

## introduction to Object Oriented Programming, and the file object.

### Week 5 | Lecture 2 (5.2)

While waiting, download the Jupyter Notebook and text file for today's lecture

Make sure both of these files are in the same folder (such as Downloads or APS106/Lectures)

#### Upcoming

- Lab 3 Due Friday 11:59 PM
- No Lab released Thursday
- Reflection 5 Released Friday 6:00 pm.
- Tutorial (Online), Practical, Office Hour sessions running all week.
- Looking for more practice? Check out the textbook.

if nothing else, write `#cleancode`

# This Week's Content

- **Lecture 5.1**
  - Strings: Conversions, Indexing, Slicing, and Immutability
- **Lecture 5.2**
  - **Introduction to Object-Oriented Programming and the File Object**
- **Lecture 5.3**
  - Term Test 1 Review: Jeopardy!

# Procedural **vs** Object-Oriented

- *"Object-oriented programming (OOP) is a programming paradigm based on the concept of "objects", which may contain data, in the form of fields, often known as attributes; and code, in the form of procedures, often known as methods."*  
— **Wikipedia**
- *"Procedural programming is a programming paradigm, derived from structured programming, based upon the concept of the procedure call. Procedures, also known as routines, subroutines, or functions, simply contain a series of computational steps to be carried out."*  
— **Wikipedia**

