Computer Science

Daire Ó Muirgheasa

Higher Level

2020-21

Intro to Python Programming (turtle)

THE INSTITUTE OF EDUCATION

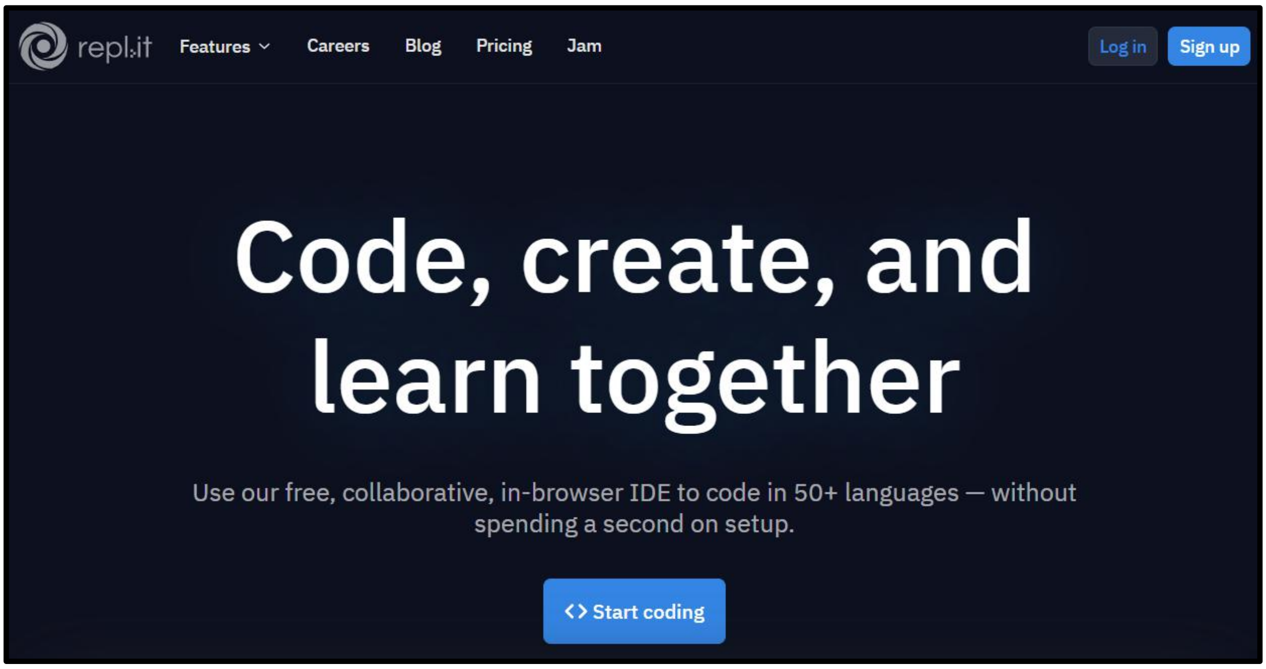
CONTENTS

Contents

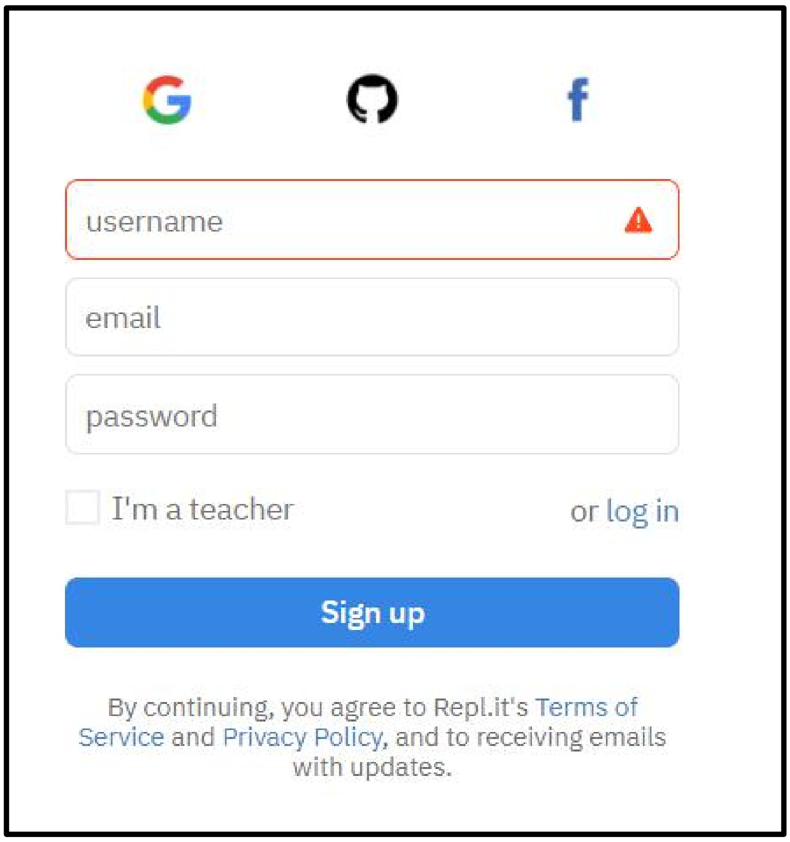
Lesson 1 - Intro to Python Turtle

Before we begin coding, we will need to find an editor or IDE to write our code in. There are many different options for this, but in our class, we are going to use an online editor. This online editor is called repl.it.

To use this editor, go to the website repl.it. You then need to create a repl account. This is where you will save all of your work throughout the course.

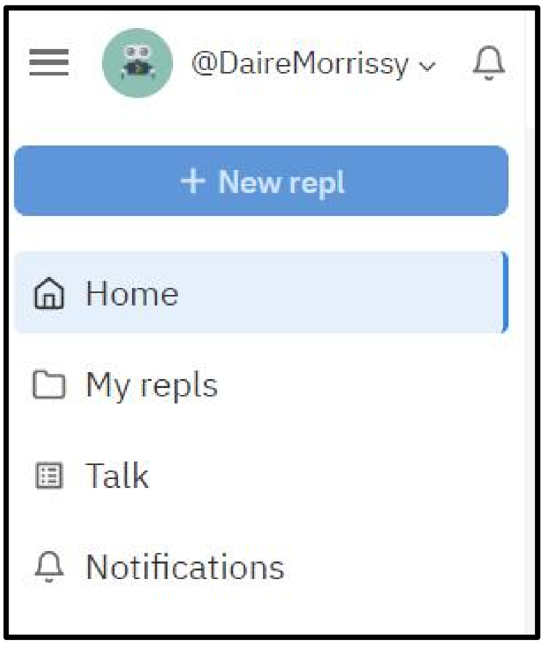

repl.it
Features
Careers
Blog
Pricing
Jam
Log in
Sign up
Code, create, and learn together
Use our free, collaborative, in-browser IDE to code in 50+ languages — without spending a second on setup.
<> Start coding

Click on “Sign up” in the top right corner of the screen. Use your college email address to sign up for repl and fill in your details. The page will look like the picture on the right.

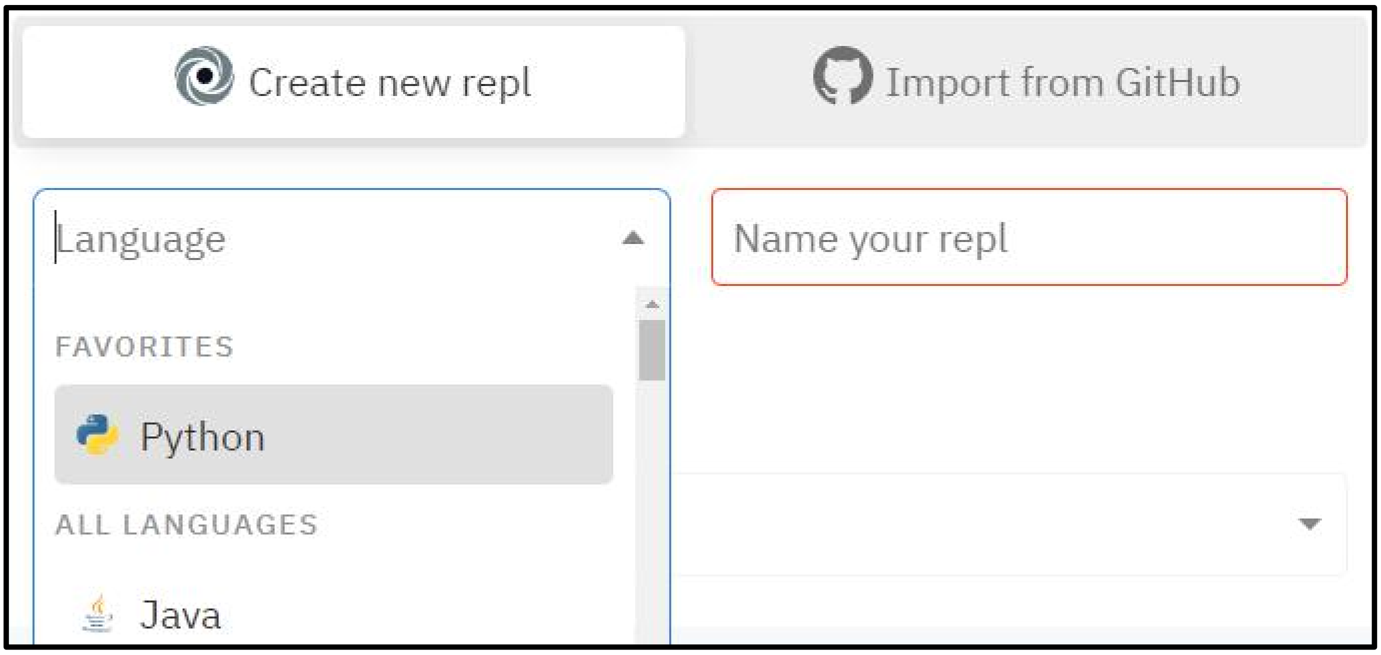


By continuing, you agree to Repl.it's Terms of Service and Privacy Policy, and to receiving emails with updates.

