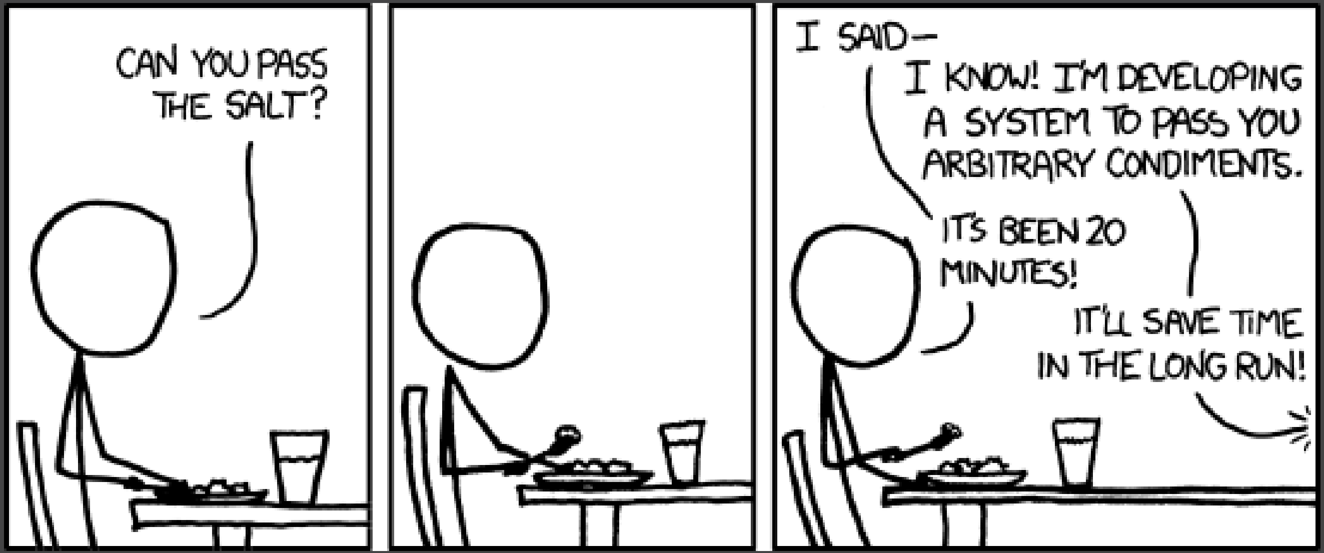
APS106

UNIVERSITY OF TORONTO

Advanced Files: CSVs and Modules

Week 11 | Lecture 2 (11.2)


CAN YOU PASS THE SALT?
I SAID—
I KNOW! I'M DEVELOPING A SYSTEM TO PASS YOU ARBITRARY CONDIMENTS.
IT'S BEEN 20 MINUTES!
IT'LL SAVE TIME IN THE LONG RUN!

While waiting, unzip the .zip file for today's lecture, open that folder and the notebook inside.

Upcoming

- Lab 7 due this Friday 11:59 pm.
- Reflection 11 Released Friday 6:00 pm.
- Tutorial (in-person **AND** online) running all week.
- Practical sessions running all week.
- Office Hours (in-person **AND** online) running all week

if nothing else, write #cleancode

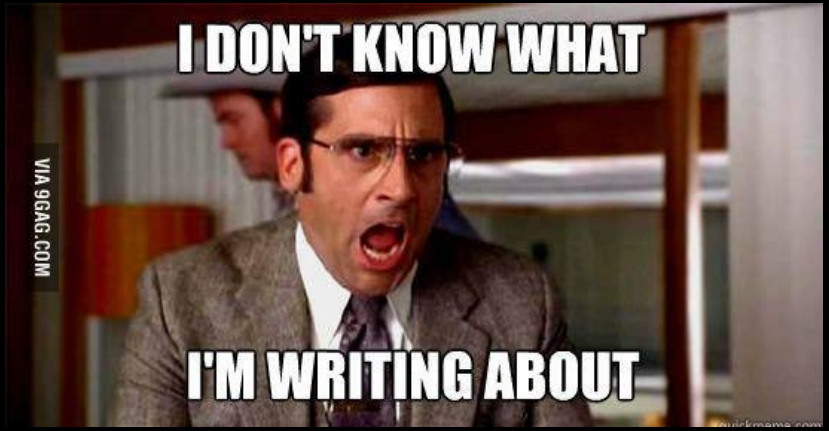
This Week's Content

- **Lecture 11.1**
  - More OOP! Encapsulation and Examples
- **Lecture 11.2**
  - **Advanced Files, CSVs, and Modules**
- **Lecture 11.3**
  - Design Problem

