Homework Assignment 1

The assignment is due by 11:55PM of the due date. The point value is indicated in square braces next to each problem. Each solution must be the student’s own work. You may seek or accept assistance only from the course instructor or the TA. Any violation of this rule will be dealt with harshly.

This assignment is primarily a review of string processing, functions, and writing modular programs with minimal code repetition. In this and all future assignments, you are graded not only on the correctness of the code, but also on clarity and readability. Hence, I will deduct points for poor indentation, poor choice of object names, and lack of documentation. For documentation, use a common sense approach. While I do not expect every line of code to be explained, all code blocks that carry out a significant task should be documented briefly in clear English.

Please read the submission guidelines at the end of this document before you start your work.

Important note: When writing each of the following programs, it is important that you name the functions exactly as described because I will assume you are doing so when testing your programs. If your program produces errors because the functions do not satisfy the stated prototype, points will be deducted. In particular,

1. I will type `craps()` in the Python shell to test `problem1.py`.
2. I will type `salary_calc()` to test `problem2.py`.

For each problem, put all your function definitions in one file. Do not create a separate file for each function. The files `problem1.py` and `problem2.py` should contain all the function definitions for Problems #1 and #2, respectively.

Problem 1 [18 points] Craps, a game of chance. A popular game of chance frequently seen at casinos is a dice game known as “craps”. The game is played as follows. A player rolls two dice. Each die is the familiar six-sided die with faces containing 1, 2, 3, 4, 5, and 6 dots. After the dice are rolled, the next step is determined by the sum of the dots on the two top faces.

- If the sum is 7 or 11 on the first throw, the player wins.
- If the sum is 2, 3, or 12 on the first throw, the player loses (called “craps”).
- If the sum is 4, 5, 6, 8, 9, or 10 on the first throw, then that sum becomes the player’s “point”. Now, the only way for the player to win is by continuing to roll the dice until she “makes her point”. If the player rolls a 7 before making her point, she loses. In other words, if the roll of the dice adds up to her point, she wins; if it adds up to 7, she loses; if it adds up to neither, she rolls the dice again.
Note that there are two ways for a player to win: Either by rolling a 7 or 11 in the first roll of the dice, or by rolling her point in a subsequent roll of the dice.

You are asked to write a program to play craps to allow wagering. The player starts with an initial bank balance of $1000. Each game starts with a wager (which of course must be no bigger than the current bank balance). After one game, the bank balance is updated and the player is allowed to play again, repeatedly, until she quits, or the bank balance falls to $0. You will accomplish all this by implementing the following functions.

1. (3 points) Write a function called roll_dice with no parameters. The function rolls two dice and return the sum of the result. You can simulate the roll of a die by using the randint(a, b) function from the random module. randint(a, b) returns a random integer N such that a \leq N \leq b.

2. (9 points) The second function called play_one_game plays a single game of craps, using the rules described above. This function does not have parameters. It should print out sentences informing the player about the sequence of actions taking place in the game (see sample runs below - your code should mimic the output shown). It returns an integer value of 1 if the player wins the game, and in integer value of 0 if the player loses the game.

3. (6 points) The third function called craps does not have any parameters. It uses a while loop to allow the player to play as many rounds of craps as she wants, as long as the bank balance is not $0. The function should first prompt the user to enter a wager. If the wager is greater than the bank balance, repeatedly prompt the player to re-enter the wager until a valid wager is entered. Then run one game of craps (using the play_one_game function), inform the player if she won or lost and update the bank balance accordingly. If the new balance is $0, print a message ("Sorry, you’re broke!") and quit. As long as the balance is greater than $0, the player may repeatedly choose to play again. If the player quits with a bank balance greater than $0, print a congratulatory message if money was made and a sympathetic message if money was lost.

A sample run is shown below.

----------------------------
Welcome to the Craps program
----------------------------

Your initial bank balance is $1000.

What is your wager? 100
Okay, let’s play.

You rolled 11
You win!!

