SI 506: Lecture 13

1. `os.path`
2. Challenges

Vocabulary

- **Docstring**. String literal that appears as the first statement in a function, class, or module. The docstring provides a terse description of an object's purpose, attributes, and behavior. The docstring is assigned to an object's **doc** attribute and is available via introspection.
- **File Object**. An object that provides a file-oriented application programming interface (API) to a either a text file, binary file (e.g., image file), or a buffered binary file. File objects include read and write methods for interacting with a file stored locally or remotely.
- **Flow of execution**. The order in which statements in a program are executed. Also referred to as *control flow*.
- **UTF-8**. UTF-8 is a variable-width character encoding that uses one to four one-byte (8-bit) code units to represent individual characters. The encoding encompases the older US-ASCII character set as well nearly all Latin-script alphabets as well as IPA extensions, Greek, Cyrillic, Coptic, Armenian, Hebrew, Arabic, Syriac, Thaana, N'Ko alphabets and most Chinese, Japanese, and Korean characters. UTF-8 is the dominant encoding used on the Web.

Data

- **resnick-citations.csv**. A comma-separated values (CSV) delimited text file containing biblometric data (e.g., citation report) of Professor Paul Resnick's articles, book chapters, and conference papers. Data sourced from the Web of Science database and exported into Endnote. Beyond illustrating this week's topic on reading from/writing to files, the data set also helps illustrate UMSI scholarly output, scholarly connections within UMSI (e.g., UMSI co-authors) as well as scholarly “influence" (citation counts).
- **umsi-faculty.csv**. List of UMSI faculty (last name, first name).

1.0 Using `os.path` to create paths

The standard library's `os.path` module includes a number of useful functions for constructing pathnames out of strings. When installing Python you get the `os.path` module suitable for the operating system Python is expected to run on (e.g., macOS, Windows 10) and is therefore usable for constructing OS-specific paths.

```
# Current working directory
cwd = os.getcwd()
print(f"\n1.0 Current working directory = {cwd}")

# Absolute path to directory in which *.py is located.
abs_path = os.path.dirname(os.path.abspath(__file__))
print(f"\n1.1 Absolute directory path = {abs_path}")
```

```
# Construct macOS and Windows friendly paths
faculty_path = os.path.join(abs_path, 'umsi-faculty.txt')
resnick_path = os.path.join(abs_path, 'resnick-citations.csv')
```

💡 an alternative library for working with path objects is the `pathlib` module, introduced as part of the Python 3.4 release.

2.0 Challenges

Today's challenges focus on implementing functions and reading from and writing to CSV files.

`resnick-citations.csv` "headers" row

```
citation_headers = [
    'Title', 'Authors', 'Book Editors', 'Source Title', 'Publication
Date', 'Publication Year',
    'Volume', 'Issue', 'Part Number', 'Supplement', 'Special Issue',
'Beginning Page', 'Ending Page', 'Article Number', 'DOI', 'Conference
Title', 'Conference Date', 'Total Citations',
    'Average per Year', '1995', '1996', '1997', '1998', '1999', '2000',
'2001', '2002', '2003',
    '2004', '2005', '2006', '2007', '2008', '2009', '2010', '2011',
'2012', '2013', '2014', '2015',
    '2016', '2017', '2018', '2019', '2020'
    ]
```

`umsi-faculty.csv` "headers" row

```
faculty_headers = ['last_name', 'first_name']
```

Challenge 01

Task. Implement a function that permits retrieval of any publication element using the CSV "headers" row as an index value lookup mechanism.

1. From `main` call the function `read_csv` and return a list of publications contained in the file `resnick-citations.csv`. Assign the return value to a variable named `publications`.

2. Access the "headers" element in `publications`. Assign the return value to a variable named `headers`.

3. Implement the function `get_attribute`. Review the docstring to better understand the function's expected behavior. You can then employ the function to return any single publication attribute (e.g., Title, Publication Year, Total Citations) by leveraging the CSV's "headers" row of column names to look up the element's index value.

