APS106

UNIVERSITY OF TORONTO

The Programming Process.

Week 1 | Lecture 3 (1.3)

if nothing else, write #cleancode

This Week's Content

- **Lecture 1.1**
  - Introduction
- **Lecture 1.2**
  - Variables, Expressions, and Operators
  - Chapters 1,2, and 3

Lecture 1.3

  - **The Programming Process**

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?

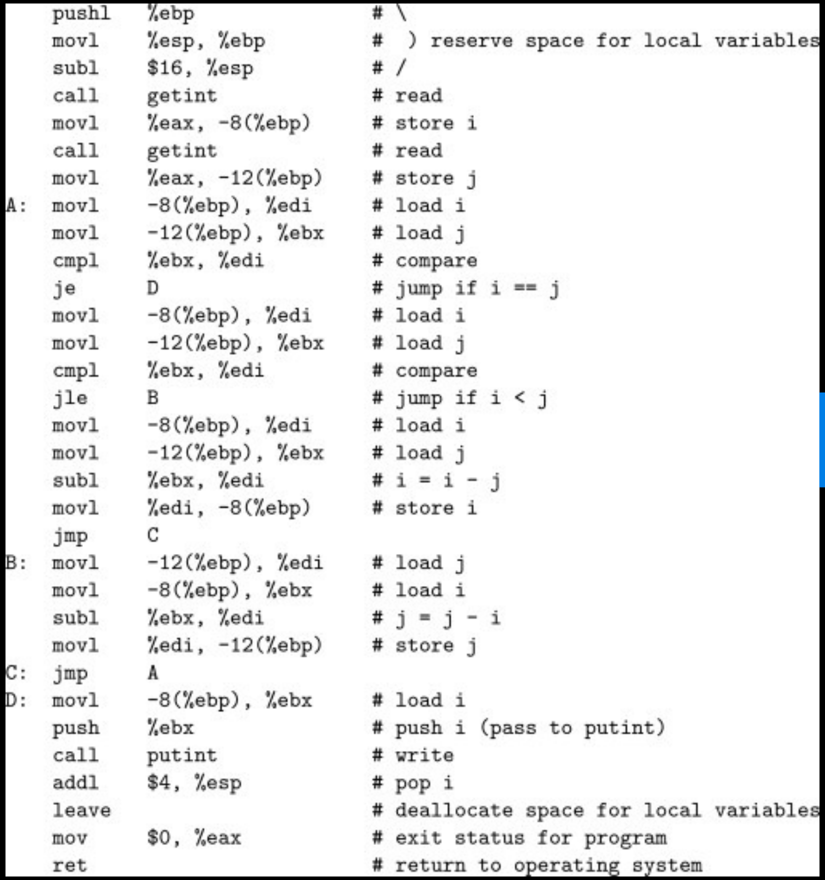
We need to tell it what to do CORRECTLY

- **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")
  ```



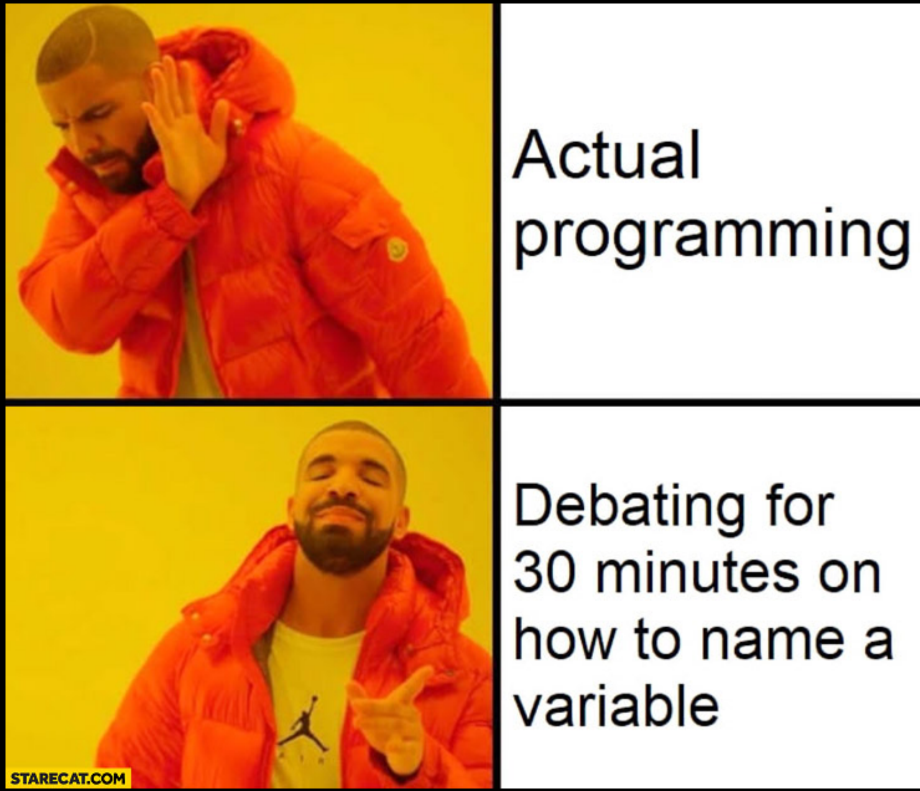
```
    pushl   %ebp              # \
    movl    %esp, %ebp        #  ) reserve space for local variables