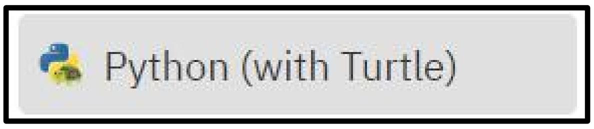
Next, we need to choose the programming language that we will be using. Throughout the course we will be using several different languages, but for today we want to use Python (with Turtle). On the left of your screen, you should see the image to the right.



You need to click on "+ New repl". You will see the image below pop up. We will be using "Python (with Turtle)" in this lesson. You can either search through the list or begin typing the name of the language.



When you have found and selected the correct language you will need to give it a name. E.g. "Lesson 1 - Turtle Graphics".

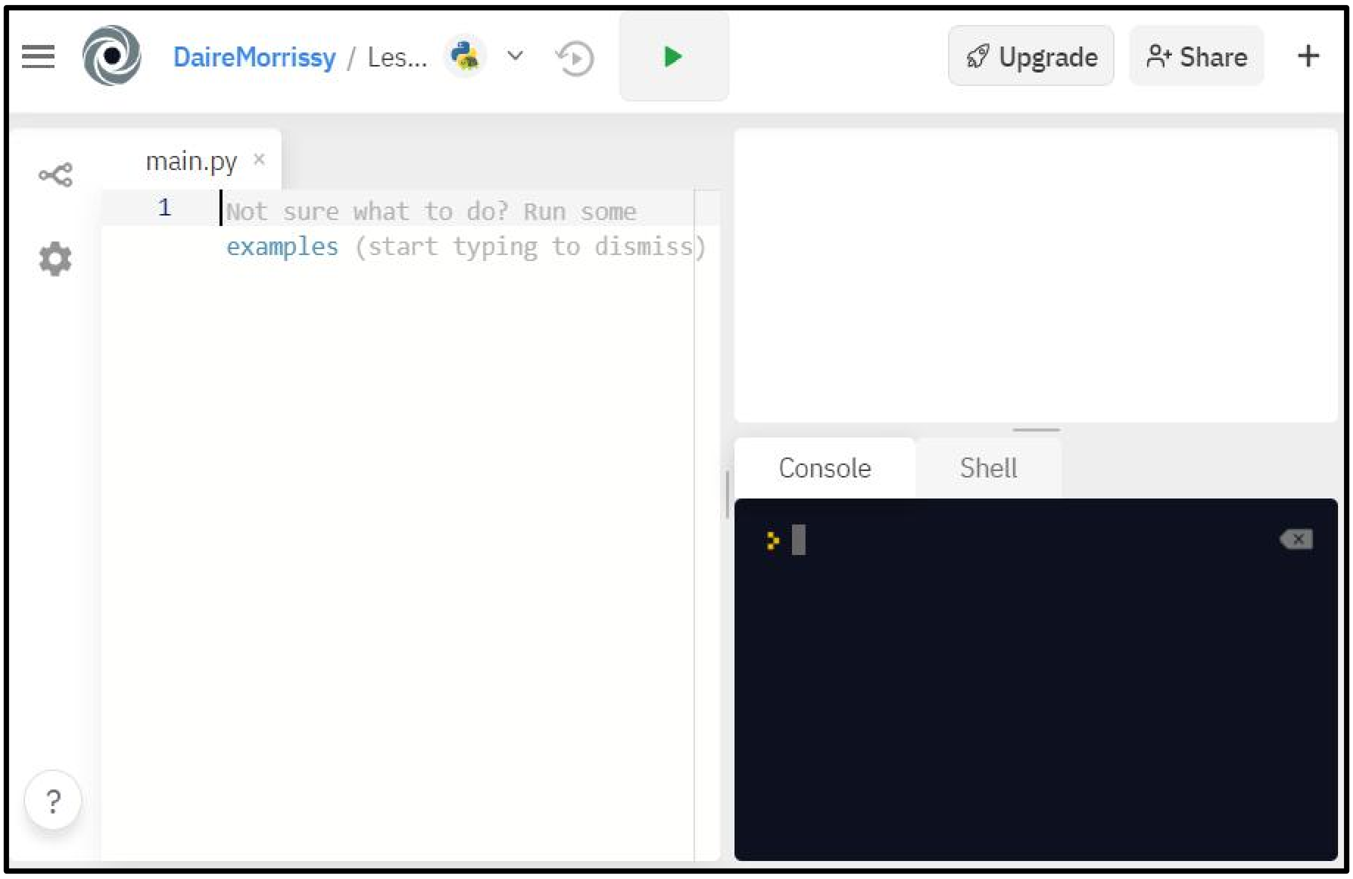


Then you need to click "Create repl". Once you click "Create repl" you will be brought to a page like the one on the next page. Depending on the language being used, this page may look slightly different.

At the moment there are four things of note that we want to look at.

1. The first is the white box on the left-hand side of the page. This is where we will write our code and is called the editor.
2. The second is the green play button at the top of the page. When we click on this, the website will try running our code.
3. The third is the white box on the right side of the page. This is where we will see any of the graphics that we draw in our code. **N.B If you don't see this box you might have chosen "Python" instead of "Python (with Turtle)".**
4. The fourth is the black box on the bottom right side of the page. This is the console, and this is where we will see certain messages that we print out and where we will see errors if they happen.

These boxes can be minimised and deleted, but when starting a new repl they should appear.



```
Not sure what to do? Run some examples (start typing to dismiss)
```

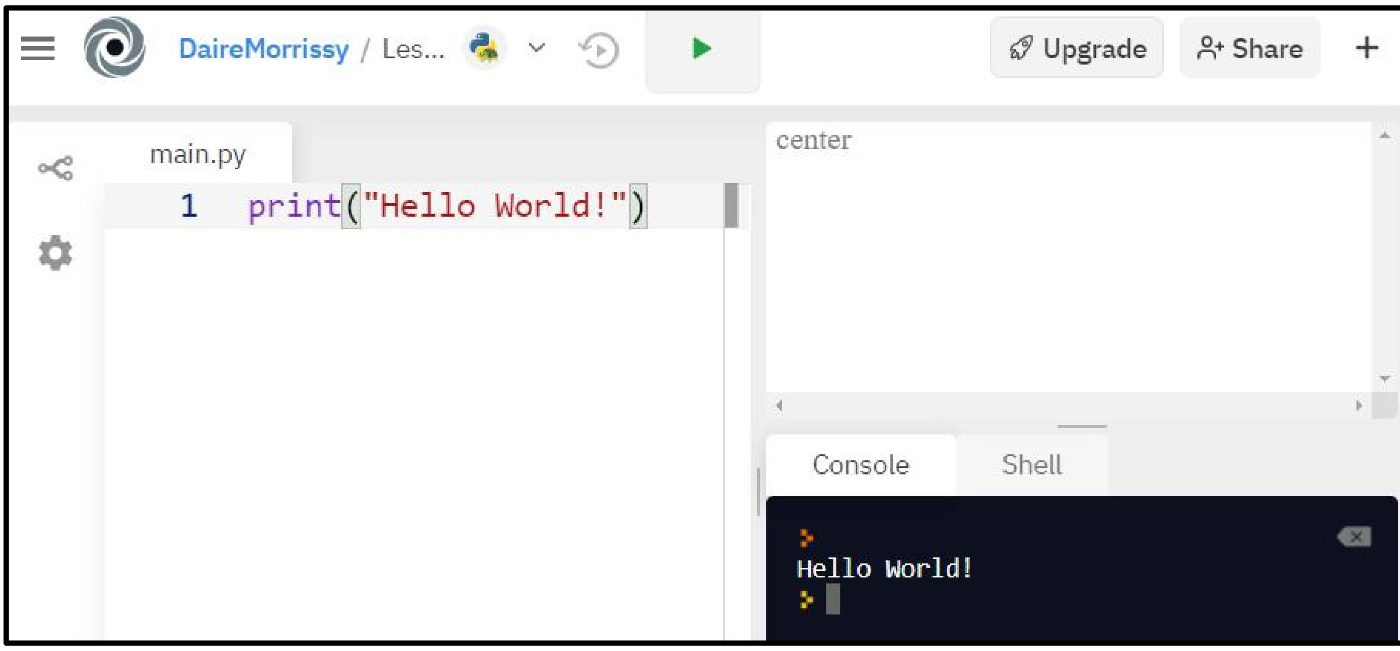
Our first program

The very first command we are going to learn is the print command. This command or function will print whatever we write between the quotation marks to the console.

Try writing the code below in the editor and click run.

```
print("Hello World!")
```

When you click run you should see your message in the console. Try changing the message and running it again.



```
print("Hello World!")
```

```
Hello World!
```

In Python there are two ways of writing a message to the console. The first is using quotation marks around our message, the second is using single quotes around our message.

Try writing and running the code below. You should see the same message.

```
print('Hello World!')
```

It is important not to mix these two different methods together in the same line. Try the code below and see what happens.

```
print('Hello World!")
```

Turtle graphics

Now that you have written your first program, we are going to look at doing something a little more exciting. For the first part of this course, we are going to be using a library called Turtle graphics. This is a visual library that is very good at introducing coding in an interesting way.

Before we can use this library, we will need to import it into our code. We will talk a lot more about this later in the course, but for now all we need is the line of code below. This line will go at the top of your code.

