

# APS106

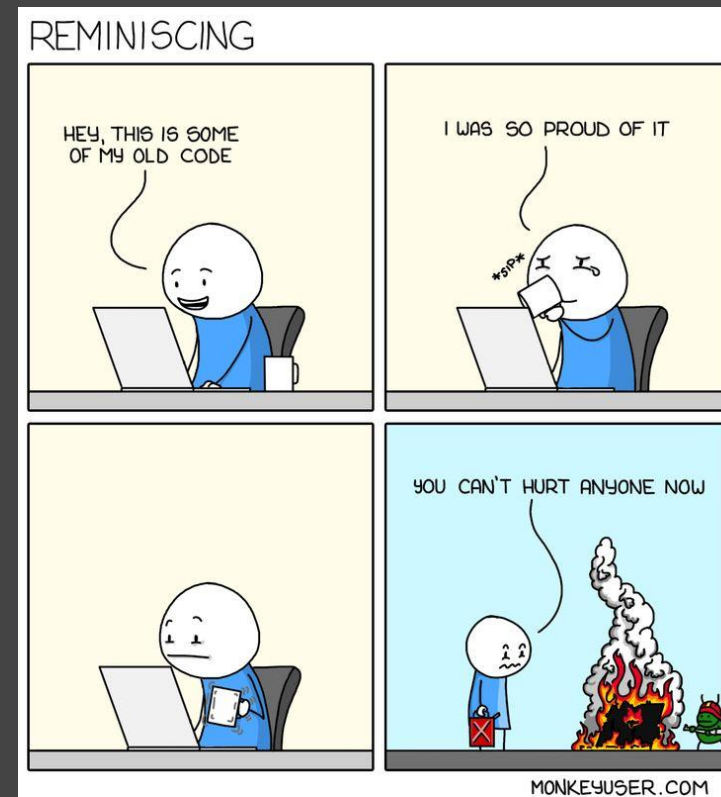
## Variables, Expressions and Operators.

Week 1 | Lecture 3 (1.3)

### While waiting for class to start:

Download and open the Jupyter Notebook (.ipynb) for Lecture 1.3 (Kinsella & Goodfellow).

You may also use this lecture's JupyterHub link instead.



UNIVERSITY OF  
TORONTO

### Upcoming:

- Lab 0 released Thursday at 6 PM
- Reflection 1 released Friday at 6 PM
- First PRA section – Friday 3-5 PM
- All TUT (Tutorials) & PRA (Labs) running next week

if nothing else, write `#cleancode`



AGAINST ALL ODDS

# UNIVERSITY OF TORONTO ENGINEERING KOMPETITIONS

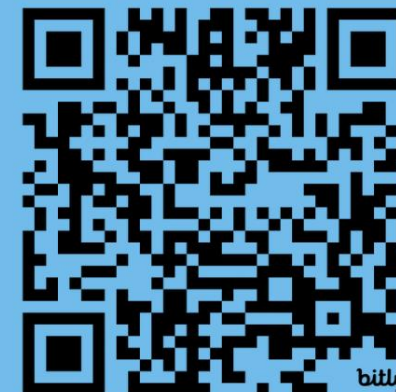


JANUARY 18-19 2025

## 7 KOMPETITIONS

- Junior Design
- Senior Design
- Programming
- Consulting
- Bio-Engineering
- Re-Engineering
- Parliamentary Debate

GET YOUR TICKETS NOW!



LEARN MORE AT:



@uoftutek



utek@skule.ca



utek.skule.ca

UNIVERSITY OF TORONTO  
ENGINEERING SOCIETY

# This Week's Content

- Lecture 1.1
  - Introduction
- Lecture 1.2
  - The Coding Toolbox
- **Lecture 1.3**
  - **Variables, Operators, and Expressions**

# Reminder for Success: Practice!

- Programming is a language
  - Everything is cumulative
- Practice Problems released weekly, with solutions released 2 weeks later
- Use labs and practice problem sets as litmus tests
  - Do them all, take note of which questions give you trouble

## Practice Problems Homepage

Week	Practice Problems Posted every Sunday at 11:59 pm	Solutions Posted 2 weeks later
Week 1 (Jan 6 - 10)	Prepare your computer by installing Python, Jupyter, and an IDE such as VSCode  <a href="#">Practice Problem Week 1</a> ↓	Prepare your computer by installing Python, Jupyter, and an IDE such as VSCode  Practice Problem Week 1 Answers

