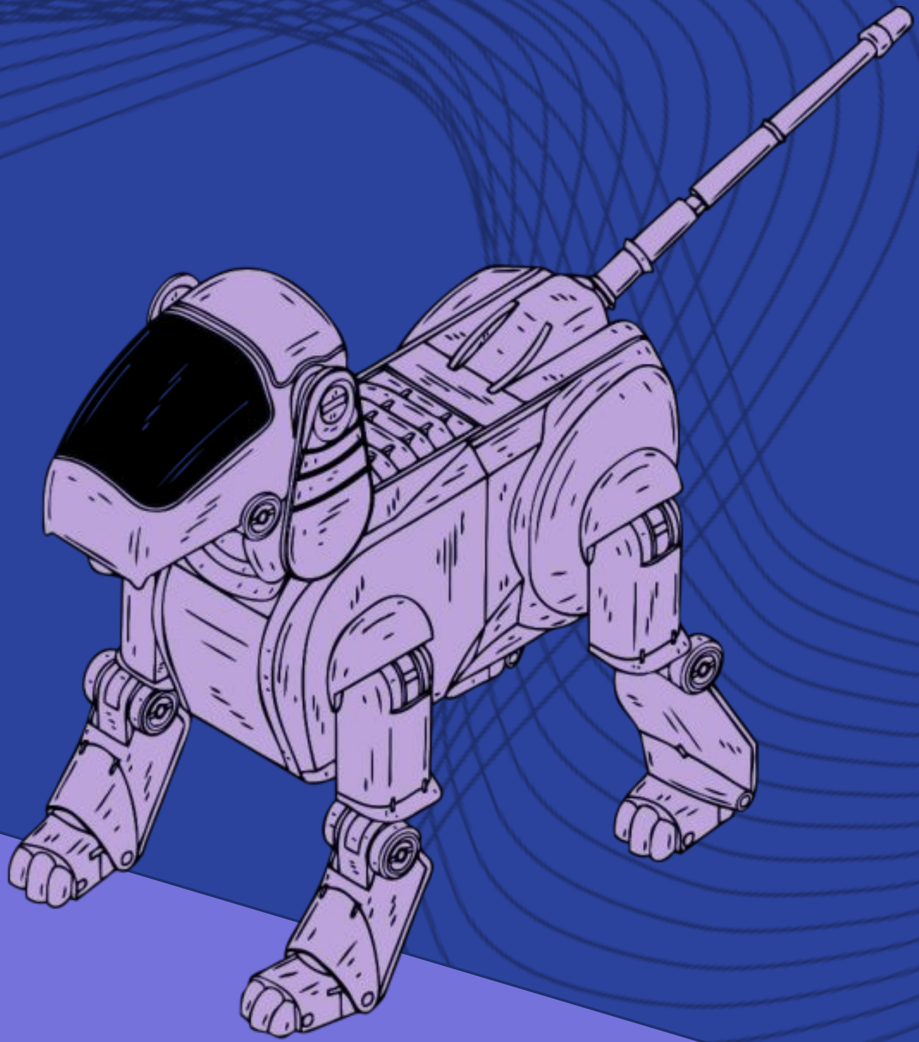
Caterpillar Game

Level 3 – Python

At Play



cair
4 YOUTH

Introduction to Coding

What is Python?

- Python is a popular general-purpose programming language that can be used for a wide variety of applications.
- Python is an interpreted, interactive, object-oriented programming language. It incorporates modules, exceptions, dynamic typing, very high-level dynamic data types, and classes.

Where to access Python?

- https://www.python.org/downloads/ - downloadable app for PCs (allows you to save files directly onto a computer)
- https://trinket.io/ - online version (allows you to create an account, much like scratch)

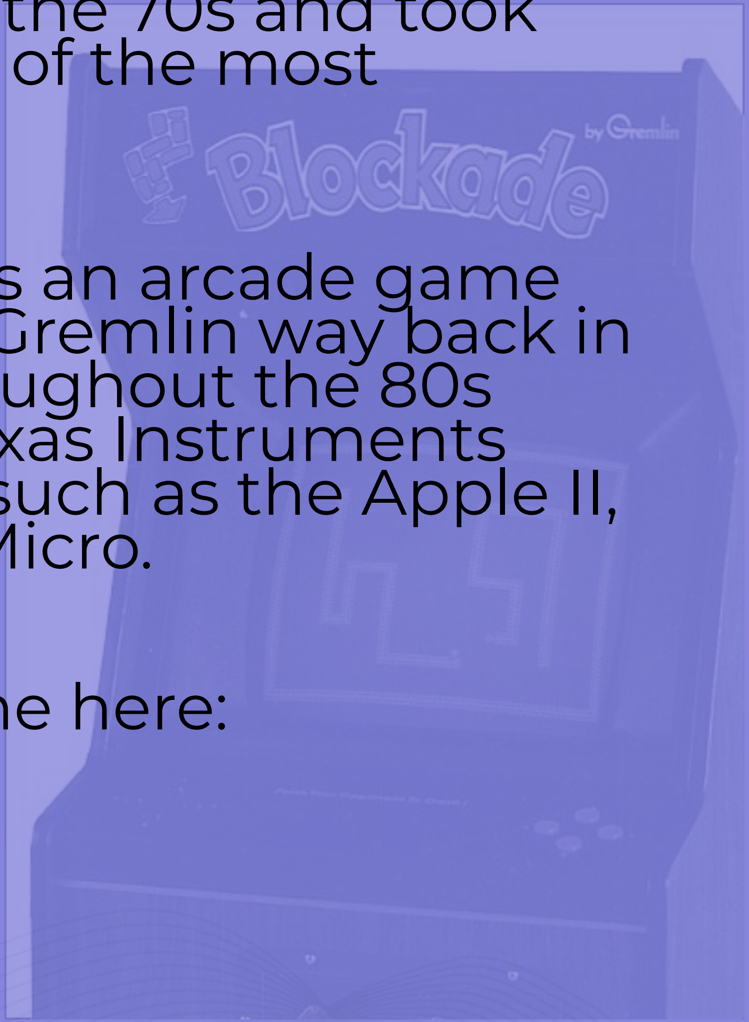
cair
4 YOUTH

Introduction

“Snake” was a game introduced in the 70s and took the world by storm, becoming one of the most addictive games of the time.

