SI 506 Midterm

1.0 Dates

- Release date: Thursday, 24 February 2022, 4:00 PM Eastern
- Due date: on or before Saturday, 26 February 2022, 11:59 AM (morning) Eastern

2.0 Overview

Review the companion *Midterm Overview* document in Canvas for general details regarding this assignment.

3.0 Points

The midterm is worth 1000 points and you accumulate points by passing a series of auto grader tests.

4.0 Solo effort

The midterm exam is open network, open readings, and open notes. You may refer to code in previous lecture exercises, lab exercises, and problem sets for inspiration.

❗ You are prohibited from soliciting assistance or accepting assistance from any person while completing the programming assignment. The midterm code that you submit *must* be your own work. Likewise, you are prohibited from assisting any other student required to complete this assignment. This includes those attempting the assignment during the regular exam period, as well as those who may attempt the assignment at another time and/or place due to scheduling conflicts or other issues.

❗ If you have formed or participate in an SI 506 study group please suspend all study group activities for the duration of the midterm assignment.

5.0 Data

The midterm data sets are drawn from the the Center for Disease Control and Prevention (CDC) and the National Center for Health Statistics (NCHS).

Before commencing the challenges review the data contained in the following files:

File	Source	Description
`mi-covid-washtenaw-20220220.csv`	CDC	COVID-19 county level community transmission levels for Washtenaw County (home of Ann Arbor)
`mi-covid-vax_counties-20220219.csv`	CDC	COVID-19 county level vaccination rates for Michigan counties located within two counties of either Macomb or Wayne counties, inclusive.
`mi-nchs-urban_rural_codes.csv`	NCHS	Urban/rural classification scheme for U.S. counties and county-equivalent entities (2013)

Examine each file. Note how the data is structured. Determine which "columns" permit the data to be linked. Inspect the string values and consider which values can/ought to be cast to a different type (e.g. string -> integer).

Two other CSV files are also included with the midterm files.

- fxt-ur_vax_levels.csv
- fxt-vax_levels.csv

You will replicate these files (assigning each a different name).

6.0 Debugging

As you write your code take advantage of the built-in `print` function, VS code's debugger, and VS Codes file comparison feature to check your work.

6.1 The built-in `print` function is your friend

💡 as you work through the challenges make frequent use of the built-in `print()` function to check your variable assignments. Recall that you can call `print` from inside a function, loop, or conditional statement. Use f-strings and the newline escape character `\n` to better identify the output that `print` sends to the terminal as expressed in the following example.

```
print(f"\nSOME_VAL = {some_val}")
```

6.2 VS Code debugger

You can also use the debugger to check your code. If you have yet to configure your debugger see the instructions at https://si506.org/resources/. You can then set breakpoints and review your code in action by "stepping over" lines and "stepping into" function calls.

6.3 Compare CSV files

Each test fixture CSV file (prefixed with `fxt-`) represents the correct file output that *must* be generated by the program you write. Use them at periodic intervals to compare your current output against the expected output.

💡 In VS Code you can compare or "diff" the file you generate against the appropriate test fixture file. After calling the `write_csv` function and generating a new file do the following:

1. Hover over the file with your cursor then right click and choose the "Select for Compare" option.

2. Next, hover over the appropriate test fixture file then right click and choose the "Compare with Selected" option.

3. Lines highlighted in red indicate character and/or indention mismatches. If any mismatches are encountered close the comparison pane, revise your code, regenerate your file, and compare it again to the test fixture file. Repeat as necessary until the files match.

❗ Your output **must** match each fixture file line for line and character for character. Review these files; they are akin to answer keys and should be utilized for comparison purposes as you work your way through the assignment.

7.0 Challenges

The midterm comprises ten (10) challenges that are based on the CDC / NCHS data. Certain challenges can be solved with a single line of code. Others may involve writing several lines of code, including implementing functions. Some functions are expected to delegate tasks to other functions implemented in previous challenges.

❗ The teaching team recommends strongly that you complete each challenge in the order specified in this README document.

Challenge 01

1. In the `main` function block, call the function `read_csv` to retrieve the data from the file `mi-covid-washtenaw-20220220.csv`. This data from the Center for Disease Control (CDC) contains information on COVID-19 community spread for Washtenaw County during the first three weeks of February 2022. Assign the return value to a variable named `wash_data`.

