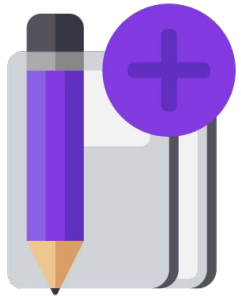
Brainstorming:

The turtle module

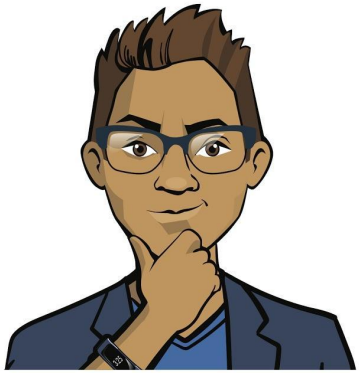


Working with graphics

Before exploring the commands for working with graphics, let's discuss how images work in the computer's memory.

We've already discussed that all **data in the computer's memory is stored in the form of ones and zeros** – "signal" or "no signal."

But how do we encrypt an image using zeros and ones?

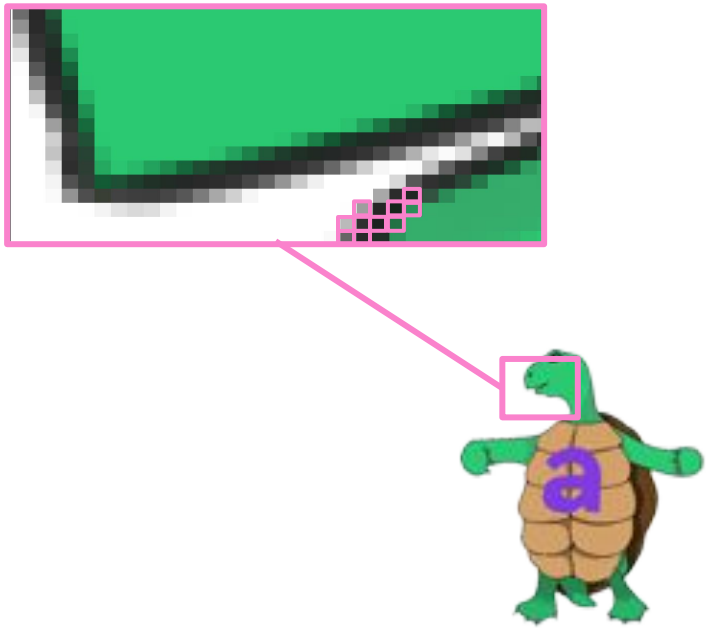


Brainstorming

A pixel
is the minimum (indivisible) part of a graphic image

A **bitmap** is a collection of pixels.

A **bitmap image** is a collection of dots (pixels) used to display an image on a computer screen.

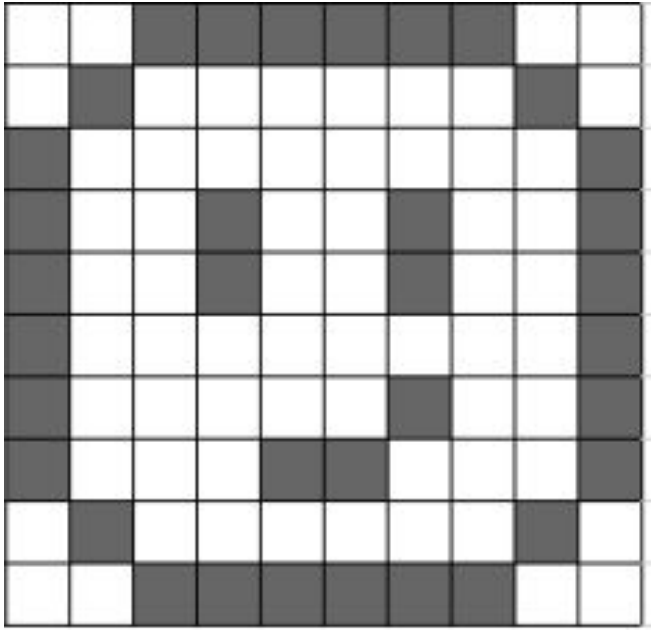


The turtle module works with bitmap graphics

Brainstorming

Working with bitmap graphics

If monitors were black and white, information about a pixel would be stored as zero (“no color”) or one (“color”).



Modern monitors are **color**. The color of a pixel is encoded with a set of zeros and ones.

Brainstorming

Working with bitmap graphics

Fortunately, we don't need to remember sequences of zeros and ones to set the color of a geometric shape.

