Task One

# Introduction

1. import turtle
2. t = turtle.Turtle()
3. t.forward(100)
4. t.left(45)
5. t.forward(30)

Click on the “Task 1” button to load in the above program, and then click the “Run” button.

Here you are creating a new turtle, called **t**, then telling it to go forward 100 units, go left 45 degrees, then go forward again by 30 units.

Modify the above code to make the turtle go forward by 200 units, turn 90 degrees left, then go forward by another 100 units.

Other commands you can use to move the **t** are **t.right** and **t.backward**. These are used in exactly the same way as **t.forward** and **t.left**.

Add to the code to make the turtle turn right by 40 degrees, then go backward by 200 units after it has done its original forward-left-forward movement.

# Numbers and variables

You are using a language called Python to tell the turtle what to do.

Python can also do other things, such as basic maths.

1. print 3
2. print 8+11
3. print 9-10
4. print 40\*5
5. print 10/2
6. print ((87+3)/30\*(51-3\*9))/6

Type in the above code and click “Run”. You should see the results of these arithmetic operations in the results box.

**+** represents addition, **-** represents subtraction, **\*** represent multiplication, and **/** represents divide. These can be combined to make very long arithmetic expressions, just like in maths.

1. x = 5
2. print x
3. x = 8
4. print x
5. x = x + 1
6. print x
7. print x\*2

Type in the above code and click “Run”.

Here, we are using a **variable**. I am creating some storage space called **x** which I am using to hold a number. Notice how the **print x** statements all result in a different number being output.

Line 5 is

1. x = x + 1

This can be thought of as **the new value of x** is set to be **the old value of x** plus 1. This has the result of increasing the stored value of **x** by 1.

# Using numbers and variables

1. import turtle
2. t = turtle.Turtle()
3. x = 100
4. t.forward(x)
5. t.left(45)
6. t.forward(x)

Click on “Task 1” and modify the code so it looks like the above. (Changes to be made have been highlighted) Try modifying the value of x on line 4 and see how this affects the shape drawn by the turtle.

Notice how the two lines drawn are the same length. The code can be modified to make the second line twice as long as first by changing lines 7 as follows:

1. t.forward(x)
2. t.left(45)
3. t.forward(x\*2)

Try changing the code to make:

* The second line three times as long as the first line
* The first line 40 units shorter than the second line.

# Draw a square

Make the turtle draw a square with edges of size 100. Try to use a variable to control the length of the square’s sides, as this will make it easy to resize the square.

Try to draw the biggest square you can without the turtle going out of the canvas.

Task Two

# A loop

1. import turtle
2. t = turtle.Turtle()
3. for i in range(6):
4. t.forward(40)
5. t.left(45)

Click on the “Task 2” button to load in the above program, and then click the “Run” button.

Here we are using something called a **for loop**. Line 4 is saying “execute the following lines 6 times”. This results in the turtle going forward and left 6 times. At the moment it is drawing ¾ of an octagon.

Tweak the above code to make the turtle draw the whole octagon. Then modify the code to draw a square.

|  |  |
| --- | --- |
|  |  |

# Blocks and indentation

In more detail, a for loop works by telling Python to run the next **block** of code some number of times. Indentation is used in python to describe where blocks of code start and end.

1. import turtle
2. t = turtle.Turtle()
3. for i in range(4):
4. t.forward(40)
5. t.left(45)

Here, lines 5 and 6 are part of the same block. This means that both lines will be executed in turn as part of the for loop on line 4.

1. import turtle
2. t = turtle.Turtle()
3. for i in range(4):
4. t.forward(50)
5. t.left(45)

Here, lines 5 and 6 are not part of the same block. This means only line 5 is executed as part of the for loop on line 4.

In order for Python to consider a set of lines to be a block, they must have the same number of spaces before the line text starts. Generally, the number of spaces used is in multiples of four.

Try both versions of the code above and see what happens to the turtle in each case.

# Loops within loops

There is nothing stopping you from putting a loop within a loop.

First write a loop to draw a square, if you have not done it already.

1. import turtle
2. t = turtle.Turtle()
3. for i in range(4):
4. t.forward(40)
5. t.left(90)

Indent these lines of code to form a block, then add the line ‘for j in range(4):’ above the block and add the line ‘t.forward(80)’ below the block.

1. import turtle
2. t = turtle.Turtle()
3. for j in range(4):
4. for i in range(4):
5. t.forward(40)
6. t.left(90)
7. t.forward(80)

Run this code and see what happens. Try changing the numbers in the code to draw more/fewer squares, larger or smaller squares, and squares that are further or closer spaced.

