ECE 364 Prelab Assignment 10

Object Oriented Programming in Python

Passing this lab will satisfy course objectives CO2, CO3, CO6

Instructions

- You must meet all *base requirements* in the syllabus to receive any credit.
- Work in your Prelab10 directory.
- Remember to add and commit all **required** files to SVN. **We will grade the version of the file that is in SVN!**
- Do *not* add any file that is *not* required. **You will lose points if your repository contains more files than required!**
- Make sure you file compiles. **You will not receive any credit if your file does not compile.**
- Name and spell the file, and the functions, exactly as instructed. Your scripts will be graded by an automated process. **You will lose some points, per our discretion, for any function that does not match the expected name.**
- Make sure your output from all functions match the given examples. **You will not receive points for a question whose output mismatches the expected result.**
- Unless otherwise specified, you cannot use any external library, but you can use any module in the **Python Standard Library** to solve this lab, i.e. anything under:

  `https://docs.python.org/3.7/library/index.html`

- Make sure you are using Python 3.7 for your Prelab.

Object Oriented Programming in Python

Required File `oop2Tasks.py`

Description

Create a Python file named `oop2Tasks.py`, and do all of your work in that file. This is the only file you need to submit. You can write any number of "**classes**" and helper functions in this file, but **DO NOT CREATE ANY MODULE VARIABLES**. Refer to the **Python File Structure** section below for a reminder on the requirements. Note that the use of type annotation is *not* required, but it is recommended for better compile-time static analysis.

Datum Class

Implement the `Datum` (singular of "Data") class, which defines a point in $\mathbb{R}^n$ Space. This class is a wrapper around, as well as an extension of, the regular `tuple` class.

Member Variables:

- `_storage` - An internal tuple holding the float values of the datum. (This variable must *not* be accessed from outside the class.)

Member Functions:

- **Initializer** - initializes an instance using `*args` to pass in a variable number of floats, and assign these values to the internal variable. Verify that each argument is a `float`[1], and raise a `TypeError` with an appropriate message otherwise.
- **String Representation** - returns a string representation of this instance in the following format:

  ```
  (X.XX, X.XX, ...)
  ```

  An example of the returned string would be:

  ```
  (3.14, -900.75, 33.00, ...)
  ```

  Note that, regardless of the actual data value, it should be formatted to have two decimals. Also note that you should implement both the `str` and `repr` functions.
- **Instance Hash** - returns a hash value of this instance using the `hash()` function, as:

  ```
  hash(self._storage)
  ```
- Write a function called `distanceFrom` that takes in a single argument of type `Datum`, then computes and returns the Euclidean distance between the current instance and the one passed. Raise a `TypeError` with a message if the input passed is of a different type.

  Note: If the argument instance has a different number of elements, treat the missing dimensions from either instances as having the value of 0, e.g. $(1.0, 2.0)$ should be treated as $(1.0, 2.0, 0.0, 0.0)$, if the distance is to be calculated from $(-1.54, 7.10, 9.00, 15.33)$.
- Write a function called `clone` that takes in no arguments, creates and returns a new instance, as a deep copy, of the current instance.

[1] Whenever you are required to validate for a "float", it goes without saying that "int" is also acceptable.

Collection Functions:

The following member functions are part of the `Collection` protocol, which means that, by implementing them, this class qualifies, or can be treated, as a collection. (Refer to the `collections.abc` module for more information.)

- `float in Datum` - implements a membership check to identify whether that `float` is present in the current instance or not. If it is present, return `True`, otherwise return `False`. Raise a `TypeError` if the item to be checked is not a float.
- `len(Datum)` - implements a size check and returns the length of the internal storage.
- `iterator over Datum` - returns an iterator object that allows for iterating over the values as:

  ```
  iter(self._storage)
  ```

  Note: This allows for iterating over the values using the syntax: `for v in Datum`.

Operator Overloads:

- `- Datum` - computes and returns a new instance, where the each value is negated.
- `Datum[i]` - returns the i^{th} element from the internal storage.
- `Datum1 +,- Datum2` - implements the `'+','-'` operator that adds/subtracts two `Datum` instances to create and return a new instance.

  Notes:

  - Addition operation is commutative, while subtraction is not.
  - If there is a size mismatch, treat the missing dimensions from either instances as having the value of 0.
  - Check if the other instance used is of the `Datum` class. If it is not, raise a `TypeError` with a descriptive error message.
- `Datum +,-,*,/ float` - implements all the mathematical operations over a `Datum` instance and a float, and creates and returns a new instance of the `Datum` class with the operation applied to each element.

