University of South Australia

UniSA STEM

COMP 1039 Problem Solving and Programming

Practical 8

Create a separate file in order to complete each of the following exercises (refer to practical 1 or ask your supervisor if you are having problems doing so). Your files should include appropriate comments consisting of the following – file name, your details, version number and date, brief program description and the University's academic misconduct statement.

Question 1

Write a function called `multiply_numbers` which will take three numbers as parameters, find the product of the numbers, and return the result. Defining the function does not execute the function – you will need to call the function from elsewhere in the program, passing the appropriate information to the function through its parameters. Call function `multiply_numbers` once it has been defined in order to ensure that it is working correctly. Display the product to the screen with an appropriate message.

Question 2

Write a function called `maximum` which will take two numbers as parameters, and return the larger of the two numbers. Call function `maximum` once it has been defined in order to ensure that it is working correctly. Display the maximum to the screen with an appropriate message.

Question 3

Write a function called `get_random` which will take one number as a parameter and generate a random number in the range 1 – `number`. The number passed in as a parameter must be a default argument set to the default value of 10, i.e. number=10. Call function `get_random` once it has been defined in order to ensure that it is working correctly. Implement a loop that calls the function and displays the result to the screen 10 times.

Question 4

Write a function called `capitalize_words` that accepts a string as a parameter and returns a copy of the string with the first character of each word capitalized. For example, if the argument is "hello! my name is Jo. what is your name?" the function should return the string "Hello! My Name Is Jo. What Is Your Name?". The program should ask the user to enter a string and then pass it to the function. The modified string should be displayed.

Question 5

Write a function called `sum_list` that accepts a list (of numbers) as a parameter and returns the sum of the elements in the list. Your solution should make use of a for loop to sum the elements in the list. Call `sum_list` once it has been defined in order to ensure that it is working correctly.

Question 6

a) Write a function called `generate_lotto_number` that generates and returns a seven-digit lottery number. The function should generate seven random numbers, each in the range of 0 through 9, and assign each number to a list element. The function does not take any parameters but returns a list containing the randomly generated numbers. Hint: Use a loop in order to assign each random number to a list element. Call function `generate_lotto_number` once it has been defined in order to ensure that it is working correctly.

b) Write a function called `display_lotto_number` that takes a list as a parameter and displays the list to the screen as follows:

```
5, 1, 4, 5, 9, 2, 3
```

Call function `display_lotto_number` once it has been defined in order to ensure that it is working correctly.

c) Implement a loop that calls the function and displays the result to the screen 20 times.

(Modified: Gaddis, Tony. Programming Exercises, Chapter 8).

Question 7

Write a function called `update_list` that accepts a `list` and the `size` of the list as parameters. The function prompts for and reads `size` (as many as the list can hold) numbers and stores them in the list. The function does not return anything.

Call function `update_list` once it has been defined in order to ensure that it is working correctly. Use the following variable definitions in your solution.

```
number_list = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
MAX_SIZE = 10
```

Display the list before and after call to function `update_list()` to ensure the list has been updated correctly. Do this outside of the function definition.

Question 8

Write a function called `create_list` that accepts a `list` as a parameter. The function prompts for and reads numbers until the user enters a negative number and stores the numbers in the list. The function returns the list of numbers.

Call function `create_list` once it has been defined in order to ensure that it is working correctly. Use the following variable definitions in your solution.

```
number_list = []
```

Display the list before and after call to function `create_list()` to ensure the list has been updated correctly. Do this outside of the function definition.

Checkpoint: Please make sure your supervisor has seen the work you have completed so far.

Question 9

Write a Python program that will perform a simple encryption on a text file. The encryption method used on the file will involve reversing the order of letters in each word.

The following stages of development are recommended:

Stage 1

Read from a file called `'message.txt'` and display the contents of the file to the screen. Use method `readlines()` to do this. The `readlines()` method returns a list containing all the lines of the data in the file. Hint: Please refer to slide 11 and 12 of week 11 reading slides (File Input and Output) for help with this if you are stuck.

Stage 2

After every tenth word displayed to the screen, print a newline character to the screen. Hint: you will need to include another for loop (i.e. nest for loops) and use the method `split()` to do this. You will also need to use a variable which keeps track of how many words have been displayed to the screen.
No other processing is done at this stage.

