This problem set is due Wednesday, September 27 at 5pm, KST. Note that it is due at 5pm, not the usual 11:59pm because I want to give you my sample solutions early enough so that we can study them before the exam the next day. Also note that the due date that you see on Blackboard is not accurate since it shows the time in EST. You should go by the due date in this handout.

Also note that:

- This problem set assumes that you have installed Python and a Python programming environment such as PyCharm on your computer.
- Once you have installed PyCharm, you can use it to create programs using the editor that comes with PyCharm or use your favorite text editor.
- Add your name and email address as a comment at the top of each file you submit.

What to submit

Hand in your work on Blackboard. Your submission should include the following. Please do not submit other files that I did not ask for.

```
ps2.py
```

Multiple submissions are allowed before the due date and time. Late submissions will not be graded.

Assignment objectives

To become comfortable with the following important concepts in Python programming:

- Using conditionals (if-statements) to solve problems.
- Using loops (simple for-loops) to solve problems.
- Working with lists to store data and use them.

Incomplete work

I would rather see a partial solution that runs without any errors than an almost complete program that crashes. Any program that you hand in that crashes will not be worth much—it may earn about 25% of the available points, if any. This will apply to any program that you hand in throughout the semester. I repeat this warning once more this time, but will not in future problem sets.
Naming conventions in Python and programming style in general

Please use these conventions as you establish your *programming style*. Programming professionals use them, too. (I will not repeat this again from next time on.)

- **Names**: Choose informative names, e.g., `hourlyRate` rather than `hr` if it is to be a variable name. If you like `hourly_rate` better than `hourlyRate`, use that form. Whichever style you choose, stay with that style consistently.

- **Variable and function names**: We start a variable or function name with a lower-case letter, e.g., `count`, `hourlyRate`, etc., but not `Count`, `HourlyRate`, etc.

- **Use of white space**: Adopt a good indentation style using white spaces and be consistent throughout to make your program more readable.

Problem 1. BMI and Heart-risk (10 points)

In `ps2.py` write a function named `heartRisk` that has two parameters (age and bmi in that order) and returns a string that indicates what they mean according to the chart given below:

```
Write a function called heart_Risk that has two parameters: age and bmi and returns an appropriate value ('Low', 'Medium', or 'High') according to the following table:

<table>
<thead>
<tr>
<th>BMI</th>
<th>Age</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;22.0</td>
<td>&lt;45</td>
<td>Low</td>
</tr>
<tr>
<td>&gt;22.0</td>
<td>&gt;=45</td>
<td>Medium</td>
</tr>
</tbody>
</table>
```

For example, `heartRisk(20, 25)` should return 'Medium'.

Your test should call the function and print the returned value. Include enough test cases in `main` so that at least one of all possible outcomes is returned in your tests.

In your implementation use *nested* conditionals, an example of which is shown in the figure below.

```
def foo(x, y):
    if x%2 == 0:
        if y%2 == 0:
            print('both even')
        else:
            print('x even, y odd')
    else:
        print('x odd, y even')
    ```
Problem 2. Club Membership Fee Calculator (10 points)

In `ps2.py` write a function named `clubFee` that has three parameters, in this order:

1. `baseFee`: an integer that represents the normal (base) fee to be a member of the club.
2. `years`: an integer that represents the number of years a person has been a member of the club.
3. `gender`: a string that can be either 'Male' or 'Female'.

and returns an integer that represents an annual fee that a member has to pay. The club has the following policies in determining an annual fee:

- If `years` is greater than 9, the annual fee is a flat $1,000, regardless of `baseFee` and `gender`.
- If `years` is between 5 (inclusive) and 9 (inclusive) and `gender` is 'Male', then the membership fee is 90% of `baseFee`.
- If `years` is between 5 (inclusive) and 9 (inclusive) and `gender` is 'Female', then the membership fee is 80% of `baseFee`.
- If `years` is less than 5 and `gender` is 'Male', then the membership fee is 100% of `baseFee`.
- If `years` is less than 5 and `gender` is 'Female', then the membership fee is 90% of `baseFee`.
- If `years` is less than 2, then regardless of `gender`, the membership fee is 120% of `baseFee`.
- Regardless of any condition, a member must pay at least $50 a year.

Here are some example calls:

