Question 1. [6 marks]

Each subquestion on this page has a small piece of code that is supposed to work as described in the comment statement but has a small part missing. For each one, add the missing part inside the box. Your solution must follow the instructions in the comment statement. Each subquestion is independent.

Part (a) [1 mark]

```
d = {1:'remove me', 2:'take me away', 3:'KEEP'}
# Remove two items from d so that the code below the box prints 'KEEP'.
# Do not reassign d to a new dictionary.
```

```
del d[1]
del d[2]
```

```
for item in d.keys():
    print d[item]
```

Part (b) [1 mark]

```
d = {}
# Add exactly one key-value pair to d so that the code below the box prints 'YES'.
```

```
d['YES'] = 'value could be anything' #many correct answers
```

```
for item in d:
    print item
```

In the box beside each piece of code below, write its output. If it would generate an error, say so, and give the reason for the error.

Part (c) [2 marks]

```
L1 = ["once", "upon"]
L2 = L1
L1 = L1 + ['a', 'time']
print L1
print L2
```

```
['once', 'upon', 'a', 'time']
['once', 'upon']
```

Part (d) [2 marks]

```
L1 = ["Mary", "had", "a"]
L2 = L1
L2.append(["little", "lamb"])
print L1
print L2
```

```
['Mary','had', 'a',['little', 'lamb']]
['Mary','had', 'a',['little', 'lamb']]
```

Question 2. [6 marks]

Suppose we are keeping track of who is working with whom on a course assignment. We could represent the groups using a nested list of student numbers like this:
`[[2, 9], [4], [3, 1]]`. (Here we use one-digit student numbers to make the example easier to read.) In this example, we have two groups of two students, and one group of one student.

Part (a) [1 mark]

Suppose we want to make sure that everyone is in a group, and no one is in more than one group. The following function checks this.

```
def valid_grouping(group_list, class_list):
    '''Return True if every student in class_list is in exactly 1 group
    according to group_list, and False otherwise.'''
```

Write a call to the function that should return `True` and involves a class list of 6 students.

Solution: There are many correct answers

```
valid_grouping([[0, 1], [3, 4, 2], [5]], [0,1,2,3,4,5])
```

Part (b) [5 marks]

Now write the function. You do not need to repeat the `def` line or the docstring.

Solution:

```
    for student in class_list:
        # The number of groups student is in.
        num_groups = 0
        for group in group_list:
            if student in group:
                num_groups = num_groups + 1
        if num_groups != 1:
            return False
    return True
```

Question 3. [7 marks]

Part (a) [5 marks]

Write the following function according to its docstring. The string matching that decides which lines to include in the result should be case-insensitive. However, the strings returned should be unmodified lines from the input file. For example, if the function is called with a file containing the line `Match\n`, and `s` has the value `maTch`, the original line (without any case changes) will be included in the list returned by the function.

```
def find_lines(f, s):
    '''Return a list of all lines in open file f that contain string s anywhere within them.'''
```

Solution:

```
def find_lines(f, s):
    answer = []
    lc_s = s.lower()
    for line in f:
        if lc_s in line.lower():
            answer.append(line)
    return answer
```

Part (b) [2 marks]

Write a main block that will use your function to print all the lines in file `poem.txt` that contain the string `love` (with any mixture of uppercase and lowercase letters.)

```
matches = find_lines(open('poem.txt'),'love')
for line in matches:
    print line
```