2. Use slicing (no loops of any kind) to return the `wash_data` list elements with a reported test positivity percentage rate (e.g. column "percent_test_results_reported_positive_last_7_days") greater than 10.0 percent and assign the value of the expression to a variable named `max_positivity_rates`.

3. Use **negative** slicing (no loops of any kind) to return all elements that contain data for the week 13-19 February 2021 and assign the value of the expression to a variable named `week_04`.

4. Use slicing (no loops of any kind) to return elements associated with *odd numbered days only*, excluding the "header" row (1st element) and assign the value of the expression to a variable named `odd_days`.

Challenge 02

Task: Retrieve the CDC data and implement a function that can return a nested list representing a county from the CDC data set.

1. In the `main` function block, call the `read_csv` function and retrieve the Center for Disease Control (CDC) county vaccination rates contained in the file `mi-covid-vax_counties-20220219.csv` for select Michigan counties. Assign the return value to a variable named `vax_data`.

   ❗ Recall that individual data elements accessed from a CSV file will be imported as strings. Some strings may need to be converted to a different data type in the challenges below.

2. The "headers" row constitutes the first element in `vax_data`. Access the element and assign it to a variable named `vax_headers`.

3. Next access the remaining "row" elements and assign them to a variable named `vax_counties`.

> 💡 the `vax_counties` list contains nested list elements, each of which represents a Michigan county and its county-specific vaccination rates data.

4. Implement the function named `get_county`. The function defines two parameters: `counties` (`list`) and `county_name` (`str`). Review the function's docstring for more information about the function's expected behavior.

   Requirements

   The function *must* perform a *case insensitive* string comparison of the county names.

5. After implementing `get_county` return to the `main` function. Call the function passing it the `vax_counties` list and the string "jackson county" (lowercase required). Assign the return value to a variable named `jackson_cty`.

Challenge 03

Task. Implement a function that permits the retrieval of any county element using the CSV "headers" row to look up individual index values.

1. Implement the function `get_county_attribute`. The function defines three parameters: `county` (`list`), `headers` (`list`), and `column_name` (`str`). Review the docstring to better understand the function's expected behavior.

   Requirements

   The function *must* use the passed in `headers` list and `column_name` (e.g., the element string value) to look up the header element's index value and then use it to return the associated county element.

   Once implemented employ the function to return any single county attribute (e.g., "Recip_County", "Series_Complete_Pop_Pct", "Series_Complete_65PlusPop_Pct") by leveraging the CSV's "headers" row of column names to "look up" its associated index value.

   > 💡 This function can be implemented with one line of code.

2. After implementing `get_county_attribute` return to the `main` function. Call the function `get_county_attribute` and retrieve the "Series_Complete_Pop_Pct" value for Jackson County by passing the appropriate arguments to the function (i.e., `jackson_cty`, `vax_headers`, etc.). Convert the return value to a `float` and then assign it to a variable named `jackson_cty_vax_complete_pct`.

   > 💡 The "Series_Complete_Pop_Pct" value represents a county's vaccination completion status (e.g., percent value of fully vaccinated residents).

Challenge 04

Task: Convert CDC numbers represented as strings to either type `int` or `float` in order to simplify working with numeric values.

1. Implement the function named `clean_county_data`. The function defines a single parameter: `county` (`list`). Review the docstring to better understand the function's expected behavior.

The following table lists CDC data that can be converted from a string to either an `int` or `float`.

Column name	type	convert to
FIPS	`str`	`int`
MMWR_week	`str`	`int`
Series_Complete_Pop_Pct	`str`	`float`
Series_Complete_Yes	`str`	`int`
Series_Complete_12Plus	`str`	`int`
Series_Complete_12PlusPop_Pct	`str`	`float`
Series_Complete_18Plus	`str`	`int`
Series_Complete_18PlusPop_Pct	`str`	`float`
Series_Complete_65Plus	`str`	`int`
Series_Complete_65PlusPop_Pct	`str`	`float`

Requirements

In the function block choose a `for` loop that facilitates assigning a new value to a list element.

