

## Files: Reading and Writing.

**Week 9** | Lecture 2 (9.2)

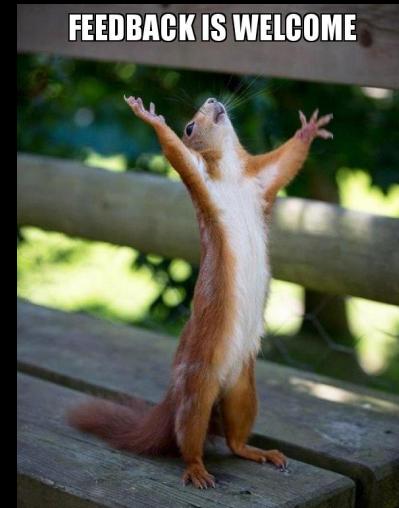
if nothing else, write `#cleancode`

# This Week's Content

- **Lecture 9.1**
  - Advanced Functions and Aliasing
- **Lecture 9.2**
  - Files: Reading and Writing
- **Lecture 9.3**
  - Design Problem! Real Estate

# All The Feedback!

- ⋮ ▾ Lectures
- ⋮  + ⋮
- ⋮  ⋮
- ⋮  ⋮
- ⋮  ⋮
- ⋮  ⋮
- ⋮  ⋮
- ⋮  ⋮
- ⋮  Kinsella & Goodfellow Anonymous Feedback



Seb



Katia



Joseph



Ben

# Motivating Examples

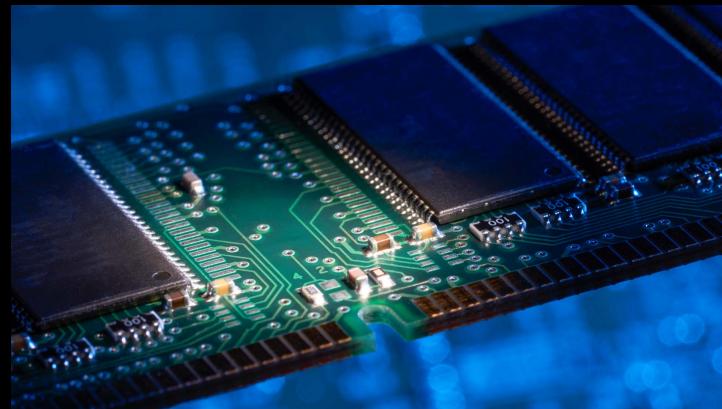
- Everything we've done so far is essentially deleted when our program ends
- What if we wanted to store information?



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

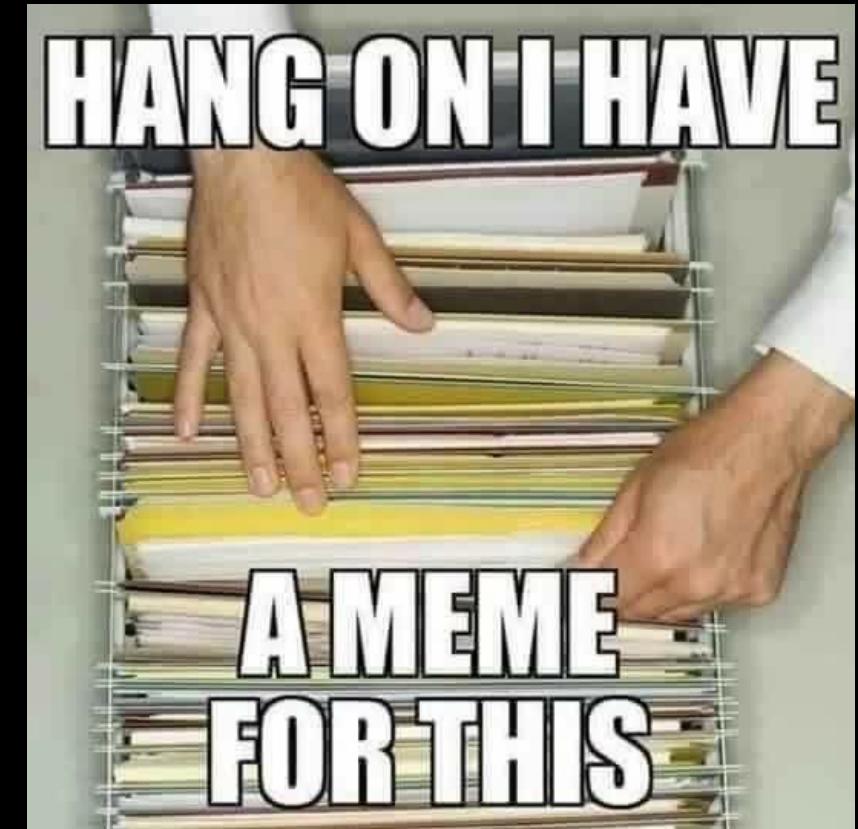
# Why work with files?

