# SI 506 Midterm

## 1.0 Dates

- Release date: Thursday, 21 October 2021, 4:00 PM Eastern
- Due date: on or before Saturday, 23 October 2021, 11:59 PM 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.

❗ However, 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.

## 5.0 Data

The three (3) midterm data sets are drawn from the US National Women's Soccer League (NWSL) site's 2021 league standings as well as various Wikipedia pages describing NWSL stadium venues.

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

- `nwsl-club_info-2021.csv`
- `nwsl-standings-2021.csv`
- `nwsl-top_scorers-2021.csv`

Examine each file. Note how the data is structured. Determine which "fields" 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).

A fourth CSV file named `nwsl-combined_data-2021.csv` is also included with the midterm files. You will replicate this file and its contents (assigning it a different name) once you complete Challenge 10.

## 6.0 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}")
```

## 5.0 Challenges

The midterm comprises ten (10) challenges that are based on the NWSL club 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.

Taken together, the functions that you implement and call from the `main()` function will help you compile additional club statistics. The final challenge will involve combining the club info, standings, and scoring data —enriched with new data— and writing it to a CSV file.

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

💡 Challenges 02-10 each include a *recommended* testing section that involves calling the function that you've just implemented and passing to it the arguments it requires in order to evaluate its return value. This work may also involve writing data to a CSV file so that you can inspect its contents and make code corrections as necessary. The "test" code that you may choose to write is *not* evaluated by the auto grader. That said, I recommend that you consider implementing the recommended "test" code in order to verify your progress as you make your way through the challenges.

### Challenge 01

**Task**: Read the data files. Demonstrate ability to access subsets of the data employing indexing and slicing.

**Required**

1. In `main()` call the function `read_csv` and retrieve the NWSL top scorers data stored in the file `nwsl-top_scorers-2021.csv`. Assign the return value to a variable named `scorers_data`.

2. Employ *slicing* to retrieve the three players from `scorers_data` who are tied for sixth place (6th) with seven (7) goals each. Assign the return value to a variable named `top_scorers_seven_goals`.

3. In `main()` call the function `read_csv` and retrieve the NSWL club standings data stored in the file `nwsl-standings-2021.csv`. Assign the return value to a variable named `standings_data`.

4. Employ *slicing* to *reverse* the order of the `standings_data` list. Assign the return value to a variable named `standings_reversed`.

5. In `main()` call the function `read_csv` and retrieve the NWSL stadiums data stored in the file `nwsl-stadiums-2021.csv`. Assign the return value to a variable named `stadiums_data`.

6. Employ *indexing* to retrieve Kansas City's stadium name. Assign the return value to a variable named `kc_stadium_name`.

## Challenge 02

**Task**: Extract "headers" and club records from `standings_data`. Implement the function `clean_club` to convert number values read in as strings from the CSV file to integers. Also implement the function `get_club_record` in order to simplify retrieving a club record. Test functions by retrieving the Orlando Pride's 2021 record.

**Required**

1. In `main()` access the "header" element in `standings_data`. Assign the list to a variable named `standings_headers`. Access the club record elements in `standings_data`. Assign the nested lists to a variable named `standings`.

2. Implement the function named `clean_club`. Read the function's Docstring to understand the task it is to perform, the parameters it defines, and the return value it computes, if any.

   ❗ In order to earn full credit for this challenge `clean_club` must utilize **a `for` loop employing the `range` object** with the correct stop value expression passed to it as an argument.

3. After implementing `clean_club` return to `main()`. Loop over the `standings` list. Mutate each club record element encountered by calling `clean_club` with the correct argument and assigning the return value to the (current) club element.

   💡 Call the `print()` function and pass it the `standings` list to confirm that the club records in `standings` have all been modified.

4. Next, implement the function named `get_club_record`. Read the function's Docstring to understand the task it is to perform, the parameters it defines, and the return value it computes, if any.

**Recommended (test code)**

1. In `main()` test `get_club_record` by calling the function and passing to it as arguments the list `standings` and the string 'Orlando Pride'. Assign the return value to a variable named `pride_record`.

2. If the list returned matches the list below (e.g., the Pride's statistics are now `int` values), proceed to the next challenge. Otherwise, review your implementations of `clean_club` and `get_club_record` and the loop you wrote in `main()` to "clean" each club record. Revise as necessary and retest the functions.

