

SI 506: Last Assignment

1.0 Dates

- Available: Thursday, 08 December 2022, 4:00 PM Eastern
- Due: on or before Monday, 19 December 2022, 11:59 PM Eastern

! No late submissions will be accepted for scoring.

2.0 Overview

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

We recommend that at a minimum 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 dict methods](#)
- [Python list methods](#)
- [Python str methods](#)
- [Python tuple methods"](#)
- [SWAPI documentation](#)

3.0 Points

The last assignment is worth **1800** points and you accumulate points by passing a series of autograder tests.

4.0 Solo effort

Please abide by the following rules:

1. The last assignment that you submit **must constitute your own work**. You are **prohibited from soliciting assistance or accepting assistance** from any person while working to complete the programming assignment. This includes but is not limited to individual classmates, study group members, and tutors.
 - ! 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.
- 2. 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.
- 3. Direct all **questions** regarding the assignment to the Slack SI 506 workspace # **last_assignment** channel. Do not post code snippets. If you encounter an issue with your code you may request a code review by members of Team 506 via a private direct message (DM).

5.0 Data

The Star Wars saga has spawned films, animated series, books, music, artwork, toys, games, fandom websites, cosplayers, scientific names for new organisms (e.g., *Trigonopterus yoda*), and even a Darth Vader *grotesque* attached to the [northwest tower](#) of the Washington National Cathedral. Leading US news organizations such as the [New York Times](#) cover the Star Wars phenomenon on a regular basis.

The last assignment adds yet another Star Wars-inspired artifact to the list. The data used in this assignment is sourced from the [New York Times](#), [Wookieepedia](#), [Wikipedia](#), and the [Star Wars API](#) (SWAPI).

Besides retrieving data from SWAPI you will also access information locally from CSV and JSON files.

6.0 Files

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

1. **README.md**: assignment instructions
2. **last_assignment.py**: script including a **main()** function and other definitions and statements
3. **five_oh_six.py**: module containing utility functions and constants
4. One or more ***.csv** and/or ***.json** files that contain assignment data
5. One or more **fxt_*.json** test fixture files that you must match with the files you produce

Please download the assignment 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 challenges. The clock is not your friend.

! **DO NOT** modify or remove the scaffolded code that we provide in the Python script or module files unless instructed to do so.

6.1 last_assignment.py

The **last_assignment.py** "script" file contains function definitions that you will implement along with a **main()** function that serves as the entry point for the script (also known as a program).

From `main()` you will implement the script's workflow that will involve the following operations:

- Import modules
- Access data from local files (*.csv, *.json)
- Retrieve items from a local cache and/or issue HTTP GET requests to the remote SWAPI service
- Call functions and perform computations
- Assign values to specified variables
- Write data to local files encoded as JSON

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.

6.2 `five_oh_six.py`

The `five_oh_six.py` module contains both constants (e.g., `SWAPI_ENDPOINT = 'https://swapi.py4e.com/api'`) and functions, a number of which you will be asked to implement. You will import the `five_oh_six` module into `last_assignment.py` so that you can access the module's statements and definitions in your program.

6.3 CSV and JSON files

We will supply you with CSV and JSON files that include data for use in your program. Locate the files in the **same directory** where you place your templated script and module files.

6.4 Test fixture JSON files

You will also be supplied with a set of test fixture JSON files (prefixed with `fxt_`). Each 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.

! Your output **must** match the fixture files line for line, indent for indent, and character for character. Review these files; they constitute the answer keys and should be utilized for comparison purposes as you work your way through the assignment.

7.0 Module imports

The template file `last_assignment.py` includes a single `import` statement:

```
import copy
import five_oh_six as utl
```

The utilities module `five_oh_six.py` includes the following `import` statements:

```
import csv
import json
import requests

from urllib.parse import quote, urlencode, urljoin
```

! **Do not** comment out or remove these `import` statements. That said, check your `import` statements periodically. If you discover that other `import` statements have been added to your Python files remove them. In such cases, VS Code is attempting to assist you by inserting additional `import` statements based on your keystrokes. Their presence can trigger `ModuleNotFoundError` runtime exceptions when you submit your code to Gradescope.

8.0 Caching

As discussed in class, this assignment utilizes a caching workflow that eliminates redundant HTTP GET requests made to SWAPI by storing the SWAPI responses locally. Caching is implemented *fully* and all you need do is call the function `get_swapi_resource()` whenever you need to retrieve a SWAPI representation of a person, droid, planet, species, starship, or vehicle either locally from the cache or remotely from SWAPI.