- 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 has to be written to a non-volatile storage medium, such a hard drive, USB drive, or CD-R.



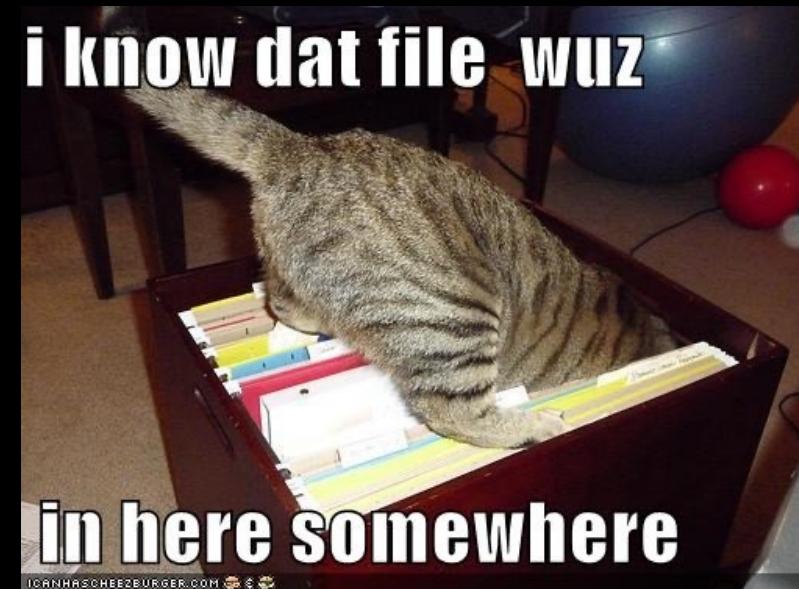
# Why work with files?

- 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 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 will be assigned to the object `myfile` that knows how to get information from the file.

# Writing to a File

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

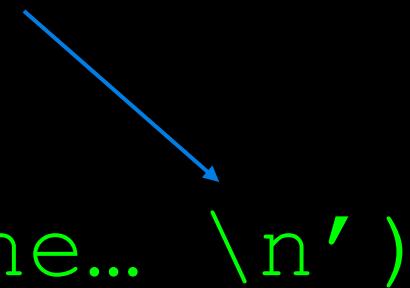
```
myfile.write('CATS!...')
```

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

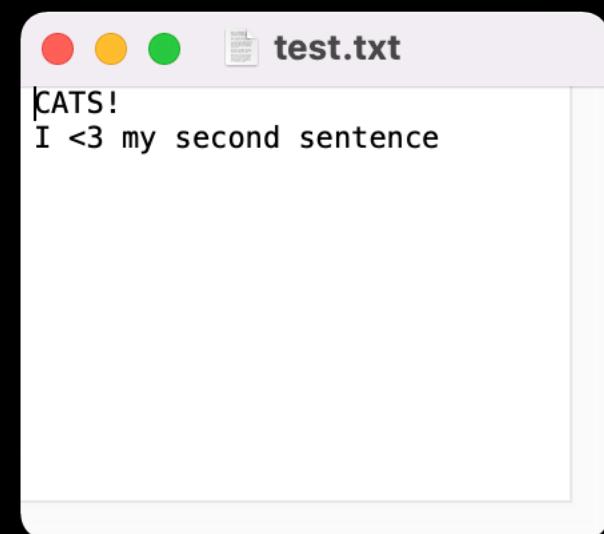
```
myfile.write('\n')
```

```
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 line... \n')
```