The very first Snake-type game was an arcade game called Blockade. It was created by Gremlin way back in 1976. TWO. Its popularity grew throughout the 80s when it made its way onto early Texas Instruments calculators, and home computers such as the Apple II, the Commodore 64 and the BBC Micro.

Have a go at an original snake game here:
https://playsnake.org



cair
4 YOUTH

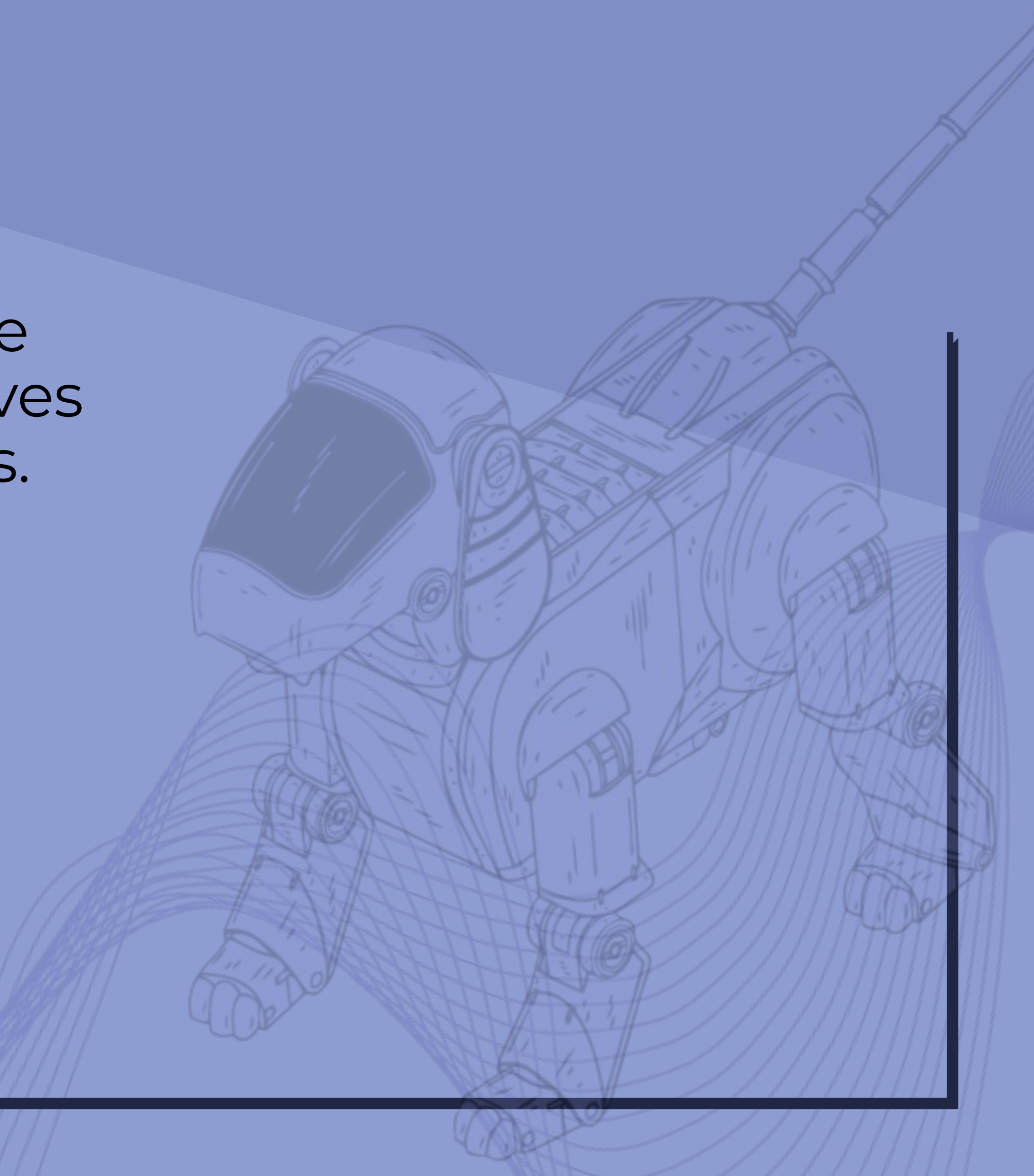
Task

Your task is to create a game that emulates the snake game you have seen on the previous slide by using Turtle graphics in **Python**.