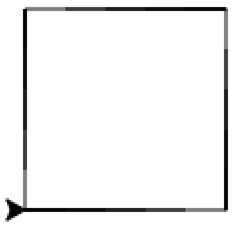
```
import turtle
```

This line of code will allow us to use the commands/functions in this library. To learn about these commands/functions we are going to write some code and see what happens. Write the code below and click run.

```
turtle.speed(1)
turtle.forward(100)
turtle.left(70)
turtle.forward(100)
```

Challenge 1

1. Change the numbers in the code one by one and see what happens.
2. Using what you have learned, draw a rectangle.



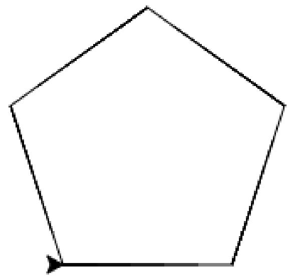
More basic commands/functions

After you have completed the challenge on the previous page you may already be able to guess some of the other functions available to us in Turtle graphics.

The opposite of `turtle.forward(100)` is `turtle.backward(100)` and the opposite of `turtle.left(70)` is `turtle.right(70)`.

Challenge 2

1. Use these new functions to create a pentagon.



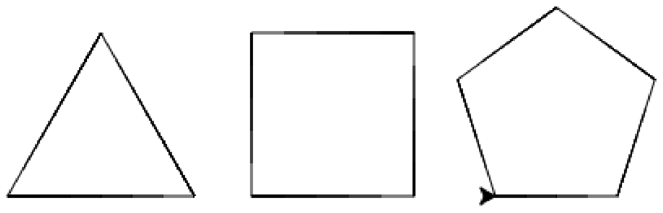
Next, we are going to learn about three more commands/functions. These are turtle.penup(), turtle.pendown() and turtle.setposition().

Copy the code below to see what these functions do.

```
turtle.setposition(100,100)
turtle.penup()
turtle.backward(100)
turtle.pendown()
turtle.setposition(0,0)
```

Challenge 3

1. Using what you have learned, draw a rectangle, a triangle, and a pentagon on the same screen in different positions. We will be reusing this code, make sure to save it.

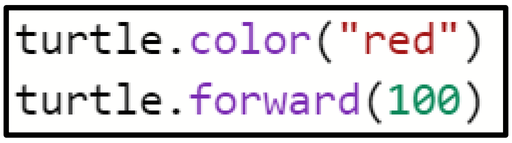


Colour and looks

Once you have drawn your shapes and become comfortable with the functions we have used so far, you may begin wondering how we can make our sketches look better and more interesting?

We can do this by adding colour to our sketches. We can do this using the turtle.color() function. There are a couple of ways we can use this function. The first is to simply write the name of a colour.

Copy the code below and run it. You should see a red line. Then try changing it to your favourite colour.



```
turtle.color("red")
turtle.forward(100)
```

The second method for choosing our colour is to use three numbers to describe the colour. These three numbers represent the amount of red, green, and blue in our colour.

For example.

(255,0,0) = red

(0,255,0) = green

(0,0,255) = blue

(255,255,0) = yellow

(255,0,255) = magenta

(0,255,255) = cyan

(255,100,0) = orange

Using a different mix of each of these colours we can find any colour we want. Fortunately, we don’t need to learn these off, we can use a colour picker to find these numbers for us: https://htmlcolorcodes.com/color-picker/, choose your colour and copy the R, G and B values.

Change the code below to draw a different colour.

```
turtle.color(255,100,0)
turtle.forward(100)
```

The third method for choosing a colour is to use a hexadecimal number. Hexadecimal numbers will be discussed more in the future, but they are basically a different method of writing the three numbers that we chose in the second method.

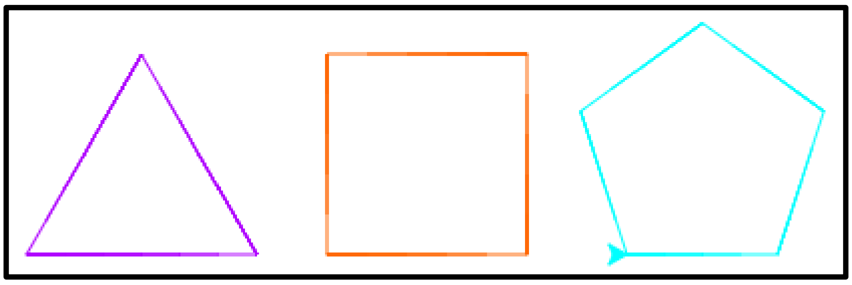
The code below gives us the same orange colour as turtle.color(255,100,0)

```
turtle.color("#FF6400")
turtle.forward(100)
```

You can also find the hexadecimal values for every colour in the colour picker.

Challenge 4

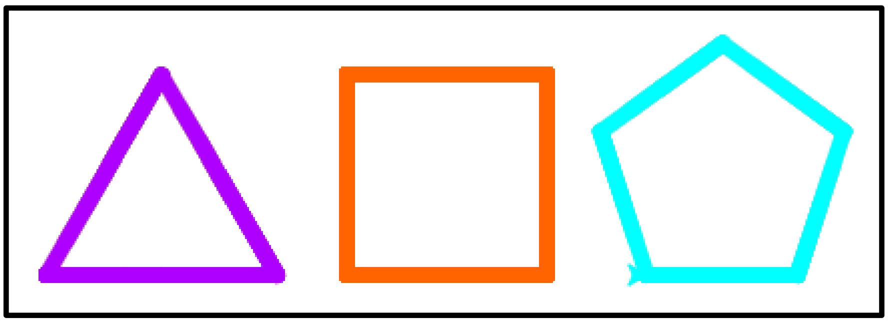
1. Using the code from the previous lesson, change the outline colour of each of your three shapes.



2. We can then use the function turtle.pensize() to change the outline size of the shapes. Add the code below at the beginning of your code.

```
turtle.pensize(8)
```

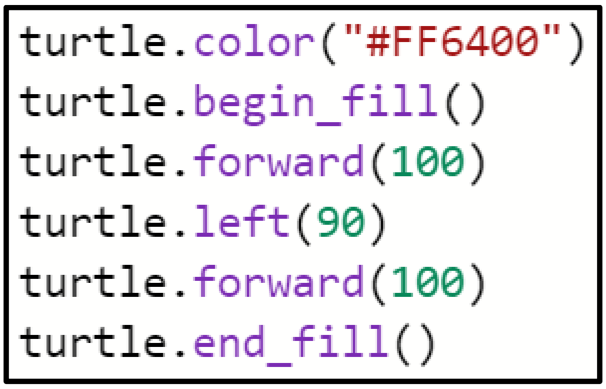
3. You should see something like the output below.



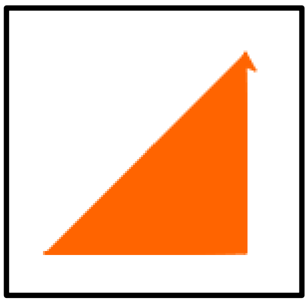
More colours

We now know how to change the colour of the outline of our shapes, but how do we colour in our shapes. To colour our shapes in we need to use two new functions. **turtle.begin_fill()** and **turtle.end_fill()**.

These go at the beginning and end of the shape we want to colour in. The code below will create the output on the right. Try this now and change the shape that is being drawn.

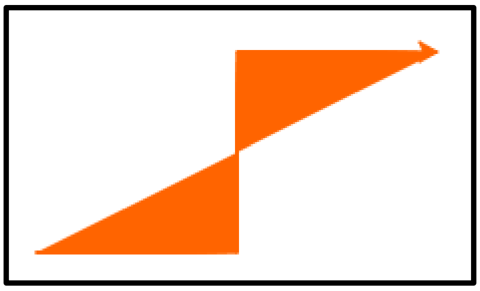


```
turtle.color("#FF6400")
turtle.begin_fill()
turtle.forward(100)
turtle.left(90)
turtle.forward(100)
turtle.end_fill()
```



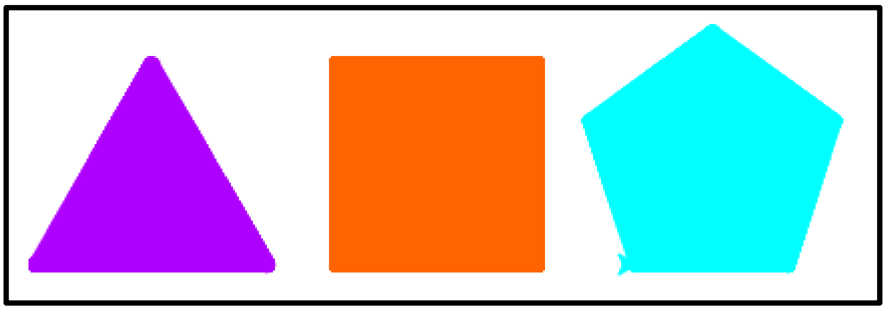
Challenge 5 (Extra)

1. Try to create the image below.



Challenge 6

1. Fill in each of the shapes that you have created.



More functions

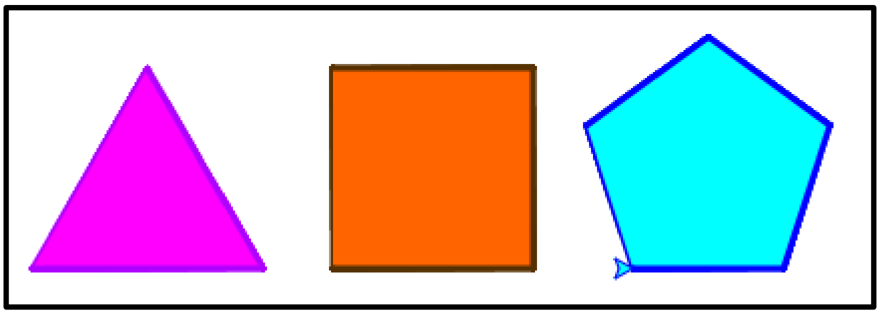
The last two functions that we will learn about in this lesson are turtle.pencolor() and turtle.fillcolor(). These functions only set the colour for a specific part of the drawing.

```
turtle.pencolor("#FF6400")
turtle.fillcolor("#00BB00")
```

Replace turtle.color() with these and see what happens.