# Procedural Programming

**Global Variable**

Pedestrian 1  
x, y Location

**Global Variable**

Pedestrian 2  
x, y Location

**Global Variable**

Pedestrian 3  
x, y Location

```
x_ped1 = 3  
y_ped1 = 5
```

**Global Variable**

Traffic Light 1  
Color

**Global Variable**

Car 1  
x, y location

```
traffic_light = 'red'
```

## Separation of Data and Functions

**Function**

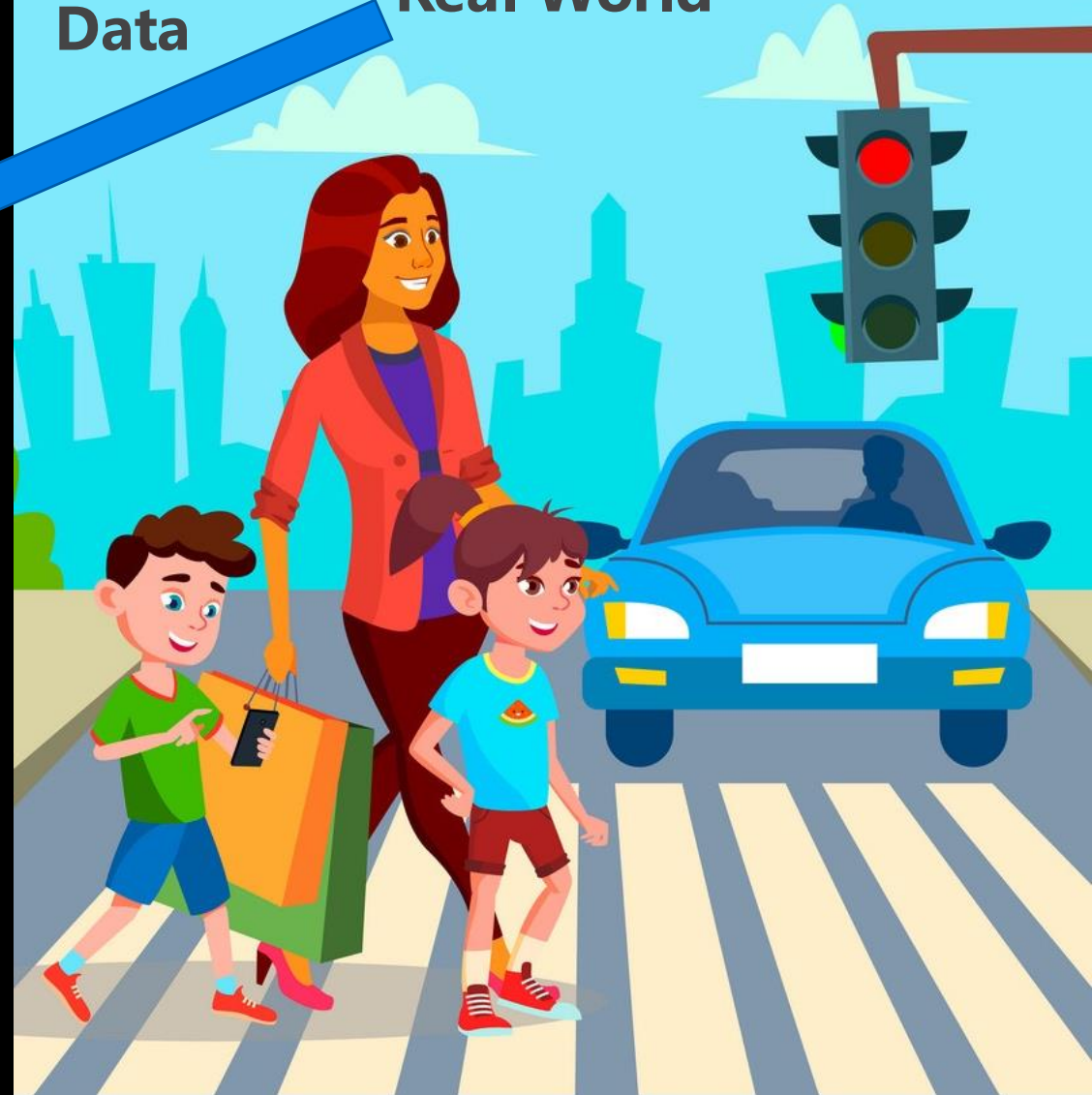
OK to Cross

**Function**

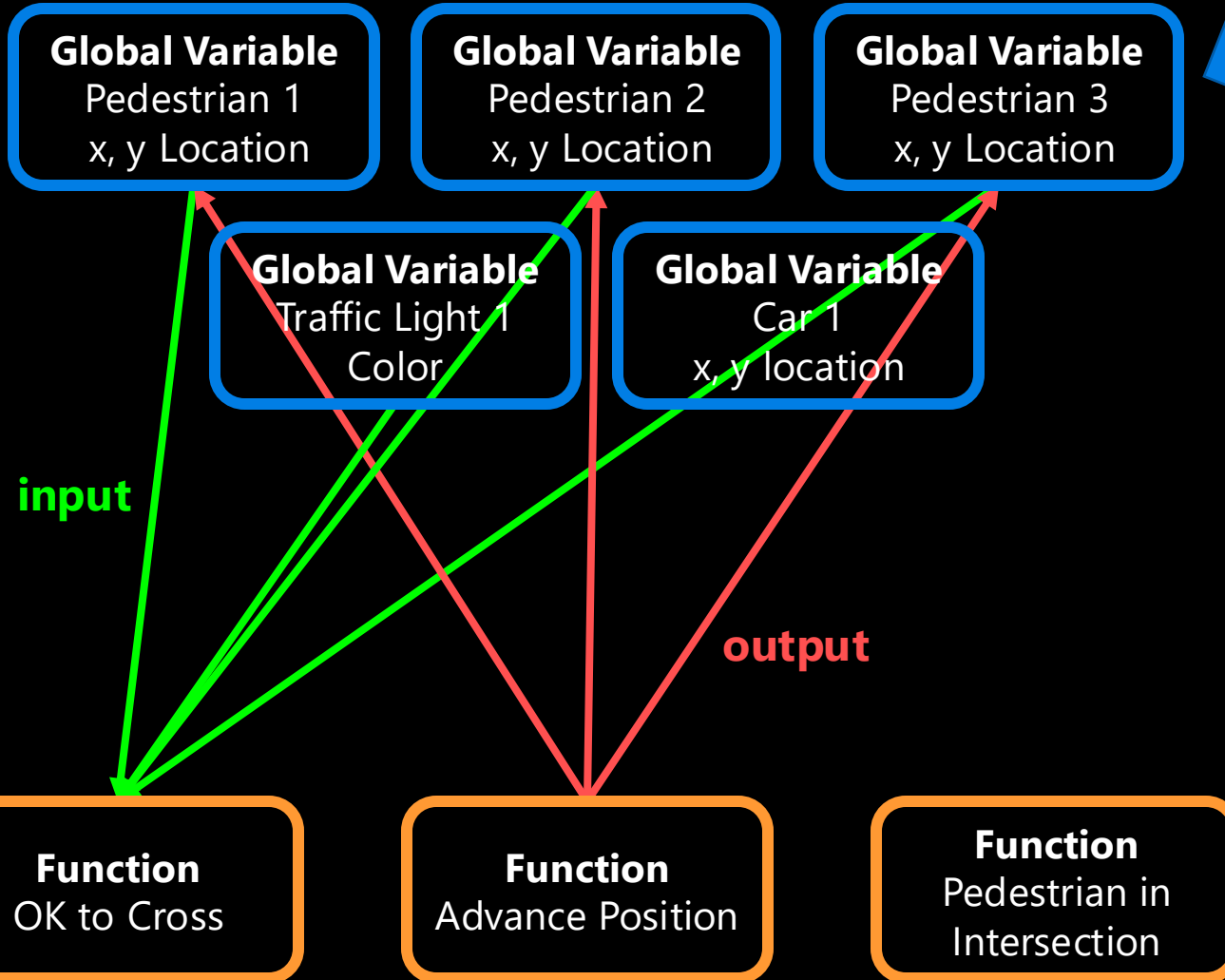
Advance Position

**Function**

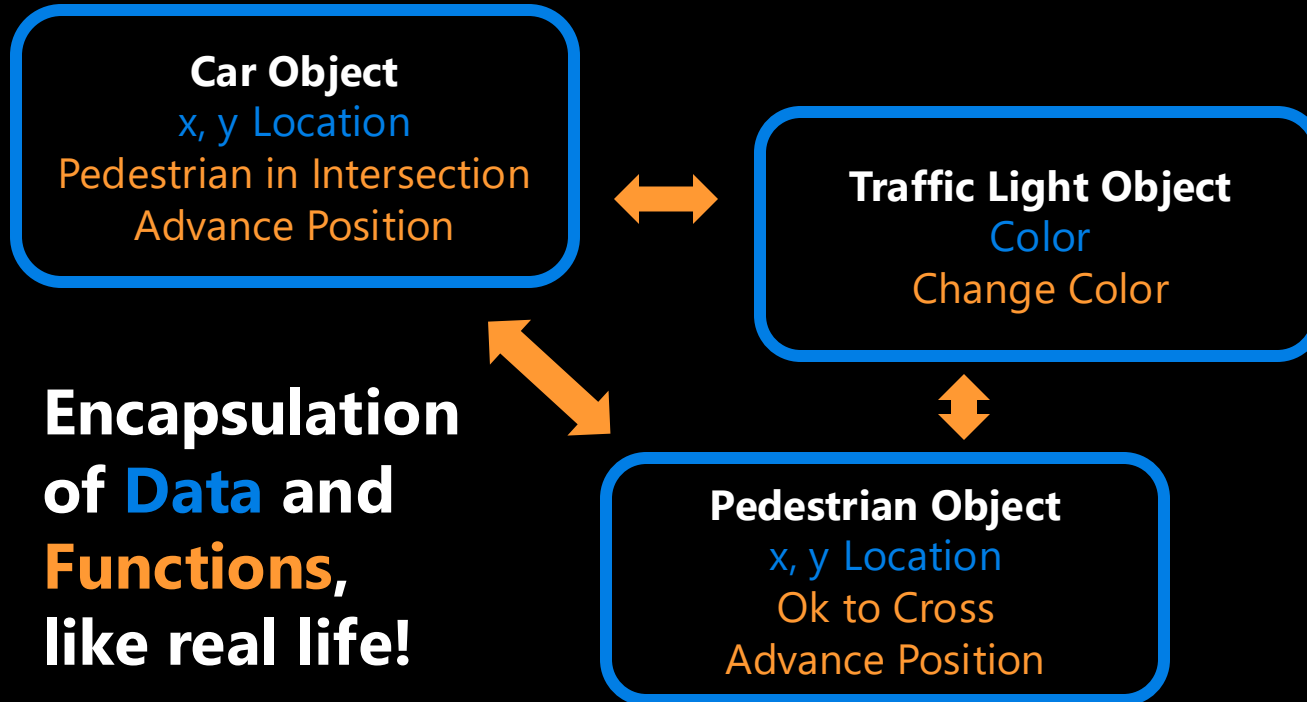
Pedestrian in  
Intersection

**Data****Real World**

# Procedural Programming



# Object-Oriented Programming





# Object-Oriented Programming

- Often, an object definition corresponds to some object or concept in the real world.
- The functions that operate on that object correspond to the ways real-world objects interact.
- Models our real-life thinking
  - Sandwich – ingredients, freshness, etc.
  - Car – model, year, fuel level, forward, reverse etc.
  - Cat – weight, name, colour, scratch, meow, sleep, etc.
  - Turtle Object – x\_position, y\_position, forward, up, left, etc.

# Object-Oriented Programming

## Data Functions

### Procedural

```
def up(y):  
    return y + 1
```

```
def goto(x_new, y_new):  
