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

CS 1110 Final, Spring 2016

This 150-minute exam has 9 questions worth a total of 63 points. When permitted to begin, scan the whole test before starting. Budget your time wisely.

When asked to write Python code on this exam, you may use any Python feature that you have learned about in class.
There is a reference list of specifications for some common functions on the reverse of this page.

Unless otherwise stated, you may write helper functions when asked to write code, but include specifications for them in their doc strings.

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 ____________

Question	Points	Score
1	8	
2	5	
3	5	
4	17	
5	3	
6	6	
7	7	
8	6	
9	6	
Total:	63	

For your reference:

`int(s)`	Returns: the `int` that `s` represents. OK if `s` is "006"; error if `s` is "5.0".
`s[i:j]`	Returns: A new string `s[i] s[i+1]` ... `s[j-1]` under ordinary circumstances. Returns `''` if `i` $\geq$ `len(s)` or `i` $\geq$ `j`.
`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.index(s1)`	Like find, but raises an error if `s1` is not found.
`s.lower()`	Returns: a copy of `s` with all letters in it converted to lowercase.
`s.count(s1)`	Returns: the number of non-overlapping appearances of string `s1` in string `s`.
`x in y`	Returns: `True` if `x` is in list or string `y`, `False` otherwise.
`lt[i:j]`	Returns: A new list`[lt[i], lt[i+1], ..., lt[j-1]]` under ordinary circumstances. Returns `[]` if `i` $\geq$ `len(lt)` or `i` $\geq$ `j`.
`lt.index(item)`	Returns: index of first occurrence of `item` in list `lt`; raises an error if `item` is not found.
`lt.count(thing)`	Returns: the number of occurrences of item `thing` in list `lt`.
`range(n)`	Returns: the list `[0, 1, 2, ..., n-1]`
`range(a, b, step)`	Returns: the list `[a, a+step, a+2*step, ...` up to but not including `b`.
`lt.append(x)`	Append object `x` to the end of list `lt`.
`lt1.extend(lt2)`	Concatenate the list `lt2` to the end of list `lt1`.
`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).

1. (a) [3 points] Assume that the variables `B1` and `B2` are initialized and boolean-valued. Give a Boolean-valued expression that is `True` if and only if *exactly* one of `B1` and `B2` is `True` (i.e., one is `True` and the other is `False`). The expression should be `False` otherwise.

   (b) [1 point] What is the value of `6*float(10/6)/10`?

   (c) [4 points] Consider the following code:

```
x = [10,20]
y = [30,40]
temp = x
x = y
y = temp
print x[0], x[1]
print y[0], y[1]
print temp[0], temp[1]
z = x[0]
print z
```

   What is the output? (If an error results, describe what the error is).

2. [5 points] Assume that `x` is a list of `int` values and that it has even length. We say that `y` is the *even-odd swap* of `x` if it has the same length as `x` and for all valid indices `k` that are even,

```
y[k]==x[k+1] and y[k+1]==x[k]
```

is `True`. Thus, if

```
x = [30,50,70,90,60,40]
```

then

```
[50,30,90,70,40,60]
```

is the even-odd swap of `x`. Complete the following function so that it performs as specified.

```
def EvenOddSwap(x):
    """ Returns a list that is the even-odd swap of x. Does not change x.
    PreC: x is a list with int values and its length is even and non-zero."""
```

3. [5 points] A *time stamp string* is a length-5 string of the form `'xx:yy'` where the first two characters encode the hours,

```
'00'  '01'  '02'    ...   '22'  '23'
```

and the last two characters encode the minutes,

```
'00'  '01'  '02'    ...   '58'  '59'
```

Complete the following function so that it performs as specified.

```
def NewDay(t,additional_minutes):
    """ Returns True if current time plus additional_minutes occurs on a
    different day as the current time.

    PreC: t is a time stamp string that represents the current time.
    additional_minutes is a positive int that represents an elapsed time in minutes.
    """
```

Hint: The number of minutes in a day is 60*24.

