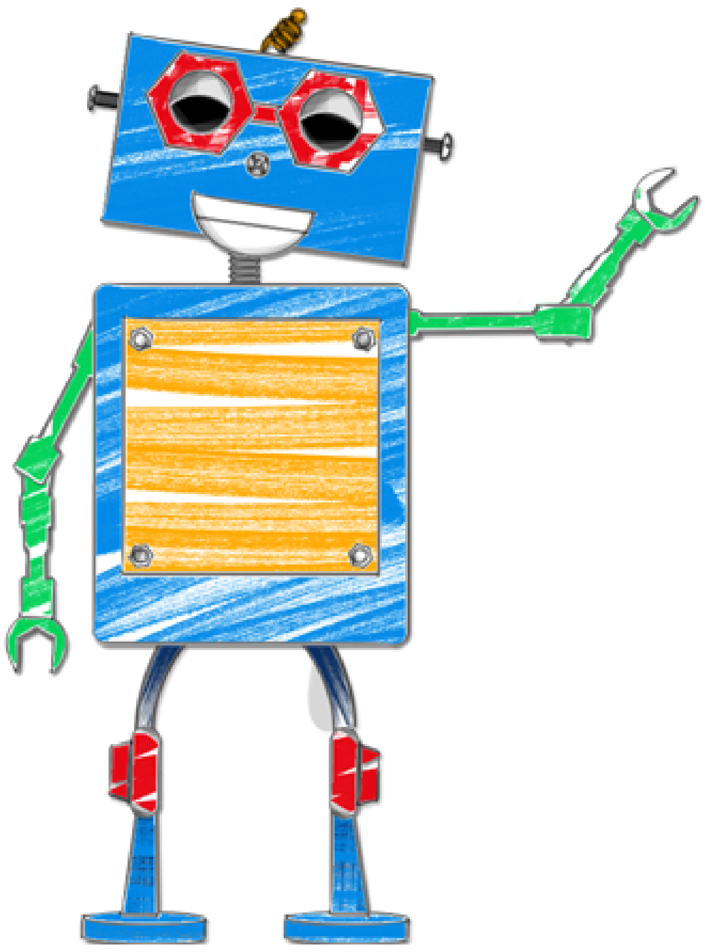
Code With Bolt

- Teacher's handbook -

version 1.1



Hello, my name is Bolt.

I'm a robot that can teach everyone how to **write code** that draws patterns, plays games, and does lots of other stuff! My playground is suitable for ages 6 through 13... to infinity... and beyond!

Everything you need to know is broken down into individual activities. All are detailed in full with **step-by-step instructions** for you, and your class.

Contents

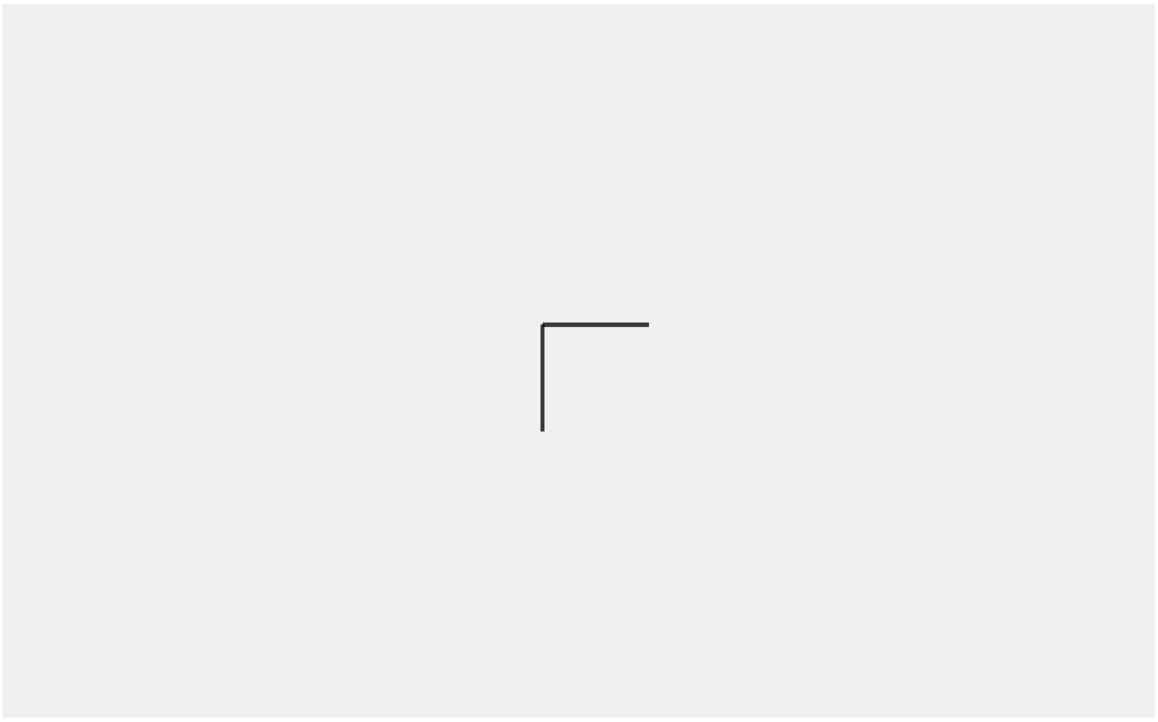
Basic shapes

Category: Drawing | *Level 1* | 6 steps

We will begin by drawing simple lines, and then work up to several different shapes

Step 1

Task: Draw two more lines, to turn the shape into a square.



Code

```
COMMENT At the start of a line tells Bolt to ignore it
COMMENT Real coders use COMMENTs to remind us how
COMMENT the program works.


COMMENT Always start a program with a 'reset' statement
COMMENT All drawing operations happen from the current
COMMENT position of the 'cursor'. Using 'reset' ensures the
COMMENT cursor is in the centre of the screen.

reset


COMMENT We're drawing everything at 100% scale
COMMENT So, if you say 'forward 50' it will draw
COMMENT 50 pixels. Setting scale to 200%, would draw 100.

scale 100
```

```
COMMENT Let's start where X=253, and Y=200

position 253 200


COMMENT Move the cursor forward by 50 units which, because
COMMENT of the scale, means 50 pixels.

forward 50


COMMENT Turn the cursor by 90 degrees. It should now point right.

turn right 90


COMMENT Move forward again. Which, because the cursor is pointing right
COMMENT means it moves to the right!

forward 50
```

Hints

1. Remember to tell Bolt to turn right by 90 degrees before you start, otherwise the line will continue in its previous direction.
2. You can turn the grid on to help you judge positions and distances
3. A square might have four lines, and four corners, but you only need to turn Bolt three times before he can return to his original position

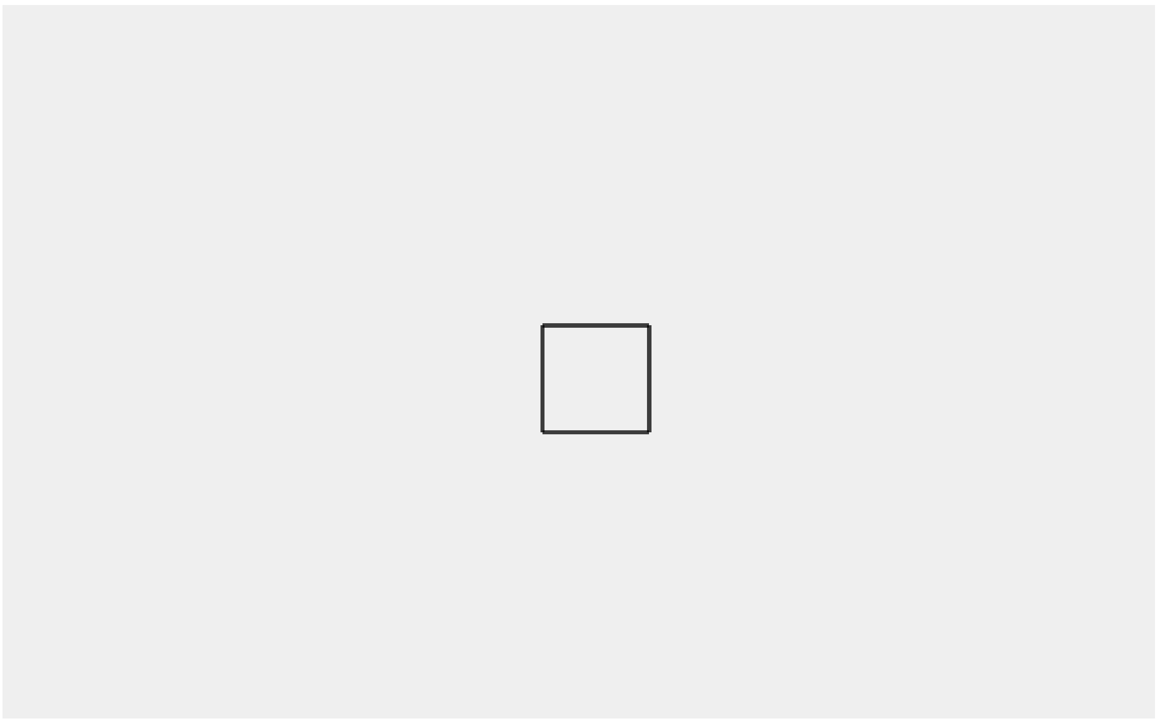
Glossary

This example introduces the new concepts of:

- reset
- scale
- position
- forward
- turn

Step 2

Task: Make each side of the square twice as big as it currently is.



Code

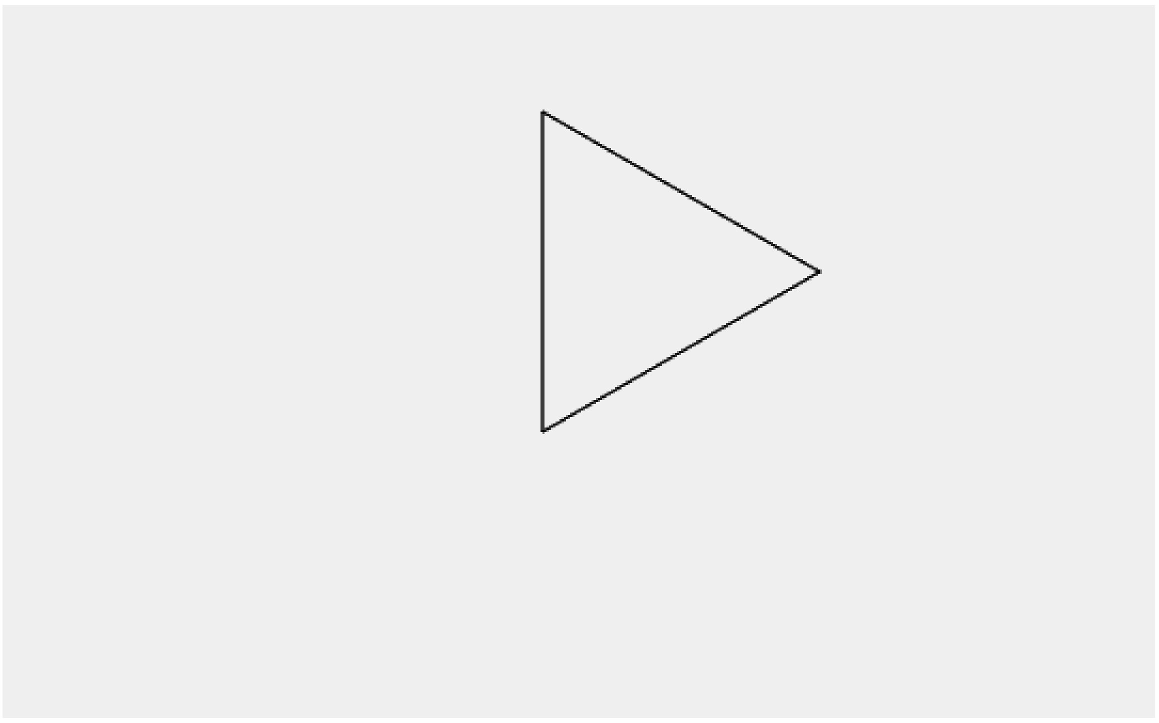
```
COMMENT We start as usual, by setting the scene
reset
scale 100
position 253 200

forward 50
turn right 90

COMMENT This 'forward' instruction is the last
COMMENT one from the first step.

forward 50


COMMENT This is the next instruction - we
COMMENT turn 90 degrees to the right, so
COMMENT we're now pointing down.

turn right 90

forward 50
turn right 90

forward 50
turn right 90
```

Hints

1. The first way of doing this is to change the number of steps forward that Bolt takes.
2. The second way of doing this is to change the 'scale' to 200, which doubles every 'forward' command, but not the 'right' commands.

Step 3

Task: Change the square into a triangle.



Code

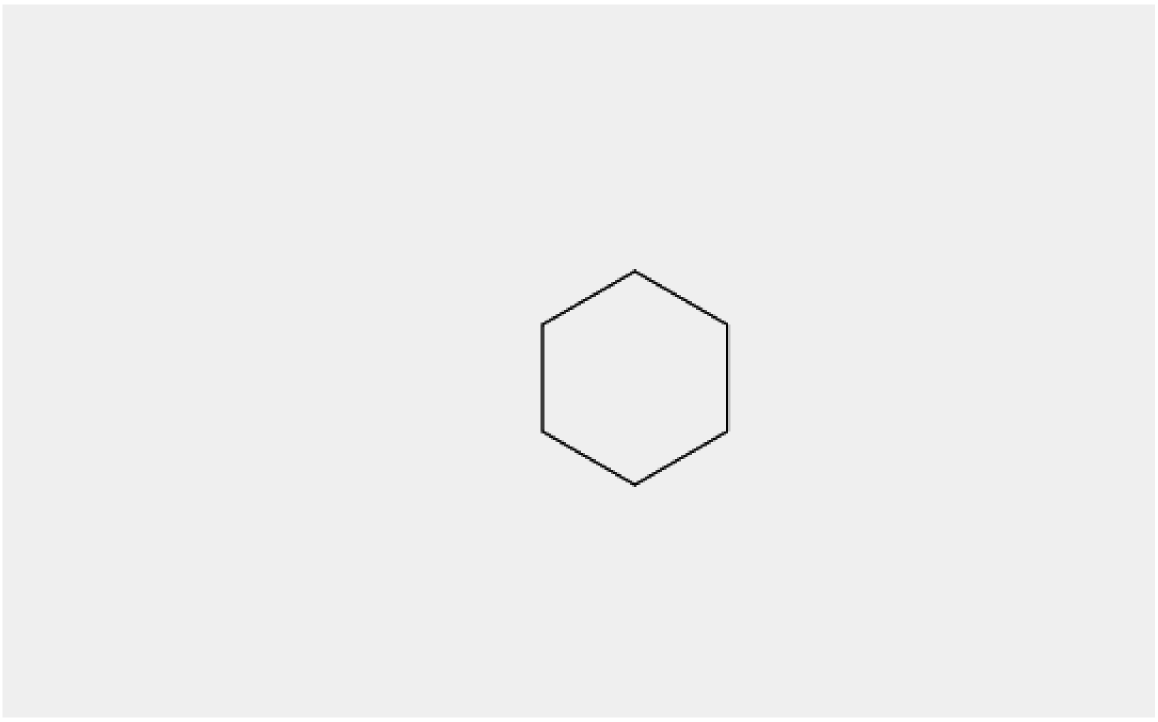
```
reset
scale 100
position 253 200

COMMENT We move forward 100 pixels, instead of 50

forward 100
turn right 90

forward 100
turn right 90

forward 100
turn right 90

forward 100
turn right 90
```

Hints

1. Remember you only need three sides for a triangle.
2. You need to turn by 120 degrees each time, because 3 * 120 degrees = 360 degrees, which is the total of the external angles in a triangle.

