CS2316 - HOMEWORK 02: ORGO'S FARMS

Important

1. Due Date: **01/29/2020 at 11:59 pm**
2. This homework is graded out of **100** points. The GradeScope autograder will only show **up to 90 points** when you submit your code. We are saving some test cases for the final autograder that we will use. This is to encourage you to start developing ways to test features of your code when you aren't given test cases in an autograder. We also reserve the right to change the test cases later on as we see fit. Therefore, the grade reflected in Gradescope does not reflect your final grade on HW02.py
3. This is an individual assignment. You may collaborate with other students in this class. Collaboration means talking through problems, assisting with debugging, explaining a concept, etc. Students may only collaborate with fellow students currently taking CS 2316, the TA's and the lecturer. You should not exchange code or write code for others. For individual assignments, each student must turn in a unique program. Your submission must not be substantially similar to another student's submission. Collaboration at a reasonable level will not result in substantially similar code.
4. For Help:
   - TA Helpdesk (Schedule posted on class website)
   - Email TA's or use Piazza Forums Notes
   - How to Think Like a Computer Scientist
   - [http://openbookproject.net/thinkcs/python/english3e/]
5. Comment out or delete all function calls and unnecessary print statements. Only global variables, and comments are okay to be outside the scope of a function. When your code is run, all it should do is run without any errors.
6. Do not wait until the last minute to do this assignment in case you run into problems.
7. **Read the entire specifications document before starting this assignment.**
8. **HAVING FUNCTION CALLS OR EXTRANEOUS CODE OUTSIDE THE SCOPE OF FUNCTIONS WILL RESULT IN AN AUTOMATIC 0**.
9. **IF YOUR CODE CANNOT RUN BECAUSE OF AN ERROR, IT IS A 0%**

Introduction

The goal of this homework is to showcase your knowledge of Object-Oriented Programming (OOP). You should test your classes and methods out first on your own computer. Then when you have one or more of them working, upload the entire file (which must be named **HW02.py**) to GradeScope to see how well your code performs on the test cases we have created. You can submit the homework file as many times as you'd like before the deadline.

Background

Happy career fair season! You have just been hired as a consultant for a global organic agricultural products supplier company, Orgo. Serving customers around the world, Orgo is headquartered in Atlanta, GA, USA and owns 7 farms located in 4 countries. The farms supply the company with a variety of livestock and their products. Since the future of Orgo relies heavily on the farms and their livestock, Orgo pays close attention to the operations of its farms. You are given some raw data about the farms and their livestock (more below) and are expected to perform some basic analysis on the data. Your code should be versatile enough to handle future additions of new farms and livestock.

Make sure you write your code in the template we provided to you (HW02.py) and follow the formatting requirements for one line functions.

One line format:

Correct:

```
def function_a(param1):
    return [i for i in param1]
```

or

```
def function_a(param1):
    return type(param1)
```

Incorrect:

```
def function_a(param1):
    val = [i for i in param1]
    return val
```

or

```
def function_a(param1):
    return helper(param1)
def helper(param1):
    return [i for i in param1]
```

Classes and Functions for HW02.py

Note: See the grading rubric at the end of the document for point distribution.

Classes

Class Name: `Farm`
Class Attribute:

`company_name`	(str)	Name of the company the `Farm` belongs to. In this case, Orgo.

Instance Attributes:

`name`	(str)	Name of the `Farm`
`owner`	(str)	Name of the `Farm` owner
`country`	(str)	Country the `Farm` is in
`size`	(int)	Size of the `Farm` in acres
`num_of_livestock`	(int)	Number of `Livestock` the `Farm` owns
`num_of_workers`	(int)	Number of workers the `Farm` employs
`assets`	(int)	Amount of assets the `Farm` owns
`compatible_livestock`	(str)	Types of `Livestock` the `Farm` is able to keep. More on the format below.

Note:
The format of `compatible_livestock` will satisfy the following rules:

- Must be all lower case
- Types of `Livestock` should be separated by semicolons
- Types of `Livestock` should be in ascending alphabetical order
- Replace with the bool `None` if the `Farm` has no compatible livestock

Example: "deer;dog;donkey;mule;pig"

Description: Write a class called `Farm` with the above attributes. Write the appropriate methods to accomplish the following tasks:

Method #1

- Initializes the attributes in the sequence listed above.

Method #2

- Makes the instances of the `Farm` class sortable based on `assets`.

Method #3

- Checks if two instances of the `Farm` class are equal to each other.

- If a **`Farm`** has the same `name` and `owner` as another, then the two instances are equal to each other.