Examples:

`NewDay("01:13", 1500)` is `True`
`NewDay("01:13", 42)` is `False`
`NewDay("23:55", 5)` is `True`
`NewDay("23:55", 4)` is `False`

4. This problem is about determining whether or not a request for a make-up final exam is legitimate at a Certain University (CU) where final exam "slots" are consecutively indexed. Assume the availability of the following two functions:

```
def time_to_slot():
    """ Returns a dictionary whose keys are strings and whose values are ints.
    Each key encodes an exam time and the corresponding value is its exam slot index.
    The length n of the returned dictionary equals the total number of exam slots and
    these are indexed from 1 to n."""

def course_to_slot():
    """ Returns a dictionary whose keys are strings and whose values are ints.
    Each key encodes a course name and the corresponding value is its exam slot index.
    The length of the returned dictionary equals the total number of courses that
    have final exams."""
```

Here is an example of a dictionary that could be returned by `time_to_slot`:

```
{"5/18 7pm":1, "5/19 9am":2, "5/19 2pm":3, "5/19 7pm":4, "5/20 9am":5}
```

If `s in time_to_slot()` is `True`, then we say `s` is a *valid* exam-period string.

Here is an example of a dictionary that could be returned by `class_to_slot`:

```
{"CS1110":1, "MATH1920":2, "ENGL2800":5, "HADM4300":3, "IS6000":2, "ILR2100":4}
```

If `s in course_to_slot()` is `True`, then we say `s` is a *valid* course-with-final string.

(a) [5 points] Implement the following function so that it performs as specified.

```
def valid(p_item):
    """Returns True if p_item[0] is a valid class-with-final string,
    p_item[1] is a valid exam-period string, and the exam period
    index associated with p_index[0] is the same as the exam period
    index associated with p_index[1]. Returns False otherwise.

    PreC: p_item is a length-2 list of strings"""
```

(b) [12 points] A *petition list* is a list whose items are length-2 lists of strings, e.g.,

```
[["CS1110", "5/18 7pm"], ["HADM4300", "5/19 7pm"], ["CS1112", "5/18 7pm"]]
```

A petition list `P` is *valid* if `valid(P[k])` is `True` for all valid `k`.

A valid petition list `P` is *make-up free* if the exam period indices associated with the `P[k]` are distinct and no three of them are consecutive.

To illustrate, suppose `P` is a valid length-6 petition list and that

```
x = [10,8,7,4,6,1]
```

is a list of `ints` with the property that `x[k]` is the exam period index associated with `P[k]`. In this case `P` would not be make-up free because we can find a consecutive triple: 6,7,8. It is easy to look for repeats and consecutive triples if `x` is sorted:

```
x = [1,4,6,7,8,10]
```

Complete the following function so that it performs as specified:

```
def isMakeUpFree(P):
    """ Returns True if P is a valid Petition list that is make-up free.
    Returns False otherwise. Does not alter P

    PreC: P is a petition list"""
```

You *must* make effective use of function `valid` from the previous page. (Assume it's correct.)

(You can use part of the next page if you need more space.)

(More space.)

5. [3 points] When you write an integer x in base-10 notation, the third digit from the right is the hundreds place. (If $x < 100$ then the hundreds place digit is zero.) Here are some examples:

Integer	The Hundreds-Place Digit
623	6
9892	8
7092	0
23	0
6	0

Complete the following function so that it performs as specified:

```
def hundreds_digit(x):
   """ Returns an int that is the value of the hundreds place (0 if x<100).

   PreC: x is a positive int
   """
```

6. Assume the availability of a class `Point` with float attributes `x` and `y` for the x and y coordinates, respectively, and the following methods:

```
def __init__(self,x,y):
    """ Creates a Point."""
def Dist(self,other):
    """ Returns a float that is the distance from self to Point other."""
def RandomNeighbor(self):
    """ Returns a random Point that has distance <= 2 from self"""
```

   (a) [2 points] The following code simulates a toy robot that starts at (0,0) and randomly "hops" from point to point in the plane.

```
P = Point(0,0)
Z = P
t = 0
while P.Dist(Z)<=100 and t<100000:
   P = P.RandomNeighbor()
   t+=1
```

   Is it possible that just after the above code finishes, `t` is less than 100000? Explain your answer in 1-3 sentences.

