



## BREAKOUT ACTIVITIES

The focus on these activities is on getting used to the Python programming environment and in particular sequential flow of control.

### BREAKOUT 1.1: Automated Teller Machine (ATM) Menu System

The Python program below displays the ATM menu shown on the right hand side.

```
# This code displays the main ATM menu
print("\t|-----|")
print("\t|\t LCCS BANK LIMITED\t|")
print("\t|\t ATM Main Menu\t\t|")
print("\t|\t\t\t\t\t|")
print("\t|\t1. Balance Enquiry\t|")
print("\t|\t2. Cash Lodgement\t|")
print("\t|\t3. Cash Withdrawal\t|")
print("\t|\t4. Cash Transfer\t|")
print("\t|\t5. Change PIN\t\t|")
print("\t|\t6. Other Services\t|")
print("\t|\t\t\t\t\t|")
print("\t|\t7. Exit\t\t\t\t|")
print("\t|-----|")
print("\t|\t\t\t\t\t|")
print("\t| CHOOSE AN OPTION >> \t\t|")
print("\t|\t\t\t\t\t|")
print("\t|-----|")
print("")
```

```

LCCS BANK LIMITED
ATM Main Menu

1. Balance Enquiry
2. Cash Lodgement
3. Cash Withdrawal
4. Cash Transfer
5. Change PIN
6. Other Services

7. Exit

CHOOSE AN OPTION >>

```

### Suggested Activities

1. Key in the above program and .....
  - Suggest and make changes to the program (e.g. add/remove/edit a menu option)
  - Discuss traditional console menus vs. GUI/touch screen interfaces
  - Discuss possible logic behind the options
2. Design and implement a menu for some other application of your choice. (For this exercise you will need to think of a system from an end-user's perspective.)

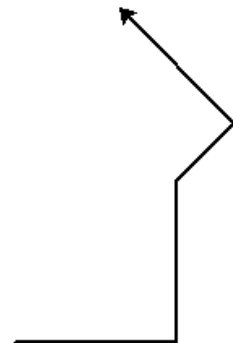
## BREAKOUT 1.2: Turtle Graphics

Turtle graphics is a popular way for introducing programming to novice programmers. It was part of the original Logo programming language developed by Wally Feurzig and Seymour Papert in 1966.

The movements of the turtle graphic object can be compared to the movements that you would see if you were looking down at a real turtle inside a rectangular shaped box. The program below causes the shape/pattern shown to the right to be drawn out on the screen.

```
1. import turtle
2.
3. # Create a turtle object and call it 'pen'
4. pen = turtle.Turtle()
5.
6. # Create a window for the turtle
7. window = turtle.Screen()
8.
9. pen.forward(100)    # move forward 100 units
10. pen.left(90)       # turn left by 90 degrees
11. pen.forward(100)
12. pen.right(45)
13. pen.forward(50)
14. pen.left(90)
15. pen.forward(100)
16.
17. # Keep looping until window is closed
18. window.mainloop()
```


**Program Listing**



**Shape**

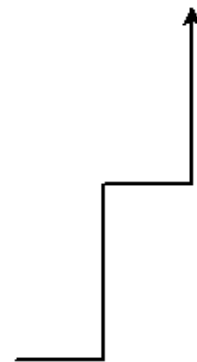
### **Program Explanation**

- Line 1 tells Python to import a *library* called 'turtle'. A library can be thought of as an external Python program that contains useful code. `import` is a Python keyword. When a library is imported into a program the functionality of that library can then be used in that program.
- Line 4 tells Python to create the turtle graphic object and refer to it as 'pen'
- Line 5 tells Python to create the window in which the turtle's movements can be seen.
- The commands on lines 9 to 15 inclusive (these lines are highlighted in bold) instruct Python to move and turn the turtle.

- Line 18 tells Python to keep displaying the window until it is closed by the end-user. Students should be reminded to close the turtle window once they have finished running your program. The window can be closed by clicking on  in the top right corner.

### Suggested Activities

1. Read lines 9-15 of the program and see if you can figure out how the shape is created
2. Type the program in and run it
3. Insert comment on lines 11 – 15 inclusive. (Lines 9 and 10 are already commented.)
4. Move the lines between 9 and 15 around into different order the lines and see if you can explain the change in output
5. Experiment with the numbers used on lines 9 – 15 until you understand what the different numbers mean.
6. Modify the program so that it displays the shape shown to the right



Some of the more common movement commands supported by `turtle` are outlined below.

Command	Explanation
<code>forward(n)</code>	This command moves the turtle forward by <code>n</code> units from whatever position the turtle is facing at the time the command is issued
<code>backward(n)</code>	When this command is issued it moves the turtle in the opposite direction to whatever direction the turtle is facing. The turtle is moved by <code>n</code> units from its current position.
<code>right(angle)</code>	This command turns the turtle in a rightwards direction. The amount of turn is specified by the programmer using <code>angle</code> .
<code>left(angle)</code>	This command turns the turtle in a leftwards direction. The amount of turn is specified by the programmer using <code>angle</code> .