Step 4

Task: Draw a hexagon.



Code

```
reset
scale 100
position 253 200

forward 150
turn right 120

forward 150
turn right 120

forward 150
turn right 120
```

Hints

1. A hexagon has 6 sides.
2. Bolt needs to turn right by 60 degrees after moving forward.

More ideas #1

Task: Can you draw a 7-sided shape? If you try dividing 360 by 7 it's not a whole number, so what will you do?

Code

```
reset
scale 100
position 253 200

forward 50
turn right 60

forward 50
turn right 60

forward 50
turn right 60

forward 50
turn right 60

forward 50
turn right 60

forward 50
turn right 60
```

More ideas #2

Task: What shapes can you create by varying the number of steps that Bolt walks forward, each time?

Shapes & loops

Category: **Drawing** | *Level 2* | 9 steps

We will now learn about using loops to perform 1 action, many times. It is easier for the computer to do a task 4 times, than it is for us to write the same code 4 times!

Step 1

Task: Draw two more lines, to turn the shape into a square.

Code

```
COMMENT At the start of a line tells Bolt to ignore it
COMMENT Real coders use COMMENTs to remind us how
COMMENT the program works.

reset
scale 100
position 253 200

COMMENT Instead of writing the same piece of code
COMMENT several times, we can wrap it inbetween
COMMENT two commands, 'repeat', and 'end repeat'
COMMENT to do an equivalent job.

repeat 2

 forward 50
 turn right 90

end repeat
```

Hints

1. Look at the number after the repeat.

Glossary

This example introduces the new concepts of:

- repeat
- end

Step 2

Task: Now recreate a hexagon, using a loop.

Code

```
reset
scale 100
position 253 200

COMMENT There are 4 sides to a square so
COMMENT we repeat the loop 4 times.

repeat 4

 forward 50
 turn right 90

end repeat
```

Hints

1. The sides can be any length, by the angle will need to be 60.

Step 3

Task: Using a variable called 'sides' to control the loop, and draw the shape.

Code

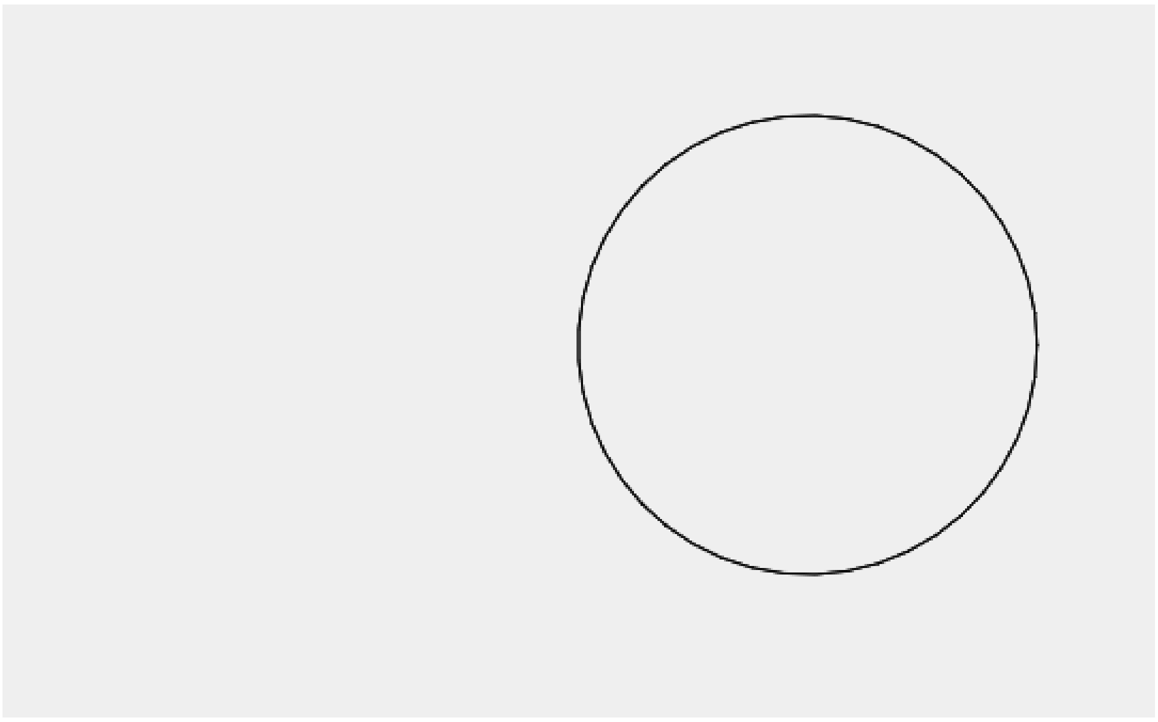
```
reset
scale 100

repeat 6

 forward 50
 turn right 60

end repeat
```

Hints

1. Again, look for the repeat instruction.

Step 4

Task: What's the smallest number of sides you need to make the shape look like a circle?



Code

```
reset
scale 100

set sides to 6

repeat sides

 forward 40

 COMMENT You don't need to use only numbers.
 COMMENT You can create 'expressions' (which
 COMMENT are basically formulas) to calculate
 COMMENT a value, instead.

 turn right 360 / sides

end repeat
```

Hints

1. Try decreasing the variable in steps of 20.
2. Then, when you can see straight lines, incresease in steps of 10.
3. If the circle gets too small to size, try increase the forward distance.

Glossary

This example introduces the new concepts of:

- set
- expression

More ideas #1

Task: Can you draw put the length of each line into a variable?

Code

```
reset
scale 100

COMMENT Instead of using numbers, we can store
COMMENT the value into a 'variable'. This is better
COMMENT because we can change the value of the variable
COMMENT in one place, and it will affect the whole code


COMMENT We set the value of the variable like this:

set sides to 45


COMMENT We can also use expressions in place of the value

set angle to 360 / sides


COMMENT In fact, anywhere a number can be used, a
COMMENT variable can be used in its place. Like 'repeat'

repeat sides

 forward 15
 turn right angle

end repeat
```

More ideas #2

Task: What happens if you use 'set' to change the length before the 'end repeat' instruction?

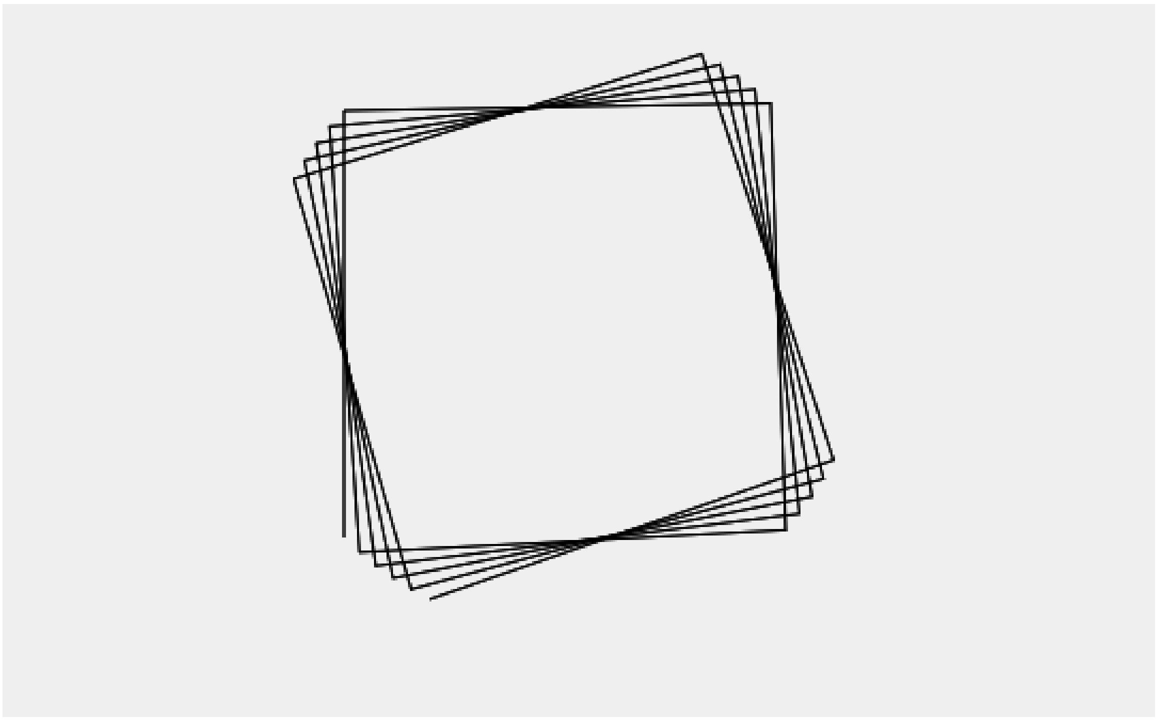
Code

```
reset
```

```
scale 100

COMMENT We set up all our variables here:

set sides to 45
set angle to 360 / sides
set length to 15

COMMENT There are no numbers in our code from
COMMENT from here, until the end of the program!

repeat sides

 forward length
 turn right angle

end repeat
```

More ideas #3

Task: What happens if you use 'set' to increase the angle before the 'end repeat' instruction?



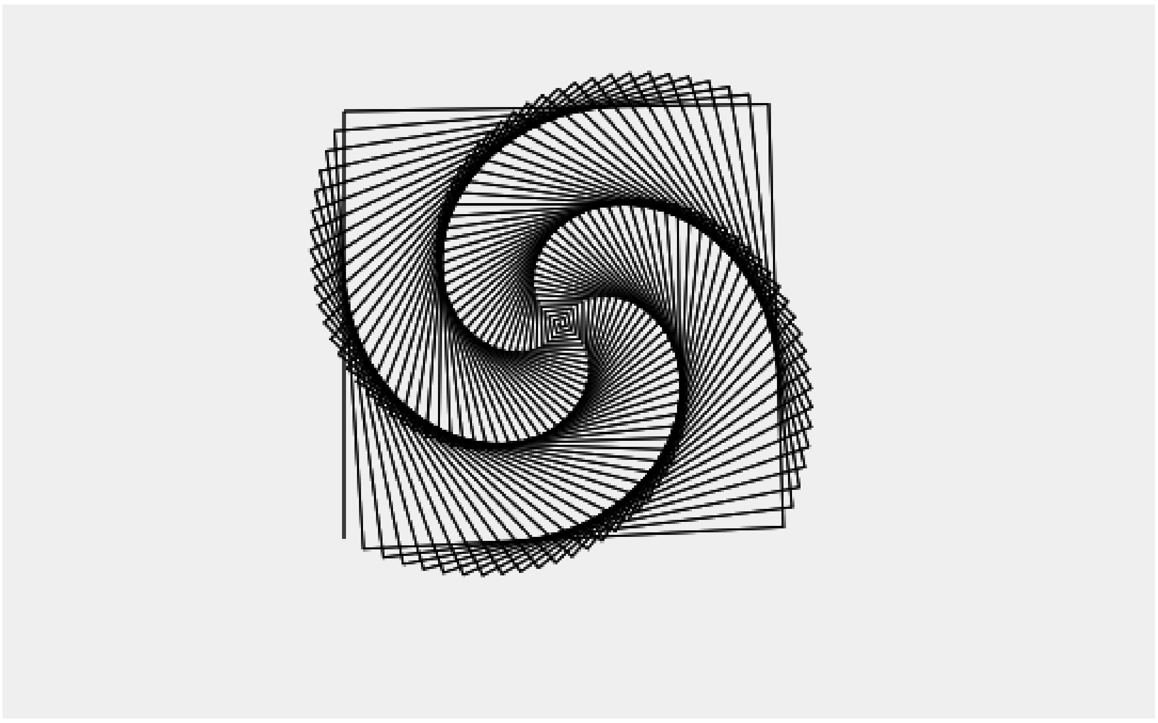
Code

```
reset
scale 100

COMMENT We set up all our variables here:

set sides to 45
set angle to 360 / sides
set length to 15
```

```
COMMENT There are no numbers in our code from
COMMENT from here, until the end of the program!

repeat sides

 forward length
 turn right angle

end repeat
```

More ideas #4

Task: What happens if the angle is wrong? Say 89 degrees.



Code

```
reset
scale 100
position 160 250

COMMENT We set up all our variables here:

set angle to 89
set length to 200
set sides to 20


COMMENT And we draw here:

repeat sides

 forward length
 turn right angle
```

```
end repeat
```

More ideas #5

Task: And what happens if you repeat the lines 20 times (or more)? And then change one of the variables in the loop?

Code

```
COMMENT This example is an example of how to
COMMENT create your own Spirograph images

reset
scale 100
position 160 250

COMMENT We set up all our variables here:

set angle to 89
set length to 200
set sides to 200


COMMENT There are no numbers in our code from
COMMENT from here, until the end of the program!

repeat sides

 forward length
 turn right angle

 set length to length-1

end repeat
```

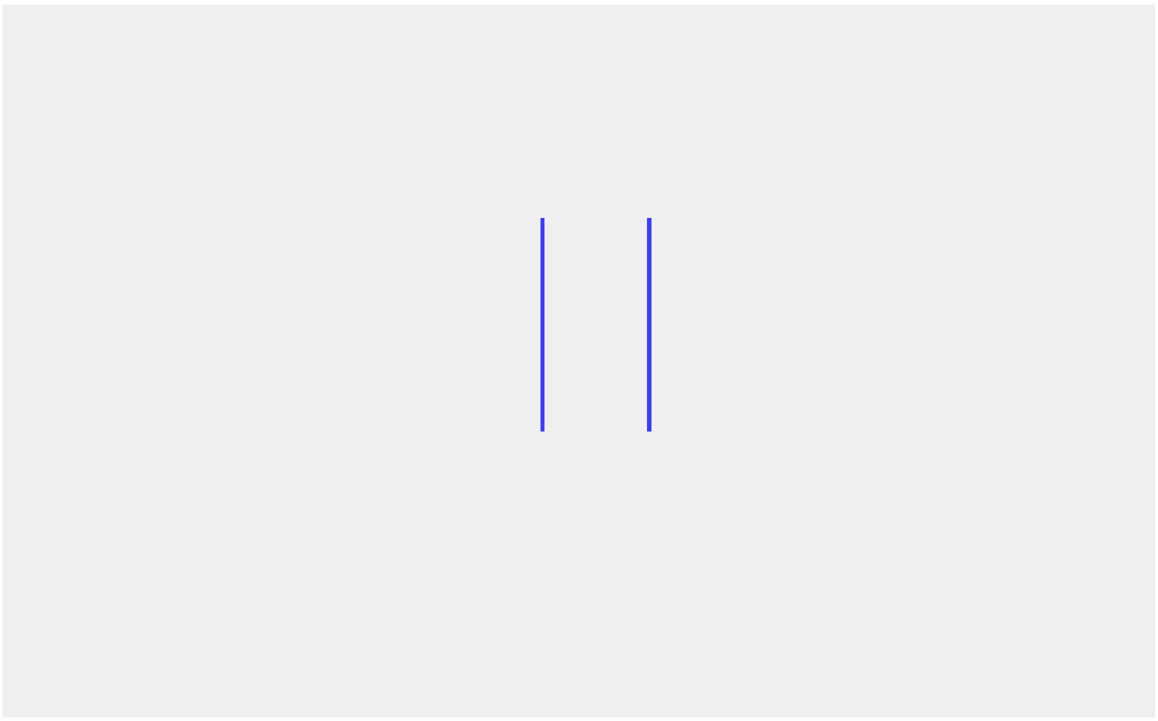
Using pen up

Category: **Drawing** | ***Level 3*** | 5 steps

Not all drawings have one continuous line. Let's see how!

