CS 1110, LAB 7: FOR-LOOPS AND CLASSES

Name: ______________________________ **Net-ID**: ______________

There is an online version of these instructions at

http://www.cs.cornell.edu/courses/cs1110/2012fa/labs/lab7

You may wish to use that version of the instructions.

The lab for this week is a bit of a hodge-podge, reflecting that we are in transition between topics. Even though you are currently working on Assignment 4, we still wanted to give you an in-lab exercise with a simple for-loop. In addition, we wanted to give you a simple class definition to work on so that you do not go over a week with no exposure to classes.

While this lab is two different exercises, we have tried to keep this lab short. You have an assignment due before the next lab, and you just got finished with a lengthy lab. Ideally, you should be able to finish this all in lab time.

Requirements For This Lab. There are no files to download for this lab. Furthermore, while you will be writing and testing code in Python, you do not need to turn in any Python modules or any unit tests. All of your code is short enough that you can write it directly on this handout. Show this handout to your instructor, and you are done.

As always, you should try to finish the lab during your section. However, if you do not finish during section, you have **until the beginning of lab next week to finish it**. You should always do your best to finish during lab hours; remember that labs are graded on effort, not correctness.

---

1. For-Loops

You might remember the following function from Lab 5:

```
def lesser_than(thelist,value):
    """Returns number of elements in thelist strictly lesser
    than value, without altering thelist.

    Example:  lesser_than([5, 9, 1, 7], 7) evaluates to 2

    Precondition:  thelist is a list of ints; value is an int"""
```

In that lab you implemented this function using the `sort()` method inside of `thelist`. This was a bit overly complicated, but it works. However, it also required that `value` be an element of the list, which is no longer part of our precondition. It is much easier to implement this function with a for-loop and a counter variable. Do that on the next page.

```
def lesser_than(thelist,value):
```

Testing it Out. As we do not require that you turn in any code, we are obviously not requiring that you write a unit test either. However, you might want to test your function before continuing. Assuming that you implemented your function in a module called `loop.py`, type the following into Python at the interactive prompt:

```
>>> from loop import lesser_than
>>> lt = lesser_than([5, 9, 1, 7], 7)
>>> lt
```

If you get the number 2, then your function is likely working correctly.

---

2. The Time Class

In the second part of this lab, you will implement a simple class, called `Time`. This class will have the following contents:

- Two fields: `hours` and `minutes`.
- A constructor method `__init__`.
- A method `__str__` to implement the `str()` function.

The class is relatively simple. The entire class, including the two fields and both methods, but not including specifications, should take no more than 20 lines (our implementation is less than 15).

Below we will take you step-by-step through this class. We assume that you will write this in a Python module and test it out as you go along. We you are done, the class should look very familiar to you. Write your final code in the space provided at the end of this handout.

The Class Definition. The first line of your class definition should be as follows:

```
class Time(object):
```

Everything else in this part of the lab should be indented underneath that.

The Fields. The two fields have the following invariants.

Attribute	Invariant
`minutes`	`int` value in range `0..59`
`hours`	`int` value ≥ 0

Right now, however, you do not know how to enforce invariants. Simply create the fields by assigning them a default value of 0. Attach the invariants as single-line comments after the assignment to remind yourself that they are there.

The Constructor. Create a constructor (e.g. a method `__init__`) with the following specification:

```
"""Constructor:  makes a Time object with the given minutes (default 0) and hours
(default 0).

Precondition:  minutes is an int between 0 and 59.  hours is an int >= 0."""
```

As shown in class, this constructor should support default values for minutes and hours. In other words, the function call `Time()` should successfully create a time object with 0 minutes and 0 hours. Remember to use the "'self" construct! (Otherwise, you will have statements like "minutes = minutes", and what do you suppose that does?)

Note that the attribute invariants have become the precondition for the constructor. As part of your implementation you should **add assert statements to check that the precondition is satisfied**. You only need to check `minutes` and `hours`; you never need to check the `self` parameter in a method.

The `__str__` Method. Right now, the function `str()` does not work properly on objects of this class. It shows the memory location of the ¡code¿Time¡/code¿ object, but not its contents. To remedy this situation, you need to implement the `__str__` method. This method should satisfy the following specification.

```
"""Returns:  the Time object formatted as a string.

The contents are formatted 'hours:minutes', where minutes has been padded (with
an initial '0') if necessary to ensure that it is always two characters."""
```

The body of this method will look very familiar after Assignment 3. It is similar to all of the `to_string` functions in that assignment. However, you do not need an assert statement this time, as the only parameter is `self`, and you never need to check the `self` parameter in a method.

Testing it Out. At this point, you will probably want one last test that your class works. We assume that you have saved your class in a file called `time.py`. Start Python with the interactive prompt and type the following commands:

```
>>> from time import Time
>>> t = Time(5,2)
>>> str(t)
```

If you get the string '2:05', then your class is likely working correctly.

Writing it All Down. We assume that you have been testing out your class in Python module. When you are done, you should write your code in the space on the next page. **You do not need to include the specifications**.

When you have done this, show your work to you instructor and you are done.

```
class Time(object):
```