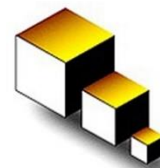
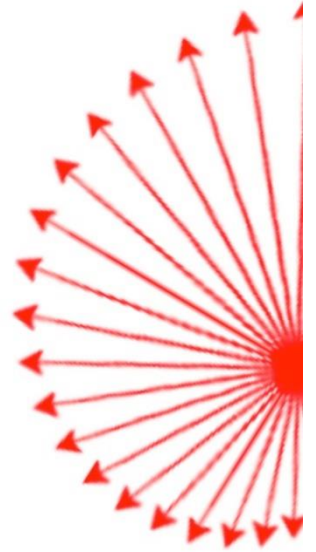
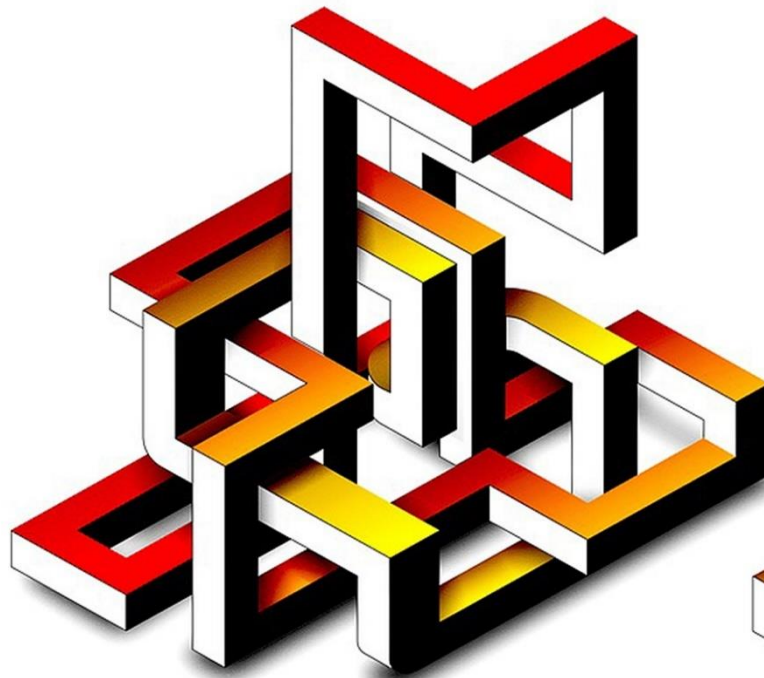
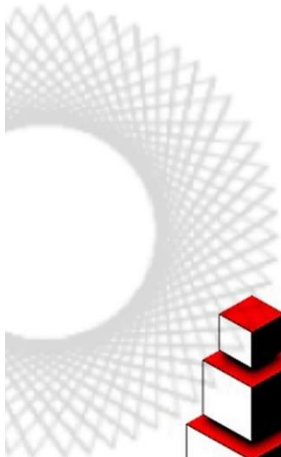




# Python Graphics

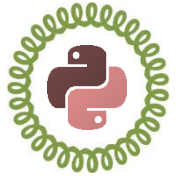


```
while True:
    try:
        sides = int(input("How many sides would you like?"))

    except ValueError:
        print("Please enter an integer.")
        print("")
        continue

    else:
        #the input was recognised, so exit the loop.
        break
```





We're going to create some patterns, pictures and other graphics using a programming language called Python. Python is one of the easier programming languages to learn as it doesn't have too many of the fiddly rules and squiggly brackets that can make coding difficult. However, the fact that it's easier to understand doesn't mean that Python is not powerful; it is used to code many of the world's biggest applications, including Dropbox, Instagram and Spotify. We will start with the very basics.

**Aim:** An introductory look at Turtle graphics and Python using a coding website.

## Task 1 – Coding Websites

We're going to create our solutions in a free coding website. These sites allow you to play around with bits of programming code without the need to install applications. For the tasks in these resources, we will use the language *Python with Turtle*. This will take the Python program and display the output as a graphic. You are essentially creating instructions to control the movement of a pen on a piece of paper.

