CS 1110 Spring 2020, Assignment 3: Tre1110

Updates:

The assignment itself, with updates marked in orange, begins on the next page. On this "page 0", we also document the time, location, and nature of the updates, in reverse chronological order,

- Monday Mar 2, 11:30pm:
  - Explanation for how to handle grouping over this two-part assignment (pg 3, pg 9)
  - Clarification of spec for `space_available` (pg 6)
  - Added Section 5.5.1 "Batch" testing allowed to `test_add_task` (pg 8)
  - (Extremely minor) Rationalized number in Section 6 (pg 9)

CS 1110 Spring 2020, Assignment 3: Tre1110*

1 Motivation: Get things done!

To-do lists. Surely you've got one, and here we are adding to something to it. But this assignment revolves around an effective strategy for dealing with to-do items: put your tasks on your calendar!

Contents

2 Rules (all the same as for previous assignment)

There is no revise-and-resubmit for this or any subsequent assignment unless otherwise noted.

2.1 You need to re-group (so to speak) on CMS, regardless of prior groupings

You may work alone or with just one other person, who can be someone you've grouped with before in CS1110, or a different person.

If you are partnering, regardless of whether you were grouped in a previous assignment, the two of you must still form a new group *specifically for this assignment* on CMS before submitting.[1]

If your partnership dissolves, see the course Academic Integrity description about "group divorce" for what to do.

2.2 (Dis-)allowed collaborations and documentation requirements

Our policies are laid out in full on the course Academic Integrity page, but we re-state here **the main rules**: where "you" means you and, if there is one, your one CMS-registered group partner,

1. Never look at, access or possess any portion of another group's work in any form.
2. Never show or share any portion of your work in any form to anyone except a member of the course staff.

*Authors: Lillian Lee, Rhea Bansal, Kevin Cook, Will Xiao.

[1]This links your submission "portals".

3. Never request solutions from outside sources; for example, on online services like StackOverflow.
4. DO specifically acknowledge by name all help you received, whether or not it was "legal" according to (1)-(3).

3 Python You Are NOT Allowed To Use In This Assignment

See Section 5.1.

4 Due dates

These deadlines were chosen to:

- strongly encourage you to write test cases *before* writing function bodies — we swear, this practice makes you a better programmer — and
- allow us to release solutions on Monday March 9, before the prelim on Tuesday March 10.

We've broken this homework into two parts, so there are two associated "assignments" on CMS.

(a) If you are partnering: well before submission, for assignment A3tests, follow the "How to form a group" instructions on the course Assignments page. Both parties need to act on CMS in order for the grouping to take effect.

(b) By 2pm on Sat Mar 7, submit whatever you have done at that point on `a3_todo_test.py` to CMS for Assignment A3tests, following steps 1-3 in the "Updating, verifying, and documenting assignment submission" section of the course Assignments page. It is OK if you haven't finished working on it yet.[2]

(c) By **11:59pm on Sat Mar 7**, make your final submission of file `a3_todo_test.py` to Assignment A3tests, again following the aforementioned steps 1-3.

(d) By 2pm on Sun Mar 8, submit whatever you have done at that point on `a3_todo.py` to CMS for Assignment A3fns, again following the aforementioned steps 1-3. It is OK if you haven't finished working on it yet. **If you grouped for A3tests**: The staff will manually group you on A3fns with your A3tests partner *after* all the A3fns submissions are in. This means you will not be able to see what your partner submitted on CMS until after the staff does the behind-the-scenes grouping. The *latest* of the submissions made by you and/or your partner will end up being the submission for your group, and it is OK if only one of you ever ends up submitting files; but you should coordinate with your partner to make sure one of you doesn't make a submission that ends up overwriting the other's in a way that wasn't intended! **If you didn't group for A3tests**: you shouldn't work with someone for A3fns.

(e) By **11:59pm on Sun Mar 8**, make your final submission of `a3_todo.py` to Assignment A3fns, again following the aforementioned steps 1-3.

5 Task

5.1 Overview

One goal of this assignment is to provide you more experience with larger code bodies and more complexly interrelating object structures. The most difficult part of this assignment will probably not be the writing of the functions, but understanding the structure of the classes and how to write testcases for functions dealing with these classes.

