BREAKOUT ACTIVITIES

BREAKOUT 2.1: Turtle Graphics

Variables, assignments and operations can be used to dynamically vary the dimensions of turtle shapes.

1. Predict what pattern would be generated by the each of the following program listings.

```python
import turtle

# Create a turtle object and call it 'pen'
pen = turtle.Turtle()

# Create a window for the turtle
window = turtle.Screen()

pen.hideturtle() # hide the turtle
pen.color("red") # set the pen colour to red

lineLength = 50
pen.forward(lineLength)
pen.left(90)
lineLength = lineLength + 50
pen.forward(lineLength)
pen.left(90)
lineLength = lineLength + 50
pen.forward(lineLength)
pen.left(90)
lineLength = lineLength + 50
pen.forward(lineLength)

# Allow the window to be closed
window.mainloop()
```

Program Listing 1

```python
import turtle

#Create a turtle object and call it 'pen'
pen = turtle.Turtle()

# Create a window for the turtle
window = turtle.Screen()

pen.hideturtle() # hide the turtle
pen.color("red") # set the pen colour to red

lineLength = 50
pen.forward(lineLength)
pen.left(90)
pen.forward(lineLength + 50)
pen.left(90)
pen.forward(lineLength + 50)
pen.left(90)
pen.forward(lineLength + 50)

# Allow the window to be closed
window.mainloop()
```

Program Listing 2

Explain the main difference between the two programs shown above

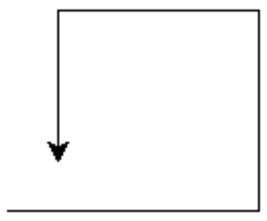
____

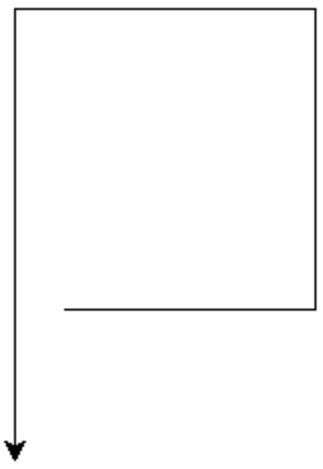
____

____

2. Match the code block below with the correct shape (drawn to scale).

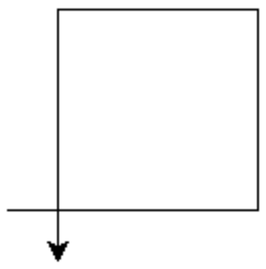
```
x=50
y=25
pen.forward(2*x+y)
pen.left(90)
pen.forward(x+2*y)
pen.left(90)
pen.forward(x+2*y)
pen.left(90)
pen.forward(2*x+y)
```



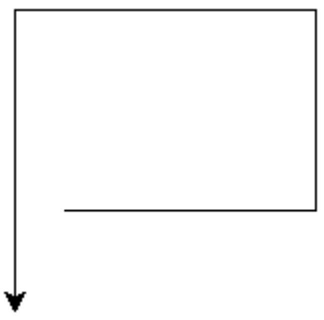


```
x=50
y=25
pen.forward(2*x+y)
pen.left(90)
y=2*y
pen.forward(x+y)
pen.left(90)
pen.forward(x+2*y)
pen.left(90)
pen.forward(2*x+y)
```

```
x=50
y=25
pen.forward(2*x+y)
pen.left(90)
x=2*x
pen.forward(x+2*y)
pen.left(90)
pen.forward(x+2*y)
pen.left(90)
pen.forward(2*x+y)
```

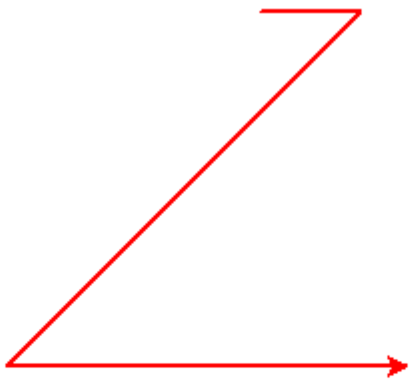


```
x=50
y=25
pen.forward(2*x+y)
pen.left(90)
pen.forward(x+2*y)
pen.left(90)
pen.forward(x+2*y)
pen.left(90)
x=2*y
pen.forward(x+y)
```



