Last Name: ______________________ First Name: ________________ Cornell NetID, all caps: ________

Circle your lab: Tu 12:20 Tu 1:25 Tu 2:30 Tu 3:35 W 12:20 W 1:25 W 2:30 W 3:35

CS 1110 Prelim 1 March 7th, 2013

It is a violation of the Academic Integrity Code to look at any exam other than your own, to look at any other reference material, or to otherwise give or receive unauthorized help. We also ask that you not discuss this exam with students who are scheduled to take a later makeup.
Academic Integrity is expected of all students of Cornell University at all times, whether in the presence or absence of members of the faculty. Understanding this, I declare I shall not give, use or receive unauthorized aid in this examination.

Signature: ______________________________ Date ____________

This 90-minute exam has 6 questions worth a total of 66 points. When permitted to begin, 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.
You may not use explicit for-loops or recursion on this exam. Beyond that, you may use any Python feature that you have learned about in class (if-statements, map, lists, and so on), *except:* please use `str` instead of backquotes (handwritten backquotes often look like single quotes).

Question	Points	Score
1	2	
2	16	
3	16	
4	4	
5	20	
6	8	
Total:	66	

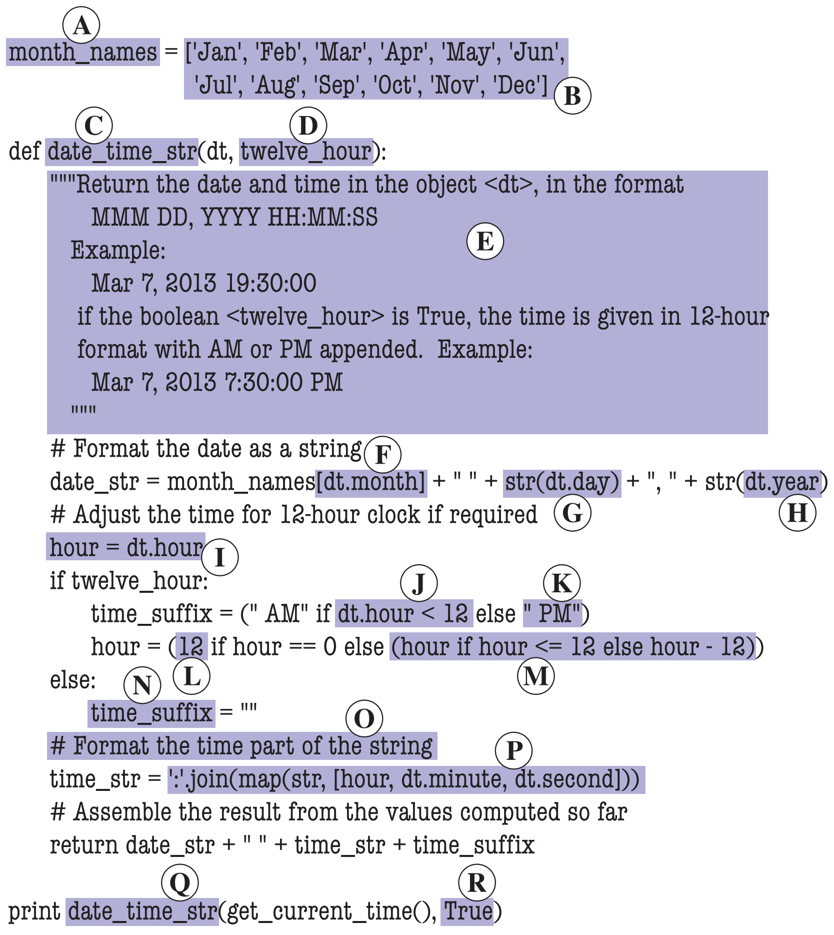
The Important First Question:

1. [2 points] When allowed to begin, write your last name, first name, and Cornell NetID at the top of each page, and circle your lab time on the top of this page.

Last Name: ______________________ First Name: ______________ Cornell NetID: ______

2. [16 points] Match the shaded parts of the following Python program to the names below. In your answer, each letter should occur exactly once.