```
clubFee(2000, 15, 'Male') returns 1000
clubFee(1000, 8, 'Male') returns 900
clubFee(1000, 3, 'Male') returns 1000
clubFee(1000, 1, 'Female') returns 1200
clubFee(50, 8, 'Male') returns 50
```

Include these example calls in your tests of this function in `main`. Also add additional cases so that each branch of your function will be taken at least once.

Problem 3. Currency Calculator (15 points)

In `ps2.py` write a function named `currencyConvert` that has three parameters, in this order:

1. `amount`: a floating-point number that represents a monetary value.
2. `fromUnit`: a string that represents the currency value that `amount` is expressed in. Valid values include '$' (for US dollar), 'B' (Ethiopian Birr), and 'W' (for Korean Won).
3. `toUnit`: a string that represents the currency value that we want to convert `amount` to. Valid values include '$', 'B', and 'W'.

and returns a floating-point number that represents a converted monetary value. Simply put, the function takes `amount`, which is expressed on the currency value identified by `fromUnit`, and uses a foreign exchange formula to express `amount` on the currency value identified by `toUnit`.

Your function may assume that `fromUnit` and `toUnit` will only take on the values '$', 'W', or 'B'. There is no guarantee, however, that `fromUnit` ≠ `toUnit`.

Please use these exchange rates that I found at the time of this writing:
1 US dollar = 23.47 Ethiopian Birr
1 US dollar = 1127.09 Korean Won
1 Birr = 0.043 US dollar
1 Birr = 48.03 Korean Won
1 Korean Won = 0.00089 US dollar
1 Korean Won = 0.021 Ethiopian Birr

Here are some example calls:

currencyConvert(10, 'S', 'W') returns 11270.9
currencyConvert(10, 'B', 'W') returns 480.3
currencyConvert(100, 'W', 'B') returns 2.1

Include these example calls in your tests of this function in main. Also add additional cases so that all possible pairs are tested at least once each.

**Problem 4. Simple Lists (15 points)**

As you write your functions, not only in this problem but also in any problem, feel free to write any helper functions if you like.

(a) In ps2.py write a function named sumDiffs that has one parameter, numbers, which is a list of at least 2 integers and returns the sum of the differences between all two adjacent elements in the list. Here are some example calls:

Here are some example calls:

```
sumDiffs([1, 3]) returns 2
sumDiffs([1, 2, 3, 4, 5]) returns 4
sumDiffs([1, 3, 5, 9, 12]) returns 11
sumDiffs([-4, 3, 5, -9, 12]) returns 16
```

Include these example calls in your tests of this function in main. Also add at least two additional test cases.

**Note:** that we seem to be able to get the correct answer by subtracting the first element of the list from the last element of the list no matter what numbers you have in between those two numbers. A surprising property, it seems to me! However, I do not want you to use this property in writing your solution. This problem was designed to give you an exercise of dealing with loops and lists.

(b) In ps2.py write a function named zip that has two parameters, u and v, which are both a list of integers of same length and returns a list with their elements zipped together. Example calls can easily illustrate how they should work. Here are some example calls:

```
zip([1], [2]) returns [1, 2]
zip([1, 2, 3], [4, 5, 6]) returns [1, 4, 2, 5, 3, 6]
zip([3, 2, 1], [4, 6, 5]) returns [3, 4, 2, 6, 1, 5]
```

Include these example calls in your tests of this function in main. Also add at least two additional test cases.

(c) In ps2.py write a function named filterOutDuplicates that has one parameter, numbers, which is a list of integers and returns a list with all the duplicates eliminated. The relative order of elements in the original list may or may not be preserved in the result.

Here are some example calls:

```
filterOutDuplicates([1, 3, 2]) returns [1, 3, 2]
filterOutDuplicates([1, 1, 3, 1, 2]) returns [1, 3, 2]
filterOutDuplicates([1, 3, 2, 2]) returns [1, 3, 2]
filterOutDuplicates([1, 1, 3, 1, 2, 3, 2, 7, 2]) returns [1, 3, 2, 7]
```
Problem 5. Random Walk (15 points)

In *ps2.py* write a function named *walk* that has one parameter, *moves*, which is a list of strings that represent movements of an ant on a desert. Assume that this ant’s movements are somewhat limited in that it only travels on a straight line. The function counts and returns an integer value of how far from the starting position the ant has traveled based on the movements that appear in the list. Each movement is represented by a character as follows:

- an ’F’ means that the ant moves forward by one unit.
- a ’B’ means that the ant moves backward by one unit. If the ant is already at the starting position, the ant does not move. Moving backward when it is already at the starting position means falling off the cliff.
- a ’J’ means that the ant jumps over (skips over) the next move. So, a ’J’ move is used only when there is at least one more movement left in the list.
- an ’S’ means that the ant jumps back to its starting position.

You may assume that the list of strings is always valid and consists of only the letters ’F’, ’B’, and ’S’.

Here are some example calls:

