Lab Practical 2: Introduction to Data Handling

This week's practical will have you working with CSV and XML files containing movie data extracted from the Open Movie Database (http://www.omdbapi.com/).

In Parts 1-5 of this Practical you will use Jupyter Notebook to write Python code to read and write movie data CSV and XML files. In Part 6 and 7, you will create a Python script from the code that you write in the earlier parts of this Practical and will test it on the command line.

Setup

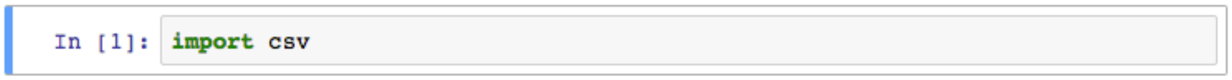
If you have not set-up your account with Anaconda, please follow the instructions in Lab Practical 1 to do the set-up.

Part 1 Reading CSV Files with csv.reader

If this is your first time using Jupyter Notebook, go back to Lab Practical 1 and run through **Part 2 Introduction to Jupyter Notebook** steps **a.** through **f**. This will introduce you the set-up of Jupyter Notebook, and how to run Python code in Notebook cells.

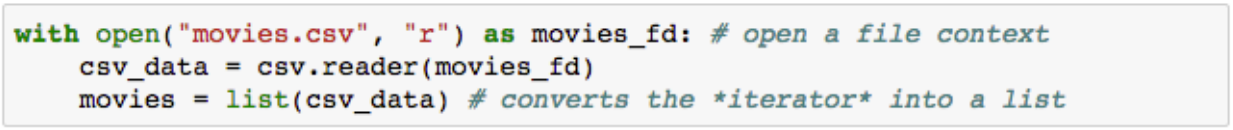
a. Open a new Jupyter Notebook from the File Manager (using the drop down menu New->Python). In the new Notebook, click on the "*Untitled*" name at the top of the Notebook and rename it something sensible (i.e. "Lab 2").

b. You will see an empty Notebook cell ready for you Python code. Add the code to import the Python csv module and run the cell. In a Notebook it is common practice to keep all Python imports in the first cell of the Notebook.



```
In [1]: import csv
```

c. Create a new Notebook cell below the cell you just ran (*Insert->Insert Cell Below*). Add code to this cell that open's a CSV file named "movies.csv" and parses the contents of the data file into a *list of lists* using the `csv.reader()` function as shown in lecture this week.

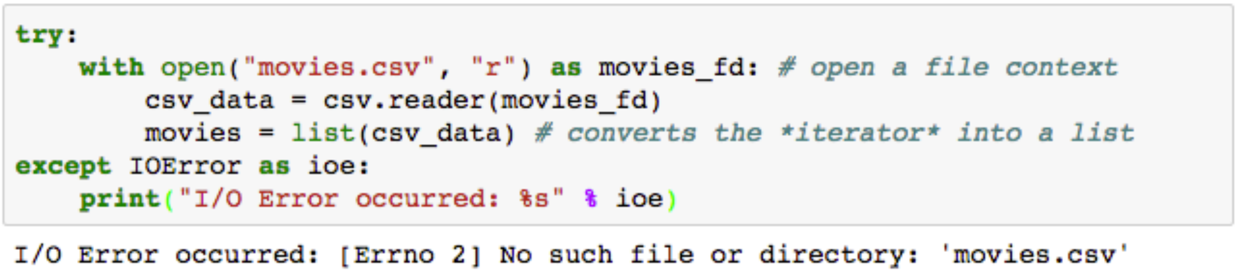


```
with open("movies.csv", "r") as movies_fd: # open a file context
    csv_data = csv.reader(movies_fd)
    movies = list(csv_data) # converts the *iterator* into a list
```

What happens when try to run this cell?

d. OH NO! Most likely an Exception happens (i.e. IOError)! *Why do you think this Exception happened?*

Add `try…` `except` statements to your code to catch the IOError Exception and to print out an error message. Modify the cell's code as shown. You should get an error message now.