  Notes:

  - Addition and multiplication operations are commutative, while subtraction and division are not.
  - Check if the other instance is a `float`, and raise a `TypeError` with a descriptive error message if it is not.
- **Rich Comparison** - implements rich comparison (i.e. all the operators $=, \neq, <, >, \leq, \geq$) between two `Datum` instances, and returns `True` or `False`, based on the Euclidean distance from the origin. Check if the other instance used is of the `Datum` class. If it is not, raise a `TypeError` with a descriptive error message.

Data Class

Implement the `Data` class that inherits from `collections.UserList`, which offers all the functionality of list, but allows for adding more functionality.

Member Functions:

- **Initializer** - initializes an instance by taking a single input argument as a list of `Datum` instances that gets passed to the parent class, defaulted to `None`, i.e. `initial=None` (Refer to the requirements for sub-classing `UserList`.) If the argument is `None`, pass an empty list to the parent class initializer, and not the `None` object. Verify that each element in the list is a `Datum` instance, and raise a `TypeError` with an appropriate message otherwise.
- Write a function called `computeBounds` that takes in no arguments, and returns a `(minDatum, maxDatum)` tuple of two `Datum` instances, representing the coordinates of the bounding hyper-cube that surrounds all of the data in the current instance.

  A bounding hyper-cube is a region represented by minimum and maximum coordinates that completely enclose a group of elements. To compute the bounding hyper-cube, we must find the minimum and maximum coordinate of each dimension. Consider the following data points:

  ```
  (1, 4, 7), (6, 3, 4), (9, 0, 5)
  ```

  The maximum in each dimension would yield the `(9, 4, 7)` and, equivalently, the minimum would yield `(1, 0, 4)` and these represent the coordinates of the bounding box (In 3D, the hyper-cube is simply a box.) The same argument holds for instances of larger sizes. (If you have a dimension mismatch, treat missing dimensions as having the value of 0.)
- Write a function called `computeMean` that takes in no arguments, and returns a single `Datum` instance, representing the mean over all `Datum` instances. Similar to the function above, compute the average value of each dimension, and use that to construct a new `Datum` instance.

Member Overrides:

- Override the following member functions to check if the passed argument is an instance of the `Datum` class:

  ```
  append(), count(), index(), insert(), remove(), __setitem__
  ```

  If the check passes, invoke the relevant parent function (e.g. `super().append(item)`,) and pass the argument to it . If the check fails, raise a `TypeError` with an appropriate message.
- Override the following member function to check if the argument passed is an instance of the `Data` class.

  ```
  extend()
  ```

  If the check passes, invoke the relevant parent function and pass the argument to it. If the check fails, raise a `TypeError` with an appropriate message.

DataClass Enum

Create a `DataClass` enum that represents a class in a pattern recognition application with the following options:

```
Class1
Class2
```

DataClassifier Class

Implement the `DataClassifier` class that performs simple data classification based on input data.

Member Variables:

- `_class1` - An internal instance of a `Data` class, holding the points defining the first class. (This variable must *not* be accessed from outside the class.)
- `_class2` - An internal instance of a `Data` class, holding the points defining the second class. (This variable must *not* be accessed from outside the class.)

Member Functions:

- **Initializer** - initializes an instance by taking two arguments, `group1` and `group2` as two instances of the `Data` class to initialize the internal members. Verify that each argument is of the correct type, and raise a `TypeError` with an appropriate message otherwise. Also, check that both inputs are *not* empty, and raise a `ValueError` with an appropriate message otherwise
- Write a function called `classify` that takes in a single `Datum` instance, and returns an instance of the `DataClass` enum based on what class the argument belongs to.

  The input `Datum` belongs to the class whose "mean" `Datum` is closer to the input `Datum`, using the Euclidean distance as measure. (If you have a dimension mismatch, treat missing dimensions as having the value of 0.)

Python File Structure

The following is the expected file structure that you need to conform to:

```
#########################################################
#   Author:     <Your Full Name>
#   email:      <Your Email>
#   ID:         <Your course ID, e.g. ee364j20>
#   Date:       <Start Date>
#########################################################

import os     # List of module import statements
import sys    # Each one on a line
...


# Module level Variables. (Write this statement verbatim.)
#########################################################
DataPath = os.path.expanduser('<Path Provided to You>')

class ClassName1:
        ...


class ClassName2:
    ...


def functionName1(a: float, b: float) -> float:
    ...



def functionName2(c: str, d: str) -> int:
    ...



# This block is optional
if __name__ == "__main__":
# Write anything here to test your code.
    ...
```