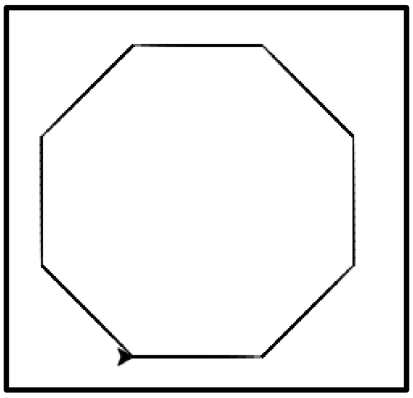
Challenge 7

1. Use pencolor and fillcolor to create the pattern below.

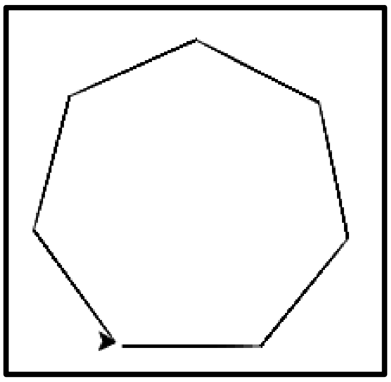


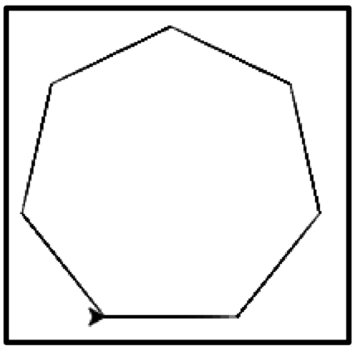
Challenge 8

1. Draw an octagon (8 sided) and a heptagon (7 sided).



2. Depending on how you write the code you might get a slightly incomplete heptagon. We'll discuss this problem in class.





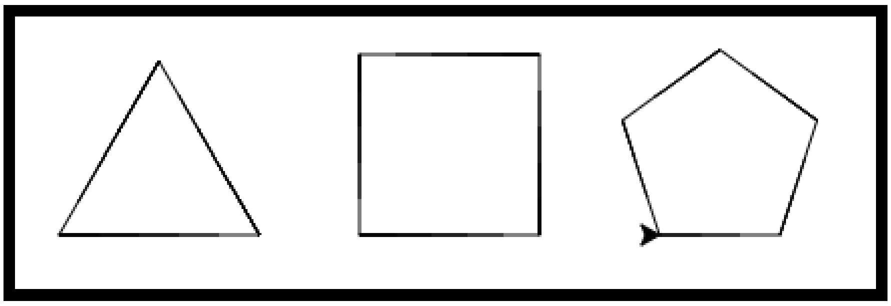
Challenge 9

Below is a list of functions, figure out what they do and use them in your sketch. Write down what each of them do.

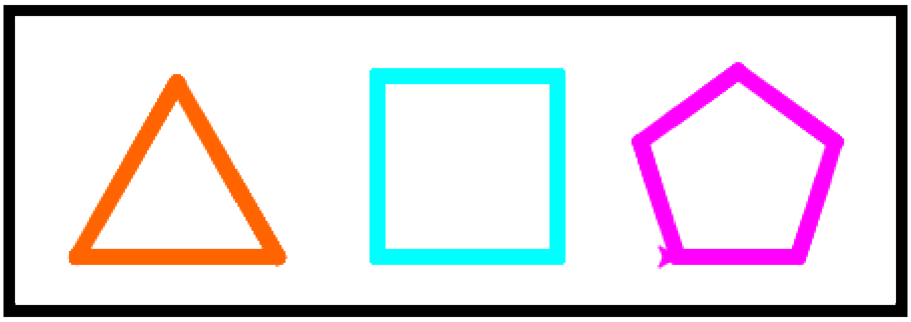
1. turtle.rt(100)
2. turtle.fd(100)
3. turtle.lt(100)
4. turtle.bk(100)
5. turtle.home()
6. turtle.circle(100)
7. Try
    a. turtle.shapesize(1,5,10)
    b. turtle.shapesize(10,5,1)
    c. turtle.shapesize(1,10,5)
    d. turtle.shapesize(10,1,5)
8. Try
    a. turtle.shape(“turtle”)
    b. turtle.shape(“arrow”)
    c. turtle.shape(“circle”)
9. turtle.undo()
10. turtle.clear()
11. turtle.stamp()

Lesson 1 - Intro to Python Turtle Worked examples

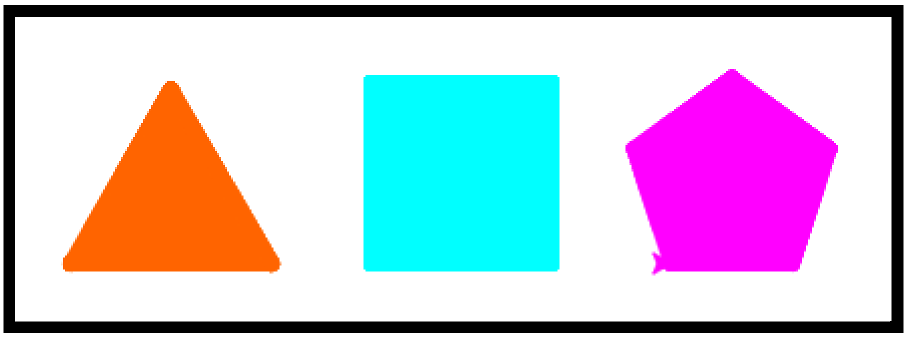
- Challenge 1 - https://repl.it/@DaireOM/Lesson-1-Challenge-1



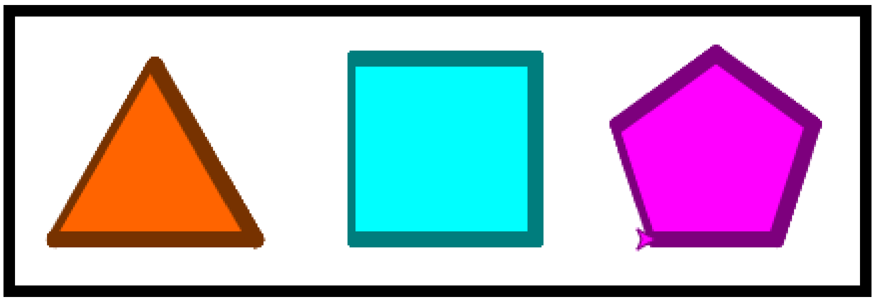
- Challenge 2 - https://repl.it/@DaireOM/Lesson-1-Challenge-2



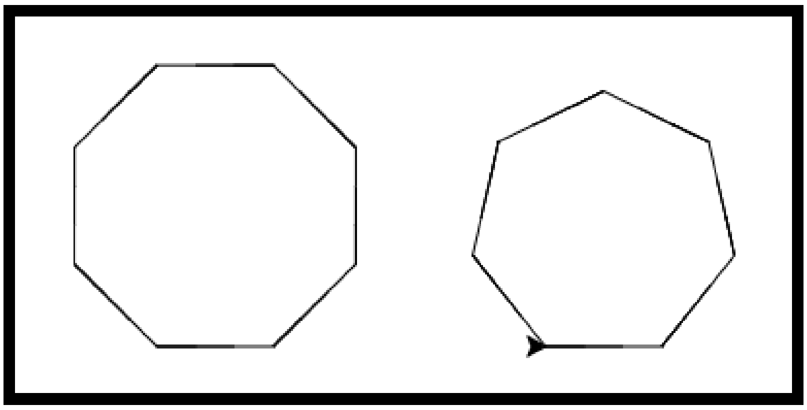
- Challenge 3 - https://repl.it/@DaireOM/Lesson-1-Challenge-3



- Challenge 4 - https://repl.it/@DaireOM/Lesson-1-Challenge-4



- Challenge 5 - https://repl.it/@DaireOM/Lesson-1-Challenge-5

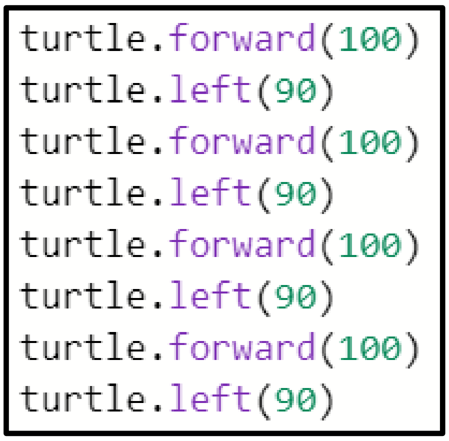


Lesson 2 - For loops and Variables

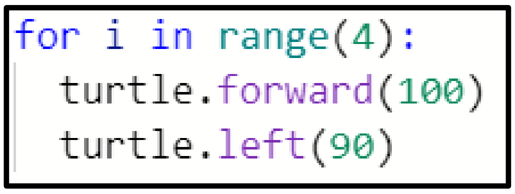
In the previous lesson we learned how to create different shapes using Python turtle. The shapes that we did last lesson were basic and didn't require too many lines of code, but we could see that as the shape had more edges the number lines of code, we needed increased quickly.