# Introducing Python

- No end-of-instruction separators, such as semicolons (like in C or Java)
- **Programs are stored in .py files**
- Comments start with a # character
- **Whitespace matters (exactly 4 spaces means indentation)**
- Python is an interpreted language (not a compiled one)
  - You can run code one statement at a time, just like a calculator or Matlab
  - This means variables can change type during runtime, and do not have to be declared before running

# 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

# Arithmetic Operator Precedence

- When multiple operators are combined in a single expression, the operations are evaluated in order of precedence (from left to right)

Operator	Precedence
<b>**</b>	<b>highest</b>
<b>- (negation)</b>	
<b>*, /, //, %</b>	
<b>+ (addition), - (subtraction)</b>	<b>lowest</b>

# Using Python as a Calculator

- Let's start with a simple use of Python to get the feel for our new environment
- Don't forget your BEDMAS (or PEMDAS)!
- Lecture Homepage -> scroll down to Week 1, Kinsella & Goodfellow Link -> Lecture 1.3 -> "Jupyter Notebook file"
- Open this file through Jupyter Notebook (open from Anaconda)
- **Jupyter Hub: [jupyter.utoronto.ca](https://jupyter.utoronto.ca)**
  - Requires no installation, not even Python!

**Open your  
notebook**

**Click Link:**

**1. Using Python as a  
Calculator**



# Variables and Memory

- The most basic thing you can do in a computer program is to assign a value to a variable.
- Assignment statements

**variable** = **expression**

- $x = 20$   
(or)
  - $y = 20 + 5 * 2$
- **Rules for assignment**
  - 1. Evaluate the expression to the right of = sign (produces memory address of the value)
  - 2. Store the memory address in the variable on the left of the = sign

# 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 (or snake\_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

When I'm searching for a meaningful variable name



# Memory Visualization Example

```
>>> difference = 20
```

```
>>> double = 2 * difference
```

```
>>> double
```

```
40
```

```
>>> difference = 5
```

```
>>> double
```

```
40
```

# Memory Visualization Example

```
>>> difference = 20
```

```
>>> double = 2 * difference
```

```
>>> double
```

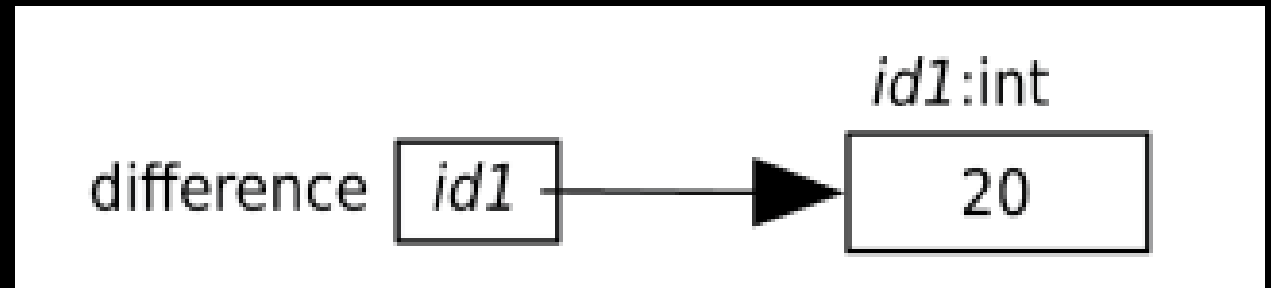
```
40
```

```
>>> difference = 5
```

```
>>> double
```

```
40
```

1. Evaluate the expression on the right of the = sign -> 20. This produces the value 20, which we'll put at memory address id1.
2. Make the variable on the left of the = sign, difference, refer to 20 by storing id1 in difference.