    subl    $16, %esp         # /
    call    getint            # read
    movl    %eax, -8(%ebp)    # store i
    call    getint            # read
    movl    %eax, -12(%ebp)   # store j
A:  movl    -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
```

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`

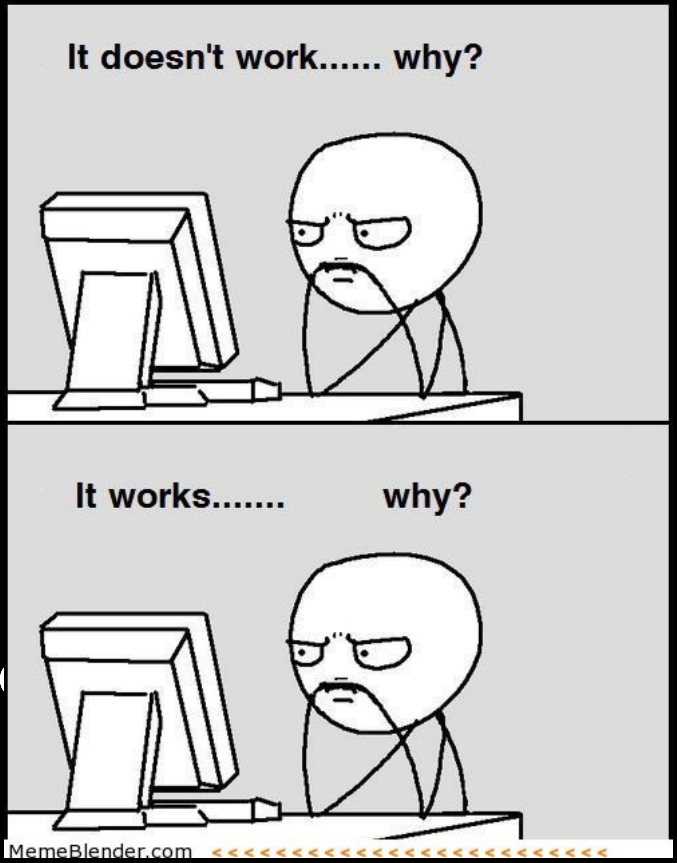
Recap: Variable Names and Conventions

- 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



Programming Guide

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



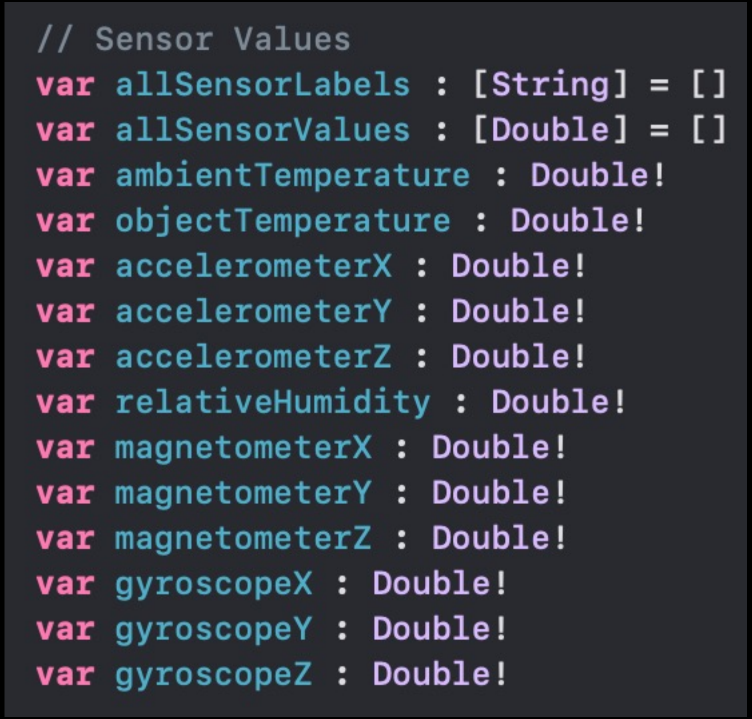
Readability Tips (#cleancode)