____	Assignment statement	____	List indexing
____	Name of a function being called	____	Function call expression
____	Name of a function being defined	____	Method call expression
____	Boolean expression	____	Docstring
____	Parameter	____	Comment
____	Argument	____	Conditional expression
____	String literal	____	Name of global variable being created
____	Integer literal	____	Name of local variable being created
____	List	____	Reference to an attribute of an object



```python
month_names = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun',
               'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec']

def date_time_str(dt, twelve_hour):
    """Return the date and time in the object <dt>, in the format
        MMM DD, YYYY HH:MM:SS
     Example:
        Mar 7, 2013 19:30:00
      if the boolean <twelve_hour> is True, the time is given in 12-hour
      format with AM or PM appended.  Example:
        Mar 7, 2013 7:30:00 PM
     """
    # Format the date as a string
    date_str = month_names[dt.month] + " " + str(dt.day) + ", " + str(dt.year)
    # Adjust the time for 12-hour clock if required
    hour = dt.hour
    if twelve_hour:
        time_suffix = (" AM" if dt.hour < 12 else " PM")
        hour = (12 if hour == 0 else (hour if hour <= 12 else hour - 12))
    else:
        time_suffix = ""
    # Format the time part of the string
    time_str = ':'.join(map(str, [hour, dt.minute, dt.second]))
    # Assemble the result from the values computed so far
    return date_str + " " + time_str + time_suffix

print date_time_str(get_current_time(), True)
```

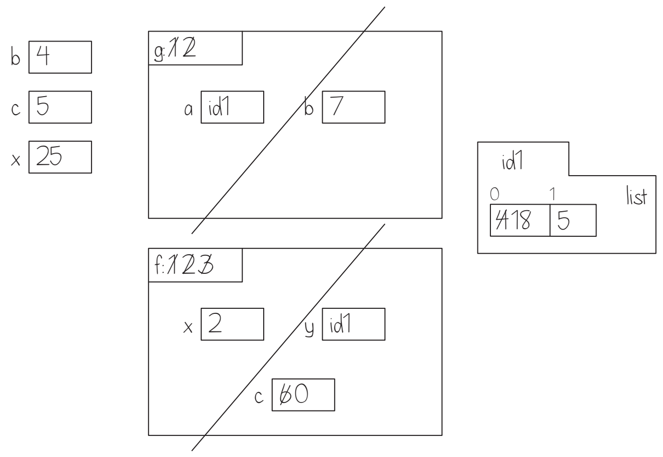
Last Name: ______________________ First Name: ______________________ Cornell NetID: __________

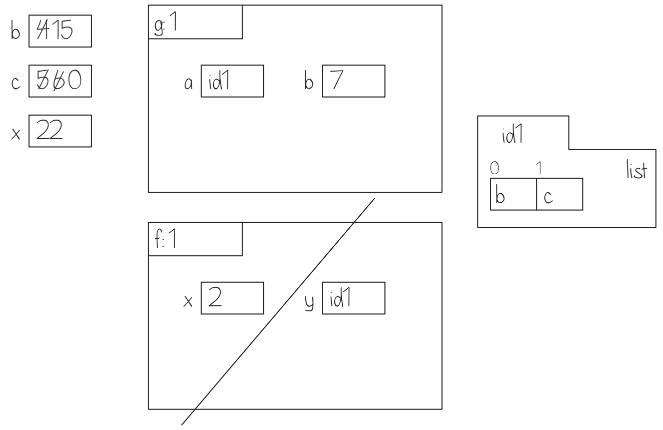
3. [16 points] Two students were assigned to diagram the execution of the following code. You are their grader; please circle all errors and write in anything that is missing. You may wish to do this question by first drawing the relevant frames and objects yourself.

```
def f(x, y):
    c = 3*x
    y[0] = b + c + y[1]
    c = 0

def g(a, b):
    f(2, a)
    return b + a[0]

b = 4
c = 5
x = g([b,c], 7)
```





4. [4 points] Here, we consider a simplified version of extracting information from a web page. Assume that variable `x` stores a string of the form

   `<a href="`<u>string1</u>`">`<u>string2</u>`</a>`

   where both <u>string1</u> and <u>string2</u> are strings that do not contain double quotes or angle brackets. The only space in the format shown above is after the first `a`, although <u>string1</u> and <u>string2</u> may themselves contain spaces. Example: if `x` were the string `'<a href=" this "> that</a>'`, then <u>string1</u> would be `' this '` and <u>string2</u> would be `' that'`.

   Write a sequence of one or more statements that result in variable `s2` holding the string <u>string2</u>.

