CSE 101: Introduction to Computational and Algorithmic Thinking

Lab #8

Fall 2018

Assignment Due: Tuesday, November 20, 2018, by 11:59 pm

Assignment Objectives

By the end of this assignment you should be able to design, code, run and test original Python functions that solve programming problems involving file I/O, classes, and other Python concepts.

Getting Started

The assignments in this course require you to write Python code to solve computational problems. To help you get started on each assignment, we will give you a "bare bones" file with a name like `lab8.py`. These files will contain *function stubs* and a few tests you can try out to see if your code seems to be correct. Stubs are functions that have no bodies (or have very minimal bodies). You will fill in the bodies of these functions for the assignment. **Do not, under any circumstance, change any class or function names that we have provided for you (or modify the parameter lists of those functions).** The automated grading system will be looking for exactly those class and function names provided in `lab8.py`. **Please note that the test cases we provide you in the bare bones files will not necessarily be the same ones we use during grading!**

Directions

Complete the following assignment to the best of your ability. This assignment is worth a total of 20 points in all, divided between two classes, their methods, and two external functions. The automated grading system will execute your solution to each problem several times, using different input values each time. Each test that produces the correct/expected output will earn 1 point. You must earn at least 16 points (80%) in order to pass (receive credit for) this lab.

- At the top of the `lab8.py` file, include the following information in comments, with each item on a separate line:
  - your full name ***AS IT APPEARS IN BLACKBOARD***
  - your Stony Brook ID #
  - your Stony Brook NetID (your Blackboard username)
  - the course number (CSE 101)
  - the assignment name and number (Lab #8)

⚠ Each of your functions must use the names and parameter lists indicated in the starter code file. Submissions that have the wrong function names (or whose functions contain the wrong number of parameters) can't be graded by our automated grading system, and may receive a grading penalty (or may not be graded at all).

⚠ Be sure to submit your final work as directed by the indicated due date and time. Late work will not be accepted for grading. Work is late if it is submitted after the due date and time.

⚠ Programs that crash will likely receive a grade of zero, so make sure you thoroughly test your work before submitting it.

A Very Fishy Problem

Previously in lecture, we discussed Python classes, which allow us to define our own data types that combine data (instance variables) and operations (instance methods). In this lab, we will use classes to create a virtual aquarium.

1. Start by completing the `Fish` class, which will represent a single fish in our aquarium. Every `Fish` has four attributes, represented as instance variables: its name, its species, its color, and its weight in ounces. A `Fish` also has the following methods (with the following headers):

   - `__init__(self, name, species, color, weight)`

     This method creates four new instance variables for the current `Fish`, and assigns them the values of the last four method parameters. Remember that an instance variable's name is always preceded by "self.", as in `self.myData`.

   - `__repr__(self)`

     This method returns a string representing the current `Fish`. This string should include a crude fish symbol (`<><`) followed by the `Fish`'s name, color, and species in parentheses. For example:

     `<>< (Topper, blue Guppy)`

   - `__lt__(self, other)`

     This method returns `True` if the current `Fish`'s weight is less than the other `Fish`'s weight instance variable, and `False` otherwise. Remember that Python does not protect instance variables from outside access, so you can directly refer to the weight instance variable from the `other` parameter.

   - `kind(self)`

     This method returns the species of the current `Fish`.

2. Complete the `createFish()` function, which takes a single string argument in the form:

   species name color weight

   where each field is separated by a single space (assume that the species and name are always single-word values). This function should create and return a new `Fish` object whose attributes are based on the contents of this string. Note that the fields of the string are in a slightly different order than the argument list of the `Fish` class, and that the "weight" value must be converted from a string to a `float` value before the new `Fish` is created.

3. We also need an aquarium to hold our fish. To do this, we need to define a second class, named `Aquarium`. An `Aquarium` has two instance variables: a list of `Fish` instances, and an integer representing the water capacity of the tank in gallons (each `Fish` requires 1 gallon of water). An `Aquarium` also has the following methods:

   - `__init__(self, capacity)`

     This method creates a new instance variable that is an empty list; this list will eventually be filled with `Fish` objects. It also assigns its second parameter to a new instance variable representing the `Aquarium`'s total capacity in gallons.

