CSE 231 **Fall 2022**

Computer Project #11

Assignment Overview

self represents the instance of the class. By using the "self" we can access the attributes and methods of the class in python.

This assignment focuses on the design, implementation, and testing of a Python class to manipulate measurements of fluid volumes, as described below.

It is worth 50 points (5% of course grade) and must be completed no later than 11:59 PM on **Wednesday, 12/07/2022**.
After the due date, 1 point will be deducted for every 1 hour late. No more submissions will be accepted after Thursday 12/08/2022.
If you submit by Tuesday, 12/06/2022, you will receive a 2-point bonus. Note that you need to click on the submit button to be considered as submitted.

Assignment Deliverables

The deliverables for this assignment are the following files:

- **`volume.py`** – the source code for your **`class Volume`**

Be sure to use the specified file name and to submit them for grading before the project deadline.

Assignment Background

In the United States, fluid volumes are commonly measured using two different systems: the metric system (which includes units such as milliliters, centiliters, liters, etc.) and US customary system (which includes units such as ounces, quarts, gallons, etc.).

You will develop a Python class named `Volume` which allows the user to manipulate volume measurements in a subset of those units: ounces and milliliters. The methods supported by class `Volume` are specified below.

For the purposes of this project, 1 fluid ounce is defined to be 29.5735295625 milliliters. You need this conversion value to match test results. It is given as a constant variable in the starter code.

Specifications for Class Volume

1. Your implementation of "class Volume" must be in the file named **`volume.py`** (and it must be the only thing in that file).

2. The constructor will accept two parameters: the `magnitude` of the volume and the `units` in which it is measured. A valid object of type `Volume` has:

- a non-negative magnitude (a value of type `int` or type `float`), and
- "ml" (meaning milliliters) or "oz" (meaning ounces) as the units.

By default, the magnitude of the volume is zero and the units are milliliters. Thus, the following statements all construct valid `Volume` objects:

```
A = Volume( 10.0, "ml" )
B = Volume( 35, "ml" )
C = Volume( 68.5, "oz" )
D = Volume( 20, "oz" )
E = Volume( 12.7 )
F = Volume( 0 )
G = Volume()
```

An invalid `Volume` object is constructed if the parameters are not valid. If the `magnitude` is not a valid magnitude, the Volume will have its `magnitude` set to 0 and its `unit` set to `None`. If the `unit` is not valid, the `Volume` will have both the `magnitude` and `unit` set to `None`. If the `Volume` object is constructed with no arguments (e.g., `G`), units defaults to `"ml"` and magnitude defaults to `0`.

3. The methods `__repr__` and `__str__` will return a representation of the current object as a character string.

For a valid object, the method `__repr__` will display the magnitude (rounded to six decimal places) and the units, with one space between them.

For a valid object, the method `__str__` will display the magnitude (rounded to three decimal places) and the units, with one space between them. For example, `3` will be printed as `3.000`. Hint: use the format() method.

Both methods will <u>return</u> the string `"Not a Volume"` for invalid objects.

Neither method will alter the current object (there will be no changes to the instance variables inside the object).

4. The following methods will behave as indicated:

```
# Assume V is an object of type Volume

V.is_valid()     # Returns a Boolean to indicate whether or not
                 # V is valid.

V.get_units()    # Returns units stored during construction
                 # ("ml" or "oz" if V is valid, None otherwise).
```

```
V.get_magnitude()# Returns magnitude stored during construction
                 # (numeric value if magnitude of V is valid or 0,
            None otherwise).



V.metric()       # Returns a Volume object equivalent to V
                 # (in the metric system if V is valid otherwise
            it returns an invalid Volume object). If V is already
            in the metric system, returns a new Volume object that
            has the same magnitude and unit as V. Do not change V.

V.customary()    # Returns a Volume object equivalent to V
                 # (in the US customary system if V is valid).
            otherwise, it returns an invalid Volume object). If V
            is already in the US customary system, returns a new
            Volume object that has the same magnitude and unit as
            V. Do not change V.
```

None of the methods will alter the current object (there will be no changes to the instance variables inside the object).

5. The class will have an __eq__ method which will be called when the == will be called (remember the __eq__ method n the Card class). For example:

```
V1 = Volume( 50.0, "ml" )
V2 = Volume( 60.0, "oz" )

V1 == V2  (this will call the method __eq__)
```

The methods will compare the magnitudes of the two "Volume" objects; if the objects are in different systems, the comparison will be performed on equivalent magnitudes in the same system.

For equality, you need to consider that you cannot effectively compare floats using == because floats are approximations and exact equality isn't practical. The solution is to define two floats as equal if the *absolute value of their difference is less than some small value*. Use
`DELTA = 0.000001`
as that small value (use this DELTA so your results will pass our tests).