Method #4

- When a **`Farm`** instance is called in the Python shell or printed, the name of the **`Farm`** should be returned**.**

Class Name: **`Livestock`**
Instance Attributes:

`name`	(str)	Name of the **`Livestock`**. Should be all lowercase.
`price_in`	(float)	Price to buy the **`Livestock`**
`utilizations`	(str)	Things the **`Livestock`** could be used for. More on the format below.

Note:
The format of `utilizations` will satisfy the following rules:

- Must be all lower case
- Utilizations should be separated by semicolons
- Utilizations should be in ascending alphabetical order
- Replace with the bool `None` if the **`Livestock`** has no potential utilization

Example: "dairy;fur;meat;wool"
Description: Write a class called **`Livestock`** with the above attributes. Write the appropriate methods to accomplish the following tasks:

Method #1

- Initializes the attributes in the sequence listed above.

Method #2

- Makes the instances of the **`Livestock`** class sortable based on the number of potential `utilizations`.

Method #3

- Checks if two instances of the **`Livestock`** class are equal to each other.
  - If a **`Livestock`** has the same `name` as another, then the two instances are equal to each other.

Method #4

- When a **`Livestock`** instance is called in the Python shell or printed, the following statement should be returned**:**
  - "*name, price_in*"
  - Example: "dog, 200.0"
  - Make sure your output matches the format above exactly.

Functions

Function name: **`clean_farm_data`**
Parameters:

`raw_farm_data`	(list)	A list of lists with uncleaned farm data

Return Type: list
Description:
Note that we have included the raw farm data in the template's main function. Each sublist contains the information for one **`Farm`**.
This function should manipulate the raw data to ***meat*** the requirements of the instance attributes of **`Farm`** defined above. In other words, for each sublist, you need to make sure that each entry is *of the correct type, contains the desired information, and is in the right format*. Refer to the **`Farm`** definition above. Keep in mind that `compatible_livestock` should be set to the bool `None` if the **`Farm`** has no compatible livestock. Do NOT change the order of the sublists and their entries. Return a new list of lists with cleaned data.

Function name: **`clean_livestock_data`**
Parameters:

`raw_livestock_data`	(list)	A list of lists with uncleaned livestock data

Return Type: list
Description:
Note that we have included the raw livestock data in the template's main function. Each sublist contains the information for one **`Livestock`**.
This function should manipulate the raw data to ***meat*** the requirements of the instance attributes of **`Livestock`** defined above. In other words, for each sublist, you need to make sure that each entry is *of the correct type, contains the desired information, and is in the right format*. Refer to the **`Livestock`** definition above. Keep in mind that `utilizations` should be set to the bool `None` if the **`Livestock`** has no potential utilization. Do NOT change the order of the sublists and their entries. Return a new list of lists with cleaned data.

Function name: **`create_farm_instances`**
Parameters:

`farm_data`	(list)	A list of lists with cleaned farm data (returned from **`clean_farm_data()`**)

Return Type: list
Description:
This function should take in a list of cleaned farm data and create a list of **`Farm`** instances using the data. Return a new list with the **`Farm`** instances. Create the instances in the order of the sublists.

In order to receive full credit for this function you must use a list comprehension and follow the formatting mentioned above. Failure to do so will result in an 80% deduction for the points allocated to this function.

Function name: **`create_livestock_instances`**
Parameters:

`livestock_data`	(list)	A list of lists with cleaned livestock data (returned from **`clean_livestock_data()`**)

Return Type: list
Description:
This function should take in a list of cleaned livestock data and create a list of **`Livestock`** instances using the data. Return a new list with the **`Livestock`** instances. Create the instances in the order of the sublists.
In order to receive full credit for this function you must use a list comprehension and follow the formatting mentioned above. Failure to do so will result in an 80% deduction for the points allocated to this function.

Function name: **`buy_new_livestock`**
Parameters:

`livestock_list`	(list)	A list of **`Livestock`** instances
`livestock_quant`	(list)	A list of integers
`farm`	(**`Farm`**)	The **`Farm`** that buys new **`Livestock`**

Return Type: int
Description:
This function should model the process of a **`Farm`**'s attempt to buy new **`Livestock`**. `livestock_list` and `livestock_quant` will have the same length since the kth entry in `livestock_quant` represents the number of kth **`Livestock`** instance in `livestock_list` the **`Farm`** attempts to buy. For example, if `livestock_quant = [2,3,4]` and `livestock_list = [Livestock1, Livestock2, Livestock3]`, it means the **`Farm`** attempts to buy `2` of `Livestock1`, `3` of `Livestock2`, and `4` of `Livestock3`. The **`Farm`** can only buy livestock that are compatible. On a successful purchase, the `num_of_livestock` and `assets` of that **`Farm`** should be mutated accordingly (Hint: you will have to use the `price_in` instance variable of the **`Livestock`** instances). Return the number of livestock the **`Farm`** ends up buying **successfully**. You may assume that the **`Farm`**'s `assets` will never drop below 0.