There are numerous coding websites that you might select from. Here is a little information about a few of them:

[trinket.io](https://trinket.io)

A good solution for most tasks. Allows you to set up an account and store your programs online. You can also copy and paste the tasks into a Word document on your computer.

[pythonsandbox.com/turtle](https://pythonsandbox.com/turtle)

No accounts, but a simple interface. You will need to copy and paste your programs in order to store them.

[app.edublocks.org](https://app.edublocks.org)

A nice website with an account to store your scripts online. It is a bit slow to run programs, however, and seems to struggle with some fairly basic scripts.

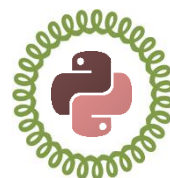
We will work in Trinket. There will be some differences if you select a different website but they all generally work in the same way.

- Navigate to <https://trinket.io> and follow the instructions to set up an account. We suggest you don't add personal information such as your real name.
- Click **New Trinket** and select **Python**.
- Carefully copy the code shown below into the *coding window*. When you are finished, click the *Run* button. If an error is reported, then check your code carefully and try again. Any mistakes in commas, brackets or colons will cause problems.



The left section of the screen is the *Coding Window*. This is where you type instructions, or *input*.

The right section is where the results are displayed. This is the *output*.



In the previous tasks, we learned about controlling the flow of instructions using count-controlled loops. We now have the knowledge needed to create some fantastic patterns

**Aim:** Using count-controlled loops to create patterns.

## Task 1 – Single Colour Patterns

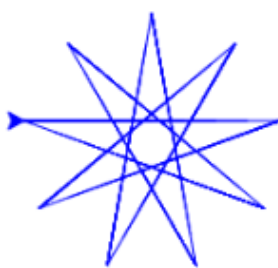
Have a go at creating the graphics below. Our turtle has now been called *Pat*. Save the programs with the names shown.



### 4.1 Five Point Star

Turns of  $144^\circ$ . We have called the variable 'i' but it can take any name.

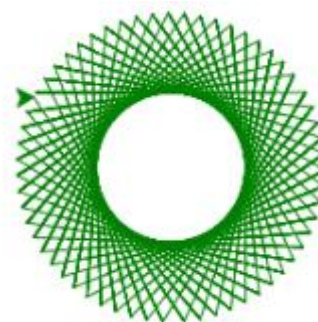
```
10 for i in range(5):
11     Pat.forward(150)
12     Pat.right(144)
```



### 4.1 Nine Point Star

Turns of  $160^\circ$   
Let's speed things up a little

```
6 Pat.speed(10)
```



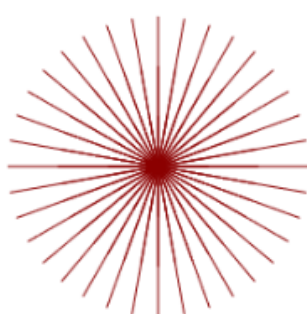
### 4.1 Many Pointed Star

Turns of  $122^\circ$   
You can always click 'Stop' if things are going wrong.



### 4.1 The Sun

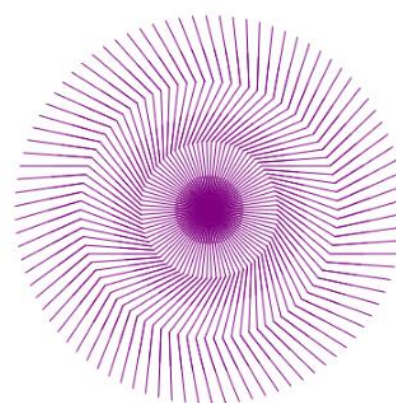
Turns of  $170^\circ$   
Go quickly around a couple of times.



### 4.1 Setposition

Use *setposition* to return the turtle to the centre after drawing a line

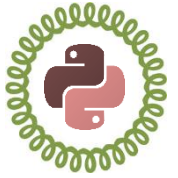
```
11 Pat.penup()
12 Pat.setposition(0, 0)
13 Pat.pendown()
14 Pat.right(10)
```



### 4.1 Ninja

Draw, return to centre, turn and repeat.





