# CSI2100-01 Lab 11

In some places and to facilitate reading, we use "␣" to depict a space (blank) character, and "¤" for a "\n" (newline) character.

## Programming Problems

1. Write a Python function that receives three arguments:

   (a) A (possibly empty) dictionary which will be used to store the average temperature for each day of the week.

   (b) A weekday "day".

   (c) The average temperature of weekday "day".

   The function should add the temperature to the dictionary only if it does **not** already contain a temperature for that day. The function should return the resulting dictionary (updated or not). Please use the following stub for your function:

   ```
   def addDailyTemp(mydict, day, temperature):
     '''Add key 'day' and value 'temperature' to the dictionary 'mydict',
        only if key 'day' does not already exist in the dictionary.
        The resulting dictionary is returned.'''
   ```

   **Hint:** You do not have to concern how a value of the function parameter "day" looks like. (It could be any imutable value, like a string 'Monday' or 'Mon', or an integer 1, to name only a few examples. The important point is that for a given key that does not already exist in the dictionary, your function must store the given temperature value.

   Please submit only your function plus the `import` declarations that your function implementation requires. Do not submit any other code (main program, test code, . . . ).

2. Exercise P2 from page 380 of the textbook.
   You can assume that the keys in the dictionary are stored as strings `'Sun'`, `'Mon'`, `'Tue'`, `'Wed'`, `'Thu'`, `'Fri'`, `'Sat'`.

   Please use the following stub for your function:

   ```
   def moderateDays(mydict):
     '''Returns a list [...] of the days for which the average
        temperature was between 70 and 79 degrees.'''
   ```

   For example, if Sunday and Monday were the only days in the asked temperature range, the returned list should be `['Sun', 'Mon']`. If no day matches the temperature range, an empty list should be returned.

   The temperature range is *inclusive*, which means that a day with temperature 70 or 79 should be included in the returned list.

   There is no particular order required for the strings in your result list. No temperature, just the days must be returned.

   Please submit only your function plus the `import` declarations that your function implementation requires. Do not submit any other code (main program, test code, ...).

3. Problem D3 from page 334 of the textbook.
   Your program should ask the user for a filename. Depending on the filename, the program should perform the corresponding action (encryption or decryption). For example:

   ```
   Enter␣a␣filename:␣foo.txt¤
   ```

   The program will generate a random key and store it in file foo.key. Then file foo.txt will be encrypted using the generated key, and stored in file foo.enc.

   Decrypting works as follows:

   ```
   Enter␣a␣filename:␣foo.enc¤
   ```

   The program will open file foo.key to load the key, decrypt file foo.enc, and store the result in file foo.txt. If a filename provided by the user does not exist, you can terminate the program with any given error message or exception.

   With this problem, it is not allowed to use the identity function $f(x) = x$ for encryption and decryption of a character $x$.

   We use the CSV file format to store a random key. For the example in the textbook, the file would have the following contents:

   ```
   A,F¤
   B,G¤
   C,L¤
   ...
   ```

   You can assume that the input-file is restricted to lower-case letters, upper-case letters, digits (0-9) and spaces (' ').

4. Problem D2 from page 381 of the textbook.
   Example:

   ```
   Enter␣a␣color:␣fF6347¤
   ```

```
Red:␣255¤
Green:␣99¤
Blue:␣71¤
```

Please note that the user may use lower or upper-case characters with hexadecimal numbers. You can assume that the user will input a valid color-code.

Please note also that in the figure on page 382 of the textbook the labels for the blue and green color values are swapped. The correct order is red, followed by green, followed by blue (RGB). For this example (color "tomato"), the hexadecimal values `FF6347` correspond to the decimal values 255 (red), 99 (green) and 71 (blue).

5. Problem M8 from page 381 of the textbook.

   Your program must ask for a seed value for the random number generator. You can assume that the user enters a valid integer as the seed value. You must convert the user's input to an integer and seed the random number generator with this integer **before** the program computes the first random number. You should seed the random number generator only once (do **not** seed the random number generator between simulation runs).

   Your program must ask for the number of simulation runs, too. You can assume that the user enters a valid integer greater than zero for the number of simulation runs.

   For example:

```
Welcome␣to␣the␣Food␣Co-op␣Schedule␣Simulation␣Program¤
Enter␣a␣seed␣value:␣29¤
Enter␣the␣number␣of␣runs:␣5¤
(1)scheduled,␣(2)unscheduled␣simulation?␣2¤
<<␣UNSCHEDULED␣WORKER␣SIMULATION␣>>¤
Average␣nr␣workers␣per␣timeslot:␣...¤
Average␣nr␣workers␣who␣worked␣the␣complete␣timeslot:␣...¤
Average␣nr␣workers␣15min␣late:␣...¤
Average␣nr␣workers␣30min␣late:␣...¤
Average␣nr␣workers␣45min␣late:␣...¤
Average␣nr␣workers␣who␣left␣15min␣early:␣...¤
Average␣nr␣workers␣who␣left␣30min␣early:␣...¤
Average␣nr␣workers␣turned␣away:␣...¤
```

   **Note 1:** The last item (average number of workers turned away) must only be output with the *unscheduled* approach. With the *scheduled* approach, the last line must be omitted in your program's output (no extra empty line instead).

   **Note 2:** For the average number of workers per timeslot, you should divide the sum of all workers that "showed up" across all simulation runs by the total number of timeslots across all simulation runs. (The total number of timeslots across all simulation runs is 35*n, where n is the number of simulation runs.)

   For the remaining figures, you should divide the sum of all workers that fulfilled a given criterion (e.g., 15 min late) by the number of simulation runs.

   **Note 3:** In the above example, your program must output the computed values in place of the dots (. . . ), with two digits precision after the decimal point.

**Note 4:** You do not have to type in the source-code from the textbook. You can find the source-code from the textbook in file FoodCoopSimulation.py in the “Labs” folder on YSCEC.

## Marking Criteria and Plagiarism

### Marking Criteria

- Score is only given to programs that compile and produce the correct output with Python version 3.5.1.
- Points are deducted for programs that are named wrongly. See the list of deliverables for the required file names.
- Points are deducted for programs that produce warnings.
- Points deductions on programming style: please provide comments in your code. Use doc-strings to comment all functions.
- Please pay particular attention to the requested output format of your programs. Deviating from the requested output format results in points deductions.

### Plagiarism (Cheating)

- This is an individual assignment. All submissions are checked for plagiarism.
- Once detected, measures will be taken for all students involved in the plagiarism incident (including the “source” of the plagiarized code).

## Deliverables for Lab 11

| Problem | File name |
|---|---|
| 1 | lab11_p1.py |
| 2 | lab11_p2.py |
| 3 | lab11_p3.py |
| 4 | lab11_p4.py |
| 5 | lab11_p5.py |

- Please submit all files in a ZIP-file named lab11_<student_id>.zip
- Please consult the specification of Lab 2 on how to create ZIP archives.
- Please submit your archive on YSCEC by Thursday, June 2nd, 23:00.