    return x_new, y_new
```

```
def right(x):  
    return x + 1
```

```
alex_x = 0  
alex_y = 0
```

```
alex_y = up(alex_y)  
alex_x, alex_y = goto(-150, 100)  
alex_x = right(alex_x)
```

```
print(alex_x, alex_y)
```

### Object-Oriented

```
alex = Turtle(0, 0)
```

```
alex.up()  
alex.goto(-150, 100)  
alex.right()
```

```
print(alex.x, alex.y)
```



# Objects in Python

- **Everything in Python is an object.**
- Every value, variable, function, etc., is an object.
- Every time we create a variable we are making a new object.

Is **this** an instance  
of **this** class.



```
>>> isinstance(4, object)  
True
```

```
>>> isinstance(max, object)  
True
```

```
>>> isinstance("Hello", object)  
True
```

# Classes

- A template or blueprint for object types
- An instance of a class refers to one object whose type is defined as the class.
- The words "instance" and "object" are used interchangeably.
- A **Class** is made up of attributes (**data**) and methods (**functions**).

**Data** →

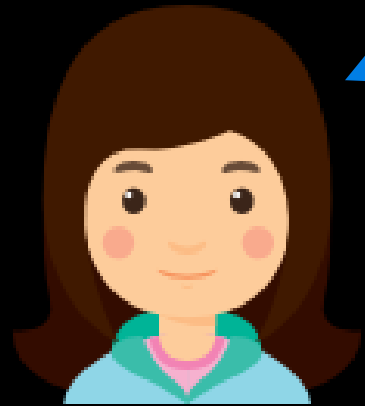
**Attributes**

**Functions** →

**Methods**

```
str.upper('hello')  
'hello'.upper()
```

# Classes

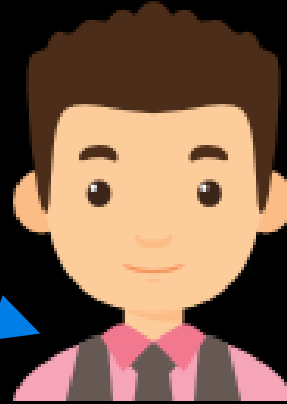


name: June  
age: 34  
city: Ottawa  
gender: she/her

Instances  
(objects) of the  
Person class.



