		x = 7
		x = x // 2
			x refers to 3

%=	x = 7
	x %= 2
		x = 7
		x = x % 2
			x refers to 1

**=	x = 7
	x **= 2
		x = 7
		x = x ** 2
			x refers to 49

Code Efficiency



1

Predicting Protein Thermostability Upon Mutation Using Molecular Dynamics Timeseries Data

Noah Fleming*, Benjamin Kinsella†, Christopher Ing‡§
*Department of Computer Science, University of Toronto, Toronto, ON, Canada M5S 1A8, †Institute of Biomaterials and Biomedical Engineering, University of Toronto, Toronto, ON, Canada M5S 1A8, ‡Department of Biochemistry, University of Toronto, Toronto, ON, Canada M5S 1A8. §Molecular Structure and Function, Research Institute, Hospital for Sick Children, Toronto, Ontario M5G 1X8, Canada.

***Abstract*—A large number of human diseases result from disruptions to protein structure and function caused by missense mutations. Computational methods are frequently employed to assist in the prediction of protein stability upon mutation. These**

found in human populations with high accuracy. This is largely due to the existence of an estimated 10,000 nonsynonymous variations in each human genome, which has prevented experimental characterization using existing methods [1]. It is for this

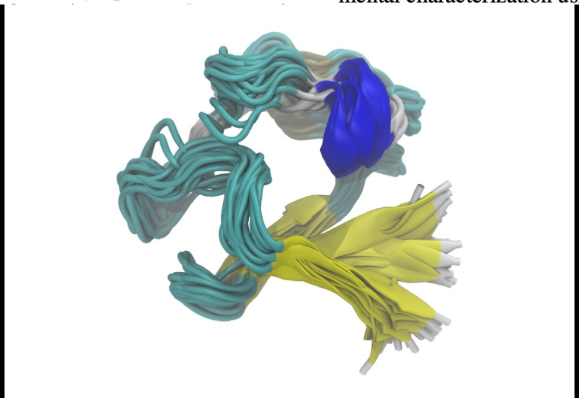


Fig. 1. Rendering of multiple time frames in simulations of the protein rubredoxin (PDB: 1BFY). Protein is colored based on secondary structure.

Supercomputer in Quebec

Augmented Assignment Operations

Operator	Expression	Identical Expression	English description

+=	x = 7
	x += 2
		x = 7
		x = x + 2
			x refers to 9

-=	x = 7
	x -= 2
		x = 7
		x = x - 2
			x refers to 5

*=	x = 7
	x *= 2
		x = 7
		x = x * 2
			x refers to 14

/=	x = 7
	x /= 2
		x = 7
		x = x / 2
			x refers to 3.5

//=	x = 7
	x //= 2
		x = 7
		x = x // 2
			x refers to 3

%=	x = 7
	x %= 2
		x = 7
		x = x % 2
			x refers to 1

**=	x = 7
	x **= 2
		x = 7
		x = x ** 2
			x refers to 49

Augmented Assignment Examples

```
>>> x = 10
>>> x += 5
>>> print(x)
15

>>> y = 17
>>> y //= 3                #y = y // 3 NOT y = 3 // y
>>> print(y)
5
```

Bringing it all together…

```
>>> x = 7
>>> y = 3
>>> x += y**2 - (4 * 3) - y
>>> print(x)
1
```

Programming Guide 101

- Readability
  - If nothing else, write #cleancode
- Comments
  - Save yourself from yourself
- Lots of testing!
  - Modular code (you will learn about functions next week)
  - Test often and with purpose
- Understanding errors
  - Types of errors
  - Error codes
- Always have a plan!