A second major goal of this assignment is practice with for-loops, since for-loops are a great way to get things done. This goal implies that, for each of the functions we have you implement, *we reserve the right to assign no credit for code that isn't fundamentally based on an explicit for-loop, or uses a for-loop in the way we ask for*, even if the code fulfills the specification. Good references on for-loops besides lecture and the text: some examples from Spring 2018; the completed for-loops we've given you in the A3 code.[3]

[2]The 2pm checkpoints provide you a chance to alert us if any problems arise. Since you've been warned to submit early, do not expect that we will accept work that doesn't make it onto CMS on time, for whatever reason. There are no so-called "slipdays" and there is no "you get to submit late at the price of a late penalty" policy. Of course, if some special circumstances arise, contact the instructor(s) immediately.

[3]You might also want to consult the assignment and solutions for A3 from Spring 2018 and Spring 2017 for extra practice, but it might not be worth trying to understand the setups in those A3s.

As an application of for-loops, the assignment files (contained in this zip file) give you the skeleton for writing and testing a program that helps you schedule your tasks. Here's a sample interaction for a completed implementation:

```
llee A3 > python a3_app.py
Welcome to the Tre1110 planner!
Here, you can plan out the next week (from Sunday - Saturday).
Type 'print <day>' to view your schedule for that day, or 'print all'
to view your whole week at once.
To add a task to the planner, type 'add task'.
Type 'q' to quit.
What would you like to do?
> add task
What is the name of your task?
> finish the assignment writeup
How many hours will your task take (integers please)?
> 1
What day would you like to add this task to?
> Monday
What hour will your task start (24 hour time please)?
> 1
Successfully added!
What would you like to do now? (add task, print <day>, print all, q)
> add task
What is the name of your task?
> answer DUS emails
How many hours will your task take (integers please)?
> 2
What day would you like to add this task to?
> Monday
What hour will your task start (24 hour time please)?
> 1
Sorry, couldn't add it to your schedule at that time!
What would you like to do now? (add task, print <day>, print all, q)
> add task
What is the name of your task?
> answer DUS emails
How many hours will your task take (integers please)?
> 2
What day would you like to add this task to?
> Monday
What hour will your task start (24 hour time please)?
> 14
Successfully added!
What would you like to do now? (add task, print <day>, print all, q)
> add task
What is the name of your task?
> meet with chair
How many hours will your task take (integers please)?
> 1
What day would you like to add this task to?
> Tuesday
What hour will your task start (24 hour time please)?
> 13
Successfully added!
```

```
What would you like to do now? (add task, print <day>, print all, q)
> print all
Sunday:
No events scheduled.

Monday:
1:00-2:00 finish the assignment writeup
14:00-16:00 answer DUS emails

Tuesday:
13:00-14:00 meet with chair

Wednesday:
No events scheduled.

Thursday:
No events scheduled.

Friday:
No events scheduled.

Saturday:
No events scheduled.

What would you like to do now? (add task, print <day>, print all, q)
> q
Goodbye and good luck getting all your tasks done!
```

5.2 Classes Task and Day

File `a3_classes.py` defines classes Task and Day. We haven't discussed classes in detail yet, but here's all you need to know about these two classes.

Task objects have two atrributes, a `name` and a `length`. A call `Task("shop at Wegmans", 1)` creates a Task with name "go to Wegmans" that takes 1 hour to complete.

Day objects have three attributes, a `name`; a 24-item list `time_slots`, one item for each hour of the day; and `num_tasks_scheduled`, which should be the number of tasks scheduled for that day. A call `Day("My birthday")` creates a Day with name "My birthday", nothing (yet) scheduled for any of its 24 hours, and attribute `num_tasks_scheduled` set to 0.

More information is available in the docstrings of the two class definitions.

5.3 Running example

Figure 1 shows an example of how these objects might be used.

In global space, you see 5 predefined Tasks whose identifiers have been stored in variables `predef1` through `predef5`; these have already been scheduled, as described later in this section. There are also two Tasks waiting to be scheduled, `todo_call` and `todo_hack`. Finally, the variable `DAY_POOL` holds a list of Days during which Tasks can be scheduled.

`DAY_POOL[0]` is the Day with name "Monday March 2", which has three tasks already scheduled for it. Looking at `DAY_POOL[0].time_slots`, which is a list of 24 hours, we see that the Task with id id1 has been scheduled for the hours 3, 4, and 5; this Task has the name "SPCA shift". Hours 10, 11, 12, and 13 are devoted to the Task with id id4, named "Study for prelims". Finally, hours 21, 22, and 23 are filled by the Task with id id3, "cure cancer". All the other entries in this list are None, meaning that nothing has been scheduled there.