The syntax to use any of the above commands is as follows - note the dot in the middle.

**<turtle-name>.<command>**

In this example the programmer's name for the turtle is `pen`.

## Further Activities

The default starting position for the turtle is the centre of the screen. It is up to the programmer to keep a track of the position of the turtle on the screen and the direction it is facing. The best way to learn how to use turtles is to experiment. The following exercise might help.

1. Match the code blocks below to the corresponding shape.

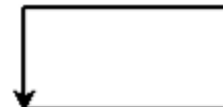
```
pen.forward(100)
pen.right(90)
pen.forward(50)
pen.right(90)
pen.forward(100)
pen.right(90)
pen.forward(50)
```



```
pen.forward(100)
pen.left(60)
pen.forward(100)
pen.left(60)
pen.forward(100)
```



```
pen.forward(100)
pen.left(120)
pen.forward(100)
pen.left(120)
pen.forward(100)
```



```
pen.forward(100)
pen.left(90)
pen.forward(50)
pen.left(90)
pen.forward(100)
pen.left(90)
pen.forward(50)
```



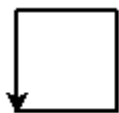
2. Now demonstrate that your answers are correct!

Do this by keying in and running each of the separate code blocks.

3. The commands listed below can be used to change the appearance of turtle objects

Command	Explanation
<code>shape(s)</code>	This command sets the appearance of the turtle object to be whatever shape is specified by <code>s</code> . Valid values are <code>arrow</code> , <code>turtle</code> , <code>circle</code> , <code>square</code> , <code>triangle</code> and <code>classic</code> . (Use quotation marks.) The arrow shape is the default.
<code>hideturtle()</code>	When this command is used it makes the turtle object disappear from the output screen.
<code>showturtle()</code>	This command makes the turtle visible again.
<code>color(c)</code>	This command sets the colour of the lines drawn by the turtle to be the colour specified by <code>c</code> . Try different values e.g. <code>red</code> , <code>blue</code> , <code>green</code> . (Don't forget to use quotation marks either side of the named colour.)
<code>pensize(n)</code>	This command sets the line thickness of the line drawn by turtle movements. The value of <code>n</code> can be any number from 1 to 10 where 1 is the thinnest and 10 is the thickest. Try it!

Write a Python program to display the shapes shown below.



A 50x50 square



A 50x100 red rectangle



A vertical blue line of length 100 units and thickness 5 units



The letter T in red (pen is hidden)

Can you come up with more than one solution for each shape? Compare and discuss your solutions with your classmates.

## **BREAKOUT 1.2: Games Programming with pygame**

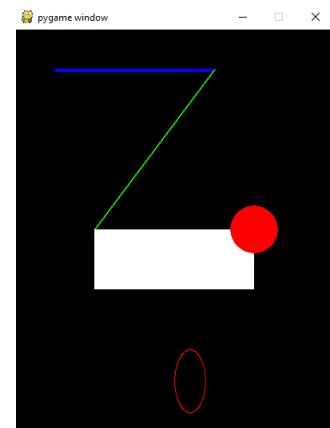
*pygame*<sup>1</sup> is a free and open source Python library useful for games programming. As it does not come with the standard Python installation, *pygame* needs to be installed separately.

When the program shown below is run it causes the output window illustrated to the right to be displayed. The output window contains 5 different shapes – a blue horizontal line, a green diagonal line, a white rectangle, a red circle and red ellipse. These shapes are drawn in response to the commands on lines 15 to 19.

Read the code carefully – focus your attention on lines 15 to 19 (highlighted in bold) - and see if you can guess which line of code is responsible for drawing which shape.

```
1. import pygame
2.
3. # set up pygame
4. pygame.init()
5. window = pygame.display.set_mode((400, 500)) # (width, height)
6.
7. # set up the colours
8. BLACK = (0, 0, 0)
9. RED = (255, 0, 0)
10. GREEN = (0, 255, 0)
11. BLUE = (0, 0, 255)
12. WHITE = (255, 255, 255)
13.
14. # draw some shapes
15. pygame.draw.line(window, BLUE, (50, 50), (250, 50), 4)
16. pygame.draw.line(window, GREEN, (100, 250), (250, 50), 2)
17. pygame.draw.rect(window, WHITE, (100, 250, 200, 75))
18. pygame.draw.circle(window, RED, (300, 250), 30, 0)
19. pygame.draw.ellipse(window, RED, (200, 400, 40, 80), 1)
20. # update the window display
21. pygame.display.update()
```

***Program Listing***



***Output Window***

The individual lines of code are explained on the next page.