Nested loops are created when one loop sits inside another in a computer program. In Python with Turtle, nested loops can be used to make intricate patterns.

**Aim:** Using loops within loops to create more complicated patterns.

## Task 1 – The Turtle Army

The program below uses a loop to create 3 rows of turtles. It then uses an inner loop so that each of these rows contains 4 turtle stamps.

```

11  for y in range(3):
12
13      for i in range(4):
14          Pat.stamp()
15          Pat.forward(50)
16
17      Pat.backward(200)
18      Pat.right(90)
19      Pat.forward(50)
20      Pat.left(90)
21
22  Pat.hideturtle()

```

### Outer Loop

Set up a loop to create 3 rows of turtles

### Nested Loop

This section creates a line of 4 turtle stamps. Notice that the lines of code in this loop are double indented.

### Start next line

This section brings the turtle back to the start of the next line.

### hideturtle()

Use this method to hide the turtle once the pattern is complete.

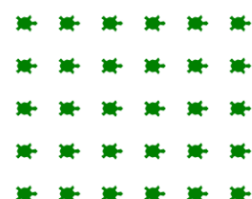


- Create the program above. Make sure that you use double tab spaces (or 4 normal spaces) for lines 14 and 15 as this is how Python identifies the inner loop. Save as “5.1 Turtle Army”.
- Which line sets up the loop for the 3 rows of turtles? Which line sets up the 4 columns?
- How many pixels does the turtle move forward in a single row before returning to the start of the next row?
- What does the turtle army look like if you delete line 22? Why do you think this happens?
- The program partially shown on the right achieves exactly the same turtle movements without using loops. Roughly how many lines would this program need to complete the pattern?

If you like, set up the full program and save as “5.1 Turtle Army Full”. Make sure you use the fork tool and the copy and paste shortcuts for a speedy job.

- Adapt the original program so that it creates 5 rows of 6 turtles, as on the right. You should only need to edit 3 lines of code from the program above.

**Note:** You will need to work out how far back the turtle must move in line 17.



```

Pat.stamp
Pat.forw

Pat.stamp
Pat.forw

Pat.stamp
Pat.forw

Pat.stamp
Pat.forw

Pat.stamp
Pat.forw

Pat.stamp
Pat.forw

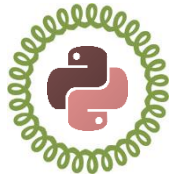
Pat.stamp
Pat.forw

Pat.stamp
Pat.forw

Pat.back
Pat.righ

```












The problem with the validation in the last activity is that we used code such as the line below to force a choice of either red, green or blue:

```
while line <> "red" and line <> "green" and line <> "blue":
```

**Aim:** To look at more advanced methods of validating user input.

Python includes a large number of named colours, so by only offering three, you are really limiting the options.

	brown		lemonchiffon		paleturquoise		indigo
	firebrick		khaki		darkslategray		darkorchid
	maroon		palegoldenrod		darkslategrey		darkviolet

How do we offer a greater choice of colours to our user? We could add more colours to the conditions set in the while loop, as below.

