Homework 1

MPCS 51042 – Python Programming

Due: October 13th 2020, 11:59 pm CT

Preliminaries

Every homework assignment may contain a preliminaries section. It's important that you read over that section and any of its subsections before beginning the assignment.

Repository Access

We will use the CS Department's Gitlab server to submit and checkout assignments. Follow the below steps to checkout your repository:

1. Your repository has already been setup on GitLab(`https://mit.cs.uchicago.edu`). You can follow the steps described there to checkout your repository using SSH. However, I will show you how to checkout your repository using https. Checkout your repository using this command in the Terminal program:

   ```
   git clone https://mit.cs.uchicago.edu/mpcs51042-aut-20/yourusername.git
   ```

   replacing `yourusername` by your CNET username in lowercase. You will be prompted for your CNET password. If you get a prompt asking you if you would like to accept the certificate, press **p (meaning to accept permanently)**. (For example, since my username is lamonts, I would type

   ```
   git clone https://mit.cs.uchicago.edu/mpcs51042-aut-20/lamonts.git
   ```

   You will notice nothing is inside your repository yet. In the next section, I will show you how to get your starter code for this homework (and possibly future assignments).

2. As a git refresher, when you add or modify files to your repository you need to ALWAYS use `git add` followed by the file or directory name. For example,

   ```
   git add problem1/day.py
   ```

   After adding the file to your repository, you need to commit the file/directory to your local repository using

   ```
   git commit -m "A commit message"
   ```

3. Once you have completed making modification locally, you will need to commit your files/directories to the central repository using

   ```
   git push origin master
   ```

   The specific steps for committing your code is described in the "Submission" section.

Pytest

Please go to the following url and install **pytest**:

`https://docs.pytest.org/en/latest/getting-started.html`

This will allow you to run the test cases for this assignment.

Course Workspace

First, you **must** watch the following video on our Panopto page (same location as the "Course Overview" video):

Course Repository Setup

Make sure you have done the following steps (only do these steps if haven't do so via watching the video):

1. Added the upstream repository to your directory. In the Terminal program, make sure you are inside your course repository. Enter the following command:

```
$: git remote add upstream https://mit.cs.uchicago.edu/mpcs51042-aut-20/mpcs51042-1-aut-20.git
```

2. Pull in the starter code for **hw1**. **You must always perform the following line each time you want to get starter code coming from me.** Enter the following command:

```
$: git pull upstream master
```

You should see all the starter code for homework 1. You are now ready to get started.

Style Guide

For this homework and all future homework assignments, we will follow the style guide used by the undergraduate Python course. It's located here: `https://classes.cs.uchicago.edu/archive/2020/fall/12100-1/style-guide/index.html`

Problem 1

Inside your *hw1* directory, Open the **problem1/find_twos.py** file. Implement the following function:

```
def find_twos(str1, str2):
    '''
    This function takes in two strings that only contains integers, commas and whitespace and
    returns a list of integers, where each integer,

       1. Appears in both strings
       2. Contains a 2 as a digit in the number.

    Inputs:
        str1, str2 (string): strings that contains integers, commas, and whitespace. You can assume
        each integer is separated by a single comma followed by zero or more whitespaces.

    Output:
        A list of integers, where the list contents is described by above. The returned list
        must not contain duplicates.
    '''
```

You can run the test cases by running the following command in the **hw1/problem1/** directory:

```
$ pytest
```

You can test this without using pytest by running `ipython` inside the **hw1/problem1/** directory and performing the following commands:

```
In [1]: import find_twos

In [2]: find_twos.find_twos("2,23,1","3,2")
Out[2]: [2]
```

Problem 2

Inside your *hw1* directory, Open the **problem2/day.py** file. Implement all functions defined within this file. Each function has a docstring that describes what it should do and return.

