# SI 506: Last Assignment

## 1.0 Dates

- Available: Tuesday, 7 December 2021, 4:00 PM Eastern
- Due: Monday, 20 December 2021, on or before 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.

The last assignment features a Star Wars theme and utilizes data sourced from the Star Wars API (SWAPI), Wookieepedia, and Wikipedia.

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

## 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

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

## 5.0 Files

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

1. `swapi.md`: assignment instructions
2. `swapi.py`: script including a `main()` function and other definitions and statements
3. `sw_utils.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.

## 5.1 `swapi.py`

The `swapi.py` file contains class and 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, )
- Make HTTP GET requests to the remote SWAPI service
- Instantiate class instances and call class methods
- Call functions
- Assign values to specified variables
- Write data to local files encoded as JSON

Implementing functions and class methods involves replacing the placeholder `pass` statement with working code. You may be asked to add missing parameters to function and/or method definitions. You may also be asked to define a function or class method in its entirety (less the Docstring) as directed per the instructions.

## 5.2 `swapi_utils.py`

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

## 5.3 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 template file.

### 5.4 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 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.

💡 In VS Code you can compare the JSON files you generate against the test fixture JSON files. After generating a file right click on it and select the "Select for Compare" option. Then right click on the appropriate test fixture JSON file and select the "Compare with Selected". Lines highlighted in red indicate character and/or indention mismatches. If any mismatches are encountered revise your code and regenerate the file.

## 6.0 Challenges

The last assignment comprises a maximum of fifteen (15) challenges. The teaching team recommends that you complete each challenge in the order specified in the instructions.

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

The challenges cover all topics introduced between weeks 01 and 14:

- 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`)
- String formatting: formatted string literal (f-string)
- Data structures
  - Sequences
    - Indexing and slicing (subscript notation)
    - 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
  - `del` statement
- Shallow and deep copying using the `copy` module
- 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
- 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()`
            - `getattr()`, `setattr()`, `hasattr()`, `delattr()`
        - Awareness
            - `dict()`, `dir()`, `id()`, `enumerate()`, `list()`, `slice()`, `str()`, `tuple()`
    - 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
        - Reading and interpreting function Docstrings
    - `main()` function (entry point for program or script)
    - Variable scope (global and local variables)
- 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`
- Classes and objects
    - Class composition
    - Instantiating class instances with `__init__()`
    - Representing a class instance as a string (`__str__()`)
    - Returning JSON-friendly representations of class instances
    - Implementing custom instance methods
    - Working with the `object.__dict__` attribute
    - Creating class instance variables dynamically using a dictionary or list

    - Reading and interpreting class and class method Docstrings
- Modules
    - Importing modules
    - Creating and using custom modules
- Web API
    - `requests` module installation (via pip) and use
    - HTTP GET request/response
    - JSON payloads
- Exceptions, exception handling, and debugging
    - `try` / `except` statements (handling `ValueError` and `AttributeError` exceptions)

## 7.0 A note on code styling

The auto grader will include tests that check whether or not your code adheres to Python 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.

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(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')
```

## 8.0 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).

## 9.0 auto grader / manual scoring

If the auto grader is unable to grade your submission successfully with a score of 1800 points the teaching team will review your submission. 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.