The "Writing to Files" Recipe

go together

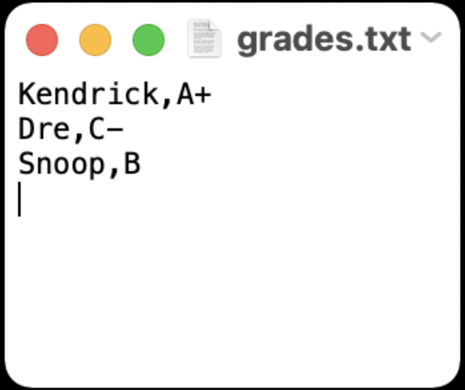
```
# open/create a file
myfile = open("grades.txt", "w")


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


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



Example: Writing to Files

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

```
students = {‘Kendrick': 'A+', ‘Dre': 'C-', ‘Snoop': '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()
```



```
Kendrick,A+
Dre,C-
Snoop,B
```

Let's Code!

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

Open your notebook

Click Link:
1. Writing a dictionary to file

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

The read approach	myfile = open(filename, 'r')
	contents = myfile.read()
	myfile.close()
		When you want to read the whole file at once and use it as a single string.

The readline approach	myfile = open(filename, 'r')
	contents = ''
	line = myfile.readline()
	while line != '':
	    contents += line
	    line = myfile.readline()
	myfile.close()
		When you want to process only part of a file. Each time through the loop line contains one line of the file.

The for line in file approach	myfile = open(filename, 'r')
	contents = ''
	for line in myfile:
	    contents += line
	myfile.close()
		When you want to process every line in the file one at a time.

The readlines approach	myfile = open(filename, 'r')
	lines = myfile.readlines()
	myfile.close()
		When you want to examine each line of a file by index.

WEEK 5

TODAY

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
{'Kendrick': A+, 'Dre': 'C-', 'Snoop': 'B'}
```

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:
2. More advanced file reading

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

Beakout Session!

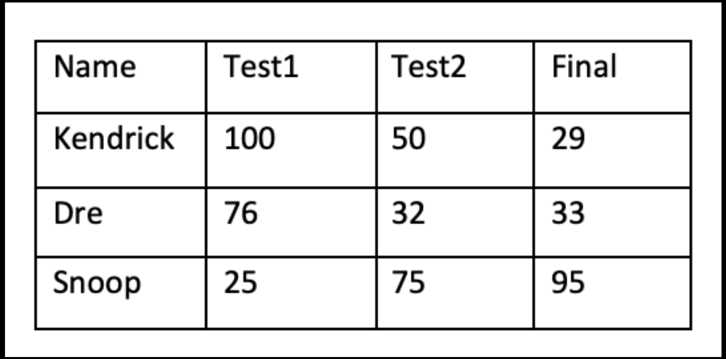
Click Link:
3. The with statement

Comma Separated Values files

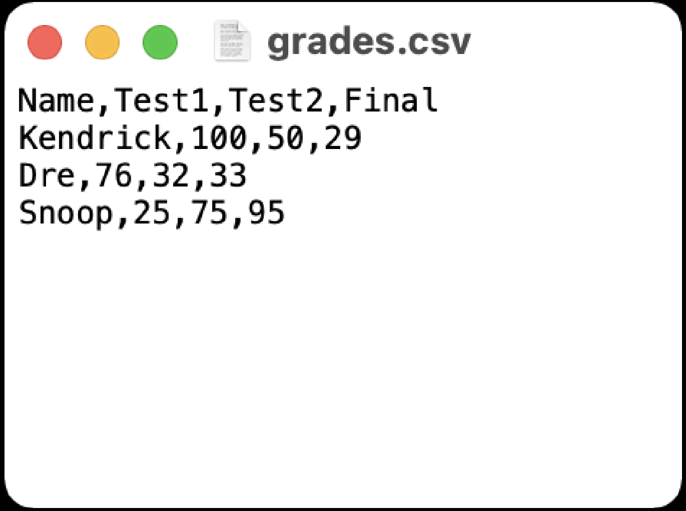
- We use them often in Excel, and other spreadsheet software
- Remember our old friend MS Excel? They work with Python too…