Stage 3

Add file output. Your program should output each word to a file named `'newMsg.txt'`. After every tenth word, print a newline character to the output file. Hint: Please refer to slide 16 of week 11 reading slides (File Input and Output) for help with this if you are stuck.

Stage 4

For each word, reverse the order of the letters in the word. Hint: use slicing to do this. i.e. `word[::-1]`.

Given the following input file:

```
Unfortunately, no one can be told what the Matrix is.  You have
to see it for yourself. This is your last chance.  After this,
there is no going back.  You take the blue pill and the story
ends.  You wake in your bed and you believe whatever you want to
believe. You take the red pill and you stay in Wonderland and I show you how
deep the rabbit hole goes. Remember that all I am offering is
the truth.  Nothing more.
```

Your program should produce the following output:

```
,yletanutrofnU on eno nac eb dlot tahw eht xirtaM .si
uoY evah ot ees ti rof .flesruoy sihT si ruoy
tsal .ecnahc retfA ,siht ereht si on gniog .kcab uoY
ekat eht eulb llip dna eht yrots .sdne uoY ekaw
ni ruoy deb dna uoy eveileb revetahw uoy tnaw ot
.eveileb uoY ekat eht der llip dna uoy yats ni
dnalrednoW dna I wohs uoy woh peed eht tibbar eloh
.seog rebmemeR taht lla I ma gnireffo si eht .hturt
gnihtoN .erom
```

Checkpoint: Please make sure your supervisor sees both your program output and code.

Question 10 - Let's go a little deeper…

Abstraction is a basic problem solving method for handling complexity, and it works well with decomposition when developing a solution to a problem.

For example, if we were asked to write a program that will count and display the number of words in a text file, we could approach this using decomposition:

1. Open the text file for input
2. Read the contents of text file into a string
3. Count the words in a string
4. Print out the result
5. Close the text file

You can most likely see how to do steps 1, 2, 4 and 5 as Python statements.

You have seen the len() function in Python – for any sequence (i.e. string, list), this tells you how many elements are in the sequence.

For example:

```
theList = [1, 2, 3, 4]
print(len(theList))
```

would produce the output:

```
4
```

The `len()` function takes as a parameter the sequence – so that it can be called with any sequence and returns the number of elements in the sequence.

This is essentially what most functions do – given (as parameters) some information, the function does something using the parameters and may return to you some information about the parameters.

If we could write our own function that takes a string as its parameter, and returns a count of the words in the string, then we can implement the algorithm above to display the number of words in our text file.

Suppose our function is called `word_count`, we can then code the above algorithm as follows:

```
infile = open('question10.txt', 'r')
lines = infile.read()
count = word_count(lines)
print('The file called question5.txt contains', count, 'words')
infile.close()
```

Now that we have implemented the above algorithm in Python code, we now just have to write our function called `word_count`.

We have used decomposition and abstraction to change our problem from printing the number of words in a file, to counting words in a string - we have simplified the problem.

If we can write the function `word_count`, then we can also use it to solve other problems.

For example: let's explore the following kindergarten counting game problem.

1.5 Kindergarten Counting Game

Everybody sit down in a circle. Ok. Listen to me carefully.
"Woooooo, you scwewy wabbit!"

Now, could someone tell me how many words I just said?

Input and Output

Input to your program will consist of a series of lines (read from a file), each line containing multiple words (at least one). A 'word' is defined as a consecutive sequence of letters (upper and/or lower case). Your program should output a word count for each line of input. Each word count should be printed on a separate line.

Sample Input

Meep Meep!
I tot I taw a putty tat.
I did! I did! I did taw a putty tat.
Shsssssssssh ... I am hunting wabbits. Heh Heh Heh Heh ...

Sample Output

2
7
10
9

The Kindergarten counting game essentially prints the word count for each line in a text file. As long as the definition of 'word' is the same in both problems, the kindergarten game can be solved as follows:

1. Open the text file for input
2. For each line in the file
    a. Count the words in the line
    b. Print out the word count
3. Close the text file

We could then implement this in Python in at least a couple of ways:

Version 1:

```
infile = open('question10.txt', 'r')
lines = infile.readlines()
for line in lines:
      count = word_count(line)
      print(count)
infile.close()
```

Version 2:

```
infile = open('question10.txt', 'r')
for line in infile:
      count = word_count(line)
      print(count)
infile.close()
```

When you use decomposition and abstraction in problem solving, you may recognise one of the steps as being the same problem that you have done before – and in programming, this means that some of the functions you write can be re-used in multiple programs.

