CS5001 HW2: Due on Canvas Sun Oct 15 at 11:59pm

You must work **either on your own or with one partner**. If you work with a partner, you and your partner must first register as a group with us then submit your work as a group on Canvas. To register, please send an email to the TAs at the following:

"Qing Chen" chen.qing1@northeastern.edu ; "Lingyu Hu" hu.lingyu@northeastern.edu ; "Mingtianfang Li" li.mingt@northeastern.edu ; "Kejian Tong" tong.ke@northeastern.edu

NOTE: If working in pairs, you can pair up with the same person for a maximum of two assignments/projects in this course. After that, you must either find another partner or complete the assignments individually. Any team in violation of this would receive a zero for their submissions after the first two submissions.

You may discuss background issues and general solution strategies with others, but the programs you submit must be the work of just you (and your partner). We assume that you are thoroughly familiar with the discussion of academic integrity that is on the course website. Any doubts that you have about "crossing the line" should be discussed with Tas or the instructor before the deadline.

Assignment Objectives. More practice with string slicing and Boolean computation. Implementing functions and procedures. The idea of one function calling another. Writing Application Scripts. The assignment is primarily based on Lectures 1 to 4. You should download and play with the associated demo files.

1 Siri Sez (50 points)

Imagine driving in a city where at every intersection you have exactly four options:

• Continue Straight • Turn Left • Turn Right • Make a U-Turn

Assume that the streets are laid out so that they are aligned with the four compass points. This means that you are either driving North (N), West (W), South (S), or East (E). We can encode a 6-intersection journey with a length-6 string that is made up of the characters N, W, S, and E. Here is an example:

'NWNEEW'

And here is a table that explains what happened at each intersection:

In this problem you are to write Python code that takes a length-6 string like 'NWNEEW' and prints out the six actions associated with the journey. Sample output:

```
NWNEEW
Start Driving North
Turn Left
Turn Right
```

Turn Right
Continue Straight
Make a U-Turn

We make some definitions in order to precisely describe the Python code that you are to write:

- A length-6 string made up of the characters N, W, S, and E is called a route string. If R is a route string then R[k] indicates the direction of travel just after leaving the kth intersection. Thus, given the route string R = 'NWNEEW' we know that the driver leaves the second intersection heading north because the value of R[2] is 'N'.
- A length-2 string made up of the characters N, W, S, and E is called an intersection string. If I is an intersection string then I[0] encodes the direction of travel entering the intersection and I[1] encodes the direction of travel leaving the intersection. Thus, if I = 'ES' then the driver is driving east heading into an intersection and then turns south.
- The strings 'Turn Right', 'Turn Left', 'Continue Straight', and 'Make a U-Turn' are called intersection instruction strings.
- The strings 'Start Driving North', 'Start Driving West', 'Start Driving South', and 'Start Driving East' are called starter strings. They will be used to indicate what happens at the 0-th intersection, i.e., what happens at the start of the trip.

We are now ready to describe the functions and Application Script that you are to submit.

1.1 Getting Set Up

Using the editor, create a module Siri.py.

```
# Siri.py
# YOUR NAME(S) AND ID(S) HERE
# DATE


""" Converts a route string into a sequence of driving instructions. """

def SiriSez(x):
    pass


def TripAdvisor(Route):
    pass


# Application Script
if __name__ == '__main__':
    I = input('Enter an Intersection String: ')
    Print(I)
```

Eventually this module will house two fully-implemented functions: SiriSez and TripAdvisor. Right now, the pass statements act as placeholders. If you run Siri.py it does something very simple. It prompts you to enter an intersection string like 'NW' and then it prints it out. Try it!

1.2 Implementing SiriSez

Your next task is to implement SiriSez(I). This function is to take an intersection string and return the appropriate intersection instruction string. Here are some examples:

Value of I	What SiriSez Returns
'EN'	'Turn Left'
'ES'	'Turn Right'
'EE'	'Continue Straight'
'EW'	'Make a U-Turn'

Start your implementation of SiriSez by removing the pass statement and writing the complete specification. (Function specifications are discussed in Lecture 4.)

Developing the body of SiriSez requires a considerable amount of "Boolean thinking." There are lots of cases to consider but the amount of required code is modest if you take full advantage of the Boolean-related discussion in Lecture 3. As usual, there are several ways that the problem can be solved and it is fine if your approach uses a "helper function". Just make sure its implementation is part of your submitted Siri.py.

