ENGR 102 – Fall 2021
Lab Assignment #4a

Deliverables for Lab Assignment #4:

Lab Assignment #4 consists of three team activities that produce deliverables. Please submit the following files Mimir. You will submit a total of 3 .py files and one .pdf file.

- `Lab4a_Act1.pdf`
- `Lab4a_Act1.py`
- `Lab4a_Act2.py`
- `Lab4a_Act3.py`

Please include the team header at the top of **each** .py file with the names of all contributing team members. Include the team header *information* in the pdf file, however you do not need to include the comment symbols (#). Please submit all files on the list as **one** submission. Please do not zip the files. (Tip: Place all the files in the same folder, then when submitting, either drag and drop them simultaneously or using the File Selection dialog, select them all simultaneously by using CTRL (PC)/Command (Mac). This is a team assignment, but everyone in the team must submit the same 4 files.

Activity #1: *To do in lab – as a team*

Write a program that computes parking fees for a garage with the fee structure given in the figure below. Your program should prompt a user to input from the keyboard the number of hours parked in the garage, and output the total fee for parking. A lost ticket is signified using a negative value for hours.

Partial hours are charged as full hours, e.g. 5 hours and 20 minutes is charged as 6 hours. You must take as input the number of hours as a single floating-point number rather than as separate values of "hours" and "minutes" (*example: 5.25 hours*). Thus, the output should always be an integer.

PARKING FEES PARKING RATES	
0-2 Hours	$4.00
2-4 Hours	$3.00
Each Additional Hour	$1.00
Maximum Daily Charge	$24.00
Lost Ticket Charge	$36.00
PAY AT FRONT DESK PROIR TO DEPARTURE	

Examples (**understand the PARKING FEES** ☺):

parked for 2 hours, pay $4

parked for 3 hours, pay $4 + $3 = $7

parked for 5 hours, pay $4 + $3 + $1 = $8

parked for 20 hours, pay $23

parked for 22 hours, pay $24

parked for 123 hours (5 days, 3 hours), pay 5*$24 + $4 + $3 = $127

parked for 123 hours but lost the ticket (user inputted -123), pay $163

This is NOT an easy task. You MUST *think before you code*. Discuss with your team how to approach the problem and sketch out a plan for how to solve it. Try to develop an algorithm (a sequence of steps to perform a computation or solve a problem). This task may take up to 30 minutes. It will take much longer if you do not think before you code.

First, your team should develop and document an algorithm for solving this problem. Manually check your algorithm and check your answers versus the given data above. Document your work in the file Lab4a_Act1.pdf for submission. You should not code until you develop and check your algorithm by hand, on paper.

After you develop an algorithm, translate it into Python code as Lab4a_Act1.py. Verify that your code delivers the correct output for the case data given above. (The examples shown on the previous page.)

Your program should perform the following general tasks:

1. Store input in appropriately named variables
2. Convert the input to a number if you plan to perform mathematical operations with the input value
3. Perform necessary decisions/procedures/calculations and store results in appropriately named variables
4. Output results to the screen using the format below

✓ *Example output (using input **123**):*

```
Enter the hours parked as a decimal number. Include a negative sign if the ticket is lost.
Please enter the hours parked: 123
Parking for 123.0 hours please pay $127
```

Activity #2: *To do in lab – as a team*

A quadratic equation is an equation of the form $Ax^2 + Bx + C = 0$ where A, B, and C are the coefficients of the equation. Write a program named Lab4a_Act2.py that takes as input the coefficients A, B, and C and outputs the quadratic equation in a nice format. Your program should not print the coefficient "1" or a term with a coefficient "0" and should print minus signs (-) for negative coefficients. Include one space around the sign and the term. You may want to review the string formatting covered in Week 3. Use the output format shown below. You may assume the user only enters integers.