```
>>> 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`

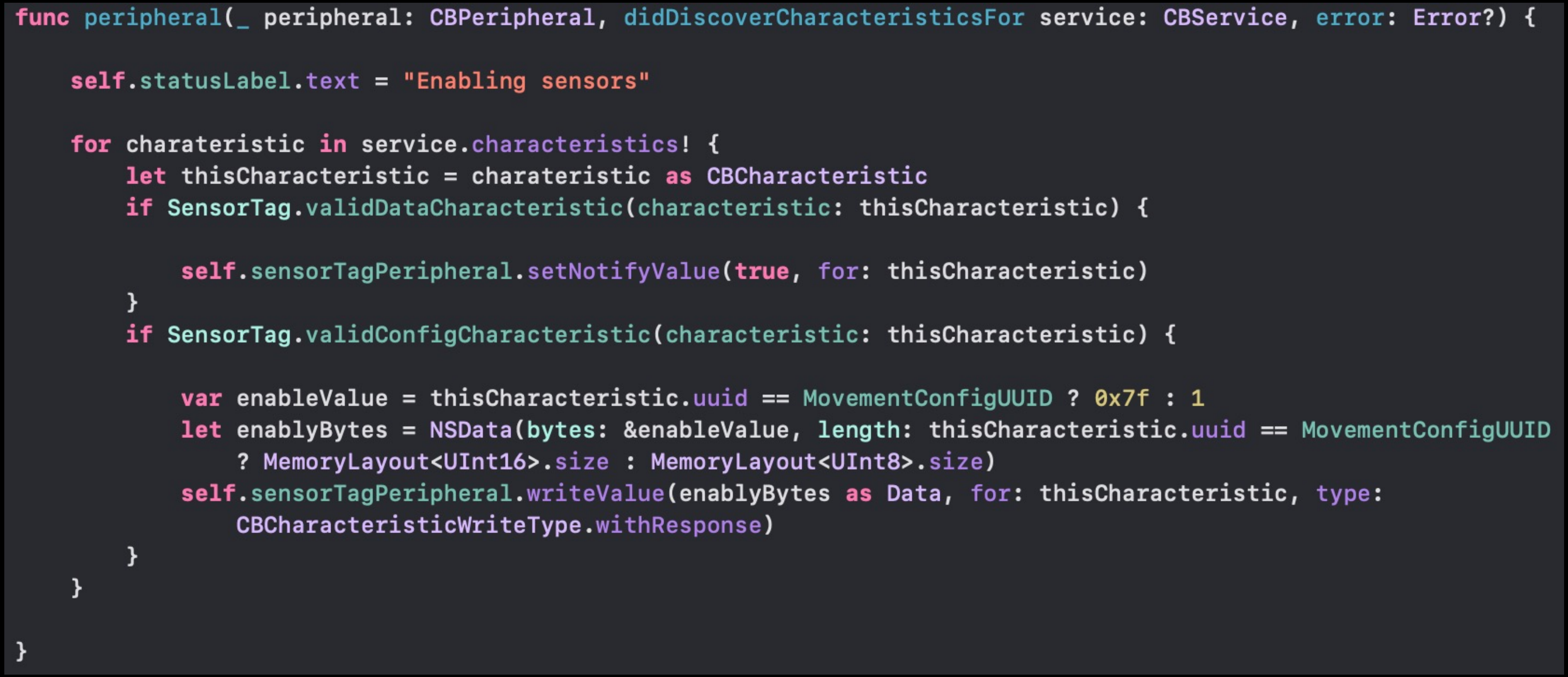
Comments

- 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!
```

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



```swift
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

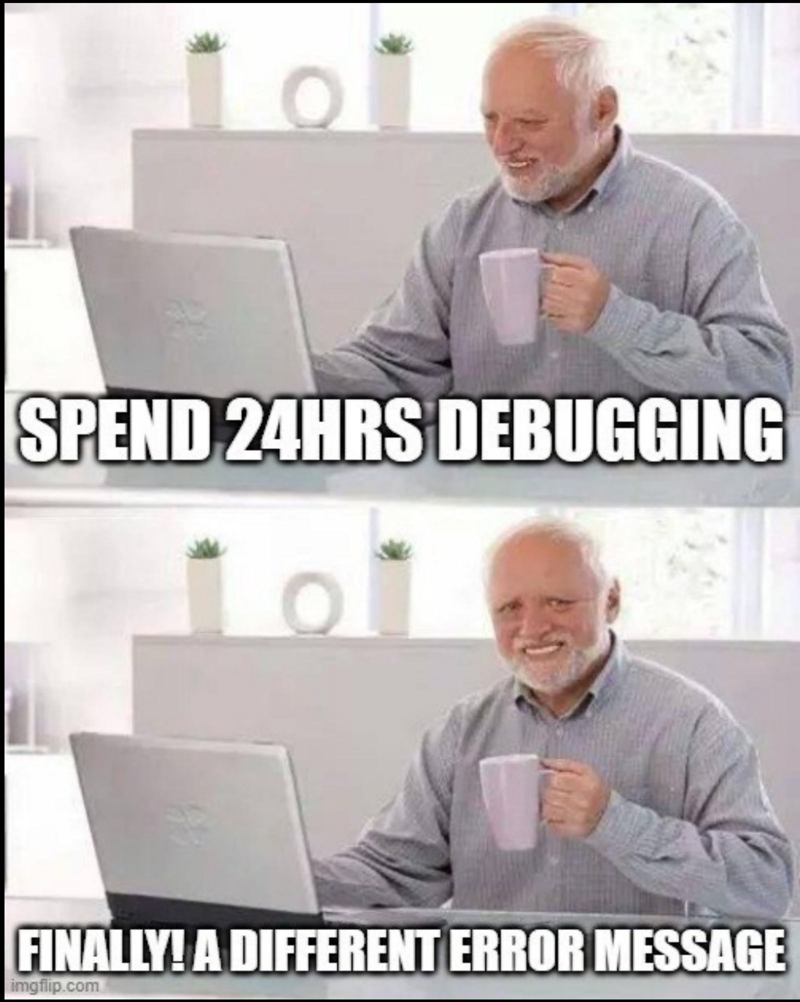
Testing!

- 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!