   💡 The function can be implemented with one line of code.

4. After implementing `get_attribute` return to the `main` function. Call the function `get_attribute`, passing to it as arguments the 9th publication in the `publications` list, the `headers` list, and the `column_name` "Total Citations". Assign the return value to a variable named `total_citations`.

Challenge 02

Task: Return the publication(s) with the highest citation count. Write the results to a file.

1. In `main` create two "accumulator" variables: `max_citations` (`list`) and `max_count` (`int`). Assign sensible default values.

   You will use the variables to identify the the publication(s) with the highest number of citations recorded between 1995-2020. If two or more publications tie for the highest number of citations, append each to the `max_citations` list.

2. Loop over the publications list and implement conditional statements that compare the current publication's "Total Citations" count to the previously recorded `max_count`. In the loop block call the function `get_attribute` to retrieve the publication's "Total Citations" count and assign the return value to a variable of your own choosing.

   Conditions

   1. If the current total citations count is greater than the previous count, *remove* all publications added previously to `max_citations` and then append the current publication to the list (the new leader). Set `max_count` to the current total citations count.

   2. If the current count is equal to the previous `max_count` append the publication to `max_citations`.

   3. Otherwise, proceed to the next iteration of the loop.

3. After exiting the loop call the function `write_csv` and write the updated `max_citations` list to a file named `resnick-citations-max_citations.csv` passing the `headers` list as the `headers` argument.

Challenge 03

Task: Return the publication(s) with the lowest citation count. Write the results to a file.

1. In `main` create two "accumulator" variables: `min_citations` (`list`) and `min_count` (`int`). Assign sensible default values.

   You will use the variables to identify the the publication(s) with the lowest number of citations recorded between 1995-2020. If two or more publications tie for the lowest number of citations, append each to the `min_citations` list.

2. Loop over the publications list and implement conditional statements that compare the current publication's "Total Citations" count to the previously recorded `min_count`. In the loop block call the function `get_attribute` to retrieve the publication's "Total Citations" count and assign the return value to a variable of your own choosing.

Conditions

1. If the current total citations count is less than the previous count, *remove* all publications added previously to `min_citations` and then append the current publication to the list (the new leader). Set `min_count` to the current total citations count.

2. If the current count is equal to the previous `min_count` append the publication to `min_citations`.

3. Otherwise, proceed to the next iteration of the loop.

3. After exiting the loop call the function `write_csv` and write the updated `min_citations` list to a file named `resnick-citations-min_citations.csv` passing the `headers` list as the `headers` argument.

Challenge 04

Task: Return list of UMSI-coauthored publications. Write the results to a file.

1. In `main` call the `read_csv` function and pass to it the filepath `./umsi-faculty.csv`. Assign the return value, a nested list of UMSI faculty members, to a variable named `umsi_faculty`.

2. Implement the function `has_umsi_faculty_author`. Review the docstring to better understand the function's expected behavior.

   ❗ when looping over the `umsi_faculty` list access each faculty member's first and last name from the nested list and assign the elements to a string formatted as follows:

   `<last name>, <first name>`

   This is required in order to match the author name strings in the passed in `authors` list.

   Conditions

   1. If the caller overrides the default value of `ignore` by passing in a faculty member name (e.g., 'Resnick, Paul') do not attempt to match the faculty member's name with any of the names in the `authors` list.

      ❗ At a minimum the caller needs to exclude Paul Resnick from the author check. Paul is a coauthor of all the publications in the data set. Your job is retrieve all publications that featured one or more of Paul's USMI faculty colleagues as a coauthor. A more sophisticated implementation of the function would permit a caller to pass in a list of names to ignore.

   2. If the faculty member's name is not equal to the `ignore` value *and* the name is in the `authors` list return `True`.