If one (or both) of the objects is invalid, the comparison method will return the Boolean value **`False`**.

the comparison method will not alter the current objects (there will be no changes to the instance variables inside the objects).

6. The class will support addition (and subtraction) of two "Volume" objects. For example:

```
V1 = Volume( 50.0, "ml" )
V2 = Volume( 60.0, "oz" )

V1.add(V2)
V1.sub(V2)
```

The sum (or difference) will be computed based on the magnitude of the two objects; if the objects are in different systems, the computation will be performed on equivalent magnitudes in the same system. The resulting object will be in the same system as the left-hand object (V1).

If one (or both) of the objects is invalid, the methods will return an invalid object.

The class will all support addition (and subtraction) of a `Volume` object and a numeric value. For example:

```
V = Volume( 50.0, "ml" )

V.add(2.5)
V.sub(3.5)

V.add(20)
V.sub(30)
```

The sum (or difference) will be computed based on the magnitude of the `Volume` object; the resulting object will be in the same system as the original `Volume` object.

If the object is invalid, the methods will return an invalid object.

Neither method will alter the current object (there will be no changes to the instance variables inside the object). Both methods will return a new Volume object with the new magnitudes and units.

9. None of the methods in class `Volume` will use function "print". That is, they will not display messages to the user, even if an error is detected.

Assignment Notes

1. The coding standard for CSE 231 is posted on the course website:

   http://www.cse.msu.edu/~cse231/General/coding.standard.html

Items 1-9 of the Coding Standard will be enforced for this project.

2. Your source code may not use any global variables. That is, you may not use any variables outside of functions.

3. Your source code may not use any nested functions. That is, you may not nest the definition of a function inside another function definition.

4. Please note that you may wish to use function "print" to display various items as you are developing your implementation of the class. However, all invocations of function "print" must be removed from the final version of your class (or turned into comments).

5. You would be wise to develop your implementation of class `Volume` and your test program incrementally and in parallel. That is, implement one class method at a time and then test that method.

Clearly, the first class method which must be implemented and tested is the constructor (`__init__`). However, you might choose to first implement the constructor without worrying about erroneous parameters (error handling could be added later).

Perhaps, the second class method to implement and test is `__str__` so that you have a way to display the value of an object of type `Volume` using function "`print`".

Then, you might be wise to implement and test a relatively simple method, such as `get_units`.

After implementing and testing those three methods, you would continue to implement and test one method at a time until you have completed the class.

Be sure to insert and test any necessary error handling at the appropriate time. For example, it makes no sense to add error handling to method "metric" until the constructor detects invalid parameters.

Expected Return Values

Consider the following Python statements and the values which should be returned by selected methods:

```
import volume

A = volume.Volume( 3, "oz" )

str(A)              '3.000 oz'
repr(A)             '3.000000 oz'
A.get_magnitude()       3
A.get_units()       'oz'

B = volume.Volume( 1.999999, "ml" )

str(B)              '2.000 ml'
repr(B)             '1.999999 ml'
```

```
B.get_magnitude()       1.999999
B.get_units()       'ml'


C = volume.Volume( 2.5, "tsp" )

str(C)              'Not a Volume'
repr(C)             'Not a Volume'
C.get_magnitude()   None
C.get_units()       None

C1 = volume.Volume( -2.5, "ml" )

str(C1)             'Not a Volume'
repr(C1)            'Not a Volume'
C1.get_magnitude()  0
C1.get_units()      None

C2 = volume.Volume( '1', "ml" )

str(C2)             'Not a Volume'
repr(C2)            'Not a Volume'
C2.get_magnitude()  0
C2.get_units()      None



D = volume.Volume( 4.5678 )

str(D)              '4.568 ml'
repr(D)             '4.567800ml'
D.get_magnitude()       4.5678
D.get_units()       'ml'

E = D.metric()

str(D)              '4.568 ml'
repr(D)             '4.567800 ml'
D.get_magnitude()       4.5678
D.get_units()       'ml'

E = D.customary()

str(E)              '0.154 oz'
repr(E)             '0.154456 oz'
E.get_magnitude()       0.15445569289747846
E.get_units()       'oz'
```

Note that none of the methods call function "print"; the value shown next to each method invocation is simply the value which should be returned by that invocation.

Grading Rubric

Computer Project #11 Scoring Summary

General Requirements:

(5 pts) Coding Standard 1-9
(descriptive comments, function headers, mnemonic identifiers, format, etc...)

Implementation:

(12 pts) __init__, __str__, __repr__, is_valid, get_units, get_magnitude

(8 pts) metric,customary
(8 pts) __eq__
(10 pts) add and sub a Volume
(7 pts) add and sub a constant

Note: hard coding an answer earns zero points for the whole project