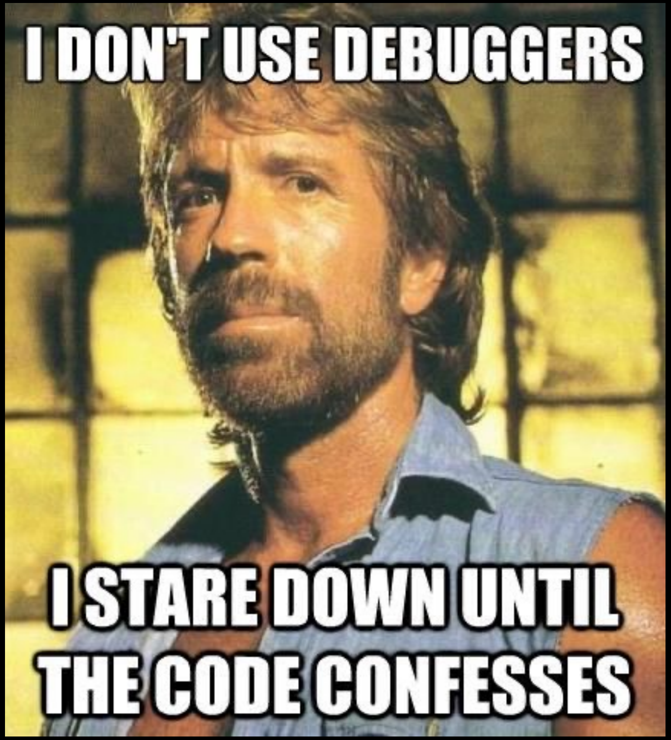
Golden Rule: Never spend more than 15 minutes programming without testing