Function name: **`mutate_workers`**
Parameters:

`farm`	(**`Farm`**)	The **`Farm`** that has a change of workers

`num_changed`	(int)	Change in the number of workers. Always positive. Equivalent to |delta worker|.
`addition`	(bool)	`True` if workers are being added. `False` otherwise.

Return Type: NoneType
Description:
This function should model the process of a **`Farm`** hiring new workers or firing existing workers. The `num_of_workers` on the **`Farm`** should be mutated accordingly. Keep in mind that the `num_changed` parameter is always positive. `addition` is `True` if a **`Farm`** is hiring new workers and `False` if it is firing existing workers.

Function name: **`sort_farms_assets`**
Parameters:

`farms_list`	(list)	A list of **`Farm`** instances to be sorted

Return Type: list
Description:
This function should return a sorted list of **`Farm`** instances based on the assets in descending order.
There is a one-line maximum requirement for this function, if this function is not written in one line you will receive 0 points.

Function name: **`sort_farms_num_of_liv`**
Parameters:

`farms_list`	(list)	A list of **`Farm`** instances to be sorted

Return Type: list
Description:
This function should return a sorted list of **`Farm`** instances based on the number of livestock in descending order.
There is a one-line maximum requirement for this function, if this function is not written in one line you will receive 0 points.

Function name: **`count_meat`**
Parameters:

`livestock_list`	(list)	A list of **`Livestock`** instances

Return Type: int
Description:
This function should return the number of livestock in `livestock_list` that could be potentially used as meat. Note that the `utilizations` attribute could be the bool `None`.

There is a one-line maximum requirement for this function, if this function is not written in one line, it will result in an 80% deduction for the points allocated to this function.

Function name: **`livestock_to_occurences`**
Parameters:

`livestock_list_dup`	(list)	A list of **`Livestock`** instances with duplicates (i.e. the `name` attribute of some **`Livestock`** instances might be the same)

Return Type: dict
Description:
This function should return a dictionary mapping the names of the **`Livestock`** in `livestock_list_dup` to their corresponding occurrences in the list. For example, one possibility could be `{'dog':2, 'cattle':8, 'llama':3}`, meaning there are 2 dogs, 8 cattles, and 3 llamas in `livestock_list_dup`.

Function name: **`remove_dup`**
Parameters:

`livestock_list_dup`	(list)	A list of **`Livestock`** instances with duplicates (i.e. the `name` attribute of some **`Livestock`** instances might be the same)

Return Type: list
Description:
This function should take in a list of **`Livestock`** instances with duplicates and returns a new list with no duplicate **`Livestock`**. Only include the first unique instance of **`Livestock`** in the list returned.

Function name: **`livestock_objs_to_dict`**
Parameters:

`livestock_list`	(list)	A list of **`Livestock`** instances

Return Type: dict
Description:
This function should return a dictionary representation of the **`Livestock`** instances in `livestock_list`. The keys of the dictionary should be the `name` attributes of the **`Livestock`** instances and the values should be tuples of the `price_in` attributes and the `utilizations` attributes. For example, one possibility could be

```
{

'goat':(1000.0, 'dairy;leather;meat;wool'),
'mule':(4400.2, 'draught')
```

```
}
```

In order to receive full credit for this function you must use a dictionary comprehension and follow the formatting mentioned above. Failure to do so will result in an 80% deduction for the points allocated to this function.

Function name: **`farm_to_density`**
Parameters:

`farms_list`	(list)	A list of **`Farm`** instances

Return Type: dict
Description:
This function should take in a list of **`Farm`** instances and return a dictionary mapping the names of the **`Farm`**'s to their livestock densities, which is calculated as the number of livestock on the farm divided by its size.
In order to receive full credit for this function you must use a dictionary comprehension and follow the formatting mentioned above. Failure to do so will result in an 80% deduction for the points allocated to this function.

Function name: **`livestock_to_util`**
Parameters:

`livestock_list`	(list)	A list of **`Livestock`** instances

Return Type: dict
Description:
This function should take in a list of **`Livestock`** instances and return a dictionary mapping the names of the **`Livestock`** to their number of potential utilizations. If the `utilizations` attribute is the bool `None`, the corresponding **`Livestock`** name should be mapped to 0.
In order to receive full credit for this function you must use a dictionary comprehension and follow the formatting mentioned above. Failure to do so will result in an 80% deduction for the points allocated to this function.