Programmers don't like writing lots of code, so we need to come up with a better solution. The better solution is to use a loop. This is code that will run repeatedly for a certain number of times.

Let's draw a square first without the for loop and then using a for loop.



```
turtle.forward(100)
turtle.left(90)
turtle.forward(100)
turtle.left(90)
turtle.forward(100)
turtle.left(90)
turtle.forward(100)
turtle.left(90)
```

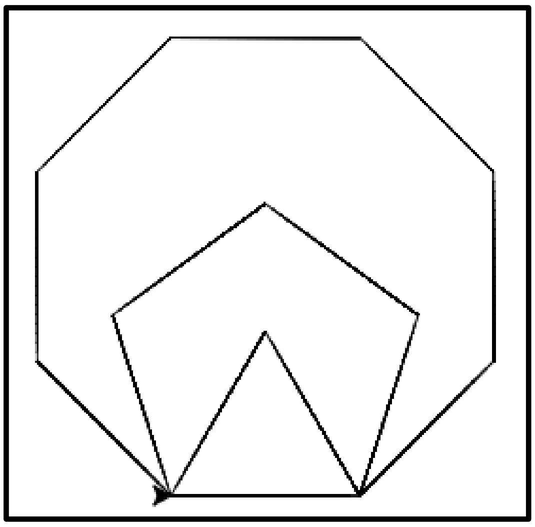


```
for i in range(4):
    turtle.forward(100)
    turtle.left(90)
```

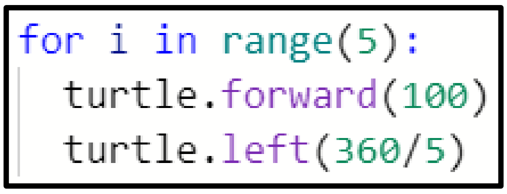
We can see that the code in the for loop is a lot shorter. Try changing this to draw a triangle, a pentagon, and an octagon. Be very careful that the code is indented the same as in example.

Challenge 1

Try drawing the shape below.



Can we make this code even easier to write? At the moment we need to calculate the number of degrees we need to turn for each shape. We can make the code do the maths for us. The code below will divide (/ symbol) 360 by the 5 (the number of edges) to find the number of degrees we need to turn.

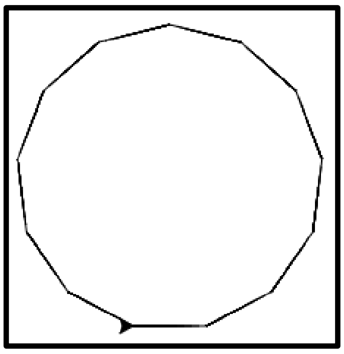


```
for i in range(5):
  turtle.forward(100)
  turtle.left(360/5)
```

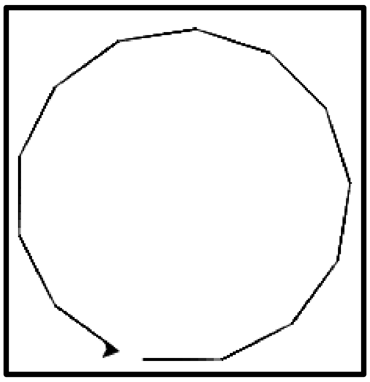
Now we only need to change the number 5 to however many of edges we want our shape to have and the code will do the rest.

Challenge 2

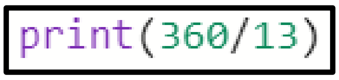
Draw a 13-sided shape and see what happens.



You will most likely find that you get something like the picture below. Why is that?



We are going to do our first piece of debugging now using the print command from the last lesson and we are going to write the code below and run it.



```
print(360/13)
```

What answer does our code give? Is this the correct answer?

This code will give the answer 27, but we can work out that the answer should be 27.692. This is a rounding error and is there because we are dividing two whole numbers and the computer thinks that the answer should be a whole number. If we change our code to the code below, we should get the correct answer. This is because by adding the decimal point we are telling the code that the answer can be a decimal.

```
print(360.0/13)
```

Homework Lesson 2

You need to be familiar with and know to use the following functions:

turtle.forward(100)	turtle.backward(50)	turtle.left(90)
turtle.right(45)	turtle.speed(10)	turtle.color(“orange”)
turtle.color(255,0,0)	turtle.penup()	turtle.pendown()
turtle.setposition(100,-50)	turtle.begin_fill()	turtle.end_fill()

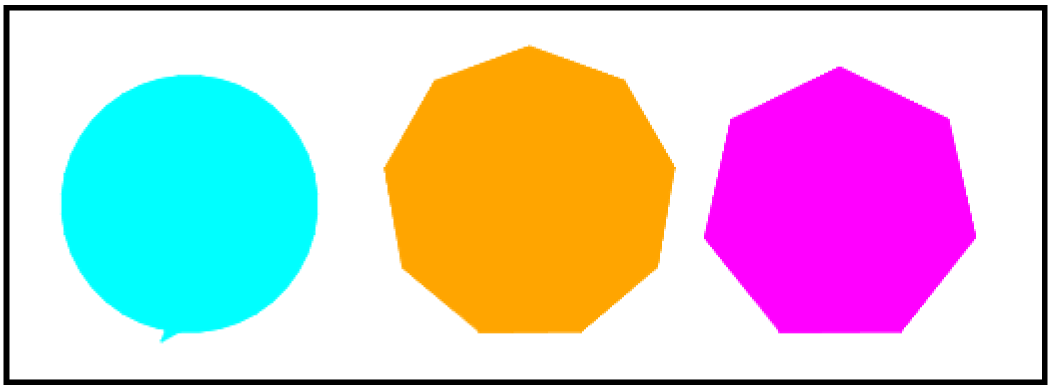
You also need to be able to create simple shapes using a for loop.

The example below shows a rectangle.

```
for i in range(4):
  turtle.forward(50)
  turtle.left(90)
```

Challenge 3

Using what you have learned, draw all of the shapes below in the same sketch.

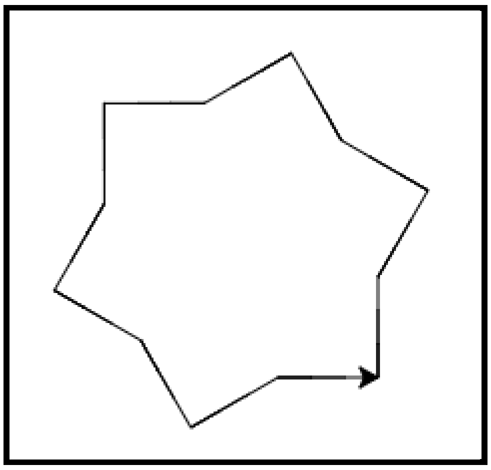


Test out all of these functions and see what they do.

- turtle.rt(100)
- turtle.fd(100)
- turtle.lt(100)
- turtle.bk(100)
- turtle.home()
- turtle.circle(100)
- Try
  - turtle.shapesize(1,5,10)
  - turtle.shapesize(10,5,1)
  - turtle.shapesize(1,10,5)
  - turtle.shapesize(10,1,5)
- Try
  - turtle.shape(“turtle”)
  - turtle.shape(“arrow”)
  - turtle.shape(“circle”)
- turtle.undo()
- turtle.clear()
- turtle.stamp()

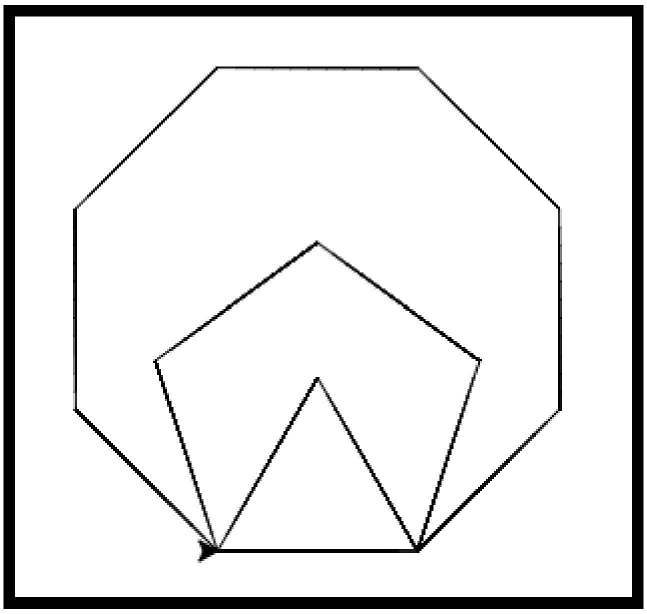
Challenge 4 (Extra)

Try to create the image below using for loops.