   3. Otherwise, return `False`.

3. After implementing `has_umsi_faculty_author` return to the `main` function. Create an empty "accumulator" list named `umsi_coauthored_publications`. The list will hold UMSI faculty-coauthored publications.

4. Loop over the publications list accessing each publication's authors. In the loop block call the function `get_attribute` to retrieve the publication's "Authors" and assign the return value to a variable of your own choosing.

5. Write an `if` statement in the loop block that calls the function `has_umsi_faculty_author` and passes to it as arguments `umsi_faculty` (a slice), `authors`, and `Resnick, Paul`. If the expression evaluates to `True` append the publication to the "accumulator" list.

6. Call the function `write_csv` and write the updated `umsi_coauthored_publications` list to the file `resnick-citations-umsi_coauthored.csv` employing `headers` as the headers argument.

Challenge 05

Task: The data set includes columns that provide an annual count of the number of citations garnered by each publication for the period 1995-2020. However, the total citation count per year is not provided and must be calculated. Implement a function that computes the total citation count across all publications for a given year.

💡 You can leverage the `headers` element and slicing to return a list of all years that the data set covers. Then loop over the list and for each year sum the citation count for each publication.

1. Return a slice of `headers` that includes all year elements ('1995'-'2000'). Assign the list to a variable named `years`.

   💡 Look up the index value for the `headers` element `1995` and use it as the start value in your slicing notation.

2. Create an empty "accumulator" listed named `annual_counts`.

3. Implement the function `get_citation_count_by_year`. Review the docstring to better understand the function's expected behavior.

   ❗ Type conversion is required since all values derived from the CSV file are of type `str`.

4. After implementing `get_citation_count_by_year` return to the `main` function. Loop over the `years` list and in the loop block call the function `get_citation_count_by_year` passing to it a slice of the `publications` list, `headers`, and the current year value. Assign the return value to a local variable of your own choosing (e.g., `count`).

5. Append the annual citation count to `annual_counts` in the form of a two-item tuple comprising the year and total citations for the year.

   ```
   [
       (< year >, < total citations >),
       (< year >, < total citations >),
       . . .
   ]
   ```

6. After exiting the loop call the function `write_csv` and write the `annual_counts` list to the file `resnick-citations-annual_counts.csv`. Since each nested tuple comprises two items pass `['year', 'citations']` as the headers argument.

Challenge 06 (BONUS)

Task: Convert all caps titles, authors, and source title strings to mixed case according to a specified set of formatting rules. Consider this is an exercise in data "cleaning".

1. Implement the `format_string` function. The function includes three locally scoped tuples: `acronyms`, `stopwords`, and `ordinals`. Employ the tuples to filter on acronyms, words, and ordinals as you clean each word. Review the docstring to better understand the function's expected behavior.

   ❗ Split the passed in string into a list of words; implement a `for in range()` loop that employs conditional logic (`if-elif-else`) to "clean" each word in the list per the rules specified in the docstring. After exiting the loop, join the "cleaned" word elements into a string and return the new string to the caller.

   💡 Implement the function one step at a time, testing each modification you make by calling the function in `main`.

   1. Split the string into a list; (re)join the list elements and return the string (there are handy `str` methods available for this task). Call function and confirm that it behaves as expected.
   2. Write a single `if` statement that implements a rule. Call function and confirm that it behaves as expected.
   3. Write accompanying `else` statement that implements the default "cleaning" task. Call function and confirm that it behaves as expected.
   4. Write accompanying `elif` statement that implements a rule. Call function and confirm that it behaves as expected.
   5. Repeat step 4 as many times as is required to implement all cleaning tasks. Call function and confirm that it behaves as expected.

2. After implementing `format_string` return to the `main` function. Loop over the `publications` list (ignore the headers) and in the loop block call the function `format_string` and clean the publication title, authors, and source strings (call the function a total of three times during each loop iteration).

3. After exiting the loop call the function `write_csv` and write the updated `publications` list to the file `resnick-citations-cleaned.csv` employing the `headers` list as the headers argument.