# 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!

**Beakout  
Session!**

**Click Link:**  
**1. Write to your  
first file!**

# 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()`



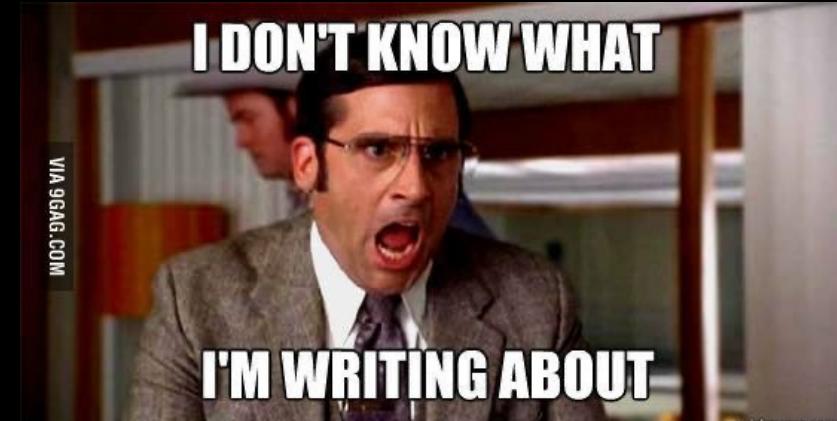
# The “Writing to Files” Recipe

go together

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

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

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



# Example: Writing to Files

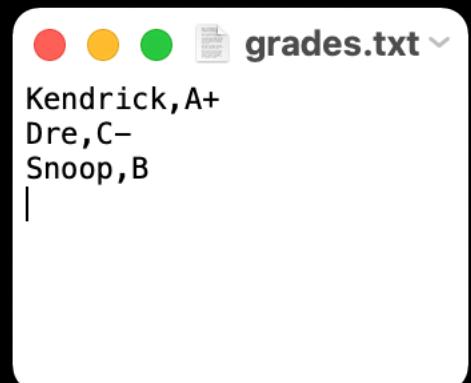
- How would we store a dictionary data structure in a file?

```
students = {'Tina': 'A+', 'Brad': 'C-', 'Sue': 'B'}
```

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

# store dictionary items to the file
for student in students:
    myfile.write(student + ',' + students[student] + '\n')

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



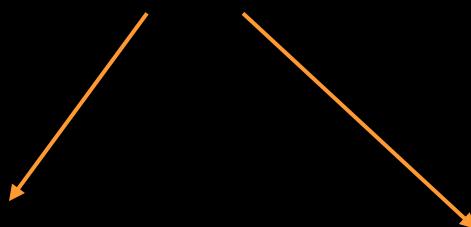
# Let's Code!

- Let's take a look at how this works in Python!
  - Using a loop to write a dictionary to file

**Beakout  
Session!**

**Click Link:  
2. Writing a  
dictionary to file**

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.

# Example: Reading a File

- Now that we have a file "grades.txt" stored. How would we go about retrieving and storing the data into a dictionary?

```
students = {}  
myfile = open("grades.txt", "r")  
  
# read each line of the file  
for line in myfile:  
    # find indices for slicing each line  
    ind1 = line.find(',')  
    ind2 = line.find('\\')  
    name = line[:ind1]  
    grade = line[ind1+1:ind2]  
    students[name] = grade  
  
myfile.close()
```

grades.txt

Kendrick, A+  
Dre, C-  
Snoop, B

>>> students  
{'Sue': 'B', 'Brad': 'C-', 'Tina': 'A+'}

# 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**

# The **with** Statement

- Every call on function `open` should have an accompanying call on the method `close`.
- Python provides a statement `with`, which automatically closes the file when the end of with block is reached.
- The general form of a with statement is as follows:

```
with open(filename, mode) as variable:  
    body
```

# Example: with Statement

- Modifying the previous example, of reading a file into a dictionary, to use the `with` statement.

```
students = {}  
myfile = open("grades.txt", "r")  
with open("grades.txt", "r") as myfile:  
  
    # read each line of the file  
    for line in myfile:  
        ind1 = line.find(',')  
        ind2 = line.find('\\')  
        name = line[:ind1]  
        grade = line[ind1+1:ind2]  
        students[name] = grade  
  
myfile.close()
```

