Question 1. [10 MARKS]

Strong hint: For this question, trace the code using the memory model if you get stuck.

Part (a) [2 MARKS] What gets printed?

```
def func1(n):
    n= n + 4
    return n

number = 2.5
number = func1(number)
print type(number)
```

<type 'float'>

Part (b) [2 MARKS] What gets printed?

```
def func2(s):
    s = s - 10

s = 25
func2(s)
print s
```

25

Part (c) [2 MARKS] What gets printed?

```
def func3(L):
    L[2]=1

list1 = [1,2,3]
func3(list1)
print list1[:]
```

[1, 2, 1]

Part (d) [2 MARKS] What gets printed?

```
def func4(L):
    for elt in L:
        elt = elt + 1
    return L

list2 = [4,5,6]
func4(list2)
print list2
```

[4, 5, 6]

Part (e) [2 MARKS] What gets printed?

```
def func5(L):
    L.append(4)
    L[0][0] = 3
    return L

list1 = [1, 2, 1]
list3 = [list1, list1, list1]
func5(list3)
print list3
```

[[3, 2, 1], [3, 2, 1], [3, 2, 1], 4]

Question 2. [6 MARKS]

Part (a) [2 MARKS] Rewrite this block of if statements so that no line is indented more than once.

```
if weather == 'rainy':
    instr = 'I should wear a raincoat'
else:
    if weather == 'windy':
        instr = 'I should wear a windbreaker'
    else:
        if weather == 'snowy':
            instr = 'Perhaps a snowsuit'
        else:
            if weather == 'sunny':
                instr = 'I need to put on sunscreen'
```

Solution:

```
if weather == 'rainy':
    instr = 'I should wear a raincoat'
elif weather == 'windy':
    instr = 'I should wear a windbreaker'
elif weather == 'snowy':
    instr = 'Perhaps a snowsuit'
elif weather == 'sunny':
    instr = 'I need to put on sunscreen'
```

Part (b) [4 MARKS]

Write a function `format_x` that allows us to replace the following block of code with the line `y = format_x(x)`. A docstring is required.

```
if x == 'A':
    y = 'My bloodtype is A.'
if x == 'B':
    y = 'My bloodtype is B.'
if x == 'O':
    y = 'My bloodtype is O.'
if x == 'AB':
    y = 'My bloodtype is AB.'
```

Solution:

```
def format_x(x):
    '''str -> str
    Return a sentence that says 'My bloodtype is x.'''

    return 'My bloodtype is %s.' % x
```

Question 3. [6 MARKS]

Part (a) [3 MARKS] Write code equivalent to the code below that uses for loops and does not use while loops.

```
i = 0
ret_list = []
while i < len(L):
    j = 0
    ret_list.append([])
    while j < len(L[i]):
        if L[i][j] == 1:
            ret_list[i].append(j)
        j += 1
    i += 1
```

Solution:

```
ret_list = []
for i in range(len(L)):
    ret_list.append([])
    for j in range(len(L[i])):
        if L[i][j] == 1:
            ret_list[i].append(j)
```

Part (b) [3 MARKS]

Write code equivalent to the code below that use while loops and does not use for loops.

```
count = 0
count_L = []
for i in range(len(L)):
    if count < 15 and L[i] < 20:
        count_L.append(i)
        count += 1
```

Solution:

```
count = 0
count_L = []
i = 0
while i < len(L) and count < 15:
    if L[i] < 20:
        count += 1
        count_L.append(i)
    i += 1
```

Question 4. [7 MARKS]

Write the following function according to its docstring.

```
def evens_only(L):
    '''(list) -> list of ints
    Return a list of all the even integers in L. Also, modify L so that every
    element that is not an even integer is replaced with the bool False.'''
```

Solution:

```
def evens_only(L):
    '''(list) -> list of ints
    Return a list of all the even integers in L. Also, modify L so that every
    element that is not an even integer is replaced with the bool False.'''

    ret_list = []
    for i in range(len(L)):
        if type(L[i]) == int and L[i] % 2 == 0:
            ret_list.append(L[i])
        else:
            L[i] = False
    return ret_list
```

Question 5. [7 MARKS]

Consider a file where each line is a sentence with the following format:

Name the *animal_type* is *x* years old.

In this format `Name` is a single word that is the animal's name (for eg. Tom, Bessie, etc.), `animal_type` is a single word that is the type of animal it is (for eg. cat, cow, dog, pig, etc.), and `x` is an integer that represents how old the animal is.
Given this file format write the body of the following function according to its docstring.

```
def animal_ages(filename):
    '''(str) -> list of (str, int) tuples
    Open the file filename and return a list where each element of the list
    contains the animal_type and age of an animal from the corresponding
    line of the file.'''
```

Solution:

```
def animal_ages(filename):
    '''(str) -> list of (str, int) tuples
    Open the file filename and return a list where each element of the list
    contains the animal_type and age of an animal from the corresponding
    line of the file.'''

    animal_file = open(filename, 'r')
    ret_list = []
    for line in animal_file:
        words = line.split()
        ret_list.append(words[2], int(words[-3]))

    animal_file.close()
    return ret_list
```

Question 6. [8 MARKS]

Consider the file type from question 5, with one additional property - that the lines are sorted by `animal_type`. So all of the types of animals that begin with the letter `'a'` come before all of the animals whose types begin with `b` and so on. You may assume that all the `animal_types` are lower case. Suppose we run `animal_ages` (from question 5) with such a file as input and get a list of all the animal types with their ages. Assume that the following function gets such a list as its input parameter and write the following function according to its docstring. You may ***NOT*** use dictionaries.

```
def oldest_animals(age_of_animals):
    '''(list of (str, int) tuples) -> list of (str, int) tuples
    Return a list of tuples that pair each animal type with the age of the oldest
    animal of that type.
    Assumes the input list is sorted by animal type.
    For example, given [('cat', 5), ('cat', 7), ('dog', 2)] the function would
    return [('cat', 7), ('dog', 2)]'''
```

Solution:

```
def oldest_animals(age_of_animals):
    '''(list of (str, int) tuples) -> list of (str, int) tuples
    Return a list of tuples that pair each animal type with the age of the oldest
    animal of that type.
    Assumes the input list is sorted by animal type.
    For example, given [('cat', 5), ('cat', 7), ('dog', 2)] the function would
    return [('cat', 7), ('dog', 2)]'''

    last_animal = ''
    ret_list = []
    for animal_age in age_of_animals:
        if last_animal != animal_age[0]:
            #Then we have a new animal.
            ret_list.append(animal_age)
            last_animal = animal_age[0]
        else:
            if animal_age[1] > ret_list[-1][1]:
                #Then we have an older animal
                ret_list[-1] = animal_age
    return ret_list
```