Step 1

Task: Draw one more line, parallel to the first two.



Code

```
COMMENT At the start of a line tells Bolt to ignore it
COMMENT Real coders use COMMENTs to remind us how
COMMENT the program works.

reset
scale 100
position 253 200

COMMENT Let's draw this in blue

colour drawing "blue"

COMMENT Words, like "blue", needs quotes so
COMMENT Bolt can tell the difference between it
COMMENT and a variable called blue


COMMENT We start by facing upwards...

forward 100
```

```
pen up
turn right 90
forward 50
turn right 90

COMMENT We should now be facing downwards


COMMENT We can now start drawing again
pen down

forward 100
```

Hints

1. Remove the 'pen down' and 'pen up' instructions to see the line
2. When you're facing down, your right hand side is to the left of the screen

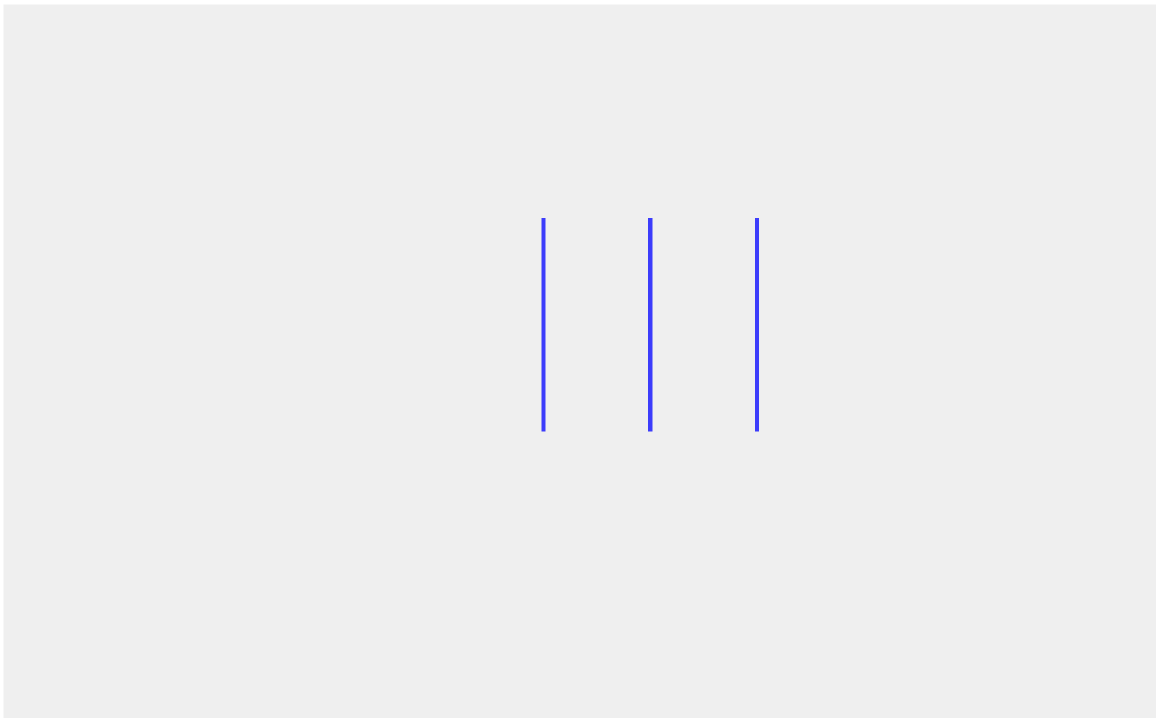
Glossary

This example introduces the new concepts of:

- pen
- colour
- left

Step 2

Task: Make the second pair of parallel lines into a rectangle.



Code

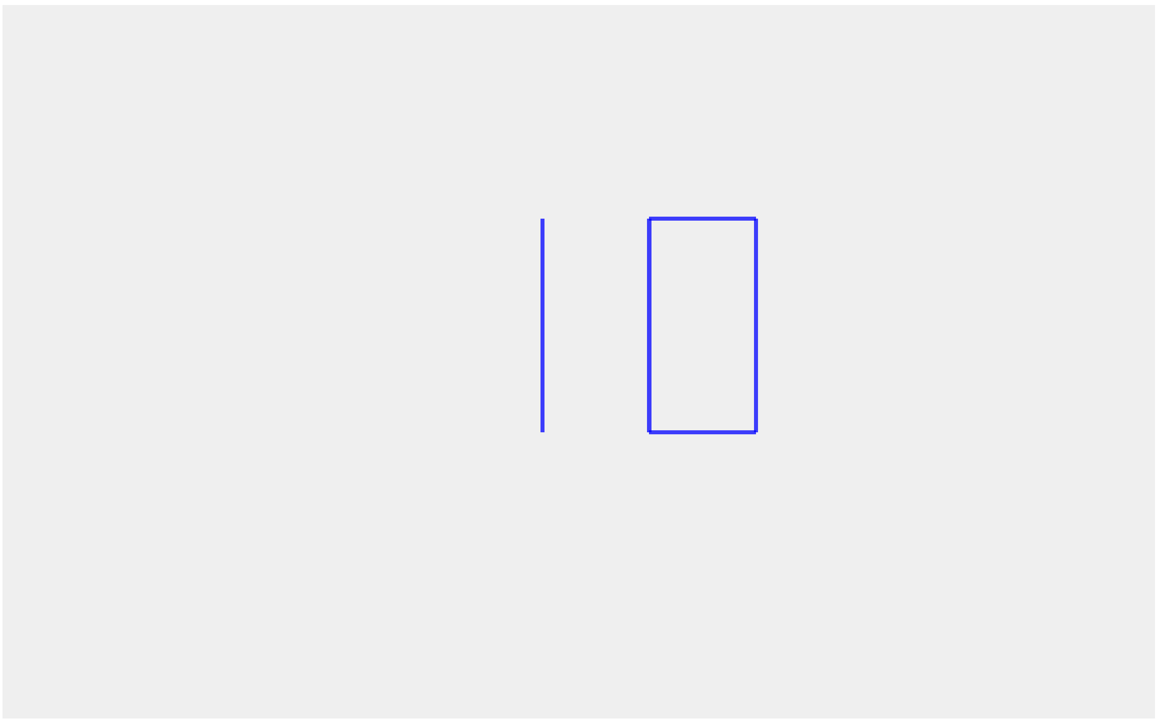
```
reset
scale 100
position 253 200

COMMENT Let's draw this in blue

colour drawing "blue"


COMMENT We start by facing upwards...

forward 100


pen up
turn right 90
forward 50
turn right 90
COMMENT We should now be facing downwards


COMMENT We can now start drawing again

pen down

forward 100

COMMENT This is ALMOST copy of the previous code
COMMENT but we switch right for left

pen up
turn left 90
forward 50
turn left 90
pen down

forward 100
```

Hints

1. Is the pen up, or down, when you're drawing?

More ideas #1

Task: Can you place the sizes into variables?



Code

```
reset
scale 100
position 253 200

COMMENT Let's draw this in blue


colour drawing "blue"

COMMENT We start by facing upwards...

forward 100


pen up
turn right 90
forward 50
turn right 90
COMMENT We should now be facing downwards


COMMENT We can now start drawing again
pen down

forward 100


COMMENT We're drawing a square using the second set of lines
turn left 90
forward 50
turn left 90

forward 100

turn left 90
forward 50
```

More ideas #2

Task: How about having a 10 pixel gap between the two shapes?

Code

```
reset
scale 100
position 253 200

colour drawing "blue"

COMMENT Let's have a rectangle of width 50 and height 100

set width to 50
set height to 100


COMMENT We start by facing upwards...

forward height

pen up
turn right 90
forward width
turn right 90
COMMENT We should now be facing downwards

COMMENT We can now start drawing again
pen down

forward height


COMMENT We're drawing a square using the second set of lines

turn left 90
forward width
turn left 90

forward height

turn left 90
forward width
```

More ideas #3

Task: Draw each part of the line with a different colour?

Code

```
reset
scale 100
position 253 200

colour drawing "blue"

COMMENT Let's have a rectangle of width 50 and height 100

set width to 50
set height to 100


COMMENT And a gap of 10

set gap to 10


COMMENT We start by facing upwards...

forward height

pen up
turn right 90
forward gap
turn right 90
COMMENT We should now be facing downwards


COMMENT We can now start drawing again
pen down

colour drawing "red"
forward height


COMMENT We're drawing a square using the second set of lines

turn left 90
forward width
turn left 90

forward height

turn left 90
forward width
```

Using methods

Category: **Drawing** | *Level 4* | 4 steps

Let's see how we can group a block of code into a 'method', so it can be re-used!

Step 1

Task: Can you change the method called 'shape' so it uses a 'repeat' loop?

Code

```
COMMENT At the start of a line tells Bolt to ignore it
COMMENT Real coders use COMMENTs to remind us how
COMMENT the program works.

reset
scale 100
position 253 200


COMMENT Here, we are defining a 'method' called shape
COMMENT Bolt doesn't do anything with the method, it
COMMENT just remembers that it exists.

define shape

 forward 100
 turn right 90

 forward 100
 turn right 90

end

COMMENT We can now 'call' the method we defined earlier

call shape
```

Hints

1. You need to start with 'repeat' followed by the number of times it should be repeated
2. You need to end with an 'end repeat' instruction.

Glossary

This example introduces the new concepts of:

- define
- method
- call

Step 2

Task: Set up variables, called 'sides', 'length', and 'angle', and use them to control the shape.

Code

```
reset
scale 100

define shape
 COMMENT Any code that works outside of a method
 COMMENT definition, can work inside it, too!

 repeat 4
  forward 100
  turn right 90
 end repeat

end

call shape
```

Hints

1. Angle can be calculated using the number of sides, as we saw in example 2.
2. Variables can be 'set' from anywhere, but it is best to do it just before the 'call'

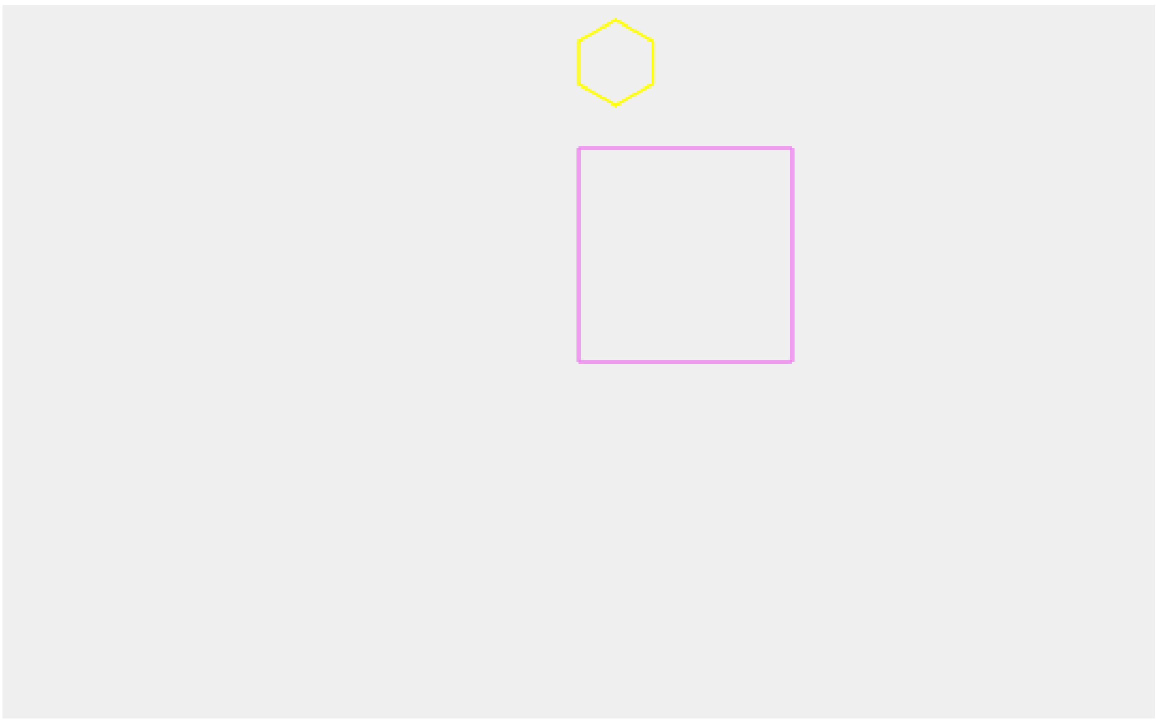
Glossary

This example introduces the new concepts of:

- repeat
- end

Step 3

Task: See if you can draw two different shapes on the same screen, without any lines connecting them!



Code

```
reset
scale 100

define shape
 COMMENT Although the variable 'sides' hasn't
 COMMENT appeared in the code yet, when the program
 COMMENT is run it will have been created (line 17).

 set angle to 360 / sides

 repeat sides
  forward length
  turn right angle
 end repeat
end

COMMENT We can set some variables here...

set sides to 4
set length to 100


COMMENT ...and then call our shape method.
COMMENT The variables can be used inside 'shape'
COMMENT because they are 'global', and exist throughout
COMMENT the whole program.

call shape
```

Hints

1. Remember to use 'pen up' and 'pen down'

2. Where is the position of the cursor, after the first shape?

3. In which direction are we facing, after the first shape?

More ideas #1

Task: How many other shapes can you draw?