Process

Your code should…

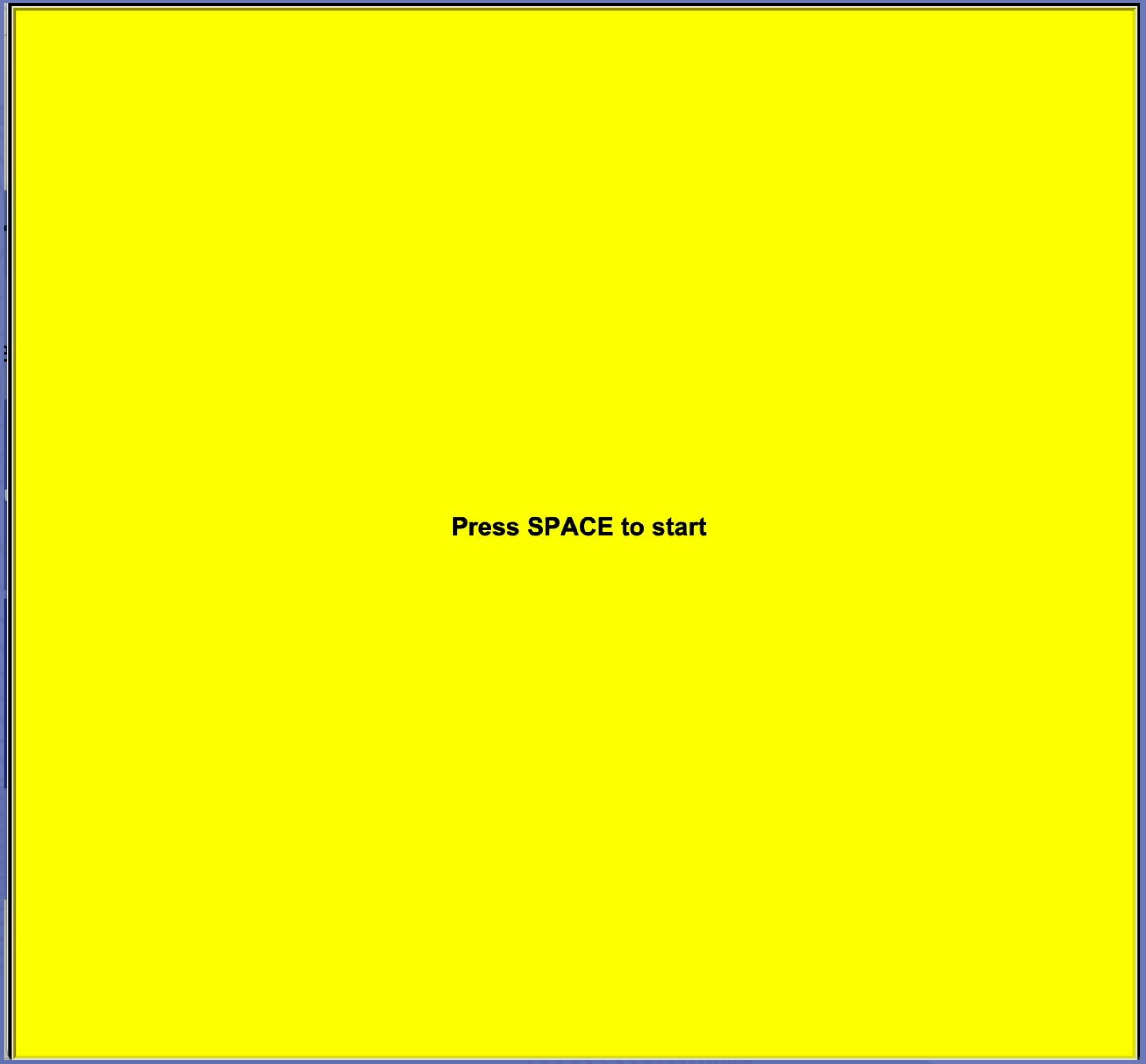
- Be written with subroutines.
- Enable the user to move around the screen as a caterpillar to collect leaves to grow its tail, using the arrow keys.
- Display a score to the user.



cair
4 YOUTH

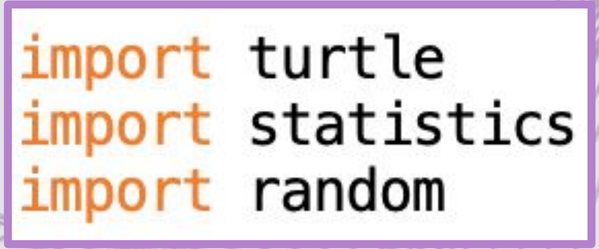
What the final game should look like...

As you can see from the video, if the caterpillar turtle touches the side walls, the game is over. The score will increase as the caterpillar eats the leaves, that are randomly put across the screen.


Press SPACE to start

Python libraries

Python libraries are a set of useful functions that eliminate the need for writing codes from scratch.



```
import turtle
import statistics
import random
```

They can be brought into the program using the "import" keyword and can save valuable time when writing complex programmes.

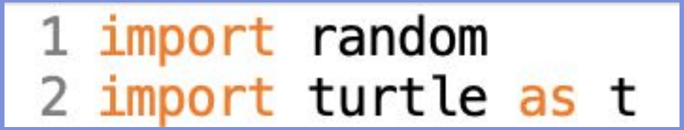
One common example of a python library that we will be using in this code is the "turtle" library which enables us access to graphical functions in python.

Step 1

Importing the turtle library and random module

Line 1: we will use the random module later on in the program as a means of creating random leaf placement.

Line 2: importing the turtle module as ”t” means that we will use t. to refer to the turtle most of the time.



```
1 import random
2 import turtle as t
```

cair 4 YOUTH

Step 2

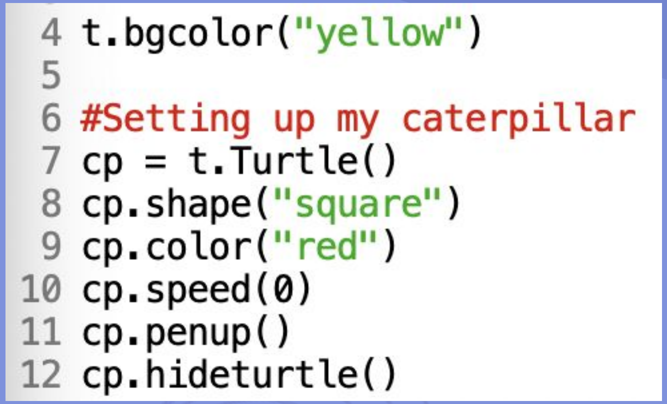
Setting up the caterpillar

Line 4- sets the background colour to yellow.

Line 7 and 8- changes the shape of the caterpillar to a red square.

Line 10- will set the speed that the caterpillar moves at to 0 (which just means fast).

Line 11 and 12- will be explained more later but essentially will allow the caterpillar to move around the screen and be hidden when commanded.



```
 4 t.bgcolor("yellow")
 5
 6 #Setting up my caterpillar
 7 cp = t.Turtle()
 8 cp.shape("square")
 9 cp.color("red")
10 cp.speed(0)
11 cp.penup()
12 cp.hideturtle()
```