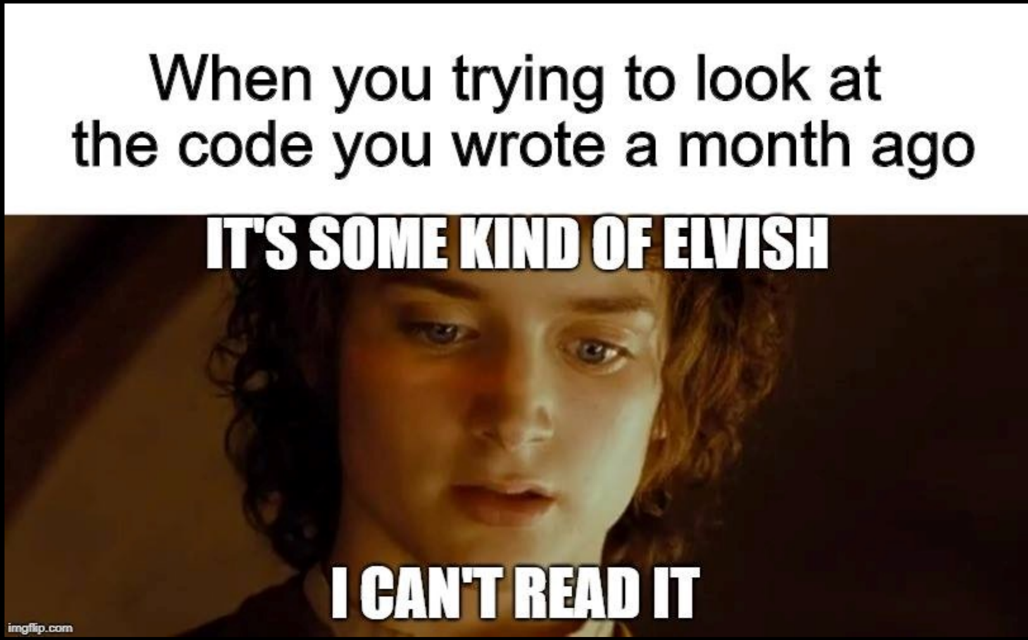
Readability Tips (#cleancode)