   - `__repr__(self)`

     This method returns a single string that contains the string representations of the `Fish` in the `Aquarium`'s internal list, in the same order that they appear in that list. Insert tab (`"\t"`) and newline (`"\n"`) characters to ensure that, when this string is displayed, no more than two `Fish` are printed per line (if there are an odd number of fish, the last one will be printed on a line by itself). For example, a call to this method might return a string like

     `"<>< (Bill, red Gourami)\t<>< (Ted, yellow Goldfish)\n<>< (Mary, pink Mollie)"`.

     Note that you can call `str()` on a `Fish` (or any other object) to automatically invoke its `__repr__()` method.

- `add(self, newFish)`

  If the number of `Fish` in the `Aquarium` is less than its capacity, this method appends its argument to the list of `Fish` and returns `True`. Otherwise, it returns `False` without modifying the list.

- `population(self)`

  This method returns the number of `Fish` that are currently held in the `Aquarium`.

- `countType(self, type)`

  This method returns the number of `Fish` in the `Aquarium` whose species matches the `type` parameter. For example, `myTank.countType("Guppy")` would return the number of `Fish` in `myTank` that have "Guppy" as their species.

4. Finally, complete the `fillAquarium()` function. This function takes two arguments: a string representing the name of a (plain text) data file, and a positive (non-zero) integer representing the tank capacity in gallons. It returns a new `Aquarium` object. `fillAquarium()` does the following:
   (a) It creates a new `Aquarium` with the specified capacity.
   (b) It uses a loop to open the specified data file. For each line in the data file:
      - The function calls `createFish()` with the current line to get a new `Fish`.
      - The function attempts to add the new `Fish` to the `Aquarium` object.
   (c) It returns the new `Aquarium`.

Sample Output

When you have completed the assignment, your program should produce output similar to the following using the supplied sample data files:

```
Creating three new fish...
Fish 1: <>< (Homer, red Guppy)
Fish 1 is a Guppy
Fish 2: <>< (Marge, blue Cichlid)
Fish 2 is a Cichlid
Fish 3: <>< (Bart, yellow Tetra)
Fish 3 is a Tetra
Fish 1 is larger than Fish 2
Fish 1 is larger than Fish 3
The test tank's current population is: 0
Adding fish now...
The test tank's new population is: 3

<>< (Homer, red Guppy) <>< (Marge, blue Cichlid)
<>< (Bart, yellow Tetra)

The test tank contains 1 Cichlid(s).
The test tank contains 1 Guppy (or Guppies).
The test tank contains 1 Tetra(s).
The test tank contains 0 Mollie(s).
```

```
Creating a new 15-gallon aquarium from tank1.txt...
tank1 contains 12 fish:

<>< (Alvin, orange Tetra) <>< (Simon, gray Guppy)
<>< (Theodore, blue Oscar) <>< (Bart, yellow Cichlid)
<>< (Lisa, green Barb) <>< (Homer, yellow Goldfish)
<>< (Marge, red Gourami) <>< (Maggie, pink Mollie)
<>< (Ned, gray Cichlid) <>< (Maude, orange Barb)
<>< (Krusty, white Gourami) <>< (Bob, green Cichlid)

Creating a new 30-gallon aquarium from tank2.txt...
tank2 contains 20 fish:

<>< (Zippy, yellow Barb) <>< (Sparky, green Goldfish)
<>< (Itchy, white Barb) <>< (Scratchy, black Cichlid)
<>< (Manny, green Mollie) <>< (Moe, yellow Gourami)
<>< (Jack, white Gourami) <>< (Dexter, blue Goldfish)
<>< (Rob, black Gourami) <>< (Eddie, yellow Tetra)
<>< (Jesse, gray Mollie) <>< (Dave, gray Cichlid)
<>< (Danny, yellow Guppy) <>< (Billy, yellow Gourami)
<>< (Mary, gray Mollie) <>< (Sue, red Oscar)
<>< (Barbara, yellow Mollie) <>< (Dinah, orange Oscar)
<>< (Ellen, gray Oscar) <>< (Joan, yellow Guppy)

Creating a new 10-gallon aquarium from tank3.txt...
tank3 contains 10 fish:

<>< (Mario, black Tetra) <>< (Luigi, white Mollie)
<>< (Peach, orange Guppy) <>< (Toad, red Guppy)
<>< (Bowser, yellow Mollie) <>< (Koopa, black Barb)
<>< (Link, black Gourami) <>< (Zelda, green Mollie)
<>< (Luke, orange Cichlid) <>< (Leia, red Tetra)
```