For reference:

`s.find(s1)`	Returns: index of the first character of the FIRST occurrence of `s1` in `s`, or -1 if `s1` does not occur in `s`.
`s.find(s1, i)`	Returns: index of the first character of the FIRST occurrence of `s1` in `s` at or after position `i`, or -1 if `s1` does not occur in `s[i:]`. If `i` is omitted, searches the whole string.
`s.index(s1)`	Like find, but raises an error if `s1` is not found.
`s.index(s1, i)`	Like find, but raises an error if `s1` is not found.
`s.rfind(s1)`	Returns: index of the first character of the LAST occurrence of `s1` in `s`, or -1 if `s1` does not occur in `s`.
`s.rindex(s1)`	Like rfind, but raises an error if `s1` is not found.

5. This question involves code for suggesting new NetIDs.

   Assume file `last.py` defines a type of object called `LastUsed`. These have two attributes:

`prefix`	non-empty string of lowercase letters
`suffix`	positive `int`

   and can be created by calls like this: `last.LastUsed('djs', 98)` (if `last` has been imported).

   File `last.py` also implements the function `ind(lulist, p)` with the following spec:

```
def ind(lulist, p):
   """Returns: index in lulist of LastUsed object with prefix p (-1 if no such object)

   Preconds: lulist is a (possibly empty) list of LastUsed objects with distinct
   prefixes. p is a non-empty string of lowercase letters."""
```

   (a) [8 points] Draw all objects and variables created by the following sequence of commands. (Don't draw any frames.)

```
import last
temp = [last.LastUsed('ljl', 2), last.LastUsed('srm',2)]
has_srm = last.ind(temp, 'srm')
```

   (b) [12 points] On the next page(s), complete file `nets.py` **by following the helpful directions given in curly braces**. Each such direction can require multiple lines to implement. For reference, here are some functions and the like you can use:

`x in lt`	Returns: `True` if `x` is in list `lt`, `False` otherwise.
`lt.append(x)`	Append object `x` to the end of list `lt`.
`lt.pop(i)`	Returns: item at position `i` in list `lt`, removing it from `lt`. If `i` is omitted, returns and removes the last item.
`lt.sort()`	Sort the items of `lt`, in place (the list is altered).

```
# nets.py {Omit other authoring info.}
{Add any necessary import statements here.}




def newid(fname, mname, lname, all_last):
    """Returns: NetID for new Cornellian named fname mname lname.  For
       people with the same initials, gives out sequentially numbered
       NetIDs starting with the number 1.

       The new NetID is a string of this person's initials (first
       initial coming first) and the next available numerical suffix,
       according to all_last.

       The list all_last keeps track of which NetIDs have been used; it
       contains a LastUsed object for each set of initials that has
       been used in a NetID, with the highest number that has been given
       out so far.  It is modified to account for the new NetID
       returned by this function.
       {Don't worry, we explain how to do this in the remarks below.}

       For instance, if all_last started out empty, and then the NetIDs
       abc1, foo1, and abc2 are generated, all_last should contain
       two LastUsed objects: one with prefix 'abc' and suffix 2, and one
       with prefix 'foo' and suffix 1.

       Preconditions: all arguments are strings containing only
           lowercase letters.  The lengths of fname and lname are at
           least 1.  The list all_last contains LastUsed objects
           indicating which NetIDs have been used.
    """

    {Store in variable inits the initials of this Cornellian.}
```

The file `nets.py` *continues on next page...*

{If `all_last` contains a LastUsed object with prefix equal to `inits`, then add one to that object's suffix and assign to variable `suf` this new suffix value. Otherwise, add to `all_last` a new LastUsed object with prefix `inits` and suffix 1, and assign to variable `suf` this new suffix value.}

{Return the NetID corresponding to `inits` and `suf`}

6. [8 points] Complete the body of testing procedure `testnew` for function `newid` from the previous problem.[1] You may make at most five calls to `newid`. Our grading will focus on the completeness of your test cases: they should cover the space of possible arguments with which `newid` could be called. To save time on this exam, do *not* directly check whether the argument list has been correctly modified; only directly check whether `newid`'s return value is correct.

```
import last
import nets
import cunittest2

def testnew():
    """Test the newid fn in nets"""
```

For reference, here are some functions in `cunittest2`:

`assert_equals(expected, received)`	Raise an AssertionError if `expected` and `received` differ.
`assert_true(received)`	Raise an AssertionError if `received` is False.

[1]Yes, for this exam we're doing the testing after the implementation. Tsk, tsk.