# Memory Visualization Example

```
>>> difference = 20
```

```
>>> double = 2 * difference
```

```
>>> double
```

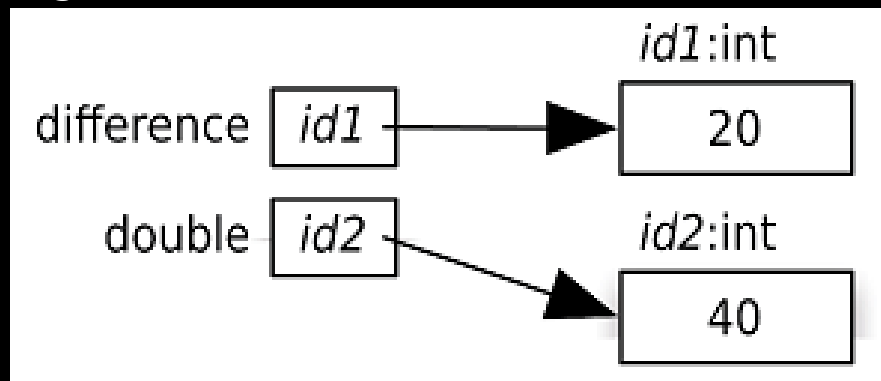
```
40
```

```
>>> difference = 5
```

```
>>> double
```

```
40
```

1. Evaluate the expression on the right of the = sign:  $2 * \text{difference}$ . As we see in the memory model, `difference` refers to the value 20, so this expression is equivalent to  $2 * 20$ , which produces 40. We'll pick the memory address `id2` for the value 40.
2. Make the variable on the left of the = sign, `double`, refer to 40 by storing `id2` in `double`.



# Memory Visualization Example

```
>>> difference = 20
```

```
>>> double = 2 * difference
```

```
>>> double
```

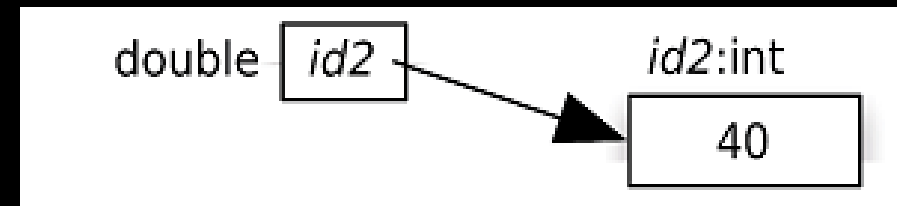
```
40
```

```
>>> difference = 5
```

```
>>> double
```

```
40
```

When Python executes the third statement, `double`, it merely looks up the value that `double` refers to (40) and displays it.



# Memory Visualization Example

```
>>> difference = 20
```

```
>>> double = 2 * difference
```

```
>>> double
```

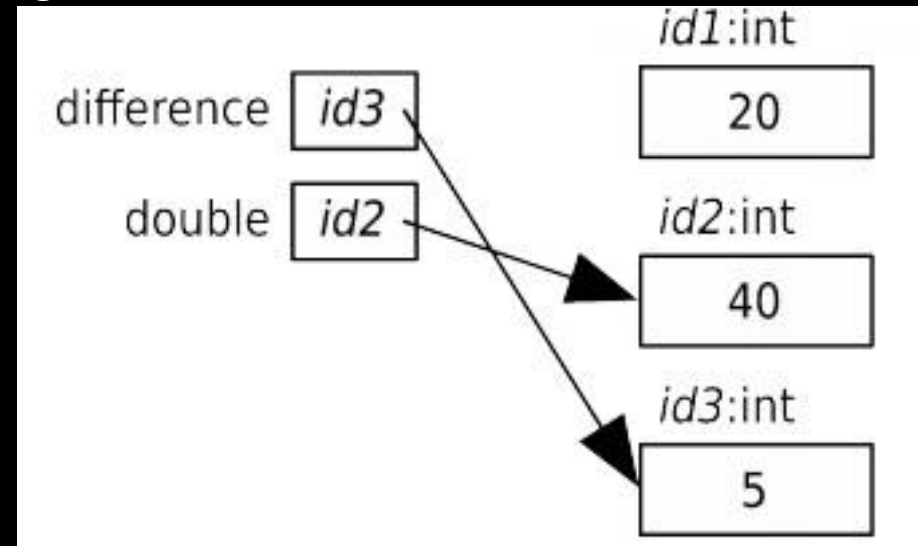
```
40
```

```
>>> difference = 5
```

```
>>> double
```

```
40
```

1. Evaluate the expression on the right of the = sign: 5. This produces the value 5, which we'll put at the memory address id3.
2. Make the variable on the left of the = sign, difference, refer to 5 by storing id3 in difference.