Employ conditional statements in tandem with the functions `is_floating_point_number` and `is_whole_number` (both provided) to determine whether or not the passed in string represents either a floating point number or a whole number.

- If a string represents a floating point number (i.e., a number with a fractional component) convert the string to a `float` and assign it to the corresponding element.
- If a string represents a whole number convert the string to an `int` and assign it to the corresponding element.
- Otherwise, do not mutate the element.

Recall that an `if` statement can include a call to a function that returns a boolean value.

2. After implementing `clean_county_data` return to the `main` function. Loop over the `vax_counties` list and inside the loop block call `clean_county_data` passing to it the appropriate argument in order to "clean" each county's "numeric" data values.

Call the `print()` function and pass it `vax_counties`. Confirm that strings representing whole numbers have been converted to `int` and strings representing floating point numbers have been converted to `float`.

Challenge 05

Task: Implement a function that retuns a count of all residents considered fully vaccinated between the ages of 18 and 64 (inclusive) across all counties.

1. Implement the function named `count_vax_adults_18to64`. The function defines two parameters: `counties` (`list`) and `headers` (`list`). Review the docstring to better understand the function's expected behavior.

   Requirements

   In the function block implement the accumulator pattern. Inside the loop block call the function `get_county_attribute` (passing to it the appropriate arguments) as needed in order to retrieve each county's "Series_Complete_18Plus" and "Series_Complete_65Plus" values. Then compute the number of county residents between the ages of 18 and 64 who are fully vaccinated and add the value to a local accumulator variable. After exiting the loop return the total count to the caller.

2. After implementing `count_vax_adults_18to64` return to the `main` function. Call the function, passing to it the requirement arguments and assign the return value to a variable named `vax_adults_18to64`.

Challenge 06

Task: Implement a function that categorizes a county's vaccination completion status as High, Medium, or Low for a given population (e.g., all, aged 12+, 18+, 65+).

1. Implement the function named `classify_county_vax_level`. The function defines three parameters: `county` (`list`), `headers` (`list`), and `column_name` (`str`). Review the function's docstring for more information about the function's expected behavior.

   Requirements

   The function *must* delegate to the function `get_county_attribute` the task of returning the percentage value of the county's target population who are fully vaccinated.

   💡 Recall that a function can specify more than one `return` statement.

2. After implementing `classify_county_vax_level` return to the `main` function. Loop over `vax_counties` and for each county encountered call the function `classify_county_vax_level` passing to it the county, headers, and string "Series_Complete_Pop_Pct". Retrieve the county's "vax level" (e.g., 'High', 'Moderate', or 'Low') and insert it into the county list as the sixth (6th) element.

3. After inserting the new "vax level" values into the nested county lists in `vax_counties` you *must* also insert a corresponding "Vax_Level" string value into the `vax_headers` list as the sixth (6th) element. Doing so ensures that both `vax_headers` and the nested county list indexes remain synchronized, thus ensuring that the "headers" describe correctly the county list elements.

4. Call the function `write_csv` and pass to it as arguments the filepath `stu-vax_levels.csv`, `vax_counties`, and `vax_headers`. Compare your output file to the test fixture file `fxt-vax_levels.csv`. Both files *must* match, line for line, and character for character.

Challenge 07

Task: Retrieve the NCHS data and implement a function to retrieve a county's CBS title and Urban/Rural (UR) code and descriptor.

1. In the `main` function, call the `read_csv` function and retrieve the National Center for Health Statistics (NCHS) urban/rural county codes contained in the file `mi-nchs-urban_rural_codes.csv` for select Michigan counties. Assign the return value to a variable named `nchs_codes_data`.

2. The "headers" row constitutes the first element in `nchs_codes_data`. Access the element and assign it to a variable named `nchs_codes_headers`.

3. Next access the remaining "row" elements and assign them to a variable named `nchs_codes`.

4. Implement the function named `get_county_nchs_codes`. The function defines two parameters: `nchs_codes` (`list`) and `county` (`list`). Review the function's docstring for more information about the function's expected behavior.

   Requirements

   The function *must* perform a *case insensitive* string comparison of the county names.

   The function *must* return a three-item `tuple` comprising a county's CBSA title, NCHS urban/rural code, and the urban/rural code descriptor.