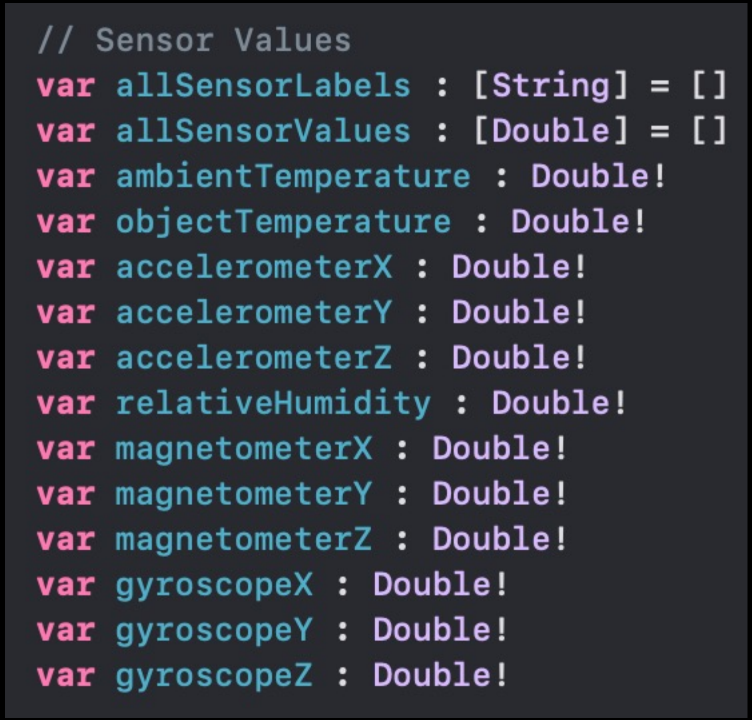
```
>>> canda = cat + panda
```

- 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`

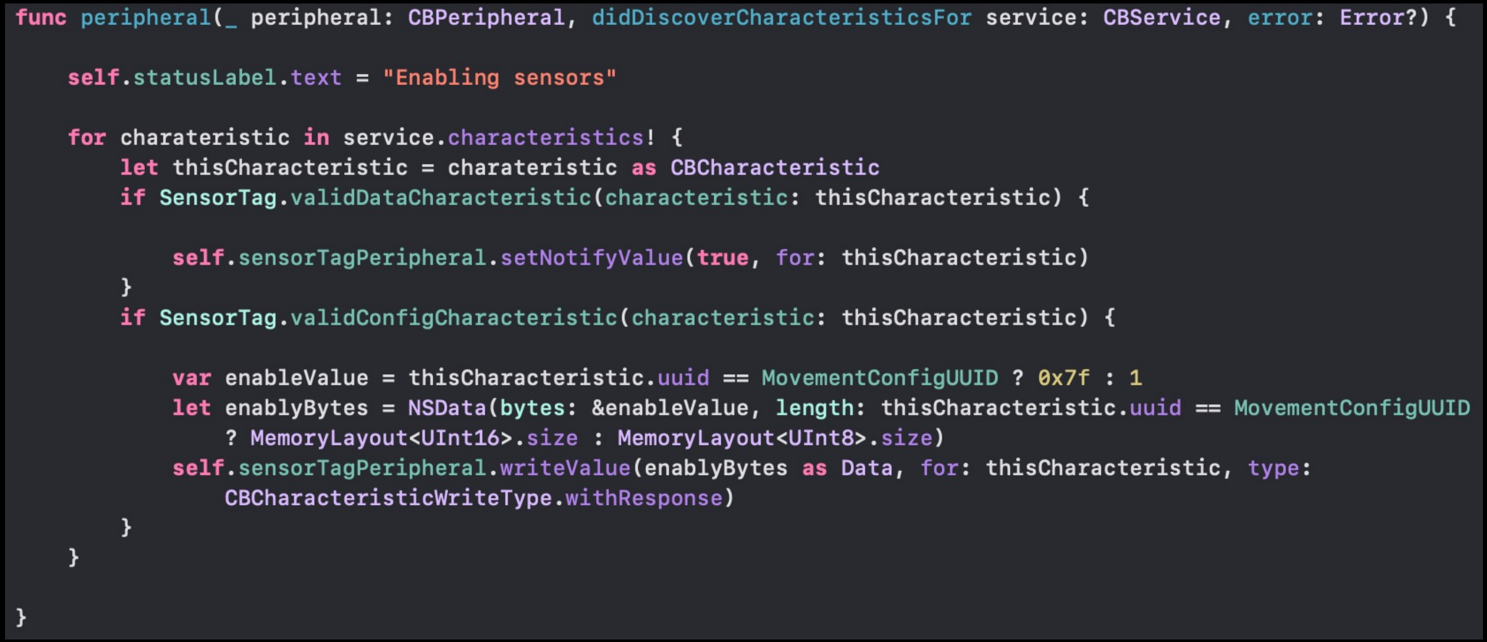
Comments

- 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

UNIVERSITY OF TORONTO

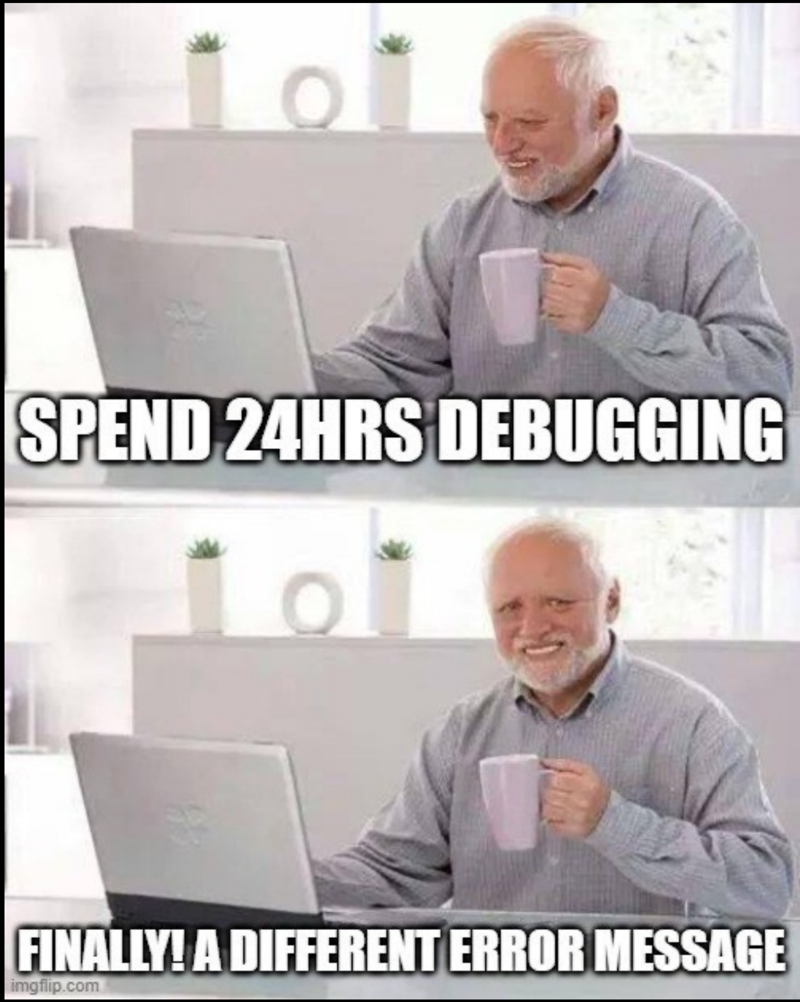
Testing!

