Last Name: _____ First: _____ Netid: _____ Section _____

CS 1110 Prelim 2 November 13th, 2014

This 90-minute exam has 5 questions worth a total of 100 points. Scan the whole test before starting. Budget your time wisely. Use the back of the pages if you need more space. You may tear the pages apart; we have a stapler at the front of the room.

It is a violation of the Academic Integrity Code to look at any exam other than your own, look at any reference material, or otherwise give or receive unauthorized help.

You will be expected to write Python code on this exam. We recommend that you draw vertical lines to make your indentation clear, as follows:

```
def foo():
    if something:
        do something
        do more things
    do something last
```

You should not use while-loops on this exam. Beyond that, you may use any Python feature that you have learned about in class (if-statements, try-except, lists, for-loops, recursion and so on).

Question	Points	Score
1	2	
2	30	
3	21	
4	24	
5	23	
Total:	100	

The Important First Question:

1. [2 points] Write your last name, first name, netid, and *lab section* at the top of each page. If you cannot remember the section number, please write down your lab's time and place.

Throughout this exam you will make use of strings, lists, and dictionaries. You are expected to understand how slicing works. In addition, the following functions and methods may be useful:

String Functions and Methods

Function or Method	Description
len(s)	**Returns**: number of characters in `s`; it can be 0.
s.find(s1)	**Returns**: index of the first character of the FIRST occurrence of `s1` in `s` (-1 if `s1` does not occur in s).

List Functions and Methods

Function or Method	Description
len(x)	**Returns**: number of elements in list `x`; it can be 0.
y in x	**Returns**: True if `y` is in list `x`; False otherwise.
x.index(y)	**Returns**: index of the FIRST occurrence of `y` in `x` (an error occurs if `y` does not occur in x).
x.append()	Adds `y` to the end of list `x`.
x.insert(i,y)	Inserts `y` at position `i` in list `x`. Elements after position `i` are shifted to the right.

Dictionary Functions and Methods

Function or Method	Description
len(d)	**Returns**: number of keys in dictionary `d`; it can be 0.
y in d	**Returns**: True if `y` is a key `d`; False otherwise.
d.keys()	**Returns**: a list containing all the keys in `d`.
d.values()	**Returns**: a list containing all the values in `d`. It may have duplicates.

2. [30 points] **Classes and Subclasses**

The next two pages have skeletons of two classes: `Time` and `Event`. `Event` is a subclass of `Time`, while `Time` is only a subclass of `object`. You are to

1. Fill in the missing information in each class header.
2. Add getters and setters as approporiate for the instance attributes
3. Fill in the parameters of each method (beyond the getters and setters).
4. Implement each method according to its specification.
5. Enforce any preconditions in these methods using asserts

Method `__str__` in `Time` is already completed, but you must finish the header for this method. All other methods are incomplete. Pay close attention specifications. There are no parameters beyond the ones specified. If a parameter has a default value, it is listed in the specification.

There are no headers for getters or setters. You are to write those from scratch. **You do not need to write specifications for the getters and setters**. **Do not write a setter if the class specification forbids it**. Do not use `type` when writing your asserts.

```
class Time(                          ):              # Fill in missing part
    """Instance represents a time period in hours and minutes.
    MUTABLE ATTRIBUTES
        _hours [int in 0..23]:  An hour of the day
        _mins [int in 0..59]:  A minute of the hour"""
    # PUT GETTERS/SETTERS HERE AS APPROPRIATE. SPECIFICATIONS NOT NEEDED.
```

```
    # INITIALIZER
    def __init__(                                  ):         # Fill in parameters
        """Initialize a Time instance with given hours and mins.
        Precondition: hours is an int in 0..23.  mins is an int in 0..59."""
```

```
    def __str__(                                   ):         # Fill in parameters
        """Returns: String representation of time in format 'hrs:mins'.
        THIS METHOD IS ALREADY COMPLETED. JUST FILL IN THE PARAMETERS."""
        hrs = str(self._hours) if self._hours > 9 else '0'+str(self._hours)
        mns = str(self._mins) if self._mins > 9 else '0'+str(self._mins)
        return hrs+':'+mns

    def shift(                                     ):           # Fill in parameters
        """Shift time ahead by given amount mins.  Hours over 23 ‘‘wrap around’’.
        Example: 20 mins shifts 23:50 to 00:10
        Precondition: mins is an int >= 0."""
```

```
class Event(                              ):                # Fill in missing part
    """Instance is an event at a specific time.  It has Time's attributes plus
    IMMUTABLE INSTANCE ATTRIBUTE:
        _name [nonempty string]:  Name of this event"""

    # PUT GETTERS/SETTERS HERE AS APPROPRIATE. SPECIFICATIONS NOT NEEDED.









    # INITIALIZER
    def __init__(                                      ):  # Fill in parameters
        """Initializes a new event with the given name and time.
        Parameters hours and mins are optional.  By default, an event is at noon.
        Precondition: name is a nonempty string.  hours is an int in 0..23.
        mins is an int in 0..59"""









    def __str__(                                     ):         # Fill in parameters
        """Returns:  string representation of this event.
        Format is 'name (time)'.  Example:  'Lunch (12:00)'
        YOUR SOLUTION MUST FIT ON ONE LINE."""
```