   ❗ Convert the NCHS urban/rural code to an `int` before returning the value to the caller.

5. After implementing `get_county_nchs_codes` return to the `main` function. Call the function `get_county` passing it the `vax_counties` list and the string "Ingham County" (mixed case required). Assign the return value to a variable named `ingham_cty`.

6. Next, call the function `get_county_nchs_codes` passing to it the `nchs_codes` list and `ingham_cty`. Assign the return value to a variable named `ingham_cty_ur_code`.

Challenge 08

Task: Implement a function that returns a list of formatted strings that represent counties with the lowest vaccination rate for a given target population.

1. Implement the function named `get_counties_with_lowest_vax_rate`. The function defines three parameters: `counties` (`list`), `headers` (`list`), and `column_name` (`str`). Review the function's docstring for more information about the function's expected behavior.

   Requirements

   The function *must* return a list comprising one or more formatted strings that represent counties with the lowest vaccination rate for a given population group (e.g., all, aged 12+, 18+, 65+).

   💡 Implement the accumulator pattern when working with the `counties` list.

   The function *must* be designed to accommodate the possibility that more than one county may share the same low vaccination rate. Make sure you adjust the function's conditional logic to handle ties between counties.

   The function *must* delegate to the function `get_county_attribute` the task of returning the county vaccination rate for a given age group. The age group is determined by passing to `get_county_attribute` the `vax_headers` and a "*_Pop_Pct" `column_name` string as arguments.

Acceptable < column_name > values that can be passed to `get_counties_with_lowest_vax_rate` by the caller include:

- "Series_Complete_Pop_Pct"
- "Series_Complete_12PlusPop_Pct"
- "Series_Complete_18PlusPop_Pct"
- "Series_Complete_65PlusPop_Pct"

Once the target rate is acquired, append a string to the list to be returned to the caller formatted as follows:

```
< Recip County > (< rate >%)
```

Example: "Hillsdale County (70.9%)"

2. After implementing `get_counties_with_lowest_vax_rate` return to the `main` function. Call the function and pass it the following arguments:

   - `vax_counties`
   - `vax_headers`
   - "Series_Complete_12PlusPop_Pct"

   Assign the return value to a variable named `lowest_vax_rates_12_plus`.

3. Call the function again and pass it the counties and headers lists and "Series_Complete_18PlusPop_Pct" as the third (3rd) argument. Assign the return value to a variable named `lowest_vax_rates_18_plus`.

4. Call the function one more time and pass it the counties and headers lists and "Series_Complete_65PlusPop_Pct" as the third (3rd) argument. Assign the return value to a variable named `lowest_vax_rates_65_plus`.

Challenge 09

Task: Combine County vaccination data with NCHS urban/rural county codes.

1. Loop over the `vax_counties` list. For each county encountered call the function `get_county_nchs_codes` passing to it as arguments the `nchs_codes` and the county. Unpack the return value (a `tuple`) and assign the items to three variables: `cbsa_title`, `ur_code`, and `ur_code_name`.

2. Insert the unpacked values into the county list *between* the < Recip_State > and < Vax_Level > values in the order specified below:

```
[..., < Recip_State >, < cbsa_title >, < ur_code >, < ur_code_name >,
< Vax_Level >,...]
```

❗ Review the test fixture file `fxt-ur_vax_levels.csv` for the proper placement of the values.

💡 There is more than one way to mutate the county lists. The approach involving the fewest lines of code utilizes slice assignment; another approach involves use of a purpose-built `list` method, and/or list concatenation in combination with slicing.

3. After mutating the county lists, you *must* also insert the corresponding *mixed case* header values into the `vax_headers` list in the following order:

    - CBSA_Title
    - UR_Code
    - UR_Code_Name

    Doing so ensures that both `vax_headers` and the nested county list indexes remain synchronized, thus ensuring that the "headers" describe correctly the county list elements. Review the test fixture file `fxt-ur_vax_levels.csv` for proper placement of the three header values.

4. Call the function `write_csv` and pass to it as arguments the filepath `stu-ur_vax_levels.csv` and the mutated counties and headers lists. Compare the file to the test fixture file. Both files *must* match, line for line, and character for character.

