You have 70 minutes.

Show your work and explain your answers. Good work often earns partial credit. A correct answer with no explanation often earns little or no credit.

If you are asked to write code but you do not know the exact Python required, then try to write code that is approximately correct. If you think that your code does not demonstrate that you understand the algorithm, then describe your idea in English as well. Be precise enough that I cannot misinterpret your solution.

Good luck.

A. Examine the following code from our Tic Tac Toe project. After the code comes a list of seven variables and other Python expressions. In the space next to each one, write all possible types that it might take on, during the execution of the function.

```
def computerVsComputer():
    board = [[" ", " " ," "], [" ", " " ," "], [" ", " " ," "]]
    player = "O"
    turns = 0
    printBoard(board)
    while turns < 9 and not hasWon(board, "X") and not hasWon(board, "O"):
        m = move(board, player)
        if m == None or board[m[0]][m[1]] != " ":
            print "Invalid move", m, "by player", player, "on this board:"
            printBoard(board)
            return
        board[m[0]][m[1]] = player
        print
        printBoard(board)
        player = otherPlayer(player)
        turns = turns + 1
    if hasWon(board, "O"):
        print "O wins."
    elif hasWon(board, "X"):
        print "X wins."
    else:
        print "Draw."
```

a. `player`:

b. `hasWon`:

c. `m`:

d. `board`:

e. `board[m[0]]`:

f. `board[m[0]][m[1]]`:

g. `m == None or board[m[0]][m[1]] != " ":`

B. Our encryption functions would be better if they handled both upper- and lower-case letters. The following function might help us improve these encryption functions. Complete the function — in other words, describe the algorithm as specifically as you can — in either English or Python.

```
# Returns the sum of the given letters, preserving the case of the former.
# That is, rotates the former letter through the alphabet by the latter,
# keeping its case the same. For example, addLetters("c", "K") returns
# "m", while addLetters("W", "G") returns "C".
# Input: English letter (upper- or lower-case). English letter (upper-case).
# Output: English letter (of same case as the first input).
def addLetters(a, b):
```

C. Working in 8-bit two's complement, compute $70 - 117$ by negating and adding. Check your answer by converting it back to decimal. Show all work.

D. Chartreuse is a bright yellowish-green. Find a reasonable 24-bit RGB value for chartreuse. Give your answer in binary, decimal, and hexadecimal. Your three answers should exactly agree.

E. We have discussed how `encryptRot13` is a special case of `encryptCaesar`, which is in turn a special case of `encryptRepeatedPad`. Now, explain how `encryptCaesar` is a special case of `encryptSub` (the substitution cipher). For full credit, write `encryptCaesar` as a complete Python function, that uses `encryptSub` to do its work.

F. Connect Four is a game similar to to Tic Tac Toe, with three differences. First, the board is 7 squares wide and 6 squares tall, rather than 3×3. Second, a player must make 4 contiguous plays in a row, column, or diagonal to win, rather than just 3. Third, the board is oriented vertically in space; whenever a piece is played in a square, it immediately falls down the board, until there are no empty squares below it.

a. In Tic Tac Toe, there are 8 ways to win. How many ways are there in Connect Four?

b. Imagine a `hasWon` function for Connect Four, written using loops, so that it is as short as possible. In Python, write out the structure of the loops — how many loops there are, how they are nested, and which winning ways they test. You do not have to fill in the bodies of the loops, but the first line of each loop (the `while` or `for` line) should be as detailed as possible.