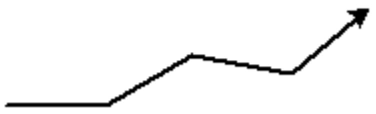
3. Modify the program below so that the angle size is increased by $135°$ before each turn. The modified program should display the pattern illustrated to the right

```
import turtle

#Create a turtle object and call it 'pen'
pen = turtle.Turtle()

# Create a window for the turtle
window = turtle.Screen()

pen.pensize(2) # set the line thickness to 2
pen.color("red") # set the pen colour to red

angle = 90
lineLength = 50
pen.forward(lineLength)
pen.left(angle)
lineLength = lineLength + 50
pen.forward(lineLength)
pen.left(angle)
lineLength = lineLength + 50
pen.forward(lineLength)
pen.left(angle)
lineLength = lineLength + 50
pen.forward(lineLength)

# Allow the window to be closed
window.mainloop()
```



4. Implement the following pseudo-code in Python to display the pattern illustrated

```
Initialise a variable called angle to 30
Move forward by 50 units
Turn left by angle degrees
Increase the angle by $10°$
Move forward by 50 units
Turn right by angle degrees
Increase the angle by $10°$
Move forward by 50 units
Turn left by angle degrees
Increase the angle by $10°$
Move forward by 50 units
```

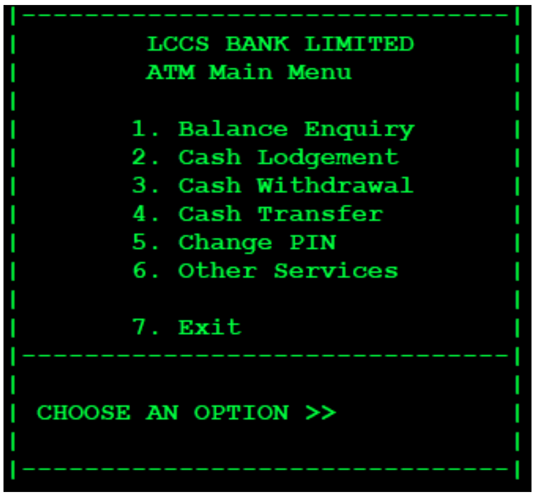


5. Write a Python program to display five lines of random length. Each line should be joined to the next at a randomly sized angle.

BREAKOUT 2.2: Automated Teller Machine (ATM) Menu System

Recall from Section 1 the menu system for our fictional LCCS bank.

Use the knowledge you have gained so far to convert the pseudo-code shown to the right of the menu below into Python.



```
|--------------------------------|
|         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 >>            |
|                                |
|--------------------------------|
```

```
Display a welcome message
Initialise a variable called balance to 123.45
Display the value of balance
Prompt the user to enter an amount to lodge
Increase the balance by the amount entered
Display the value of balance
Prompt the user to enter an amount to withdraw
Decrease the balance by the amount entered
Display the value of balance
```

Hint: You will need to consider what variables you will need as well as their datatype.

Log your thoughts.
What did you find most/least challenging about this task?

BREAKOUT 2.3: Data Processing (average height)

The short program shown below was designed to calculate the mean height from five values entered by an end-user. Study the program carefully until you are satisfied you understand how it works.

```
# A program to calculate the average height of 5 people
print("Average height calculator")
print("========================")

# Read in the 5 values
h1 = int(input("Enter first height (cm): "))
h2 = int(input("Enter second height (cm): "))
h3 = int(input("Enter third height (cm): "))
h4 = int(input("Enter fourth height (cm): "))
h5 = int(input("Enter fifth height (cm): "))

# Calculate the average height
avgHeigth = h1+h2+h3+h4+h3/5

# Display the result
print("The average height is ", avgHeigth, "cm")
```

The test data shown below threw up some differences between expected and actual outputs. Although the program is syntactically correct it contains at least one semantic error (bug).