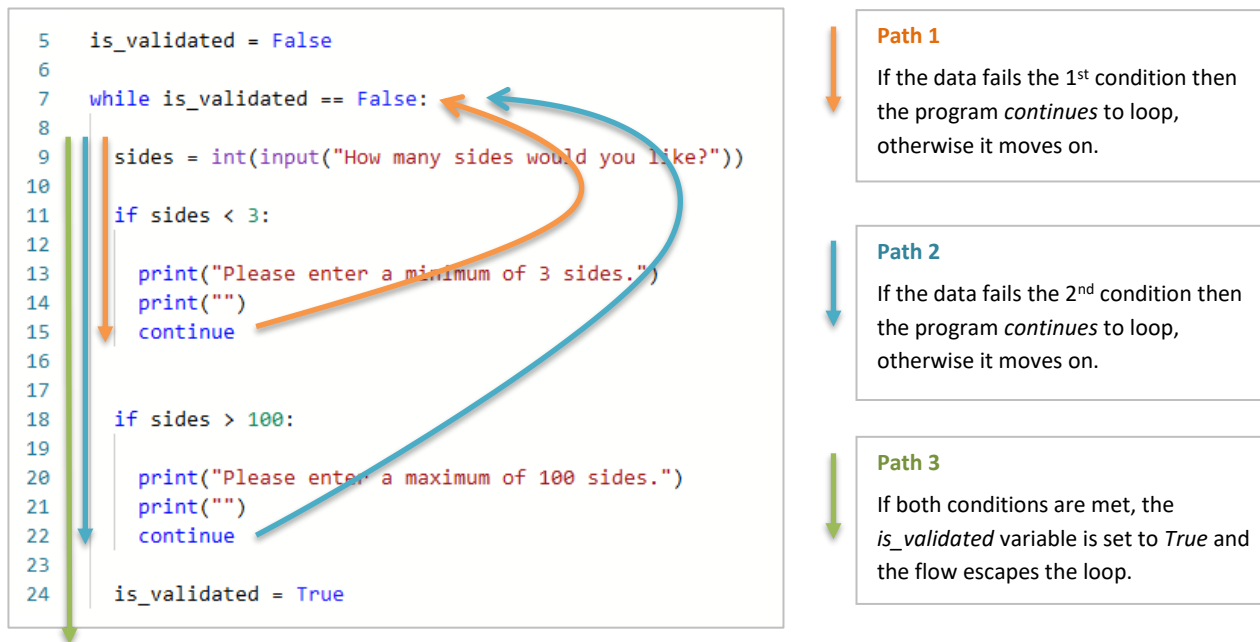
```
line <> "brown" and line <> "lemonchiffon" and line <> "paleturquoise" and line <> "indigo" and
```

However, this is both untidy and poor programming. We need a better method.

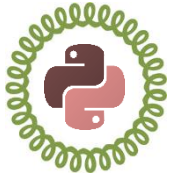
## Task 1 – Boolean Data and the Continue Statement

The *Boolean* data type can be one of only two possible values, *True* or *False*. A Boolean variable is useful in validation because we can start off setting it to *False* and then make it *True* if all the conditions are met, therefore exiting the *while* loop.

The *continue* statement sends us back to the start of the loop without executing the remainder of the code.



Fork the program named “10.3 Text Validation” and edit it to create the one above. Add comments to each line explaining how it works. Save as “11.1 Is\_Validated”.



Many simple games have action that takes place inside an enclosed space, often slightly smaller than the screen. We will look at various ways of setting up boundaries so that our turtle can only move within a confined space.

**Aim:** To create a confined space for the turtle to move in.

## Task 1 – Creating the Border

We will start by creating a rectangular border on the screen then return the turtle invisibly to the centre ready for action. These instructions will be placed in a separate function called *setup*.

The (partly hidden) program on the right sets up the border shown below. Create the program and save as “15.1 Border”.

### Notes:

- Decide on a background colour first and set this in place using code such as:  
`screen.bgcolor("orange")`

We will draw a rectangle over the background for our playing area.

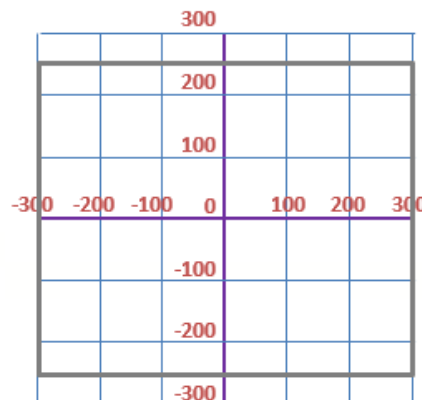
- Create the rectangle with a width of 600 pixels and a height of 500 pixels. The top-right corner will have the coordinates (300, 250). Use the *setposition* method, e.g.:

`Jill.setposition(300,250)`

*Note: You might find the code in the task “2.7 Position” a useful starting point for the rectangle. Open two browser windows so that you can easily copy and paste code from previous activities.*

- Fill the rectangle with white. Once this step is complete, lift the pen up and move the turtle back to (0, 0).
- Increase the speed of the turtle. Perhaps use the special speed of 0 to make things happen instantaneously.