✓ *Example output (using inputs **-1**, **-2**, **3**):*

```
Please enter the coefficient A: -1
Please enter the coefficient B: -2
Please enter the coefficient C: 3
The quadratic equation is - x^2 - 2x + 3 = 0
```

Activity #3: *To do in lab – as a team*

The purpose of this program is to give you practice with performing Boolean logic with Python. Create a file named Lab4a_Act3.py for this activity. Each team member is responsible for knowing this material, and it may be a good idea for all team members to practice keying in the code given. You are required to include code for Part A, Part B, and Part C below in your file for submission. Part D is optional and should be included in your file if you attempt it. *Please separate the various parts of your code with the following comment to identify the separate sections (copy/paste into your file with the appropriate letter)*.

```
########## Part A ##########
```

Part A: Inputting Boolean values from the keyboard

Write a simple code to take as input from the user Boolean values from the keyboard for variables `a`, `b`, and `c`. The user should be able to enter a value indicating Boolean True and Boolean False from the keyboard and have the code create a Boolean variable representing the user's input for use in Boolean expressions that appear later in the code. You will need a simple code to make some decisions based on the values of these Boolean variables to verify that your scheme works.

The user should enter `True`, `T`, or `t` for Boolean `True`. To indicate Boolean `False`, the user should enter `False`, `F`, or `f`. Note: Numeric input is **not** permissible.

If this proves to be a bit of a challenge, review Lecture 3 slides covering Boolean conversions. You will need to use your new-found knowledge of `if-elif-else` blocks to make this work as described.

Part B: Evaluating Booleans

Add to your program code that evaluates the following Boolean expressions using the variables `a`, `b`, and `c` from Part A. The program should output the value `True` or `False` of the expression for the entered values. Use Boolean expressions; do **NOT** use `if-elif-else` blocks. See the example output below.

1. a and b and c
2. a or b or c

Part C: Writing Boolean expressions

Extend your program above to include Boolean expressions (**NOT** `if-elif-else` blocks) that meet the criteria in each item below using the variables `a`, `b`, and `c` from Part A. Your program should output the value `True` or `False` based on the previously entered values.

1. Given values for two Boolean variables a and b, create a Boolean expression for "exclusive or" or "XOR" between a and b. This expression evaluates to `True` if just one of a or b is True, but not if both are True or both are False. Do **NOT** use `^`, instead use **only** `not`, `and`, and `or` to construct an equivalent statement.

2. Given values for three Boolean variables a, b, and c, create a Boolean expression to determine if `True` was entered an odd number of times. This expression evaluates to `True` if exactly 1 or 3 of the variables a, b, and c is True, and is `False` otherwise. Use **only** `not`, `and`, and `or`.

✓ *Example output (using inputs **T**, **T**, **T**):*

```
Enter True or False for a: T
Enter True or False for b: T
Enter True or False for c: T
a and b and c: True
a or b or c: True
XOR: False
Odd number: True
```

Part D: Other Activities (Optional 5 bonus points)

- Part D is strongly suggested for those wanting to major in Computer Science, Computer Engineering, and Electrical Engineering.
- If you would like to learn more about Boolean Expression Simplification, follow this link to see the list of rules used for the Boolean expression simplifications
  - http://sandbox.mc.edu/~bennet/cs110/boolalg/rules.html.
- Keep in mind that multiplication represents Boolean **and** and addition represents Boolean **or**. That means **AB** is equivalent to **A and B**, **A+B** is equivalent to **A or B**, and **A-bar** is equivalent to **not A**.
- Then take a look at the examples at http://sandbox.mc.edu/~bennet/cs110/boolalg/simple.html. Note that T stands for True.

Add to your program above using the variables `a`, `b`, and `c` from Part A and evaluate the following Boolean expressions.

1. (not (a and not b) or (not c and b)) and (not b) or (not a and b and not c) or (a and not b)
2. (not ((b or not c) and (not a or not c))) or (not (c or not (b and c))) or (a and not c) and (not a or (a and b and c) or (a and ((b and not c) or (not b))))

Then, create simpler (reduced) versions that still give the same results. Your program should output the value `True` or `False` based on the previously entered values. Output the results of the complex expression and the simple version.

Hint: There are 2 possible values for each of a, b, and c. Any expression that evaluates the same for all possible combinations of `True`/`False` of those values is an equivalent Boolean expression!

If your team completes Part D correctly, you will receive 5 bonus points on this assignment.

✓ *Example output for Part D (using inputs **T**, **T**, **T**):*

```
Complex 1: False
Complex 2: True
Simple 1: False
Simple 2: True
```