# Memory Visualization Example

```
>>> difference = 20
```

```
>>> double = 2 * difference
```

```
>>> double
```

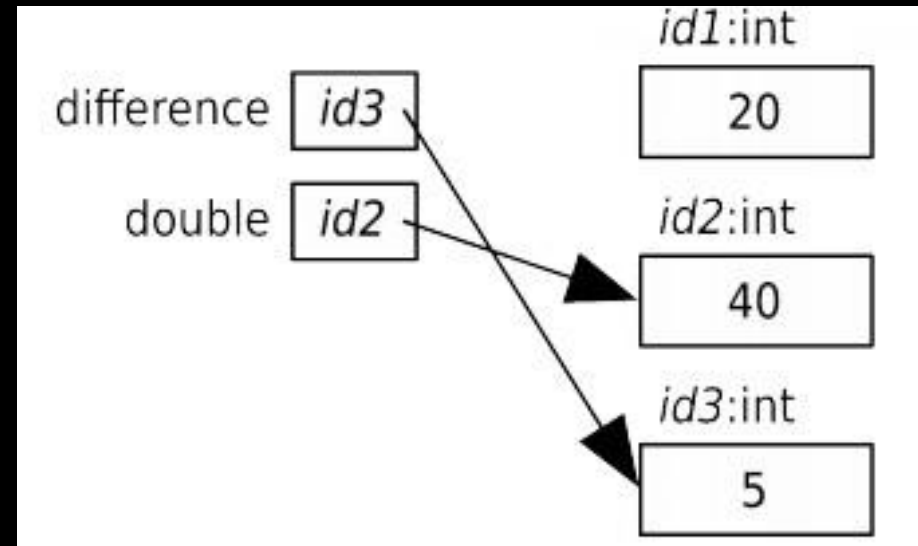
```
40
```

```
>>> difference = 5
```

```
>>> double
```

```
40
```

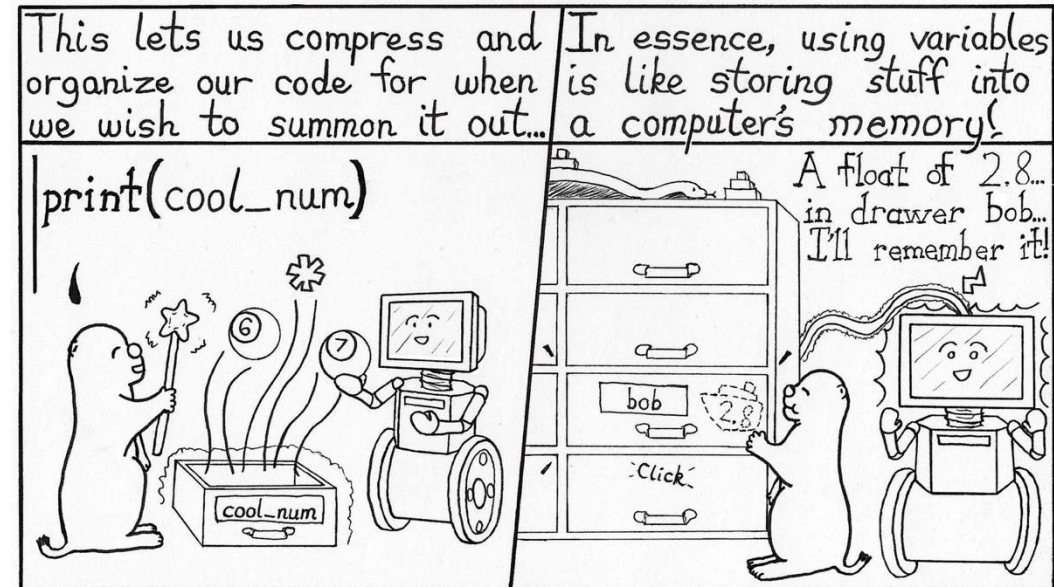
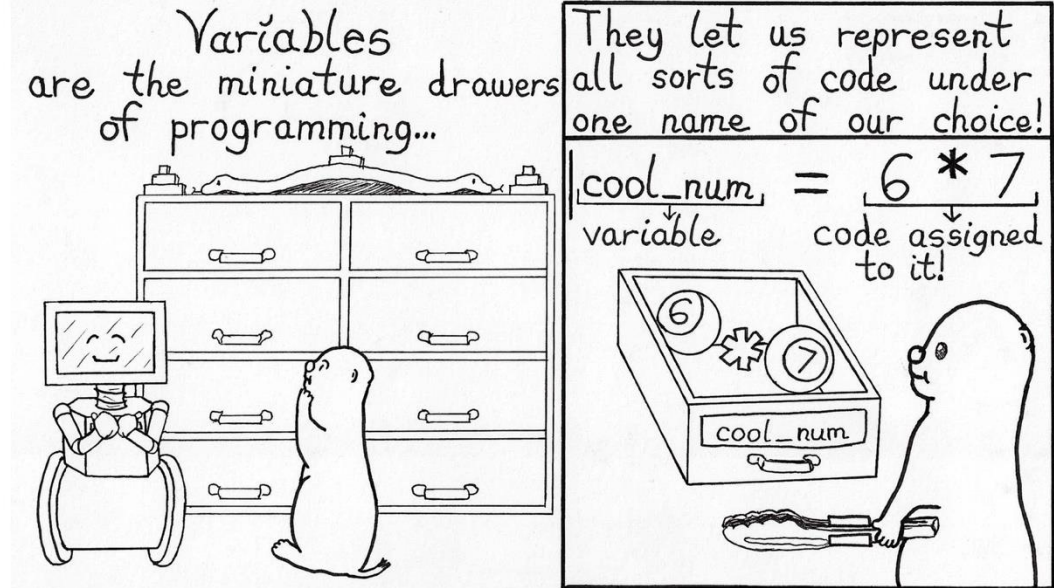
Variable `double` still contains `id2`, so it still refers to 40. Neither variable refers to 20 anymore which is OK as Python will take care of recycling of memory when it is no longer used.