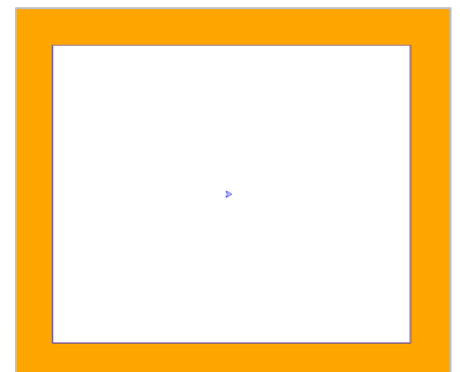
`Jill.speed(0)`

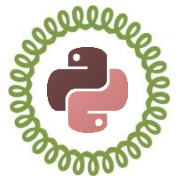


```

1  import turtle
2  Jill = turtle.Tur
3  screen = turtle.S
4
5  #DEFINE FUNCTIONS
6
7  def setup():
8      screen.bgcolor(
9      Jill.color("blu
10     Jill.penup()
11     Jill.speed(0)
12     Jill.setpositio
13     Jill.pendown()
14     Jill.fillcolor(
15     Jill.begin_fill
16     Jill.setpositio
17     Jill.setpositio
18     Jill.setpositio
19     Jill.setpositio
20     Jill.end_fill()
21     Jill.penup()
22     Jill.setpositio
23
24
25  #ACTUAL PROGRAM
26
27  setup()

```





### Task 2 – Testing the Border

Add a second function to your program. This one should test the boundaries, sending out lines until you hit the border and then stamping a mark before returning to the centre.

- Copy and paste the code from the task “8.4 Box” as a starting point.

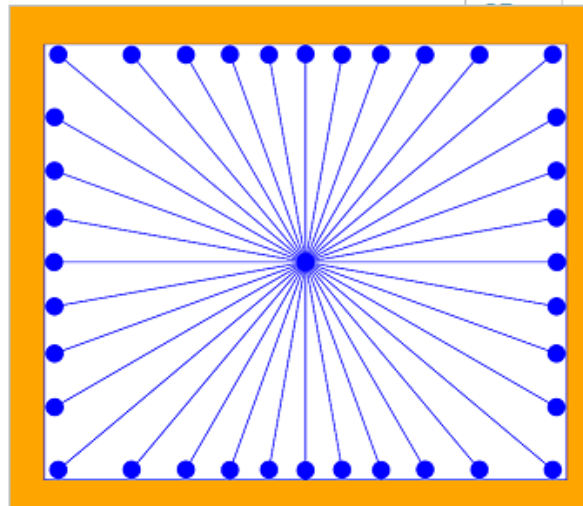
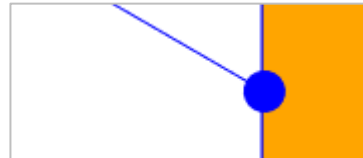
- A stamp placed with an x coordinate of 300 will be centred on the boundary line. Set the limits of movement slightly closer to the centre so that the circle stamp is placed inside the border. We found that an extra 12 pixels worked well.

`while xpos <= 288 and...`

- Moving the turtle forward 1 pixel at a time will give the neatest results, with the circles being stamped just touching the border. However, it will be a slow process (even if you have set the speed to 0). This is because the program is looping through the code hundreds of times for each line. Setting the movement to 5 pixels is much faster but a little untidy around the edges. We chose 2 pixels when creating the image on the right and waited patiently.
- A better solution might set a thicker line when drawing the border so that the end position is disguised. You may then be able to set the forward value to a much faster 5 pixels without it looking too messy.

Reduce the pen size back to 1 when you have finished drawing the border.