The interpreter recognizes lots of colors by their names:

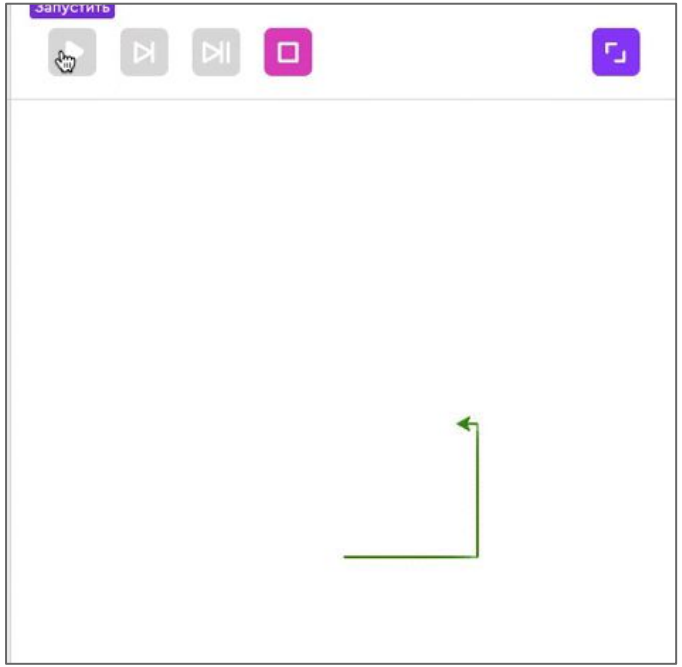
Color	*Name*
red	`"red"`
green	`"green"`
blue	`"blue"`
yellow	`"yellow"`
black	`"black"`

Color	*Name*
pink	`"pink"`
light blue	`"light blue"`
orange	`"orange"`
lime	`"lime"`
violet	`"violet"`

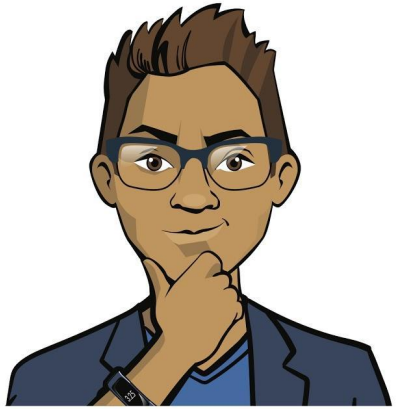
Brainstorming

Drawing a bitmap image

Turtle module graphic objects are drawn in a separate part of the window by a special executor (command executor) – a **turtle** .



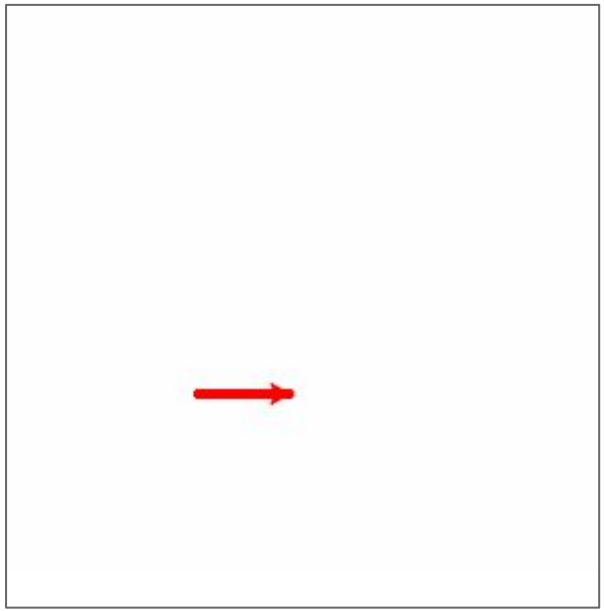
The turtle is shown on the platform with an arrow by default.



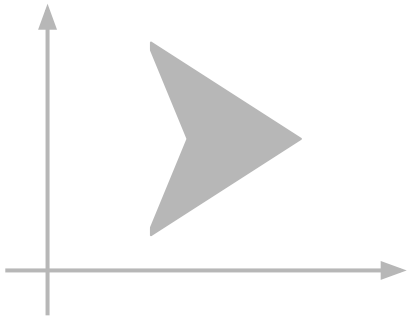
Brainstorming

Drawing a bitmap image

Turtle module graphic objects are drawn in a separate part of the window by a special executor (command executor) – a **turtle** .



Initial position of the executor when starting the program:



Brainstorming

Drawing a bitmap image

Connecting the turtle module commands:

```
from turtle import *
```

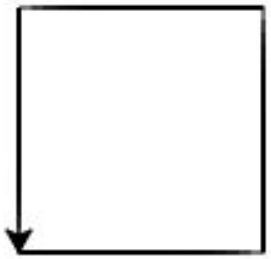
Basic commands:

Command	*Purpose*
`forward(<number of pixels>)`	Move the turtle **forward** the specified number of pixels
`left(<number of degrees>)`	Turn the turtle **left** the specified number of degrees
`right(<number of degrees>)`	Turn the turtle **right** the specified number of degrees
`color(<color name>)`	Set a **new color** for the ACExecutor tor with the specified name
`exitonclick()`	**Keep the image on the screen** after the program has been executed

Brainstorming

Let’s go over the task

Task. The customer prefers minimalism and wants to buy tile with a square pattern. Write a program that draws a black square with no fill and with a side of 150 pixels.