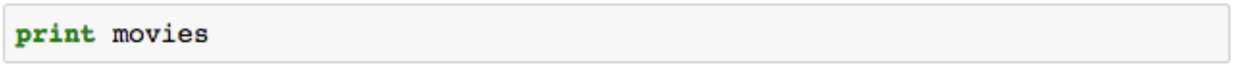


```
try:
    with open("movies.csv", "r") as movies_fd: # open a file context
        csv_data = csv.reader(movies_fd)
        movies = list(csv_data) # converts the *iterator* into a list
except IOError as ioe:
    print("I/O Error occurred: %s" % ioe)
```

```
I/O Error occurred: [Errno 2] No such file or directory: 'movies.csv'
```

e. Download the `data.zip` for this practical from KEATs and unzip it in the same directory as your Notebook. Modify the file path in your code to point to the `movies.csv` file. You should now be able to run the cell with out an error.

f. In a new Notebook cell, print the movies list, *what is the format that csv.reader() parses movies.csv into?*



```
print movies
```

What is the contents of the first element in the movies list? Is it movie data?

g. In a new Notebook cell, use a for-loop iterate through the *list of lists* of the movie information and display each record of the data file. Come up with a human readable record format to print out, for example:

```
Casablanca (1942)
Directed by: Michael Curtiz
Starring: Humphrey Bogart, Ingrid Bergman, Paul Henreid, Claude Rains
Plot: Set in Casablanca, Morocco during the early days of World War II: An
American expatriate meets a former lover, with unforeseen complications.
```

Note that you do not need to print out all of the information for each movie. Try to use the *string formatting* technique that we covered in lecture this week using the `%` operator. For example, to print out the first line of the above record (“`Casablanca (1942)`”), you would use the following `print` statement (where `row` is for-loop iterator variable that is the list representation of a row of data in the file):

```
print(“%s (%s)” % (row[0], row[1]))
```

h. You will notice that the first entry in the movies list is the header information. To skip the header row, you can use list *slicing* on the movies list variable to specify the second item in the list through the end of the list:

```
movies[1:]
```

i. In a new Notebook cell, compute and display the *number of records* in the data file. Refer to this week’s lecture to see how to print out the number of records.

Part 1 Challenge

Only attempt this challenge when you have completed the rest of the Parts of this practical.

The `csv.reader()` function will parse the data into a list of lists. See if you can use your Python skills to convert the *list of lists* into a *list of dicts* (dictionaries).

Hint: Take advantage of the fact that the first line in the data file (and the first entry in the list of list representation) contains the names of the fields in each record (i.e. the *header* row).

Part 2 Reading CSV Files with csv.DictReader

a. In a new Notebook cell, open the `movies.csv` file again like you did in **Part 1**, but instead of using `csv.reader()` use the `csv.DictReader()` this time to part the contents of the data file into a list of dictionary representation. *Hint:* use the example from this week's lecture which demonstrates reading a CSV file with `csv.DictReader()`.

   You will notice that the example requires adding the field names to the function call (i.e. `fieldnames`). If you open the `movies.csv` in a text editor, you will notice that the first line of the data file is the header (i.e. the field names).

b. In a new Notebook cell, use a for-loop to iterate through the data records in the *list of dicts* and display each data record for each movie like you did in Part 1.

c. You will notice again that the first row (i.e. *header* row) is read in as data. *Without modifying the `movies.csv` data file, how can you make sure that the header line (i.e. the first line) in the data file is not parsed in by `csv.DictReader`?*

   Hint: Look at the Documentation for csv.DictReader? https://docs.python.org/2/library/csv.html#csv.DictReader

   What does the documentation say will happen when the fieldnames parameter is removed from the function call?

d. Modify your code that uses `csv.DictReader()` so that you can read in the csv file without parsing in the *header* line.