Input Values (cm)					Expected Output	Actual Output
h1	h2	h3	h4	h5		
150	160	170	180	190	170.0 cm	694.0 cm
190	172	172	178	187	179.8 cm	746.4 cm
171	175	169	182	178	175.0 cm	730.8 cm

Log your thoughts.
What is your opinion of the above program?
In what way(s) could the program be improved?

Suggested Activities

1. Key in the full program and make sure it runs without any syntax errors.

2. Experiment by rearranging lines 6-10 into different orders. Each time you make a change, run your program to see if they make any difference to the output display.

3. Fix the bug(s)

4. Modify the code so that it can accept decimal values for height as well as whole numbers. (Make sure to round your result.)

5. Rearrange the lines below into a program that does the same thing.
   Note, only five of the lines (excluding comments) are needed but two of these will be needed more than once.

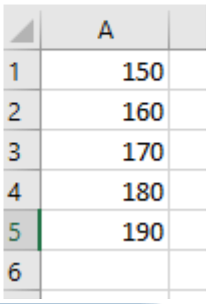
```
# Calculate the average

h1 = int(input("Enter first height (cm): "))

# Display the result

height = input("Enter height (cm): ")

avgHeigth = totalHeight/5

totalHeight = 0

print("-------------------------")

height = float(input("Enter height (cm): "))

# A program to calculate the average height of 5 people

print("The average height is "+str(round(avgHeigth,2))+"cm")

totalHeight = totalHeight + height

print("Average height calculator")

print("The average height is", round(avgHeigth,2), "cm")

# Read in the 5 values

avgHeigth = (h1+h2+h3+h4+h5)/5
```

6. Add a line of code to display the result in feet and inches as well as centimetres.
   $(1cm = 0.393701\ inches)$

Further Activities

By this stage you are more than likely getting tired of having to enter 5 different values for height every time you run your program.

Wouldn't it be nice if you could enter the values into a spreadsheet and every time you run your program it would read the file and calculate the mean based on the 5 values contained in the file?
This is exactly what the following program does.



	A
1	150
2	160
3	170
4	180
5	190
6	

```
# A program to calculate the average height of 5 people
# The heights are stored in a file called 'heights.csv'

heightFile = open("heights.csv","r") # Open the file

totalHeight = 0 # Initialise a running total for all the heights to zero

height = float(heightFile.readline())   # read the first value
totalHeight = totalHeight + height      # keep a running total

height = float(heightFile.readline())   #
totalHeight = totalHeight + height      #

height = float(heightFile.readline())   #
totalHeight = _____________ + height    #

height = float(heightFile.readline())   #
totalHeight = totalHeight + __________  #

height = float(______________________)  #
totalHeight = totalHeight + height      #

# Calculate the average
avgHeigth = __________________

# Display the result
print("The average height is "+str(round(avgHeigth,2))+"cm")
print("The average height is", round(avgHeigth*0.393701,2), "inches")

heightFile.close()
```

Before reading the detailed explanation of the program on the next page see if you can fill in the blanks (and complete the comments)

The file `heights.csv` was created in a spreadsheet and saved into the same folder as the Python source program. This is called the *runtime folder*.

Each of the five lines in the data file contain a numeric value that represents a person's height in centimetres.

Program Explanation

- Line 4 of the program opens a data file called `heights.csv` in read mode. This tells Python that the file will be used for reading (as opposed to writing) purposes. The variable `heightFile` is the program's reference to this file. This is called the *file pointer* (also referred to as a *file handle*). Any operations on the file such as reading the file must use this file handle.

  It is useful to think of a file pointer as an imaginary finger. When a data file is initially opened in read mode the pointer is positioned at the start of that file.

- Line 6 initialises a variable called `totalHeight` to zero. This variable will be used to store the sum of all the height values.

- Line 8 tells Python to read a line from the data file, convert the result to a floating point number and store the resulting value in a variable called `height`. (There's lots going on here so make sure you understand this line as it is a very common type of pattern in Python programming.)