# Let's Code!

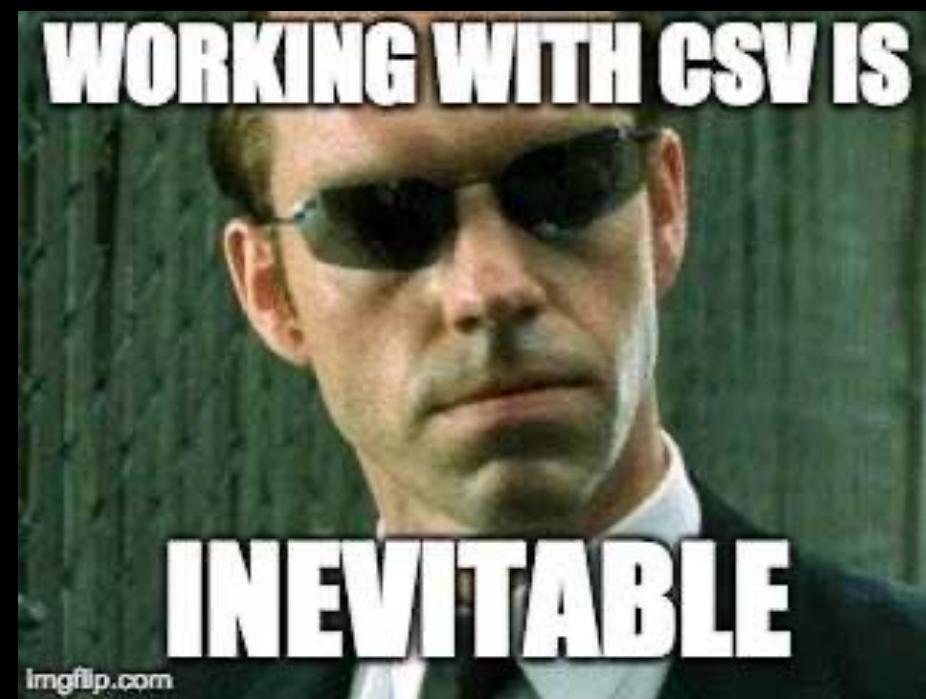
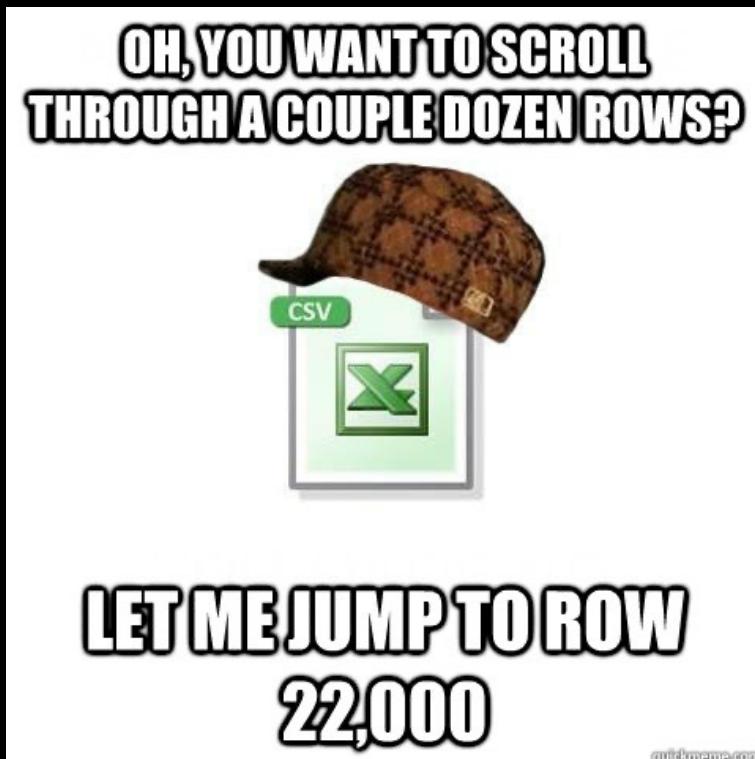
- Let's take a look at how this works in Python!
  - Opening and closing files with the with statement