Your new bank balance is $1100

Do you want to play again? [y/n] y

What is your wager? 500
Okay, let’s play.
You rolled 12
Sorry, you lose!

Your new bank balance is $600

Do you want to play again? [y/n] y

What is your wager? 700
Cannot wager more than $600. Re-enter wager: 800
Cannot wager more than $600. Re-enter wager: 400
Okay, let's play.

You rolled 4
Your point is 4

You rolled 5
You rolled 9
You rolled 10
You rolled 9
You rolled 3
You rolled 7
Sorry, you lose!

Your new bank balance is $200

Do you want to play again? [y/n] y

What is your wager? 500
Cannot wager more than $200. Re-enter wager: 200
Okay, let's play.

You rolled 10
Your point is 10

You rolled 6
You rolled 11
You rolled 6
You rolled 10
You win!!

Your new bank balance is $400

Do you want to play again? [y/n] n

Sorry you lost money. Better luck next time!

Problem 2 [32 points] | Salary calculator. In this problem, you are asked to write a Python program to calculate the daily salary for employees of a small software company. An employee of the company may be part of the managerial staff, the technical staff, or the janitorial staff. The hours of employment and rate of pay depend on the employee’s category, as summarized below:
The managerial staff are required to work 10 hours per day at the rate of $30 per hour.
The technical staff are required to work 8 hours per day at the rate of $25 per hour.
The janitorial staff are required to work 7 hours per day at the rate of $10 per hour.

In addition, the following rules apply for the hours of employment:

- The office building is closed between the hours of midnight and 6:00AM. Hence, an employee’s work day must start on or after 6:00AM and must end strictly before midnight.
- If an employee works fewer hours than the number of hours required for her category, she will get paid at an hourly rate that is 0.8 times the regular rate. For example, if a member of the technical staff works only 6 hours on a given day, she will get paid $20 per hour (not $25). Hence, her salary for that day is $120.
- If an employee works overtime (that is more than the number of hours required for her category), she will get paid at an hourly rate that is 1.25 times the regular rate for the extra hours. For example, if a member of the janitorial staff works 9 hours and 20 minutes on a given day, she will get paid $70 for the 7 hours she is required to work. For the two hours and 20 minutes of overtime, she will get paid $12.50 per hour, which amounts to $29.17. Hence, her salary for that day is $99.17.

You are asked to write a program that first asks the user for their category of employment (‘M’ for managerial, ‘T’ for technical, and ‘J’ for janitorial), the time at which her work day started, and the time at which her work day ended. The start and end times should be entered in military format. Recall that time in military format is based on a 24 hour clock. Some examples: 8:45AM is written as 0845, midnight is written as 0000, 12:45AM is written as 0045, noon is written as 1200, and 8:45PM is written as 2045.

Your program should check for the validity of input. In particular, if the user enters a character other than ‘M’, ‘T’, or ‘J’ for the employee type, your program should print an appropriate error message and quit. Also, if the user enters an invalid military time for either the start or end time (keep in mind that both the hour and minute values should be valid), then too the program should print an appropriate error message and quit. If the start and/or end time occurs in the period midnight to 5:59AM, print an appropriate message and quit. Finally, if the start time occurs after the end time, your program should print an appropriate message and quit.

Once the program has read all input and determined its validity, it should print out a daily salary report with the following information: (1) Type of the employee, (2) the start time in AM/PM format, (3) the end time in AM/PM format, (4) the total hours worked, (5) the rate at which the employee is paid, including overtime (if any), and (6) the total salary for that day. (See sample runs for some examples.)

You are required to implement the following functions to carry out the above tasks.

1. (4 points) Implement a function called valid_mil_time with a single parameter called mtime, which is a string. This function returns the boolean value True if mtime is a valid military time and False otherwise. To be valid, mtime should be exactly 4 characters long. Furthermore, the hour value and the minute value should be valid as well.

