Solution: CS 1110 Final Spring 2016

1. (a) [3 points] Assume that the variables $B_1$ and $B_2$ are initialized and boolean-valued. Give a Boolean-valued expression that is True if and only if exactly one of $B_1$ and $B_2$ is True (i.e., one is True and the other is False). The expression should be False otherwise.

Solution: Some possible solutions 

(B1 and not B2) or (B2 and not B1)
(B1 and not B2) or (not B1 and B2)
not(B1 and B2) and not(not(B1) and not(B2)),
not(B1 and B2) and (B1 or B2)
not (B1 == B2)
B1 != B2

(b) [1 point] What is the value of $6\times\text{float}(10/6)/10$? Solution: 0.6 (The grading is all or none. There must be a decimal point, so the type is clearly float. 6/10 is not an allowable answer.)

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

```python
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). Solution:

```
30 40 # +1
10 20 # +1
10 20 # +1 for being the same as previous line
30   # +1 for being the same as entry 0 of first line
```

Do not deduct points for extraneous printing (e.g., ”10, 20” or ”[10, 20]”)
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,

\[
y[k] == x[k+1] \quad \text{and} \quad y[k+1] == x[k]
\]

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.

```python
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.""

    Solution:
    y = []
k = 0
    while k < len(x):
        y.append(x[k+1])
        y.append(x[k])  # alternately. y.extend([x[k+1], x[k]])
        k+= 2  # note the increment by *two*
    return y
```

#alternate implementation
y = []
for k in range(len(x)-1):  #OK because only acts for even values
    if k%2 == 0:
        y.append(x[k+1])
        y.append(x[k])
    return y

#alternate implementation
y = []
for k in range(len(x)):
    if k%2 == 0:
        y.append(x[k+1])
    else:
        y.append(x[k-1])

Page 2
return y

# alternate implementation
y = list(x) # just want a new same-length list. copy/deepcopy ok even
    # without import of module
for k in range(0,len(x),2): # all the even indices
    y[k] = x[k+1]
y[k+1] = x[k]
return y

# Alternate implementation
m = len(x)/2
y = []
for k in range(m):
    y.append(x[2*k+1])
y.append(x[2*k])
return y

# +1 initialize and return new list (do not change x)
# +1 loop through potential indices (requires increment of while-loop variable )
# +1 proper syntax on append/extend
# +2 handle even and odd correctly

OK for mistakes like "2k" instead of 2*k", missing colos

We don't actually need the list to be non-empty, but didn't want students
to worry about empty inputs.
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.

```python
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.
    '''

    colon = t.index(':')  # actually, guaranteed that colon is 2.
    current_minutes = int(t[:colon]) * 60 + int(t[colon+1:])
    return current_minutes + additional_minutes >= 60 * 24
```

Solution:

```python
colon = t.index(':')  # actually, guaranteed that colon is 2.
current_minutes = int(t[:colon]) * 60 + int(t[colon+1:]))
return current_minutes + additional_minutes >= 60 * 24
```

# +1 for getting the colon position (might be hardcoded as 2)
# +2 getting the hours and getting the minutes strings (-1 if off by one in indexing)
# +1 remembering to convert these to ints
# +1 conceptually correct computation (don't take off a point if they are just
    off-by-1, <$ vs $<=$ kind of thing.)
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:

```python
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 `course_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.

```python
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""
```

Solution:

```python
DC = course_to_slot()
DT = time_to_slot()
if p_item[0] not in DC:
    return False
if p_item[1] not in DT:
    return False
return DC[p_item[0]] == DT[p_item[1]]
```

# ANYTHING THAT'S 'KIND OF MESSED UP' --> GIVE TO **MGMT** TO GRADE

# +1 correct use of "in" for both dictionaries (all or none)
# +1 syntactically correct check of being (not) in dictionaries (may be through if/else)
# -1 if omit parens from call (e.g. \verb+time_to_slot+ bad,
#    \verb+time_to_slot()+ good.
# +1 conceptually correct condition on being in class_to_slot and time_in_slot,
#    returning False if not.
# -1 (lose the point) if they check the wrong dictionary.
# -1 (lose the point) if check in dictionary AFTER checking for index equality
# (syntax errors already accounted for; no point if only checked one)
# +1 syntactically correct access of value in dictionary
# +1 conceptually checked equality of values, returning True/False as appropriate
(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.)