You have already been doing this with the `print()` function – it is a really useful low level function that allows you to print a list of values to the screen.

Both versions of the kindergarten problem above require you to write a function `word_count()` that counts the number of words in a string.

For this question of the practical, we will look at several ways of defining what constitutes a word and how to count the words.

So… let's start with a simple approach – use `split()` to chop up the text into 'words'. This is essentially saying that groups of characters between 'white space' (space, tabs, etc) is a word.

Exercise 10a:

Write a version of the function `word_count()` that uses `split()` to chop the line up (using white spaces).
Put the function definition before the top level code (use version 1 of the code provided above – an arbitrary choice).

Test your code by downloading and saving the text file provided (question5.txt). Work out what the word count for each line should be, and then run your program with the test file provided as input.

Did the code produce the correct number of words in each line?

Exercise 10b:

You will notice that some of the lines had a wrong word count as there were groups of punctuation characters that created artificial words. Problem! So how do we fix this…?? …Let's do so by replacing these characters by spaces and then splitting the line.
To do this, we will utilize the string module – we have already spent some time with string methods (refer to practical 4 for a refresher if you need to). The string module also contains a number of constant strings. One such constant string is `string.punctuation` – it contains all of the ASCII punctuation characters.
Within the python interpreter, import the string module and print out `string.punctuation`.

```
import string
print(string.punctuation)
```

In order to replace punctuation characters with spaces, we will need to make use of the replace method:

`s.replace(t, u, n)` Replaces within s occurrences of t by u (up to n times if n given).

Modify the `word_count()` function by – Iterating across the characters of `string.punctuation` and for each character replace occurrences of the character in the string s by a space character (use a for loop to do this). Use `split()` to chop the string s up and count the words.

Does this work?

Exercise 10c (a final version of word_count):

The problem with the previous version of `word_count` is that words such as it's get split into two ('it' and 's'). You can avoid this by removing the character rather than replacing it with a space. Modify your previous `word_count` function so that the punctuation characters are replaced by `''` rather than `' '`. This now seems to produce the desired result.

This is a better `count_word()` function, but may still have problems in many situations. For example, at the moment nothing has been said about numbers – do they constitute words or not? Are there odd situations in English that are not catered for? Writing a really good test file could be used to clarify the definition of a 'word' and help in modifying the function. That's just a little something to think about, but that's all the work we will be doing on the `word_count` function.

Question 11

part a)

Download the GirlNames.txt and BoyNames.txt files from the course website (week 12 topic under the 'Practical Work' heading.

GirlNames.txt – This file contains a list of the 200 most popular names given to girls born in the United States from the year 2000 through 2009.

BoyNames.txt – This file contains a list of the 200 most popular names given to boys born in the United States from the year 2000 through 2009.

Write a program that reads the contents of the two files into two separate lists. The user should be able to enter a boy's name, a girl's name, or both, and the application will display messages indicating whether the names were among the most popular and if so, the position (index) the name appears in the list. You MUST use a loop in order to identify whether the name exists in the list and the position (index) in the list.

part b)

Now let's solve the problem in part a) by using functions. When looking at your solution to part a), can you identify sections of code that are repeated (i.e. find the patterns)? These repeated sections of code are good candidates for functions.

- Write a function called read_list() that accepts a file name as a parameter, opens the file for reading and reads the popular names into a list. The function returns a list containing the popular names read in from the file.
- Write a function called find_name() that accepts a list and a name as parameters. If the name (passed in as a parameter) is found in the list (passed in as a parameter) the function returns the position (index) it was found, otherwise the function returns -1. You MUST use a loop in your solution.

Implement a solution to part a) by using the functions implemented in this part.

Optional Practical Work: More with the graphics module…

Last week we started using the module graphics.py, with the associated documentation (graphics.pdf). **If you have not done so and you wish to undertake the optional practical work, please complete the optional section of practical 7 (Starting with the graphics module) before commencing this practical.** This week we will introduce another module called `game.py` and look at simple animation and handling key press events.