The function that will be tested is the following `find_day` function:

```
def find_day(dates):
    '''
    This function takes in a list of dates as strings. Each valid date will be in the following
    format:

          'month/day/year'

    This function only processes 'valid' date formats. Please check the is_valid_date function
    for the correct format (i.e.,  You should use this function within find_day).

    The find_day determines the day of the week (i.e. Sunday, Monday, Tuesday, Wednesday, ...)
    that each date falls on.

    Inputs:
        dates(list): A list of dates as strings

    Output:
        It converts each date inside the dates input into its day of the week and returns
        them as a list of strings. The day of the week must be in the same position that
        its date was in from the dates list. If a date is not a valid date then this
        function returns the string: 'error'.
    '''
```

To calculate the day of the week use the following formula:

```
day_of_week = day + compute_month_val(month, year) +
              compute_year_val(year) + compute_century_val(year)
```

where `day` is the day component of the string. Once you compute the `day_of_week` value then you will need to mod it (i.e., use %) by 7. This will provide you with a number between 0 and 7. Use the following to assign the correct day of the week for a date:

```
0 -> "Sunday"
1 -> "Monday"
2 -> "Tuesday"
3 -> "Wednesday"
4 -> "Thursday"
5 -> "Friday"
6 -> "Saturday"
```

You can test this without using pytest by opening `ipython` and performing the following commands:

```
In [1]: import day

In [2]: day.find_day(["1/8/2020"])
Out[2]: ['Wednesday']

In [3]: day.find_day(["1/8/2020","1/9/2020"])
Out[3]: ['Wednesday', 'Thursday']

In [4]: day.find_day(["1/8/2020","1/9/2020","4/4/4","2/29/2019"])
Out[4]: ['Wednesday', 'Thursday', 'error', 'error']
```

You can run the test cases by running the following command in the **hw1/problem2/** directory:

```
$ pytest
```

Place your solution inside the **day.py** file.

Problem 3

ComEd tracks the status of its power service throughout the city of Chicago in a grid representation. From a programming perspective, the representation is in the form of a lists of lists. You should think of this lists of lists as a two-dimensional matrix. Each cell within the grid is a home where its value is a boolean value. The `True` means that the cell (i.e., a home) has power and `False` means a cell has no power. The diagram below represents a visual of a 3x3 grid with indices:

	0	1	2
0	T	T	T
1	T	F	F
2	F	T	T

From a programming perspective, the above matrix will be represented as a list of lists:

```
grid = [ [True,  True,  True],
         [True,  False, False],
         [False, True,  True]
       ]
```

We say the home at `grid[0][1]` has a value of `True`, which means that it has power. Whereas, the home at `grid[1][1])` has a value of `False`, which means it has no power.

Your job is to implement the following function:

```
def power_status(grid, row_bounds, col_bounds):
    '''
    Determines the status of the homes within a sector. A sector is defined by a row  and column
    boundary. A row boundary is a tuple of two integers from ([0, nRows), [0, nRows)) and the column
    boundary is a tuple of two integers ([0, nColumns), [0, nColumns)), where nRows is equal to the
    number of rows in the grid and nColumns is equal to the number of columns in the grid. Make note
    the upper-bound is exclusive!

    Based on the boundary specified, the function returns a list indicating the power status of the
    homes within the bounds of the sector. Order in the list does matter! Start at the top-left of the
    sector and work your way down towards the bottom-right corner. You increase by in the columns. Once
    you finish with a row then you move to the next row (i.e., increase by 1).

    You can assume the the first component of the boundary is always less than the second component.

    Inputs:
        grid(list of list of bools): the status of the homes
        row_bounds(tuple of ints): the row boundary for the sector
        col_bounds(tuple of ints): the column boundary for the sector

    Outputs:
        Returns a list of booleans indicating the power status of the homes within the bounds of
        the sector.

    '''
```

You can test this without using pytest by opening `ipython` and performing the following commands:

```
In [1]: import power

In [2]: grid = [ [True,  True,  True],
   ...:          [True,  False, False],
   ...:          [False, True,  True]
   ...:        ]

In [3]: power.power_status(grid,(0,1),(0,2))
Out[3]: [True, True]
```

You can run the test cases by running the following command in the **hw1/problem3/** directory:

```
$ pytest
```

Place your solution inside the **power.py** file.