To debug your implementation of SiriSez you should augment the application script with

```
M = SiriSez(I)
print (M)
```

Thus, by running Siri.py you have a handy way of testing SiriSez on arbitrary input strings. Do not proceed until you are confident that your implementation of SiriSez is correct.

1.3 Implementing TripAdvisor

Next you are to implement TripAdvisor. This function takes a valid route string and prints seven lines of output:

Line 1 The route string.
Line 2. The associated starter string.
Lines 3-7. The associated intersection instruction strings.

Here is what TripAdvisor('NWNEEW') would print:

```
NWNEEW
Start Driving North
Turn Left
Turn Right
Turn Right
Continue Straight
```

Make a U-Turn

Rubrics: For the above output, each correct print would be graded as follows. (The correctness of your program will be tested against many test cases. The program should pass all the test cases to get full credit. A program that fails several test cases will get a very low grade.)

NWNEEW **(5 points)**
Start Driving North **(5 points)**
Turn Left **(8 points)**
Turn Right **(8 points)**
Turn Right **(8 points)**
Continue Straight **(8 points)**
Make a U-Turn **(8 points)**

Note that TripAdvisor does not return a value. It is a void function. A void function is sometimes referred to as a procedure.

Start your implementation of TripAdvisor by removing the pass statement and writing its specification. In writing the body of TripAdvisor, you are required to make effective use of SiriSez. It is perfectly legal for one function (that you write) to call another function (that you write). Indeed, the point of having you develop TripAdvisor is to give you practice calling SiriSez.

Modify the Application Script so that it solicits a valid route string and then calls TripAdvisor.

Submit Siri.py to Canvas. It should include your finished implementations of SiriSez, TripAdvisor, and the Application Script just described. Finally, your implementations of SiriSez and TripAdvisor do not have to compensate for "bad input." Just worry about the correctness of your functions given that the preconditions are satisfied.

2. Slash-Date to Dash-Date (50 points)

There are several ways that you can encode a date as a string:

'7/4/2016' '07-04-2016' '7JUL16' 'July 4, 2016'

In this problem you are to write a script **Slash2Dash.py** that uses keyboard input to solicit a date string in slash format and then displays the equivalent date in dash format. Here are some defining examples:

Example	Slash Format	Dash Format
1	'7/4/2016'	'07-04-2016'
2	'12/23/15'	'12-23-2015'
3	'12/23/16'	'12-23-2016'
4	'12/23/25'	'12-23-1925'

5	'12/23/28'	'12-23-1928'
6	'12/23/2017'	'12-23-2017'
7	'13/42/0001'	'13-42-0001'
8	'2/29/2015'	'02-29-2015'

We need rules that define what it takes to be a slash-date string and rules that tell us how to produce the equivalent dash-date string. Here we go. A string is a slash-date string if

S1. Each of its characters is either a digit or a slash '/'.

S2. It contains exactly two slashes.

S3. There are either one or two digits before the first slash. These digits define the month slice.

S4. There are either one or two digits in between the two slashes. These digits define the day slice.

S5. There are either two or four digits after the second slash. These digits define the year slice.

Notice that a string can be a dash-date string without encoding an actual date. See Examples 7 and 8.

The definition of a dash-date string is similar:

D1. Each of its characters is either a digit or a dash '-'.

D2. It contains exactly two dashes.

D3. There are two digits before the first dash. These digits define the month slice.

D4. There are two digits in between the two dashes. These digits define the day slice.

D5. There are four digits after the second dash. These digits define the year slice.

Next, we need rules that tell us how to convert a slash-date string to an equivalent dash-date string:

Month Conversion. The month slice in the dash-date string is the same as the month slice in the slash-date string unless the latter consists of a single digit. In that case the former is obtained by concatenating '0' to the front of the latter. Thus, in Example 1 '7' becomes '07'.

Day Conversion. The day slice in the dash-date string is the same as the day slice in the slash-date string unless the latter consists of a single digit. In that case the former is obtained by concatenating '0' to the front of the latter. See Example 1 where the '4' becomes '04'.

Year Conversion. The year slice in the dash-date string is the same as the year slice in the slash-date unless the latter consists of two digits. In that case the former is obtained by concatenating either '19' or '20' to the front of the latter. If the numerical value of the year slice in the slice-date slice-dash string is between and including 0 and 23, then concatenate '20' on to the front. (See Examples 1,2, and 3.) Otherwise, concatenate '19' as in Examples 4 and 5.