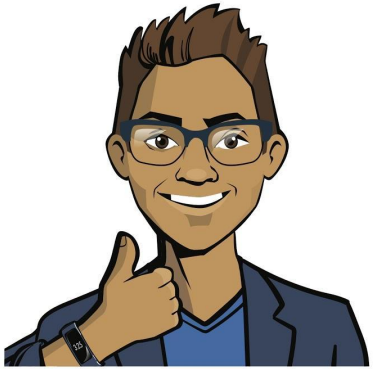
How do we connect the turtle module? What commands should we use?

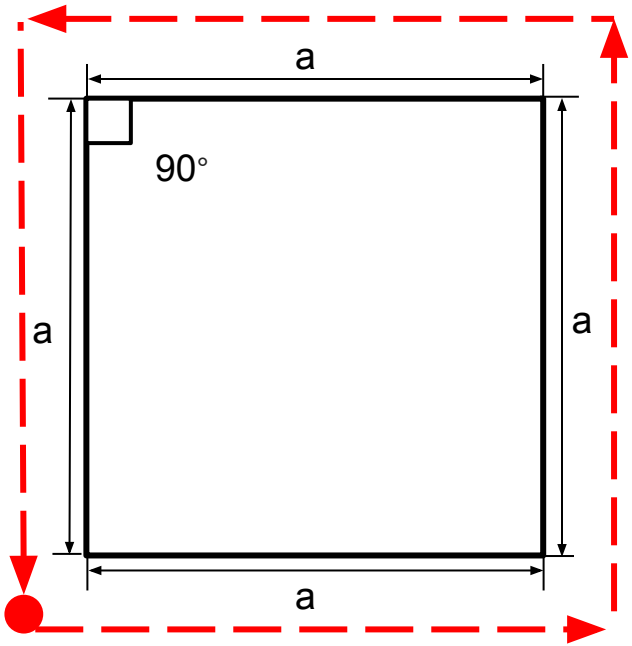
Brainstorming

Let’s go over the task

Task. The customer prefers minimalism and wants to buy tile with a square pattern. Write a program that draws a black square with no fill and with a side of 150.

Sample solution:



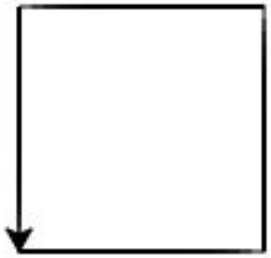


Brainstorming

Let’s go over the task

Task. The customer prefers minimalism and wants to buy tile with a square pattern. Write a program that draws a black square with no fill and with a side of 150 pixels.

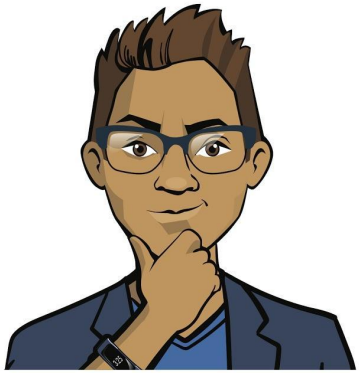
```
from turtle import *
forward(150)
left(90)
forward(150)
left(90)
forward(150)
left(90)
forward(150)
exitonclick()
```



Brainstorming

Before we continue:

1. **How can we change the program** to make the turtle draw a red square instead of the black one?
2. The client changed their mind and decided to design the tile with smaller squares with a side of 90. **How many lines of code do we need to change? How?**



Brainstorming

Drawing a bitmap image

Connecting the turtle module commands:

```
from turtle import *
```

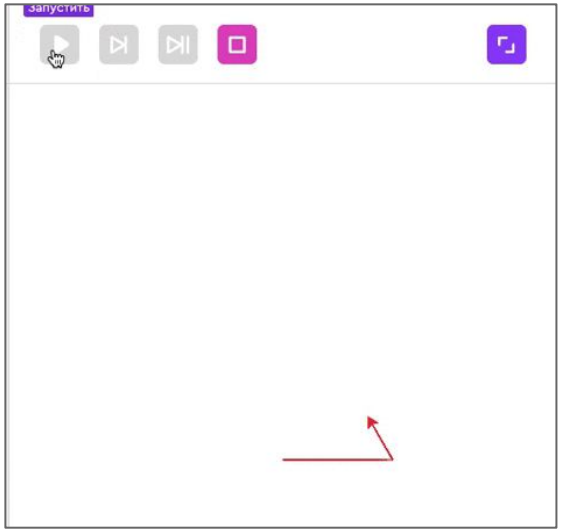
Another set of commands:

Command	*Purpose*
`pensize(`<number of pixels>`)`	Change the size of the pen the executor is drawing with (initially it's 1)
`circle(`<circle radius>`)`	Draw a circle with the given radius in pixels

Brainstorming

Let’s go over the task

Task. The customer wants tile with triangles. They have not yet decided on the color, so we need to program both red and blue. The triangle side length is 110 pixels.