2. (4 points) Implement a function called minutes_elapsed with two parameters, mtstart and mtend, both of which are string representing military times. You may assume that
both are valid military times and also that \texttt{mtend} occurs on or after \texttt{mtstart}. This function returns the number of minutes that have elapsed between \texttt{mtstart} and \texttt{mtend}. For example, \texttt{minutes \_elapsed("0830", "1745")} returns 555, and \texttt{minutes \_elapsed("1135", "1710")} returns 335.

3. (6 points) Implement a function called \texttt{ampm} with a single parameter called \texttt{mtime}, which is a string representing a military time. This function returns a string representing the AM/PM time corresponding to \texttt{mtime}. You may assume that \texttt{mtime} is a valid military time. For example, \texttt{ampm("0845")} should return the string "8:45AM", \texttt{ampm("0000")} should return the string "12:00AM", and \texttt{ampm("2359")} should return the string "11:59PM".

4. (10 points) Implement a function called \texttt{daily \_salary \_report} with three parameters: \texttt{emp \_type} (a single character representing the type of employee), \texttt{mtstart} (the start time) and \texttt{mtend} (the end time). This function should print a daily salary report as described above (also see sample runs). \textit{Note:} Call the functions \texttt{ampm} and \texttt{minutes \_elapsed} when implementing this function. \textit{Do not repeat code unnecessarily.}

5. (8 points) Implement a function called \texttt{salary \_calc} without any parameters. (This is the function I will call when testing your program in the Python shell.) This function should use the \texttt{input} function to read in three values from the user (see sample runs): the type of employee, the start time in military format, and the end time in military format. \textit{This function must check for validity of input.} This means that the employee type should be valid (’M’, ’T’, or ’J’), the start and end times should be valid (use \texttt{valid \_mil \_time} to check validity of time), the start time should occur on or after 6:00AM, the end time should occur before midnight, and the end time should occur after start time. If any of these validity requirements are violated, this function prints an appropriate message and quits. Otherwise, once you have ensured that all the input is valid, call \texttt{daily \_salary \_report} to print out the daily salary report.

Some sample runs of the program are given below.

```python
>>> salary_calc()
-----------------------
DAILY SALARY CALCULATOR
-----------------------
Enter the type of employee: M
Enter the start time: 0830
Enter the end time: 1745

Daily salary report
---------------------

Type of employee: Managerial
Start time: 8:30AM
End time: 5:45PM

Total hours worked: 9 hours and 15 minutes
Pay rate(s): $24.00/hr for 9 hours and 15 minutes
Total Salary: $222.00
```
>>> salary_calc()
 ------------------------
DAILY SALARY CALCULATOR
 ------------------------
Enter the type of employee: T
 Enter the start time: 1135
 Enter the end time: 2000

Daily salary report
 -------------------
Type of employee: Technical
 Start time: 11:35AM
   End time: 8:00PM

Total hours worked: 8 hours and 25 minutes

Pay rate(s): $25.00/hr for 8 hours
 $31.25/hr for 25 minutes

Total Salary: $213.02

>>> salary_calc()
 ------------------------
DAILY SALARY CALCULATOR
 ------------------------
Enter the type of employee: J
 Enter the start time: 0545
 Enter the end time: 1245

Invalid input: Start and/or end time occurs before 6:00AM!

Submission Guidelines

Implement the first problem in a file called problem1.py and the second one in a file called problem2.py. Your name and RUID should appear as a comment at the very top of each file.

Test each of your programs thoroughly before submitting your homework. When you are ready to submit, upload your files on Sakai as follows:

1. Use your web browser to go to the website https:sakai.rutgers.edu.
2. Log in by using your Rutgers login id and password, and click on the OBJECT-ORIENTED PROG S18 tab.
3. Click on the 'Assignments' link on the left and go to 'Homework Assignment #1' to find the homework file (hw1.pdf).
4. Use this same link to upload your homework files (problem1.py and problem2.py) when you are ready to submit.

You must submit your assignment at or before 11:55PM on February 5, 2018.