cair 4 YOUTH

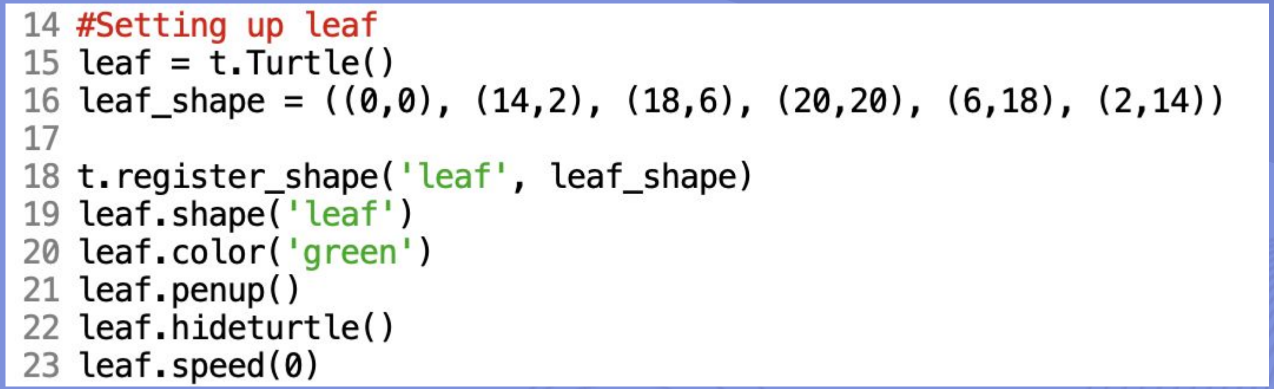
Step 3

Setting up the leaves

Line 15 and 16- creates the variable ”leaf” and ”leaf_shape” which will tell the program that the leaf is to be a turtle with the shape defined by the coordinates (0,0), (14,2), (18,6), (20,20),(6,18),(2,24)

Line 18, 19 and 20- will register this new shape that we have created, which will be green every time it appears on the screen.

Line 21, 22, 23- will look familiar to lines in step 2, which we will come back to later.

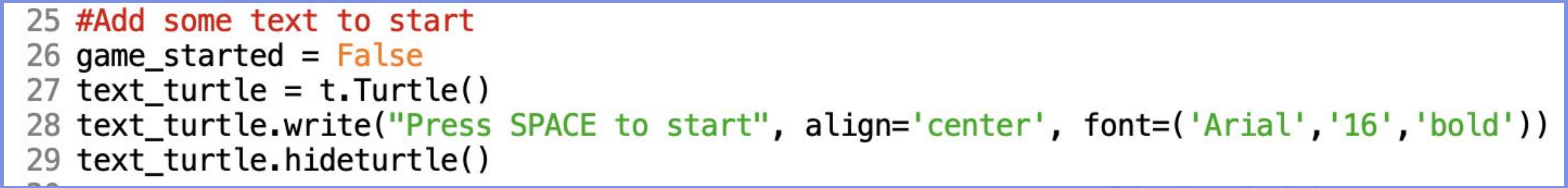


```
14 #Setting up leaf
15 leaf = t.Turtle()
16 leaf_shape = ((0,0), (14,2), (18,6), (20,20), (6,18), (2,14))
17
18 t.register_shape('leaf', leaf_shape)
19 leaf.shape('leaf')
20 leaf.color('green')
21 leaf.penup()
22 leaf.hideturtle()
23 leaf.speed(0)
```

Step 4

Adding some text to start the game

Lines 26,27,28 and 29- This will tell the user the instructions of the game and will be positioned in the middle of the screen. It will be hidden when instructed to do so later in the programme.



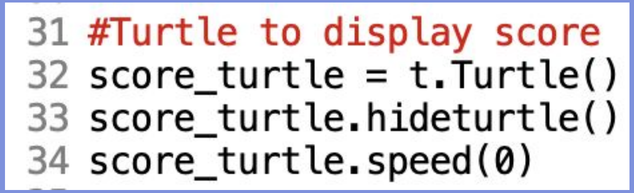
```
25 #Add some text to start
26 game_started = False
27 text_turtle = t.Turtle()
28 text_turtle.write("Press SPACE to start", align='center', font=('Arial','16','bold'))
29 text_turtle.hideturtle()
```

Step 5

Score turtle

Line 32- creates a turtle called score_turtle which will eventually hold the score as and when the caterpillar grows in size from eating the leaves.

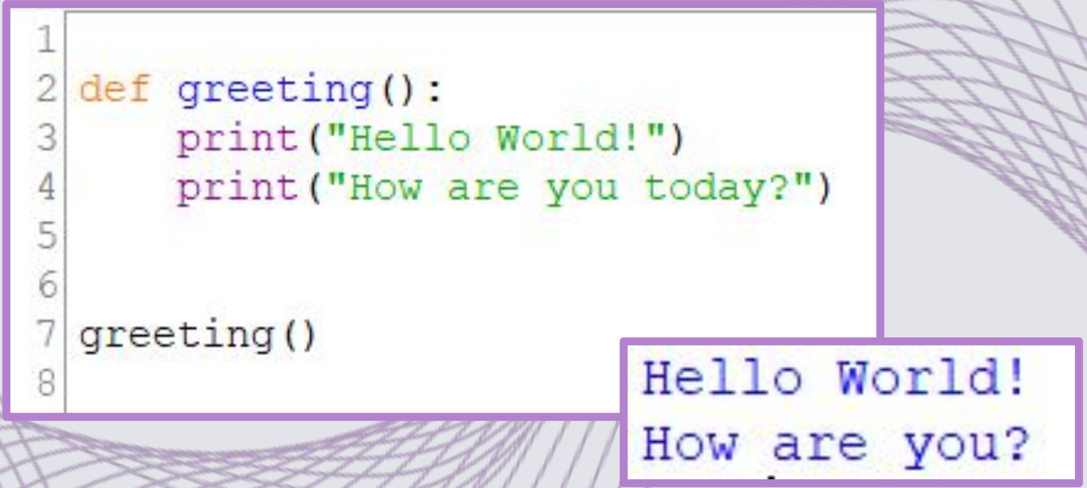
Line 33 and 34- will enable the score to be hidden and the speed to be fast.



```
31 #Turtle to display score
32 score_turtle = t.Turtle()
33 score_turtle.hideturtle()
34 score_turtle.speed(0)
```

Subroutines

Subroutines are sets of instructions designed to perform a frequently used operation within a programme.



```
def greeting():
    print("Hello World!")
    print("How are you today?")


greeting()
```

```
Hello World!
How are you?
```

Subroutines can store code and will only be run when 'called'.

There are two main types of the subroutine: procedures and functions.

Procedures are not required to return a value, whereas functions must return a value.