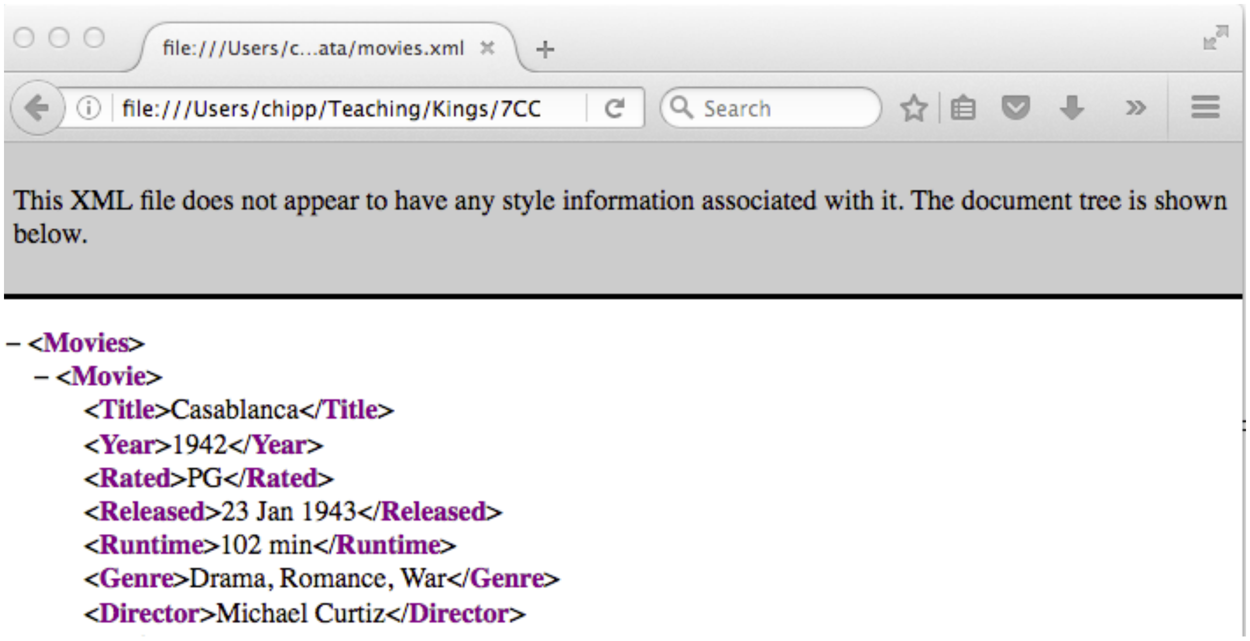
Part 3 Reading and parsing XML File

You will use the `xml.etree.ElementTree` class to parse the contents of the XML version of the movie data file (`movies.xml`).

a. At the top of the notebook in the first cell where you imported the `csv` module. Add the line to import the `xml.etree.ElementTree` module. Be sure to run the cell after you modify the cell.

```
import csv
import xml.etree.ElementTree as et
```

b. Open the `movie.xml` file in a text editor and get to know the tag structure of the data file. Alternatively, you can also drag the `movie.xml` file into a web-browser window (i.e. Firefox). This produces a *collapsable tree view* of the XML file (i.e. try clicking on the '-' next to the tag names).


file:///Users/c...ata/movies.xml
file:///Users/chipp/Teaching/Kings/7CC
Search

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
– <Movies>
  – <Movie>
      <Title>Casablanca</Title>
      <Year>1942</Year>
      <Rated>PG</Rated>
      <Released>23 Jan 1943</Released>
      <Runtime>102 min</Runtime>
      <Genre>Drama, Romance, War</Genre>
      <Director>Michael Curtiz</Director>
```

c. In a new Notebook cell, write code that opens the data file using the `xml.etree.ElementTree` class and gets the root Element of the ElementTree.

```
tree = et.ElementTree(file='data/movies.xml')
root = tree.getroot()
```

d. In a new Notebook cell, use a for-loop to iterate over every movie record as an XML `Element`. Use the ***non-recursive*** example from this week's lecture using the xml.etree.ElementTree class to help you. Display each movie record like you did in **Part 1**.

   Hint: each Element has a `find(<tag name>)` function which you can use to retrieve the first Element with the `<tag name>`, for example, if m is a `Movie` tag (i.e. Element) in our XML file, this codes finds the `Rated` child tag:

