Question 1. [6 MARKS]

The program below creates and displays a picture.

```
import media

def create_graphic(width, height):
   pic = media.create_picture(width, height)
   media.add_rect_filled(pic, 0, 0, width, height, media.gray)
   for pixel in pic:
      x = media.get_x(pixel)
      y = media.get_y(pixel)
      if x < width / 2 and y < height / 2:
            media.set_color(pixel, media.blue)
   return pic

if __name__ == '__main__':
   pic = create_graphic(100, 50)
   media.show(pic)
```

Draw the picture that is displayed. Label any colours clearly, and indicate all relevant dimensions.

Solution:

(You were to draw a picture of this, which is probably easier than describing it in words.)

A rectangle that is 100 pixels wide by 50 pixels high, with a gray background and a blue smaller rectangle in the upper-left corner, 50 pixels wide by 25 pixels high. The four corners of the overall picture are (0,0), (99,0), (99,49), and (0,49). The four corners of the smaller blue rectangle are (0,0), (49,0), (49,24), and (0,24). It was not necessary to give both the x,y coordinates and the dimensions. Either one is fine.

Question 2. [6 MARKS]

For Assignment 1, you wrote function `average_red`:

```
def average_red(pic):
    '''Given a picture pic, return a float that is the average red value
    among all its pixels.'''
```

Complete the following function according to its docstring description. You should call `average_red` in your function. You may assume that it works correctly. You do not have to import it.

```
def extreme_red(pic):
    '''Modify Picture pic so that its red values are all extreme: Give each pixel whose
    red value is greater than the average red value of the picture a new red value of 255,
    and give all other pixels a new red value of 0.'''
```

Solution:

```
def extreme_red(pic):
    average = average_red(pic)
    for pixel in pic:
        if media.get_red(pixel) > average:
            media.set_red(pixel, 255)
        else:
            media.set_red(pixel, 0)
```

Question 3. [8 MARKS]

The left-hand column in the table below shows a series of statements to be interpreted by the Python shell. For each print statement, show the expected output in the right-hand column.

Solution:

Python statements	**Output**
`print (5267 % 100) / 10`	6
`list = [1,3,5,9,2,0,11]` `print list[2:5]`	[5, 9, 2]

`x = 9.6`
`y = x + 5`
`x = 20.2`
`print y`
	14.6

`name1 = "Monty"`
`name2 = "Python"`
`name1 = name2`
`name2 = name2[2:]`
`print name1`
	'Python'

Question 4. [5 MARKS]

Suppose we have a function `valid_password` that checks whether a string can be used as a password (it must not be too short, for example):

```
def valid_password(s):
    '''Return True if string s is a valid password, and False otherwise.'''
```

The code below reads in the new password and asks the user to type it a second time. In order for the new password to be acceptable, it must be a valid password, and the user must have typed it exactly the same way twice. If not, the program will print an error message. Fill in the missing `if` condition below so that the error message will be printed if appropriate. You may call `valid_password` as needed; assume that it works correctly. You do not have to import it. Do not change or add to the code in any way other than to add the missing `if` condition.

```
    p1 = raw_input("Please enter your new password: ")

    p2 = raw_input("Please retype your new password: ")

    if

        print "Sorry, your new password has been rejected."
```

Solution:

This is the missing `if` condition:

```
p1 != p2 or not valid_password(p1):
```