Graphics object provided by game.py

The `game.py` module provides a drawable object called `GameBlock`.

GameBlock(x, y, width, height, colour, imageName, blockType, points)	Constructs a block with given center position x, y, width, height, colour, imageName, blockType, and points.

Methods:

getColour()	Returns the colour of the block.
setColour(colour)	Sets the colour of the block.
getCentreX()	Returns the x coordinate (center position) of block.
getCentreY()	Returns the y coordinate (center position) of block.
getWidth()	Returns the width of the block.
getHeight()	Returns the height of the block.
move(x, y)	Moves the block dx units in the x direction and dy units in the y direction.
getBlockType()	Returns the block type.
setBlockType(type)	Sets the type of the block.
getScrollSpeed()	Returns the scroll speed of the block.
setScrollSpeed(speed)	Sets the scroll speed of the block.
getPoints()	Returns the block points.
setPoints(points)	Sets the block points.
getBlock()	Returns the block.
getImage()	Returns the displayed image.
getImageName()	Returns the name of the displayed image.
draw(graphicsWindow)	Draws the block into the given graphics window.
undrawn()	Undraws the block from the graphics window.

You can construct a GameBlock object with coordinates (250, 250), size 20 and colour red, as seen in the following example:

```
import graphics
import game

win = graphics.GraphWin("Practical Example", 500, 500)
win.setBackground("white")

# Create game block at position x=250, y=250, size of 20 pixels and colour red
block = game.GameBlock(250, 250, 20, 20, 'red', None, None, 0)
block.draw(win)

# The following code moves the block dx (20) units in the x direction and
# dy (50) units in the y direction
block.move(20, 50)

win.getMouse()
win.close()
```

Exercise 1

In a new Python file, write a program that:

- creates a window (400x400) and sets the title to 'Practical 8 - Exercise 1'
- sets the background colour to black
- draws a yellow GameBlock object at position (200, 200) with a size of 50 pixels.

Scroll to the end of this practical for a sample solution if you are having difficulties with this.

SECTION I - Simple animation using game module

Now let's look at performing simple animation. We will create a square block (GameBlock) positioned within a window at a random location and that is moving in a random direction. That is, we will animate the block and have it 'bounce' off the walls.

Design:

Generate the centre of the block so that it is completely inside the window.
Generate a displacement for a step in time for the block – use random numbers
    for the x-component and y-component displacements.
Draw the block.
For each unit of time in the animation:
    Move the block by the given displacement.
    If the block centre is closer than half-size to a vertical wall,
        reverse the x-component of the displacement.
    If the block center is closer than the half-size to a horizontal wall,
        reverse the y-component of the displacement.

Now let's implement the above design…

Exercise 2 – Create the window and block (GameBlock)

For this exercise, create a square graphics window. To allow easy changes to be made, define an integer variable winsize for the window size, and then create the window with the following:

```
winsize = 300
win = graphics.GraphWin("Bouncing Block", winsize, winsize)
win.setBackground("white")
```

Don't forget that you need to import the module graphics to be able to create the window.

Create a centre for the block using two random integers (called xPos and yPos).
Notes:

- You will need to import random module.
- The method random.randint(low, high) generates a random integer between low and high.

Create the block (GameBlock):

- Use game.GameBlock (you will need to import game module).
- To allow easy changes to be made, define an integer variable blocksize for the block size (20 pixels).
- Set the block colour to 'red'
- Draw the block in the graphics window.

In the last practical, to stop the window from closing, we waited for user input (from the mouse). That is, we waited for the user to perform some action, such as clicking inside the window. If the displayed window is referred to by a variable called win, you can wait for a mouse click and then finish using:

```
win.getMouse()
win.close()
```

Another way of waiting is to call:

```
win.mainloop()
```

This essentially waits for an 'event' to occur, process the event, and then waits for the next event. You can finish the program by closing the graphics window.

Scroll to the end of this practical for a sample solution if you are having difficulties with this.

Exercise 3 –Move the block (GameBlock)

Create a random displacement for a unit of time – using two random integers to store x and y displacements (say between -10 and 10).

Now, within a loop:

Move the block by the displacement (there is a move() method for GameBlock).
Note that the centre position of the block can be returned from the variable for the block using getCentreX() and getCentreY() methods.
If the block overlaps with either vertical wall,
change the sign of the x-component of the displacement.
If the box overlaps either horizontal walls,
change the sign of the y-component of the displacement.

