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

# CS 1110 Prelim 1 October 17th, 2013

This 90-minute exam has 6 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, to look at any other reference material, or to 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 recursion 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 and so on).

| Question | Points | Score |
|---|---|---|
| 1 | 2 | |
| 2 | 20 | |
| 3 | 20 | |
| 4 | 20 | |
| 5 | 20 | |
| 6 | 18 | |
| Total: | 100 | |

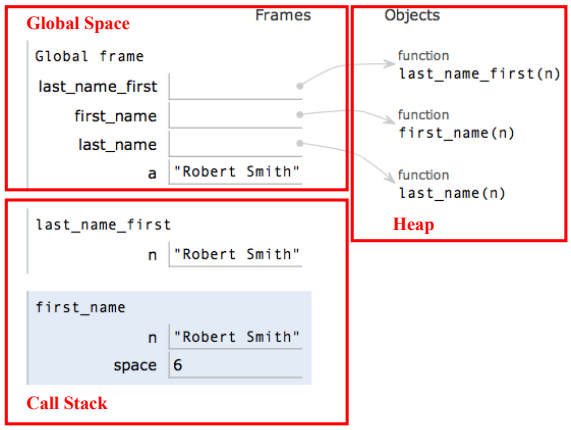
**The Important First Question:**

1. [2 points] Write your last name, first name, netid, and *lab section* at the top of each page.

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

2. [20 points total] **Terminology**.
The screenshot below is taken from the Python Tutor.

```
1  def last_name_first(n): Parameter
2      """Returns: n as 'last-name, first-name'
3      Precondition: n has exactly one space"""
4      assert n.count(' ') == 1
5  Local first = first_name(n)
6  Local last  = last_name(n)
7      return last+', '+first
8
9  def first_name(n): Parameter
10 Local space = n.find(' ')
11     return n[:space]
12
13 def last_name(n): Parameter
14 Local space = n.find(' ')
15     return n[space+1:]
16
17 a = 'Robert Smith'
18 b = last_name_first(a)
```




(a) [5 points] Circle and clearly label global space, heap space, and the call stack.
See above.

(b) [6 points] Identify and clearly label all of the parameters in the code above and to the left (do not worry about the frames and folders on the right). You can circle or underline them, but you must write "parameter" next to each. Do the same for local variables.
See above.

(c) [5 points] What is the difference between a function definition and a function call? Give an example of each from the picture above.
A function definiton is code that describes what a function should do when it is executed (but it does not execute that code. It starts with the keyword def and a function header. Lines 1-7 above are an example of a function definition. A function call is an expression that instructs Python to execute the function. Line 18 has an example of a function call: `last_name_first(a)`.

(d) [4 points] What is the purpose of the `assert` statement on line 4? How does Python execute this line?
The purpose of assert statement is to "enforce the precondition". When executing this line, Python checks whether the boolean (in this case, the precondition) `n.count(' ') == 1` is True. If it is, it does nothing. It is False, it causes and error, with the string after the comma as the error message.

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

3. [20 points total] **Objects and Functions**.
Assignment 3 introduced RGB objects. They had three attributes – `red`, `green`, and `blue` – which all had the invariant that they must be an `int` between 0 and 255 (inclusive). The constructor for an RGB object is the function `RGB(r,g,b)` where `r`,`g`,`b` are the sttribute values. (you do not need to worry about importing `colormodel` for this problem).

   (a) [10 points] Greyscale is an integer between 0 and 255, given by the formula

$$\texttt{grey} = 0.21\texttt{R} + 0.71\texttt{G} + 0.07\texttt{B}$$

   We often store grey colors as an RGB object where `red`, `green`, and `blue` are all equal (to the grey value). With this in mind, implement the function below.

```
def rgb_to_grey(rgb):
    """Returns: a new RGB object that is the greyscale version of rgb

    All three attributes should be equal to the greyscale value.

    Precondition: rgb is an RGB object"""
    # Compute grey and round it to nearest int
    grey = 0.21*rgb.red+0.71*rgb.green+0.07*rgb.blue
    grey = int(round(grey))

    # Put in range and return object
    grey = min(grey,255) # Formula guarantees cannot happen, but in case.
    return RGB(grey,grey,grey)
```

   (b) [10 points] The functions in Assignment 3 created new colors. The procedure below is different; it changes the contents of an RGB object. Implement it.

```
def red_tint(rgb,amt):
    """Alter the red attribute of rgb by amt.

    Value amt is a float between -1 and 1, representing a percentage.
    If amt < 0, reduce the red by that percent (-1 reduces red to 0).
    If amt > 0, increase by that percent (1 doubles red, to max of 255).
    This procedure modifies rgb and does not return anything.

    Example:  If amt = -0.25, a red value of 128 becomes 96.
    If amt = 0.5, a red value of 128 becomes 192

    Precondition: rgb an RGB object, amt a float between -1 and 1"""
    # Compute "raw" new red value
    red_adjust = amt*rgb.red
    new_red = rgb.red+red_adjust
    new_red = int(round(new_red))

    # Make sure it is in range and assign
    new_red = min(255,new_red) # Do not worry about going negative
    rgb.red = new_red
```