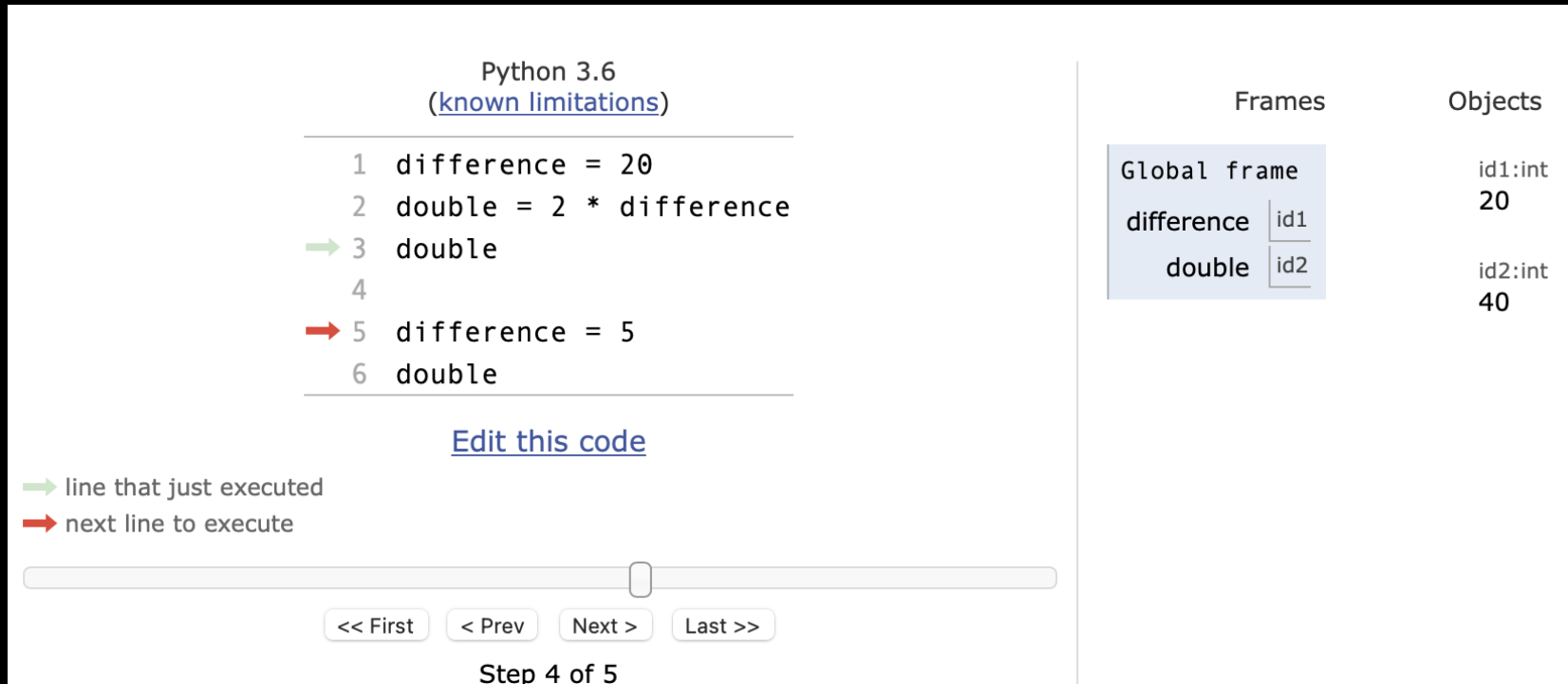
# Variables are like labelled drawers

- Have a unique label/name and some contents
  - "Top" contains socks, "Seb's" contains records
- Labels can change, or contents can be moved
  - We can rename "Top" to "Socks" or move Seb's records to "Bottom" drawer
- A variable/drawer can be used more than once
  - "Top" can be opened as often as we want (or never)
- Variables/drawers can hold different contents of different sizes
  - "Sock" drawer can be huge or tiny
- It is possible to label the same drawer with multiple names
  - "Top" and "Sock" could both refer to the same drawer
- Cannot simultaneously label multiple drawers the same name