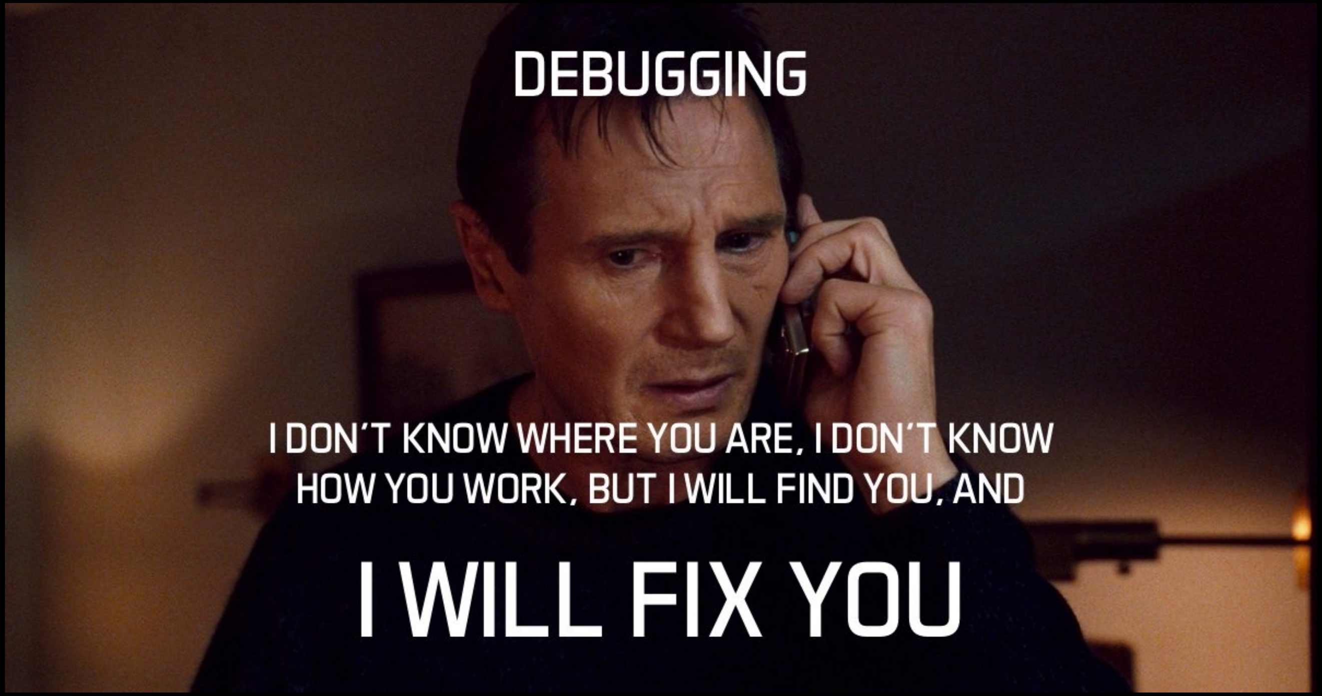
Error Reduction vs Debugging

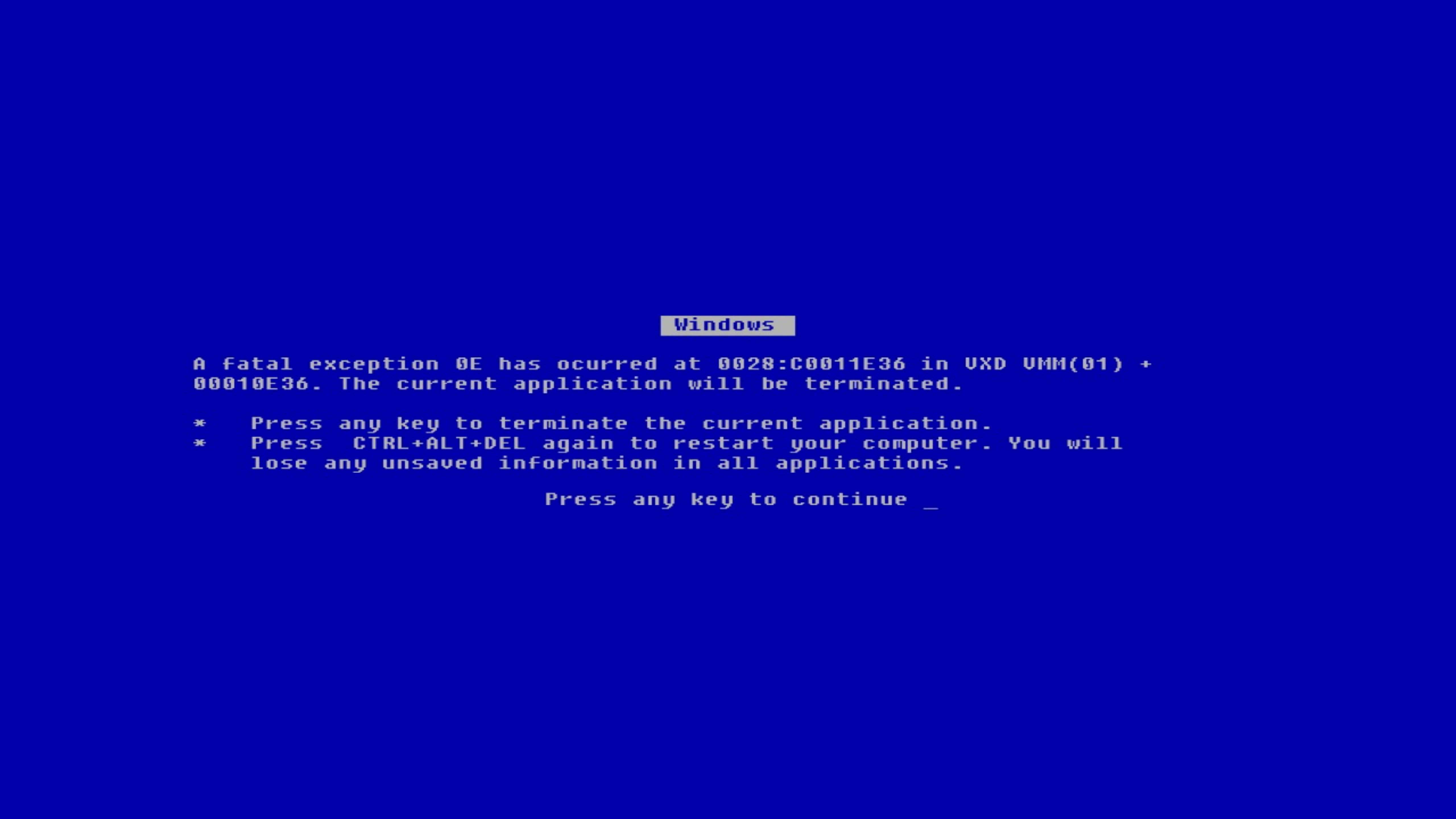
- It is pretty much impossible to write code without errors.
  - *Error Reduction*: techniques we can use to reduce the number and severity of errors.
  - *Debugging*: techniques for identifying and correcting errors