  Every time the program reads a line from the file, the file pointer is moved to the start of the next line in the data file. This is a subtle side effect of the `readline` command.

- Line 9 adds the current value of `height` to the value stored in the variable `totalHeight` (which was initialised to zero on line 6). In this case, the answer will be the same as the value in `height`. This answer is then stored in `totalHeight`.

- This pattern of reading the next line from the data file and adding the height value to the running total is repeated four times i.e. once for each line in the data file.

- Line 24 computes the average and lines 27 and 28 display the result.

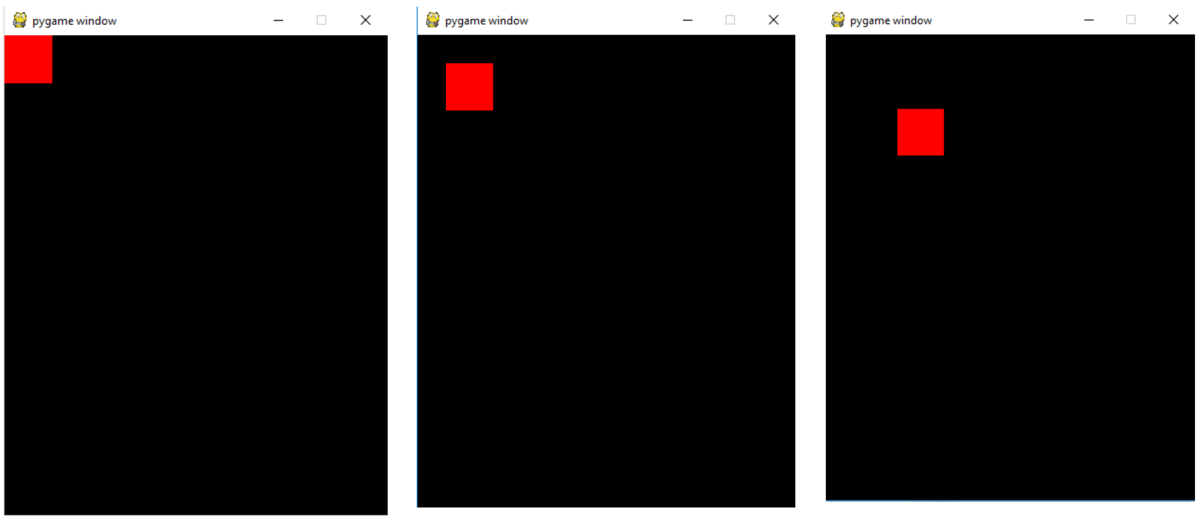
What one question would I still like to ask in relation to this example?

BREAKOUT 2.4: Games Programming with `pygame` (Animation)

In this activity we will use our knowledge of variables to create the illusion of a 50x50 unit square block moving from a starting position on the top left hand corner of a window towards (and beyond) the bottom right hand corner of the same window.

Before writing any code it is good practice to first design a solution. Start by analysing the problem – make sure you understand what is required. It is always helpful to try to visualise what the final running program will look like.

The three screenshots below depict a red block moving diagonally (downwards and to the right) towards the bottom right corner.



Before writing any code it is often helpful to ask probing questions such as ….

Have I done something similar before? Can the task be decomposed (broken down)?

What Python commands are there to meet my need? How do these command work?

Do I need variables?

Can I find a solution for a similar problem using the internet? What would I be looking for?

STUDENT TIP
Students should be encouraged to design solutions (with pen and paper) before writing any code.