  - Can't have two "Sock" drawers, but you could have "Dress Socks" and "Gym Socks"



# Python Tutor Visualization

- <http://pythontutor.com/visualize.html>
- Code might seem simple now...



Python 3.6  
([known limitations](#))

```
1 difference = 20
2 double = 2 * difference
→ 3 double
4
→ 5 difference = 5
6 double
```

[Edit this code](#)

→ line that just executed  
→ next line to execute

Global frame

Frames	Objects
Global frame	id1:int 20
difference	id1
double	id2:int 40

Step 4 of 5

# Variables and Memory

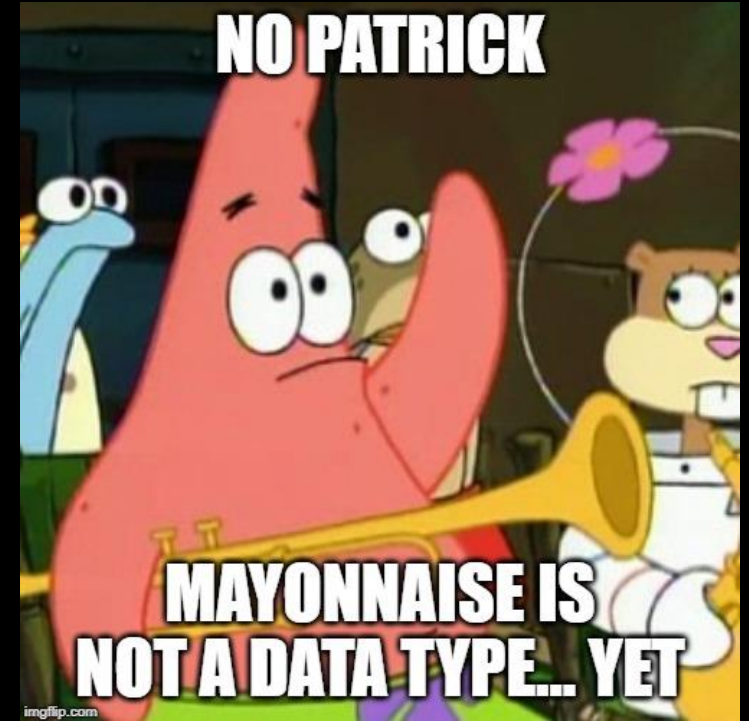
- Let's go experiment with some of what we just saw
  - Assignment statements
  - Declaring variables
  - Different variable names
  - Memory locations
  - (and the famous print statement!)

