SI 506 Lecture 14

1. Absolute paths
    1. `pathlib.path` module
    2. `os` module (ye olde way)
2. Error handling
    1. `try` and `except` statements in action
3. Challenges
    1. Challenge 01
    2. Challenge 02
    3. Challenge 03
    4. Challenge 04 (Bonus)
    5. Challenge 05 (Bonus)

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.

Lecture 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 Absolute paths

A path points to a specific location in a hierarchical file system. File paths are either *absolute* or *relative*.

An **absolute path** includes the root element (i.e., `/`, `C:`) and the directory list (delimited by either a forward `/` (macOS/*nix or a backwords slash `\` (Windows)) required to reach the target directory or file.

Absolute file path

```
# macOS/*nix
path = '/Users/arwhyte/Documents/umich/courses/SI506/lectures/lecture_14/lecture_14.py'

# Windows (Git Bash)
path = '/c/Users/arwhyte/Documents/umich/courses/SI506/lectures/lecture_14/lecture_14.py'

# Windows (Command Prompt)
path = 'C:\Users\arwhyte\Documents\umich\courses\SI506\lectures\lecture_14\lecture_14.py'
```

A **relative path** is defined in relation to the current working directory (cwd). Given the absolute paths above the relative path `lectures/lecture_14` or `./lectures/lecture_14` implies that the current working directory is either `/Users/arwhyte/Documents/umich/courses/SI506/lectures` (macOS/*nix) or `/c/Users/arwhyte/Documents/umich/courses/SI506/lectures/` (Windows Git Bash).

One drawback to using absolute paths is that they are generally **not portable** between operating systems and file systems. With rare exceptions, the absolute paths listed above would trigger a runtime `FileNotFoundError` exception if included in Python `*.py` file shared with you.

❗ Avoid submitting assignments to Gradescope that include *hard-coded* filepath strings. An absolute path included in your `*.py` file that is appropriate for your local file system is guaranteed to trigger a runtime `FileNotFoundError` exception when encountered by Gradescope. In other words, your machine's macOS or Windows file system absolute paths do not match the absolute paths required for Gradescope's Linux environment (or a classmate's laptop file system). In other words, absolute paths are rarely if ever portable between file systems.

1.1 `pathlib.path` module

As might be expected the Python standard library includes modules to deal with these challenges. The `pathlib.path` module provides an object-oriented approach to creating and managing paths. The `pathlib` module included in your Python 3.x download is designed to work with your operating system (OS) so no special configuration is required for it to recognize and work OS-specific file system paths.

Like the `csv` module you must import `pathlib.path` module to use it. Once imported you instantiate (i.e., create) an instance of the `Path` class and then access its methods and other attributes. In the example below a `Path` instance's `cwd()` method is called to return the current working directory path.

```
from pathlib import Path

cwd = Path.cwd() # method call
```

You can also call the `Path` instance's `resolve()` and `absolute()` methods to construct an absolute path to this `*.py` file's parent directory:

💡 the "dunder" `__file__` attribute is assigned to the module that is currently being imported (e.g., `lecture_14.py` file).

```
parent_path = Path(__file__).resolve().parent

# OR

parent_path = Path().absolute()

# OR

parent_path = Path().resolve()
```

As hinted above, you can leverage the "dunder" `__file__` attribute when instantiating `Path()` to retrieve the absolute path of a target file such as `lecture_14.py`:

```
abs_path = Path(__file__).absolute()

# OR

abs_path = Path(__file__).resolve()

# OR

abs_path = Path('lecture_14.py').resolve()
```

You can construct paths by calling the `joinpath()` method:

```
parent_path = Path(__file__).resolve().parent # parent directory

faculty_path = parent_path.joinpath('umsi-faculty.csv')
resnick_path = parent_path.joinpath('resnick-citations.csv')
```

And you can retrieve segments from a path:

```
print('\n1.2.5 Path parts',
    f"\nname = {resnick_path.name}",
    f"\nstem = {resnick_path.stem}",
    f"\nsuffix = {resnick_path.suffix}",
    f"\nparent dir = {resnick_path.parent}",
    f"\nparent.parent dir = {resnick_path.parent.parent}"
    )
```

There is much more to `pathlib.path` than returning absolute paths. For more information on using the module see this week's recommended readings.

1.2 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. Like `pathlib.path` you get the `os.path` module designed for the operating system Python 3.x is expected to run on (e.g., macOS, Windows 10).

```
# Current working directory
os_cwd = os.getcwd()

# Absolute path to directory in which *.py is located.
os_parent_path = os.path.dirname(os.path.abspath(__file__))

# Construct macOS and Windows friendly paths
os_faculty_path = os.path.join(os_parent_path, 'umsi-faculty.csv')
os_resnick_path = os.path.join(os_parent_path, 'resnick-citations.csv')
```

2.0 Error handling

The Resnick publication/citation data contains a variety of numbers masquerading as strings. However, given the number of list elements looping over the data and attempting to determine which elements to convert to type `int` appears problematic.

Python's `try` and `except` statements can help reduce the complexity of the challenge. The idea behind the statements is to *try* and perform an operation inside the `try` statement block and if the action triggers a runtime exception allow the accompanying `except` statement to *catch* the exception before it terminates execution and perform one or more operations in response.