```
['Orlando Pride', 23, 7, 7, 9, 27, 31, 197, 97]
```

## Challenge 03

**Task**: Implement the function `calculate_points`. Test function. Calculate each club's points earned and append it to the club record in the `standings` list along with a new "header" value named 'points'. Write updated `standings` data to a CSV file with the appropriate headers row inserted.

**Required**

1. Implement the function named `calculate_points`. Read the function's Docstring to understand the task it is to perform, the parameters it defines, and the return value it computes, if any.

   💡 The function can be implemented with a single line of code.

2. After implementing `calculate_points` return to `main()`. Append the header value 'points' to the `standings_headers` list.

3. Next, loop over `standings` in `main()`. As you iterate over each club record calculate the points the club has earned so far during the 2021 season (i.e., call `calculate_points` and pass to it the club record as the argument). Assign the return value to the nested club record in order to mutate it.

**Recommended (test code)**

1. In `main()` call the function `get_club_record` and retrieve the Chicago Red Star's club record. Assign the return value to `chicago_record`.

2. Test `calculate_points` by calling the function and passing to it the argument `chicago_record`. Assign the return value to `chicago_points`.

   💡 After 23 matches the 2021 Chicago Red Stars have earned `35` points.

   If you computed this value correctly move on to the next step, otherwise, review your implementation of `calculate_points`. Revise as necessary and retest the function.

3. Call the function `write_csv` and write the mutated `standings` list to a file named `stu-nwsl_standings_pts-2021.csv`. Also pass as an argument to `write_csv` the list `standing_headers` in order to ensure that the CSV file's first row is a self-documenting "headers" row.

   ❗ Take a moment to review the `write_csv` Docstring in order to better understand the function's parameter list and behavior.

4. Below are the rows and values that `stu-nwsl_standings_pts-2021.csv` *must* contain:

   ```
   club,played,win,draw,loss,goals_for,goals_against,shots,shots_on_goal,points
   Portland Thorns FC,23,13,4,6,33,17,324,140,43
   OL Reign,23,12,3,8,34,24,290,132,39
   Washington Spirit,23,10,6,7,28,26,231,109,36
   Chicago Red Stars,23,10,5,8,27,28,231,109,35
   NJ/NY Gotham FC,21,8,8,5,26,18,197,73,32
   Houston Dash,23,9,5,9,31,30,213,94,32
   North Carolina Courage,23,9,5,9,28,23,246,118,32
   Orlando Pride,23,7,7,9,27,31,197,97,28
   Racing Louisville FC,22,5,5,12,19,38,143,61,20
   Kansas City,22,3,6,13,14,32,224,95,15
   ```

   If the content of the CSV file matches the rows above (e.g., header includes 'points'; each club row includes the points value), proceed to the next challenge. Otherwise, review your implementation of `calculate_points` and the loop you've written to mutate each club record in standings. Revise as necessary and retest the function.

## Challenge 04

**Task**: Classify clubs by points earned. Implement the function `classify_club` which assigns a club to one of three tiers: 'top_tier', 'middle_tier', and 'bottom_tier'. Test the function by looping over `standings`, calling `classify_club` for each club record and appending each tuple returned to an accumulator list. Write the list of nested tuples to a CSV file with the appropriate headers row inserted.

### Required

1. Implement the function named `classify_club`. Read the function's Docstring to understand the task it is to perform, the parameters it defines, and the return value it computes, if any.

   💡 This function delegates to `calculate_points` the task of obtaining a club's point value.

### Recommended (test code)

1. In `main()` create an empty accumulator list named `clubs_classified`.
2. Loop over `standings`. For each club record encountered call `classify_club` passing to it the club record as the argument. Assign the return value to a variable of your own choosing (I like `tier`).
3. While still inside the loop, create a two-item tuple consisting of the club name and the tier value, e.g., `(< club name >, < tier >)`. Append the tuple to the accumulator list.
4. After exiting the loop, call the function `write_csv` and write `clubs_classified` to a CSV file named `stu-nwsl_clubs_classified-2021.csv`. Also pass as an argument to `write_csv` a list comprising the string elements 'club_name' and 'tier' in order to ensure that the CSV file's first row is a self-documenting "headers" row.