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
```

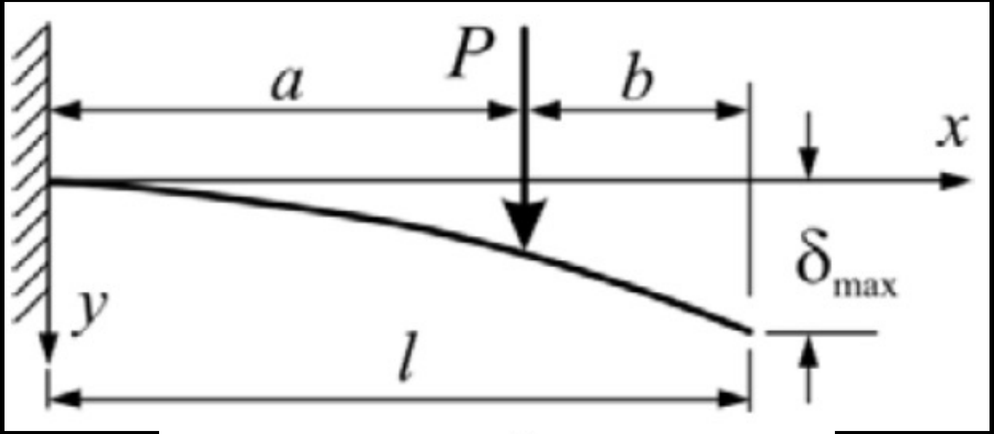
Correct logic: celsius = (fahrenheit – 32) * 5/9