Code

```
reset
scale 100

COMMENT Define a method, as usual.
COMMENT Before calling this method, we must
COMMENT 'set' two variables, one called 'sides'
COMMENT and one called 'length'.

define shape
 set angle to 360 / sides
 repeat sides
  forward length
  turn right angle
 end repeat
end


COMMENT Draw the first shape

set sides to 4
set length to 100
colour drawing "violet"

call shape

COMMENT We're now facing to the up, so moving foward puts the
COMMENT next shape above the first.
COMMENT Also, we're at the base of the square, so we need
COMMENT to move forward further than the size of the square.

pen up
forward 130
pen down

set sides to 6
set length to 20
colour drawing "yellow"

call shape
```

Shapes & positions

Category: **Drawing** | *Level 5* | 5 steps

Remembering where your cursor is, and in what direction it's facing is tricky. Let's solve that problem!

Step 1

Task: Instead of starting the square at 100 300, move it's X co-ordinate to 200.

Code

```
reset
scale 100

define shape
 set angle to 360 / sides
 repeat sides
  forward length
  turn right angle
 end repeat
end

COMMENT Start drawing at a point where x is 100, and y
COMMENT is 300. We always give co-ordinates in this order

position 100 300

set sides to 4
set length to 100
call shape
```

Hints

1. Co-ordinates are always given in the same order : x and then y

Step 2

Task: Instead of starting with the cursor pointing straight up (an angle of 90), what happens if you set it to 180?

Code

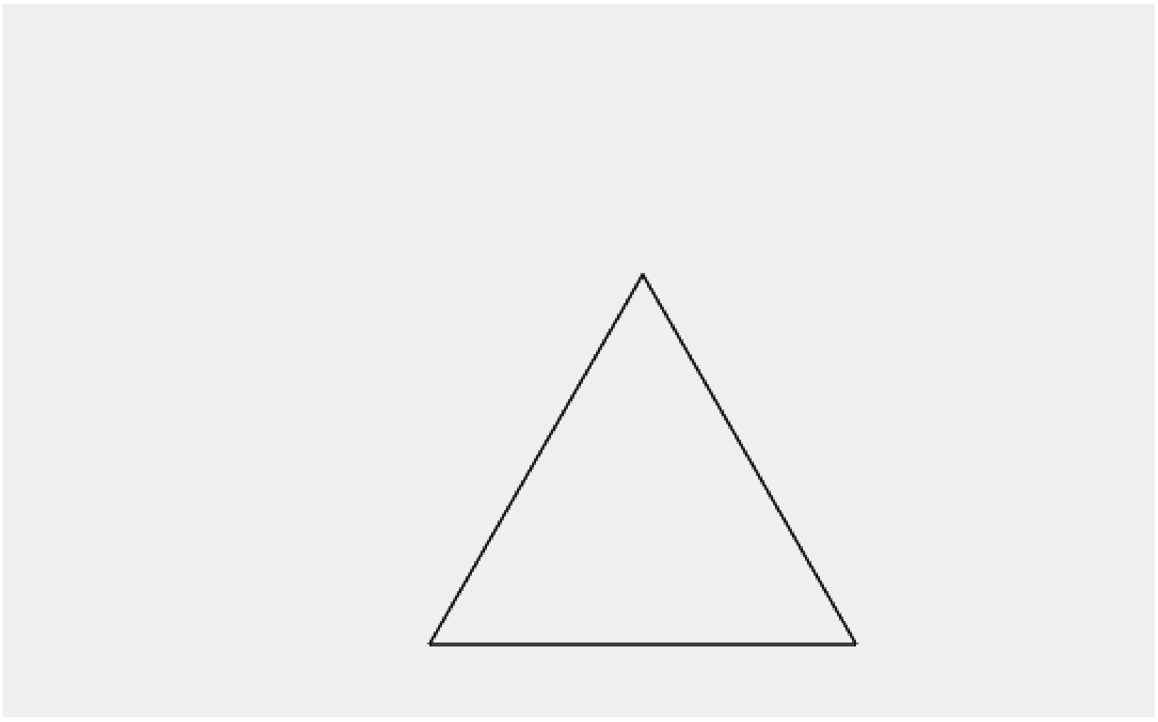
```
reset
scale 100

define shape
 set angle to 360 / sides
 repeat sides
  forward length
  turn right angle
 end repeat
end

position 200 300

set sides to 4
set length to 100
call shape
```

Hints

1. Look for the new 'angle 90' line
2. Make the shape into a triangle, as that might be easier to see

Step 3

Task: Try drawing a large triangle with a flat base.



Code

```
reset
scale 100
```

```
define shape
 set angle to 360 / sides
 repeat sides
  forward length
  turn right angle
 end repeat
end

COMMENT We can set the direction that the Bolt cursor
COMMENT is facing. That command is called 'angle'.

angle 180
position 200 300

set sides to 4
set length to 100
call shape
```

Hints

1. If the triangle goes off the screen, you'll need to position it somwhere else
2. The angle will be either 0, 90, 180, or 270

More ideas #1

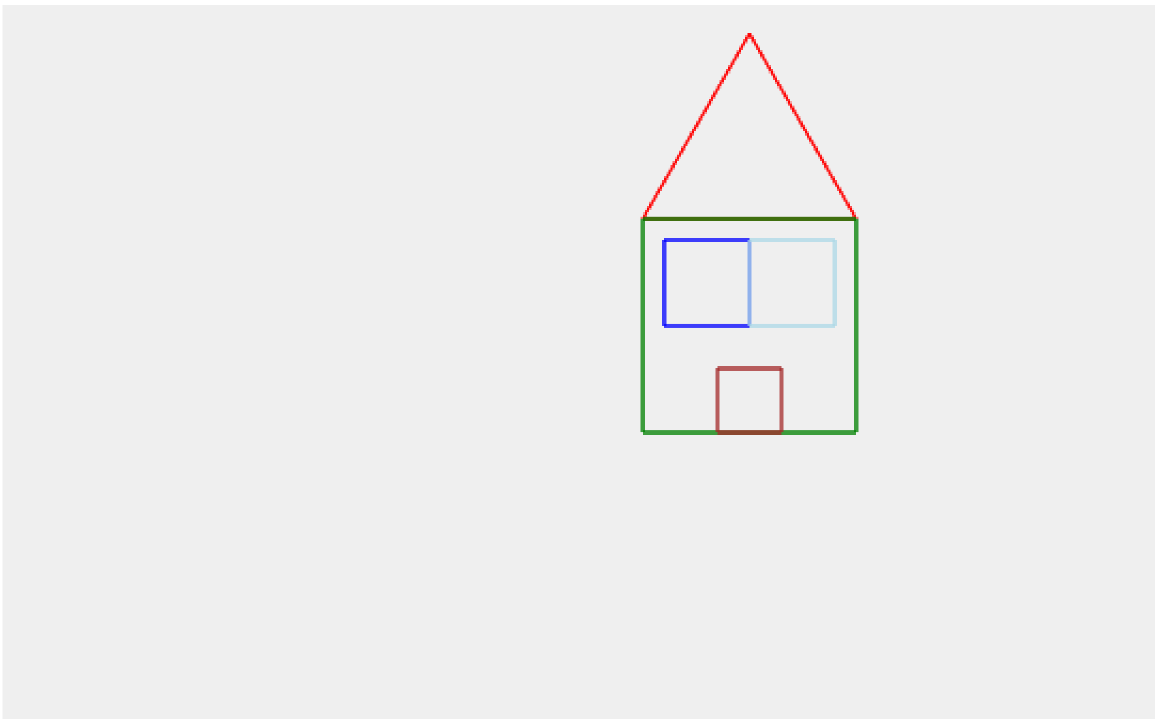
Task: Can you draw a simple house?

Code

```
reset
scale 100

define shape
 set angle to 360 / sides
 repeat sides
  forward length
  turn right angle
 end repeat
end

COMMENT Setting the angle with 'turn to' using a
COMMENT location with 'position' instruction is
COMMENT called 'absolute positioning'.
COMMENT When moving the cursor with 'forward' or
COMMENT 'right', it's called 'relative positioning'

turn to 180
position 400 300

set sides to 3
set length to 200
call shape
```

More ideas #2

Task: What else can you draw?



Code

```
reset
scale 100

define shape
 set angle to 360 / sides
 repeat sides
  forward length
  turn right angle
 end repeat
end

COMMENT Draw the roof first

angle 180
position 400 100


set sides to 3
set length to 100
colour drawing "red"
call shape


COMMENT Now the main house
```

```
position 400 200
set sides to 4
set length to 100
colour drawing "green"
call shape


COMMENT A door

position 365 200
set length to 30
colour drawing "brown"
call shape


COMMENT And finally some windows

position 350 150
set length to 40
colour drawing "blue"
call shape

position 390 150
colour drawing "lightblue"
call shape
```

A painting app

Category: Interactions | *Level 1* | 4 steps

We are going to write a simple paint application.

Step 1

Task: Instead of drawing a dot when the mouse is pressed, draw a dot when is down

Code

```
COMMENT At the start of a line tells Bolt to ignore it
COMMENT Real coders use COMMENTs to remind us
COMMENT how the program works.

reset

COMMENT Let's draw all images in a nice color

colour image "mediumblue"


COMMENT The 'update' method is magical! It is called
COMMENT automatically, by Bolt, 30 times every second.
COMMENT You use this method for all interactive apps

define update

 COMMENT The 'if' command is a conditional. It gives
 COMMENT the program a choice about whether to do
 COMMENT something, or not.

 if mousepressed

  COMMENT Instead of drawing lines, we can draw
  COMMENT whole graphics. This line draws an image
  COMMENT that we've called 'smalldot'

  draw image smalldot at mousex mousey

 end if

end
```

Hints

1. The glossary will tell you about mousedown.

Teacher Notes

If the mouse moves fast then there will be gap between the dots. This is because the update loop is only run 30 times each second, and the mouse position is therefore only checked that often, even though it is moving many 100's of times each second.

Glossary

This example introduces the new concepts of:

- if
- mousepressed
- mousex
- mousey
- draw

Step 2

Task: Pretend there's a mirror in the middle of the screen. Reflect anything on the left hand side to appear on the right.

Code

```
reset

COMMENT There is a list of colours in the glossary

colour image "mediumblue"


define update

 COMMENT When the 'if' expression is 'true' Bolt executes
 COMMENT every line of code it sees until it gets to 'end if'

 if mousedown
  draw image smalldot at mousex mousey
 end if

end
```

Hints

1. The screen is 540 pixels wide. So any dot that is 10 pixels from the left edge, must be 10 pixels from the right edge.

2. The reflection is 540-mousex

Glossary

This example introduces the new concepts of:

- mousedown

Step 3

Task: Do not draw anything if the mouse is in the right hand side of the screen.

Code

```
reset

colour image "mediumblue"

define update

 if mousedown
  draw image smalldot at mousex mousey

  COMMENT Again, we can perform expressions in
  COMMENT any place a number, or variable, can
  COMMENT be used.

  draw image smalldot at 540-mousex mousey
 end if

end
```

Hints

1. The middle of the screen is 270, so only respond 'if' the mousex is less than 270

More ideas #1

Task: How about reflecting on the Y axis instead? Or as well?

Code

```
reset

colour image "mediumblue"

define update

 COMMENT We can have more than one 'condition'
 COMMENT in an 'if' statement. In this example,
 COMMENT both parts must be 'true' for Bolt to
 COMMENT execute the code inbetween

 if mousedown and mousex < 270

  draw image smalldot at mousex mousey
  draw image smalldot at 540-mousex mousey

 end if

end
```

Glossary

This example introduces the new concepts of:

- and

A better painting app

Category: Interactions | *Level 2* | 4 steps

We are going to improve our painting application.

Step 1

Task: Instead of drawing a dot at the exact position of the mouse, offset it by a random amount.

Code

```
COMMENT At the start of a line tells Bolt to ignore it
COMMENT Real coders use COMMENTs to remind us
COMMENT how the program works.

reset

define update

 if mousedown
  set x to 0
  set y to 0
  draw image smalldot at mousex+x mousey+y
 end if

end
```

Hints

1. The glossary will tell you about random.

Step 2

Task: Instead of just one dot, draw several. Maybe 5.

Code

```
reset

define update

 if mousedown
```

```
   COMMENT Random is a system function that gives you
   COMMENT an unpredictable number. See Glossary!
   COMMENT A system function can be used anywhere a
   COMMENT number, or a variable, can be used...

   set x to random(1,20)
   set y to random(1,20)


   COMMENT ...this includes inside expressions!

   draw image smalldot at mousex+x mousey+y
  end if

end
```

Hints

1. Don't forget about the repeat instruction
2. You can change mousedown to mousepressed to check it's working

Glossary

This example introduces the new concepts of:

- random

Step 3

Task: At the moment, all dots appear to the right (or below) the mouse. Now change it so it could be to the left or above.

Code

```
reset

COMMENT This code uses methods, conditions, loops and funtions!

define update

 if mousedown
  repeat 5
   set x to random(1,20)
   set y to random(1,20)
   draw image smalldot at mousex+x mousey+y
  end repeat
 end if

end
```

HINTS

1. You could change the numbers in the random function
2. You could subtract a value from the random result

More ideas #1

Task: Could you combine the mirror idea from the previous example?

CODE

```
reset

define update

 if mousedown
  repeat 5
   set x to random(-20,20)
   set y to random(-20,20)
   draw image smalldot at mousex+x mousey+y
  end repeat
 end if

end
```

Paint Shop Bolt

Category: Interactions | *Level 3* | 6 steps

We are going to add colour to our painting application. Literally!

Step 1

Task: Using the 'AND' operation combine two IF statements into one. Only allow the colour change when the user clicks on the pencil.

Code

```
reset


COMMENT Previously, the only image you've drawn is a
COMMENT smalldot. Now we introduce one called 'pencil'

colour images "white"
draw image pencil at 20 20

colour images "blue"

define update

 COMMENT Here we introduce the 'else' command. When
 COMMENT the if statement is true, Bolt executes the
 COMMENT 'color' command, and jumps to the end if.
 COMMENT When the 'if' is false it jumps over that code
 COMMENT and executes code between 'else' and 'end if'

 if mousepressed AND mousey < 40

  colour images "red"

 else

  if mousedown
   draw image smalldot at mousex mousey
  end if

 end if

end
```

Hints

1. The centre of the pencil image is drawn at 20,20

2. The pencil is 40 by 40 pixels
3. The mouse X co-ordinate needs to be greater or equal to 0, and less than 40
4. When inside the first IF statement, you only need to check the mousex position

Glossary

This example introduces the new concepts of:

- and

Step 2

Task: Don't draw any dots that appear in the top part of the screen.

Code

```
reset

colour images "white"
draw image pencil at 20 20

colour images "blue"

define update

 if mousepressed AND mousey < 40

  if mousex >= 0 AND mousex < 40
   colour images "red"
  end if

 else

  if mousedown
   draw image smalldot at mousex mousey
  end if

 end if

end
```

Hints

1. This is the area where y is great than, or equal to, 40
2. Greater or equal to is the standard maths symbol '>='

Step 3

Task: The colour of the image is tinted by the last 'colour image' instruction you use. Draw pencils in 'red', 'green', and 'blue'.

