# *Advanced Topics in Computer Science (420-G40-HR)*

# *Assignment 1 – To Do List*

Date assigned: September 15, 2021

Date due: October 1, 2021

**Learning Objectives**

Upon successful completion of this assignment, the student will be able to:

* Use lists, tuples and dictionaries
* Use comprehensions and other list operations
* Use Python classes as applicable
* Read and write to a file
* Use exceptions for error handling

To do:

1. You are going to create a Python program to manage a To Do List. There will be no GUI for this assignment; you will be using a command line interface (CLI) style WITHIN Python.
2. The functionality for the To Do list is:
3. Add a new task
4. Change an existing task
5. Remove an existing task
6. Mark a task as complete
7. Print a formatted list of all tasks
8. Print a formatted list of not-completed tasks in priority order
9. Purge (remove) completed tasks
10. Change/add the (optional) priority of a task
11. Change/add the (optional) project for a task
12. The To Do list is:
13. Stored in a flat file called tasks.txt. Read the file when the program starts. If it is not there, create the file when creating the first task.
14. Stored, read and managed as a Python Dictionary. The key for the dictionary is task\_id and the value is comprised of the other task fields (see below).
15. The task\_id is a unique number, but do not maintain the next task id externally. Simply get the largest task id from the list and increment it.
16. In addition to the task\_id, the fields in the dictionary for each task are:
    * 1. desc; a string describing the task
      2. completed: True or False depending on whether or not the task is complete.
      3. priority, Value: 1 to 4 with 1 being the highest. Note: Tasks do NOT have to have a priority, it is an optional field. When printing priority tasks are printed first from highest to lowest (1-4) followed by the tasks without priority
      4. project: a string consisting of ONE word which is the project the task belongs to.
17. A CLI is to be used. The user can enter the entire command on one line or be prompted for the fields. The commands are: add, upd, rem, done, list, purge. Optionally, the user can enter the commands when starting the program.
18. Examples:

|  |  |
| --- | --- |
| **Command** | **Result** |
| add Complete work on prototype conversion | Creates a new task with the latest task\_id, the desc of Complete work on prototype conversion, completed of False and no other entries |
| add Study for Programming test !1 #school | Creates a new task with the latest task\_id, the desc of Study for Programming test, completed of False, priority of 1 and project of school |
| add | Prompts the user for description and creates a task per first example |
| upd 2 Study for Web test | Updates the desc of task 2 with the new desc. No other fields change. Error if no task 2 exists. Prompt if updated desc is not provided. |
| rem 3 | Removes task 3. Error if task is not there. Confirmation that task is removed. |
| done 2 | Marks task 2 as completed. Error if no task 2 exists. |
| list all | Lists all tasks in task id order. All fields listed. |
| list todo | Lists all tasks not complete in order by priority. Tasks with no priority at bottom in number order. |
| purge | Removes all completed tasks |

1. So, the options are:
2. python g40a01.py  
   >>>add  
   Enter task description>>>  
   Enter completed date>>>  
   …  
   Task 2 added
3. python g40a01.py  
   >>> add This new task !1 #school  
   Task 2 added  
   >>>
4. python g40a01.py add This new task !1 #school  
   Task 2 added
5. Do not write massive blocks of code. Each block of code that performs a specific task should be in a function! Think singularity of purpose.
6. Make sure you use the efficient methods for outputting strings with the f’ format.
7. Use magic functions appropriately. At a minimum you should have \_\_init\_\_, \_\_str\_\_, \_\_repr\_\_, \_\_eq\_\_ functions.
8. Make sure you perform error checking. In Python this is done by attempting the operation and then catching the error if it occurs. More next week.
9. Think Pythonically. Try to use comprehensions where applicable. Follow the guidelines established in the [PEP 8 Styling Guide](https://www.python.org/dev/peps/pep-0008/). This must include variable and function names.
10. Don’t worry about graphics or a GUI.

To do:

1. Generate a list of test case scenarios to fully test the program. Each test case should describe the scenario, the input and the expected result. Save this in a Word document.
2. Code the program and fully test the program for all error conditions described in the specification.

**To submit**

1. A ZIP format (*youruserid\_*G40A01.zip) containing all submitted files on Moodle.