name: Ted  
age: 31  
city: Kingston  
gender: he/him



name: Majid  
age: 28  
city: Toronto  
gender: they/them

Person

name  
age  
city  
gender

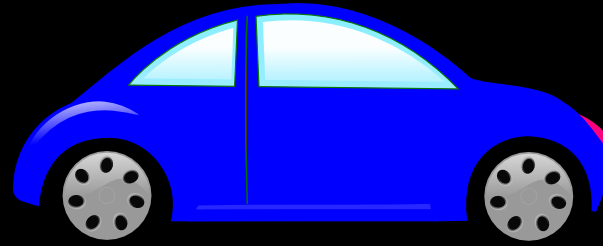
eat  
study  
sleep  
play

# Classes



**model:** Corolla  
**company:** Toyota  
**year:** 1980  
**color:** red

Instances  
(objects) of the  
Car class.



**model:** Model S  
**company:** Tesla  
**year:** 2017  
**color:** blue



**model:** Bus  
**company:** Volkswagen  
**year:** 1976  
**color:** orange

**Car**

**model**  
**company**  
**year**  
**color**

**brake**  
**accelerate**  
**open trunk**

# Classes

Instances  
(objects) of the  
**Turtle** class.



**name:** Lucy  
**x location:** 24  
**y location:** 35



**name:** Vinodh  
**x location:** 134  
**y location:** 45



**name:** Brian  
**x location:** 92  
**y location:** 62

**Turtle**

**name**  
**x location**  
**y location**

**move up**  
**move down**  
**move left**  
**move right**  
**go to**

# We've already been using objects!

**integer**

**is\_integer**  
**as\_integer\_ratio**  
**to\_bytes**  
**from\_bytes**  
**bit\_count**  
...

**string**

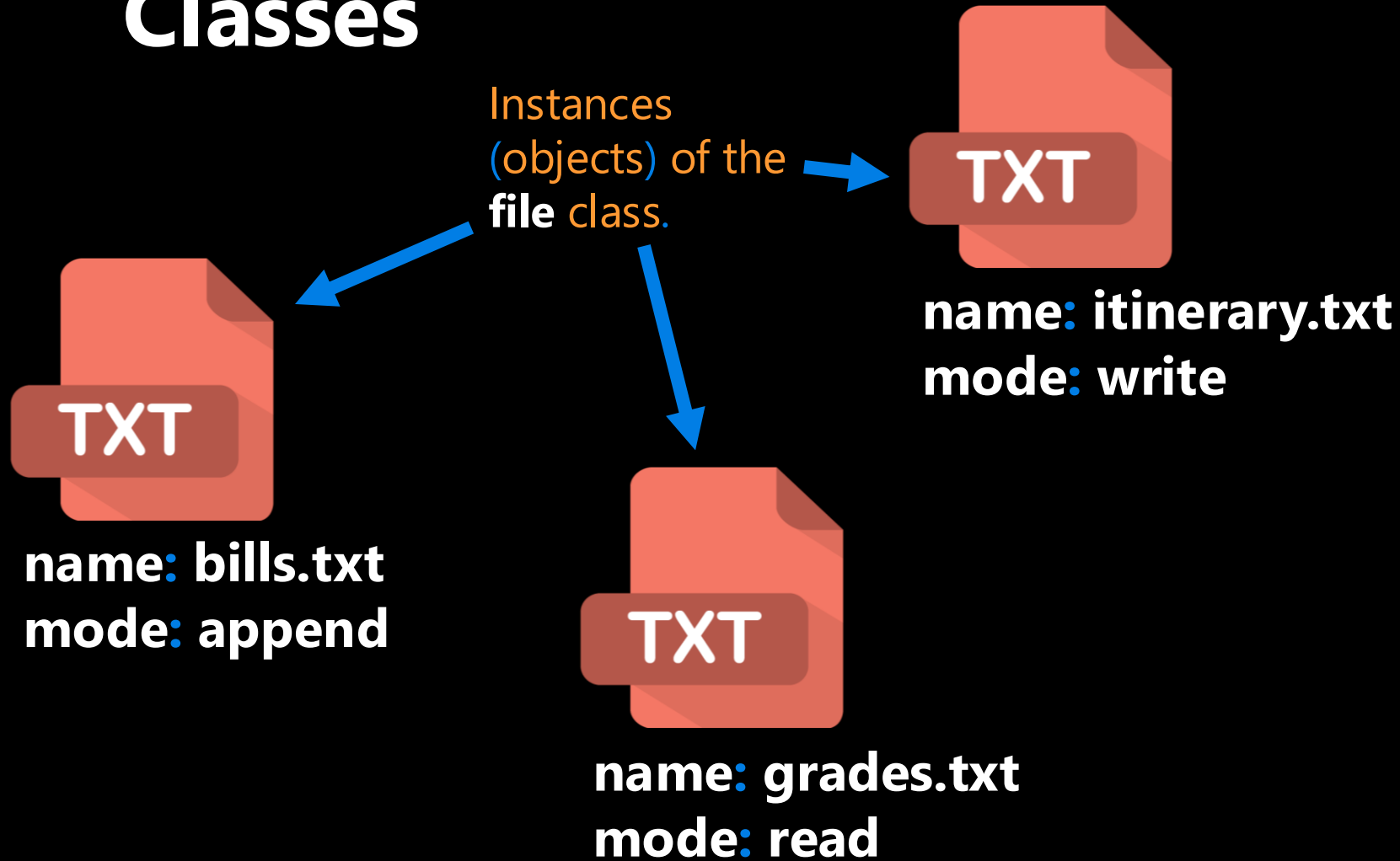
**replace**  
**find**  
**rfind**  
**upper**  
**isupper**  
**islower**  
**capitalize**  
**count**  
...

