SI 506 Midterm

1.0 Dates

- Available: Thursday, 19 October 2023, 4:00 PM Eastern
- Due: on or before Saturday, 21 October 2023, 11:59 PM Eastern

2.0 Overview

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

The teaching team recommends that you bookmark the following w3schools Python pages and/or have them open in a set of browser tabs as you work on the last assignment:

- Python keywords
- Python operators
- Python built-in functions
- Python `list` methods
- Python `str` methods
- Python `tuple` methods"

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

Please abide by the following rules:

The coding solution that you submit **must constitute your own work**. You are **prohibited from soliciting assistance or accepting assistance** from any person.

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

- If you work with a tutor please **suspend contact** with the tutor for the duration of the assignment.

Likewise, you are **prohibited from assisting any other student** who is required to complete this assignment. This includes students 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 Generative AI

You are permitted to leverage generative artificial intelligence (AI) tools while working to complete this programming assignment. Approach such tools and their output with caution. AI tools are imperfect and can produce content that is inappropriate, offensive, or otherwise problematic. AI tools can produce code that is inelegant, inefficient, insecure, unreliable, unreadable, and/or unmaintainable. AI tools can produce code that performs the desired computation but nevertheless fails to meet the requirements of an assignment problem or challenge.

You are responsible for all the code you submit. If you are unfamiliar with the syntax, structure, and/or semantics of AI-generated code do not include the code in your assignment solution.

:exlamation: If you cannot explain to your instructor the computations that AI-generated code performs **do not** include the code in your assignment solution. You do yourself no favors by submitting code that you do not understand.

6.0 Questions/Help

Direct all **questions** regarding the assignment to the Slack SI 506 workspace `#midterm` channel. Do not post code snippets to the `#midterm` channel. Avoid sending private direct messages (DMs) to teaching team members about the midterm challenges. Doing so is inefficient since it limits the response to a single team member and increases the likelihood of a delayed response. Surfacing your question on the `#midterm` channel also helps reduce message duplication.

That said, if a personal issue arises during the assignment period please send a private DM to a GSI or Anthony.

Please adopt the following rules when asking assignment-related questions in the Slack `# midterm` channel:

6.1 Message prefix

Please prefix each message with a bracket `[...]` that indicates the challenge number and any relevant sub-number:

[< Challenge number>. < sub-number >] ...
[1.0] ...
[3.2] ...
[7.1.5] ...

6.2 One issue per message

To ensure that the prefixing approach is more focused please limit each message to a single issue. If you have more than one issue, create one message for each.

6.3 Add short description

Finally, please attempt to describe the issue in a manner that is more descriptive than the otherwise opaque request for feedback/take a look at my problem. Doing so helps teaching team locate your issue faster and helps classmates determine if the issue described is relevant and the resulting thread worth reviewing.

If you experience a runtime exception please include the error message in your description (e.g., “NameError: name 'nums' is not defined.”, “SyntaxError: invalid syntax ....”)

7.0 README, template, and data files

In line with the weekly problem sets you will be provided with a number of files:

1. A `midterm-README.md` that contains the assignment instructions
2. A `midterm.py` template file for you to write your code.
3. One or more `*.txt` and/or `*.csv` files that contain assignment data.

❗ Please download the files from Canvas Files as soon as they are released. This is a timed event and delays in acquiring the assignment files will shorten the time available to engage with the assignment. The clock is not your friend.

The template file will contain function definitions that you will implement along with a `main()` function that you will implement to orchestrate the program's flow of execution.

Implementing other functions involves replacing the placeholder `pass` statement with working code. You may be asked to add missing parameters to function definitions. A function may need to delegate one or more tasks to other functions. You may also be asked to define a function in its entirety (less the docstring) as directed per the instructions.

8.0 Template `midterm.py` scaffolding

❗ *DO NOT* modify or remove the scaffolded code that we provide in the template unless instructed to do so.

The template `*.py` file resembles the following skeletal scaffolding (verbose multi-line docstrings are shortened to a single line in the example code below):

```python
import some_module


def some_function(some_parameter, another_parameter):
    """A description of expected behavior.

       Parameters:
            ...

        Returns:
            ...
    """

    pass # TODO Implement


def another_function(): # TODO Add missing parameter(s)
    """A description of expected behavior.

       Parameters:
            ...

        Returns:
            ...
    """

    pass # TODO Implement
```

```python

# TODO define and implement Challenge X function here


def main():
    """Entry point for the program. Orchestrates execution flow.

       Parameters:
            ...

        Returns:
            ...
    """

    # CHALLENGE 01

    data = None # TODO Read data file
    var_01 = None # TODO Assign value

    # . . .

    # CHALLENGE 0X

    # TODO Implement loop(s), conditional statement, add filtered value(s)
to specified variable

    # TODO Write to file

    # . . .

    # CHALLENGE 10

    var_02 = None # TODO Call function; assign return value

    # TODO Write to file

# Conditional statement below checks if the Python interepreter knows this
file as "__main__"
# (i.e., a script/program intended to be run from the terminal). If True,
calls the main() function
# which serves as the entry point to the program.

if __name__ == "__main__":
    main()
```

9.0 Challenges

The midterm comprises a maximum of ten (`10`) challenges. The teaching team recommends that you complete each challenge in the order specified in the README.

Certain challenges can be solved with a single line of code. Others may involve writing several lines of code including implementing one or more functions in order to solve the challenge.

💡 As you work through a challenge *decompose* the problem into discrete subproblems and solve each subproblem in turn rather than attempting to tackle the whole in one go.