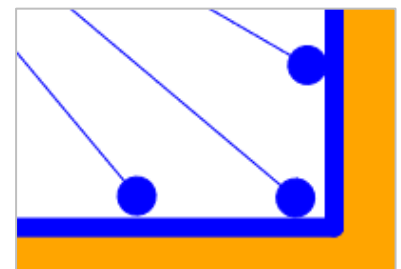
**Note:** When testing this method, we found we needed to further reduce the limits set in the while loop.



```

25
26 def testing():
27
28     Jill.pendown()
29     Jill.color("blue")
30     Jill.pensize(1)
31     Jill.shape("circle")
32     Jill.right(50)
33
34     for x in range(360):
35
36         xpos = 0
37         ypos = 0
38
39         while xpos <= 288 and ypos <= 288:
40
41             Jill.forward(2)
42
43             xpos = Jill.xcor()
44             ypos = Jill.ycor()
45
46             Jill.stamp()
47             Jill.penup()
48             Jill.setposition(0,0)
49             Jill.left(10)
50             Jill.pendown()
51
52
53 #ACTUAL PROGRAM
54
55 setup()
56 testing()
57

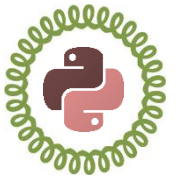
```



### Extension

Start thinking about how you might get the turtle to bounce off the wall. If you would like to have a go at producing a solution, fork your program so that you keep the above code intact.

We will work through the bouncing effect in a later task but it's always better to think about things yourself first.



Release in 1972, Pong was one of the first arcade video games. It consists of two player paddles and a ball that bounces around the court. We will build a basic version of the game.

**Aim:** To develop a version of the classic Pong game.

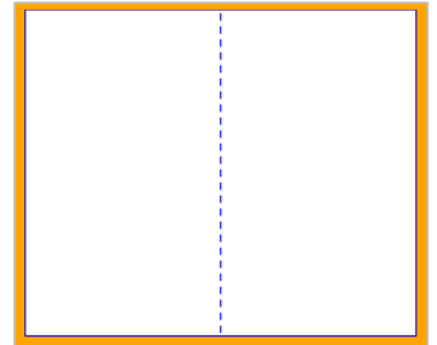
## Task 1 – Setting Up the Screen

- a. Create a new program named “17.1 Pong Setup” and set up the screen as on the right. It is a playing area as used before with a single dotted line down the middle (remember your count-controlled loops).

**Note:** The code in the program “15.3 Control” will be quite useful for this activity, so perhaps open it in a separate browser window.

Hide the turtle used to set up the screen when finished.

```
Jill.end_fill()
Jill.hideturtle()
```



- b. Add a second turtle for Player 1's paddle. This can simple be a square shape positioned just inside the left wall (see the picture below). Use the *penup* method so that the turtle doesn't leave a trace.

**Note:** The traditional paddle was actually a rectangle but (at the time of writing) this version of Turtle didn't recognise the *shapsize* method normally used to change the shape of the turtle. Our paddles will have to be squares.

```
# Player 1
Player1 = turtle
Player1.speed(-1)
Player1.shape("square")
Player1.color("blue")
Player1.penup()
Player1.setpos(-180, 0)
```

- c. Write a couple of functions that send Player 1's paddle up and down the playing area. The code from “15.3 Control” will be useful again here.

```
def P1_go_up():
    ypos = Player1.ycor()

    if ypos <= 130:
        Player1.setheading(90)
```

```
screen.onkey(P1_go_up, "a")
screen.onkey(P1_go_down, "z")

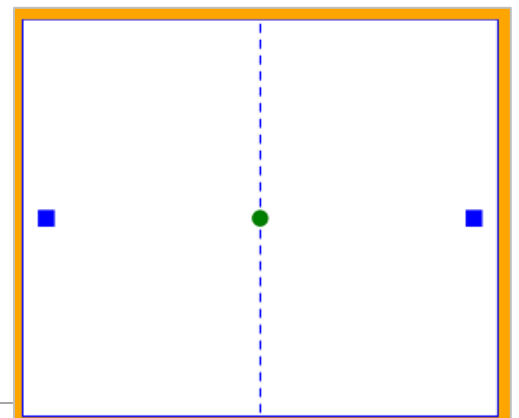
screen.listen()
```

Also, add the function calls and set the keys used for up and down. We have chosen 'a' and 'z'. Finally, make sure the screen is listening for events.