4. [20 points] **String Slicing**.
Implement the function below, according to its specification. You **do not need to use a for-loop to implement this function**, but you will need to use string slicing and string methods. You might find the following functions and/or methods useful.

| Function or Method | Description |
|---|---|
| len(s) | **Returns**: number of characters in `s`; it can be 0. |
| s.isupper() | **Returns**: True if all letters in `s` are upper case, False otherwise. |
| s.upper() | **Returns**: a copy of `s` with all letters upper case. |
| s.islower() | **Returns**: True if all letters in `s` are lower case, False otherwise. |
| s.lower() | **Returns**: a copy of `s` with all letters lower case. |
| 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). |
| s.rfind(s1) | **Returns**: index of the first character of the LAST occurrence of `s1` in `s` (-1 if `s1` does not occur in s). |

**Hint**: Methods can be applied to a single letter.

```
def unify_first_word(s):
    """Return: copy of s with the letters of the first word all the same case

    The first word of s is everything up to the first space (if it exists). If
    the first letter of s is lower case, the first word becomes lower case. If
    the first letter is upper case, the first word becomes upper case.

    Examples: 'Boo' becomes 'BOO', 'Hello World' becomes 'HELLO World', and
    'aLL Good' becomes 'all Good'

    Precondition: str s has only letters and spaces. It may have no spaces.  It
    starts with a letter.
    # Find and extract first word
    pos = s.find(' ')
    if (pos == -1):
        word = s
    else:
        word = s[:pos]

    # Find if first letter is capitalized, and do right thing
    iscap = s[0].isupper()
    if iscap:
        word = word.upper()
    else:
        word = word.lower()

    # Compute the result
    if pos == -1:
        return word
    return word+s[pos:]
```

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

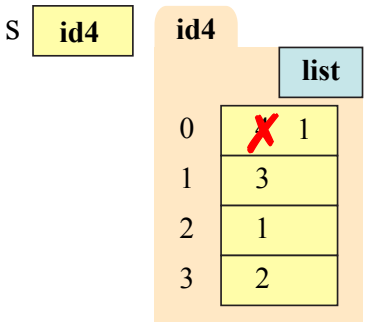
5. [20 points total] **Call Frames.**

   (a) [12 points] The function call `bubble(s)`.

Suppose you have the following variable assignment:

```
s = [4,3,1,2]
```

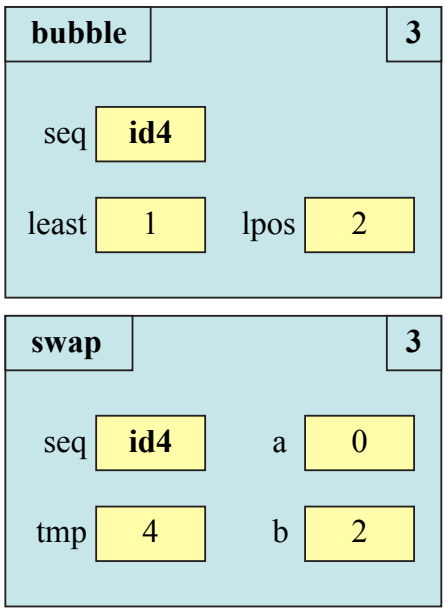
Because lists are mutable, we store the contents of this list in a folder in heap space, and then store the identifier of this folder in the variable `s`. If `s` is a global variable, the result of this assignment would be the illustration to the right (note that we are showing both global space and heap space in one picture).




For the call `bubble(s)`, draw the *entire call stack* when `bubble` is in the middle of executing line 3 and `swap` has just completed line 2. You do not need to draw what the two call frames look like before or after this time. In addition **if any changes were made in global space or heap space by this time, indicate them in the picture above**.

```
def bubble(seq):
    """Move least elt. to front"""
1   least = min(seq)
2   lpos  = seq.find(least)
3   swap(seq,0,lpos)
```

```
def swap(seq,a,b):
    """Swap pos. a and b"""
1   tmp = seq[a]
2   seq[a] = seq[b]
3   seq[b] = tmp
```

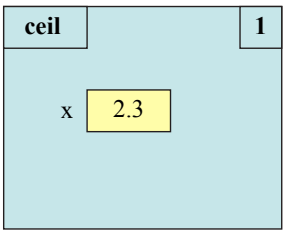


(b) [8 points] The function call `ceil(2.3)`.