Solution:

DT = time_to_slot()
# Make sure P is valid...
for p in P:
    if not valid(p):
        return False

# Create sorted list of exam period slots (ints)
t = []
for p in P:
    t.append(DT[p[1]])  #if used course_to_slot, should have p[0]
t.sort()

# Are they Distinct?
for i in range(len(t)-1):
    if t[i] == t[i+1]:
        return False

# Check each possible sequence of three slots for consecutive-ness
for i in range(len(t) - 2):
    if t[i+2] == t[i+1] + 1 and t[i+1] == t[i] + 1:
        return False
    return True

#### grading
(You can use part of the next page if you need 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:

<table>
<thead>
<tr>
<th>Integer</th>
<th>The Hundreds-Place Digit</th>
</tr>
</thead>
<tbody>
<tr>
<td>623</td>
<td>6</td>
</tr>
<tr>
<td>9892</td>
<td>8</td>
</tr>
<tr>
<td>7092</td>
<td>0</td>
</tr>
<tr>
<td>23</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
</tr>
</tbody>
</table>

Complete the following function so that it performs as specified:

```python
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
    """

    Solution: One solution:

    \[
    y = x/100 \quad \# \text{ hundreds digit becomes the rightmost one}
    z = y \% 10 \quad \# \text{ get the last (rightmost) digit}
    \]

Alternate solution using string processing:
```
if x < 100:
    return 0
else:
    s = str(x)
    digit_char = s[len(s) - 3]
    return int(digit_char)

One-liner that is not so readable:
return int((str(x))[-3]) if x >= 100 else 0

Grading: +1 get exactly the third digit from the end
+1 return an int (probably a freebie if never convert to string)
(lose both points above for mixing up strings and ints incorrectly)
+1 handle case x < 100
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:

```python
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.

```python
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.

Solution: Z is always pointing to the point at (0,0), but since P is changing, `P.Dist(Z)` can increase beyond 100. Hence, `t` can be far less than 1000000, although it can’t be smaller than 100 or so.

+1: (perhaps implicitly) observing that Z and P do not stay aliases. +1: (perhaps implicitly) observing that this implies their distance may exceed 0.

-1: for saying incorrect things. No credit for no explanation.

(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.

Solution:

```python
P = Point(0,0)
for step in range(100):
    Q = P.RandomNeighbor()
    if P.Dist(Q) > 1:
        P = Q
```

# +1: proper init to (0,0)
# +1: correct call of RandomNeighbor
# +1: store new point in temporary new variable
# +1: correct assignment of new variable to P when appropriate
7. Consider the following class definition.

```python
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 ""

    Solution:
    if self.fav is None:  # OK if do "self.fav == None"
        return self.name + "has no favorite person"
    else:
        return self.name + " has favorite Person: " + self.fav.name

    # +1: if/else on self.fav being None
    # +1: uses self.fav.name correctly to get the name of the  favorite
    # +1 correct string concatenation
    # +1 return values set correctly

(b) [3 points] Implement the following function so that it performs as specified. Your implement-
    ation 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."""

    Solution:
    p1 = Person(n1); p2 = Person(n2)  # +1 (all or none for these two stmts)
    p1.make_fav(p2); p2.make_fav(p1)  # +1 (all or none for these two stmts)
    return [p1, p2]                  # +1

Note: they do not get the second point if they directly access the attributes. Note also
that one can’t do something like return [Person(n1), make_fav(p2)].
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.

```python
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 'aaaaaaabbbbbbbb'."
    if n == 0:
        return s
    else:
        return expand(spawn(s, n-1))
```

PreC: `s` is a non-empty string, `n` is a non-negative int.

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.

Solution: First one: works
Second one: works
Third one: doesn’t work: for `s='ab'` and `n=1`, produces 'abaabb'

+3 for first two lines (both must be correct to get any credit)
+1 for stating third line is wrong
+2 for giving a correct example of an incorrect output
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 
    """

    Solution:

    L = []
    for s in P.theSpeeches:
        if s.theSpeaker == A:
            for x in s.lines:  # get +2 for one-line L.extend(x),
                L.append(x)
    
    return L

Grading: 1 point per line (meaning there are six concepts, like "loop through" the speeches 
(although note use of "extend" should therefore get 2 points, because it is two-lines worth.)