```
rated_tag = m.find("Rated")
print(rated_tag.text)
```

```
PG
```

   There is also a `findall(<tag name>)` function which returns a *list* of Elements with the `<tag name>`. For an example, see the Documentation: https://docs.python.org/2/library/xml.etree.elementtree.html#parsing-xml

e. Once you have displayed all of the XML records, In a new Notebook cell, *compute* and display the number of *records* in the data file.

Part 3 Challenge

Only attempt this challenge when you have completed the rest of the Parts of this practical.

In a new Notebook cell, copy and modify your XML parsing code from **Part 3** to read the movie data records into a *list of dicts* similar to the representation that you get from **Part 2**.

This means for every movie `Element`, there will be a dictionary representation of the movie records from **Part 2**.

Part 4 What Movie were they in?

Now you will write a function that will search the movie records for movies a specific actor starred in.

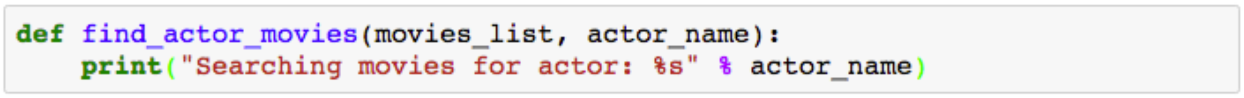
a. In a new Notebook cell, define a function `find_actor_movies(movies_list, actor_name)`, that takes two arguments as input:

   - `movies_list`: the *list of dict* representation of the movies data file (what you parsed in **Part 2** using the `csv.DictReader()` class).

   - `actor_name:` a string for an actors name

   and returns a list of movies (as dicts) that the specified actor was in.

   Here is the function started for you:



```
def find_actor_movies(movies_list, actor_name):
    print("Searching movies for actor: %s" % actor_name)
```

   Hint: You'll notice that the `"actors"` field in the movie dictionary is a comma separated string of actors names. Convert this string into a list of actors name using the `str.split()` function, and search for your actor in this list. See: https://docs.python.org/2/library/stdtypes.html#str.split

   Hint 2: Or as an alternative, *u*se the `in` operator to search for the `actor_name` as a substring in the "`actors`" field for a move

b. In a new Notebook cell, test your function with the following code:

```
bergman_movies = find_actor_movies(movies, "Ingrid Bergman")
print bergman_movies
```