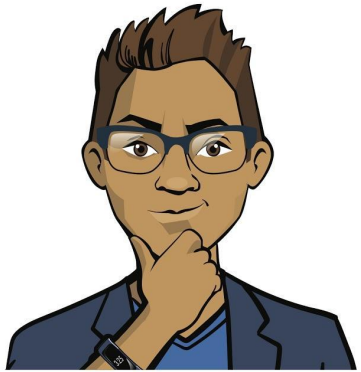
How do we draw the triangle? What commands should we use?

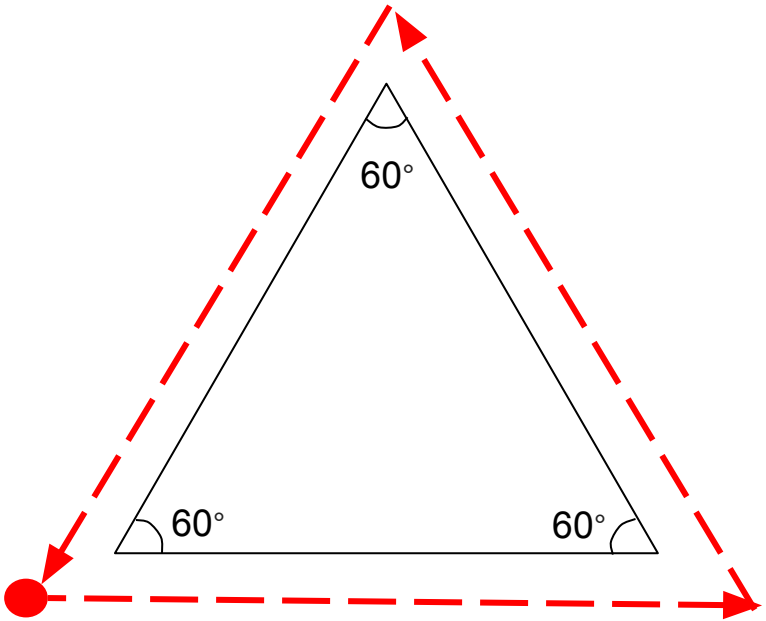
Brainstorming

Let’s go over the task

Task. The customer wants tile with triangles. They have not yet decided on the color, so we need to program both red and blue. The triangle side length is 110 pixels.

Sample solution:

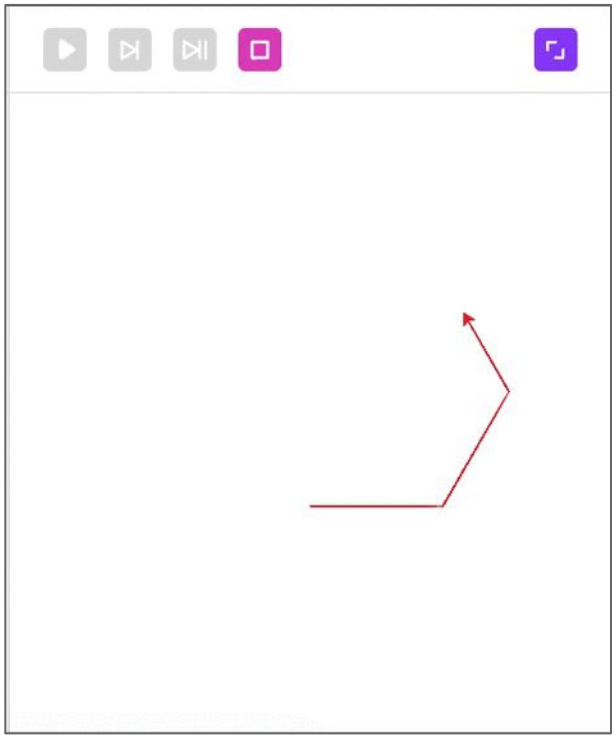




Brainstorming

Sample solution:

```
from turtle import *
color('red')
forward(110)
left(60)
forward(110)
left(60)
forward(110)
left(60)
color('blue')
forward(110)
left(60)
forward(110)
left(60)
forward(110)
exitonclick()
```



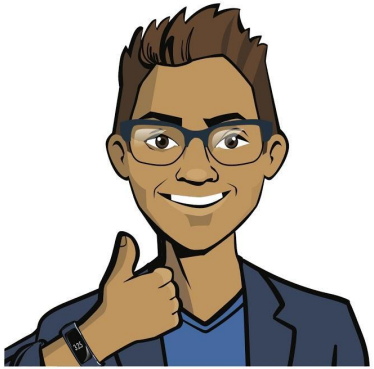
It doesn't work right!
Looks like the angle is wrong...

