1. Intro Turtle Graphics

1.1. Introduction to Python Turtles

Welcome to the **Code Avengers Introductory Turtle Challenge**!

In this activity you will learn how to use Python code to draw images with a **Turtle**.

Turtle graphics started in the 1960s and will help you understand **sequence** in programming.

We've named our turtle `talia`, and the code in the editor shows you how to create her by first **importing** the `turtle` module.

Now imagine `talia` is on a canvas holding a variety of pens and buckets of paint, and can move and draw depending on the commands you give.

Lines 4 and 5 shows you how to **move `talia` forward 150 pixels** and **turn `talia` right by 90 degrees**.

Click to see what happens with the existing code.

1. Repeat lines 4 and 5 of the code so that `talia` draws a **square.**
2. Click to see the image and when you are done.

1.2. Changing the style of the lines

You've turned `talia` right and moved her forwards, and in this task you'll use `talia.left()`.

We'll also change the **color and size** of the line that gets drawn by using: Insert
```
talia.pencolor("red")
talia.pensize(6)
```

There are lots of different colors we can use other than red, and we can use any positive whole number for the line size.

This time we're going to style the lines and draw a **hexagon**.

1. On line 3 set the **pen color** to **darkViolet**.
2. Below that, set the **pen size** to **8**.

We've included the first couple of lines of code for a hexagon.

3. Complete the code to draw the whole hexagon.

1.3. Moving to an exact position

Right now our shapes are being drawn in one corner of the window.

Sometimes we might want to start drawing a shape from a particular position. We can use a **function** called `goto()` to do this:
```
# Move the turtle to the top left corner
talia.goto(40, 40)
```

The **coordinates** say **how far across** and **how far down** to put the turtle.

If we use `goto()` just by itself, however, the turtle will draw a line while it's **"going to"** the position. You can see this by clicking and using the slider.

To avoid this, we can use two other functions: `penup()` and `pendown()`.

Let's use these to draw an envelope.

1. Before the `goto()`, on line 6, **lift the pen up.**
2. After the `goto()`, on line 8, **put the pen down.**
3. Use `forward()` and `right()` to draw a *rectangle* that is **320 pixels** wide and **240** high.
4. Use `goto()` to send `talia` to the position `(200, 180)`.
5. Use `goto()` to send `talia` to the position `(360, 80)`.

1.4. Filling shapes with colors

We can also **fill shapes** with a color. There are 3 commands we need for this:
```
tom.fillcolor() # sets the color for filling.
tom.begin_fill() # goes before the start of the shape.
tom.end_fill() # goes after the end of the shape.
```

If we want the **line and fill color** to be the same, we can use `tom.color("blue")`.

In this task you'll draw a window. You might find it useful to the code first to see what the solution image looks like. We've drawn the first shape for you.

1. Under the **comment** `# Draw the glass` and the `goto()`, set the color to **lightCyan**
2. Use `begin_fill()` and `end_fill()` to draw a filled rectangle that is **200px wide and 280px high**.
3. Use `penup()`, `pendown()` and `goto()` to add the inner frames using the **coordinates** below.

Coordinates for inner frames: `(200, 50)` `(200, 350)` `(90,200)` `(300, 200)`

1.5. Draw the flag of Mali

Let's get a little more practice with what you've learned by drawing the flag of Mali.

For this task you will need the following functions, and there is a hint to help you below: `forward()`
`right()`
`penup()`
`pendown()`
`goto()`
`color()`
`begin_fill()`
`end_fill()`

Click to see what the existing code looks like.

1. Replace the numbers on lines 11, 13, 15 and 17 to correctly draw the **limeGreen** rectangle.
2. Fix up the coordinates on line 23 to start the **gold** rectangle in the correct position.
3. Fix the numbers on lines 28, 30, 32 and 34 to make the **gold** rectangle the correct size.
4. Copy, paste and change the code for one of the other stripes to draw the **red** rectangle under the last comment.

Hint: The **limeGreen**, **gold** and **red** stripes are all **100px wide** and **240px high**.

1.6. Using loops to repeat code

The code for drawing a square can be pretty repetitive–just imagine how long our code would be if you wanted to draw a **dodecagon**!

In programming, **loops** are structures that will repeat code for us. Python has a **for loop** that looks like this: `for i in range(4):`
`  # Do something`

This loop would repeat **4 times** because of the `range(4)` part.

