Students:

This content is controlled by your instructor, and is not zyBooks content. Direct questions or concerns about this content to your instructor. If you have any technical issues with the zyLab submission system, use the **Trouble with lab** button at the bottom of the lab.

11.3 "List" menu option

Background

(*Skip to the "What is a task" section below if you do not care about the real-world connection and relevance of this project.*)

Oftentimes, data is stored in a very compact but not human-friendly way. Think of how our forum questions may be stored in Piazza databases somewhere in the cloud. One way a question-object may be stored is this:

```
{
    'title': 'Extension',
    'description': 'hey i submitted my lab 2 hours after the
deadline. Do you think there is...',
    'author_uid': 'a_39',
    'answers': [],
    'is_private': False,
    'tags': ['logistics', 'week7'],
    'created': '2022-05-21T05:28:43Z',
    ...
}
```

Obviously, this format is not user-readable, so Piazza's frontend team produced many lines of code to render and display this content in a way that's more understandable to us, Piazza users.

We have a mini-version of the same task coming up. Given a list of similar complex objects (represented by python dictionaries), display them in a user-friendly way that we define.

Since we will be doing this for to-do items, we need to figure out how and what to store for each task.

What is a task

- In this project, for the ease of testing, you can **hard-code** a nested list of tasks at the top of your **main program**, after the dictionary `the_menu`. We will refer to it as `all_tasks`. This **nested list** will contain **dictionaries** with each task's information.

In this project, **a task** is a **dictionary** object that is guaranteed to have the following keys:

- "name": a string with the task's name
- "info": a string with the task's details/description
- "priority": an integer, representing the task's priority (defined by a dictionary in the main program)
- "duedate": a valid date-string in the US date format: <MM>/<DD>/<YEAR> ("01/02/2022" represents January 2nd, 2022).
- "done": a string representing whether a task is completed or not

Here is an example of what a dictionary with a **single task** could look like:

```
{
    'name': 'Finish CSW8 project',
    'info': 'Submit all files to Gradescope',
    'priority': 5,
    'duedate': '06/05/2022',
    'completed': 'no'
}
```

In this project, for the ease of testing, you can **hard-code** a nested list of tasks at the top of your **main program**, after the dictionary `the_menu`. We will refer to it as `all_tasks`.

```
all_tasks = [
    {
        "name": "Call XYZ",
        "info": "",
        "priority": 3,
        "duedate": '05/28/2022',
        "done": 'yes'
    },
    {
        "name": "Finish checkpoint 1 for CSW8",
        "info": "Submit to Gradescope",
        "priority": 5,
        "duedate": '06/02/2022',
        "done": 'no'
    },
    {
        "name": "Finish checkpoint 2 for CSW8",
        "info": "Implement the new functions",
        "priority": 5,
        "duedate": '06/05/2022',
        "done": 'no'
    }

]
```

Your TODO

- Create a variable to hold a nested list of tasks at the top of your **main program**. We will refer to it as `all_tasks`.
- Additionally, you need to add two more dictionaries to your **main program**:
  - `list_menu` will contain the "List" menu suboptions
  - `priority_scale` will contain the mapping of the integer priority values to their textual interpretation

```
list_menu = {
    "A": "all tasks",
    "C": "completed tasks",
    "I": "incomplete tasks"
}

priority_scale = {
    1: "Lowest",
    2: "Low",
    3: "Medium",
    4: "High",
    5: "Highest"
}
```

- Next, add the following code to your **main program** to implement the listing of the tasks - you do not need to change it:

```
    elif opt == 'L':
        if all_tasks == []:
            print("WARNING: There is nothing to display!")
            # Pause before going back to the main menu
            input("::: Press Enter to continue")
            continue
        subopt = get_selection(the_menu[opt], list_menu)
        if subopt == 'A':
            print_tasks(all_tasks, priority_scale)
        elif subopt == 'C':
            print_tasks(all_tasks, priority_scale, completed =
'yes')
        elif subopt == 'I':
            print_tasks(all_tasks, priority_scale, completed =
'no')
```

- In your **task_functions.py**, copy the `get_selection()` function given below - you will use this function as is. We have updated the code for it, so that the user can change their mind and return to the main menu.

```
######## LIST OPTION ########

def get_selection(action, suboptions, to_upper = True, go_back
= False):
    """
    param: action (string) - the action that the user
            would like to perform; printed as part of
            the function prompt
    param: suboptions (dictionary) - contains suboptions
            that are listed underneath the function prompt.
    param: to_upper (Boolean) - by default, set to True, so
            the user selection is converted to upper-case.
            If set to False, then the user input is used
            as-is.
    param: go_back (Boolean) - by default, set to False.
            If set to True, then allows the user to select the
            option M to return back to the main menu

    The function displays a submenu for the user to choose
from.
    Asks the user to select an option using the input()
function.
    Re-prints the submenu if an invalid option is given.
    Prints the confirmation of the selection by retrieving the
    description of the option from the suboptions dictionary.

    returns: the option selection (by default, an upper-case
string).
            The selection be a valid key in the suboptions
            or a letter M, if go_back is True.
    """
    selection = None
    if go_back:
        if 'm' in suboptions or 'M' in suboptions:
            print("Invalid submenu, which contains M as a
key.")
            return None
    while selection not in suboptions:
        print(f"::: What would you like to {action.lower()}?")
        for key in suboptions:
            print(f"{key} - {suboptions[key]}")
        if go_back == True:
            selection = input(f"::: Enter your selection "
                              f"or press 'm' to return to the
main menu\n> ")
        else:
            selection = input("::: Enter your selection\n> ")
```

```
        if to_upper:
            selection = selection.upper() # to allow us to
input lower- or upper-case letters
        if go_back and selection.upper() == 'M':
            return 'M'

    print(f"You selected |{selection}| to",
          f"{action.lower()} |
{suboptions[selection].lower()}|.")
    return selection
```