**math**

**pi**  
**e**  
**inf**  
...

**factorial**  
**ceil**  
**floor**  
**isfinite**  
**log**  
...

# Classes



**File**

**filename**  
**mode (ex: read  
or write)**

**read**  
**readline**  
**write**  
**close**  
**...**



# Classes



```
alex = Turtle(0, 0)
```

```
alex.up()
```

```
alex.goto(-150, 100)
```

```
alex.down()
```

```
print(alex.x, alex.y)
```

We don't need to know these details to be able to use them!

We will learn more about building custom classes later...

```
class Turtle:
```

```
    def __init__(self, x, y):  
        self.x = x  
        self.y = y
```

```
    def up(self):  
        body
```

```
    def goto(self, x, y):  
        body
```

```
    def down(self):  
        body
```

# Let's Code!

- Let's take a look at how this works in Python!
  - Objects in Python
  - One potential ACORN design

## Breakout Session!

Click Link:

**1. Objects in Python**

# Why do we care about files?

- Everything we've done so far is essentially deleted when our program ends
- What if we wanted to store information?



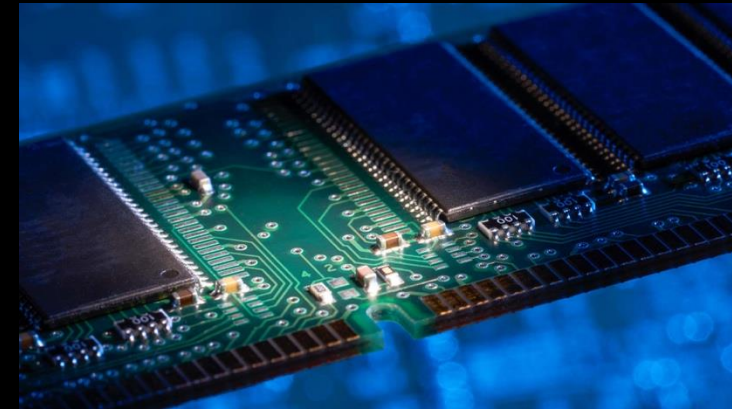
Final ACADEMIC MARKS						
1st Qtr	2nd Qtr	1st Sem	3rd Qtr	4th Qtr	Fnl Exm	Fnl Mrk
A	A		B+	B		B+
B+	C+		C+	C		C+
B+	B+		B+	B		B+
B	D+		C	C+		C
B+	C		C	C+		C+
B+	C		B+	B		B
B	B+		B+	A		B+

# Why work with files?

- Recall from Week 1:

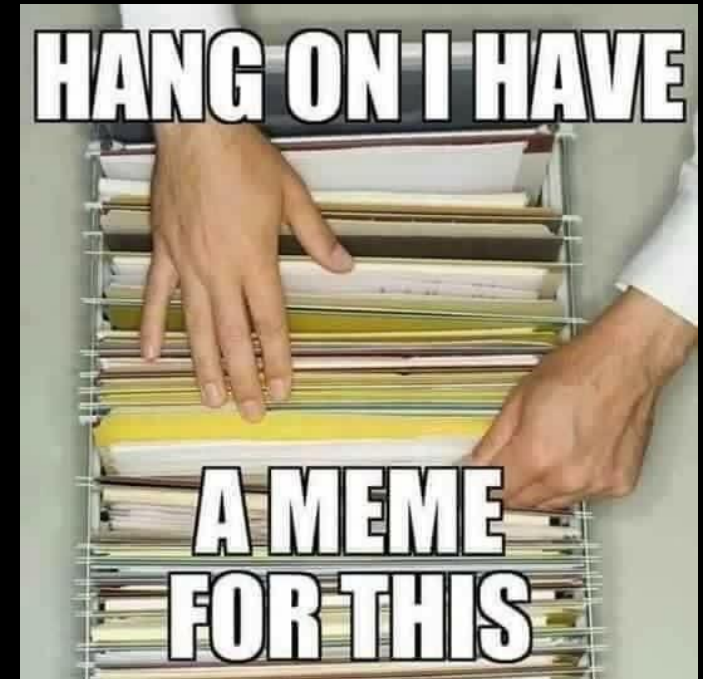
While a program is running, its data is stored in random access memory (RAM). RAM is fast and inexpensive, but it is also volatile, which means that when the program ends data in RAM disappears.

- To make data available the next time the program is started, it must be written to a non-volatile storage medium, such a hard drive, USB drive, cloud storage, or CD-R.