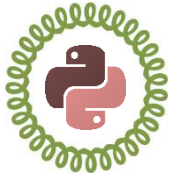
- d. Copy, paste and edit all the code necessary to create Player 2's paddle. Select some different control keys over to the right of the keyboard for the up and down movement (we chose 'k' and 'm').

- e. Add another turtle for the ball, positioned in the centre of the screen. Ours is a green circle.

```
# Ball
Ball = turtle.Turtle()
Ball.shape("circle")
Ball.color("green")
```







We're going to make a game where you catch turtles that are moving randomly around the screen.

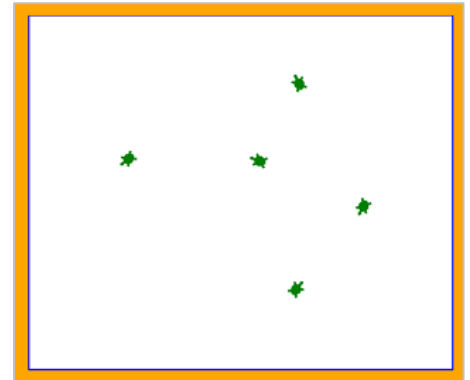
**Note:** Most of the programming ideas have been covered before but we will assume that you completed the extension task creating random turtles in the Lists resource. If you haven't done so, return to that activity before continuing with this one.

**Aim:** To create a game where players must speedily catch bouncing turtles.

## Task 1 – Setting Up the Game

- Fork your program "16.E Random Turtles" and save the new copy as "18.1 Turtle Catcher".
- Remove the request for user input and set the number of turtles to 5. You should also change the range of the loop to 5.
- Remove the colour lists (if used) and make all turtles green. They should also be turtle shapes. Lift the pen up so that no lines are left.
- Check that the program sets five green turtles off bouncing around the playing area.
- Rather than hiding the turtle used to set up the screen, place it in the bottom left corner ready for the game.
- Find one of your programs that controlled the turtle using the keyboard and copy the functions into your new program.
- Copy the *onkey* method calls into your *while* loop (or timer function) and make sure your program is listening.
- There is a very useful method called *distance*, which will tell you the distance between two objects at any point. The function on the right takes one of the turtle names as a parameter and checks how far the turtle is from the player (called Jill, in our case). If the distance between the player and turtle is less than 10 pixels then the turtle is hidden. Add this function to your game.
- This *distance* function should be called each time a turtle is nudged forward.

```
for i in range(5):
```



```
Jill.penup()
Jill.setposition(-100,-75)
```

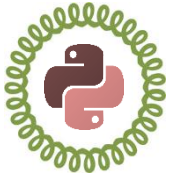
```
def go_up():
    ypos = Jill.ycor()
```

```
screen.onkey(go_up,"Up")
screen.onkey(go_down,"Down")
screen.onkey(go_left,"Left")
```

```
def distance(turtle):
    the_distance = Jill.distance(turtle)
    if the_distance < 10:
        turtle.hideturtle()
```

```
turtle_list[i].forward(10)
```

```
# Check for a collision between turtle and player
distance(turtle_list[i])
```



# Answers

## Task 1 – Controlling the Flow

Scenario
Keep rolling the die until you get a 6.
If you are a member then join the queue on the left, else join the queue on the right.
Go around the whole course three times.
First activities: Group 1 is climbing; Group 2 is kayaking; Group 3 is hiking.



Control in Place
Repeat something until a certain condition has been met.
Do one thing if one condition is true, otherwise do another thing.
Repeat something a set number of times.
Select from a range of possibilities depending on a condition.