   (b) [4 points] Suppose when the robot makes a hop from point p to a point more than distance 1 away from p, your little sibling pulls the robot back to p. Write code that simulates this process for the robot starting at (0,0) and attempting 100 hops.

7. Consider the following class definition.

```
class Person(object):
  """name     this Person's name [non-empty string]
     fav      this Person's favorite Person [Person or None]"""

    def __init__(self, n):
        """Initialize a new Person with name n and fav set to None.
        (Nobody has a favorite person when they first come into existence.)

        PreC: n is a non-empty string."""
        self.name = n
        self.fav = None

    def make_fav(self, p):
        """Changes this Person's favorite person to p. (Doesn't return anything.)
        PreC: p is a Person or None".   """
```

(The implementation of make_fav is not important.) Note that the fav attribute of a Person is NOT a string.

(a) [4 points] Implement the following method for Person so that it performs as specified.

```
        def __str__(self):
            """Returns a string so that if p is a Person with a favorite person,
            then the string looks like this:

                      Lady Macbeth has favorite person Macbeth

              If there is no favorite person, then the string looks like this:

                      Macbeth has no favorite person """
```

(b) [3 points] Implement the following function so that it performs as specified. *Your implementation must make effective use of method* `make_fav`.

```
def make_couple(n1, n2):
    """Returns a list of two new Persons where the first one's name is n1,
    the second one's name is n2, the first Person's favorite Person is
    the second, and the second Person's favorite Person is the first.

    PreC: n1, n2 are non-empty strings."""
```

8. [6 points] Assume the existence of a function `expand` that takes a non-empty string `s` as input, and, if `s` has length `ell`, returns `s[0]*ell + s[1]*ell + s[2]*ell + ... + s[ell-1]*ell`. Examples:

   `expand('ab')` is `'aabb'`   `expand('abc')` is `'aaabbbccc'`   `expand('aa')` is `'aaaa'`

   Then, consider the following function, which is implemented *recursively*.

```
def spawn(s, n):
    """If n is 0, returns s. Otherwise, returns the result of applying function
    expand to s n times.  Example: spawn('ab',2) is 'aaaaaaaabbbbbbbb'.

    PreC: s is a non-empty string, n is a non-negative int."""
    if n == 0:
        return s
    else:

        --------------------------------------------------
```

   Here are three alternatives for the "else" line (assuming proper indentation):

   1. `return expand(spawn(s, n-1))`
   2. `return spawn(expand(s), n-1)`
   3. `return spawn(s, n-1) + expand(s)`

   State which (if any) alternatives constitute a correct solution and which (if any) don't.

   Furthermore, for each that is incorrect (if any), either (1) give an example `s` and `n` and the corresponding output of `spawn` showing it computes the wrong answer, or (2) if an error results, state what the error is.

9. [6 points] These classes were involved in Assignment 6:

```
class Speech(object):
    """
    attributes:
        theSpeaker       the name of the speaker [str]
        lines            each item is a (file) line in the speech [list of str]
    """

class Play(object):
    """
    attributes:
        theTitle        the name of the play [str]
        theSpeeches     a list of all the speeches in the play [list(Speech)]
        theScenes       a list of all the scenes in the play [list(Scene)]
        nLines          the total number of lines in the play [int]
    """
```

Implement the following function so it performs as specified.

```
def AllTheLines(P,A):
  """ Returns a list of strings each of which is a line from the play
  that is represented by P and each of which is spoken by the speaker
  whose name is A. All the lines spoken by A are encoded in the list that
  is returned.

  PreC: P is a play and A is a string
  """
```

We suggest you carefully re-read all instructions and specifications before turning this exam in.
Have a great summer!