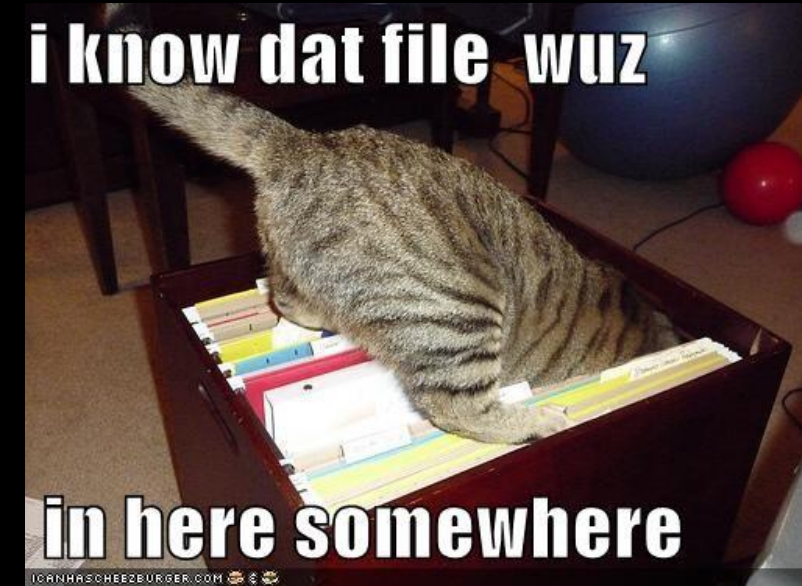
# Why work with files?

- Data on non-volatile storage media are stored in named locations on the media called **files**.
- By **reading** and **writing** files, programs can save information between program runs.



# It's like working with a notebook!

- A file must be **opened**.
- When you are done, it has to be **closed**.
- While the file is open, it can either be **read from or written to**.
- Like a bookmark, the file **keeps track of** where you are reading to or writing from.
- You can read the whole file in its **natural order** or you can **skip** around





# Opening a File

- Python has a built-in function `open` that creates and returns a file object with a connection between the file information on the disk and the program.

- The general form for opening a file is:

```
open(filename, mode)
```

- where **mode** is `'r'` (to open for reading), `'w'` (to open for writing), or `'a'` (to open for appending to what is already in the file).
- **filename** is an external storage location.



# Writing to a File

- The following statement opens the file `test.txt` in write mode `'w'`.

```
myfile = open('test.txt', 'w')
```

- If there is no file named "`test.txt`" on the disk, it will be created. *If there already is one, it will be replaced by the file we are writing.*
- The file information will be assigned to the file object `myfile`.

# Writing to a File

- To write something we need to use the `write` method as shown:

```
myfile.write('CATS!...')
```

- The `write` method does not attach a **new line character** by default.

```
myfile.write('\n')
```

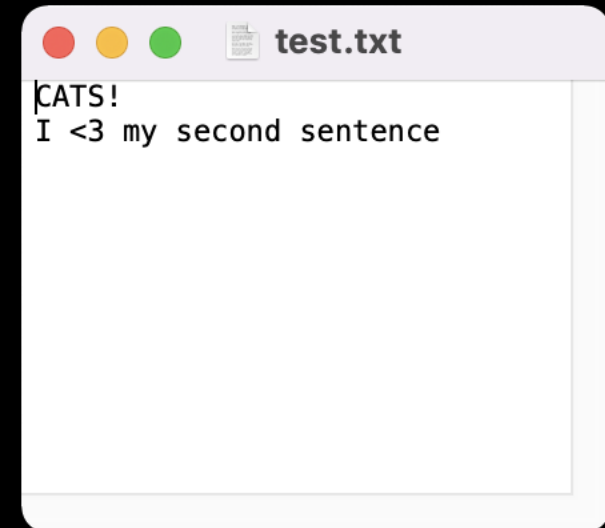
```
myfile.write('I <3 my second line.. \n')
```



- Every time we use `myfile.write()` a string is added to file "test.txt" where we left off.

```
myfile = open('test.txt', 'w')  
myfile.write('CATS!...')  
myfile.write('\n')  
myfile.write('I <3 my second sentence.. \n')
```

```
# and eventually we close our file  
myfile.close()
```



# Closing a File

- Once you have finished with the file, you need to close it:

```
myfile.close()
```

- This tells the system that we are done writing and makes the disk file available for reading or writing by other programs (or by our own program).

# Let's Code!

- Let's take a look at how this works in Python!
  - Writing to your first file
  - Figure out where your files are getting saved!
    - Current working directory
  - Try playing with "w" and "a" mode!

