Name: ______________________________ Date: __________________

Pledge: ____

Total: ____ / 75 = ____

Closed book: no textbook, no electronic devices, one sheet of paper with notes. Read each question carefully before answering, as there is **no partial credit for any response**! You may work out the solution to each question within the test itself, but **only your answers on this cover page will be graded**.

Question 1

a) **False** (2 points)

b) **3** (3 points)

Question 2

a) **11010** (3 points)

b) **1A** (3 points)

c) **32** (3 points)

d) **1111 1111** (3 points)

e) **0001 1001** (3 points)

Question 3

a) **[]** (3 points)

b) **-1** (3 points)

c) **L[-1] % 2 == 0** (3 points)

d) **len(L) - 1** (3 points)

e) **L[:-1]** (3 points)

Question 4

a) **homework** (2 points)

b) `self.assertEqual(homework.index_of_last_even([7, 5, 1, 3, 9]), -1)`
(4 points)

c) `self.assertEqual(homework.index_of_last_even([5, 2, 8, 10, 7, 2]), 5)`
(4 points)

Question 5

a) **1** (2 points)

b) **0** (2 points)

c) **1** (2 points)

d) **0** (2 points)

e) $\overline{P}\,\overline{Q}$ (2 points)

Question 6

a) **[0, []]** (4 points)

b) **lst[2:]** (4 points)

c) **lst[1:]** (4 points)

d) **[lst[0]] + use_it[1]** (4 points)

e) **lose_it** (4 points)

Question 7

7 (5 points)

Question 1 (5 points)
Consider the following code:

```
score = {}
score[('spot', 'pot')] = 3
score[('', 'a')] = 0
score[('maps', 'spam')] = 1
```

(a) What is printed on the screen after this statement has executed? (2 points)

```
print( score[('', 'a')] in score and 'spot' in score )
```

(b) What is printed on the screen after this statement has executed? (3 points)

```
print( max(score[('spot', 'pot')], score[('maps', 'spam')]) )
```

Question 2 (15 points, each 3 points)

(a) Convert 26_{10} to binary.

(b) Convert 26_{10} to hexadecimal.

(c) Convert 26_{10} to octal.

(d) Using two's complement with 8 bits, what is the binary representation of negative 1 (i.e., -1_{10})? Write your answer with **exactly 8 bits**.

(e) Using your answers from (a) and (d), what is 26_{10} - 1_{10} in **binary**? Perform the operation using **addition in binary with 8 bits**. If your answer to part (a) or part (d) is incorrect, you cannot get credit for part (e).

Question 3 (15 points)
Implement the following function, using recursion on L. You may access the len function and expressions [], L[-1], and L[:-1], but do not use slicing or indexing with non-negative indices.

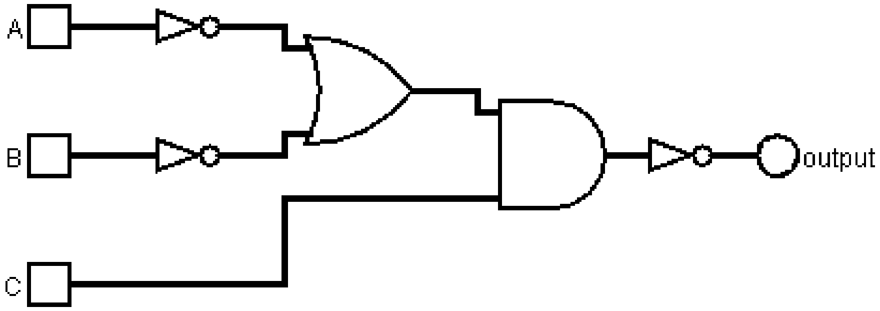
```
def index_of_last_even(L):
    '''Assume L is a non-empty list of integers.
    Returns the index of the last even number in the list. In other words,
    the function returns the highest index which is occupied by an even
    number. If no even numbers are in L, the function returns -1.

    Example 1:
    index_of_last_even([7, 5, 1, 3, 9]) => -1

    Example 2:
    index_of_last_even([5, 2, 8, 10, 7, 2]) => 5

    Example 3:
    index_of_last_even([1, 3, 5, 8, 7, 9, 11]) => 3'''


    if L == ___(a)___:

        return ___(b)___

    if _______________(c)_______________:

        return _______________(d)_______________

    return index_of_last_even(_______________(e)_______________)
```

Question 4 (10 points)
Implement two PyUnit tests for the `index_of_last_even` function you wrote in question 3. `test1` should cover example 1 from the docstring, and `test2` should cover example 2 in the docstring in question 3. Assume the `index_of_last_even` function appears in module `homework`. Fill in the blanks in the PyUnit script. You must use assertEqual.

```
import unittest
import _____(a)_____ (2 points)

class Test(unittest.TestCase):

    def test1(self):
        _______________________________(b)_____________________________________
        (4 points – no partial credit)


    def test2(self):
        _______________________________(c)_____________________________________
        (4 points – no partial credit)

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

Question 5 (10 points)

Consider the following circuit:



Write out first four rows of the truth table for this circuit. Your answers should output (0 or 1) for the given tuple of inputs (A, B, C).

```
 A | B | C |         OUTPUTS
---+---+-----------------------------
 0 | 0 | 0 |       ___(a)___          (2 points)
 0 | 0 | 1 |       ___(b)___          (2 points)
 0 | 1 | 0 |       ___(c)___          (2 points)
 0 | 1 | 1 |       ___(d)___          (2 points)
```

(e) Use De Morgan's Law to write an equivalent expression for $\overline{P+Q}$. _______________ (2 points)

Question 6 (20 points)

Complete the implementation of this function for the coin row problem with the *"use it or lose it"* paradigm.

```
coin_row_with_values([]) => [0, []]
coin_row_with_values([5, 1, 2, 10, 6, 2]) => [17, [5, 10, 2]]
coin_row_with_values([10, 5, 5, 5, 10, 50, 1, 10, 1, 1, 25]) =>
    [100, [10, 5, 50, 10, 25]]

def coin_row_with_values(lst):
    '''lst represents a row of n coins whose values are some positive integers
    c1, c2, ..., cn, not necessarily distinct.
    Returns a list containing
     - the maximum amount of money subject to the constraint that no two
       coins adjacent in lst can be picked up, and
     - a list of the coins used to create the maximum sum, appearing from left to
       right in the same order as in lst.'''
    if lst == []:
        return __________(a)__________
    use_it = coin_row_with_values(__________(b)__________)
    lose_it = coin_row_with_values(__________(c)__________)
    new_sum = use_it[0] + lst[0]
    if new_sum > lose_it[0]:
        return [new_sum, __________(d)__________]
    return __________(e)__________
```

Question 7 (5 points)
How many calls to **`fib_memo`** are made when calling **`fib(4)`**, as defined in the code below? ______

```
def fib(n):
    '''Returns the nth Fibonacci number using memoization. Assume that the 0th
    Fibonacci number is 0, so
    fib(0) = 0,
    fib(1) = 1.
    The rest of the sequence is 1, 2, 3, 5, 8, ...'''
    def fib_memo(n, memo):
        if n in memo:
            return memo[n]

        if n <= 1:
            result = n
        else:
            result = fib_memo(n - 1, memo) + fib_memo(n - 2, memo)

        memo[n] = result
        return result

    return fib_memo(n, {})
```