MASSACHVSETTS INSTITVTE OF TECHNOLOGY
Department of Electrical Engineering and Computer Science
6.01—Introduction to EECS I
Fall Semester, 2007

SOLUTIONS – NanoQuiz #2 (R 20 Sep) – SOLUTIONS

1. Define a class `FruitSalad` with class attributes `fruits` and `servings`. Write an `__init__` method that takes arguments `ingredients` (a list of strings) and `numServings` (an integer) and stores the supplied values in instance attributes `fruits` and `servings` respectively. Write a `__str__` method that returns a string containing the number of servings[1] and the contents of the fruit salad. Write a method `FruitSalad.add(ingredient)` that appends the (string) `ingredient` to `fruits`. Finally, write a method `FruitSalad.serve()` that prints `"enjoy"` if it has been called a number of times that is less than or equal to the value of `numservings` supplied when the associated instance was created, or `"sorry"` otherwise.

   Solutions are in **bold** below.

```
class FruitSalad:
    fruits = ["melon", "pineapple"]          # class attribute
    servings = 4                             # class attribute
    def __init__( self, ingredients, numServings ):
        self.fruits = ingredients            # instance attribute
        self.servings = numServings          # instance attribute

    # there were many ways to write this; pretty much everything got full credit
    def __str__( self ):                     # must supply self as first parameter
        return str(self.servings) + " servings with " + str(self.fruits)

    def add( self, ingredient ):             # must supply self as first parameter
        self.fruits += [ingredient]          # append 1-element list to existing list
# OR    self.fruits.append( ingredient )     # this would work just as well

    def serve( self ):                       # must supply self as first parameter
        if ( self.servings > 0 ):
            print "enjoy"
            self.servings -= 1               # account for this serving
        else:
            print "sorry"
```

   Remarks on common errors, and on grading:

   (a) In general, answers that showed a clear understanding of OOP concepts scored 5 or nearly 5. Answers with a significant missed concept scored 4 or nearly 4. Answers with several significantly missed concepts scored 3 or nearly 3. Answers that showed little or no understanding received a 2.

[1] Admittedly, this would have been clearer for some students had it read "the number of *remaining* servings".

(b) Some answers tried to define a new locally scoped variable `numCalls` or the like within `__init__` and modify it within `serve`; this won't work since the two instances of `numCalls` will have different scopes (why?). Indeed, attempting to modify `numCalls` within `serve` will cause an error (why?). This kind of error caused a deduction of one point.

Answers that defined and initialized a new instance attribute `self.numCalls` within `__init__`, and modified it within `serve()`, got full credit, even though this wasn't the most elegant solution to the problem.

(c) Some answers confused *class* attributes (those associated with the class `FruitSalad`) with *instance* attributes (those associated with a particular instance of an object of class `FruitSalad`, e.g. the object `salad`). Class/instance confusion caused a deduction of between one-half and one point, depending on severity and how many times it occurred.

(d) Some answers omitted the obligatory first argument (named `self` by convention) of one or more class methods. Omission of one `self` cost nothing; omission of two or more cost half a point.

(e) Many answers had minor errors such as storing values to `salad.numServings`, swapping `"enjoy"` and `"sorry"`, adding a string to a list (rather than a list to a list), omission of `str()` when printing a list or integer, using `return` within `add()` or `serve()`, spurious quotes around the printed output of `serve()`, and so forth. The first few minor mistakes cost nothing; if there were many such mistakes we deducted up to half a point total.

(f) A nit: many students unconditionally modified the counter variable within `serve()`. This would be a bug in a language that, unlike Python, provided finite-precision integers (why?). Even in Python, this implementation could be considered a bug since it performs unnecessary work.

Fill in the interpreter's responses in the blanks below. (Solutions in **bold**.)

```
>>> salad = FruitSalad(["bananas", "apples"], 2)
>>> print salad
2 servings of fruit salad with ['bananas', 'apples']
>>> salad.add("cherries")
>>> print salad.fruits              # instance attribute of salad
['bananas', 'apples', 'cherries']
>>> print FruitSalad.fruits
[”melon”, ”pineapple”]              # class attribute of FruitSalad
>>> salad.serve()
enjoy
>>> salad.serve()
enjoy
>>> salad.serve()                   # must do this on 3rd serving, as per spec'n.
sorry
>>> print salad.servings            # instance attribute now decremented to zero
0
>>> print FruitSalad.servings       # class attribute still 4 as per class def'n.
4
```