Function name: **`shortage_or_surplus`**
Parameters:

`demand_list`	(list)	A list of integers representing demands
`supply_list`	(list)	A list of integers representing supplies

Return Type: list
Description:
This function should take in a list representing customers' demands of livestock and a list representing Orgo's supplies of livestock and calculate the shortage or surplus of the livestock. Shortages should be represented as a negative number and surplus should be represented as a positive number. Essentially, you are calculating the difference

between these two lists. You may assume the `demand_list` and `supply_list` will always have the same length.

Example Case:

```
>>> demand_list = [300, 200, 900]
>>> supply_list = [600, 850, 100]
>>> shortage_or_surplus(demand_list, supply_list)
[300, 650, -800]
```

In order to receive full credit for this function you must use a list comprehension and follow the formatting mentioned above. Failure to do so will result in an 80% deduction for the points allocated to this function.

Function name: **`livestock_to_shortage`**
Parameters:

`livestock_list`	(list)	A list of **`Livestock`** instances
`diff`	(list)	A list of positive/negative integers

Return Type: dict
Description:
This function should return a dictionary mapping the names of the **`Livestock`** within `livestock_list` to the corresponding integer in `diff` only if the corresponding integer is **negative**. If the corresponding integer from `diff` is **positive**, you should not include the **`Livestock`** instance or the positive integer in your dictionary. Return an empty dictionary if all of the **`Livestock`** in `livestock_list` has a corresponding positive integer. You may assume `livestock_list` and `diff` will always have the same length, since each element in `diff` corresponds to a **`Livestock`** instance.

Example Case:

```
>>> livestock_list = [Livestock1, Livestock2, Livestock3]
>>> diff = [-200, 100, -550]
>>> livestock_to_shortage(livestock_list, diff)
{'dog':-200, 'cattle': -550}
```

*** Note that `'dog'` and `'cattle'` are the `name` attributes corresponding to `Livestock1` and `Livestock3` instances. ***

In order to receive full credit for this function you must use a dictionary comprehension and follow the formatting mentioned above. Failure to do so will result in an 80% deduction for the points allocated to this function.

Function name: **`livestock_shallow_copy`**
Parameters:

`livestock_list`	(list)	A list of **`Livestock`** instances

Return Type: list
Description:
This function should take in a list of `Livestock` instances and return a shallow copy of that list.
There is a one-line maximum requirement for this function, if this function is not written in one line you will receive 0 points.

Function name: `livestock_deep_copy`
Parameters:

`livestock_list`	(list)	A list of `Livestock` instances

Return Type: list
Description:
This function should take in a list of `Livestock` instances and return a deep copy of that list.
There is a one-line maximum requirement for this function, if this function is not written in one line you will receive 0 points.

Testing Your Code

We recommend that you call the functions in the main(), print out the output, and see if the output matches your expectations.

Gradescope/Canvas Requirements

- **NO UNNECCESARY PRINT STATEMENTS** as this will break the autograder
- **NO UNNECCESARY FUNCTION CALLS** outside of function definitions as this will also break the autograder
- Make sure the file submitted is titled **HW02.py**
- **Do not import any modules, packages, or libraries other than copy**
- **Only submit HW02.py file**

Grading Rubric

`class Farm:`	5 pts
`class Livestock:`	5 pts
`clean_farm_data:`	8 pts
`clean_livestock_data:`	8 pts
`create_farm_instances:`	5 pts
`create_livestock_instances:`	5 pts
`buy_new_livestock:`	5 pts
`mutate_workers:`	5 pts
`sort_farms_assets:`	5 pts
`sort_farms_num_of_liv:`	5 pts
`count_meat:`	5 pts
`livestock_to_occurences:`	5 pts
`remove_dup:`	5 pts
`livestock_objs_to_dict:`	5 pts
`farm_to_density:`	5 pts
`livestock_to_util:`	5 pts
`shortage_or_surplus:`	5 pts
`livestock_to_shortage:`	5 pts
`livestock_shallow_copy:`	2 pts
`livestock_deep_copy:`	2 pts
`total`	100/100 pts

Note that the autograder on Gradescope does not include test cases for `livestock_to_occurences` and `livestock_objs_to_dict`. Therefore, you are able to get up to 90 points on Gradescope before the assignment is due. We are saving those test cases for later use. You are encouraged to write your own test cases for these two functions.