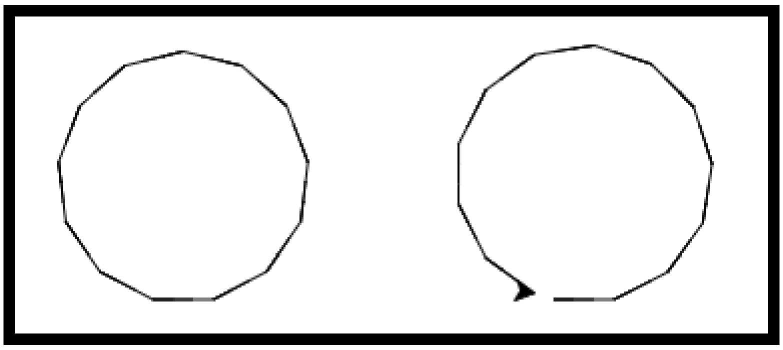


Lesson 2 - For loops and Variables Worked examples

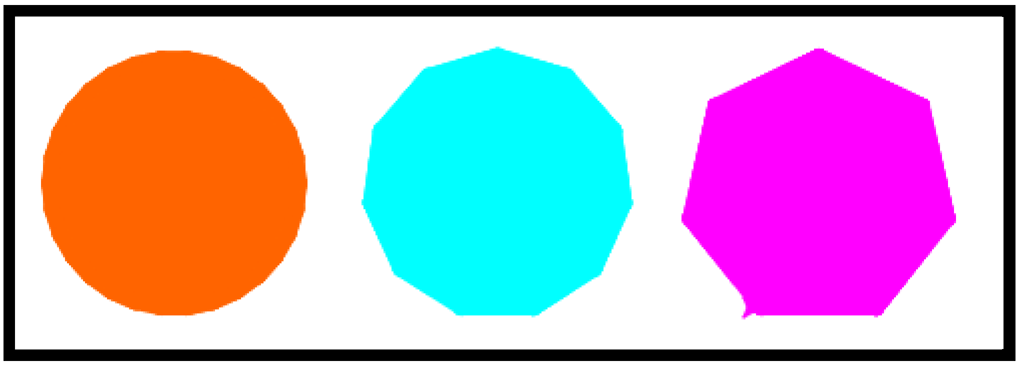
- Challenge 1 - https://repl.it/@DaireOM/Lesson-2-Challenge-1



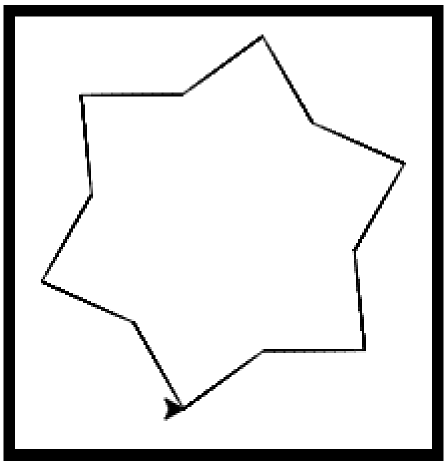
- Challenge 2 - https://repl.it/@DaireOM/Lesson-2-Challenge-2



- Challenge 3 - https://repl.it/@DaireOM/Lesson-2-Challenge-3



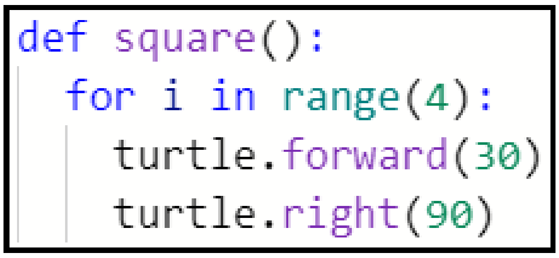
- Challenge 4 - https://repl.it/@DaireOM/Lesson-2-Challenge-4



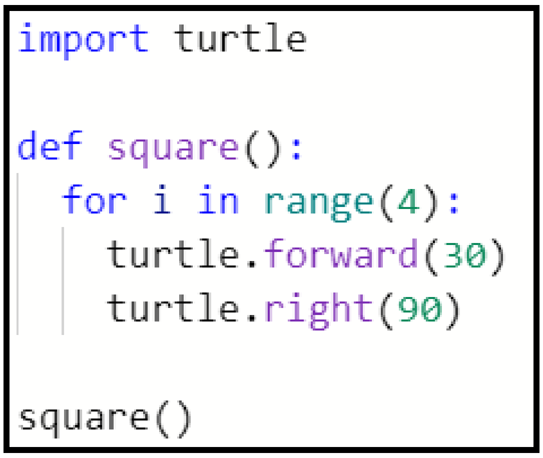
Lesson 3 - Subroutines/Functions

In this lesson we are going to learn about subroutines. Subroutines or functions allow us to break our code down into sections. These sections can then be reused elsewhere in our code and can save us time.

A subroutine starts with **def**, which stands for define. This tells the program that we are defining a subroutine. This is followed by the **name** of the subroutine, **2 brackets** and a **colon**.


```
def square():
  for i in range(4):
    turtle.forward(30)
    turtle.right(90)
```

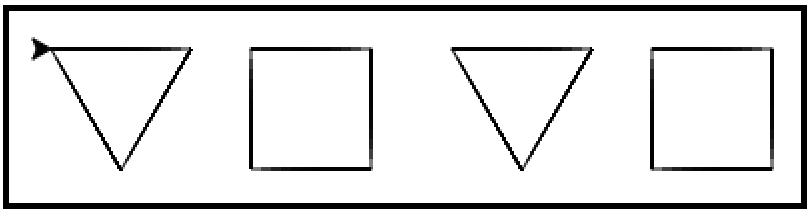
The subroutine above will draw a square. To run the subroutine or “call it” we simply type the name of the subroutine in our program.


```
import turtle

def square():
  for i in range(4):
    turtle.forward(30)
    turtle.right(90)

square()
```

The full program.

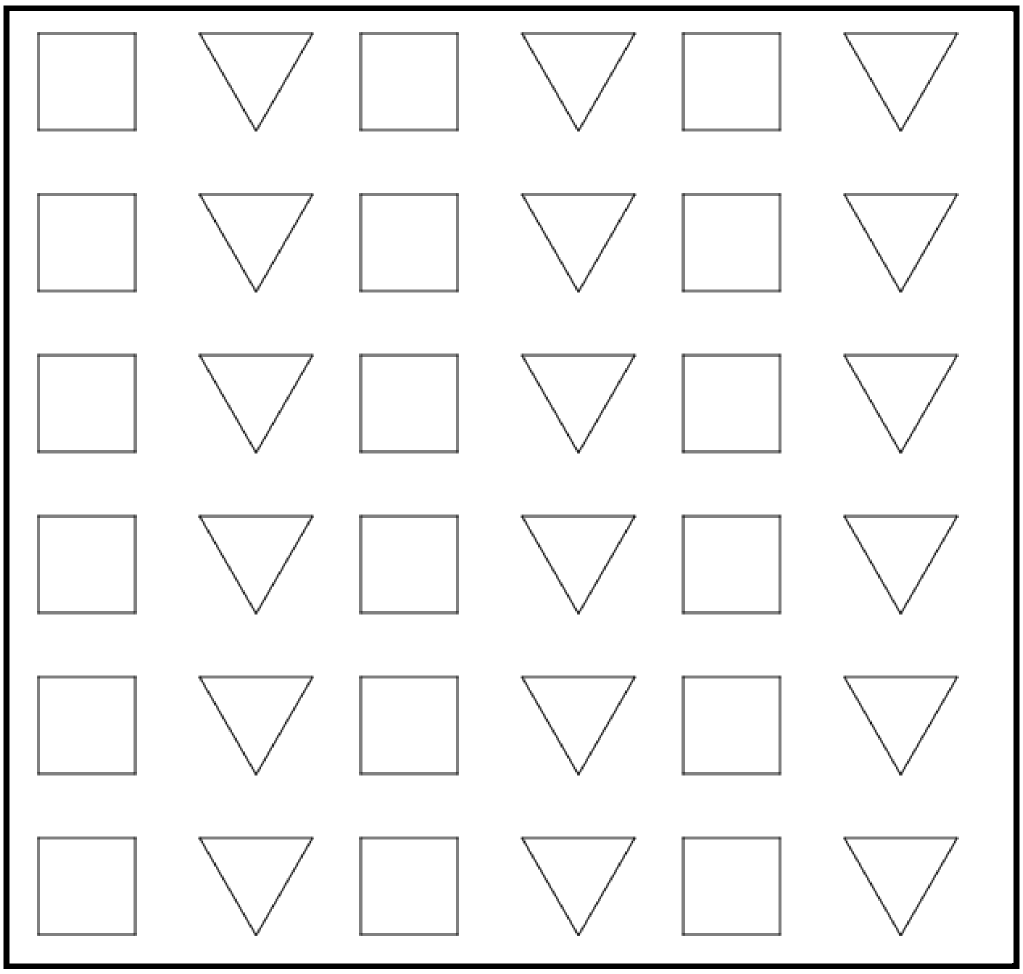
Challenge 1

- Run the program and ensure the subroutine works.
- Create a new subroutine to draw a triangle.
- Use both subroutines to draw two squares and two triangles in the pattern below.
- Use a for loop to draw this pattern.



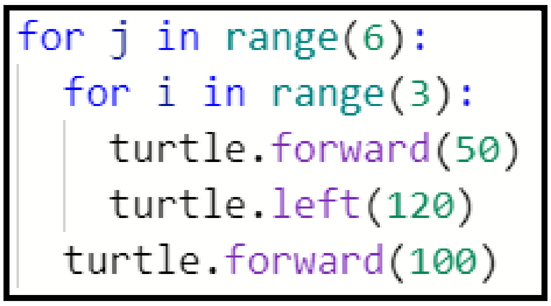
Challenge 2

Now that we have got a basic pattern done, we want to add to this pattern and create a pattern similar to the picture below.



What do we need?

- Create a subroutine to move the turtle to the top left corner of the screen.
- Create a subroutine to move to the position of the next shape in the pattern.
    - We will need to use **setheading(), penup(), pendown()** and **forward()**.
- Create a subroutine to move to the next row in the pattern.
    - We will move down the screen and all the way back to the left of the screen.
- Use a nested for loop to easily draw all the shapes. Below is a reminder of roughly what a nested for loop should look like.



```
for j in range(6):
  for i in range(3):
    turtle.forward(50)
    turtle.left(120)
  turtle.forward(100)
```

Random Colours

Now that we have all of our shapes, we are going to look at how to colour in each of these shapes a random colour.