Subroutines are great ways of writing more maintainable code and leads to more structured, organized and understandable programmes.

cair
4 YOUTH

Step 6

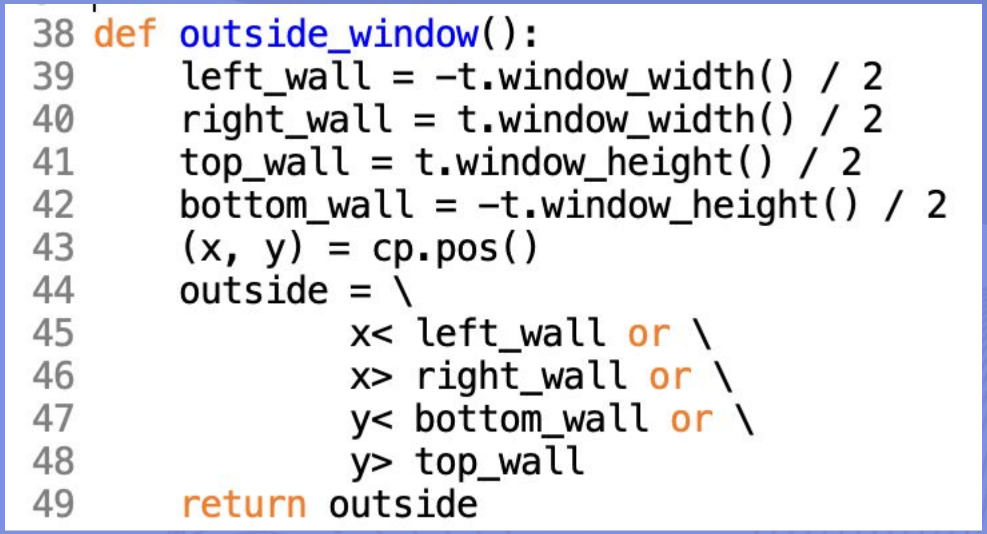
The outside border

Line 38- defines a subroutine called outside_window, which will create a border around the game, which if the caterpillar hits, will cause the game to end.

Line 39, 40, 41 and 42- defines the size of this outer border as half of the window height and width.

Line 43, 44, 45, 46, 47 and 48- could all be written on the same line (the backslash is just shown it looks nicer), but this means that a variable called outside is created, which is when the coordinates of the caterpillar turtle fall outside the range of this border.

Line 49- returns the variable outside, meaning that outside_window() is a function.



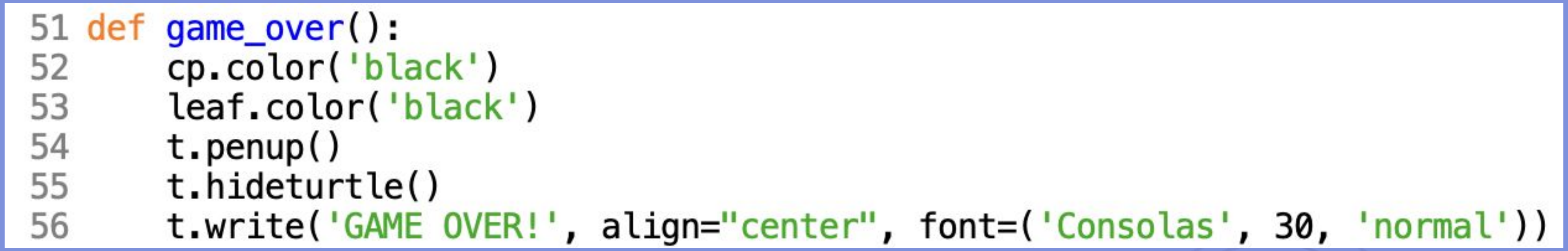
```
38 def outside_window():
39     left_wall = -t.window_width() / 2
40     right_wall = t.window_width() / 2
41     top_wall = t.window_height() / 2
42     bottom_wall = -t.window_height() / 2
43     (x, y) = cp.pos()
44     outside = \
45             x< left_wall or \
46             x> right_wall or \
47             y< bottom_wall or \
48             y> top_wall
49     return outside
```

cair
4 YOUTH

Step 7

Game over

Line 51- defines a subroutine called game_over, which much like the name suggests, will run when the user hits the border.



```
51 def game_over():
52     cp.color('black')
53     leaf.color('black')
54     t.penup()
55     t.hideturtle()
56     t.write('GAME OVER!', align="center", font=('Consolas', 30, 'normal'))
```

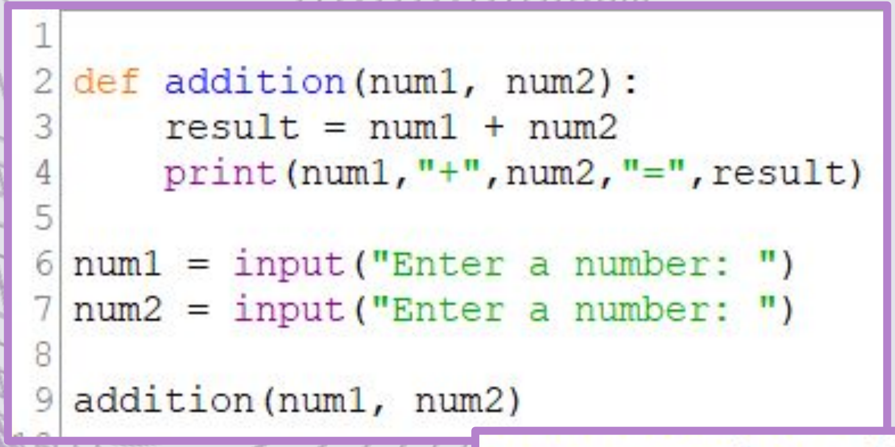
Lines 52 and 53- the colour of the leaf and the caterpillar will turn black to indicate the end of the game.

Lines 54, 55 and 56- will hide the turtle and stop it from moving, and will print game over in the middle of the screen in the font Consolas.

Parameters and arguments

Information can be passed into subroutines using parameters, which act as placeholders for the real values assigned when called.

The values put in the brackets when the subroutine is called are called arguments.