```
>>> 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

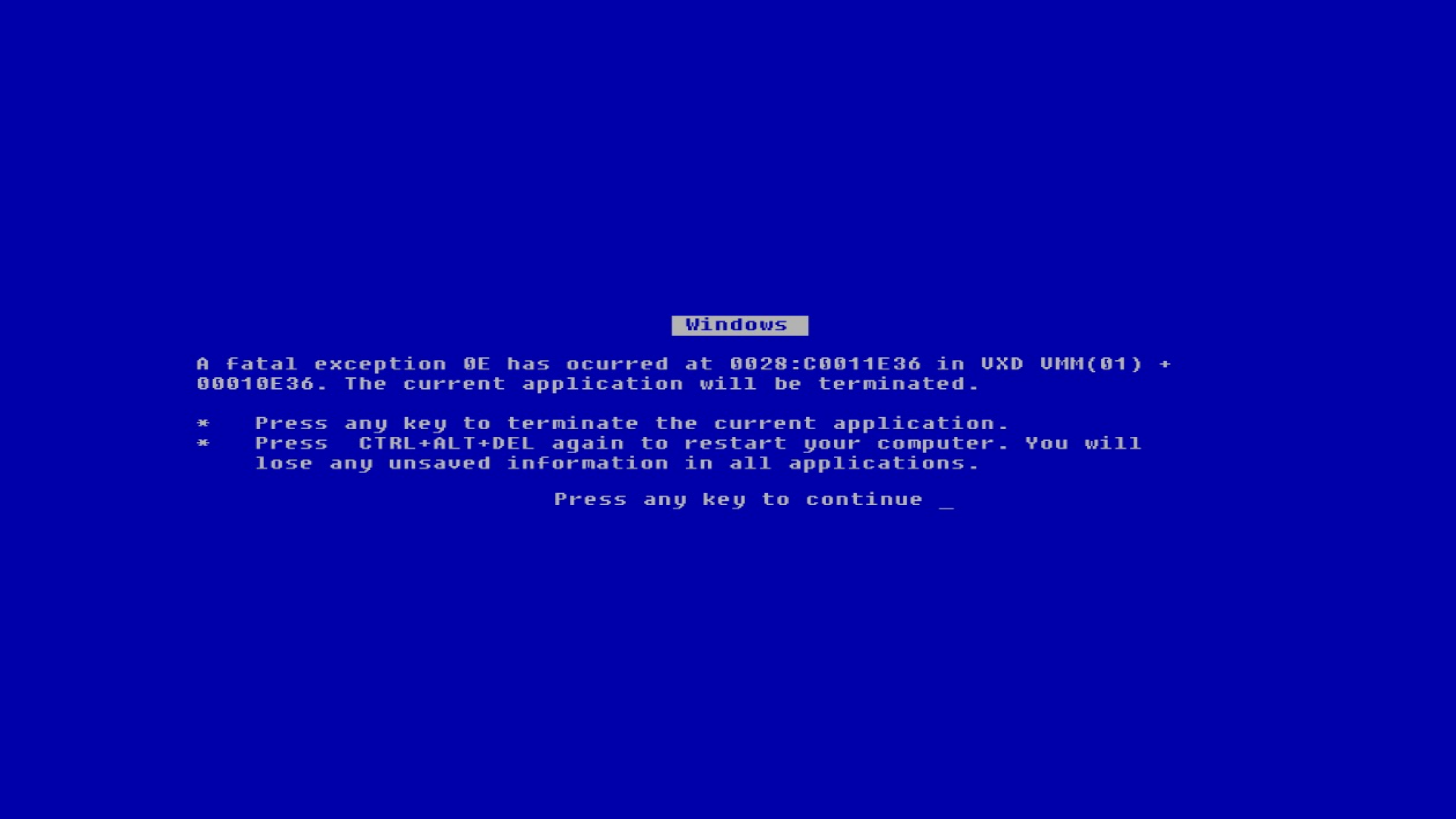
UNIVERSITY OF TORONTO

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)



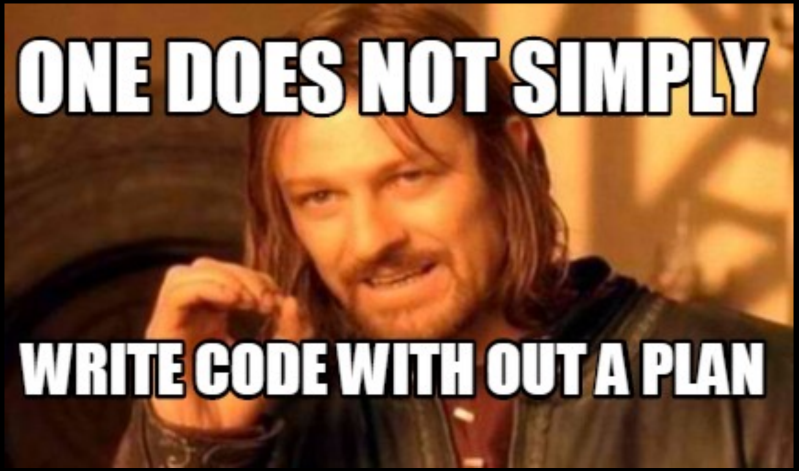
```
                              Windows

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 _
```

Please… please… PLEASE have a plan!



Open your notebook

Click Link:
2. Calculating Chemical Rate Constants

APS106

UNIVERSITY OF TORONTO

The Programming Process.

Week 1 | Lecture 2 (1.2)

if nothing else, write #cleancode