**Breakout  
Session!**

**Click Link:  
3. The with  
statement**

# Comma Separated Values files

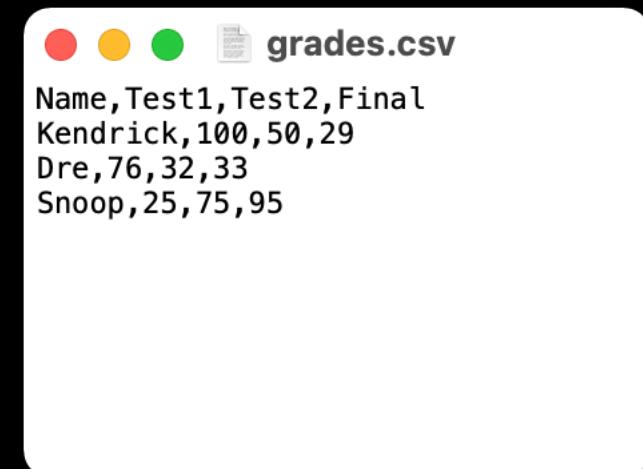
- We use them often in Excel, and other spreadsheet software
- Remember our old friend MS Excel? They work with Python too...



# CSV Files

- Text data is commonly organized in a spreadsheet format using columns and rows.
- A common way to do this is to use a comma-separated value (CSV) file format that uses commas to separate data items, called fields.

Name	Test1	Test2	Final
Kendrick	100	50	29
Dre	76	32	33
Snoop	25	75	95



# Example: Opening a CSV File

- Let's see what happens when we try to read the CSV file, 'grades.csv' using the file reading techniques discussed earlier.

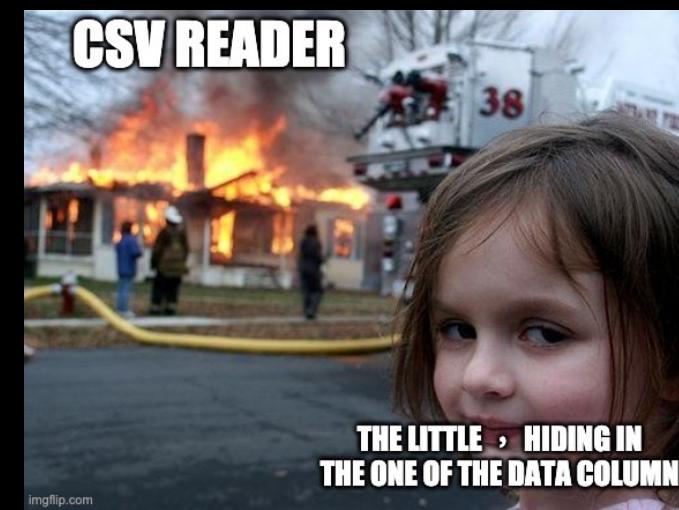
```
with open('grades.csv', 'r') as file:  
    contents = file.read()
```

```
>>> contents  
'Name,Test1,Test2,Final\nKendrick,100,50,29\nDre,76,32,33\nSnoop,25,75,95\n'
```

- How can we use this to obtain column and row information?

# Reading CSV Files

- The CSV module is a powerful solution developed for working with CSV files.
- Reading of CSV files is done using the CSV reader. You can construct a reader object using `csv.reader()` which takes the file object as input.
- The reader object can be used to iterate through the contents of the CSV file, similarly to how a file object was used to iterate through the contents in a text file.



# Example: Reading a CSV File (open)

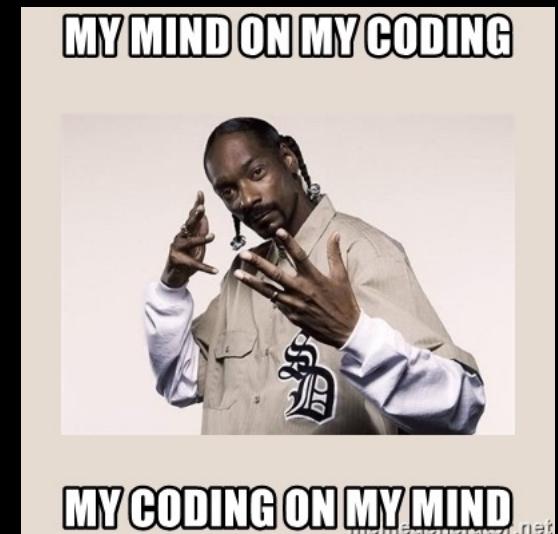
- Read each row of a CSV file using open

```
import csv
csvfile = open("grades.csv", "r")
grades_reader = csv.reader(csvfile)

row_num = 1
for row in grades_reader:
    print('Row #', row_num, ':', row)
    row_num += 1

 csvfile.close()

Row # 1 : ['Name', 'Test1', 'Test2', 'Final']
Row # 2 : ['Kendrick', '100', '50', '29']
Row # 3 : ['Dre', '76', '32', '33']
Row # 4 : ['Snoop', '25', '75', '95']
```