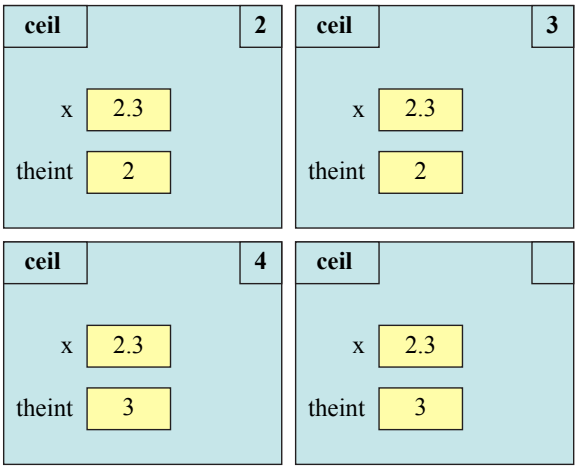
The function `ceil` below is the *ceiling function*. It returns the next integer after the number in `x`, or `x` if `x` is already an integer. Diagram **all steps of the function call** `ceil(2.3)`. For the step in which you erase the frame, simply draw a line through it (though this step should be a different picture than the last step of execution).

```
def ceil(x):
    """Least int >= x
    Pre:  x is a float"""
    1   theint = int(x)
    2   if theint < x:
    3       theint = theint+1
    4   return theint
```

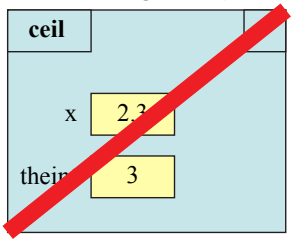
Initial Diagram (Frame Start):




Intermediate Diagrams (Execution):




Final Diagram (Frame Deletion):



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

6. [18 points total] **Testing and Debugging.**

(a) [6 points] The function defined below is buggy and does not work. There are (at least) two bugs in it. In order to find the bugs, we have added several watch statements throughout the code. The result of running the code with these watch statements is shown to the right. Using this information as a guide, identify and fix the two bugs.

```
def time_to_minutes(s):
    """Returns: min since midnight

    Examples:
       '2:45 PM' => 14*60+45 = 885
       '9:05 AM' => 9*60+5 = 545
      '12:00 AM' => 0

    Pre: s a str in 12-hour format;
         <hours>:<min> AM/PM"""
    # Find the separators
    pos1 = s.index(':')
    print 'pos1 is '+str(pos1)
    pos2 = s.index(' ')
    print 'pos2 is '+str(pos2)

    # Get hour and convert to int
    hour = s[:pos1]
    print 'hour is "'+hour+'"'
    hour = int(hour)
    print 'hour is '+str(hour)

    # Adjust hour to be correct.
    suff = s[pos2+1:]
    print 'suff is "'+str(suff)+'"'
    if (suff == 'PM'):
        hoar = hour+12
    elif (hour == 12):
        hour = 0
    print 'hour is '+str(hour)

    # Get min and convert to int
    mins = s[pos1:pos2]
    print 'mins is "'+mins+'"'
    mins = int(mins)
    print 'mins is '+str(mins)

    return hour*60+mins
```

```
>>> time_to_minutes('2:45 PM')
pos1 is 1
pos2 is 4
hour is "2"
hour is 2
suff is "PM"
hour is 2
mins is ":45"
Traceback (most recent call last):
   File "<stdin>", line 1, in <module>
   File "<stdin>", line 33, in
        time_to_minutes
ValueError: invalid literal for
int() with base 10: ':45'
```

**Error 1: change `hoar` to `hour`**

```
if (suff == 'PM'):
    hour = hour+12
```

**Error 2: fix `mins` assignment**

```
# Get min and convert to int
mins = s[pos1+1:pos2]
```

**Error 3 (unintended): 'noon' problem**

```
if (suff == 'PM' and hour < 12):
    hour = hour+12
```

(b) [12 points] We saw the `strip()` method in class. This method removes spaces from the beginning and end of a string, but does nothing to spaces in the middle. If we really wanted all spaces removed from a string, we would need a function like the following:

```
despace(s):
    """Returns: copy of s with all spaces removed.

    Example: 'Hello World' becomes 'HelloWorld'

    Precondition:  s is a string"""
```

**Do not implement this function**. We only want you to test it.
In the space below,provide at least five different test cases to verify that this function is working correctly. For each test case provide:

- The input, or function arguments.
- The expected output, or what the function should return.
- An explanation of why that test is important, and different from the others.

So that we can clearly see the spaces in your test cases, please use an underline (_) to represent each space in your string.

This problem was the most controversial during grading. However, we were fair, in the sense that everyone is graded the same way. Therefore, we are unlikely to give back any points on this problem during regrades (as we would have to regrade everyone).

The issue at hand is that we want "different test cases". It is not enough that the test cases have different inputs; they have to be *significantly different*. What constitutes "significantly" different depends on the function specification. For example, `'_Hello'` and `'Hello_'` are significantly different for `before_space` and `after_space`, because the position of the space matters for those functions. Space position does not matter (as much) for `despace`, and so they are not different test cases. The ability to understand this difference is what we were testing here.

Below are the different solutions we were thinking of. If you had (at least) five test cases that matched all of the below, you got full credit. Otherwise, we checked if your test cases were different enough, and subtracted 2 points for each less than 5 (for a maximum of -6).

| Input | Output | Reason |
|---|---|---|
| `'abc'` | `'abc'` | No spaces |
| `'_ _'` | `''` | Spaces only |
| `'Hello_World'` | `'HelloWorld'` | Single space |
| `'Hello_ _ _World'` | `'HelloWorld'` | Multiple adjacent spaces |
| `'See_spot_run'` | `'Seespotrun'` | Multiple, non-adjacent spaces |