3. [21 points total] **Iteration**.

   The functions specified on the next page take dictionaries as one of their arguments. The dictionaries associated grades with net-ids, as shown below.

   example = { 'qsq4' : 80, 'dju6' : 90, 'wgo5' : 75, 'zkw9' : 60, 'wj8': 74 }

   Use for-loops to implement both of these functions. You may wish to refer to the dictionary functions on the second page. You **do not need to enforce** the function preconditions.

(a) [8 points]

```
def average(grades):
    """Returns: the average of all grades in the dictionary as a float.
    Example: average(example) is 75.8; average({}) is 0.0
    Precondition: grades is a grade dictionary (may be empty)"""
```

(b) [13 points]

```
def histogram(grades,acutoff,bcutoff):
    """Returns: A histogram (dictionary) with the number of As, Bs, and Cs.
    The letter grades are the dictionary keys, and the values are the number of
    each grade.  Any grade >= acutoff is an A. Any other grade >= bcutoff is a B.
    All other grades are Cs.
    Example: histogram(example,90,75) returns {'A':1, 'B':2, 'C':2}
             histogram(example,80,60) returns {'A':2, 'B':3, 'C':0}
             histogram({},80,60) returns {'A':0, 'B':0, 'C':0}
    HINT: The accumulator is a dictionary, but the initial value is NOT {}.
    Precondition: grades is a grade dictionary (may be empty).  acutoff
    and bcutoff are ints or floats with 0 < bcutoff < acutoff."""
```

4. [24 points total] **Recursion**.

   Use recursion to implement the functions specified below; **do not use loops**. You may not use any helpers functions other than those listed on the tables on the second page (e.g. you cannot use `count()`). You do not need to enforce preconditions.

   (a) [12 points]

```
def oddsevens(thelist):
    """Returns: copy of the list with odds at front, evens in the back.
    Odd numbers are in the same order as thelist. Evens are reversed.
    Example: oddsevens([1,2,3,4,5,6]) returns [1,3,5,6,4,2].
    Precondition: thelist is a list of ints (may be empty)"""
```

   (b) [12 points]

```
def histogram(s):
    """Return: a histogram (dictionary) of the # of letters in string s.
    The letters in s are keys, and the count of each letter is the value. If
    the letter is not in s, then there is NO KEY for it in the histogram.
    Example: histogram('') returns {}, while histogram('abracadabra')
    returns {'a':5,'b':2,'c':1,'d':1,'r':2}
    Precondition: s is a string (possibly empty)."""
```

5. [23 points total] **Folders and Name Resolution**

Consider the two (undocumented) classes below.

```
class A(object):
    x = 10
    def __init__(self,y):
        self.y = y

    def f(self,x):
        return x*self.y

    def g(self):
        return self.x+self.y
```

```
class B(A):
    x = 20
    def __init__(self,x,y):
        A.__init__(self,x)
        self.z = y

    def f(self):
        return self.z

    def h(self):
        return self.x+self.z
```

(a) [8 points] Execute the following in the interactive shell:

```
>>> a = A(2)
>>> b = B(3,4)
```

The following expressions are either valid or the produce an error. Indicate which is which. If the expression is valid, tell us what it evaluates to.

i. `a.x`

ii. `b.y`

iii. `B.z`

iv. `a.f(5)`

v. `b.f(5)`

vi. `A.f(b,5)`

vii. `b.g()`

viii. `a.h()`

(b) [15 points] On the next page, diagram the evolution of the assignment statement

```
>>> b = B(3,4)
```

You will need **seven diagrams**. In each diagram, draw the call stack, global space and heap space. If the contents of any space are unchanged between diagrams, you may write *unchanged*. You do **not** need to draw the class folders in heap space.

Answer Question 5(b) Here