```
Searching movies for actor: Ingrid Bergman
[{'Plot': 'Set in Casablanca, Morocco during the early days of World War II: An American expa
triate meets a former lover, with unforeseen complications.', 'Rated': 'PG', 'Response': 'TRU
E', 'Language': 'English, French, German, Italian', 'Title': 'Casablanca', 'Country': 'USA',
 'Metascore': 'N/A', 'imdbRating': '8.6', 'Director': 'Michael Curtiz', 'Released': '23-Jan-4
3', 'Writers': 'Julius J. Epstein (screenplay), Philip G. Epstein (screenplay), Howard Koch
 (screenplay), Murray Burnett (play), Joan Alison (play)', 'Actors': 'Humphrey Bogart, Ingrid
 Bergman, Paul Henreid, Claude Rains', 'Year': '1942', 'Genre': 'Drama, Romance, War', 'Award
s': 'Won 3 Oscars. Another 6 wins & 6 nominations.', 'Runtime': '102 min', 'Type': 'movie',
 'Poster': 'http://ia.media-imdb.com/images/M/MV5BMjQwNDYyNTk2N15BMl5BanBnXkFtZTgwMjQ0OTMyMjE
@._V1_SX300.jpg', 'imdbVotes': '357332', 'imdbID': 'tt0034583'}]
```

c. In a new Notebook, use the returned list of movies that includes the actor and print out the following:

- the title of any movie the actor that includes this actor
- the total number of movies that include this actor

Or, if the actor is not found in any movie, then display a message explaining that. Test your `find_actor_movies()` with an actor that is not listed in these movies.

Part 4 Challenge

Only attempt this challenge when you have completed the rest of the Parts of this practical.

What if I submitted part of an actor's name, instead of the full name? How can you test that the part of the actor's name is in the full name provided in the `movies.csv`?

Modify your `find_actors_movies()` function to find all movies whose actor's name is part of.

What if I submitted the actor's name in a difference case? i.e.

`find_actor_movies(movies, “bogart”)`

Modify your `find_actor_movies()` function to find all the movies that the actor `“bogart”` is a part of.

Part 5 Writing CSV Files with csv.DictWriter

a. In a new Notebook cell, export the movies that you found using your `find_actor_movies()` to a new CSV file using the `csv.DictWriter`.

   Output this new csv file into the `data/` directory (i.e. the same directory where you read in the movies.csv file).

   Name the new csv file: `<actor_name>_movies.csv`, where `<actor_name>` is a *lower* case version of the actor's name and all *blank spaces* in the name are replaced with underscores '_'. i.e.: "Ingrid Bergman"s movies is saved in:

   ```
   ingrid_bergman_movies.csv
   ```

   Hint: Use the `str.lower()` function (https://docs.python.org/2/library/stdtypes.html#str.lower) to convert the actor's name to lower case. Also, use the `str.replace()` function to replace the blank spaces with underscores.

   Use the example from this week's lecture to help you. Make sure that you also write the header for the csv file, and that you export all of the movie data fields.

Congratulations! You have now successfully read in CSV and XML files, wrote a small function to search through the movie data, and exported a CSV file with your results!

Part 6 Parse Movies Python Script

So far in the Jupyer Notebook that you have been working on, you have tested code that does the following things:

1. Reads in the `movies.csv` file into a list of dicts representation that you can iterate over (**Part 3**)

2. Wrote a function `find_actor_movies()` that searches the list of dicts of movies, and returns a new list of dicts of only those movies that the actor is in (**Part 4**)

3. Wrote out the movies the actors was in to a CSV file whose name included (**Part 5**)

You will now pull together the code that you tested in the Notebook into a Python script file (.py) that will do those above steps when you run it from the command line.

Download the `parse_movies.py` file from KEATs and use it as a template. ***After you modify the file with each step below***, run your `parse_movies.py` file on the command line and check that it runs without errors!

a. Copy the necessary Python module imports from the Notebook to top of the python file.

b. Copy the `find_actor_movies()` function from **Part 4** above the `main()` function in the file.

c. In the `main()` function in the template, insert the code to parse the `movies.csv` file into a *list of dicts* (from **Part 3**)

d. Print out the number of movie records in the file (from **Part 3**). Note that you do not have to include the for-loop to display out each record of the file.

e. Use the `find_actor_movies()` function to search the movies for an actor that stars in them (i.e. look in the `movies.csv` to find an actor's name of your choosing).

f. With the result from `find_actor_movies()` print out the number of movies this actor was in.

g. Output the movies the actor was in to a file name `<actor>_movies.csv,` where `<actor>` is the actor name *lower cased* and having the blank spaces replaced with *underscores* ('_') (from **Part 5**).

Run and test your script on the command line and make sure that it outputs the correct movies in the correctly named output CSV file.

Part 7 Find any actor!

In the Python script we wrote in **Part 6**, the actor's name was *hard-coded* as a search term when passed into the `find_actor_movies()` function. It would be nice to have the option to search for different actors without modifying the program.

We will use a new csv file, queries.csv, to specify a list of actors names, who we want to look up their movies for. Take a look at the file `queries.csv`. You'll see it is a single column CSV file (i.e. with a heading `query_string`). Each line lists an actor's name we wish to look up in our movie records with the `find_actor_movies()` function.

Modify your Python script from **Part 6**, to read in these actors names (use one of the csv methods from **Part 2** or **Part 3**) from the `queries.csv` file. For ***each*** actor name in the queries file, output a separate csv file (with the same format specified in **Part 5**) that contains the movies that they starred in.

Test your script out on the `movies.csv` file with the `queries.csv` file.

Finally, **submit** your final Python script file on KEATs under this week's Lab Practical heading. *You will not be assessed* for this lab practical, but this will give us a chance to test your Python script and to give you feedback on how it performs. Submitting the script will also give you practice submitting code for when you have to do Coursework 1.

Part 7 Challenge 1

Only attempt this challenge when you have completed the rest of the Parts of this practical.

Create a version of your python script that instead uses the XML version of the data file.

Part 7 Challenge 2

Only attempt this challenge when you have completed the rest of the Parts of this practical.

Assume that we would like to be able to search in any data field in our movie data file, not just the "`Actors`" data field. For example, let us assume that we might want to look for a keyword in the Plot text for a movie.

In order to accommodate this, we can add another column to the `queries.csv` file which would have the following format:

```
field_name, query_string
```

Where the `field_name` is the name of the field in the movie data record that we want to search, and the `query_string` is a keyword that we want to look for in the that field. So for example,

```
Actor,bogart
```

will search for any movie with the string "*bogart*" in the "`Actor`" field. And,

```
Plot,love
```

Searches for any movie with the string "*love*" in the "Plot" field.

Output your search results in separate CSV files, and decide on a file name format based on the `field_name` and `query_string` from the `queries.csv` file.