Note: that this does not stop the box overlapping the walls, so it doesn't quite 'bounce' off the walls, but this is hardly noticeable.

Scroll to the end of this practical for a sample solution if you are having difficulties with this.

If your code is anything like mine, the box is moving quite fast. How can we slow it down??

One way is to add in a delay after each move. You can put a Python program to 'sleep' for a specified number of seconds using time.sleep(delaytime). Add this to the loop and experiment with the delay time – let's say between 0.01 and 0.05 (you can try more if you like) seconds to see the effect of delaying. You will need to import the time module.

Scroll to the end of this practical for a sample solution if you are having difficulties with this.

Exercise 4

Write a function called move_block that is passed the block to be moved, and the displacement values. This method should move the block, check for collisions with any walls and adjust the velocity if collisions occur. The function should return the updated displacement values (refer to your lecture notes to see how to return and store values from a function – hint: you will need to use a list).

In your while loop, you will need to call function move_block and pass it the appropriate parameters.

Run your code to ensure it is still behaving correctly now that you have introduced the function.

Scroll to the end of this practical for a sample solution if you are having difficulties with this.

Exercise 5 – Adding a second block.

In your code, create a second block (use yellow for this block) with random starting values and displacements. Use the function move_block to move it around the screen.

Scroll to the end of this practical for a sample solution if you are having difficulties with this.

Exercise 6

Modify the code so that every time a block hits a wall, it changes colour. Use the setColour method to do this – block.setColour(gen_colour()). You should reuse the function gen_colour that you created in practical 7 (you will need to copy the function definition into your current file).

Scroll to the end of this practical for a sample solution if you are having difficulties with this.

SECTION II - Interacting with a graphics program

An interactive graphics program typically allows you to interact via the mouse and the keyboard, through user interface objects such as buttons, menus and text entry fields. We won't be exploring the development of user interfaces in this course, however, we will look at the minimum required in order to have you interact with a graphical application.

Programs with a graphical user interface typically:

Create and display the interface, initialising any objects needed for when the program starts.
Goes into a loop where it waits for the user to do something.

Each time the user does something (creating an 'event') the program checks if it needs to do something for that event. If an 'event handler' has been defined within the program for that event, the event handler is called, with the event passed to the event handler.

The event handler does whatever it should do and returns.

The program is then back into a loop waiting for the next event.

Events can be bound to a specific component of the user interface, or to several components. We are going to start by binding an event to everything in the user interface.

Exercise 7

The graphics module is built on top of a module called Tkinter, which is quite large and complicated, and so the graphics module provides an easy way to access some of the features of Tkinter without having to understand Tkinter. For this practical, we will have to use some feature of Tkinter that are not in the graphics module. As the graphics module imports Tkinter, you should not have to do so.

Let's start by creating a graphics window, and binding an event handler to everything in the window. The event handler will simply print to the Python shell the details of any keys pressed.

Create a Python program that creates a graphics.GraphWin object that is 800x800 pixels with a title 'Week 9 Practical Exercise 7'.

Define a function:

```
def handleKeys(event):
    print('Key pressed:', event.keysym, event.keycode)
```

In your program, if your GraphWin object is called win, add the lines:

```
win.bind_all('<Key>', handleKeys)
win.mainloop()
```

The first line binds the function handleKeys() as an event handler for the event '<Key>' – any key being pressed – for all components of the user interface.

Scroll to the end of this practical for a sample solution if you are having difficulties with this.

Event handler – event information

When an event occurs and an event handler is called to handle the event, the handler is passed a value with information about the event.

For keyboard events, this includes:

- char – the character code as a string
- keysym – a string name for the key
- keycode – the key code
- type – the event type

If the value passed to the event handler is called event, then you access the even information using a dot notation for example, event.keysym, event.keycode.

Bind, Bind_all – events you can handle

The event handler in the example was for the event <Key> - any keyboard key pressed. You can handle specific keys – use the keysym string names that were printed when you pressed a key.

Exercise 8

Add a red game.GameBlock object (assign it to a variable so we can change it later) that is 50 pixels in size and positioned in the centre of the window.

Run your program to check that this works.