In this case we will decompose the problem into 2 parts:

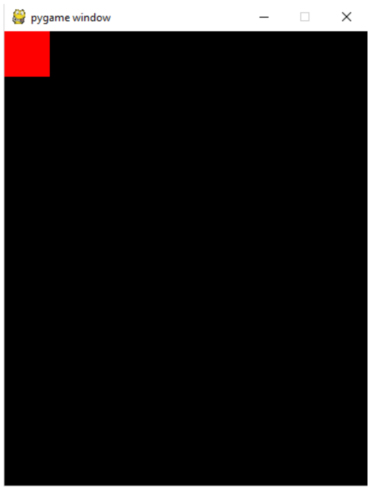
a) Display a 50x50 block in the top left hand corner of a pygame window

b) Create the required illusion of movement from top left towards bottom right

The solution to part a) along with the output window is presented in the program listing below. (You should key in the program and make sure it runs without any syntax errors.)

```
1.  import pygame, sys
2.
3.  # set up pygame
4.  pygame.init()
5.
6.  window = pygame.display.set_mode((400, 500))
7.
8.  # set up the colours
9.  BLACK = (0, 0, 0)
10. RED = (255, 0, 0)
11. GREEN = (0, 255, 0)
12. BLUE = (0, 0, 255)
13. WHITE = (255, 255, 255)
13.
15. # draw a 50x50 square at (0,0)
16. pygame.draw.rect(window, RED,(0, 0, 50, 50))
17.
18. # update the window display
19. pygame.display.update()
20.
21. # run the game loop
22. while True:
23.     for event in pygame.event.get():
24.        if event.type == 12:
25.            pygame.quit()
26.            sys.exit()
```

Program Listing



Output Window

Program Explanation

- It should be possible to understand most of the code from the activity at the end of section 1 (if not you should revisit section 1)
- Line 16 is the key line – this line uses the command `pygame.draw.rect` to draw the 50x50 rectangle (i.e. a square) at position (0,0) in the display window. The 'clever' bit is recognising that by using (0, 0) as the co-ordinates the block will be positioned at the top left of the display window as required.
- Line 18 updates the display causing the red square to actually appear in the display window.

The solution to part b) of the problem - presented below - is slightly more intricate and requires a little more in depth knowledge of the ‘game loop’.

```
1.  import pygame, sys, time
2.
3.  # set up pygame
4.  pygame.init()
5.
6.  window = pygame.display.set_mode((400, 500))
7.
8.  # set up the colours
9.  BLACK = (0, 0, 0)
10. RED = (255, 0, 0)
11. WHITE = (255, 255, 255)
12.
13. # draw a 50x50 red square
14. x = 0 # x-value of top left co-ordinate
15. y = 0 # y-value of top left co-ordinate
16. pygame.draw.rect(window, RED,(x, y, 50, 50))
17.
18. # update the window display
19. pygame.display.update()
20.
21. # run the game loop
22. while True:
23.     for event in pygame.event.get():
24.         if event.type == 12:
25.             pygame.quit()
26.             sys.exit()
27.
28.     x = x + 5 # add 5 to the top position
29.     y = y + 5 # add 5 to the left position
30.     pygame.draw.rect(window, RED,(x, y, 50, 50))
31.
32.     pygame.display.update() # show the block
33.     window.fill(BLACK) # paint the entire window black
34.     time.sleep(0.02) # pause for 2 milliseconds
```

The ‘trick’ here is to create the illusion movement by displaying the block it in a rapid sequence of different positions. The co-ordinates of the block will be needed.

We introduce two variables `x` and `y` to store the co-ordinates of the top left corner of the block. The variables are initialised to 0 on lines 14 and 15 respectively.

Our illusion is achieved by changing the value of each variable inside the game loop (lines 28 and 29) before redrawing the block and updating the display.

Line 28 increases the `x` position by 5 – this causes the horizontal movement.

Line 29 increases the `y` position by 5 – this causes the vertical movement.

Experiment! Try making the following changes.

1. Comment out line 33. What happens? What is the purpose of this line of code?

2. Change the fill colour on line 33 from BLACK to RED. Explain what happens.

3. Comment out line 28. What happens? Can you explain why?

4. Change the offset amount from 5 to 10 on line 29. Explain any changes in the way the program behaves.

Further Activities

1. Adapt your program so that the block moves as follows:

- from the top left corner to the bottom left corner
- from the top left corner to the top right corner
- from a different starting position in one dimension e.g. top right to bottom right
- from a different starting position in two dimensions e.g. bottom left to top right
- from a random starting position to a random ending position

2. Write a program to make a ball (circle) appear to move through the screen.