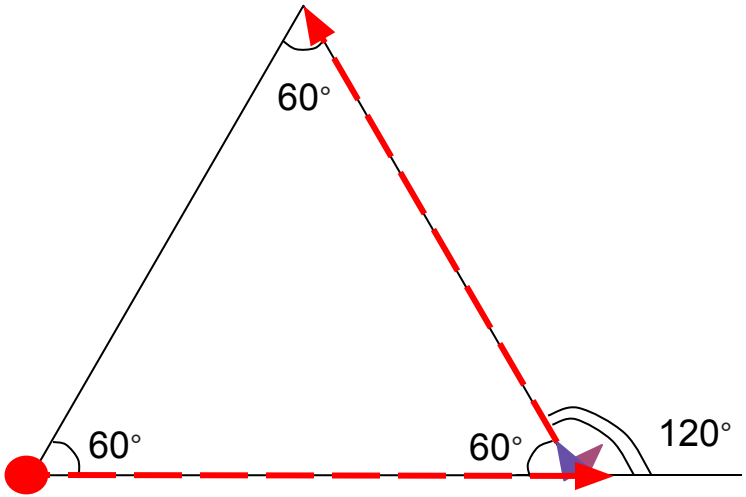
Brainstorming

Correct solution

Task. The customer wants tile with triangles. They have not yet decided on the color, so we need to program both red and blue. The triangle side length is 110 pixels.

*The turtle doesn't turn 60 degrees, it turns **120** degrees (it moves "outside" not "inside" the triangle)!*



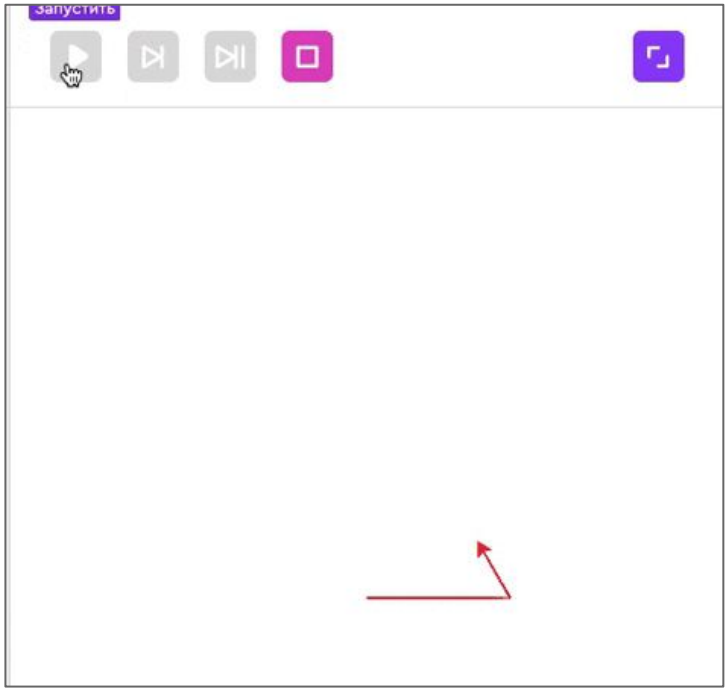


Brainstorming

Correct solution:

```
from turtle import *
color('red')
forward(110)
left(120)
forward(110)
left(120)
forward(110)
left(120)
color('blue')
forward(110)
left(120)
forward(110)
left(120)
forward(110)
exitonclick()
```

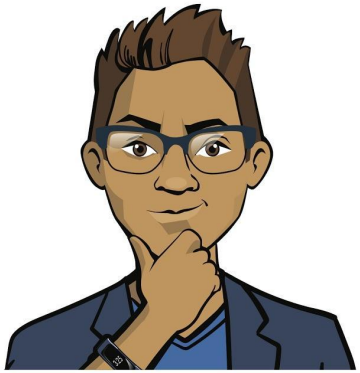
After the executor has drawn the first triangle, it must be turned all the way to its starting position.



Brainstorming

Before we continue:

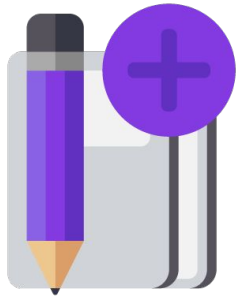
1. **How can we change the program** so that the turtle draws a larger triangle with a side of 150?
2. The customer wants to add one more color to compare – green. **How can we change the program** to draw triangles in three colors?