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



grades.csv

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

Example: Opening a CSV File

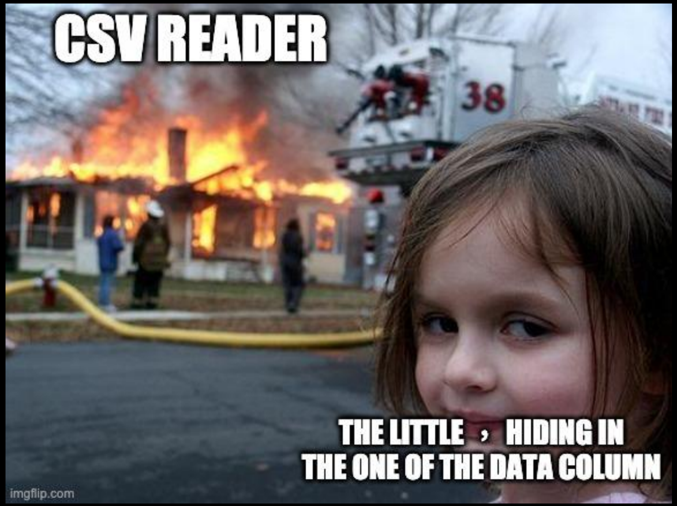
- Let's see what happens when we try to read the CSV file, `'grades.csv'` using the file reading techniques discussed earlier.

```
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.

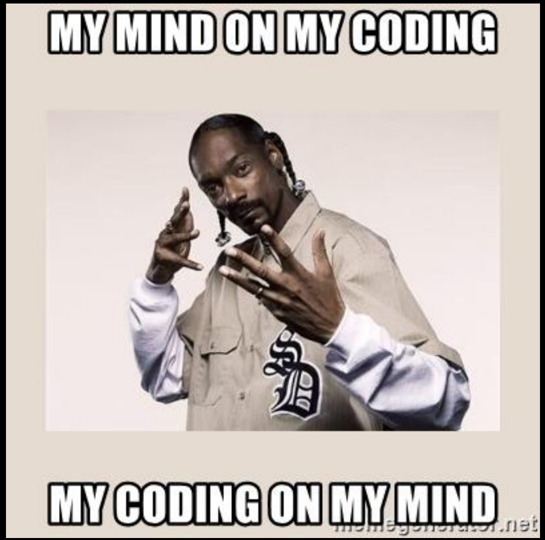


Example: Reading a CSV File (open/close)

- Read each row of a CSV file using open/close

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

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']
```

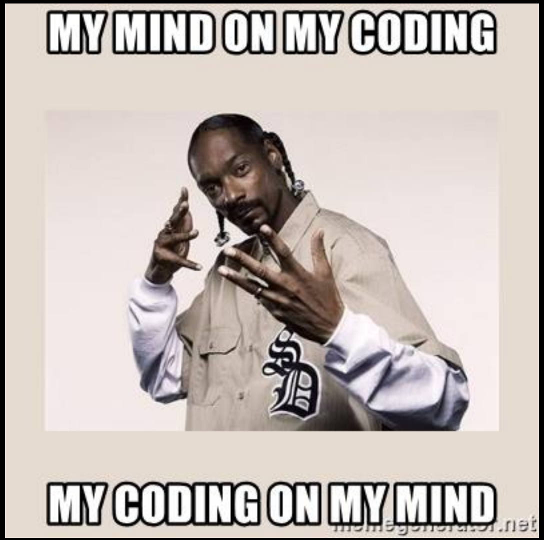


Example: Reading a CSV File (with)

