Introduction to Computer Science I
Fall 2014
Final exam — Solutions

1. Question: Provide short answers—no more than a few sentences—to the following questions:
   (a) Why do computers work in *binary*, and not (say) *ternary*?
   (b) How was *Deep Blue* designed to play chess? That is, how did it determine its next move?

   Answer:
   (a) It is economical to make extremely fast devices that perform logic operation, which are equivalent to binary arithmetic operations.
   (b) It performed an extensive search by simulating each possible move by itself and its opponent for many moves ahead in the game, evaulating the quality of each move and choosing the best.

2. QUESTION: Consider *Pascal's Triangle*, although shown somewhat lopsidedly:

```
1
1  1
1  2  1
1  3  3  1
1  4  6  4  1
1  5  10 10 5  1
1  6  15 20 15 6 1
```

Specifically, values at the *edges* of this grid of values are always 1. Meanwhile, an *interior* value at row r and column c, written as (r, c), are the sum of the values at coordinates $(r-1, c-1) + (r-1, c)$. That is, each value is the sum of the value "north" of it and the value "northwest" of it.

Write a function, `printPascal(n)`, that prints the first n rows of Pascal's Triangle.

ANSWER:

```
def printPascal (n):
    for r in range (1, n+1):
        l = [1] * r
        for c in range (1, r-1):
            l[c] = p[c-1] + p[c]

        printRow(l)
        p = l

def printRow (l):
    for v in l:
        print(v, end='\t')
    print()
```

3. QUESTION: Consider a *list of tuples of integers*, where each tuple is of the same length. Furthermore, to consider one tuple as *dominating* another, then each of the elements of the first must be greater than the corresponding elements of the second. For example, if we have two tuples of length 3, then we can say that (a_0, a_1, a_2) *dominates* (b_0, b_1, b_2) *iff* $a_0 > b_0$ AND $a_1 > b_1$ AND $a_2 > b_2$.

   Write a function, `orderTuples(l)`, that sorts this list of tuples from least to most dominant. If, for some pair of tuples, neither dominates the other, then they are considered *equivalent*, and thus their order with respect to each other does not matter.

   ANSWER:

```
def orderTuples (l):
    for i in range (0, len(l)):
        minIndex = i
        for j in range (i, len(l)):
            if dominated(l[j], l[minIndex]):
                minIndex = j
        l[i], l[minIndex] = l[minIndex], l[i]

def dominated (a, b):
    for i in range(len(a)):
        if a[i] >=  b[i]:
            return False
    return True
```

4. QUESTION: Recall our *Sudoku solver*. Our state-space search function to solve a given puzzle could yield one of two results: either the puzzle was *unsolvable*, or the puzzle *was solved* (that is, had been completely filled-in without violating the rules). However, this algorithm could not detect whether a puzzle was invalid because it might have **more** than one solution.

   Write a modified version of the Sudoku solver, `solve(g)`, that accepts the Sudoku grid `g`, attempts to solve it, and returns one of the following integer results:

   - `0` to indicate that **no solutions** exist.
   - `1` to indicate that **exactly one solution** exists (and the grid contains it).
   - `2` to indicate that **more than one solution** exists.

   ANSWER:

```
def solve (grid):
    empty = findEmpty(grid)
    if empty == None:
        return 1
    (r,c) = empty
    solutions = 0
    for v in range(1, 10):
        grid[r][c] = v
        if isValid(grid, r, c):
            solutions += solve(grid)
            if solutions > 1:
                grid[r][c] = 0
                return 2
    grid[r][c] = 0
    return solutions
```

5. QUESTION: **Write a function**, `permute(l)`, that generates a *list of all possible permutations of the items in the list* `l`.[1] That is, if `l = [1, 2, 3]`, then this function should return:

```
[ [1, 2, 3], [2, 1, 3], [2, 3, 1],
  [1, 3, 2], [3, 1, 2], [3, 2, 1] ]
```

HINT: Think recursively! Given the list `[1, 2, 3]` to permute, what if some "magic function" provided you the possible orderings of just `[2, 3]`, without the `1`, by returning `[ [2, 3], [3, 2] ]`? How could you use this partial result to re-inject the `1` in every possible position, creating a complete list of the orderings?

ANSWER:

```
def permute (l):
    if len(l) <= 1:
        return [l]
    first = [l[0]]
    rest = l[1:]
    perms_rest = permute(rest)
    perms = []
    for perm in perms_rest:
        for i in range(len(perm)):
            perms.append(perm[:i] + first + perm[i:])
        perms.append(perm + first)
    return perms
```

[1] For those not used to the term, a *permutation* of a list is an *ordering* of its elements.