# Example: Reading a CSV File (with)

- Read each row of a CSV file using with

```
import csv
with open('grades.csv', 'r') as csvfile:
    grades_reader = csv.reader(csvfile)

    row_num = 1
    for row in grades_reader:
        print('Row #', row_num, ':', row)
        row_num += 1

Row # 1 : ['Name', 'Test1', 'Test2', 'Final']
Row # 2 : ['Kendrick', '100', '50', '29']
Row # 3 : ['Dre', '76', '32', '33']
Row # 4 : ['Snoop', '25', '75', '95']
```

MY MIND ON MY CODING



MY CODING ON MY MIND

meme-generator.net

# Writing CSV Files

- To write to the file we would first need to create a CSV writer object, `csv.writer()`, which similar to how we made a, CSV reader object.
- Once the CSV writer object is created, we can use the `writerow()` method to populate it with data.
- The `writerow()` method can only write a single row to the file at a time.

# Example: CSV Files

- In the previous grade example there were a few marking errors on the final exam and both John and Mark should have received a higher grade. Update the grades using the CSV writerow() method.

```
import csv

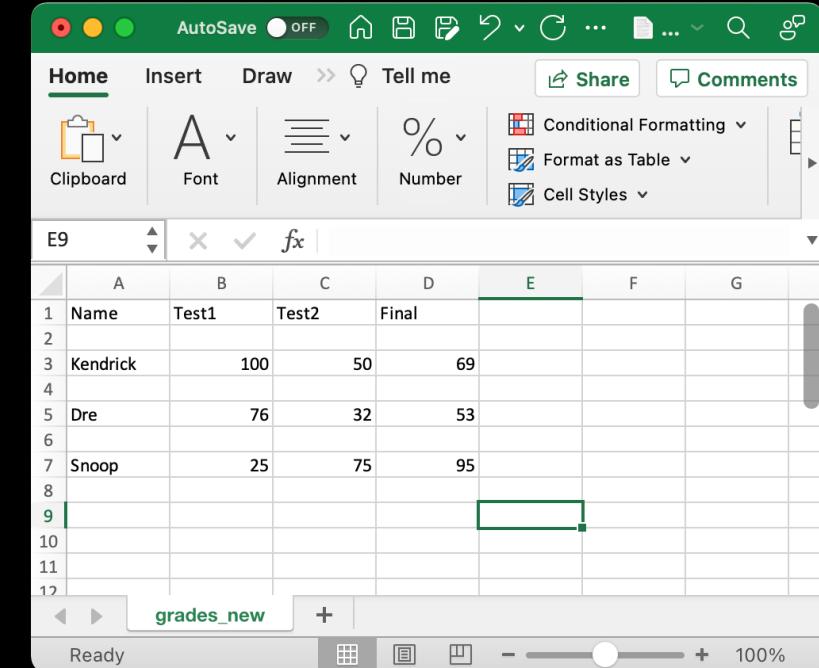
grades = [['Name', 'Test1', 'Test2', 'Final'],
          ['Kendrick', '100', '50', '69'],
          ['Dre', '76', '32', '53'],
          ['Snoop', '25', '75', '95']]


with open('grades_new.csv', 'w') as csvfile:
    grades_writer = csv.writer(csvfile)

    for row in grades:
        grades_writer.writerow(row)
```

# Opening CSV File in Excel

- In the previous example we created a CSV file which can be opened in any commonly used spreadsheet software (e.g. Excel).
- In some cases a formatting error may occur, probably due to the difference between newlines and carriage returns in Windows vs. Mac/Linux.
- To correct this error in formatting, we will need to prevent the new line from forming. Add the parameter `newline=''`, and that should resolve the problem.