You should similarly be able to see that `DAY_POOL[1]` "Monday March 9" has the Task "work-study" (id2) from midnight to 2am but is otherwise free; `DAY_POOL[2]` "Ides of March" is (ominously?) completely free, and `DAY_POOL[3]` "Monday March 16" is completely occupied with Task id5 "Conquer enemies".

The Tasks `todo_call` and `todo_hack` have not yet been slotted into `DAY_POOL`; your task[4] is to complete the functions in file `a3_todo.py` to make it easy to add Tasks to a specific Day.

[4]That wasn't intentional!

5.4 Specifications For The Functions To Write

The name `sirialize_days` is meant to evoke Siri, as in “Siri, tell me what I have scheduled in the next couple of days.”

```python
def sirialize_days(day_list):
    """
    Prints out the tasks for each day in day_list, first by day in the order
    in which they appear in day_list, and then chronologically within each day.

    Parameter day_list: The list of days to print data out for.
    Precondition: day_list is non-empty list of Day objects (no None objects).
    """
```

```python
def num_hours_busy(day):
    """
    Returns: the number of hours that are busy in Day `day`.

    Parameter day: the Day to check.
    Precondition: day is a Day object.
    """
```

Addition to spec below: ”and False otherwise”

```python
def space_available(slots, start_time, end_time):
    """
    Returns: True if there is space available to schedule an event in `slots`
    from `start_time` (inclusive) to `end_time` (exclusive) and False otherwise}.
    So, `space_available(agenda, 14, 16)` queries whether 2-4 pm is available,
    and it doesn't matter is something is scheduled at 4pm.

    There is space available if all the elements of slots starting from
    `start_time` (inclusive) to `end_time` (exclusive) are all None.

    Parameter slots: The list of timeslots to check.
    Precondition: slots is a list of length 24, each element of which is either
    a Task object or None.

    Parameter start_time: The starting time to check from, inclusive.
    Precondition: start_time is an int, and a valid index into `slots`
    (0 <= start_time < 24), and start_time < end_time.

    Parameter end_time: The ending time to check to, exclusive.
    Precondition: end_time is an int for a valid ending time
    (0 <= end_time <= 24), and start_time < end_time.
    """
```

```python
def add_task(day, task, start_time):
    """
    Returns: True if Task `task` is added to Day `day` at time `start_time`
    successfully, False otherwise.

    This function attempts to add the given task to the given day and time.
    If the specified timeslot is free, then the `time_slots` attribute of the
    Day object is modified to reflect that the task has been added, meaning that
    each hour timeslot that this task occupies must be filled in the list.
    This means that if a task takes multiple hours, there must be multiple
    references to that Task object in the timeslot list, one for each hour
    the task takes.

    If the task can successfully be added, this function also increments the
    num_tasks_scheduled attribute of Day `day` by 1 to reflect this new task.

    A task cannot be added to the Day object if any Task that is currently
    scheduled for that day would overlap with `task` if it were added at
    `start_time`. For example, adding a 3 hour long task at 2:00 should fail if
    there are any tasks scheduled between the hours of 2:00 and 5:00.

    A task also cannot be added if scheduling the task at the given
    time would require rolling over to the next day to finish. As an example,
    trying to schedule a task that takes two hours to complete at 23:00
    (i.e. start_time == 23) should fail.

    If the given task cannot be added, this function returns False and does
    not modify anything.

    Parameter day: The Day object to attempt to add Task `task` to.
    Precondition: day is a Day object.

    Parameter task: The Task object to add to Day `day`.
    Precondition: task is a Task object.

    Parameter start_time: The hour to start this task at.
    Precondition: start_time is an int , and 0 <= start_time < 24.
    """
```

5.5 Writing Test Cases — due Sat Mar 7

From reading the specifications above, we hope you'll have a sense of some situations that need testing: for example, does `num_hours_busy` return the right answer for a completely empty or a completely full day? And does `space_available`? What if someone tries to add a Task to a time slot that is already occupied?