- 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!

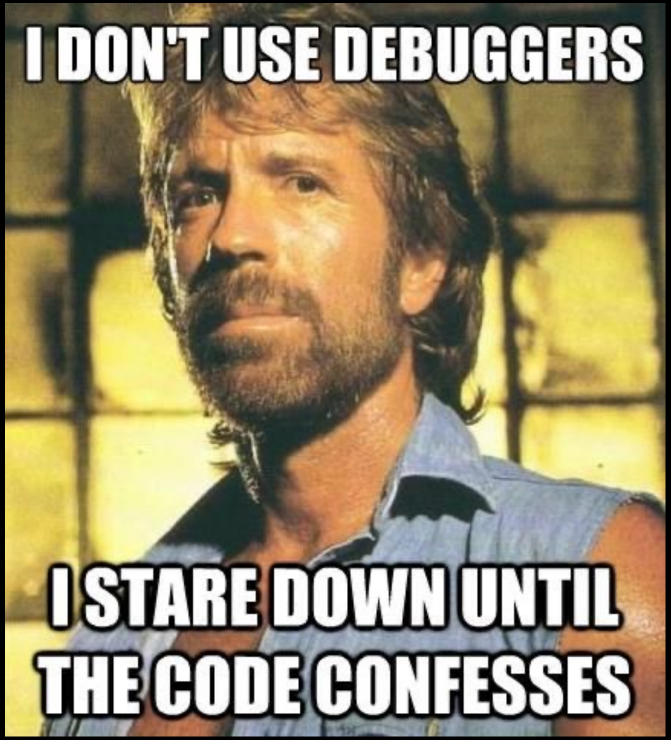
Golden Rule: Never spend more than 15 minutes programming without testing