Code

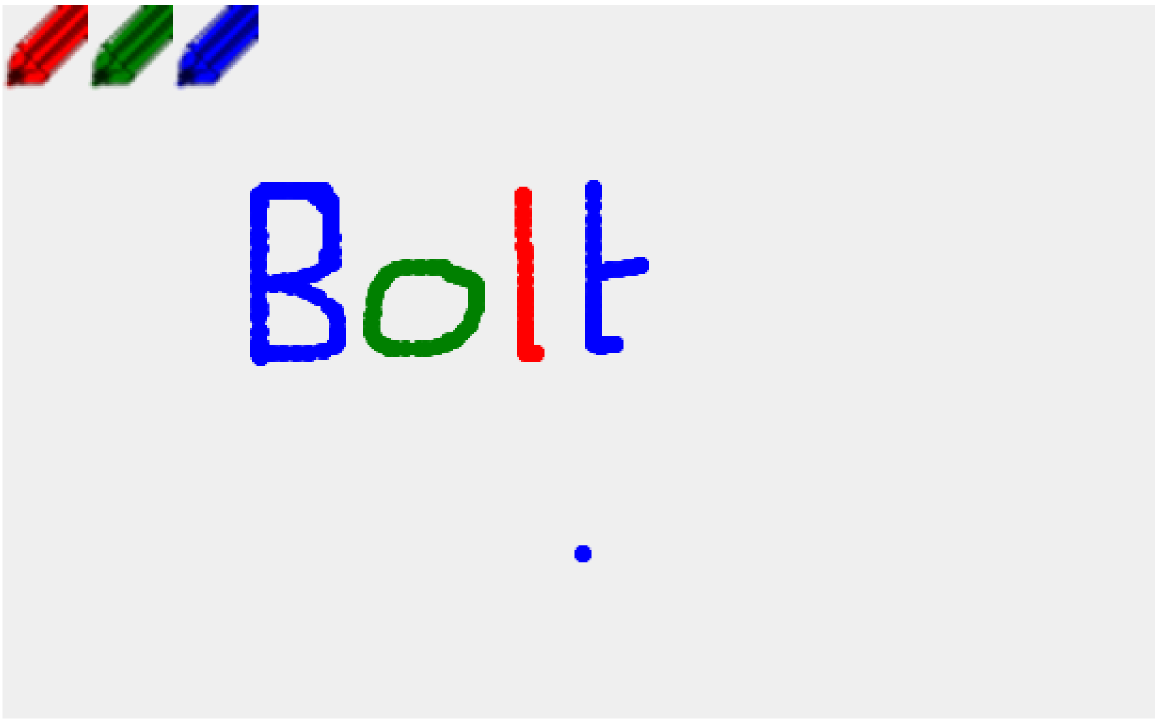
```
reset

colour images "white"
draw image pencil at 20 20

colour images "blue"

define update

 if mousepressed AND mousey < 40

  if mousex >= 0 AND mousex < 40
   colour images "red"
  end if

 else

  if mousedown AND mousey >= 40
   draw image smalldot at mousex mousey
  end if

 end if

end
```

Hints

1. Make sure the pencils aren't drawn on top of each other.
2. The pencil graphic is 40 pixels wide
3. The images need to be at 20, 60, and 80, because we draw from the middle

Step 4

Task: Can you click on each pencil, and change to the correct colour?



Code

```
reset


colour images "red"
draw image pencil at 20 20

colour images "green"
draw image pencil at 60 20

colour images "blue"
draw image pencil at 100 20


colour images "blue"

define update

 if mousepressed AND mousey < 40

  if mousex >= 0 AND mousex < 40
   colour images "red"
  end if

 else

  if mousedown AND mousey >= 40
   draw image smalldot at mousex mousey
  end if

 end if

end
```

Hints

1. You only need to change the colour once
2. Once Bolt executes the instruction 'colour images red', it will colour all future images, until told something different.
3. An image drawn at 30,10 - for example - has an area of 20,0 to 40,20

More ideas #1

Task: Could you combine the random spray can idea from the previous examples?

Code

```
reset


colour images "red"
draw image pencil at 20 20

colour images "green"
draw image pencil at 60 20

colour images "blue"
draw image pencil at 100 20


colour images "blue"

define update

 if mousepressed AND mousey < 40

  COMMENT We can 'nest' if statements inside one
  COMMENT another if we like.

  if mousex >= 0 AND mousex < 40
   colour images "red"
  end if

  if mousex >= 40 AND mousex < 80
   colour images "green"
  end if

  if mousex >= 80 AND mousex < 120
   colour images "blue"
  end if

 else

  if mousedown AND mousey >= 40
   draw image smalldot at mousex mousey
  end if
 end if

end
```

More ideas #2

Task: Can you add other colours to the app?

Code

```
reset


colour images "red"
draw image pencil at 20 20

colour images "green"
draw image pencil at 60 20

colour images "blue"
draw image pencil at 100 20


colour images "blue"

define update

 if mousepressed AND mousey < 40

  COMMENT We can 'nest' if statements inside one
  COMMENT another if we like.

  if mousex >= 0 AND mousex < 40
   colour images "red"
  end if

  if mousex >= 40 AND mousex < 80
   colour images "green"
  end if

  if mousex >= 80 AND mousex < 120
   colour images "blue"
  end if

 else

  if mousedown AND mousey >= 40
   draw image smalldot at mousex mousey
  end if

 end if

end
```

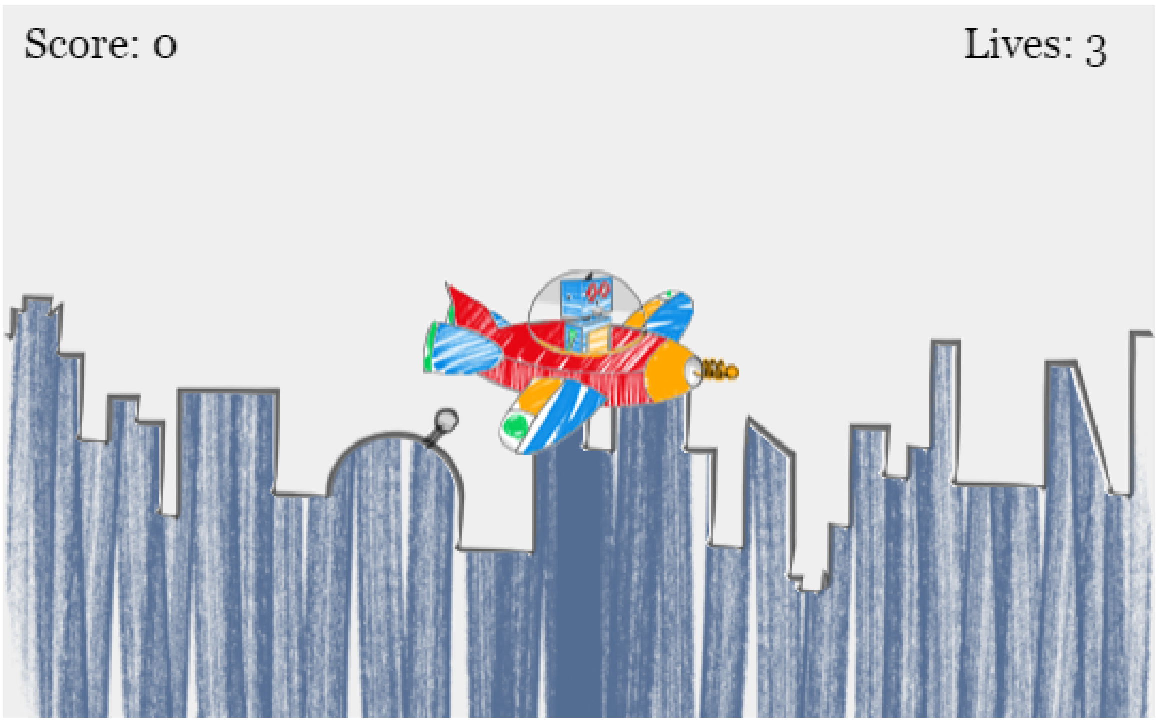
Catch Bolt

Category: **Games** | ***Level 1*** | 7 steps

Whenever Bolt appears on the screen - find him! (By clicking on him.)

Step 1

Task: At the start of each level, position Bolt somewhere randomly on the screen.



Code

```
COMMENT At the start of a line tells Bolt to ignore it
COMMENT Real coders use COMMENTs to remind us
COMMENT how the program works.

set score to 0
set lives to 3

define newlevel

 draw background blank

 COMMENT Find a position for Bolt to appear on
 COMMENT the screen, and draw him.

 draw image bolt at canvas_width/2 canvas_height/2

 print at 10 25 "Score: " score
 print at 450 25 "Lives: " lives

end
```

```
define update

 COMMENT We don't 'clear' or 'reset' the screen in the
 COMMENT 'update' code, because that would erase the
 COMMENT background and Bolt image we drew in the
 COMMENT 'newlevel' method

end


call newlevel
```

Hints

1. What function do you need to calculate a random position?
2. To be visible on-screen, both X and Y positions need to be within the canvas area
3. You can restart the program to see if Bolt's position is truly random

Glossary

This example introduces the new concepts of:

- print

Step 2

Task: Create a timer, by drawing a longer line on each update. Also, make it wider, and position it at the bottom of the canvas.

Code

```
set score to 0
set lives to 3

define newlevel

 draw background blank

 set x to random(0,canvas_width)
 set y to random(0,canvas_height)

 draw image bolt at x y

 print at 10 25 "Score: " score
 print at 450 25 "Lives: " lives

 set time to 0
```

```
end


define update

 width 1
 color drawing "red"
 position 0 300
 line time 300

end


call newlevel
```

Glossary

This example introduces the new concepts of:

- width
- line

Step 3

Task: When the timer ends you should lose a life and start a new level. The game should stop when you have no lives left.

Code

```
set score to 0
set lives to 3

define newlevel

 draw background blank

 set x to random(0,canvas_width)
 set y to random(0,canvas_height)

 draw image bolt at x y

 print at 10 25 "Score: " score
 print at 450 25 "Lives: " lives

 set time to 0

end


define update

 COMMENT We draw a line that's 10 pixels wide, so we
 COMMENT must use a Y position that's 5 pixels away
```

```
  COMMENT from the bottom of the canvas. i.e. it will
  COMMENT be 5 pixels above the Y co-ordinate, and 5
  COMMENT 5 pixels below it, making 10 in total.

  width 10
  color drawing "red"
  position 0 canvas_height-5
  line time canvas_height-5

  COMMENT Our 'time' variable is also our X co-ordinate
  COMMENT We increase it each 'update' which appears to
  COMMENT make it move.

  set time to time+10

end


call newlevel
```

Hints

1. The time variable is both the time spent hunting for Bolt, and its position on screen
2. Remember the 'stop' instruction
3. The 'if' instruction can have an 'else' statement to handle the other condition

Step 4

Task: Let the player click on, or near Bolt, to score points and start a new level

Code

```
set score to 0
set lives to 3

define newlevel

  draw background blank

  set x random(0,canvas_width)
  set y random(0,canvas_height)

  draw image bolt at x y

  print at 10 25 "Score: " score
  print at 450 25 "Lives: " lives

  set time to 0

end
```

```
define update

 width 10
 color drawing "red"
 position 0 canvas_height-5
 line time canvas_height-5

 COMMENT Our 'time' variable is also our X co-ordinate
 COMMENT We increase it each 'update' which appears
 COMMENT to make it move.

 set time to time+10

 if time > canvas_width

  set lives to lives-1

  if lives=0
   print at 190 70 "GAME OVER!"
   stop
  else
   call newlevel
  end if
 end if

end


call newlevel
```

Hints

1. You don't want to use 'mousedown' here, because that's True all the time whilst the button is being pressed. You only want to check the position when the button is _first_ pressed
2. Bolt is about 150 pixels by 90

Glossary

This example introduces the new concepts of:

- stop

More ideas #1

Task: How would score more points for finding Bolt more quickly?

Code

```
set score to 0
set lives to 3
```

```
define newlevel

 draw background blank

 set x to random(0,canvas_width)
 set y to random(0,canvas_height)

 draw image bolt at x y

 print at 10 25 "Score: " score
 print at 450 25 "Lives: " lives

 set time to 0

end


define update

 width 10
 color drawing "red"
 position 0 canvas_height-5
 line time canvas_height-5

 set time to time+10

 if time > canvas_width

  set lives to lives-1

  if lives=0
   print at 190 70 "GAME OVER!"
   stop
  else
   call newlevel
  end if
 end if

 if mouseclick
  COMMENT We know that Bolt is about 150 pixels
  COMMENT by 90. We also know that when he's is
  COMMENT drawn at x y, he covers an area from
  COMMENT x-75 to x+75, and from y-45 to y+45

  if mousex > x-75 AND mousex < x+75 and mousey > y-45 AND mousey < y+45

   set score to score+1
   set time to 0
   call newlevel

  end if

  end if

end


call newlevel
```

More ideas #2

Task: When you run out of lives, the screen still says 'Lives: 1' - how would you fix this?

More ideas #3

Task: Could you make Bolt fly around the screen while the player tries to catch him?

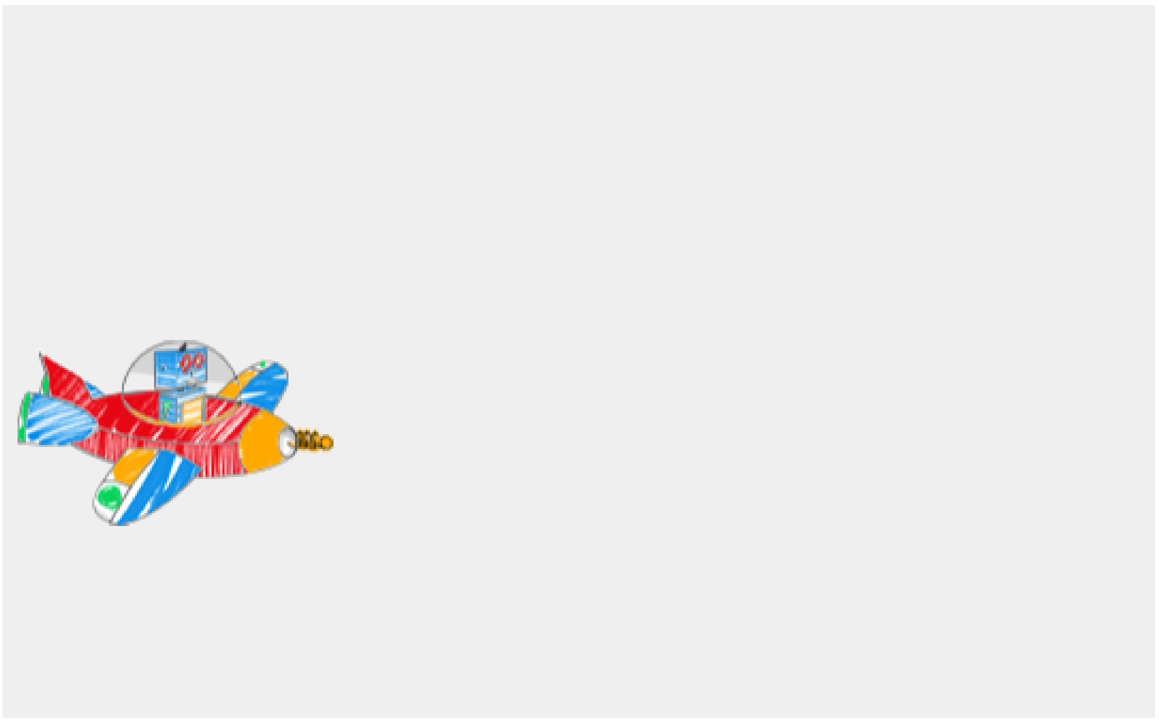
Flappy Bolt

Category: **Games** | *Level 2* | 8 steps

Fly Bolt safely through the obstacles in Robot City by pressing the mouse button, which makes him fly higher

Moving bolt

Task: Control Bolt's plane: the mouse button fires his engines to go higher, otherwise make the plane descend



Code

```
reset

set bolt_y to 200

define move_bolt

 COMMENT Move bolt down the screen every time this
 COMMENT function is called, unless the mouse button
 COMMENT is down - in which case move him up!

end

define draw_bolt
 draw image bolt plane at 80 bolt_y
end

define update
```

```
  clear

  call move_bolt
  call draw_bolt

end
```

Hints

1. Moving down the screen means lowering the value for Y. Moving up the screen requires you to increase it.
2. Try moving down 2 pixels each time
3. If the mouse is down, try moving up 3 pixels

Glossary

This example introduces the new concepts of:

- clear

Adding a background

Task: Let's add a scrolling background to make it look like he's flying!