   💡 Below are the rows and values that `stu-nwsl_clubs_classified-2021.csv` *must* contain:

   ```
   club_name,tier
   Portland Thorns FC,top_tier
   OL Reign,top_tier
   Washington Spirit,top_tier
   Chicago Red Stars,middle_tier
   NJ/NY Gotham FC,middle_tier
   Houston Dash,middle_tier
   North Carolina Courage,middle_tier
   Orlando Pride,bottom_tier
   Racing Louisville FC,bottom_tier
   Kansas City,bottom_tier
   ```

   If the content of the CSV file matches the rows above, proceed to the next challenge. Otherwise, review your implementation of `classify_club` and the loop you've written to accumulate values. Revise as necessary and retest the function.

## Challenge 05

**Task**: Get top scorers by club affiliation. Implement the function `get_club_top_scorers`. Test the function.

**Required**

1. Implement the function named `get_club_top_scorers`. Read the function's Docstring to understand the task it is to perform, the parameters it defines, and the return value it computes, if any.
2. After implementing `get_club_top_scorers` return to `main()`. Access the "header" element in `scorers_data`. Assign the list to a variable named `standings_headers`. Access the top scorer elements in `scorers_data`. Assign the nested lists to a variable named `scorers`.

**Recommended (test code)**

1. In `main()` call `get_club_top_scorers` passing to it the `scorers` list and the string argument 'OL Reign'. Assign the return value to `ol_reign_scorers`.

   💡 If the `ol_reign_scorers` returned matches the list below, proceed to the next challenge.

   ```
   [
      ['Bethany Balcer', 'USA', 'OL Reign', '9', '2'],
      ['Megan Rapinoe', 'USA', 'OL Reign', '6', '9'],
      ['Jess Fishlock', 'GBR', 'OL Reign', '5', '11'],
      ['Eugenie Le Sommer', 'FRA', 'OL Reign', '5', '11']
   ]
   ```

   Otherwise, review your implementation of `get_club_top_scorers`. Revise as necessary and retest the function.

## Challenge 06

**Task**: League-recognized top scorers are unevenly distributed across the clubs. Tabulate their numbers per club. Implement the function `get_clubs_top_scorers_counts`. Call the function and retrieve a list of tuples containing each club's League-recognized top scorers. Write the list to a CSV file with the appropriate headers row inserted.

**Required**

1. Implement the function named `get_clubs_top_scorers_counts`. Read the function's Docstring to understand the task it is to perform, the parameters it defines, and the return value it computes, if any.

   💡 This function delegates to `get_club_top_scorers` the task of retrieving each club's league-recognized top scorers.

   💡 there is a handy built-in function that you should call to obtain the count.

**Recommended (test code)**

1. In `main()` call `get_clubs_top_scorers_counts`, passing to it as arguments the `standings` and `scorers` lists. Assign the return value to a variable named `clubs_top_scorers_counts`.

   💡 The list `clubs_top_scorers_counts` is comprised of tuples. For example, the OL Reign tuple in the list *must* match the tuple below:

   ```
   ('OL Reign', 4)
   ```

   If OL Reign tuple does not match the above, review your implementation of `get_clubs_top_scorers_counts`. Revise as necessary and retest the function.

2. Call the function `write_csv` and write `clubs_top_scorers_counts` to a file named `stu-nwsl_clubs_top_scorers_counts.csv`. Also pass as an argument to `write_csv` a list comprising the string elements 'club' and 'top_scorers_count' in order to ensure that the CSV file's first row is a self-documenting "headers" row.

   💡 Below are the rows and values that `stu-nwsl_clubs_top_scorers_counts.csv` *must* contain:

   ```
   club,top_scorers_count
   Portland Thorns FC,3
   OL Reign,4
   Washington Spirit,2
   Chicago Red Stars,0
   NJ/NY Gotham FC,2
   Houston Dash,1
   North Carolina Courage,2
   Orlando Pride,2
   Racing Louisville FC,1
   Kansas City,0
   ```

   If the content of the CSV file matches the rows above, proceed to the next challenge. Otherwise, review your implementation of `get_clubs_top_scorers_counts`. Revise as necessary and retest the function.

## Challenge 07

**Task**: Converting shots into goals is a key efficiency metric. It provides clubs with an indicator of the effectiveness of shots taken by its players. The shot conversion rate can be calculated by dividing the club's "goals for" (GF) by either the total number of shots taken (S) or the total number of shots considered on target, i.e., shots on goal (SOG).

For this challenge, implement the function `calculate_shot_conversion_rate`. Test the function by calculating the Portland Thorns FC shots on goal conversion rate.