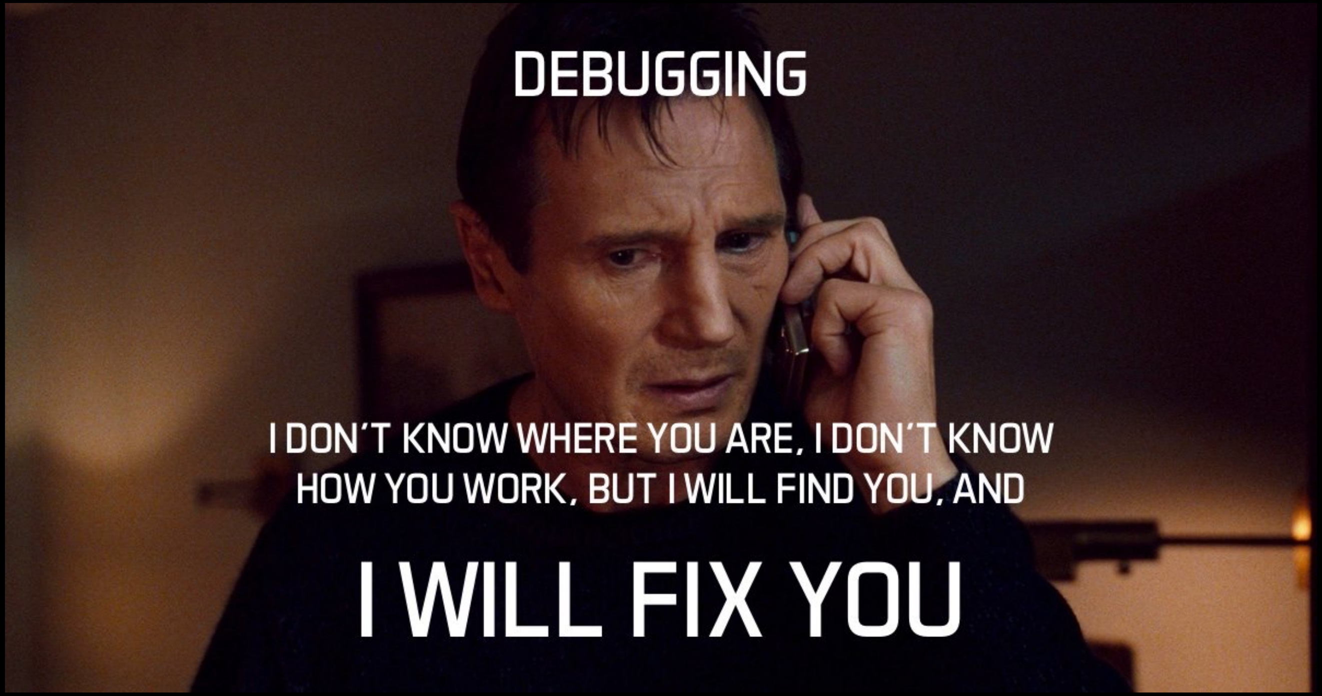
Error Reduction vs Debugging

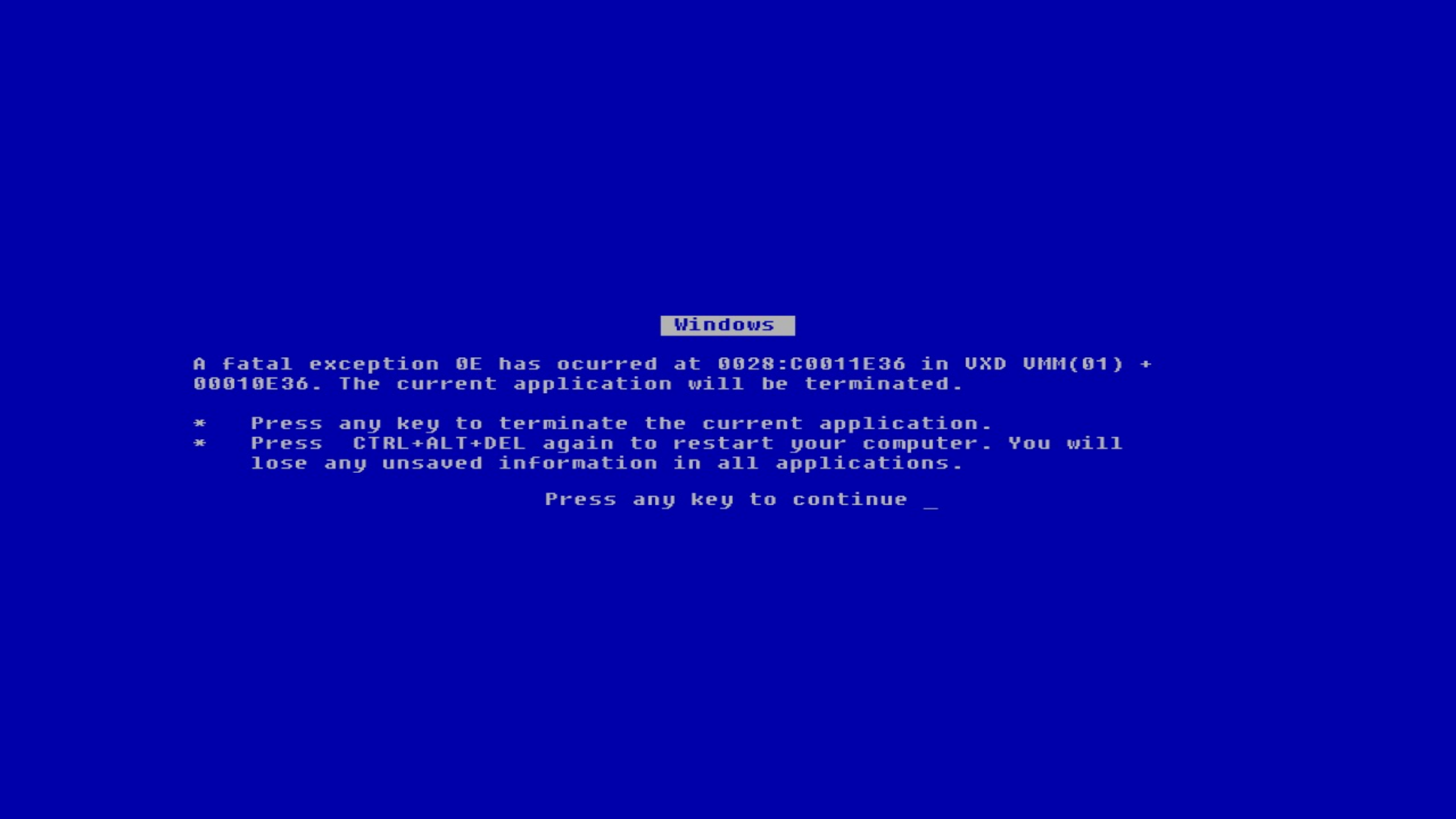
- It is pretty much impossible to write code without errors.
  - *Error Reduction*: techniques we can use to reduce the number and severity of errors.
  - *Debugging*: techniques for identifying and correcting errors