Code

```
reset

set bolt_y to 200
set background_offset to canvas_width/2

define move_bolt

 COMMENT We could also write this as if-else

 set bolt_y to bolt_y+2
 if mousedown
  set bolt_y to bolt_y-5
 end if

end

define draw_bolt
 draw image bolt plane at 80 bolt_y
end


define move_background

 COMMENT Change background_offset to make it look
 COMMENT like Bolt's plane is flying past the city

end

define draw_background

 COMMENT After moving the city background

 draw background city at background_offset 167

end


define update

 COMMENT What happens if we remove this 'clear' instruction? Why?

 clear

 call move_background
 call move_bolt

 call draw_background
 call draw_bolt

end
```

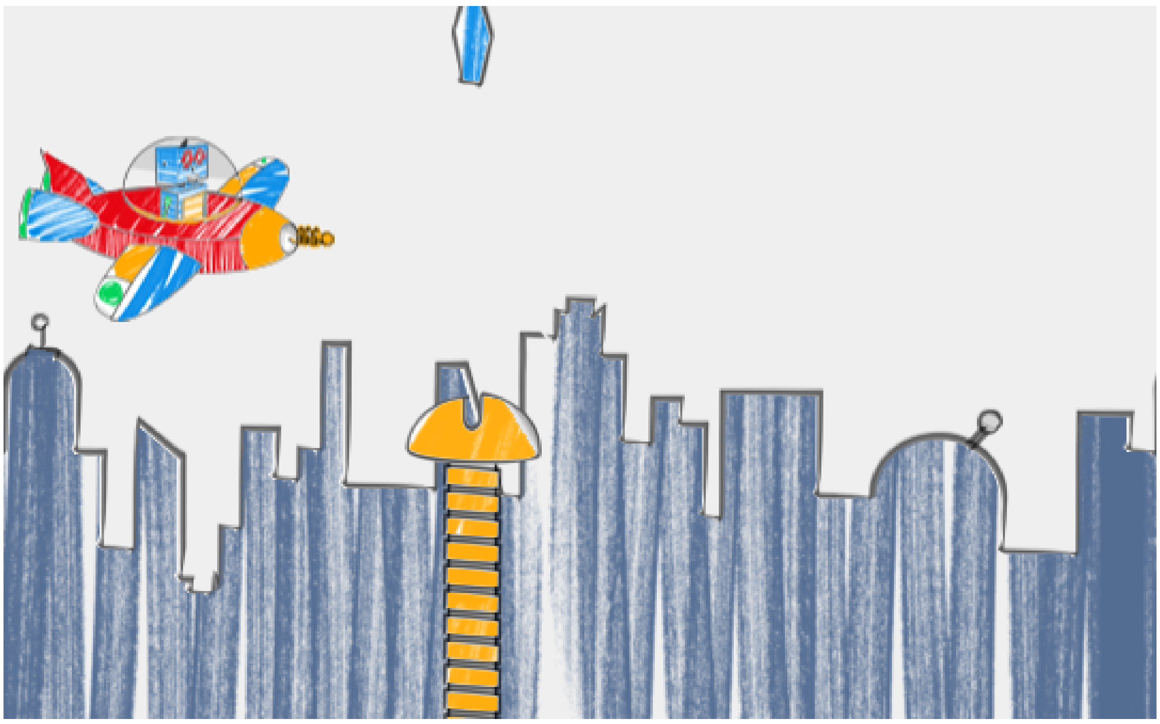
Hints

1. To make it look like the plane is moving from left to right, how should the background move?
2. You need to vary the background_offset variable

3. You might need to draw the background more than once to cover the whole screen
4. The width of the background image is the same as the canvas

Adding some hazards

Task: We'll add moving barriers in an attempt to block Bolt's path!



Code

```
reset

set bolt_y to 200
set background_offset to canvas_width/2

define move_bolt

 set bolt_y to bolt_y+2
 if mousedown
  set bolt_y to bolt_y-5
 end if

end

define draw_bolt
 draw image bolt plane at 80 bolt_y
end


define move_background
 set background_offset to background_offset-1
 if background_offset < -270
  set background_offset to 270
 end if
```

```
end

define draw_background
 draw background city at background_offset 167
 draw background city at background_offset+540 167
end


define create_new_barrier

 COMMENT Here is the first barrier, positioned at the
 COMMENT top of the screen.

 set barrier_top to random(0,140)


 COMMENT Is this the best position to start
 COMMENT the barrier?

 set barrier_x to 200


 COMMENT Now add a second barrier at the bottom

end

define move_barrier
 COMMENT Move both barriers left, each time this
 COMMENT method is called.

 COMMENT If the barrier has left the screen, then
 COMMENT we create a new barrier

 if barrier_x < 0
  call create_new_barrier
 end if
end

define draw_barrier

 COMMENT Drawn the bottom barrier, too

 draw image top barrier at barrier_x barrier_top-167

end


define update

 call move_background
 call move_barrier
 call move_bolt

 call draw_background
 call draw_barrier
 call draw_bolt

end

COMMENT Our program actually starts here!

COMMENT Create the new barrier
```

```
call create_new_barrier

COMMENT And let the system call our 'update' function
```

Hints

1. Do both barriers move together, maintaining the same X position?
2. Moving barriers is exactly like moving the background
3. Remember that all graphics are drawn from their centre point
4. If you have trouble positioning the graphics, the barriers are the same height as the canvas

Collisions

Task: We now need to check that Bolt really has avoided those obstacles!

Code

```
reset

set bolt_y to 200
set background_offset to canvas_width/2

define move_bolt

 set bolt_y to bolt_y+2
 if mousedown
  set bolt_y to bolt_y-5
 end if

 COMMENT First, check to see if Bolt's X
 COMMENT co-ordinate is likely to collide with
 COMMENT the barriers.

 if barrier_x>5 and barrier_x < 100
  COMMENT Then check his Y co-ordinates...

  if false
   print at 200 120 "GAME OVER!"
   stop
  end if
 end if

 COMMENT We should also stop the game if he
 COMMENT flies off the top of the screen,
 COMMENT or drops below the bottom.

end

define draw_bolt
 draw image bolt plane at 80 bolt_y
```

```
end


define move_background
 set background_offset to background_offset-1
 if background_offset < -270
  set background_offset to 270
 end if
end

define draw_background
 draw background city at background_offset 167
 draw background city at background_offset+540 167
end


define create_new_barrier

 COMMENT Here is the first barrier, positioned at the
 COMMENT top of the screen.

 set barrier_top to random(0,140)


 COMMENT We position the bottom barrier between
 COMMENT 80 and 150 pixels away from the top.
 COMMENT These numbers indicate the gap, and
 COMMENT how difficult the game will be.

 set barrier_bottom to barrier_top+random(120,250)


 COMMENT Both barriers move together, so we only
 COMMENT need only X co-ordinate. (But you could
 COMMENT change this if you like!)
 COMMENT Also note we start the barrier off-screen

 set barrier_x to canvas_width

end

define move_barrier

 COMMENT Move both barriers left, each time this
 COMMENT method is called.

 set barrier_x to barrier_x-5

 if barrier_x < 0
  call create_new_barrier
 end if
end

define draw_barrier

 draw image top barrier at barrier_x barrier_top-167
 draw image bottom barrier at barrier_x barrier_bottom+167

end


define update
```

```
  call move_background
  call move_barrier
  call move_bolt

  call draw_background
  call draw_barrier
  call draw_bolt

 end

 COMMENT Our program actually starts here!

 COMMENT Create the new barrier

 call create_new_barrier

 COMMENT And let the system call our 'update' function
```

Hints

1. Do both barriers move together, maintaining the same X position?
2. Moving barriers is exactly like moving the background
3. Remember that all graphics are drawn from their centre point
4. If you have trouble positioning the graphics, the barriers are the same height as the canvas

Final polish

Task: Let's keep score! Add 10 points every time Bolt passes an obstacle.

Code

```
reset

set bolt_y to 200
set background_offset to canvas_width/2

define move_bolt

 set bolt_y to bolt_y+2
 if mousedown
  set bolt_y to bolt_y-5
 end if

 COMMENT First, check to see if Bolt's X
 COMMENT co-ordinate is likely to collide with
 COMMENT the barriers.

 if barrier_x>5 and barrier_x < 100
  COMMENT Then check his Y co-ordinates...
```

```
   if bolt_y-40 < barrier_top or bolt_y+40 > barrier_bottom
    print at 200 120 "GAME OVER!"
    stop
   end if
  end if

  COMMENT We should also stop the game if he
  COMMENT flies off the top of the screen,
  COMMENT or drops below the bottom.

  if bolt_y < 0 or bolt_y > 340
   print at 210 120 "You crashed!"
   stop
  end if

 end

 define draw_bolt
  draw image bolt plane at 80 bolt_y
 end


 define move_background
  set background_offset to background_offset-1
  if background_offset < -270
   set background_offset to 270
  end if
 end

 define draw_background
  draw background city at background_offset 167
  draw background city at background_offset+540 167
 end


 define create_new_barrier

  COMMENT Here is the first barrier, positioned at the
  COMMENT top of the screen.

  set barrier_top to random(0,140)


  COMMENT We position the bottom barrier between
  COMMENT 80 and 150 pixels away from the top.
  COMMENT These numbers indicate the gap, and
  COMMENT how difficult the game will be.

  set barrier_bottom to barrier_top+random(120,250)


  COMMENT Both barriers move together, so we only
  COMMENT need only X co-ordinate. (But you could
  COMMENT change this if you like!)
  COMMENT Also note we start the barrier off-screen

  set barrier_x to canvas_width

 end

 define move_barrier

  COMMENT Move both barriers left, each time this
  COMMENT method is called.

  set barrier_x to barrier_x-5

  if barrier_x < 0
   call create_new_barrier
  end if
end

define draw_barrier

  draw image top barrier at barrier_x barrier_top-167
  draw image bottom barrier at barrier_x barrier_bottom+167

end


define update

  call move_background
  call move_barrier
  call move_bolt

  call draw_background
  call draw_barrier
  call draw_bolt

end

COMMENT Our program actually starts here!

COMMENT Create the new barrier

call create_new_barrier

COMMENT And let the system call our 'update' function
```

Hints

1. Do both barriers move together, maintaining the same X position?
2. Moving barriers is exactly like moving the background
3. Remember that all graphics are drawn from their centre point
4. If you have trouble positioning the graphics, the barriers are the same height as the canvas

More ideas #1

Task: Can you add a high score?

Code

```
reset

set score  to 0
set bolt_y  to 200
set background_offset to canvas_width/2

define move_bolt

 set bolt_y to bolt_y+2
 if mousedown
  set bolt_y to bolt_y-5
 end if

 COMMENT First, check to see if Bolt's X
 COMMENT co-ordinate is likely to collide with
 COMMENT the barriers.

 if barrier_x>5 and barrier_x < 100
  COMMENT Then check his Y co-ordinates...

  if bolt_y-40 < barrier_top or bolt_y+40 > barrier_bottom
   print at 200 120 "GAME OVER!"
   stop
  end if
 end if

 COMMENT We should also stop the game if he
 COMMENT flies off the top of the screen,
 COMMENT or drops below the bottom.

 if bolt_y < 0 or bolt_y > 340
  print at 210 120 "You crashed!"
  stop
 end if

end

define draw_bolt
 draw image bolt plane at 80 bolt_y
end


define move_background
 set background_offset to background_offset-1
 if background_offset < -270
  set background_offset to 270
 end if
end

define draw_background
 draw background city at background_offset 167
 draw background city at background_offset+540 167
end


define create_new_barrier

 COMMENT Here is the first barrier, positioned at the
 COMMENT top of the screen.

 set barrier_top to random(0,140)
 COMMENT We position the bottom barrier between
 COMMENT 80 and 150 pixels away from the top.
 COMMENT These numbers indicate the gap, and
 COMMENT how difficult the game will be.

 set barrier_bottom to barrier_top+random(120,250)


 COMMENT Both barriers move together, so we only
 COMMENT need only X co-ordinate. (But you could
 COMMENT change this if you like!)
 COMMENT Also note we start the barrier off-screen

 set barrier_x to canvas_width

end

define move_barrier

 COMMENT Move both barriers left, each time this
 COMMENT method is called.

 set barrier_x to barrier_x-5

 if barrier_x < 0
  set score to score+10
  call create_new_barrier
 end if
end

define draw_barrier

 draw image top barrier at barrier_x barrier_top-167
 draw image bottom barrier at barrier_x barrier_bottom+167

end


define draw_scoring
 print at 10 20 "Score: " score
end

define update

 call move_background
 call move_barrier
 call move_bolt

 call draw_background
 call draw_barrier
 call draw_bolt
 call draw_scoring

end

COMMENT Our program actually starts here!

COMMENT Create the new barrier

call create_new_barrier

COMMENT And let the system call our 'update' function
```

More ideas #2

Task: There's background called 'clouds'. Draw two of them above the city, and make them move at different speeds.