**Open your  
notebook**

**Click Link:**  
**2. Variables and  
Memory**

# Variable Types

- A **type** is a set of *values* and the *operations* that can be performed on those values.
- *int*: integer
  - ex. 3, 4, 894, 0, -3, -18
- *float*: floating point number
  - ex. -5.6, 7.342, 53452.0, -89.34



# Numerical Type Examples

```
>>> 5 + 2 * 4
```

```
13
```

```
>>> 5.0 + 2 * 4
```

```
13.0
```

```
>>> 30/6
```

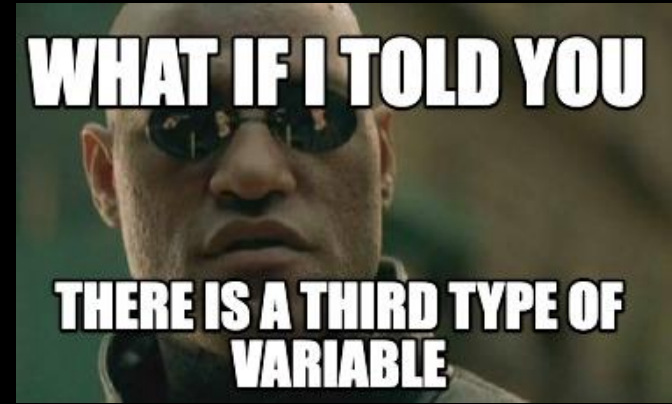
```
5.0
```

```
>>> 30//6
```

```
5
```

# Type: str (pronounced string)

- *str*: string is a sequence of characters
- Start and end with single quotes (') or double quotes (")
  - ex. 'hello', "What is  $10 * (2 + 9)$ ?"
  - Just like writing in English, the quote type must match (i.e. 2 singles or 2 doubles, not 1 of each)



# String Type Examples

```
>>> 'how are you?'
```

```
>>> "short- and long-term"
```

```
>>> '"APS106" is already my favourite course'
```

```
>>> "APS106 stinks'
```

SyntaxError

# More data types?!

- You can get the data type of any object by using the `type()` function

```
>>> x = 5  
>>> print(type(x))  
<class 'int'>
```

- Other Python data types:
  - List
  - Tuple
  - Dictionary
  - Set
  - Boolean

*BUT we will worry about these later...*





# Variables Types

- Let's go experiment with some of what we just saw
  - "int"s vs "float"s
  - The "str" type
  - Using types and variables in expressions
  - Combining our type knowledge with some arithmetic operations

**Open your  
notebook**

**Click Link:**  
**3. Different Types  
of Variables**

# Augmented Assignment Operations

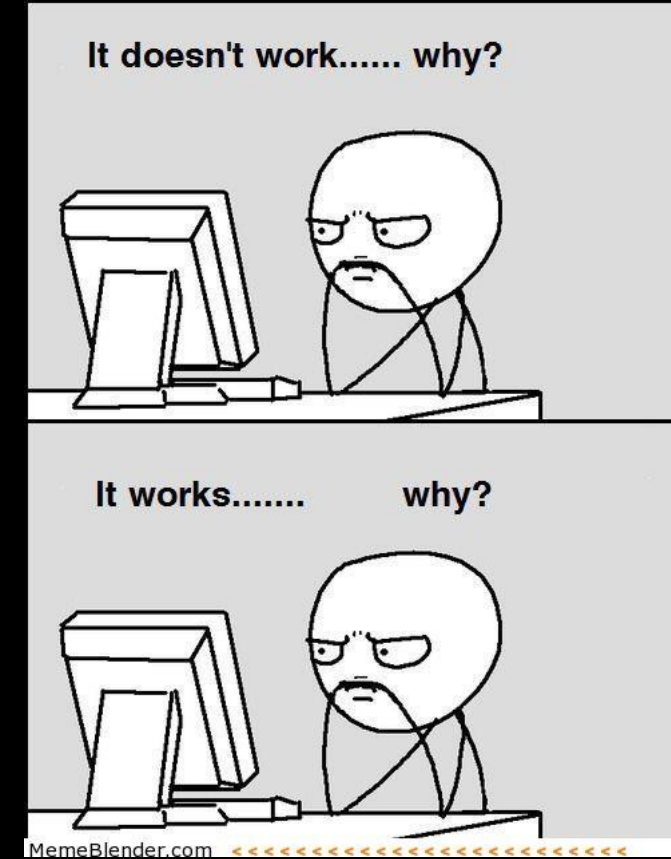
Operator	Expression	Identical Expression	English description
<b>+=</b>	<b>x = 7</b> <b>x += 2</b>	<b>x = 7</b> <b>x = x + 2</b>	<b>x refers to 9</b>
<b>-=</b>	<b>x = 7</b> <b>x -= 2</b>	<b>x = 7</b> <b>x = x - 2</b>	<b>x refers to 5</b>
<b>*=</b>	<b>x = 7</b> <b>x *= 2</b>	<b>x = 7</b> <b>x = x * 2</b>	<b>x refers to 14</b>
<b>/=</b>	<b>x = 7</b> <b>x /= 2</b>	<b>x = 7</b> <b>x = x / 2</b>	<b>x refers to 3.5</b>
<b>//=</b>	<b>x = 7</b> <b>x //= 2</b>	<b>x = 7</b> <b>x = x // 2</b>	<b>x refers to 3</b>
<b>%=</b>	<b>x = 7</b> <b>x %= 2</b>	<b>x = 7</b> <b>x = x % 2</b>	<b>x refers to 1</b>
<b>**=</b>	<b>x = 7</b> <b>x **= 2</b>	<b>x = 7</b> <b>x = x ** 2</b>	<b>x refers to 49</b>