Programmers are often presented with code they have not written themselves and need to figure out for themselves what the code does. One tried and trusted method used by programmers to familiarise themselves with 'new' code is to 'play with it' i.e. make small incremental changes to build up an understanding. Try the following suggested activities.

---

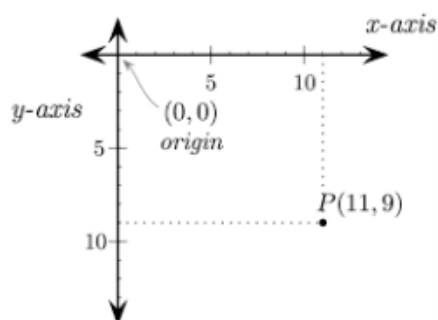
<sup>1</sup> <https://www.pygame.org>

## Suggested Activities

1. Key in the full program and make sure it runs properly.
2. Devise your own theories about the code. For example, you might suspect that line 15 draws the blue horizontal line. In order to test this theory, you could comment out lines 16, 17, 18 and 19 and then run your program to see if you are correct.  
This process should be repeated until you have confirmed your understanding of which line of code is responsible for which shape.
3. Experiment by rearranging lines 15-19 into different orders. Each time you jumble them around, run your program to see if they make any difference to the output display.
4. Change the code so that the shapes are displayed in different colours
5. Modify the numbers used in the commands used to draw the lines and the rectangle (lines 15, 16 and 17). Can you figure out what the numbers mean?

## Co-ordinate System

The diagram below explains the window co-ordinate system used by `pygame`.



**window co-ordinate system used by `pygame`**

The co-ordinates of the four corners of a window having width,  $w$  and height,  $h$  are:

- $(0, 0)$  top left
- $(w, 0)$  top right
- $(0, h)$  bottom left
- $(w, h)$  bottom right

## Game Loop

You may have noticed that the output window does not close. To fix this problem you will need to make two changes

- a) Modify line 1 as shown
- b) Add the (game loop) code shown here to the end of the program listing.

```
1. import pygame, sys
```

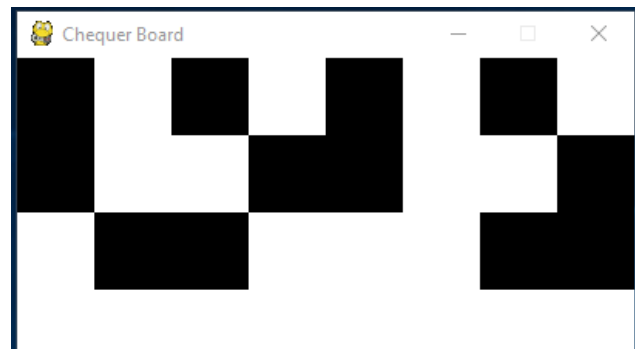
```
22.  
23. # run the game loop  
24. while True:  
25.     for event in pygame.event.get():  
26.         if event.type == 12:  
27.             pygame.quit()  
28.             sys.exit()
```

### Further pygame activities

1. The commands on lines 17, 18 and 19 draw a rectangle, circle and ellipse respectively. Modify the numbers used inside the brackets. Can you figure out what the numbers mean?
2. Open the file “Page 17 - Display ChequerBoard (problem)”. Study the program code carefully. When the program is run, it displays the first three rows of the pattern as shown.

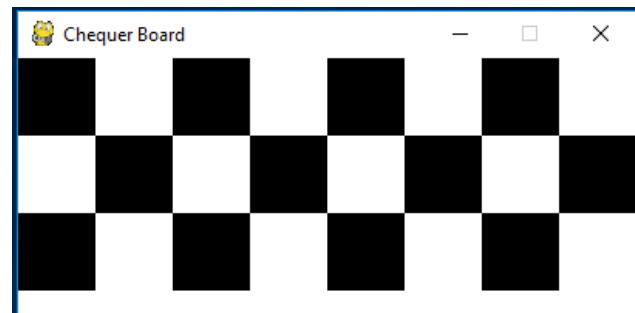
The background is painted white by the `fill` command, so, in actual fact, the program draws four black squares on each of the three rows.

Each individual black square is drawn in response to the command `pygame.draw.rect`. The squares are 50×50 units in size.



There is a problem however.

The programmer had intended the program to display the chequer board pattern shown here to the right.



Modify the program so that it displays the first three rows of a proper chequer board pattern. Can you make the necessary changes?

How might the program differ if the background was painted BLACK instead of white?

3. Calculate the co-ordinates of the centre of a 400x500 window.  
Can you generalise this calculation with a formula that would work for a window of any size?
4. Write a program to display a circle centred on the output window. (You choose the size!)  
Generalise your solution so that it works for any window size.