To this we need to import the random library and use the randint function from it. There are two different methods of doing this.

```
import random
random.randint(0,255)
```

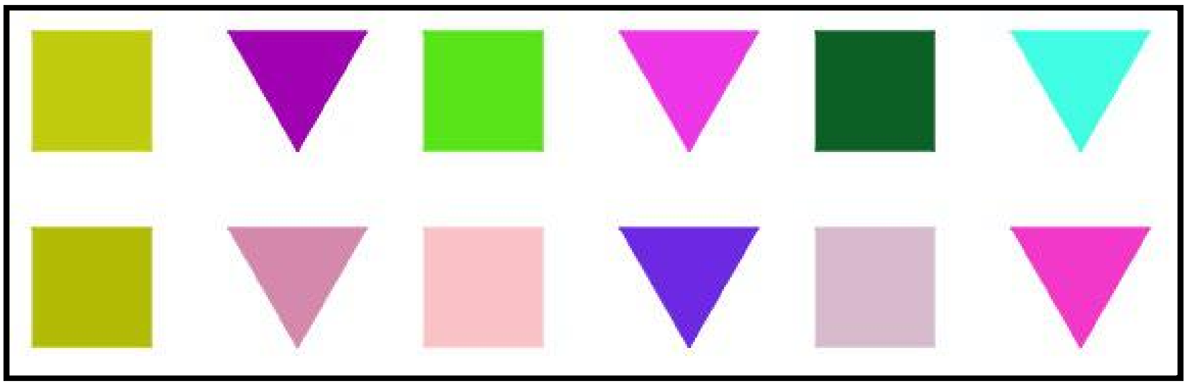
In this method we are importing the whole random library and then we specify that we want to use the function randint from the random library. Then we put the smallest number we want and the biggest number.

```
from random import randint
randint(0,255)
```

In this method we are only importing the randint function. Therefore, our program only needs the name of the function and the smallest number we want and the biggest number we want.

Challenge 3

- Use randint(), color(), begin_fill() and end_fill() to color each of the shapes a different colour.

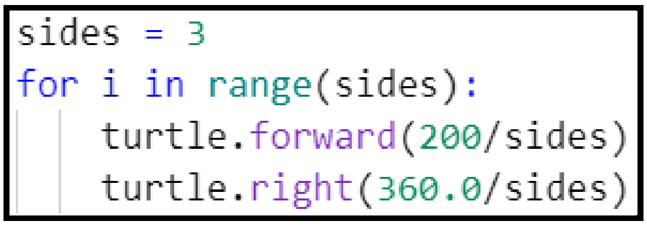


Passing Arguments

Although we now have subroutines to draw triangles and squares, we still don’t have any easy way of drawing other shapes, like pentagons and octagons.

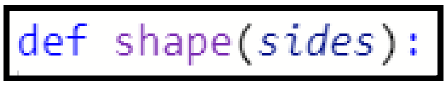
The subroutines that we wrote only do one thing and if we want to change it, we need to change the code completely.

Last week we learned about variables and how to use them to draw any regular shape we wanted. The code for this is below. All we needed to do to change the shape was to change the value of the sides variable.



```
sides = 3
for i in range(sides):
    turtle.forward(200/sides)
    turtle.right(360.0/sides)
```

We are now going to turn this into a subroutine, and we will pass in the argument for the number of sides. The first line of the subroutine is below, the rest of the subroutine will look similar to the code above.



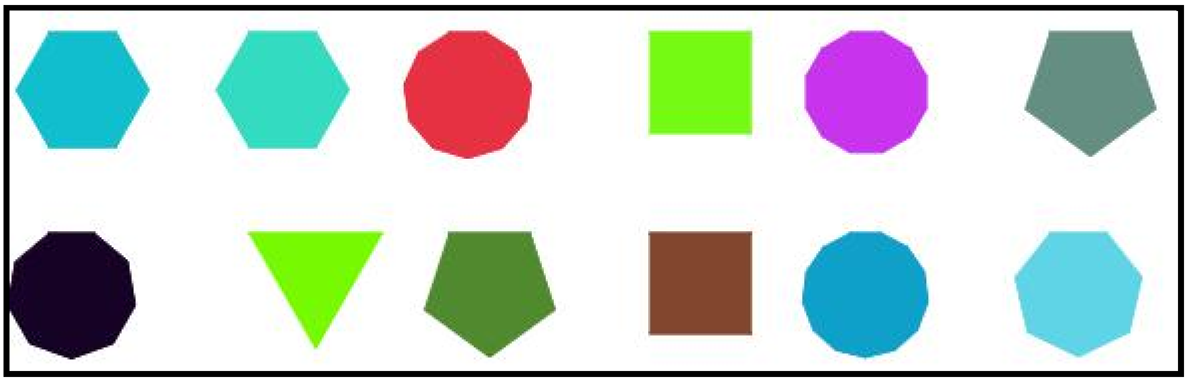
```
def shape(sides):
```

When we want to call the shape subroutine, we need to give it a number. It will look like this, shape(5).

Note: You can create subroutines that take more than a single number.

Challenge 4

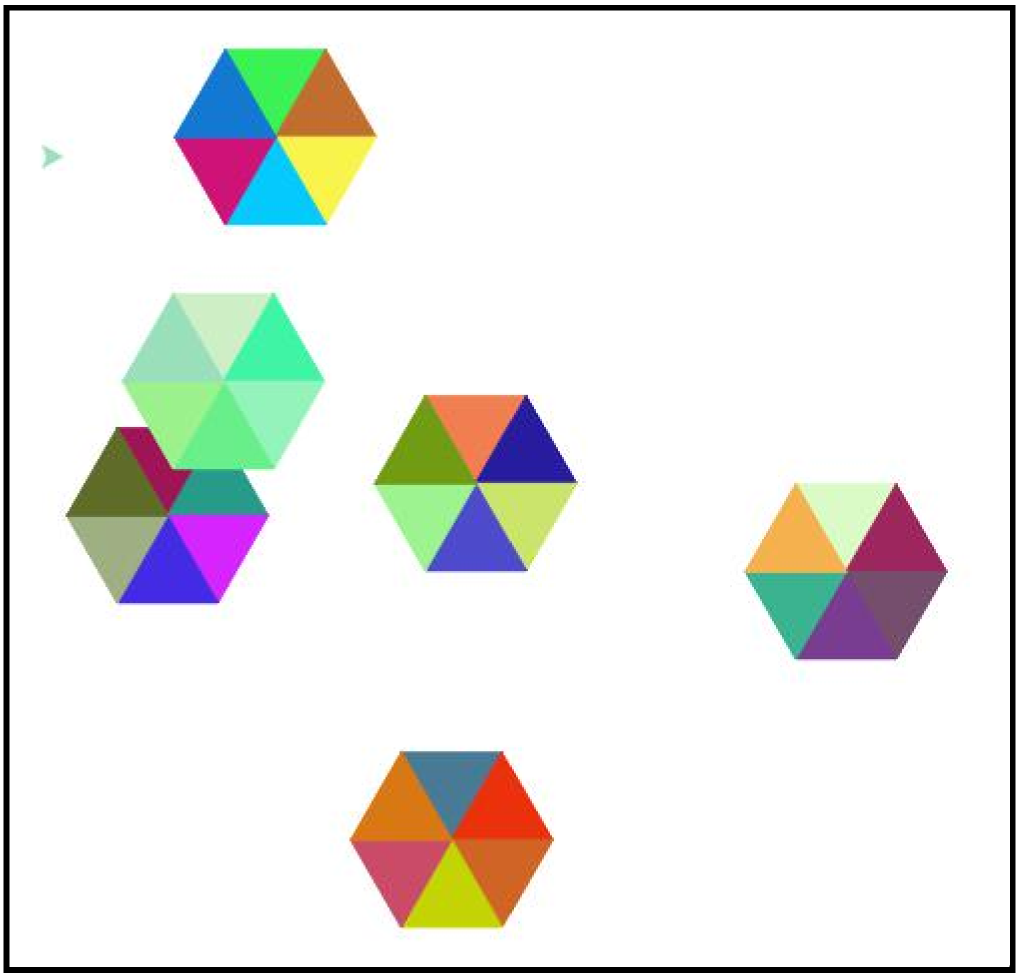
- Create a subroutine to draw regular shapes of any number of sides. Make sure that they are all roughly the same size.
- Using the randint function, change your code from the previous part to draw random shapes in a pattern like below.



Challenge 5
In a new program we are going to create the picture below. We are drawing the patterned hexagons 6 times and in random places.

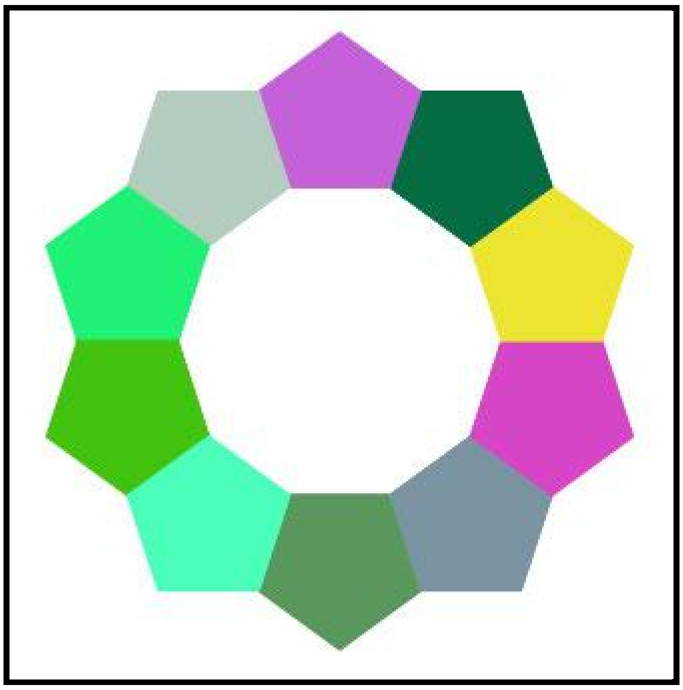
Tasks:

- Create a subroutine that will give us a random colour.
- Create a subroutine that will draw a triangle.
- Create a subroutine that will draw the hexagon pattern. You will need to use the triangle subroutine and it may be helpful to make the pattern without the subroutine or for loop first.
- Add code to move the turtle to random position after drawing each pattern.