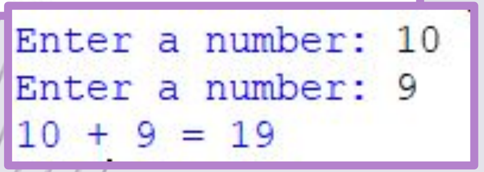


```

def addition(num1, num2):
    result = num1 + num2
    print(num1,"+",num2,"=",result)

num1 = input("Enter a number: ")
num2 = input("Enter a number: ")

addition(num1, num2)
```

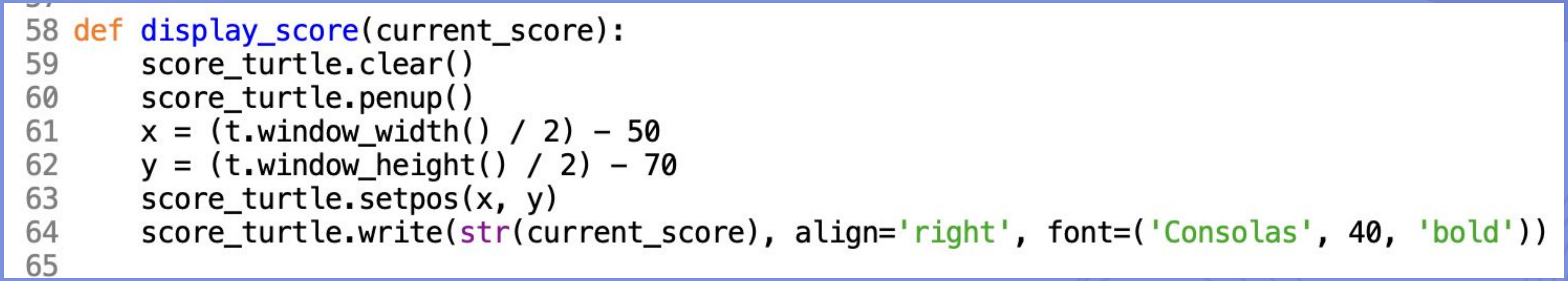


```
Enter a number: 10
Enter a number: 9
10 + 9 = 19
```

Step 8

Displaying the score

Line 58- the current_score variable is passed in as a parameter into the display_score subroutine that is defined in this line of code.



```
58 def display_score(current_score):
59     score_turtle.clear()
60     score_turtle.penup()
61     x = (t.window_width() / 2) - 50
62     y = (t.window_height() / 2) - 70
63     score_turtle.setpos(x, y)
64     score_turtle.write(str(current_score), align='right', font=('Consolas', 40, 'bold'))
65
```

Lines 59 and 60- will clear the score turtle, and reset it to be written to.

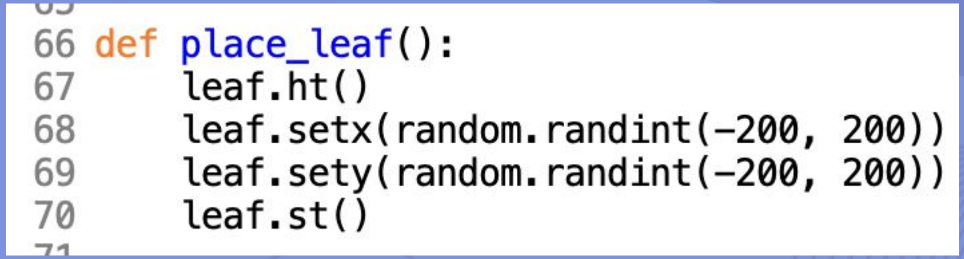
Lines 61, 62, 63 and 64- will position the score on the screen to the right and display it in the font Consolas, much like the fonts used throughout the programme.

Step 9

Random leaf position

Line 66- defines the place_leaf subroutine which will run every time that a leaf is eaten and needs to be replaced with a new one.

Line 67, 68, 69 and 70- will randomly position the leaf at a coordinate between (-200,-200) and (200,200).



```
66 def place_leaf():
67     leaf.ht()
68     leaf.setx(random.randint(-200, 200))
69     leaf.sety(random.randint(-200, 200))
70     leaf.st()
```

cair
4 YOUTH

Step 10

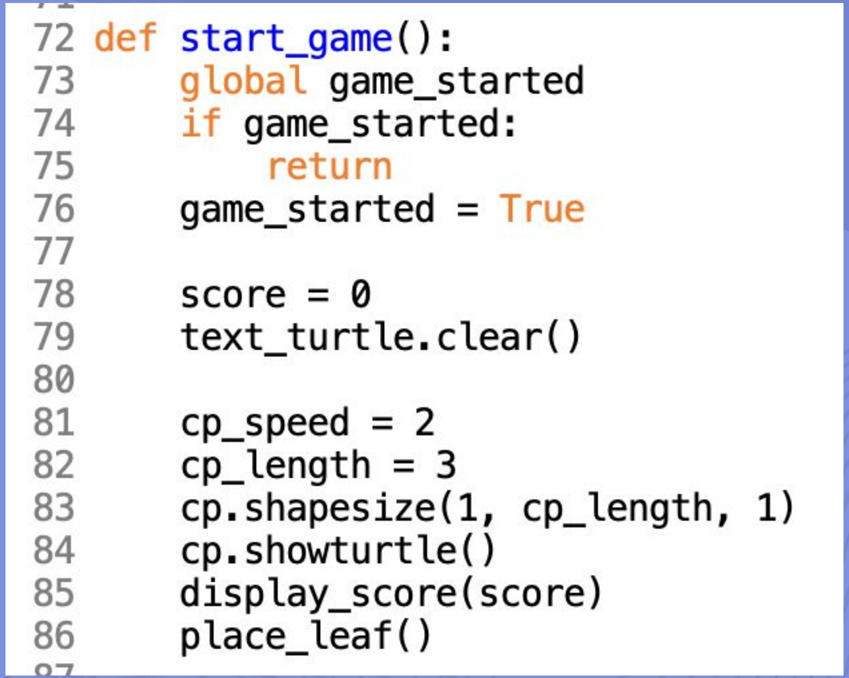
Starting the game

Line 72- defines the subroutine start_game, which will be run only once, when the space bar is hit at the very start of the game.

Lines 73, 74, 75 and 76- creates a global Boolean variable called game_started, which will only be declared as true if the user hits the spacebar.

Lines 78 and 79- will set the score to 0 and clear the text from the screen that gives the users the instruction.