## Breakout Session!

Click Link:  
**2. The File Object**

# CLOSE THE FILE!

- Always close a file that you've opened!
  - Many changes don't occur until **after** the file is closed
  - Leaving open is a waste of a computer's resources
    - Slows down program, uses more space in RAM, impacts performance
  - Other files may treat the file as "open" and won't be able to read the file
  - Could run into limits about how many files you have open
  - Not #cleancode



```
myfile.close()
```



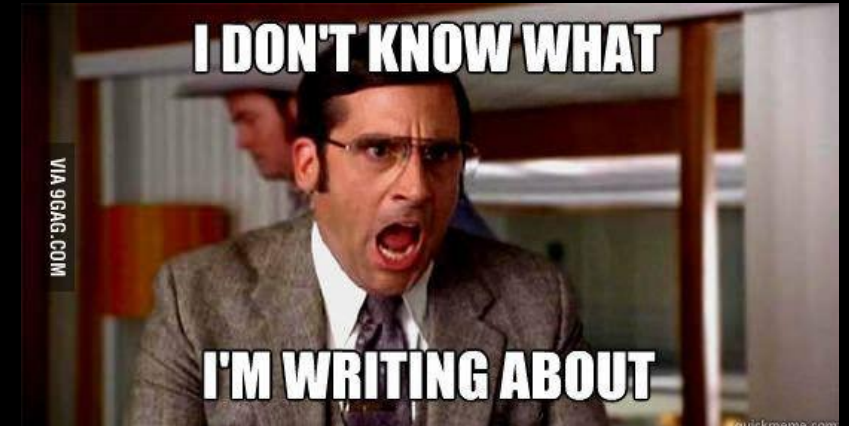
# The "Writing to Files" Recipe

go together

```
# open/create a file  
myfile = open("grades.txt", "w")
```

```
# write to a file  
myfile.write('string')
```

```
# close the file  
myfile.close()
```





# Example: Writing to Files

- How would we write a string to a file?

```
students = 'Kendrick,A+\nDre:C-\nSnoop:B\n'
```

```
# create a file
```

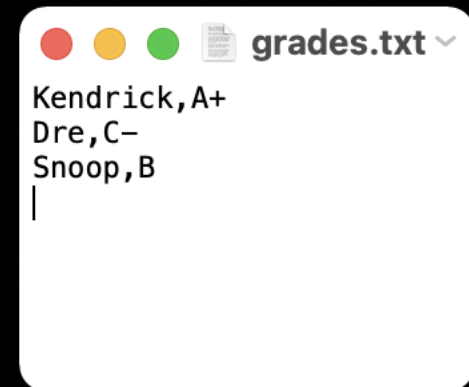
```
myfile = open("grades.txt", "w")
```

```
# store string to the file
```

```
myfile.write(students)
```

```
# close the file
```

```
myfile.close()
```