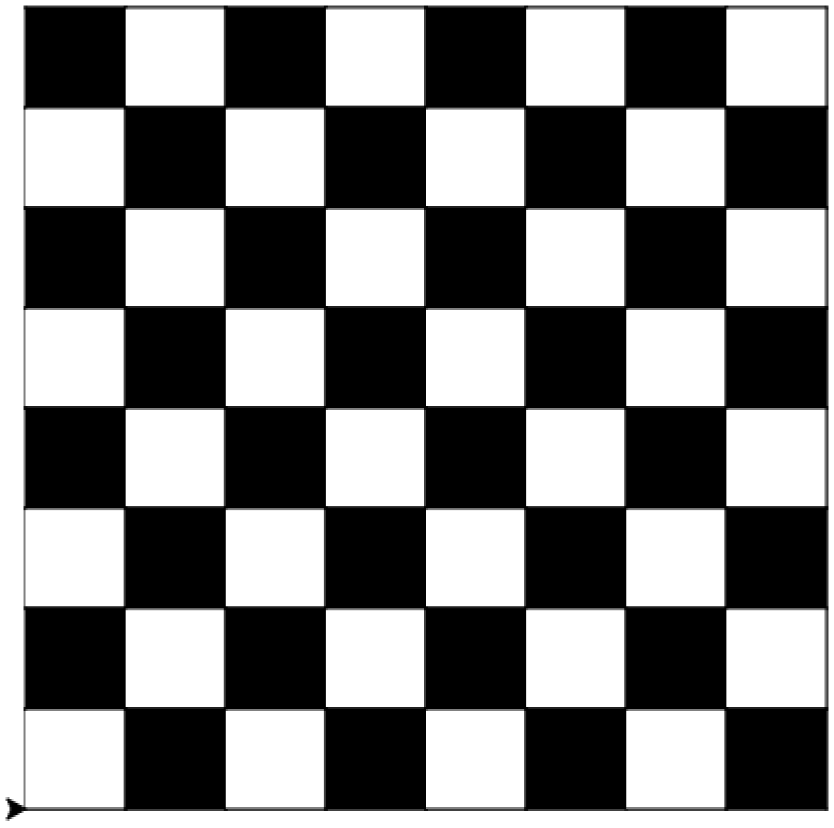


Challenge 6

In a new program create the pattern below. This will be quite similar to the triangular pattern made in challenge 1. Reuse code from previous lessons where appropriate.

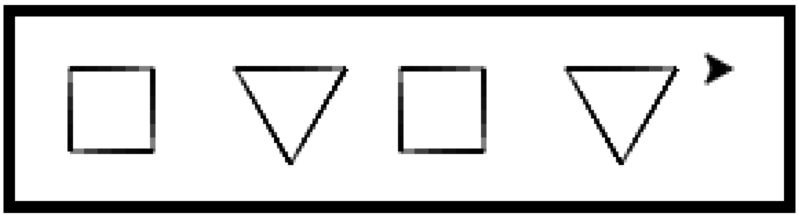


Challenge 7
In a new program create a chess board using what you have learned in this and previous lessons. Reuse code and create subroutines where applicable.

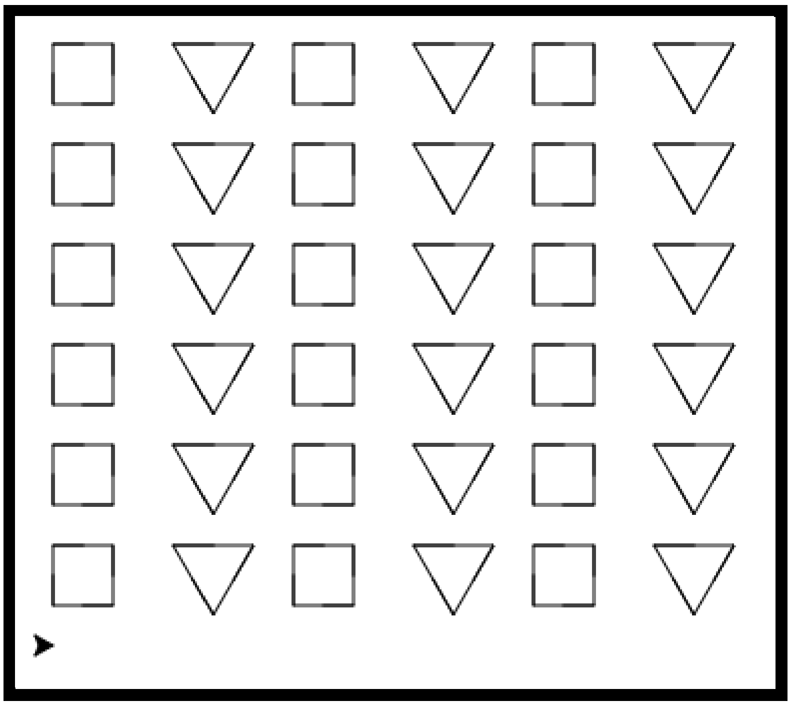


Lesson 3 - Subroutines/Functions Worked examples

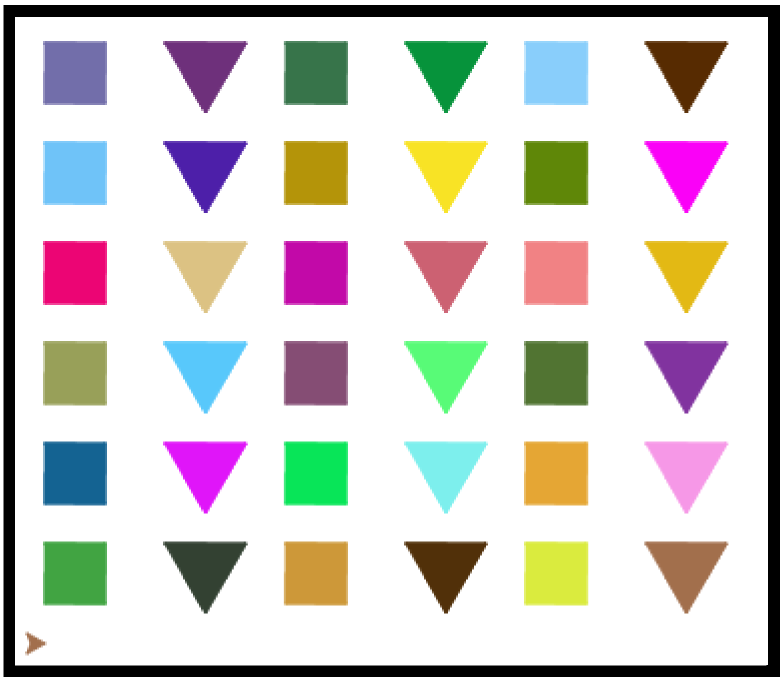
- Challenge 1 - https://repl.it/@DaireOM/Lesson-3-Challenge-1



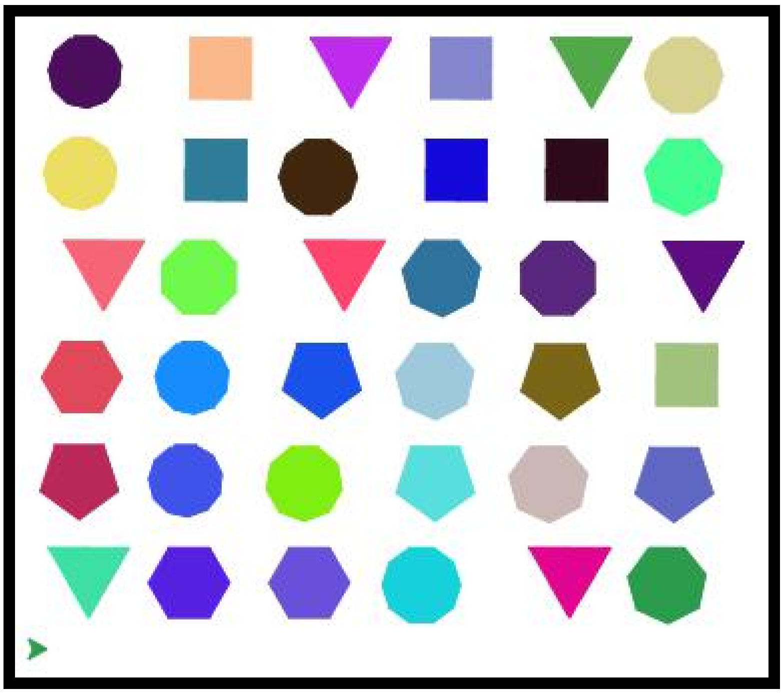
- Challenge 2 - https://repl.it/@DaireOM/Lesson-3-Challenge-2



- Challenge 3 - https://repl.it/@DaireOM/Lesson-3-Challenge-3



- Challenge 4 - https://repl.it/@DaireOM/Lesson-3-Challenge-4



- Challenge 5 - https://repl.it/@DaireOM/Lesson-3-Challenge-5