Lines 81, 82, 83, 84, 85 and 86- sets the initial speed and length of the caterpillar as well as displaying it to the user along with the score, and a leaf, which is placed when the place_leaf() subroutine is run.



```
72 def start_game():
73     global game_started
74     if game_started:
75         return
76     game_started = True
77 
78     score = 0
79     text_turtle.clear()
80 
81     cp_speed = 2
82     cp_length = 3
83     cp.shapesize(1, cp_length, 1)
84     cp.showturtle()
85     display_score(score)
86     place_leaf()
```

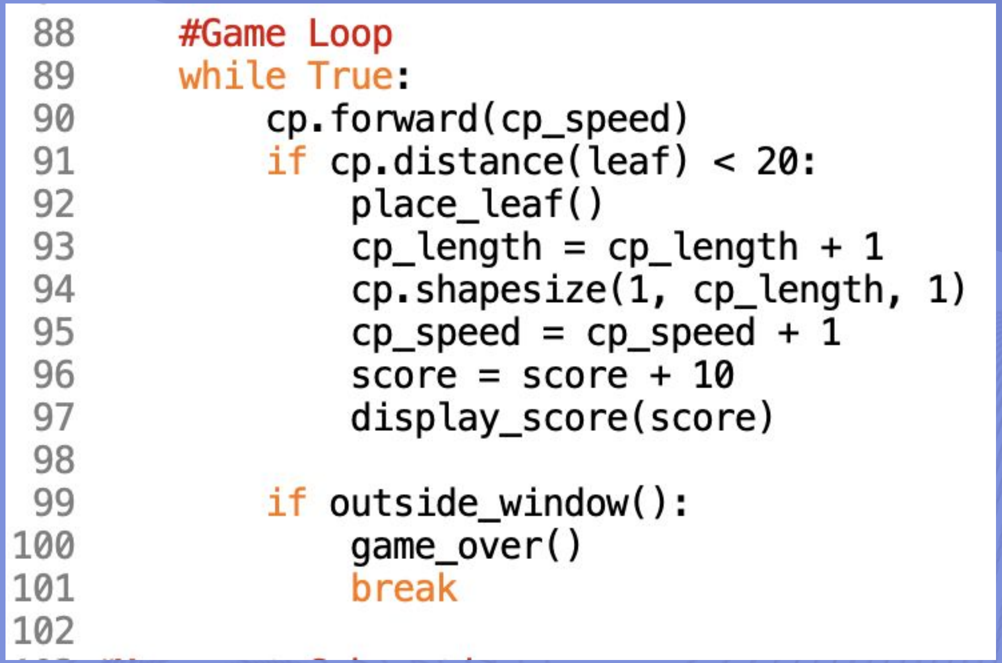
cair 4 YOUTH

Step 11

The game itself

Lines 89- 97- this code will run whilst the game is running. It continuously moves the caterpillar forward and if the distance between the leaf and caterpillar is less than 20, then the score will increase by 10 and be displayed, the caterpillar will grow in length and increase in speed (making the game harder), and a new leaf will be placed. See if you can match up each of these with the correct line number as a challenge!

Lines 99, 100 and 101- The code will also check to see if the caterpillar has hit the outside border, and if it has, the game_over() subroutine will run and the while loop will stop.



```
88      #Game Loop
89      while True:
90          cp.forward(cp_speed)
91          if cp.distance(leaf) < 20:
92              place_leaf()
93              cp_length = cp_length + 1
94              cp.shapesize(1, cp_length, 1)
95              cp_speed = cp_speed + 1
96              score = score + 10
97              display_score(score)
98
99          if outside_window():
100             game_over()
101             break
102
```

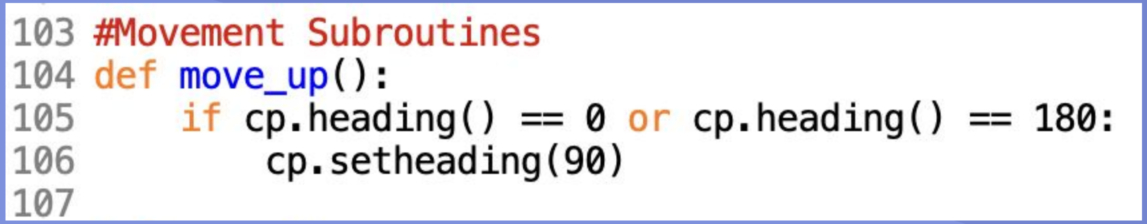
cair
4 YOUTH

Step 12

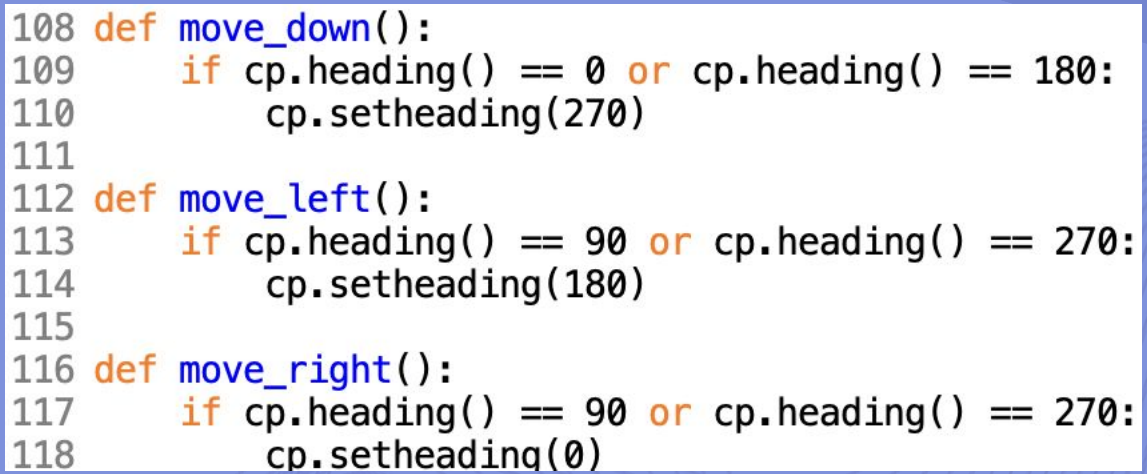
Movement of the caterpillar

Lines 104, 105 and 106- will create a subroutine called move_up, which will check the direction that the caterpillar is going in and then if horizontal, will move the caterpillar upwards.