Brainstorming

Brainstorming:

Math for Developers



Drawing a bitmap image

Here are some more important commands:

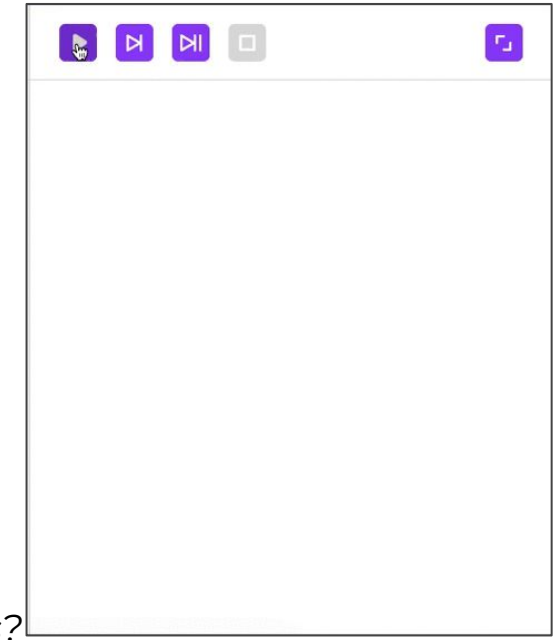
Command	*Purpose*
`begin_fill()`	Begin filling the shape (next comes the shape drawing command)
`end_fill()`	Complete filling the shape
`penup()`	Raise the executor's pen (useful when drawing multiple shapes)
`pendown()`	Lower the executor's pen
`goto(<coordinate X>, <coordinate Y>)`	Move the executor to the specified coordinates

Brainstorming

Let’s go over the task

Task. Program a three-circle pattern for wallpaper. Circle parameters:

- ❏ Top left: a small yellow circle with a radius of 30 pixels.
- ❏ Bottom center: a medium-sized green circle with a radius of 40 pixels.
- ❏ Top right: a big red circle with a radius of 50 pixels.



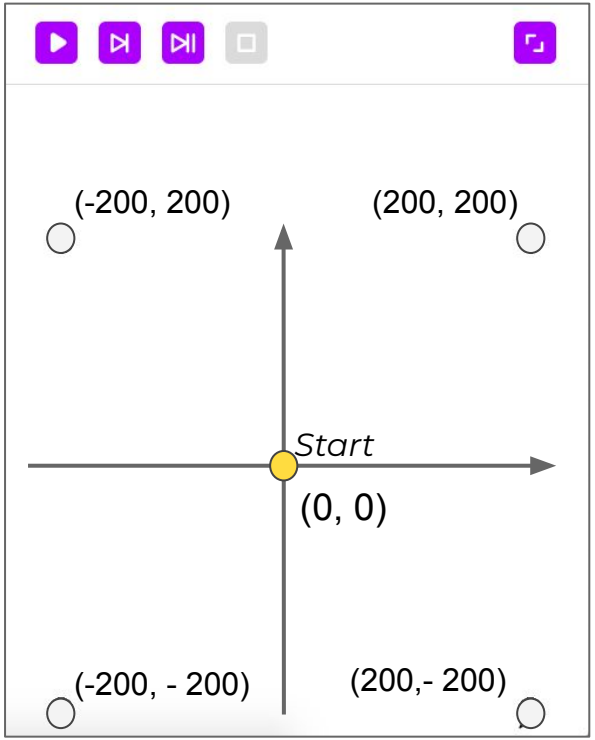
How do we solve this task?
How can we position the circles in different places?

Brainstorming

Coordinate plane

The turtle's position on the plane is determined by two numbers – its **coordinates**.

When the program starts, the turtle appears at the starting point (0, 0).

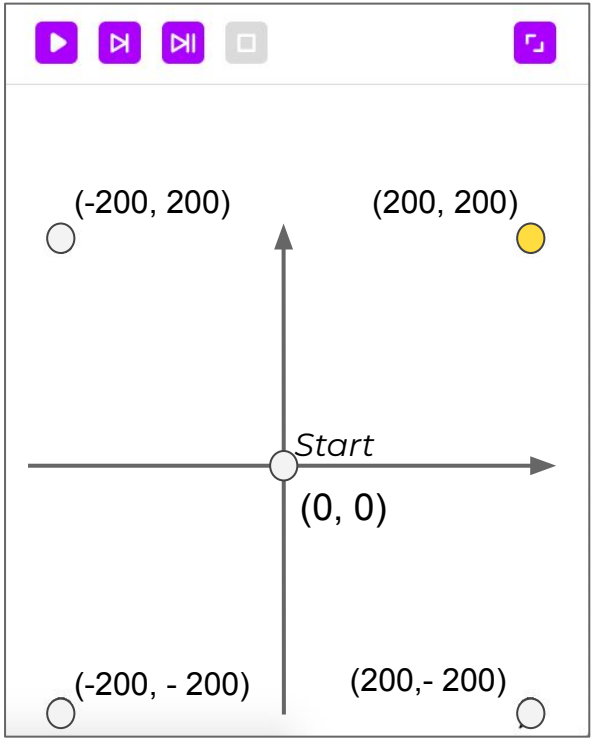


Brainstorming

Coordinate plane

The turtle's position on the plane is determined by two numbers – its **coordinates**.