Add an if statement to your event handler that checks the event.keysym value against the values for the cursor keys, and moves the block object by 50 pixels in the appropriate direction for each cursor key press. The GameBlock object has a method move() that allows you to move a rectangle object by a specific x and y displacement (x increasing to the right, y increasing downwards). You only have to call the move() method to move the object – the display is updated for you after the call to move.

Scroll to the end of this practical for a sample solution if you are having difficulties with this.

Exercise 9

Modify your solution to exercise 8 so that the keys 'r', 'o', 'y', 'g', 'b', 'i,' 'v' to change the colour of the rectangle to the appropriate rainbow colour.

Note: for some reason 'indigo' is not one of the predefined colour names – use graphics.color_rgb(75,0,130).

Scroll to the end of this practical for a sample solution if you are having difficulties with this.

Optional Practical Work (graphics and game modules)
Sample Solutions
SECTION I & II: Sample Solutions

Exercise 1

```
import graphics
import game

win = graphics.GraphWin("Practical 8 - Exercise 1", 400, 400)
win.setBackground("black")

# Create game block at position x=200, y=200, size of 50 pixels and colour yellow
block = game.GameBlock(200, 200, 50, 50, 'yellow', None, None, 0)

# Draw the block object to the graphics window
block.draw(win)

win.getMouse()
win.close()
```

Exercise 2

```
import graphics
import game
import random

winsize = 300               # size of the window
blocksize = 20              # size of the block
halfsize = blocksize // 2   # half size of the block

win = graphics.GraphWin("Bouncing Block", winsize, winsize)
win.setBackground("white")

# Create a random centre for the block
# Make sure it is at least halfsize from any window 'walls'
xPos = random.randint(halfsize, winsize-halfsize)
yPos = random.randint(halfsize, winsize-halfsize)

# Create game block at random position (xPos, yPos),
# blocksize big and colour red
block = game.GameBlock(xPos, yPos, blocksize, blocksize, 'red', None, None, 0)

# draw the block object to the graphics window
block.draw(win)

# Wait for events
win.mainloop()
```

Exercise 3

```
import graphics
import game
import random
import time

winsize = 300               # size of the window
blocksize = 20              # size of the block
halfsize = blocksize // 2   # half size of the block

# Create our graphics window - winsize big
win = graphics.GraphWin("Bouncing Block", winsize, winsize)
win.setBackground("white")

# Create a random centre for the block
# Make sure it is at least halfsize from any window 'walls'
xPos = random.randint(halfsize, winsize-halfsize)
yPos = random.randint(halfsize, winsize-halfsize)

# Create game block at random position (xPos, yPos),
# blocksize big and colour red
block = game.GameBlock(xPos, yPos, blocksize, blocksize, 'red', None, None, 0)

# Draw the block object to the graphics window
block.draw(win)

# Generate a displacement using random numbers
velocityX = random.randint(-10,10)
velocityY = random.randint(-10,10)

keepLooping = True
while keepLooping:

    # Move the block by the displacement
    block.move(velocityX, velocityY)

    # Test to see whether the block has 'hit' either of the vertical walls
    if (block.getCentreX() < halfsize or block.getCentreX() > winsize-halfsize):

        # If hit, change the sign of the x-component displacement
        velocityX *= -1

    # Test to see whether the block has 'hit' either of the horizontal walls
    if (block.getCentreY() < halfsize or block.getCentreY() > winsize-halfsize):

        # If hit, change the sign of the y-component displacement
        velocityY *= -1

    time.sleep(0.03)

# Wait for events
win.mainloop()
```