```python
walk(['B']) returns 0
walk(['B', 'F', 'S', 'F', 'B']) returns 0
walk(['B', 'F', 'S', 'F', 'F']) returns 2
walk(['B', 'F', 'B', 'F', 'S', 'B', 'F']) returns 1
walk(['B', 'F', 'F', 'F', 'J', 'S', 'F']) returns 4
walk(['B', 'F', 'S', 'F', 'F', 'B', 'F', 'F']) returns 3
```

Include these example calls in your tests of this function in *main*. Also add at least two additional test cases.

Problem 6. Splitting a List (15 points)

In *ps2.py* write a function named *splitList* that has two parameters, in this order:

1. *numbers*: a list of integers.
2. *subListLength*: a positive integer that is used to split the list into smaller sublists; see below for more details.

The function splits the list into a list of sublists and returns a list of sublists. How this is done is easily understood by example. Suppose the given list *numbers* is [1, 2, 3, 4, 5, 6] and *subListLength* is 2. This value of *subListLength* indicates that each sublist will have two items.

Your function may assume that the length of *numbers* is always divisible by *subListLength*.

Here are some example calls:

```python
splitList([1, 2, 3, 4, 5], 1) returns [[1], [2], [3], [4], [5]]
splitList([1, 2, 3, 4, 5, 6], 2) returns [[1, 2], [3, 4], [5, 6]]
splitList([1, 2, 3, 4, 5, 6], 3) returns [[1, 2, 3], [4, 5, 6]]
splitList([1, 2, 3, 4, 5, 6], 6) returns [[1, 2, 3, 4, 5, 6]]
```

Include these example calls in your tests of this function in *main*. Also add at least three more interesting test cases.
Problem 7. Car Rental (20 points)

In `ps2.py` write a function named `carRental`, that takes `rentals`, which is a list of sublists, as its only parameter.

Each sublist always has three elements in the following order:

1. Type of membership: a string that is one of ‘S’ (for student), ‘F’ (for faculty), or ‘V’ (for visitor).
2. Type of car: a string that is one of ‘Sedan’, ‘Coupe’, ‘SUV’, or ‘Hybrid’.
3. Duration: the number of hours the car was rented for.

The cars provide free service for up to a certain number of hours based on the type of membership as follows:

1. ‘S’: first three hours are free.
2. ‘F’: first two hours are free.
3. ‘V’: first hour is free.

After free service is over, the members are charged based on the type of car they have rented:

- Sedan: $10 per hour
- Coupe: $12 per hour
- SUV: $13 per hour
- Hybrid: $15 per hour

Your function calculates the total money generated by each car type, i.e., how much each charged after free service was over. Store the integer income generated by each type of car at its corresponding index in the list `carType`, which has been provided for you as follows:

- at index 0 store the total income generated by Sedans.
- at index 1 store the total income generated by Coupes.
- at index 2 store the total income generated by SUVs.
- at index 3 store the total income generated by Hybrids.

Here are some example calls:

```python
carRental([[‘S’,‘Coupe’,4],[‘F’,‘Coupe’,4],[‘V’,‘Coupe’,4]])
returns [0, 72, 0, 0]
carRental([[‘S’,‘Coupe’,4],[‘F’,‘SUV’,4],[‘V’,‘Hybrid’,4],[‘V’,‘Sedan’,4]])
returns [30, 12, 26, 45]
carRental([[‘S’,‘Coupe’,3],[‘F’,‘SUV’,2],[‘V’,‘Hybrid’,1],[‘V’,‘Sedan’,4]])
returns [30, 0, 0, 0]
```

Include these example calls in your tests of this function in `main`. Also add at least two more test cases.