Code

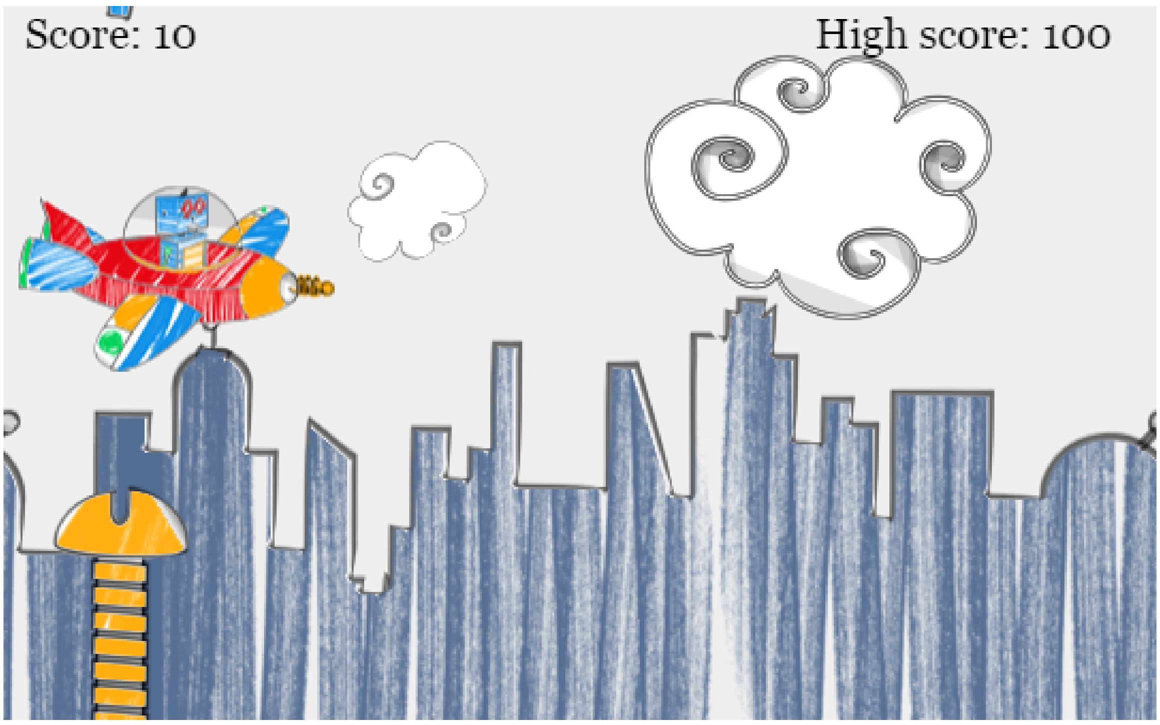
```
reset

set score to 0
set high_score to 100
set bolt_y to 200
set background_offset to canvas_width/2

define move_bolt

 set bolt_y to bolt_y+2
 if mousedown
  set bolt_y to bolt_y-5
 end if

 COMMENT First, check to see if Bolt's X
 COMMENT co-ordinate is likely to collide with
 COMMENT the barriers.

 if barrier_x>5 and barrier_x < 100
  COMMENT Then check his Y co-ordinates...

  if bolt_y-40 < barrier_top or bolt_y+40 > barrier_bottom
   print at 200 120 "GAME OVER!"
   stop
  end if
 end if

 COMMENT We should also stop the game if he
 COMMENT flies off the top of the screen,
 COMMENT or drops below the bottom.

 if bolt_y < 0 or bolt_y > 340
  print at 210 120 "You crashed!"
  stop
 end if

end

define draw_bolt
 draw image bolt plane at 80 bolt_y
end


define move_background
 set background_offset to background_offset-1
 if background_offset < -270
  set background_offset to 270
 end if
```

```
end

define draw_background
 draw background city at background_offset 167
 draw background city at background_offset+540 167
end


define create_new_barrier

 COMMENT Here is the first barrier, positioned at the
 COMMENT top of the screen.

 set barrier_top to random(0,140)


 COMMENT We position the bottom barrier between
 COMMENT 80 and 150 pixels away from the top.
 COMMENT These numbers indicate the gap, and
 COMMENT how difficult the game will be.

 set barrier_bottom to barrier_top+random(120,250)


 COMMENT Both barriers move together, so we only
 COMMENT need only X co-ordinate. (But you could
 COMMENT change this if you like!)
 COMMENT Also note we start the barrier off-screen

 set barrier_x to canvas_width

end

define move_barrier

 COMMENT Move both barriers left, each time this
 COMMENT method is called.

 set barrier_x to barrier_x-5

 if barrier_x < 0
  set score to score+10
  if score > high_score
   set high_score to score
  end

  call create_new_barrier
 end if
end

define draw_barrier

 draw image top barrier at barrier_x barrier_top-167
 draw image bottom barrier at barrier_x barrier_bottom+167

end


define draw_scoring
 print at 10 20 "Score: " score
 print at 380 20 "High score: " high_score
end
```

```
define update

 call move_background
 call move_barrier
 call move_bolt

 call draw_background
 call draw_barrier
 call draw_bolt
 call draw_scoring

end

COMMENT Our program actually starts here!

COMMENT Create the new barrier
call create_new_barrier

COMMENT And let the system call our 'update' function
```

More ideas #3

Task: How about adding more than one barrier on-screen at once?



Code

```
reset

set score to
set high_score to 100
set bolt_y to 200
set background_offset to canvas_width/2
set cloud_offset to 0
```

```
define move_bolt

 set bolt_y to bolt_y+2
 if mousedown
  set bolt_y to bolt_y-5
 end if

 COMMENT First, check to see if Bolt's X
 COMMENT co-ordinate is likely to collide with
 COMMENT the barriers.

 if barrier_x>5 and barrier_x < 100
  COMMENT Then check his Y co-ordinates...

  if bolt_y-40 < barrier_top or bolt_y+40 > barrier_bottom
   print at 200 120 "GAME OVER!"
   stop
  end if
 end if

 COMMENT We should also stop the game if he
 COMMENT flies off the top of the screen,
 COMMENT or drops below the bottom.

 if bolt_y < 0 or bolt_y > 340
  print at 210 120 "You crashed!"
  stop
 end if

end

define draw_bolt
 draw image bolt plane at 80 bolt_y
end


define move_background
 set background_offset to background_offset-1
 if background_offset < -270
  set background_offset to 270
 end if

 set cloud_offset to cloud_offset-14
 if cloud_offset < -270
  set cloud_offset to 810
 end if
end

define draw_background
 draw background city at background_offset 167
 draw background city at background_offset+540 167
 draw background clouds at cloud_offset 167
end


define create_new_barrier

 COMMENT Here is the first barrier, positioned at the
 COMMENT top of the screen.

 set barrier_top to random(0,140)
```

```
  COMMENT We position the bottom barrier between
  COMMENT 80 and 150 pixels away from the top.
  COMMENT These numbers indicate the gap, and
  COMMENT how difficult the game will be.

  set barrier_bottom to barrier_top+random(120,250)


  COMMENT Both barriers move together, so we only
  COMMENT need only X co-ordinate. (But you could
  COMMENT change this if you like!)
  COMMENT Also note we start the barrier off-screen

  set barrier_x to canvas_width

 end

 define move_barrier

  COMMENT Move both barriers left, each time this
  COMMENT method is called.

  set barrier_x to barrier_x-5

  if barrier_x < 0
   set score to score+10
   if score > high_score
    set high_score to score
   end

   call create_new_barrier
  end if
 end

 define draw_barrier

  draw image top barrier at barrier_x barrier_top-167
  draw image bottom barrier at barrier_x barrier_bottom+167

 end


 define draw_scoring
  print at 10 20 "Score: " score
  print at 380 20 "High score: " high_score
 end

 define update

  call move_background
  call move_barrier
  call move_bolt

  call draw_background
  call draw_barrier
  call draw_bolt
  call draw_scoring

 end

 COMMENT Our program actually starts here!

 COMMENT Create the new barrier
```

```
call create_new_barrier

COMMENT And let the system call our 'update' function
```

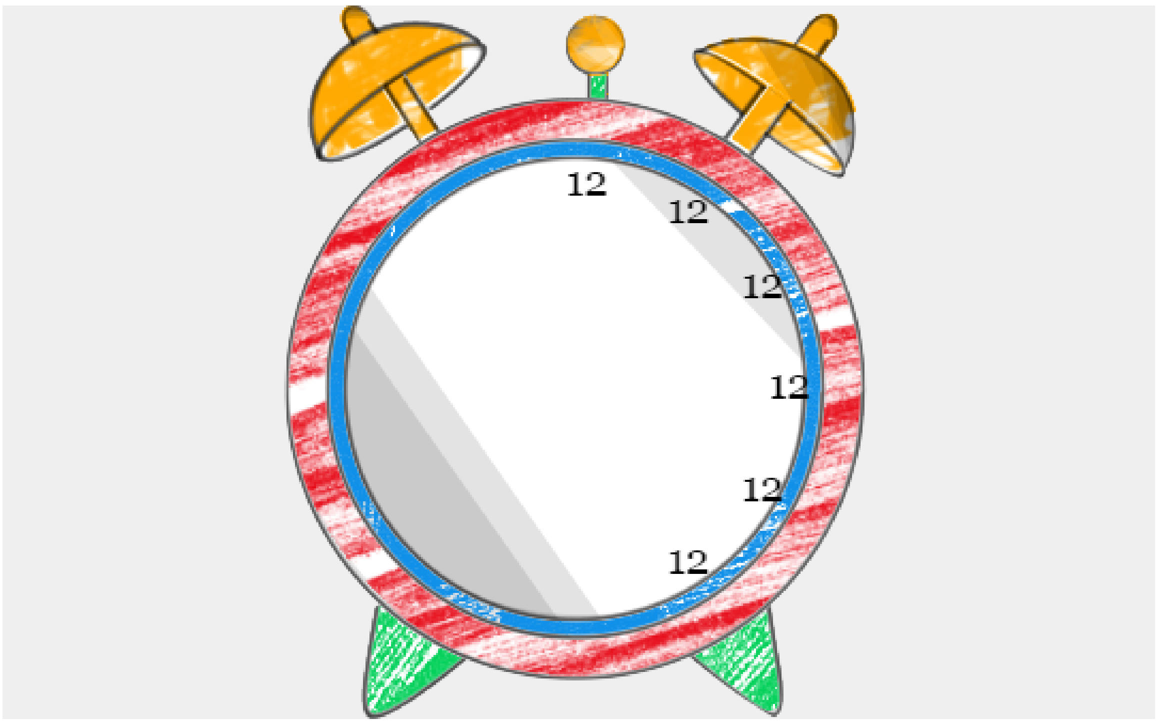
Clock

Category: Apps | *Level 2* | 7 steps

A working clock, with hands, just like the olden days

Step 1

Task: Complete the clock face with all 12 numbers, in the correct positions



Code

```
COMMENT At the start of a line tells Bolt to ignore it
COMMENT Real coders use COMMENTs to remind us
COMMENT how the program works.

set centrex to 264
set centrey to 184

define update

 COMMENT If we draw a background with any 'at position then
 COMMENT it will draw it in the centre of the screen. This is
 COMMENT as a 'default'

 draw background clockface


 COMMENT The 'number' variable indicates the hours on the clock face

 set number to 12
```

```
  COMMENT We start the angle at 90 degrees, which points towards
  COMMENT the top of the screen

  set angle to 90

  repeat 6

   COMMENT Calculate the position around the clock face using maths
   COMMENT Each number is 95 units (or pixels) away from the centre

   set x to centrex + (cos(angle)*95)
   set y to centrey - (sin(angle)*95)
   print at x y number


   COMMENT There are 360 degrees in a circle, and 12 numbers.
   COMMENT Therefore each number is 360/12 (i.e. 30) degrees apart.

   set angle to angle-30
  end


 end
```

Hints

1. You will need to change the repeat loop
2. Incrementing the number by one can be done with 'set number number+1'
3. You might need to use the 'if' statement to check for number=13

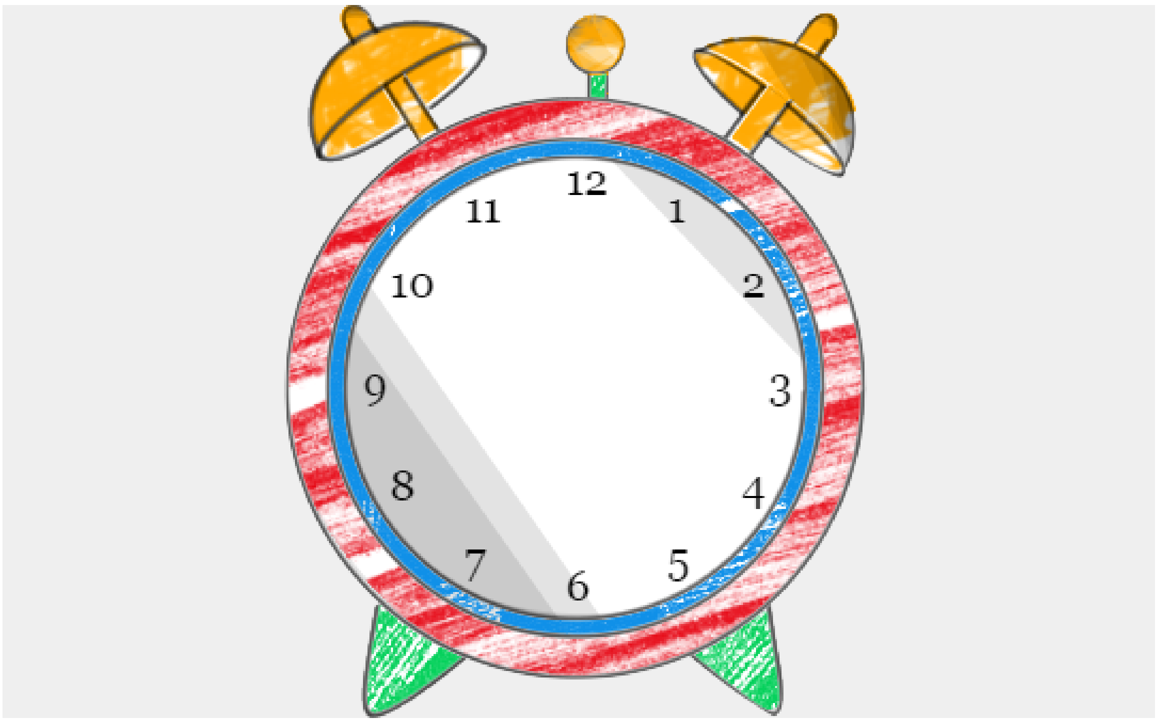
Glossary

This example introduces the new concepts of:

- sin
- cos

Step 2

Task: Change x and y to draw the second hand.