In order to make it more clear when the turtle is drawing each line, you can add in **pencolor** commands to change the colour.

for j in range(4):

for i in range(4):

t.pencolor('blue')

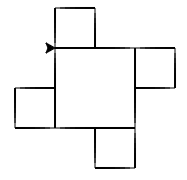
t.forward(40)

t.left(90)

t.pencolor('red')

t.forward(80)

Using two loops, draw a square with a square at each corner.



Task Three

# Procedures

At the moment, Python runs code line-by-line.

1. import turtle
2. t = turtle.Turtle()
3. x = 60
4. t.pencolor('red')
5. t.forward(x)
6. t.pencolor('blue')
7. t.forward(x)
8. t.pencolor('yellow')
9. t.forward(x)

This code will draw a red, blue, yellow, and then green line.

Sometimes we want to modify the order in which Python runs the lines.

1. import turtle
2. t = turtle.Turtle()
3. def drawSquare():
4. for i in range(4):
5. t.forward(40)
6. t.left(90)
8. drawSquare()

Click on the “Task 3” button to load in the above program, and then click the “Run” button.

Here, the code is **defining** a **procedure** called **drawSquare** on lines 4-7. It is then **calling** that procedure on line 9.

If you did not do task two, line 5 is saying ‘execute the following block of code 4 times’.

Defining a procedure tells Python **how** to do something (in this case, we are telling it how to draw a square). When a procedure is called it is telling Python **when** to do something. A procedure must be defined before it is used.

When defining a procedure, we use the keyword **def**, then give it a name (**drawSquare** in this case). When calling that procedure, you need to type the produce’s name exactly as it appears in the definition (including any capitals), followed by parentheses.

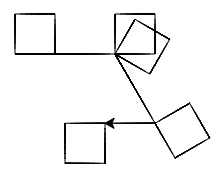
Change line 9 to include a spelling mistake (e.g. ‘drowSquar()’) and see what happens when you click “Run”. Remember to change it back to drawSquare() afterwards.

The main benefits of using procedures are:

* Abstraction – You can now tell Python to ‘draw a square’ rather than ‘go forward and left 4 times’. You are telling Python **what** to do rather than **how** to do it.
* Code re-use – At any point in the program you can just write ‘drawSquare()’ and Python will draw a square. There is no need to type out the loop again.

Using drawSquare() and the t.forward, t.left, and t.right commands, draw 5 squares at various places.

The results of your drawing may look something like this:



# Procedures of procedures

There is nothing to stop you using a procedure you have already defined to define another procedure. As an example, this procedure will draw four squares along the edge of a square.

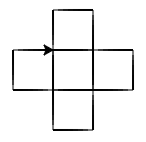
def squareOfSquares():

for j in range(4):

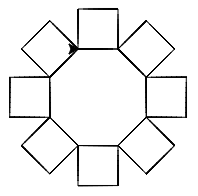
drawSquare()

t.forward(40)

t.right(90)



Using the code above as a template, write code to draw an octagon with a square on each edge.



# Letters

Write a subroutine to draw a letter of your choice.

Here is an example to draw an A.

def drawA():

t.left(70)

t.forward(150)

t.right(140)

t.forward(150)

t.backward(60)

t.right(110)

t.forward(63)

Task Four

# Random numbers

1. import turtle
2. import random
3. t = turtle.Turtle()
4. length = random.randint(10,100)
5. for i in range(4):
6. t.forward(length)
7. t.left(90)

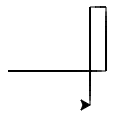
Click on the “Task 4” button to load in the above program, and then click the “Run” button.

Try clicking the button multiple times. What happens to the size of the square?

Every time we run the program, the **length** variable is assigned a random number between 10 and 100.

Move line 5 to be inside the for loop on line 7 (Put the line between lines 7 and 8, and put 4 spaces before the line if necessary). See how the turtle acts now.

1. import turtle
2. import random
3. t = turtle.Turtle()
4. for i in range(4):
5. length = random.randint(10,100)
6. t.forward(length)
7. t.left(90)



Originally, the **length** was randomly rolled before entering the loop. Each time the loop was executed the line **t.forward(length)** would move the turtle forward by the same amount. By moving line 5 “inside the loop” the length is randomly calculated with for each line drawn.

When the length was rolled randomly just once, turtle was stopped after drawing 4 lines. After this point, the turtle was just retracing its own steps, so there was no point continuing. However, now that you are randomly choosing the length for each line, there is no reason to stop after 4 iterations.

Change line 5 to “for i in range(100):”