- Define the `print_task()` and `print_tasks()` functions.
  - ⚠ Add the implementation based on the documentation provided below.
  - ⚠ See the example of the `print_task()` output provided below in the Sample Program Flow section.
  - you should have already added the `get_written_date()` function implementation from LAB 7.18

List an individual task (dictionary).

```
def print_task(task, priority_map, name_only = False):
    """
    param: task (dict) - a dictionary object that is expected
            to have the following string keys:
    - "name": a string with the task's name
    - "info": a string with the task's details/description
            (the field is not displayed if the value is empty)
    - "priority": an integer, representing the task's priority
        (defined by a dictionary priority_map)
    - "duedate": a valid date-string in the US date format:
<MM>/<DD>/<YEAR>
            (displayed as a written-out date)
    - "done": a string representing whether a task is completed
or not

    param: priority_map (dict) - a dictionary object that is
expected
            to have the integer keys that correspond to the
"priority"
            values stored in the task; the stored value is
displayed for the
            priority field, instead of the numeric value.
    param: name_only (Boolean) - by default, set to False.
            If False, then only the name of the task is
printed.
            Otherwise, displays the formatted task fields.
```

```
    returns: None; only prints the task values

    Helper functions:
    - get_written_date() to display the 'duedate' field
    """
```

List the tasks stored in a nested task list.

- Make sure that the `completed` field correctly displays either all tasks or **only** the completed/incomplete tasks, depending on what was selected by the user in the main program.

```
def print_tasks(task_list, priority_map, name_only = False,
                show_idx = False, start_idx = 0, completed =
"all"):
    """
    param: task_list (list) - a list containing dictionaries
with
            the task data
    param: priority_map (dict) - a dictionary object that is
expected
            to have the integer keys that correspond to the
"priority"
            values stored in the task; the stored value is
displayed
            for the priority field, instead of the numeric
value.
    param: name_only (Boolean) - by default, set to False.
            If False, then only the name of the task is
printed.
            Otherwise, displays the formatted task fields.
            Passed as an argument into the helper function.
    param: show_idx (Boolean) - by default, set to False.
            If False, then the index of the task is not
displayed.
            Otherwise, displays the "{idx + start_idx}." before
the
            task name.
    param: start_idx (int) - by default, set to 0;
            an expected starting value for idx that
            gets displayed for the first task, if show_idx is
True.
    param: completed (str) - by default, set to "all".
            By default, prints all tasks, regardless of their
            completion status ("done" field status).
            Otherwise, it is set to one of the possible task's
"done"
```

```
            field's values in order to display only the tasks
with
            that completion status.

    returns: None; only prints the task values from the
task_list

    Helper functions:
    - print_task() to print individual tasks
    """
    print("-"*42)
    for ...: # go through all tasks in the list
        if show_idx: # if the index of the task needs to be
displayed
            print(f"{...}.", end=" ")
        if completed == "all":
            print_task(task, priority_map, name_only)
        elif ... == completed:
            print_task(task, priority_map, name_only)
```

Sample Program Flow

Assuming the tasks hard-coded above, below is a sample program output for listing All Tasks:

```
You selected option L to > List.
::: What would you like to list?
A - all tasks
C - completed tasks
I - incomplete tasks
P - incomplete tasks, sorted by priority
D - incomplete tasks, sorted by deadline
::: Enter your selection
> a
You selected |A| to list |all tasks|.
------------------------------------------
Call XYZ
  * Due: May 28, 2022  (Priority: Medium)
  * Completed? yes
Finish checkpoint 1 for CSW8
  * Submit to Gradescope
  * Due: June 2, 2022  (Priority: Highest)
  * Completed? no
Finish checkpoint 2 for CSW8
  * Implement the new functions
  * Due: June 5, 2022  (Priority: Highest)
```

```
  * Completed? no
::: Press Enter to continue
```

If C or I is selected, only the tasks that have that status are shown:

```
::: Enter your selection
> c
You selected |C| to list |completed tasks|.
-----------------------------------------
Call XYZ
  * Due: May 28, 2022  (Priority: Medium)
  * Completed? yes
::: Press Enter to continue
```

398092.2571084.qx3zqy7

LAB ACTIVITY 11.3.1: "List" menu option 0 / 1

main.py

```
1
```

Develop mode | Submit mode

Run your program as often as you'd like, before submitting for grading. Below, type any needed input values in the first box, then click **Run program** and observe the program's output in the second box.

Enter program input (optional)

If your code requires input values, provide them here.

Run program

Input (from above) → main.py (Your program)

Program output displayed here

Coding trail of your work What is this?

```
History of your effort will appear here once you begin
working on this zyLab.
```

Trouble with lab?