Challenge 10

Task: Implement a function that returns a list of formatted "county" strings filtered on county UR codes and vaccination levels.

1. Implement the function named `get_counties_by_ur_codes_and_vax_level`. The function defines four parameters: `counties` (`list`), `headers` (`list`), `ur_codes` (`tuple`), and `vax_level` (`str`). Review the function's docstring for more information about the function's expected behavior.

    Requirements

    The function *must* return a list comprising zero or more formatted strings.

    :bulb; Loop over the `counties` list and in the loop block write an `if` statement that uses the passed in tuple of `ur_codes` (e.g., `(3, 4, 5)`) and `vax_level` code (e.g., 'High', 'Moderate', or 'Low') to determine whether or not any of `ur_codes` *and* the `vax_level` match a county's "UR_Code" and "Vax_Level" values. If `True` then build the string to represent the county and append it to a local accumulator list.

    UR codes are listed below.

UR_Code	UR_Code_Name
1	Large central metro
2	Large fringe metro
3	Medium metro
4	Small metro

UR_Code	UR_Code_Name
5	Micropolitan
6	Noncore

When building the string the function *must* utilize the `headers` list to look up the index value associated with each of the following `headers` elements:

- "Recip_County"
- "Vax_Level"
- "UR_Code"
- "UR_Code_Name"

Employ the appropriate `list` method to return the index value and use it to access the corresponding element in each nested county list.

Each string appended to the accumulator list to be returned to the caller *must* be formatted as follows:

```
< Recip_County > (< UR_Code >-< UR_Code_Name >) vax level: < Vax_Level
>
```

Example: "Lapeer County (2-Large fringe metro) vax level: Low"

❗ The passed in `headers` list *must* include "UR_Code" and "Vax_Level" elements *and* each nested county list in `counties` must include the corresponding UR code and vax level values or a runtime exception will be triggered. This should not be an issue if you have completed the previous challenges successfully.

2. After implementing `get_counties_by_ur_codes_and_vax_level` return to the `main` function. You will utilize the function to explore the vaccination levels in the six counties that comprise the core-based statistical area (CBSA) of Detroit-Warren-Dearborn as well as "Noncore" rural counties. Call the function and pass it the following arguments:

   - `vax_counties`
   - `vax_headers`
   - a two-item tuple comprising the "Large central metro" and "Large fringe metro" UR_Code *numeric* values.
   - a vax level value of "Moderate"

   Assign the return value to a variable named `metro_moderate_vax_rates`.

3. Call the function again, passing only the *required* arguments, including the following value for the third argument:

   - a two-item tuple comprising the "Large central metro" and "Large fringe metro" UR_Code numeric values.

   Assign the return value to a variable named `metro_low_vax_rates`.

4. Call the function one last time, passing only the *required* arguments, including the following value for the third argument:

   - a one-item tuple comprising the "Noncore" UR_Code numeric value.

   Assign the return value to a variable named `noncore_low_vax_rates`.

8.0 Gradescope submissions

You may submit your solution to Gradescope as many times as needed before the expiration of the exam time.

❗ Your **final** submission will constitute your exam submission.

9.0 Auto grader / manual scoring

The autograder runs a number of tests against the Python file you submit, which the autograder imports as a module so that it can gain access to and inspect the functions and other objects defined in your code. The functional tests are of two types:

1. The first type will call a function passing in known argument values and then perform an equality comparison between the return value and the expected return value. If the function's return value does not equal the expected return value the test will fail.

2. The second type of test involves checking variable assignments in `main()` or expressions in other functions. This type of test evaluates the code you write, character for character, against an expected line of code using a regular expression to account for permitted variations in the statements that you write. The test searches `main()` for the expected line of code. If the code is not located the test will fail.

If the auto grader is unable to grade your submission successfully with a score of 1000 points the teaching team will grade your submission **manually**. Partial credit **may** be awarded for submissions that fail one or more autograder tests if the teaching team (at their sole discretion) deem a score adjustment warranted.

If you submit a partial solution, feel free to include comments (if you have time) that explain what you were attempting to accomplish in the area(s) of the program that are not working properly. We will review your comments when determining partial credit.