! *Do not* call the function named `utl.get_resource` directly. Doing so sidesteps the cache and undercuts the caching optimization strategy.

The cache dictionary is serialized as JSON and written to `CACHE.json` every time you run `last_assignment.py`.

9.0 Challenges

The last assignment comprises a maximum of twenty (20) 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.

The challenges draw upon topics introduced between weeks 01 and 14 and outlined in the [syllabus](#) and described in more detail below:

1. Basic syntax and semantics
 - Values (objects), variables, variable assignment
 - Expressions and statements
 - Data types

- numeric values (`int`, `float`)
- boolean values (`True`, `False`)
- NoneType object (`None`)
- sequences: `list`, `range`, `str`, `tuple`
- associative array: `dict`
- Operators: arithmetic, assignment, comparison, logical, membership
- Escape sequences (e.g., newline `\n`)
- 2. String formatting: formatted string literal (f-string)
- 3. Data structures
 - Sequences
 - Indexing and slicing (subscript operator)
 - List creation, mutation (add, modify, remove elements), and element unpacking
 - `list` methods
 - Tuple packing (creation) and unpacking
 - `tuple` methods
 - `str` methods
 - String, list, and tuple concatenation
 - Associative array
 - Dictionary creation (add, modify, remove key-value pairs), subscript notation
 - `dict` methods
 - Working with nested lists, tuples, and dictionaries
 - List and dictionary comprehensions
 - `del()` statement
- 4. Control flow
 - Iteration
 - `for` loop, `for i in range()` loop, `while` loop
 - Accumulator pattern
 - Counter usage (e.g. `count += 1`)
 - `break` and `continue` statements
 - Nested loops
 - Conditional execution
 - `if`, `if-else`, `if-elif-else` statements
 - Truth value testing (`if object:`)
 - Compound conditional statements using the logical operators `and` and `or`
 - Negation with `not` operator
- 5. Functions
 - Built-in functions featured in lecture notes/code, lab exercises and problem sets
 - Literacy/competency
 - `print()`
 - `len()`
 - `type()`, `isinstance()`
 - `int()`, `float()`
 - `min()`, `max()`, `sum()`
 - `round()`
 - `open()`
 - `sorted()`

- Awareness
 - `dict()`, `dir()`, `id()`, `enumerate()`, `list()`, `slice()`, `str()`, `tuple()`
 - Object methods (e.g., `str.split()`, `list.append()`, `dict.items()`)
 - 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
 - Passing a function to a higher-order function as an argument
 - Assigning a function's return value to a variable
 - Reading and interpreting function docstrings
 - `main()` function (entry point for program or script)
 - Variable scope (global and local variables)
6. Files read / write
- `with` statement
 - Reading from and writing to *.txt files
 - `file_obj.read()`, `file_obj.readline()`, `file_obj.readlines()`
 - Reading from and writing to *.csv files
 - `csv` module import; `csv.reader()`, `csv.writer()`, `DictReader`, `DictWriter`
 - Reading from and writing to *.json files
 - `json` module import; `json.load()`, `json.dump()`
 - Creating absolute and relative filepaths with `os.path` or `pathlib.Path()`
7. Modules
- Importing modules
 - Creating and using custom modules
8. Web API
- `requests` module installation (via pip) and use
 - HTTP GET request/response
 - JSON payloads
 - caching
9. Exceptions, exception handling, and debugging
- `try` and `except` statements

10.0 A note on code styling

The auto grader will include tests that check whether or not your code adheres to Python's [PEP 8](#) styling guidelines relative to the [use of whitespace](#) 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. Best to learn now rather than have to unlearn bad practices in the future.

In particular, 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(uniquname, domain='umich.edu'):
    """TODO"""
    return f"{uniquname}@{domain}"

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

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

# Incorrect
email_address = create_email_address(uniquname = '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
```

11.0 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 and debug your code.

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.


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

11.2 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/guides/>. You can then set breakpoints and review your code in action by "stepping over" lines and "stepping into" function calls.

11.3 File compare

Each test fixture CSV and JSON file (prefixed with `fmt-`) 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 indentation 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.

12.0 Gradescope submissions

You may submit your problem solution to Gradescope as many times as needed before the expiration of the exam time. Your **final** submission will constitute your exam submission.

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

13.0 auto grader / manual scoring

If your auto-graded submission does not earn full points the teaching team will review your submission manually. Partial credit **may** be awarded for submissions that fail one or more auto grader tests if the teaching team (at our sole discretion) deem a score adjustment warranted.

If you submit a partial solution, please include (if you have time) comments 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.