The strategy is to handle errors *after* they are encountered rather than before, an approach known by the acronym **EAFP** (i.e., easier to ask forgiveness than permission).

The function `convert_to_int` illustrates how to leverage `try` and `except` statements to keep a program/script running despite encountering a runtime exception (in this case a `ValueError`) if a value is passed to the function that cannot be converted to an integer by the built-in `int()` function.

```
def convert_to_int(value):
    """Attempts to convert a string, number or boolean value to an int. If
    a runtime ValueError exception is encountered, the function returns
the
    value unchanged.

    Parameters:
        value (str|bool): string or boolean value to be converted

    Returns:
        int: if value successfully converted else returns value unchanged
    """

    try:
        return int(value)
    except ValueError:
        return value
```

When a `value` is passed to `convert_to_int` the Python interpreter will attempt to execute the `try` clause. Should the `try` block result in an exception, the interpreter will proceed directly to the `except` clause and execute its statement block, thus avoiding the termination of the program's execution due to an exception (such as a `ValueError`).

An `except` clause may specify a specific exception type or multiple exceptions expressed as a parenthesized tuple:

```
except ValueError:
   ...
```

```
except (AttributeError, TypeError, ValueError):
      ...
```

Also a `try` statement may be accompanied by more than one `except` clause in order to specify different handlers for different exceptions. An `else` clause can be added after the `except` statement(s) in order to include code that *must* be executed if the `try` statement block does not raise an exception.

❗ If an exception occurs that does not match the specified exception(s) named in the `except` clause an *unhandled* exception is triggered and code execution will cease as a result.

2.1 `try` and `except` statements in action

The `resnick-citations.csv` will be read in by the `read_csv` function as a nested list. In order to convert strings to integers in each nested publication list we will need to either call the function `convert_to_int` from inside a nested loop (a future topic) or pair the function with another function that allows us to traverse each publication's elements. The function `clean_data` is designed to handles that task.

```
def clean_data(publication):
    """Mutates the passed in < publication > list by converting numbers
masquerading as
    strings to an integer (int).

    Checks each string element in the < publication > list. Delegates to
the function
    < convert_to_int > the task of attempting to convert the target string
to an integer.
    Strings that cannot be converted are returned unchanged.

    Parameters:
      publication (list): represents a publication

    Returns:
        list: mutated publication list
    """

    for i in range(len(publication)):
        publication[i] = convert_to_int(publication[i])
    return publication
```

The program/script work flow is managed by the `main` function. Converting certain publication strings to integers will require the following steps:

1. Read in the Resnick publications/citation data.

2. Loop over the publications. Given that each "publication" element is itself a mutable list a `for` loop can be employed to engage with each element.

3. Call the function `clean_data` during each loop iteration passing it the current publication (a `list`) as the required argument.

    1. `clean_data` will loop over the passed in publication's elements. For each element encountered the function will call `convert_to_int` in an attempt to convert the string to an integer.

    2. `convert_to_int` will attempt to convert the value to an integer. If the passed in element triggers a runtime `ValueError` exception the function will catch the exception and return the element unchanged; otherwise it will return an integer after passing the element to the built-in `int()` function.

4. Assign the return value to back to the current element.

If all goes well each mutated publication element will resemble the following list:

```
['Recommender systems', 'Resnick, Paul; Varian, HR', '', 'COMMUNICATIONS
OF THE ACM', 'MAR 1997',
1997, 40, 3, '', '', '', 56, 58, '', '10.1145/245108.245121', '', '',
1552, '64.67', 0, 0, 3, 4,
16, 24, 19, 24, 34, 49, 52, 38, 73, 79, 91, 82, 80, 83, 87, 107, 126, 132,
112, 113, 91, 33]
```

3.0 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. 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.

2. After implementing `get_attribute` return to the `main` function. The "Average per Year" value of each publication remains a string that could be converted to a float. Loop over the `publications` list, call `get_attribute` to retrieve the "Average per Year" value for each publication. Assign to a local variable (name your choice).

3. Call the function `convert_to_float` passing to it the value to be converted. Assign the return value to the current publication's "Average per Year" element.

   💡 Utilize `headers` to look up the "Average per Year" index value so that you can assign the float value to the correct element.

4. Uncomment `print()` and check your work.

Challenge 02

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

1. Replace the `pass` statement in the function `has_umsi_faculty_author` with compound conditional statement that implements the following conditions:

   1. The local `name` value must not equal the passed in or default `ignore` value (e.g., Paul Resnick).
   2. The local `name` value is a member of the `coauthors` list.
   3. If a match is obtained return `True`.

   Review the docstring to better understand the function's expected behavior.

2. After fixing `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.

3. 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 (name your choice).

4. 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), and `authors`. If the expression evaluates to `True` append the publication to the "accumulator" list.

5. 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 03

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

3. 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 the `publications` list, `headers`, and the current year value. Assign the return value to a local variable of your own choosing (e.g., `count`).

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

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

5. 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 04 (BONUS)

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 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.

   Requirements

   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 05 (BONUS)

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.

   Requirements

   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.