**Required**

1. Implement the function named `calculate_shot_conversion_rate`. Read the function's Docstring to understand the task it is to perform, the parameters it defines, and the return value it computes, if any.

💡 Utilize conditional logic when implementing the function. The return value *must* be rounded to the *third* (3rd) decimal position.

**Recommended (test code)**

1. In `main()` call the function `get_club_record` and retrieve the Portland Thorns FC club record. Assign the return value to `thorns_record`.

2. Test `calculate_shot_conversion_rate` by calling the function and passing to it the argument `thorns_record` and the optional argument 'shots_on_goal'. Assign the return value to `thorns_shot_conversion_rate`.

   💡 the Portland Thorns shot conversion rate from *shots on goal* is `0.236`.

   If you computed the Thorns' shots on goal conversion rate value correctly move on to the next challenge, otherwise, recheck your implementation of `calculate_shot_conversion_rate`.

## Challenge 08

**Task**: calculate shot conversion rates for all the clubs by implementing the function `get_clubs_shot_conversion_rates`. Call the function and return a list of nested tuples of shots on goal conversion rates. Write the list to a CSV file with the appropriate headers row inserted.

**Required**

1. Implement the function named `get_clubs_shot_conversion_rates`. Read the function's Docstring to understand the task it is to perform, the parameters it defines, and the return value it computes, if any.

   💡 This function delegates to `calculate_shot_conversion_rate` the task of calculating the shots on goal conversion rate for each club.

**Recommended (test code)**

1. In `main()` call `get_clubs_shot_conversion_rates` passing to it the arguments `standings` and the string 'shots_on_goal'. Assign the return value to a variable named `shots_on_goal_conversion_rates`.

   💡 The list of nested tuples returned by `get_clubs_shot_conversion_rates` must include two-item tuples like the NJ/NY Gotham FC tuple below:

   ```
   [
       ...
       ('NJ/NY Gotham FC', 0.356)
       ...
   ]
   ```

   If the NJ/NY Gotham FC tuple matches the above NJ/NY Gotham FC tuple move on to the next step. Otherwise, recheck your implementations of `calculate_shot_conversion_rate` *and* `get_clubs_shot_conversion_rates`. Revise one or both of the functions as necessary and retest each.

2. Call `write_csv` and write the `shots_on_goal_conversion_rates` list to a CSV file named `stu-nwsl_shots_on_goal_conversion_rates-2021.csv`. Also pass as an argument to `write_csv` a list comprising the string elements 'club' and 'shots_on_goal_conversion_rate' in order to ensure that the CSV file's first row is a self-documenting "headers" row.

   💡 Below are the rows and values that `stu-nwsl_shots_on_goal_conversion_rates-2021.csv` *must* contain:

   ```
   club,shots_on_goal_conversion_rate
   Portland Thorns FC,0.236
   OL Reign,0.258
   Washington Spirit,0.257
   Chicago Red Stars,0.248
   NJ/NY Gotham FC,0.356
   Houston Dash,0.33
   North Carolina Courage,0.237
   Orlando Pride,0.278
   Racing Louisville FC,0.311
   Kansas City,0.147
   ```

   If the content of the CSV file matches the rows above, proceed to the next challenge. Otherwise, review your implementation of `get_clubs_shot_conversion_rates`. Revise as necessary and retest the function.

## Challenge 09

**Task**: What proportion of a club's goals are scored by its league-recognized top scorers? Calculate the percentage of club `goals_for` (GF) scored by top scorers and write the results to a file employing a specified format.

**Required**

1. Implement the function named `calculate_goals_by_top_scorers_pct`. Read the function's Docstring to understand the task it is to perform, the parameters it defines, and the return value it computes, if any.

   💡 This function delegates to `get_club_top_scorers` the task of retrieving each club's league-recognized top scorers.

   💡 be sure to cast any number masquerading as a string to an `int`.

**Recommended (test code)**

1. In `main()`. Create an empty accumulator list named `club_goals_top_scorers_pct`.

2. Loop over `standings`. For each club record encountered call the function `calculate_goals_by_top_scorers_pct` passing to it as arguments the club record and the `scorers` list.