- Read each row of a CSV file using with

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

    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']
```



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.

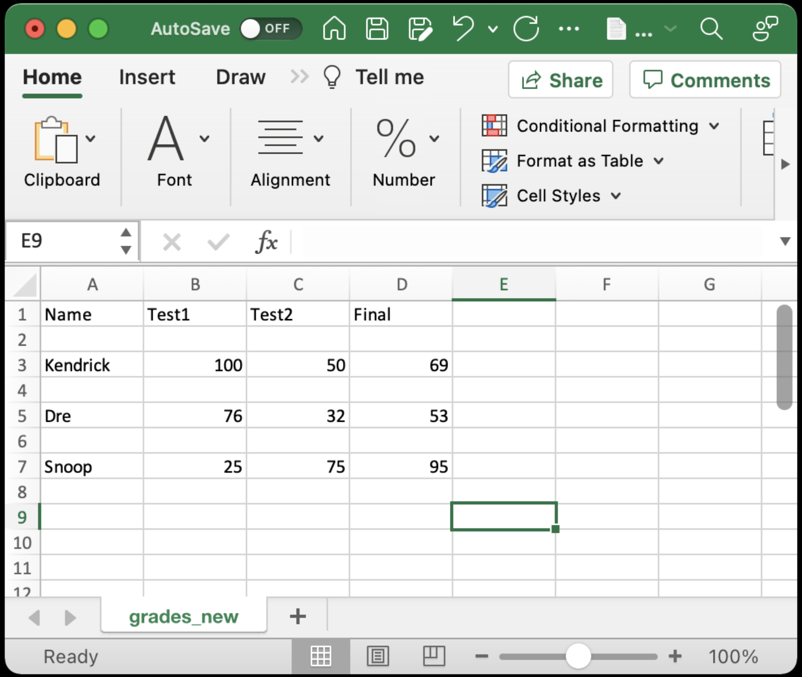
Example: CSV Files

- In the previous grade example there were a few marking errors on the final exam and both Kendrick and Dre 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 myfile:
    grades_writer = csv.writer(myfile)

    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							

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

Let’s Code!

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

Open your notebook

Click Link:
4. CSV Files

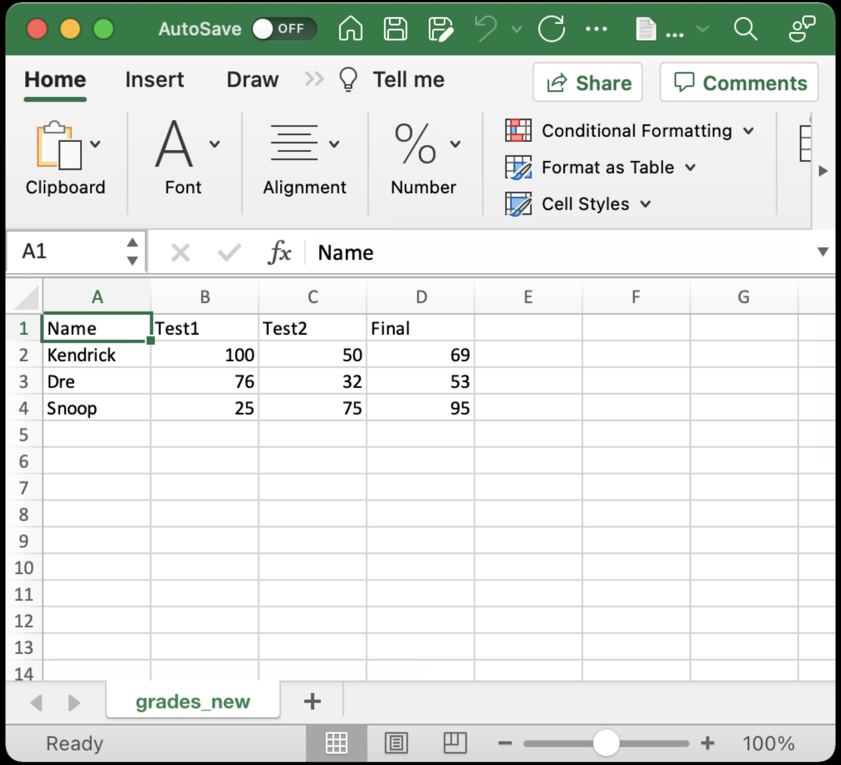
Summary: Reading Files

Requirement: “grades.csv”

```
###############################
 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()
```



	A	B	C	D
1	Name	Test1	Test2	Final
2	Kendrick	100	50	69
3	Dre	76	32	53
4	Snoop	25	75	95

Summary: Writing Files

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 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()
```

```
#################################
#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()
```

Modules

- A module is simply a file containing Python definitions (functions, classes, variables) that can be imported into your script.
- By organizing code into modules, we:
  - Keep code maintainable and organized
  - Reuse functions or classes across multiple projects
  - Avoid cluttering a single file with too much logic

- Tip: Any file named *something*.py can be considered a module.

Importing Modules

- `import module_name`
  - Imports the entire module.
  - You call functions with module_name.function().

```
import math
print(math.sqrt(16))  # 4.0
```

- `from module_name import something`
  - Imports only something (function, class, or variable)
  - You can access something without needing module_name.something
- `from module_name import *`
  - Imports everything from that module directly into your file (and scope or ‘namespace’)
  - Use sparingly to avoid name conflicts

UNIVERSITY OF TORONTO

Why Modules?

- Modularity: Split large applications into smaller, manageable pieces
  - Think back our fancy dictionary printing function called `dict_print` that we imported from our custom `utils` module.
- Reusability: Reuse functions across multiple projects (don’t rewrite the wheel!)
  - Now you can import `dict_print` into any new python file, instead of needing to copy and paste the function into each file to use it
- Maintainability: Isolate functionality in separate files for easier updates
  - Improve your `utils` module or add more fancy printing functions to utils without breaking any of your existing files
- Collaboration: Multiple developers can work on different modules in parallel

Build Your Own Modules

- **Step 1**: Create a .py file with functions, variables, etc.

```
#inside mymodule.py

pi = 3.14

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

- **Step 2**: Import your module into another file

```
import mymodule

my_pi = mymodule.pi

print(mymodule.and_one(my_pi) #prints 4.14
```

Let’s Code!

- Let’s take a look at how this works in Python!
  - Importing modules
  - Renaming modules and controlling our namespace
  - Building our own custom modules

Open your notebook

Click Link:
5. Modules

APS106

UNIVERSITY OF TORONTO

Advanced Files: CSVs and Modules

Week 11 | Lecture 2 (11.2)

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

Upcoming

- Lab 7 due this Friday 11:59 pm.
- Reflection 11 Released Friday 6:00 pm.
- Tutorial (in-person **AND** online) running all week.
- Practical sessions running all week.
- Office Hours (in-person **AND** online) running all week

if nothing else, write #cleancode