To make writing up test cases easier for you, we provided code you can use in `a3_todo_test.py` that sets up some of the situation depicted in Figure 1, although it doesn't put any of the Tasks into `DAY_POOL`:

```
import a3_classes

# globals setting things up somewhat like assignment diagram
predef1   = a3_classes.Task("SPCA shift", 3)
predef2   = a3_classes.Task("work-study", 2)
predef3   = a3_classes.Task("cure cancer", 3)
predef4   = a3_classes.Task("Study for prelims", 4)
predef5   = a3_classes.Task("Conquer enemies", 24)
todo_call = a3_classes.Task("Call mom", 1)
todo_hack = a3_classes.Task("Programming sprint", 6)
DAY_POOL = [a3_classes.Day("Monday March 2"),  # all time_slots are None
            a3_classes.Day("Monday March 9"),  # all time_slots are None
            a3_classes.Day("Ides of March"),   # all time_slots are None
            a3_classes.Day("Monday March 16")  # all time_slots are None
            ]
```

You'll also see in `test_sirialize_days` how you can add Tasks to a Day without having written `add_task` yet:

```
day = a3_classes.Day("Day in the City")
task_driveout = a3_classes.Task("Drive to NYC", 4)
day.time_slots[4] = task_driveout; day.time_slots[5] = task_driveout
day.time_slots[6] = task_driveout; day.time_slots[7] = task_driveout
```

In addition, we've also provided some testing code for you for each of your functions. You do not need to add anything to `test_sirialize_days`, but you do need to add test cases to the other test functions we've given you. Follow the directions given as comments in the code.

5.5.1 "Batch" testing allowed for `test_add_task`

Given the timeframe of this assignment relative to the upcoming prelim, we've decided that for *just* the test function `test_add_task`, you may save time by "batching" testcases rather than interleaving them, even though this is a less thorough form of testing.

The skeleton we provided you does this "batching", as follows. On the left in the figure below, we illustrate "batched" testing; on the right, "interleaved", and *better* structure.

batched	interleaved
action1	action1
action2	test1 of expectations
action3	action2
action4	test2 of expectations
action5	action3
Test that everything is as expected after the 5 actions are performed.	test3
	action4
	test4
	action5 test5

In the "batched" case, you only have to write one block of testing-expectations code. *But*, if, say action1 made a mistake and action3 made a mistake, but the two mistakes *cancelled out*, the "batched" testing wouldn't catch this.

So, interleaved testing is what you should do in most situations! But in the interests of time, you are permitted to submit batched testing code for A3.

Do note that this leniency towards what test code you submit does *not* relieve you of the responsibility for making sure your function bodies are actually correct according to *our* testing.

5.6 Writing Function Bodies — due Sun Mar 8

Follow the directions given as comments in `a3_todo.py`.

5.7 Grand Finale

You needn't turn in anything for this part, but once you've got your code thoroughly tested and working, if you'd like to see your code in action, see if you can reproduce the interaction depicted in reproduce the interaction in Section 5.1, by running `python a3_app.py`.

6 Pre-Submission Checklist and What to Submit

Files to submit: `a3_todo_test.py` for Assignment A3tests, `a3_todo.py` for Assignment A3fns.

Before submitting, ensure your code obeys the following.[5]

1. Lines are short enough (˜80 characters) that horizontal scrolling is not necessary.
2. You have indented with spaces, not tabs
3. Functions are separated from each other by at least two blank lines.
4. You have removed any debugging `print` statements.
5. You have removed all `pass` statements.
6. You have removed "instruction" comments, such as "`# IMPLEMENT THIS FUNCTION`".
7. If you added any helper functions, these have good docstring specifications. In your specification docstrings, include descriptions of your key test-cases.[6]

Make sure the following are all true before you submit.

8. You've changed the header comments in all files to list the entire set of people and sources that contributed to the code.
9. You (and your partner) have included your NetIDs in the header of all files.
10. The date in the header comments has been changed to when the files were last edited.
11. You have set your CMS notifications settings to receive email regarding grade changes, and regarding group invitations.
12. (reminder) If working with a partner, you have grouped on CMS in A3tests. (One has invited on CMS, and one has accepted on CMS.) The staff will group A3tests partners in A3fns after the deadlines have passed.

7 Advice

7.1 Debugging

1. Many bugs are caused by unintentionally changing the semantics of a variable. Pick informative variable names and/or comment what your intents are. Make sure you update variable values correctly when the situation changes.
2. Section 13.10 of the text ("...especially if you are working on a hard bug") is good advice.
3. Only implement a little bit at a time and test incessantly. Add temporary print statement to check your partial progress as necessary. You don't want an uncaught bug early one messing up a lot of things downstream.
4. You can use Python Tutor to visualize what your code is doing.