	A	B	C	D	E	F	G
1	Name	Test1	Test2	Final			
2							
3	Kendrick		100	50	69		
4							
5	Dre		76	32	53		
6							
7	Snoop		25	75	95		
8							
9							
10							
11							
12							

```
with open('grades_new.csv', 'w', newline='') as csvfile:
```

# Summary: Reading Files

```
#####
# reading a file using standard approach
#####

# open communication to a file (to read)
myfile = open('grades.csv', 'r')

# read from file
L = myfile.readlines()
for row in L:
    print(row, end = "")

myfile.close()
```

Requires find()  
method to  
obtain data!

```
#####
# reading a file using CSV Module
#####

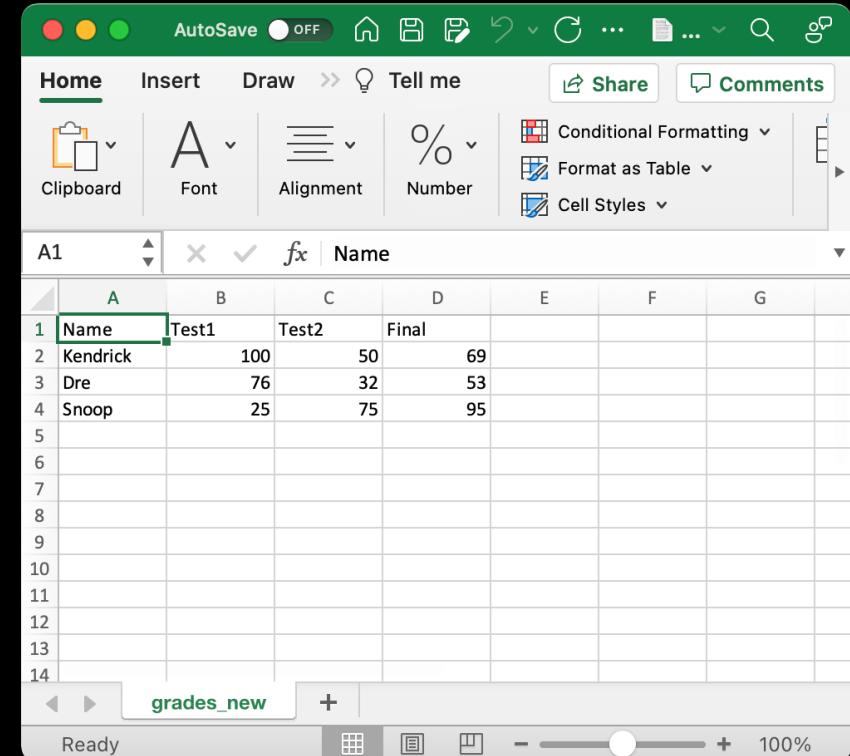
import csv

# open communication to a file (to read)
myfile = open('grades.csv', 'r')

# read from file
csv_reader = csv.reader(myfile)
for row in csv_reader:
    print(row)

myfile.close()
```

Requirement: "grades.csv"



	A	B	C	D	E	F	G
1	Name	Test1	Test2	Final			
2	Kendrick	100	50	69			
3	Dre	76	32	53			
4	Snoop	25	75	95			
5							
6							
7							
8							
9							
10							
11							
12							
13							
14							

# Summary: Writing Files

```
#####
#writing to a file using standard approach
#####

# open communication to a file (to write)
myfile = open('new_grades.csv', 'w')

# write to file
for row in grades:
    for col in row:
        myfile.write(col + ',')
    myfile.write('\n')

myfile.close()
```

Requirement: grades needs to be defined

```
grades = [['Name', 'Test1', 'Test2', 'Final'],
          ['Kendrick', '100', '50', '69'],
          ['Dre', '76', '32', '53'],
          ['Snoop', '25', '75', '95']]
```

```
#####
#writing to a file using CSV Module
#####
import csv
```

```
# open communication to a file (to write)
myfile = open('new_grades.csv', 'w')

# write to file
grades_writer = csv.writer(myfile)
for row in grades:
    grades_writer.writerow(row)

myfile.close()
```

# Let's Code!

- Let's take a look at how this works in Python!
  - Reading and Writing to CSV Files
  - Parsing through CSV Files

**Open your  
notebook**

**Click Link:  
4. CSV Files**

## Files: Reading and Writing.

**Week 9** | Lecture 2 (9.2)

if nothing else, write `#cleancode`