When the program starts, the turtle appears at the starting point (0, 0).



➢ To move the executor, we need to set new coordinates for it.

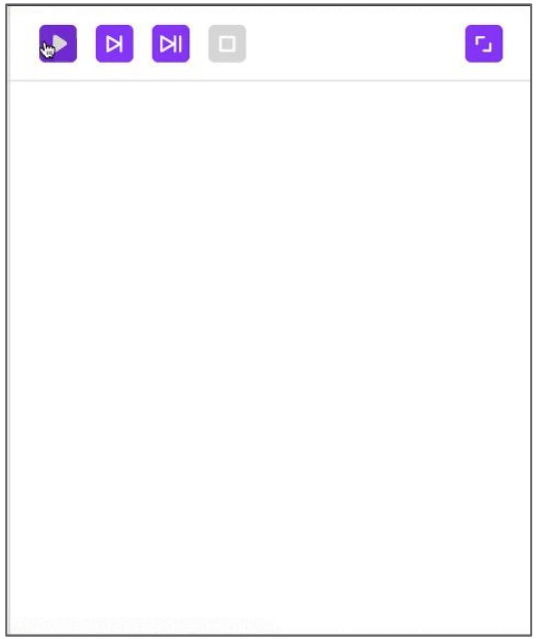
`goto(<coordinate X>, <coordinate Y>)`

Brainstorming

Let’s go back to the task

Task. Program a three-circle pattern for wallpaper. Circle parameters:

- ❏ Top left: a small yellow circle with a radius of 30 pixels.
- ❏ Bottom center: a medium-sized green circle with a radius of 40 pixels.
- ❏ Top right: a big red circle with a radius of 50 pixels.



Let’s try ***to draw the first circle.***
How can we do that?

Brainstorming

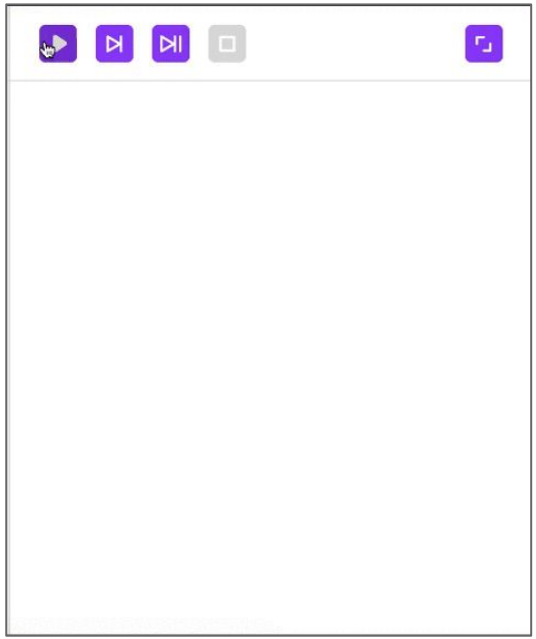
Let’s go back to the task

Task. Program a three-circle pattern for wallpaper. Circle parameters:

- ❏ Top left: a small yellow circle with a radius of 30 pixels.
- ❏ Bottom center: a medium-sized green circle with a radius of 40 pixels.
- ❏ Top right: a big red circle with a radius of 50 pixels.

```
from turtle import *
penup()
goto(-150, 130)
pendown()
color('yellow')
begin_fill()
circle(30)
end_fill()
exitonclick()
```

To avoid leaving traces when moving the turtle, you need to raise and lower the pen.



Brainstorming

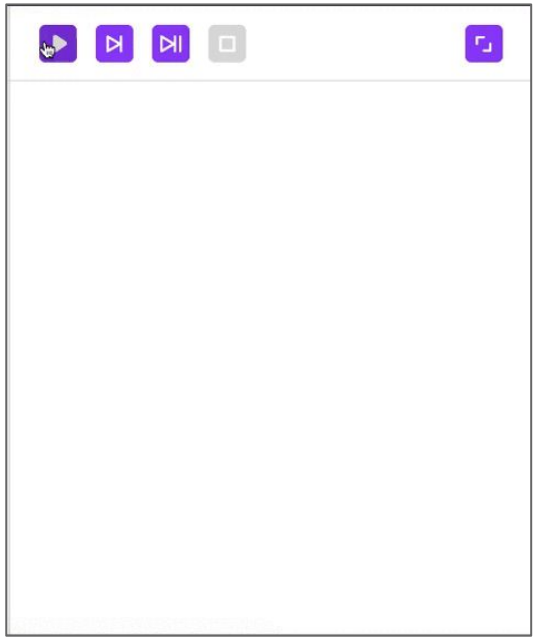
Let’s go back to the task

Task. Program a three-circle pattern for wallpaper. Circle parameters:

- ❏ Top left: a small yellow circle with a radius of 30 pixels.
- ❏ Bottom center: a medium-sized green circle with a radius of 40 pixels.
- ❏ Top right: a big red circle with a radius of 50 pixels.

```
from turtle import *
penup()
goto(-150, 130)
pendown()
color('yellow')
begin_fill()
circle(30)
end_fill()
exitonclick()
```

To fill the shape with color, we need to place the drawing of the shape between begin_fill() and end_fill().