Which student will you be?




DEBUGGING
I DON'T KNOW WHERE YOU ARE, I DON'T KNOW HOW YOU WORK, BUT I WILL FIND YOU, AND
I WILL FIX YOU



```
                          Windows

A fatal exception 0E has ocurred at 0028:C0011E36 in VXD VMM(01) +
00010E36. The current application will be terminated.

*   Press any key to terminate the current application.
*   Press  CTRL+ALT+DEL again to restart your computer. You will
    lose any unsaved information in all applications.

                    Press any key to continue _
```

Types of Errors

- 🚫 Syntax error
- 🚫 Semantic error
- 🚫 Logical error
- 🚫 Runtime error

Syntax Errors

- *Syntax error*: results when the programming language cannot understand your code.
- Examples: missing an operator or two operators in a row, illegal character in a variable name, missing a parentheses or bracket etc.
- In English, a syntax error is like a spelling error

```
>>> 3) + 2 * 4
Syntax Error: unmatched ')':  line 1, pos 2
```

Semantic Errors

- *Semantic error*: results from improper use of the statements or variables.
- Examples: using an operator not intended for the variable type, calling a function with the wrong argument type, or wrong number of arguments, etc.
- In English, a semantic error is like a grammar error

```
>>> “Hello” - 4

TypeError: unsupported operand type(s) for -: 'str' and 'int’

>>> number = number * 2

NameError: name 'number' is not defined
```

Runtime Errors

- *Runtime error:* is an error that occurs during the execution (runtime) of a program. Generally do not occur in simple programs.
- The code could run fine most of the time, but in certain circumstances the program may encounter an unexpected error and crash.
- Examples: infinite loops, attempting to access an index out of bounds, etc.

```
>>> x = 10
>>> while x > 0:
        print("This is the song that never ends")
```

Logical Errors

- *Logical Error:* results from unintended result due to a miscalculation or misunderstanding of specifications.
- Examples: miscalculation, typo, misunderstanding of requirements, indentation mistakes, operator precedence, integer instead of floating-point division, etc.
- **Most difficult to fix** because the code will execute without crashing. There are no error messages produced.