Lines 108-118- the subroutines for moving horizontally and downwards. Imagine the set heading() numbers to be like the angles on a protractor to help visualize the direction.



```
103 #Movement Subroutines
104 def move_up():
105     if cp.heading() == 0 or cp.heading() == 180:
106         cp.setheading(90)
107
```



```
108 def move_down():
109     if cp.heading() == 0 or cp.heading() == 180:
110         cp.setheading(270)
111
112 def move_left():
113     if cp.heading() == 90 or cp.heading() == 270:
114         cp.setheading(180)
115
116 def move_right():
117     if cp.heading() == 90 or cp.heading() == 270:
118         cp.setheading(0)
```

cair 4 YOUTH

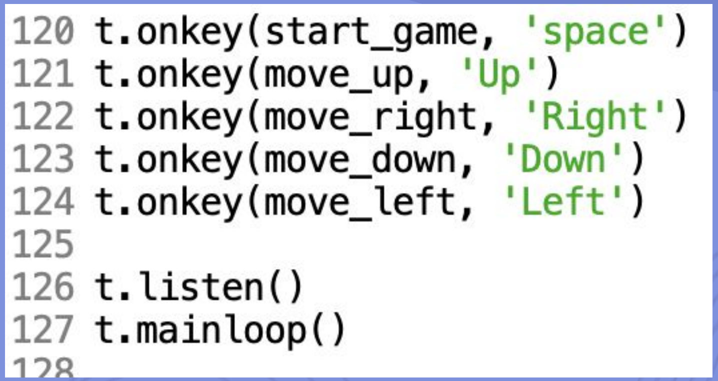
Step 13

Staring and moving and calling the subroutines

Lines 120, 121, 122, 123 and 124- depending on which key is pressed depends on which subroutine is run. If the user enters a:

- space the game will start
- Up arrow moves the caterpillar up
- The right arrow moves the caterpillar right
- Down arrow moves caterpillar down
- Left arrow moves caterpillar left

Lines 126 and 127, will begin the main loop of code and the game will continue to be run until the player eventually loses.



```
120 t.onkey(start_game, 'space')
121 t.onkey(move_up, 'Up')
122 t.onkey(move_right, 'Right')
123 t.onkey(move_down, 'Down')
124 t.onkey(move_left, 'Left')
125
126 t.listen()
127 t.mainloop()
128
```

cair
4 YOUTH

Links to everyday life

Vintage games

Simple graphical games like this used to be all of the crazes in the early days of computers and were revolutionary and addictive at the time.

Coding is fun

Making little games like this in python requires a degree of skill to both create and play along afterwards.

Caterpillars

Caterpillars do eat leaves and a lot of them. They are a common pest found in the agricultural world, much to the dismay of many farmers.

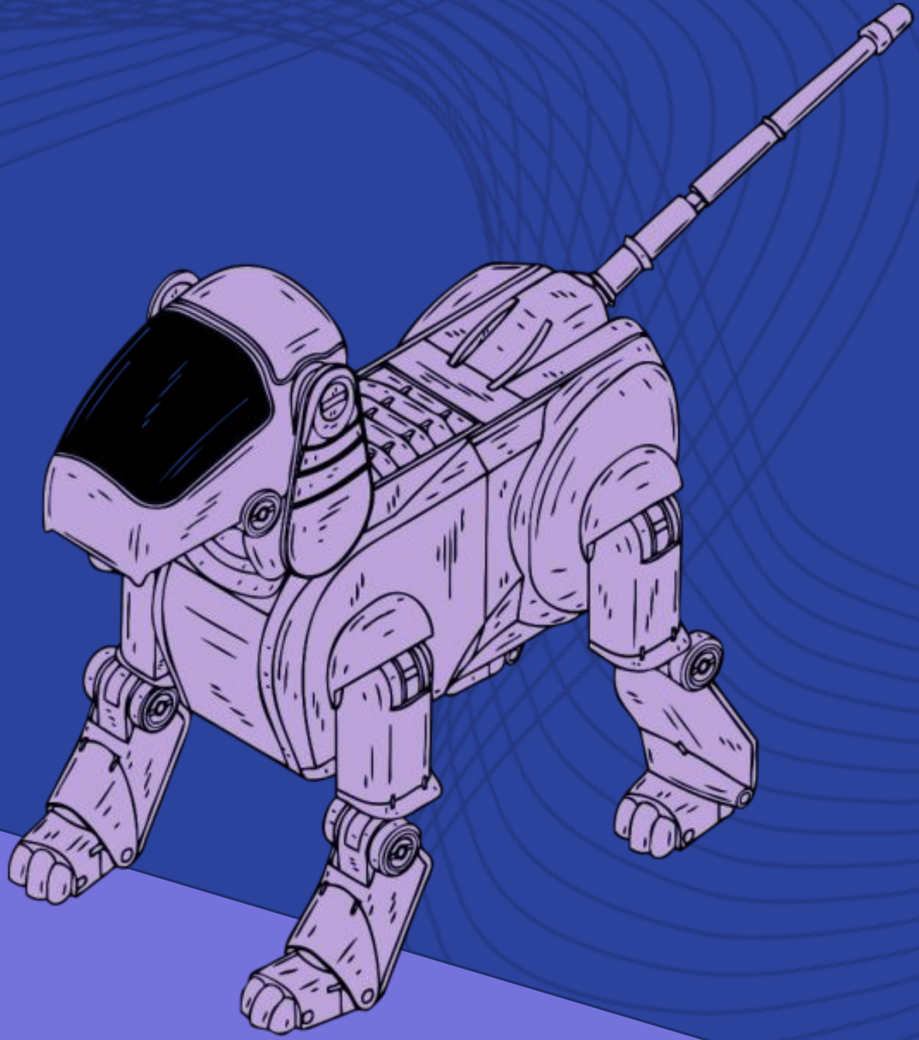
Conclusion

Learning outcomes

- ✔ You should be able to confidently use subroutines to separate code.
- ✔ You should be able to use turtle graphics to create a simple shape that can move around the screen.

Congratulations!

You have completed the Caterpillar Game using Turtle graphics



cair
4 YOUTH