## Task 2 – Looping through a Range

```
import turtle

Bethyl = turtle.Turtle()

Bethyl.color("blue")
Bethyl.pensize(5)
Bethyl.fillcolor("silver")

Bethyl.begin_fill()

for x in range(6):
    Bethyl.forward(50)
    Bethyl.left(60)

Bethyl.end_fill()
```



<https://trinket.io/python/49c1c12d10ee>

```
import turtle

Bethyl = turtle.Turtle()

Bethyl.color("orange")
Bethyl.pensize(10)
Bethyl.fillcolor("yellow")

Bethyl.begin_fill()

for x in range(8):
    Bethyl.forward(40)
    Bethyl.left(45)

Bethyl.end_fill()
```



<https://trinket.io/python/4adfe3bdd76c>

## Count-Controlled Loops Answers (page 2)

### Task 2 – Looping through a Range (Cont.)

```
import turtle

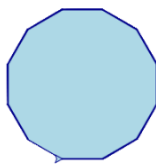
Bethyl = turtle.Turtle()

Bethyl.color("blue")
Bethyl.pensize(1)
Bethyl.fillcolor("light blue")

Bethyl.begin_fill()

for x in range(12):
    Bethyl.forward(30)
    Bethyl.left(30)

Bethyl.end_fill()
```



<https://trinket.io/python/55ce97c3faa5>

```
import turtle

Bethyl = turtle.Turtle()

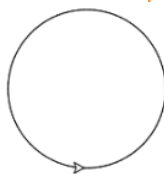
Bethyl.color("black")
Bethyl.pensize(1)
Bethyl.fillcolor("white")

Bethyl.begin_fill()

for x in range(90):
    Bethyl.forward(5)
    Bethyl.left(4)

Bethyl.end_fill()
```

<https://trinket.io/python/a399f439b607>



### Task 3 – Looping through a List

<https://trinket.io/python/e6b72d600380>

```
import turtle

Bethyl = turtle.Turtle()

for distance in [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120, 130, 140, 150, 160, 170, 180, 190, 200]:

    Bethyl.forward(distance)
    Bethyl.left(90)
```

## Count-Controlled Loops Answers (page 3)

### Task 4 – How the Test Program Works

```
1  import turtle
2
3  t = turtle.Turtle()
4
5  for c in ['red', 'green', 'yellow', 'blue']:
6      t.color(c)
7      t.forward(75)
8      t.left(90)
```

**Line 1** Imports the *turtle* module into the program.

This enables the program to understand the turtle language; it's like the translation guide. This line must stay in place whenever you write a turtle program.

**Line 3** Says *create a turtle and name it 't'*.

With our turtle ready to use, we can refer to it by the name 't' whenever we need to. You may change the name of your turtle if you prefer, providing you use the new name in the remainder of the code.

**Line 5** Places four colours in a list and sets up a loop.

There is a list of 4 colours to work through. The 'for c in' part of line 5 creates the loop. The program will run through the loop repeatedly, working through the list of colours until there are no more left.

The colour being used at any point is held in a variable called 'c'. A variable is like a box that holds a piece of data for later use (in this case, it is used again in line 6).

**Line 6** Gives the turtle (named 't') whatever colour is presently held in the variable 'c'.

The first time the program runs through the loop, the turtle is given the colour red. The second time, it is given the colour green etc.

**Line 7** Instructs the turtle to move forward 75 pixels.

**Line 8** Instructs the turtle to rotate left (or anticlockwise) through 90 degrees.

### Questions

- In which line is the *turtle* object created (or defined)?  
**Line 3**
- What happens if you delete line 1 and run the program? Look at the error description in the console. Why do you think this happens?  
**NameError: name 'turtle' is not defined on line 3. We haven't imported the turtle module so the program doesn't understand what the word 'turtle' means.**
- How many times will line 6 be read by the computer when the program is run?  
**4 times**
- What happens if you misspell one of the colours?  
**The last known colour is used again. If you misspell the first colour in the list, then black is used for that line instead.**
- What do you think would happen if you simply add more colours to the list without changing anything else? Try it.  
**The lines are redrawn with the new colours.**