Logical Error Examples

71.6 degrees F is about 22 degrees C

```
>>> fahrenheit = 71.6
>>> celsius  = fahrenheit – 32 * 5/9
>>> celsius
53.82222222222216
```

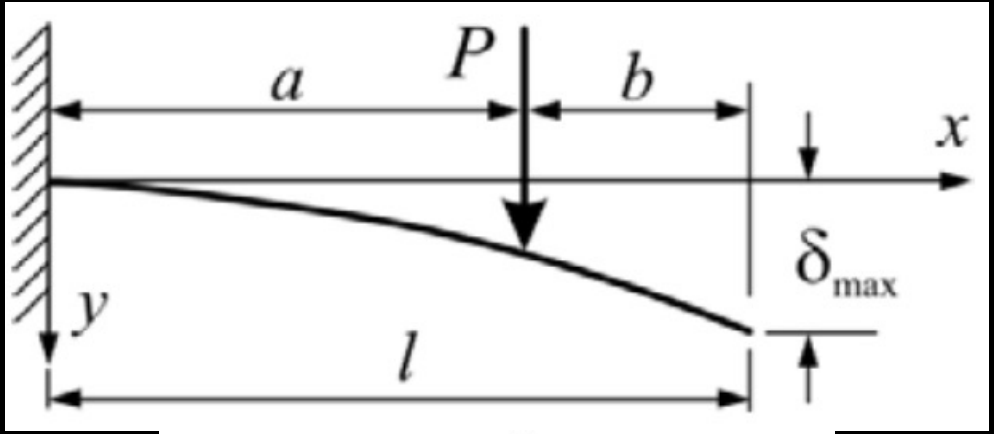
Correct logic: celsius = (fahrenheit – 32) * 5/9

```
>>> fahrenheit = 716
>>> celsius  = (fahrenheit – 32) * 5/9
>>> Celsius
380.0
```

Whoops, typo! Forgot the decimal.

Let's Practice!

- The diagram and formula below introduces variables for the calculation of the deflection in a beam. Write a program that can calculate the δmax, or deflection of a beam.



$$\delta_{max} = \frac{Pa^2}{6EI}(3l - a)$$

Open your notebook

Click Link:
1. Calculate Deflection of a Beam

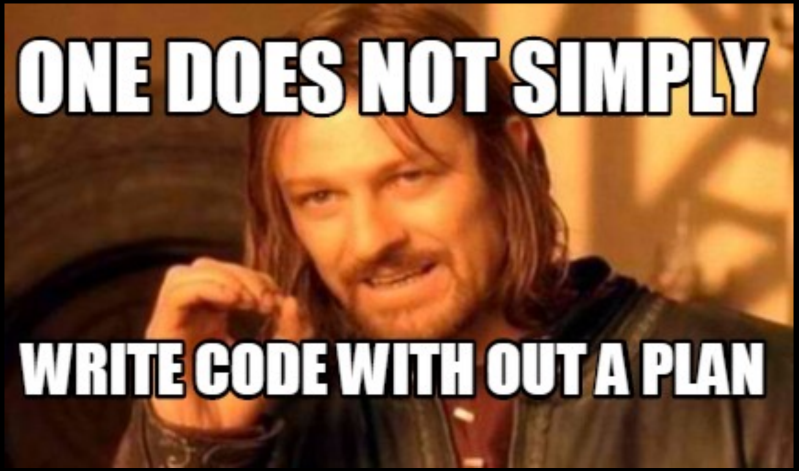
Planning an Essay

- How do you start writing an essay?
  - Read the question carefully and with intent
  - Think about what information was provided in the topic that you should include in your answer
  - Brainstorm different ways to answer the question
  - Skim through course material to see what could help
  - Scaffold or quickly structure each paragraph
  - Figure out what you want to conclude and think of ways to get there
  - Make sure each section has purpose (you aren't repeating yourself)
  - Think about order (what needs to be said at the beginning vs what needs to be said at the end)