Exercise 4

```
import graphics
import game
import random
import time


def move_block(block, velocityX, velocityY):

    # Move the block by the displacement
    block.move(velocityX, velocityY)

    # Test to see whether the block has 'hit' either of the vertical walls
    if (block.getCentreX() < halfsize or block.getCentreX() > winsize-halfsize):

        # If hit, change the sign of the x-component displacement
        velocityX *= -1

    # Test to see whether the block has 'hit' either of the horizontal walls
    if (block.getCentreY() < halfsize or block.getCentreY() > winsize-halfsize):

        # If hit, change the sign of the y-component displacement
        velocityY *= -1

    return [velocityX, velocityY]


winsize = 300               # size of the window
blocksize = 20              # size of the block
halfsize = blocksize // 2   # half size of the block

# Create our graphics window - winsize big
win = graphics.GraphWin("Bouncing Block", winsize, winsize)
win.setBackground("white")

# Create a random centre for the block
# Make sure it is at least halfsize from any window 'walls'
xPos = random.randint(halfsize, winsize-halfsize)
yPos = random.randint(halfsize, winsize-halfsize)

# Create game block at random position (xPos, yPos),
# blocksize big and colour red
block = game.GameBlock(xPos, yPos, blocksize, blocksize, 'red', None, None, 0)

# Draw the block object to the graphics window
block.draw(win)

# Generate a displacement using random numbers
velocityX = random.randint(-10,10)
velocityY = random.randint(-10,10)

keepLooping = True
while keepLooping:

    # Move the block by calling function move_block
    velocityX, velocityY = move_block(block, velocityX, velocityY)

    time.sleep(0.03)
```

```
# Wait for events
win.mainloop()
```

Exercise 5

```
import graphics
import game
import random
import time


def move_block(block, velocityX, velocityY):

    # Move the block by the displacement
    block.move(velocityX, velocityY)

    # Test to see whether the block has 'hit' either of the vertical walls
    if (block.getCentreX() < halfsize or block.getCentreX() > winsize-halfsize):

        # If hit, change the sign of the x-component displacement
        velocityX *= -1

    # Test to see whether the block has 'hit' either of the horizontal walls
    if (block.getCentreY() < halfsize or block.getCentreY() > winsize-halfsize):

        # If hit, change the sign of the y-component displacement
        velocityY *= -1

    return [velocityX, velocityY]


winsize = 300               # size of the window
blocksize = 20              # size of the block
halfsize = blocksize // 2   # half size of the block

# Create our graphics window - winsize big
win = graphics.GraphWin("Bouncing Block", winsize, winsize)
win.setBackground("white")

# Create a random centre for the block
# Make sure it is at least halfsize from any window 'walls'
xPos = random.randint(halfsize, winsize-halfsize)
yPos = random.randint(halfsize, winsize-halfsize)

# Create game block at random position (xPos, yPos),
# blocksize big and colour red
block = game.GameBlock(xPos, yPos, blocksize, blocksize, 'red', None, None, 0)

# Draw the block object to the graphics window
block.draw(win)

# Generate a displacement using random numbers
velocityX = random.randint(-10,10)
velocityY = random.randint(-10,10)
```

```
# Add second block
xPos2 = random.randint(halfsize, winsize-halfsize)
yPos2 = random.randint(halfsize, winsize-halfsize)
anotherBlock = game.GameBlock(xPos2, yPos2, blocksize, blocksize, 'yellow', None,
  None, 0)
velocityX2 = random.randint(-10, 10)
velocityY2 = random.randint(-10, 10)
anotherBlock.draw(win)

keepLooping = True
while keepLooping:

    # Move first block - block
    velocityX, velocityY = move_block(block, velocityX, velocityY)

    # Move second block - anotherBlock
    velocityX2, velocityY2 = move_block(anotherBlock, velocityX2, velocityY2)

    time.sleep(0.03)

# Wait for events
win.mainloop()
```