3. Unpack the tuple return value into three variables named `goals_for`, `top_scorers_goals`, and `top_scorers_goals_pct`. After unpacking the return values and while still inside the loop create a list comprising the following elements:

   ```
   [ < club name >, goals_for, top_scorers_goals, top_scorers_goals_pct]
   ```

   💡 Create either a list literal or a list you assign to a variable of your own choosing and append it to the accumulator list.

   Append the list to `club_goals_top_scorers_pct` and repeat for each iteration of the loop.

4. After exiting the loop, call `write_csv` and write the `club_goals_top_scorers_pct` list to a file named `stu-nwsl_club_goals_top_scorers_pct.csv`. Also pass as an argument to `write_csv` a list comprising the string elements 'club', 'goals_for', 'top_scorer_goals', and 'top_scorer_goals_pct' in order to ensure that the CSV file's first row is a self-documenting "headers" row.

   💡 Below are the rows and values that `stu-nwsl_club_goals_top_scorers_pct.csv` *must* contain:

   ```
   club,goals_for,top_scorer_goals,top_scorer_goals_pct
   Portland Thorns FC,33,17,51.52
   OL Reign,34,25,73.53
   Washington Spirit,28,15,53.57
   Chicago Red Stars,27,0,0.0
   NJ/NY Gotham FC,26,15,57.69
   Houston Dash,31,9,29.03
   North Carolina Courage,28,12,42.86
   Orlando Pride,27,13,48.15
   Racing Louisville FC,19,6,31.58
   Kansas City,14,0,0.0
   ```

   If the content of the CSV file matches the rows above, proceed to the next challenge. Otherwise, review your implementation of `calculate_goals_by_top_scorers_pct` and the loop you've written to accumulate values. Revise as necessary and retest the function.

### Challenge 10

**Task**: Leverage your previous work and generate a new CSV file that contains data based on all three data sets. Implement the function `combine_data`. Mutate the `club_info` list with additional data and write the list to a CSV file with the appropriate headers row inserted.

**Required**

1. Implement the function named `combine_data`. Read the function's Docstring to understand the task it is to perform, the parameters it defines, and the return value it computes, if any.

   💡 Consider implementing the function employing a nested loop when working with the `club_info` and `standings` data. For each club record encountered either extend or append the club record list with new values returned from calls made to functions implemented during earlier challenges.

   💡 `combine_data` delegates to other functions the task of computing and returning the additional data values required to enrich `club_info`. The new header values described below align with the function calls that will need to be made from within `combine_data` to build out the new data set.

2. After implementing `combine_data` return to `main()`. Access the "header" element in the `club_info_data` list. Assign the list to a variable named `club_info_headers`. Access the club info elements in `club_info_data`. Assign the nested lists to a variable named `club_info`.

3. Combine `club_info_headers` and a *slice* of `standings_headers` (you *must* exclude the duplicate 'club' header element) into a list named `combined_headers`. Then append the following header values to the new list:

   - 'top_scorers_count',
   - 'top_scorers_goals',
   - 'top_scorers_goals_pct',
   - 'shot_conversion_rate',
   - 'shots_on_goal_conversion_rate',
   - 'club_classification'

   💡 Create either a list literal or a list you assign to a variable of your own choosing and append it to the new list.

4. Call the function `combine_data` passing to it `club_info`, `standings`, and `scorers` employing **keyword arguments passed in reverse order**. Assign the return value to a variable named `combined_data`.

   ❗ To reiterate you must call keyword arguments passed in reverse order when you call the function `combine_data`.

5. Call `write_csv` and write the `combined_data` list to a file named `stu-nwsl_combined_data.csv`. Also pass as an argument to `write_csv` the list `combined_headers` in order to ensure that the CSV file's first row is a self-documenting "headers" row.

   ❗ The file you generate *must* replicate the data stored in the file named `nwsl-combined_data-2021.csv`.

## 6.0 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.

## 7.0 Auto grader / manual scoring

The autograder runs 19 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.