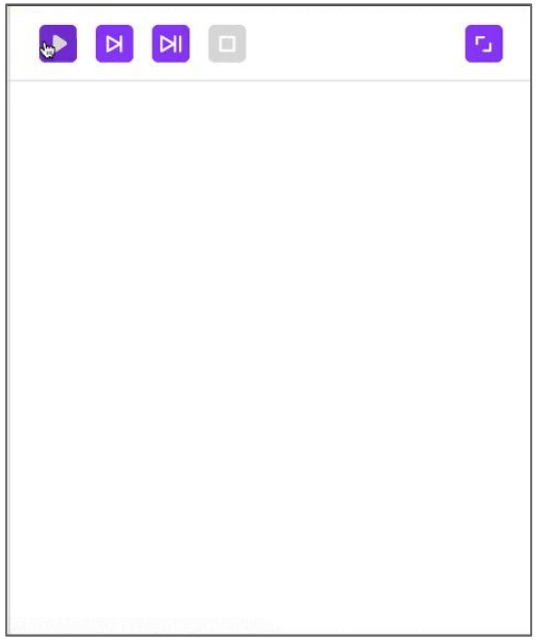
Brainstorming

Let’s go back to the task

Task. Program a three-circle pattern for wallpaper. Circle parameters:

- ❏ Top left: a small yellow circle with a radius of 30 pixels.
- ❏ Bottom center: a medium-sized green circle with a radius of 40 pixels.
- ❏ Top right: a big red circle with a radius of 50 pixels.

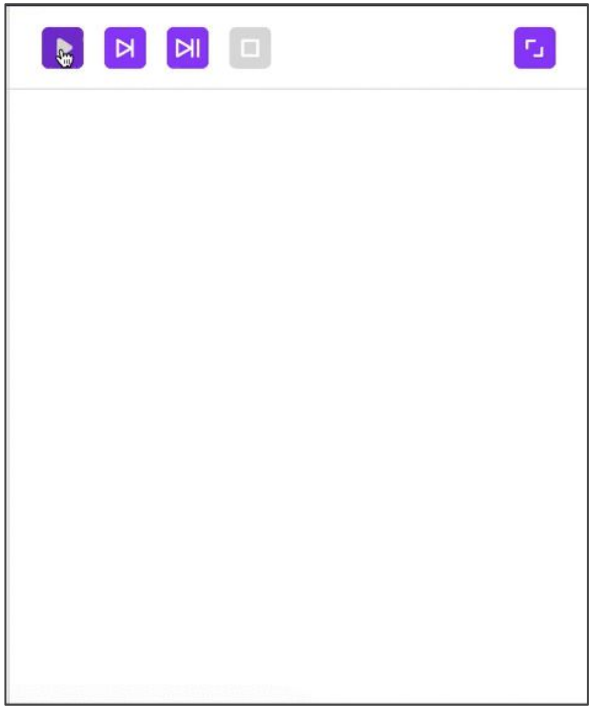
```
from turtle import *
penup()
goto(-150, 130)
pendown()
color('yellow')
begin_fill()
circle(30)
end_fill()
exitonclick()
```



*How do we add **the second and third circles**?*

Brainstorming

```
from turtle import *
#yellow circle
penup()
goto(-150, 130)
pendown()
color('yellow')
begin_fill()
circle(30)
end_fill()
#green circle
penup()
goto(0, -100)
pendown()
color('green')
begin_fill()
circle(40)
end_fill()
#red circle
penup()
goto(130, 110)
pendown()
color('red')
begin_fill()
circle(50)
end_fill()
exitonclick()
```



Brainstorming

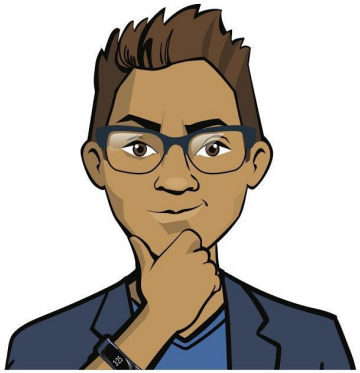
Before we continue:

1. The coordinates of the points along which the turtle moves are given:

   (100, 120), **(0, -100)**, **(-150, 0), (100, 120)**.

   What shape will be displayed on the coordinate plane as a result of such movement (if the pen is not raised)?

1. How can we change the code of the previous program to make a similar pattern but with squares instead of circles?
2. Can we optimize our previous program using functions? How?



Brainstorming