Exercise 6

```
import graphics
import game
import random
import time


def gen_colour():
    colour = graphics.color_rgb(random.randint(0, 255),
                                random.randint(0, 255),
                                random.randint(0, 255))
    return colour


def move_block(block, velocityX, velocityY):

    # Move the block by the displacement
    block.move(velocityX, velocityY)

    # Test to see whether the block has 'hit' either of the vertical walls
    if (block.getCentreX() < halfsize or block.getCentreX() > winsize-halfsize):

        # If hit, change the sign of the x-component displacement
        velocityX *= -1

        # Change block to random colour
        block.setColour(gen_colour())

    # Test to see whether the block has 'hit' either of the horizontal walls
    if (block.getCentreY() < halfsize or block.getCentreY() > winsize-halfsize):

        # If hit, change the sign of the y-component displacement
        velocityY *= -1

        # Change block to random colour
        block.setColour(gen_colour())

    return [velocityX, velocityY]



winsize = 300               # size of the window
blocksize = 20              # size of the block
halfsize = blocksize // 2   # half size of the block

# Create our graphics window - winsize big
win = graphics.GraphWin("Bouncing Block", winsize, winsize)
win.setBackground("white")

# Create a random centre for the block
# Make sure it is at least halfsize from any window 'walls'
xPos = random.randint(halfsize, winsize-halfsize)
yPos = random.randint(halfsize, winsize-halfsize)

# Create game block at random position (xPos, yPos),
# blocksize big and colour red
block = game.GameBlock(xPos, yPos, blocksize, blocksize, 'red', None, None, 0)
```

```
# Draw the block object to the graphics window
block.draw(win)

# Generate a displacement using random numbers
velocityX = random.randint(-10,10)
velocityY = random.randint(-10,10)

# Add second block
xPos2 = random.randint(halfsize, winsize-halfsize)
yPos2 = random.randint(halfsize, winsize-halfsize)
anotherBlock = game.GameBlock(xPos2, yPos2, blocksize, blocksize, 'yellow', None,
  None, 0)
velocityX2 = random.randint(-10, 10)
velocityY2 = random.randint(-10, 10)
anotherBlock.draw(win)

keepLooping = True
while keepLooping:

    # Move first block - block
    velocityX, velocityY = move_block(block, velocityX, velocityY)

    # Move second block - anotherBlock
    velocityX2, velocityY2 = move_block(anotherBlock, velocityX2, velocityY2)

    time.sleep(0.03)

# Wait for events
win.mainloop()
```

Exercise 7

```
import graphics
import game


def handleKeys(event):
    print('Key pressed:', event.keysym, event.keycode)


winsize = 800               # size of the window
blocksize = 20              # size of the block
halfsize = blocksize // 2   # half size of the block

# Create our graphics window - winsize big
win = graphics.GraphWin("Week 9 Practical Exercise 7", winsize, winsize)
win.setBackground("white")

win.bind_all('<Key>', handleKeys)

win.mainloop()
```

Exercise 8

```
import graphics
import game


def handleKeys(event):
    print('Key pressed:', event.keysym, event.keycode)

    if event.keysym == 'Left':
        block.move(-50, 0)
    elif event.keysym == 'Right':
        block.move(50, 0)
    elif event.keysym == 'Up':
        block.move(0, -50)
    elif event.keysym == 'Down':
        block.move(0, 50)


winsize = 800               # size of the window
blocksize = 50              # size of the block
halfsize = blocksize // 2   # half size of the block

# Create our graphics window - winsize big
win = graphics.GraphWin("Week 9 Practical Exercise 7", winsize, winsize)
win.setBackground("white")

xPos = 400
yPos = 400

block = game.GameBlock(xPos, yPos, blocksize, blocksize, 'red', None, None, 0)
block.draw(win)

win.bind_all('<Key>', handleKeys)

win.mainloop()
```

Exercise 9

```
import graphics
import game


def handleKeys(event):
    print('Key pressed:', event.keysym, event.keycode)

    if event.keysym == 'Left':
        block.move(-50, 0)
    elif event.keysym == 'Right':
        block.move(50, 0)
    elif event.keysym == 'Up':
        block.move(0, -50)
    elif event.keysym == 'Down':
        block.move(0, 50)
    elif event.keysym == 'r':
        block.setColour('red')
    elif event.keysym == 'o':
        block.setColour('orange')
    elif event.keysym == 'y':
        block.setColour('yellow')
    elif event.keysym == 'g':
        block.setColour('green')
    elif event.keysym == 'b':
        block.setColour('blue')
    elif event.keysym == 'i':
        block.setColour(graphics.color_rgb(75,0,130))
    elif event.keysym == 'v':
        block.setColour('violet')



winsize = 800               # size of the window
blocksize = 50              # size of the block
halfsize = blocksize // 2   # half size of the block

# Create our graphics window - winsize big
win = graphics.GraphWin("Week 9 Practical Exercise 7", winsize, winsize)
win.setBackground("white")

xPos = 400
yPos = 400

block = game.GameBlock(xPos, yPos, blocksize, blocksize, 'red', None, None, 0)
block.draw(win)

win.bind_all('<Key>', handleKeys)

win.mainloop()
```

Please make sure you save and keep all of your practical and assignment work. Please ask your supervisor if you are having difficulties doing so.

End of practical 8.