The challenges draw upon topics introduced between weeks 01 and 07:

1. Basic syntax and semantics
    - Values (objects), variables, and variable assignment
    - Expressions and statements
    - Data types
        - numeric values (`int`, `float`)
        - boolean values (`True`, `False`)
        - NoneType object (`None`)
        - sequences: `list`, `range`, `str`, `tuple`,
    - Operators: arithmetic, assignment, comparison, logical, membership
    - Basic arithmetic operations (addition, subtraction, multiplication, floating point division, integer/floor division, modulus)
    - String formatting employing a formatted string literal (f-string)
    - `del` statement
    - `pass` statement
2. Sequences
    - Sequence indexing and slicing
    - Sequence method calls (e.g., `list` and `str` methods discussed in class, labs, and used in assignments)
    - List creation, mutation, and element unpacking
    - Tuple creation and unpacking items
    - String, list, and tuple concatenation
    - Nested list and nested tuple
3. Control flow
    - Iteration
        - `for` loop, `for i in range()` loop, `while` loop
        - Accumulator pattern
        - Counter usage (e.g. `count += 1`)
        - `break` and `continue` statements
    - Conditional execution
        - `if`, `if-else`, `if-elif-else` statements
        - Truth value testing (`if < expression >:`)
        - Compound conditional statements constructed with the logical operators `and`, `or`, and `not`
    - Assertions
        - `assert` statement

4. Functions
    - Built-in functions discussed in class and/or featured in problem sets or challenges
    - User-defined functions
        - Defining a function with/without parameters, parameters with default values, and with/without a return statement
        - Calling a function and passing to it arguments by position and/or keyword arguments
        - Calling a function or functions from within another function's code block
        - Assigning a function's return value to a variable
    - `main()` function (included in midterm template scaffolding)

5. Files read / write
    - File read/write using the `with` statement and built-in `open()` function
    - Read from and write to *.txt file
        - Implement `read_file` function
        - Implement `write_file` function
    - Read from and write to a *.csv file
        - `csv` module `import` statement (included in scaffolded code)
        - Implement `read_csv` function (calls `csv.reader()`)
        - Implment `write_csv` function (calls `csv.writer()`)

6. Error handling
    - `try` and `except` statements

10.0 A note on code styling

The auto grader includes tests that check whether or not your code adheres to Python's PEP 8 styling guidelines relative to the use of spaces in certain expressions and statements. The goal is to encourage you to write code that adheres to the Python community's styling practices. Doing so enhances code readability and aligns you with other Python programmers. We think it best to learn how to style your code now now rather than unlearn practices later that diverge from community styling.

10.1 Spaces in expressions and statements

Always surround the following operators on either side with a single space:

- assignment (`=`)
- augmented assignment (`+=`, `-=`, etc.)
- comparisons (`==`, `<`, `>`, `!=`, `<>`, `<=`, `>=`, `in`, `not in`, `is`, `is not`)
- Booleans (`and`, `or`, `not`).

```
# Correct
var = search_entities(entities, search_term)

# Incorrect
var=search_entities(entities,search_term)

# Correct
```

```
count += 1

# Incorrect
count+=1
```

Note however that an exception exists with respect to function parameters and arguments. Do *not* surround the assignment operator with spaces when either:

1. defining a parameter with a default value
2. passing a keyword argument

```
# Correct
def create_email_address(uniqname, domain='umich.edu'):
    """TODO"""
    return f"{uniqname}@{domain}"

# Incorrect
def create_email_address(uniqname, domain = 'umich.edu'):
    """TODO"""
    return f"{uniqname}@{domain}"

# Correct
email_address = create_email_address(uniqname='anthwhyte',
domain='gmail.com')

# Incorrect
email_address = create_email_address(uniqname = 'anthwhyte', domain =
'gmail.com')
```

Finally, when employing the subscript operator in an expression do not place a space between the sequence and the accompanying bracket(s).

```
# Correct
element = some_list[-1]

# Incorrect
element = some_list [-1] # eliminate space
```

10.2 Black Formatter and line length

The teaching team assumes that you have installed VS Code's Black Formatter extension and are using it to format your code. If you have yet to install the extension please do so now.

❗ After installing the Black Formatter extension you must change the maximum line length from 88 characters (default) to 100 characters. See the SI 506 VS Code install guide relevant for your operating system (macOS, Windows) for instructions on how to override Black's default settings.

11.0 Testing and debugging

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

11.1 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.

11.2 `assert` statements

You may encounter commented out `assert` statements in the template file. Uncomment the statements as necessary to check your code. If an `assert` statement triggers a runtime `AssertionError` review your code to determine why the assertion failed.

❗ If you submit a partial solution to Gradescope be sure to comment out `assert` statements that trigger runtime exceptions. Failure to do so will result in the same exception being triggered in the Gradescope environment. A runtime exception occurring anywhere in your code will terminate execution of your program/script resulting in the auto grader returning a score of zero (`0`) for the current submission.

11.3 VS Code debugger

You can also use the debugger to check your code. If you have yet to configure your debugger instructions are available. You can then set breakpoints and review your code in action by "stepping over" lines and "stepping into" function calls.

12.0 Gradescope submissions

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

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 or expressions in `main()` and 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()` or another function for the expected line of code. If the code is not located the test will fail.

If your final submission results in a score that is lower than a previous submission score you will be permitted to activate the earlier submission and claim the higher score.

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) deems 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.

❗ You *must* submit your solution to *Gradescope* before the expiration of exam time. Solution files sent to the teaching team after the expiration of exam time will receive a score of zero (`0`).