Name: _____ Date: _____

Pledge: _____

Closed book: no textbook, no electronic devices, one sheet of paper with notes. Read each question carefully before answering! Write your answers on the test paper and turn in your notes.

Question 1 (5 points) Assess: [execution]
Consider the following code:

```
student_scores = {}
student_scores['Brian'] = 76
student_scores['Brian'] = 56
student_scores['brian'] = student_scores['Brian']

class_scores = [ 96, 76, 56 ]
print(student_scores['brian'] – class_scores[1])
```

What is printed on the screen after these statements have executed?
-20

Rubric: (5 points, all or nothing)

Question 2 (15 points) Assess: [execution]

(a) Using two's complement with exactly 8 bits, what is the binary representation of negative 21 (i.e., -21_{10})?

1110 1011
Rubric: (2 points for 8 bits, 3 points for correct value)

(b) Multiply 1010_2 by 111_2. Perform the operation **in binary**. Show your work. No credit will be given otherwise.

```
   1010
 X  111
-------
   1010
  10100
 101000
-------
1000110
```

Rubric: (2 points for setting up the multiplication correctly, 3 points for adding correctly)

(c) Convert 31_{10} to binary, octal, and hexadecimal.

Binary: 11111
Octal: 37
Hexadecimal: 1F
Rubric: (2 points for correct binary, 1 for correct octal, 2 for correct hexadecimal)

Question 3 (15 points) Assess: [design]
Implement the following function, using recursion on L. That means you can access L only through the expressions L[0], L == [], and L[1:].

```
def my_max(L):
    '''Assume L is a non-empty list of integers.
    Returns the maximum element in the list.

    Example 1:
    my_max([-1, 0, 1, 4, 5, 3, 2, 15]) => 15

    Example 2:
    my_max(range(10, -11, -1)) => 10
    '''
    def max_helper(L, max):
        '''Write your code here. Do not change any of the framework
        provided.'''

        if L == []:                              2 points
            return max                           1 point
        if L[0] > max:                           2 points
            return max_helper(L[1:], L[0])       4 points
        return max_helper(L[1:], max)            4 points

        OR

        if L == []:                              2 points
            return max                           1 point
        if L[0] > max:                           2 points
            max = L[0]                           3 points
        return max_helper(L[1:], max)            5 points

    return max_helper(L[1:], L[0])
```

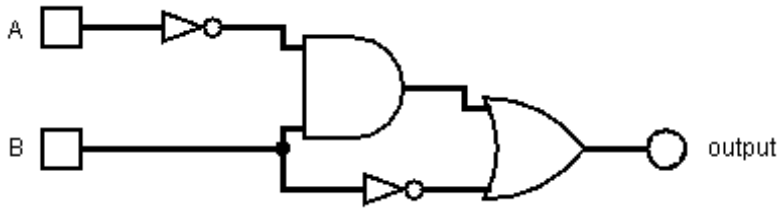
Rubric: (2 points for correct syntax, remaining points assigned next to each line)

Question 4 (10 points) Assess: [testing]
Implement two PyUnit tests for the `my_max` function you wrote in question 3. `test1` should cover example 1 from the docstring, and `test2` should cover example 2. Assume the `my_max` function appears in module `test2fall`. Fill in the blanks in the PyUnit script:

```
import unittest
import test2fall

class Test(unittest.TestCase):
    def test1(self):
        self.assertEqual(
            test2fall.my_max([-1, 0, 1, 4, 5, 3, 2, 15]), 15)

    def test2(self):
        self.assertEqual(
             test2fall.my_max(range(10, -11, -1)), 10)

if __name__ == "__main__":
    unittest.main()
```

Rubric: (
1 point for importing unittest,
1 point for importing test2fall,
For each test case:
  1 point for self.assertEqual,
  1 point for test2fall.my_max,
  1 point for correct arguments to my_max,
  1 point for correct expected integer value)

Question 5 (10 points)
Consider the following circuit:



(a) Write out the expression for the circuit using the boolean notation we discussed in class (not Python syntax).

$$\overline{A}B + \overline{B}$$

Rubric: (4 points, all or nothing)

(b) For what value or values of A and B is the output 1?

A	B	$\overline{A}B + \overline{B}$
T	T	F
T	F	T
F	T	T
F	F	T

Answer (AB = 10, 01, 00)

Rubric: (2 point for each correct pair of values as long as part a is correct)
If part a is incorrect, part b is automatically incorrect.

Question 6 (20 points) Assess: [coding]
Complete the implementation of the function `powerset` with the "*use it or lose it*" strategy.

```
powerset([]) =>
   [[]]
powerset(['a']) =>
   [[], ['a']]
powerset(['a', 'b']) =>
   [[], ['b'], ['a'], ['a', 'b']]
powerset(['a', 'b', 'c']) =>
   [[], ['c'], ['b'], ['b', 'c'], ['a'], ['a', 'c'], ['a', 'b'], ['a', 'b', 'c']]
```

```
def powerset(L):
    '''Returns the power set of the list, that is, the set of all subsets of
    the list.'''

    if L == []:

        return [[]]                                          2 points

    lose_it = powerset(L[1:])                                4 points

    use_it = map(lambda subset: [L[0]] + subset, lose_it)    8 points

    return lose_it + use_it                                  4 points
```

Rubric: (
2 points for correct syntax,
remaining points assigned next to each line)

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' Question 7 (20 points)
' You are given a rod of length n feet and a list of pairs that contain the
' prices of pieces of various sizes. For instance, [[1, 1], [2, 3]] means
' that a 1-foot rod costs $1 and a 2-foot rod costs $3. Complete the
' rod_cutter function found below. The function should compute the maximum
' value that can be obtained by cutting up the rod and selling the pieces.
' For example, if length of the rod is 4 feet long and the values are
' [[1, 1], [2, 3]], the maximum value is 6, obtained by cutting the rod into
' two pieces of length 2, each worth $3.
' Hints:
' a. Implement a use-it or lose-it algorithm
' b. Consider multiple bases cases
' c. values[0][0] refers to the length of the rod,
'    values[0][1] refers to the cost of the rod
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
def rod_cutter(values, n):
    if values == [] or n <= 0:                                  # 2 points
        return 0                                                # 1 point
    lose_it = rod_cutter(values[1:], n)                         # 4 points
    if values[0][0] > n:                                        # 2 points
        return lose_it                                          # 1 point
    use_it = rod_cutter(values, n-values[0][0]) + values[0][1] # 7 points
    return max(lose_it, use_it)                                 # 3 points


''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' Question 8 (10 points)
' The LCS function written below runs slowly when large strings are passed
' as arguments. Rewrite it to use memoization to improve performance. You may
' write a nested helper function, but the outer function must remain
' LCS(s1, s2). When you are done, the test cases below should complete
' instantaneously.
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
```

```
def LCS(s1, s2):
    '''Returns the length of the longest common subsequence in strings s1
    and s2. Uses memoization to improve performance.'''

    def fast_LCS_helper(s1, s2, memo): # 1 point for helper function
        # If key exists, return value already associated with key. 2 points
        if (s1, s2) in memo:
            return memo[(s1, s2)]

        # Do work. 4 points
        if s1 == '' or s2 == '':
            result = 0
        elif s1[0] == s2[0]:
            result = 1 + fast_LCS_helper(s1[1:], s2[1:], memo)
        else:
            result = max(fast_LCS_helper(s1, s2[1:], memo),
                         fast_LCS_helper(s1[1:], s2, memo))

        # Store and return result. 2 points
        memo[(s1, s2)] = result
        return result

    return fast_LCS_helper(s1, s2, {}) # 1 point for function call
```