**Open your  
notebook**

**Click Link:**  
**4. Augmented  
Assignment  
Operators**

# 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
  - Reading and interpreting error codes
- Always have a plan!



# Readability Tips (#cleancode)

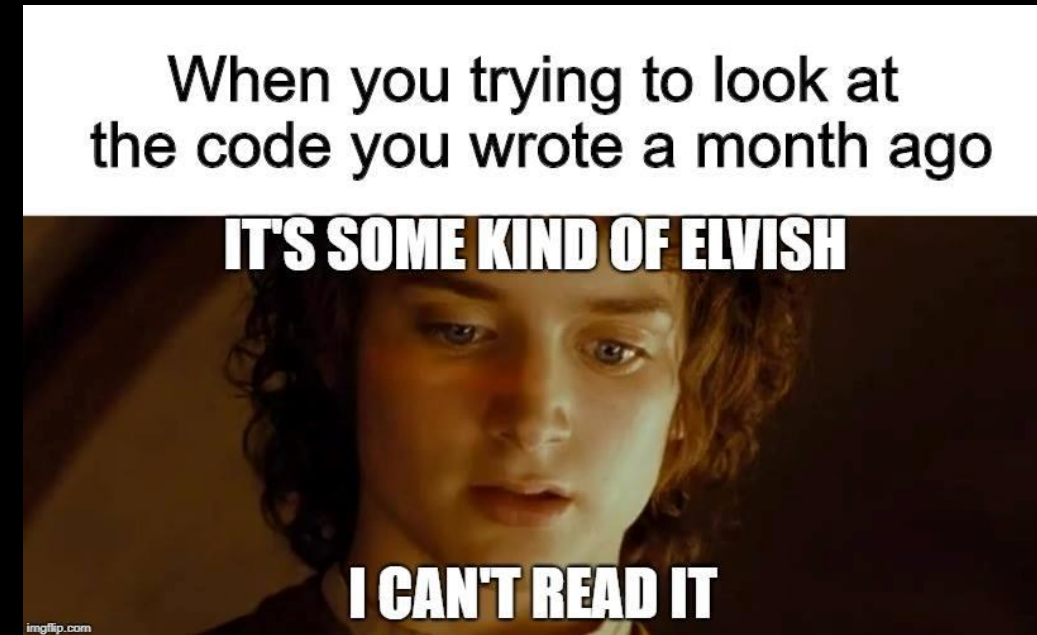
```
>>> canda = cat + panda
```

- 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`
  - `>>> ca = c + p`
- 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 characteristic in service.characteristics! {
        let thisCharacteristic = characteristic 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 enableBytes = NSData(bytes: &enableValue, length: thisCharacteristic.uuid == MovementConfigUUID
                ? MemoryLayout<UInt16>.size : MemoryLayout<UInt8>.size)
            self.sensorTagPeripheral.writeValue(enableBytes 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. Meaningful variable names on the left saved me, but the lack of comments on the right was a nightmare. (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!

# 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:**  
**5. Let's Code!**



## Variables, Expressions and Operators.

Week 1 | Lecture 3 (1.3)

if nothing else, write `#cleancode`