7.2 Navigating complex files

1. Atom lets you "fold up" parts of code, such as function bodies, to temporarily hide them. Look for a little down-pointing arrow-head in the lefthand "gutter" of a code window, and click on it.

[5]These requirements up speed up the process of reading/grading hundreds of files.

[6]This is a workaround for the fact that you'll be submitting your testcase files for the required A3 functions earlier, and might write your helpers afterwards.

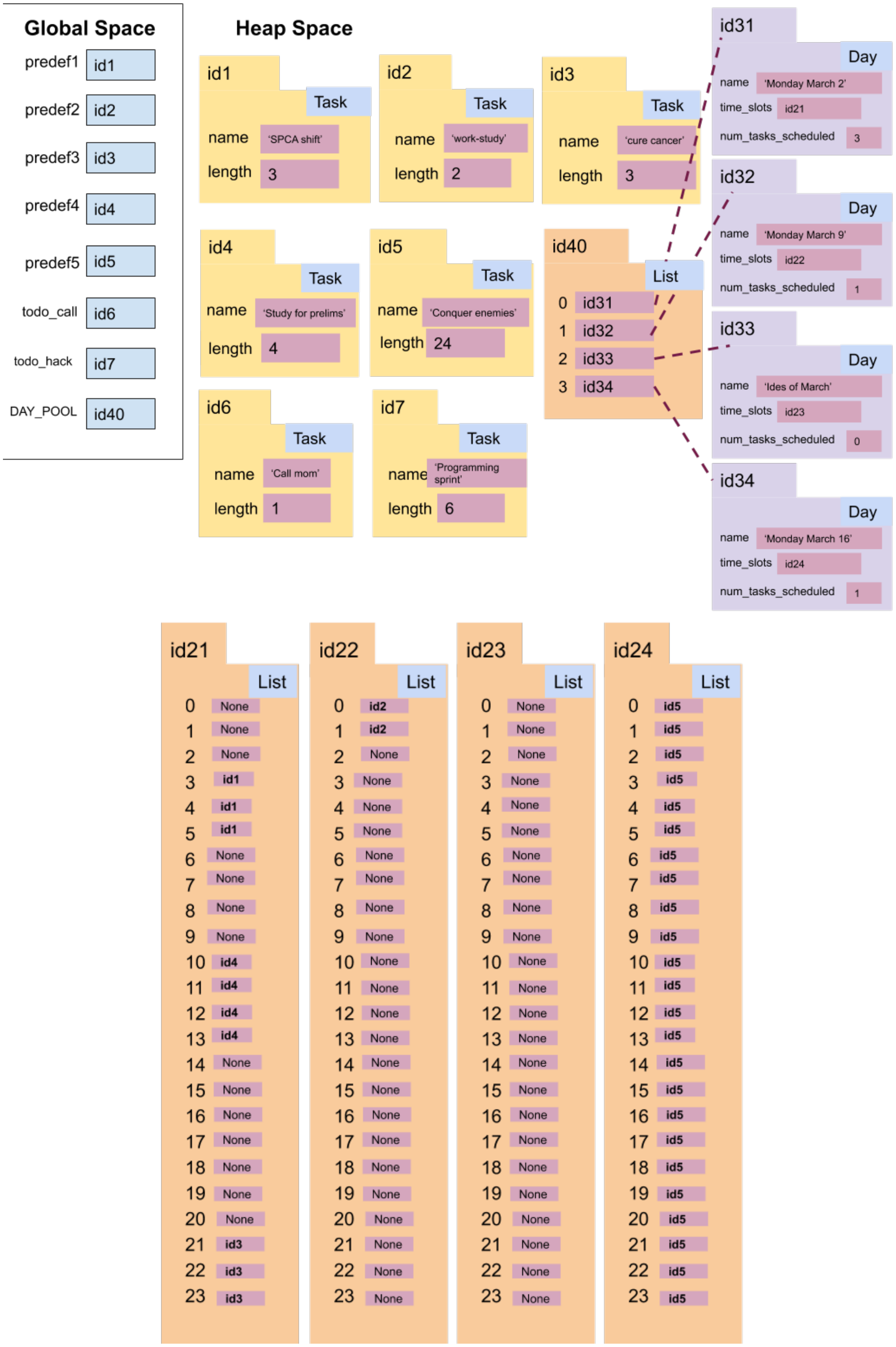

Figure 1: Example set of days and tasks.