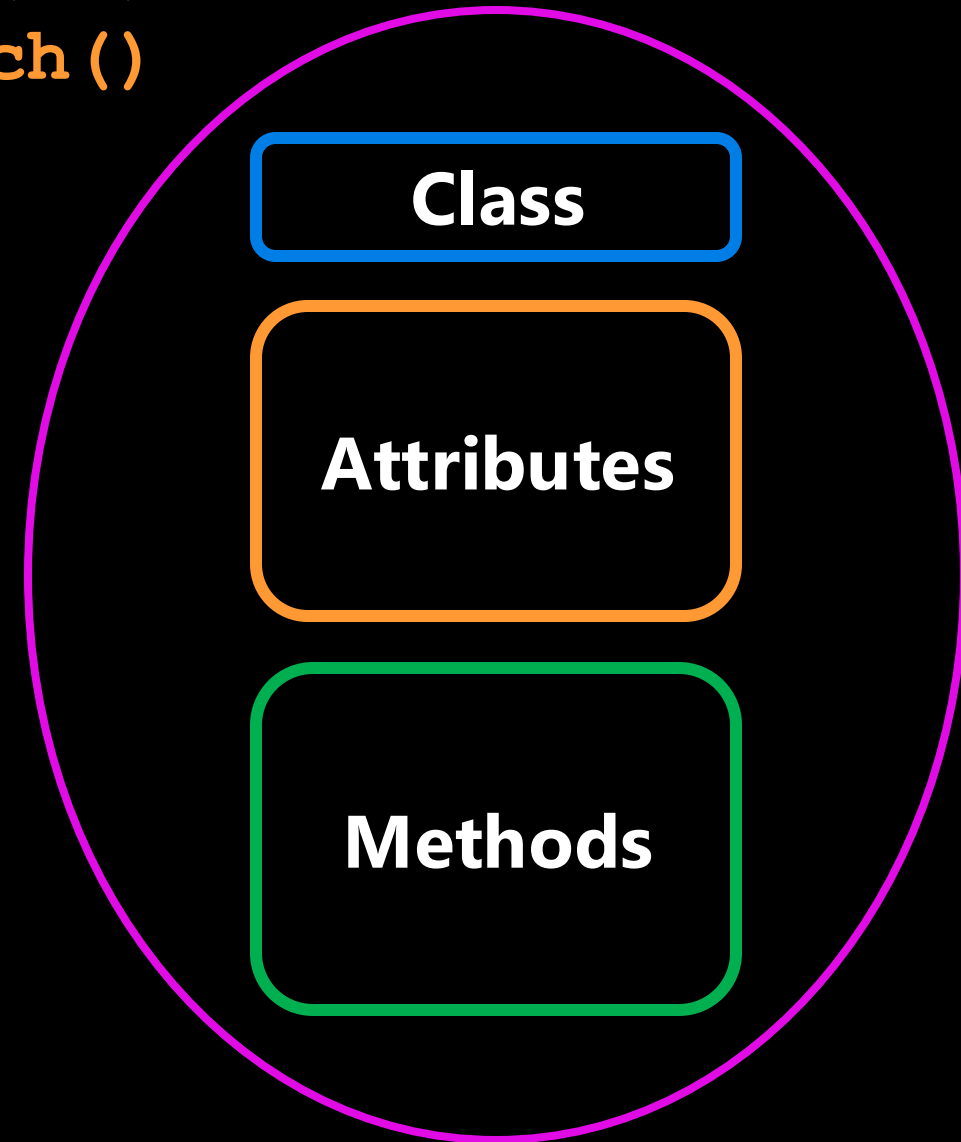


# Encapsulation

```
seb.forward(50)  
kitty.scratch()
```

## Encapsulation

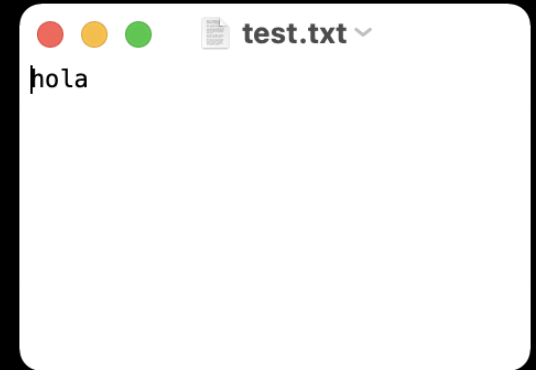
- The core of object-oriented programming is the organization of the program by **encapsulating** related **data** and **functions** together in an object.
- To encapsulate something means to enclose it in some kind of container.
- In programming, encapsulation means keeping **data** and the **code** that uses it in one place and hiding the details of exactly how they work together.



# Encapsulate it!

- For example, each instance of class `file` keeps track of which file on the disk it is reading/writing and where it currently is on that file.
- The class hides the details of how it is done, so we (as programmers) can use it without needing to know how it is implemented

```
f = open('test.txt', 'w')  
f.write('hola')  
f.close()
```



```
print(f)
```

```
<_io.TextIOWrapper name='test.txt' mode='w' encoding='UTF-8'>
```

Can be any name!



## Writing and Reading

```
myfile = open('test.txt', 'w')  
                'w' OR 'a'  
myfile = open('test.txt', 'a')
```

```
myfile.write('CATS!...')
```

# Different ways of Reading a File

- Reading a file is similar to writing a file. First we need to open a file for reading ("**r**");

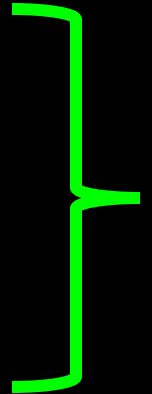
```
myfile = open('test.txt', 'r')
```

- If file doesn't exist? **ERROR**
- Then to read a file we apply one of the following approaches which take advantage of various read methods:
  1. The `read` approach
  2. The `readline` approach
  3. The `for line in file` approach
  4. The `readlines` approach

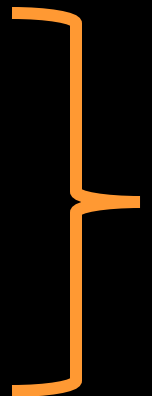
No correct approach!  
Multiple methods to help  
with contexts and purposes

# Different ways of Reading a File

Approach	Code	When to use it
<b>The read approach</b>	<pre>myfile = open(filename, 'r') contents = myfile.read() myfile.close()</pre>	When you want to read the whole file at once and use it as a single string.
<b>The readline approach</b>	<pre>myfile = open(filename, 'r') contents = "" line = myfile.readline() while line != "":     contents += line     line = myfile.readline() myfile.close()</pre>	When you want to process only part of a file. Each time through the loop line contains one line of the file.
<b>The for line in file approach</b>	<pre>myfile = open(filename, 'r') contents = "" for line in myfile:     contents += line myfile.close()</pre>	When you want to process every line in the file one at a time.
<b>The readlines approach</b>	<pre>myfile = open(filename, 'r') lines = myfile.readlines() myfile.close()</pre>	When you want to examine each line of a file by index.



TODAY



FUTURE

# Let's Code!

- Let's take a look at how this works in Python!
  - Different read approaches
    - `read()`
    - `readline()`
    - `for line in file`
    - `readlines()`

**Open your  
notebook**

**Click Link:**  
**3. Reading Files**





- Madlibs is a story game
- A story is written, and a few important words are taken out, replaced by blanks
- The blanks are labelled with their part of speech or other category ("noun", "adjective", "an animal", and so on)
- Replace the categories with specific words without knowing the story
- When all the blanks have been filled in, the story is read out, usually with comic results
- Example: <https://youtu.be/kM9Wuzj4k24>

# Let's Code!

- Let's take a look at how this works in Python!
  - Bringing it all together with Madlibs!

**Open your  
notebook**

**Click Link:**  
**4. Madlibs**

## introduction to Object Oriented Programming, and the file object.

**Week 5** | Lecture 2 (5.2)

**While waiting, open the Jupyter Notebook for today's lecture**

### Upcoming

- Lab 3 Due Friday 11:59 PM
- No Lab released Thursday
- Reflection 5 Released Friday 6:00 pm.
- Tutorial (Online), Practical, Office Hour sessions running all week.
- Looking for more practice? Check out the textbook.

if nothing else, write `#cleancode`