To use **loops** with turtle graphics, you have to think carefully about **which steps** need to be repeated.

These steps go inside the loop where it says `# Do something` and they are **indented** to show they are **inside the loop**.

To draw a regular shape, inside the loop you will need the line of code that moves `talia` forward, and the line that turns her the correct angle.

In the editor is the code for a square like you saw in task 1.

1. On line 8 write a **for loop** statement that will repeat **4 times**.
2. Edit the remaining code so that the loop will draw the square with only a couple of lines of code.

1.7. Draw a Rubik's cube

Great, we can use a loop to draw a square! Let's get some practice. What better picture to practice drawing squares than a Rubik's Cube?

For this task we'll just draw one face of the cube. We'll use **black** for the line color and a range of colors for the fill, as listed below.

We have drawn the first row of squares for you. Use this code, **for loops**, the comments in the code, and the information below to complete these tasks:

1. Draw the squares in the second row.
2. Draw the squares in the third row.

Size of squares: 80px.

Color order:

orange lime blue

yellow orange red

blue yellow white

Fun fact: Erno Rubik invented the Rubik's Cube in 1974. When he first invented it, he didn't know how to solve it either!

1.8. Using a list to run a loop

We can use a number to run a loop, as we have seen when using `range(4)`. Another way we can run a **for loop** is using a **list**: `lengths = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120, 130]`

```
for length in lengths:
  tia.forward(length)
  tia.right(90)
```

A list is a set of values stored together.

In this example we have a list of `lengths`. To create the loop we write: `for length in lengths`

This lets us use each value in the list by typing the word `length` wherever we want to use it. The loop repeats once for each value, so the image at the top is the end result.

You will draw a square with 4 different colored sides using the `colors` list.

1. Under the last comment write a **for loop** statement that will repeat for each `color` in the `colors` list.
2. Inside the loop, set the color for the turtle.
3. Then write the code needed for the turtle to draw a square with **320px sides**.

1.9. Drawing circles

Python turtles also have a built in function called `circle()` which, you guessed it, let's you draw circles.

The code that follows will draw a circle with a **radius** (half the circle's width) of 60px: `tia.circle(60)` .

The circle is always drawn on the **turtle's** left. You can see this in the image at the top. If the turtle is facing a different direction, the circle will be drawn in a different place.

To fill a circle, do the following: `tia.begin_fill()`
`tia.circle(50)`
`tia.end_fill()`

So, let's draw a surprised snowperson!

We have drawn the bottom circle for you. You can use this code to help with these tasks:

1. Move the turtle to `(200, 250)` and draw the middle circle with a **radius of 80px**.
2. Next, move the turtle to `(200, 130)` and draw the head circle with a **radius of 60px**.
3. Set the fill color to **black** then draw the left eye at `(180, 60)` with a **radius of 6px**.
4. Draw the right eye the same size at `(220, 60)`
5. Finally, draw the mouth at `(200, 100)` with a **radius of 15px**.

We can also draw **semicircles** by putting a second number inside the brackets: `tina.circle(60, 180)`

This code draws half of a circle, because we've told the turtle to only draw **180 degrees** of it.

Because circles are always drawn on the **turtle's left**, sometimes we might want to tell the turtle exactly which direction to face.

You can see this on line 19: `talia.setheading(90)` points `talia` straight up when you run the code.

To finish off, let's draw an umbrella using semicircles.

> We've drawn the background and the first semicircle for you. There's also a list of **x positions** for you to use in a loop to draw the bottom of the umbrella.

1. On line 28 set the turtle color to **lightSkyBlue**
2. Write a loop that will repeat for each `x` value in the `positions` list.
3. Inside the loop, move Tia to `(x, 181)`, use `setheading(90)` to point her upwards, then draw a **filled 180 degree** semicircle with a **radius of 30px.**
4. Under the next comment set the pensize to **15**, the color to **black** and use `goto()` to draw a line from `(210, 150)` to `(210, 340)`.
5. Under the next comment, draw an **unfilled 180 degree** semicircle with a **radius of 20px**.
6. Finally, uncomment the very last line by deleting the `#` at the start.

Look, `talia` is hiding from the rain! Congratulations on completing this course. If you enjoyed it you might like to try some of our other courses!