Planning Code

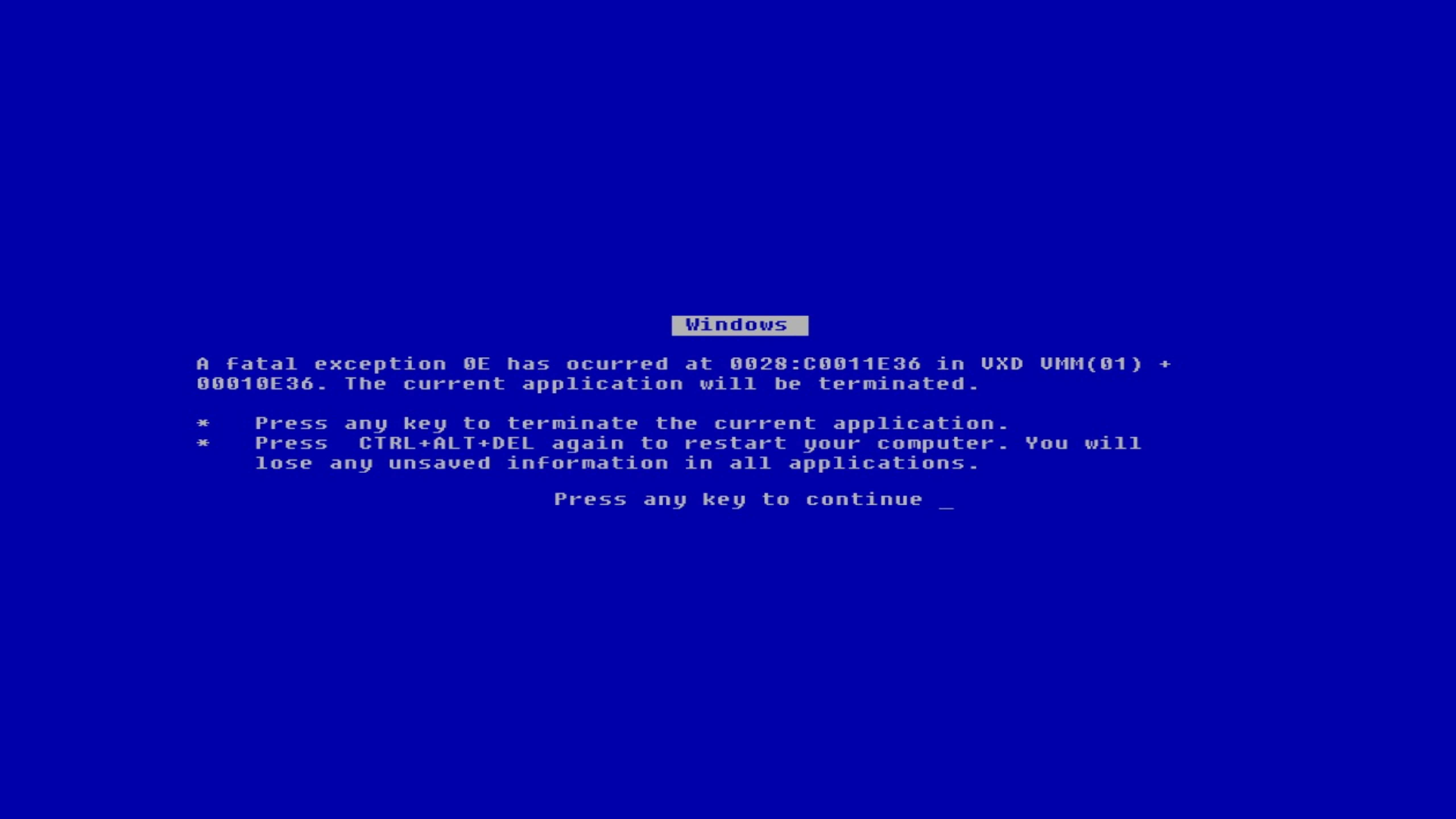
- How do you start writing code?
  - Read the question carefully and with intent
  - Think about what information was provided in the topic that you should include in your answer
  - Brainstorm different ways to answer the question
  - Skim through course material to see what could help
  - Scaffold or quickly structure each paragraph
  - Figure out what you want to conclude and think of ways to get there
  - Make sure each section has purpose (you aren’t repeating yourself)
  - Think about order (what needs to be said at the beginning vs what needs to be said at the end)

Failing to Plan is Planning to Fail!



Open your notebook

Click Link:
2. Calculating Chemical Rate Constants



```
                              Windows

A fatal exception 0E has ocurred at 0028:C0011E36 in VXD VMM(01) +
00010E36. The current application will be terminated.

*   Press any key to terminate the current application.
*   Press  CTRL+ALT+DEL again to restart your computer. You will
    lose any unsaved information in all applications.

                        Press any key to continue _
```

APS106

UNIVERSITY OF TORONTO

The Programming Process.

Week 1 | Lecture 2 (1.2)

if nothing else, write #cleancode