11.4 "Add" menu option

Add the following branch to your **main program**; replace the ellipses with the appropriate values:

```
    elif opt == 'A':
        continue_action = 'y'
        while continue_action == 'y':
            print("::: Enter each required field, separated by commas.")
            print("::: name, info, priority, MM/DD/YYYY, is task done? (yes/no)")
            ... = input("> ") # TODO: get and process the data into a list
            ...
            result = get_new_task(...) # TODO: attempt to create a new task
            if type(result) == dict:
                ... # TODO: add a new task to the list of tasks
                print(f"Successfully added a new task!")
                print_task(result, ...)
            elif type(result) == int:
                print(f"WARNING: invalid number of fields!")
                print(f"You provided {result}, instead of the expected 5.\n")
            else:
                print(f"WARNING: invalid task field: {result}\n")

            print("::: Would you like to add another task?", end=" ")
            continue_action = input("Enter 'y' to continue.\n> ")
            continue_action = continue_action.lower()
            # ----------------------------------------------------------------
```

In your **functions file**, define `get_new_task()` that takes two parameters: a list and a dictionary (see more details below).

```
def get_new_task(..., ...):
    """
    Document the function correctly
    """
```

The function **returns** different types of values, depending on whether it succeeds or fails.

- The function expects the first parameter to be a list with 5 strings.
  - If the size of the list is not correct, then the function returns the **integer** size of the provided list. E.g., calling `get_new_task()` with an empty list as its first argument should return `0` and so on.
  - If any of the elements on the list are not of type string, the `get_new_task()` returns a **tuple** with `("type", <value>)`, where the `<value>` is substituted with the first corresponding value from the list that was not a string.
  - Each validation function will also be in charge of validating that its input parameter (the item from the list) is of the correct type (just in case it is called separately).
- If the size of the list is correct, the function calls the helper functions to validate the fields.
  - If the validation succeeds, returns a **new dictionary** with the task keys set to the provided parameters (stripped of whitespace and converted to the proper type, if necessary).
  - If any of the validation functions fail, returns a **tuple** with the name of the parameter and the corresponding value/parameter that caused it to fail.

The first input parameter is a **list** of all needed task components/values stored **as strings**: * the order of the fields is **name, info, priority, date, task completion** (is the task done? yes/no) * example valid input list might look like `['Book tickets', 'Verify dates', '3', '05/05/2022', 'no']` * if the provided field values are valid, return a new dictionary:

```
{
    "name": ...,
    "info": ...,
    "priority": ...,
    "duedate": ...,
    "done": ...
}
```

- the second parameter to `get_new_task()` is a dictionary that contains the mapping between the integer priority value (key) to its representation (e.g., key 1 might map to the priority value "Highest" or "Lowest" - do not hard-code these values in the function!)

```
# using the default priority_scale
sample_task_list = ['Book tickets', 'Verify dates', '3',
'05/05/2022', 'no']
expected_result = {'name': 'Book tickets', 'info': 'Verify
dates', 'priority': 3, 'duedate': '05/05/2022', 'done': 'no'}
assert get_new_task(sample_task_list , priority_scale) ==
expected_result
```

Note that `get_new_task()` relies on a lot of helper functions that validate the different task fields. The helper functions will need to be added first, before defining it. ⚠ **Pay close attention to the number of parameters and their types: *each* validation function *first* checks that the input parameters are of the correct type.** Make sure that you use the `type()` function!

- define `is_valid_name()`
  - takes as input a string that is supposed to contain between 3 and 25 characters (inclusive of both)
  - returns a Boolean True if the text is of the valid length; False, otherwise
- the `"info"` field can be empty and has no specific requirements, so we do not need to validate it.
- define `is_valid_priority()` that expects **two** input parameters:
  - a string that is expected to contain an integer priority value (uses `str.isdigit()`); gets validated against the keys in the dictionary, provided as the second parameter
  - a dictionary that contains the mapping between the integer priority value (key) to its representation (e.g., key 1 might map to the priority value "Highest" or "Lowest")
  - returns a Boolean True if the text contains an integer value that maps to a key in the provided dictionary; False, otherwise
- define `is_valid_date()`
  - takes as input a string that is expected to contain a date in the U.S. format (MM/DD/YYYY)
  - validates each of the date components - call the date component validation functions from LA 7.19
- define `is_valid_completion()`
  - takes as input a string that is expected to contain a text "yes" or "no"
  - returns a Boolean True if it's a text with the valid value; False, otherwise

Sample Program Flow

Below is a demo of adding the incorrect task values to check each validation step, until the correct task info is formed (note the empty `"info"` field):

```
You selected option A to > Add.
::: Enter each required field, separated by commas.
::: name, info, priority, MM/DD/YYYY, is task done? (yes/no)
> ,,,,
WARNING: invalid task field: ('name', '')

::: Would you like to add another task? Enter 'y' to continue.
> y
```

```
::: Enter each required field, separated by commas.
::: name, info, priority, MM/DD/YYYY, is task done? (yes/no)
> implement get_new_task(),,,,
WARNING: invalid task field: ('priority', '')

::: ...
> implement get_new_task(),, 5,,
WARNING: invalid task field: ('duedate', '')

::: ...
> implement get_new_task(),, 5, 6/1/22,
WARNING: invalid task field: ('duedate', '6/1/22')

::: ...
> implement get_new_task(),, 5, 6/1/2022,
WARNING: invalid task field: ('done', '')

> implement get_new_task(),, 5, 6/1/2022, y
WARNING: invalid task field: ('done', 'y')

> implement get_new_task(),, 5, 6/1/2022, no
Successfully added a new task!
implement get_new_task()
  * Due: June 1, 2022  (Priority: Highest)
  * Completed? no
::: Would you like to add another task? Enter 'y' to continue.
>
```

398092.2571084.qx3zqy7

LAB ACTIVITY | 11.4.1: "Add" menu option 0 / 1

main.py

```

```

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?