Use input() to acquire the input slash-date string. (You may want to play with the ShowInput.py demo associated with Lecture 2.) Your program can assume that the input string satisfies S1-S5. Later on in the course we will discuss graceful ways to handle "bad input."

Your implementation of Slash2Dash.py will require string slicing, string concatenation, and some Boolean expressions. Your code will have to handle a couple of different situations. Do not try to figure out all these situations at once. Instead, you should develop your solution in stages; something like this:

Stage 1. Your initial version of Slash2Dash should use input() to store the incoming slash-date string in a variable (say s) and then compute the location of the two slashes. That is, write code that assigns the index of the first slash to a variable and the index of the second slash to another variable. Because of **S3** you know that either s[1] or s[2] houses the first slash. Figure out which. Reason similarly for the second slash. You might find it handy to involve the len function.

Use print statements to check your location variables for correctness. In particular, print out their values to affirm that the location of the first slash and the second slash are properly computed. Note that there are 8 = 2 × 2 × 2 cases to check because the month, day, and year slices each have two possible lengths, e.g., '7/4/67', '7/14/67', '12/1/67', '12/14/67', '7/4/1967', '7/14/1967', '12/1/1967', '12/14/1967'.

Stage 2. (Don't do this until you have completed Stage 1.) Extract the month slice from s and compute the month slice for the dash-date string. Look at the Month Conversion rule above. Use print statements to check your computed slice for correctness.

Use print statements to check your location variables for correctness. That is, don't write the rest of the program yet. Instead, just get it to where it is getting the month part write. To test your program at this stage, try the input '7' and see if the output of your program is '07'. (What other kind of input should you try to make sure the month part of your program is working?)

Stage 3. (Don't do this until you have completed Stage 2.) Extract the day slice from s and compute the day slice for the dash-date string. Look at the Day Conversion rule above. Use print statements to check your computed slice for correctness.

Stage 4. (Don't do this until you have completed Stage 3.) Extract the year slice from s and compute the year slice for the dash-date string. Look at the Year Conversion rule above. Use print statements to check your computed slice for correctness.

Stage 5. (Don't do this until you have completed Stage 4.) Using concatenation, assemble the dash-date string from what has been computed in Stages 2, 3, and 4. Print the final result.

Some sample dialogs:

Enter a valid slash-date: 7/4/76 07-04-1976

Enter a valid slash-date: 7/4/2020 07-04-2020

Enter a valid slash-date: 20/99/1976 20-99-1976

Thus, once you compute the dash-date string you should just print it “as is”. Be careful about blanks, since they’re hard to see! For instance, in this example,

Enter a valid slash-date: 7/4/76

the user has actually entered '__7/4/76', and your program should not work on that input. Submit Slash2Dash.py to Canvas. It is important that you remove all print statements that were part of your debugging strategy.

Rubrics:

10 points each for correctly printing the month, day, year, dash-1, dash-2. The correctness of your program will be tested against many test cases. The program should pass all the test cases to get full credit. A program that fails several test cases will get a very low grade.

What to submit?

1. Prepare a README.txt file containing the following:

a) Include a quick summary of how you run your program Siri.py.
b) Include a quick summary of how you run your program Slash2Dash.py.
c) If any of the programs is not working, include what is the issue in your opinion and how would you fix it if you had more time?

2. Submit your files Siri.py, Slash2Dash.py, README.txt inside a single zip file on Canvas.

Additional directions

Following directions apply to both the problems above. Negative numbers indicate the penalty to be applied on your final score for a problem/direction if the directions are not followed.

1. Start your programs with a docstring (comment) summarizing what it is doing, what are the inputs, and the expected output. (-5 points)
2. At the beginning, as three comments, include your name (both teammates if applicable), date, and your email ids. (-5 points)
3. At the beginning of every function, include a docstring comment describing what the function does, returns, and the preconditions, as discussed in class. (-5 points)
4. For every variable defined in your program, include a brief comment alongside explaining what it stores. (-5 points)
5. README.txt file needs to be submitted as described above. (-10 points)
6. Submit all the files (.py and .txt files) as a single zip folder/file on Canvas. (-5 points)
7. No new submissions, code files can be accepted after the deadline. Please double check and ensure that you are submitting everything needed to run your programs, and the code files are the best versions you want evaluated. Only the material submitted in the zip file uploaded on Canvas before the deadline shall be graded.