Code

```
set centrex to 264
set centrey to 184

define update

 draw background clockface

 COMMENT The 'number' variable indicates the hours on the clock face

 set number to 12


 COMMENT We start the angle at 90 degrees, which points towards
 COMMENT the top of the screen

 set angle to 90

 repeat 12

  COMMENT Calculate the position around the clock face using maths
  COMMENT Each number is 95 units (or pixels) away from the centre

  set x to centrex + (cos(angle)*95)
  set y to centrey - (sin(angle)*95)
  print at x y number

  set number to number+1
  if number=13
   set number to 1
  end if

  COMMENT There are 360 degrees in a circle, and 12 numbers.
  COMMENT Therefore each number is 360/12 (i.e. 30) degrees apart.

  set angle to angle-30
 end
```

```
  COMMENT Now draw the second hand. We have a pre-defined variable
  COMMENT called 'seconds' which tells us the time.
  COMMENT We need to work out the angle of the hand, given the
  COMMENT number of seconds.

  set angle to (seconds*360)/60


  COMMENT When seconds=0, angle=0 points to the right, where as it should be up
(i.e. 90)
  COMMENT When seconds=15, angle=90 points up, where as it should be right (i.e.
0, or 360)
  COMMENT When seconds=30, angle=180 points left, where as it should be down
(i.e. 270)
  COMMENT When seconds=45, angle=270 points down, where as it should be left
(i.e. 180)

  set angle to (90-angle)


  COMMENT We could use the same calculations from earlier to
  COMMENT work out the end point of the second hand.

  set x to 0
  set y to 0


  COMMENT We can now draw the hand

  colour drawing "red"
  position centrex centrey
  line x y

end
```

Hints

1. The code that works out the position of the numbers can be used
2. The numbers are 130 pixels away from the centre. The end of the second hand should only be 100 pixels away

Teacher Notes

This example begins with an extra red line to give a placeholder for the student to draw the clock hand

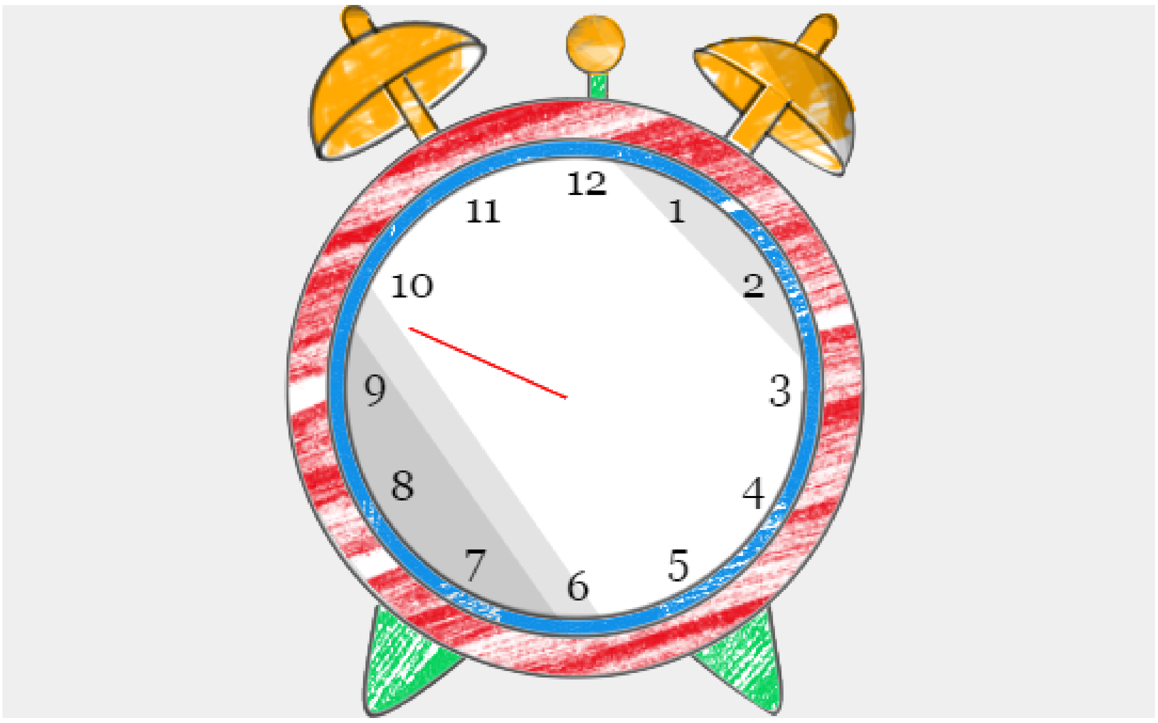
Glossary

This example introduces the new concepts of:

- seconds

Step 3

Task: Add the minute hand.



Code

```
set centrex to 264
set centrey to 184

define update

 draw background clockface

 COMMENT The 'number' variable indicates the hours on the clock face

 set number to 12


 COMMENT We start the angle at 90 degrees, which points towards
 COMMENT the top of the screen

 set angle to 90


 repeat 12

  COMMENT Calculate the position around the clock face using maths
  COMMENT Each number is 95 units (or pixels) away from the centre

  set x to centrex + (cos(angle)*95)
  set y to centrey - (sin(angle)*95)
  print at x y number

  set number to number+1
  if number=13
   set number to 1
```

```
    end if

    COMMENT There are 360 degrees in a circle, and 12 numbers.
    COMMENT Therefore each number is 360/12 (i.e. 30) degrees apart.

    set angle to angle-30
   end


   COMMENT Now draw the second hand. We have a pre-defined variable
   COMMENT called 'seconds' which tells us the time.
   COMMENT We need to work out the angle of the hand, given the
   COMMENT number of seconds.

   set angle to (seconds*360)/60


   COMMENT When seconds=0, angle=0 points to the right, where as it should be up
  (i.e. 90)
   COMMENT When seconds=15, angle=90 points up, where as it should be right (i.e.
  0, or 360)
   COMMENT When seconds=30, angle=180 points left, where as it should be down
  (i.e. 270)
   COMMENT When seconds=45, angle=270 points down, where as it should be left
  (i.e. 180)

   set angle to (90-angle)


   COMMENT We could use the same calculations from earlier to
   COMMENT work out the end point of the second hand.
   COMMENT The cos and sin functions will also handle cases
   COMMENT where the number is negative, or greater than 360

   set x to centrex + (cos(angle)*80)
   set y to centrey - (sin(angle)*80)


   COMMENT We can now draw the hand

   colour drawing "red"
   position centrex centrey
   line x y

  end
```

Hints

1. The 'minutes' variable tells you the minutes, between 0 and 59.
2. The minute hand is shorter than the second hand

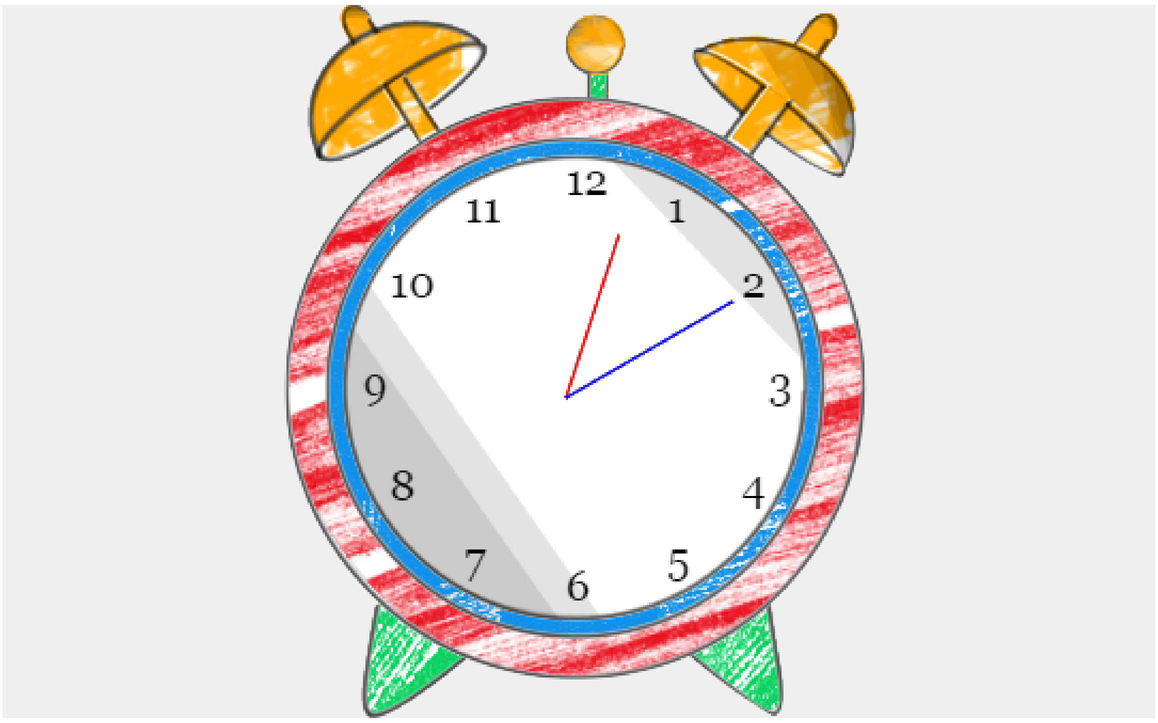
Glossary

This example introduces the new concepts of:

- minutes

Step 4

Task: Add the hour hand.



Code

```
set centrex to 264
set centrey to 184

define update

 draw background clockface

 COMMENT The 'number' variable indicates the hours on the clock face

 set number to 12


 COMMENT We start the angle at 90 degrees, which points towards
 COMMENT the top of the screen

 set angle to 90


 repeat 12

  COMMENT Calculate the position around the clock face using maths
  COMMENT Each number is 95 units (or pixels) away from the centre

  set x to centrex + (cos(angle)*95)
  set y to centrey - (sin(angle)*95)
  print at x y number

  set number to number+1
  if number=13
   set number to 1
  end if

  COMMENT There are 360 degrees in a circle, and 12 numbers.
  COMMENT Therefore each number is 360/12 (i.e. 30) degrees apart.

  set angle to angle-30
 end


 COMMENT Now draw the second hand. We have a pre-defined variable
 COMMENT called 'seconds' which tells us the time.
 COMMENT We need to work out the angle of the hand, given the
 COMMENT number of seconds.

 set angle to (seconds*360)/60


 COMMENT When seconds=0, angle=0 points to the right, where as it should be up
(i.e. 90)
 COMMENT When seconds=15, angle=90 points up, where as it should be right (i.e.
0, or 360)
 COMMENT When seconds=30, angle=180 points left, where as it should be down
(i.e. 270)
 COMMENT When seconds=45, angle=270 points down, where as it should be left
(i.e. 180)

 set angle to (90-angle)


 COMMENT We could use the same calculations from earlier to
 COMMENT work out the end point of the second hand.
 COMMENT The cos and sin functions will also handle cases
 COMMENT where the number is negative, or greater than 360

 set x to centrex + (cos(angle)*80)
 set y to centrey - (sin(angle)*80)


 COMMENT We can now draw the hand

 colour drawing "red"
 position centrex centrey
 line x y



 COMMENT We can repeat the process for the minute hand

 set angle to 90-(minutes*360)/60


 COMMENT The computer always performs multiplications (*) before additions +
 COMMENT therefore, we don't need the brackets.

 set x to centrex + cos(angle)*90
 set y to centrey - sin(angle)*90
 colour drawing "blue"
 position centrex centrey
 line x y

end
```

Hints

1. The 'hours' variable tells you the hour, between 0 and 23.
2. The hour hand is shorter than the rest

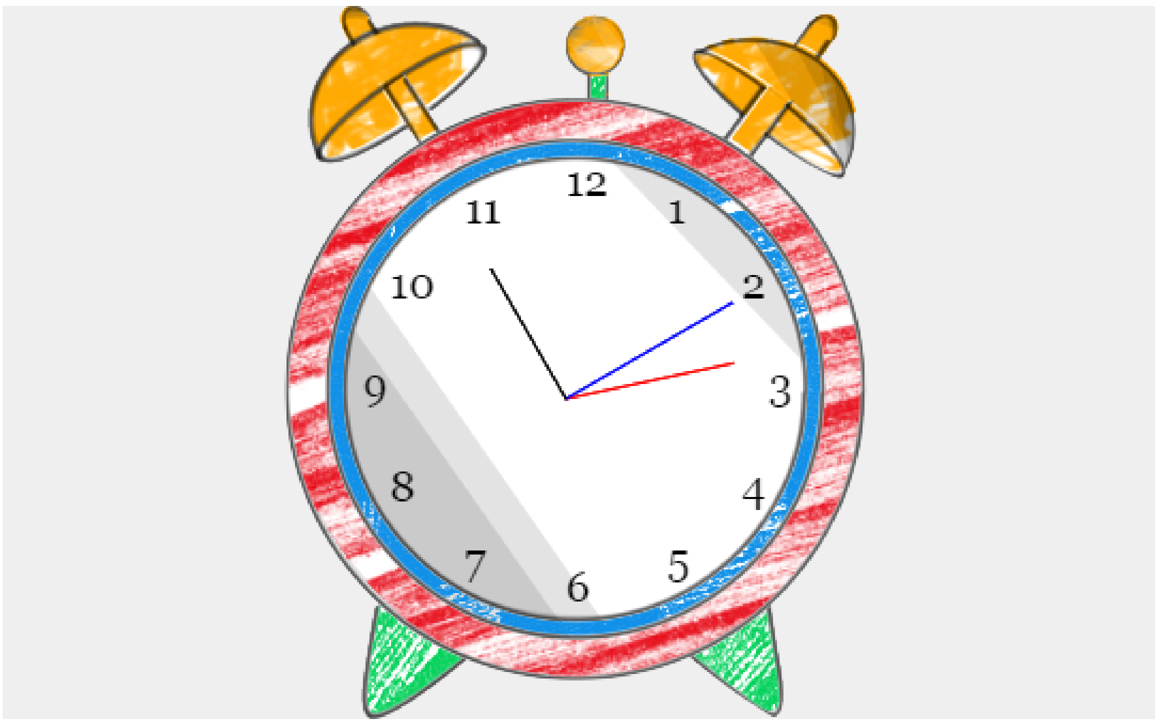
Glossary

This example introduces the new concepts of:

- hours

More ideas #1

Task: Hour hands move gradually between each hour. How would you accomplish this?



Code

```
set centrex to 264
set centrey to 184

define update

 draw background clockface

 COMMENT The 'number' variable indicates the hours on the clock face

 set number to 12
 COMMENT We start the angle at 90 degrees, which points towards
 COMMENT the top of the screen

 set angle to 90

 repeat 12

  COMMENT Calculate the position around the clock face using maths
  COMMENT Each number is 95 units (or pixels) away from the centre

  set x to centrex + (cos(angle)*95)
  set y to centrey - (sin(angle)*95)
  print at x y number

  set number to number+1
  if number=13
   set number to 1
  end if

  COMMENT There are 360 degrees in a circle, and 12 numbers.
  COMMENT Therefore each number is 360/12 (i.e. 30) degrees apart.

  set angle to angle-30
 end


 COMMENT Now draw the second hand. We have a pre-defined variable
 COMMENT called 'seconds' which tells us the time.
 COMMENT We need to work out the angle of the hand, given the
 COMMENT number of seconds.

 set angle to (seconds*360)/60


 COMMENT When seconds=0, angle=0 points to the right, where as it should be up
(i.e. 90)
 COMMENT When seconds=15, angle=90 points up, where as it should be right (i.e.
0, or 360)
 COMMENT When seconds=30, angle=180 points left, where as it should be down
(i.e. 270)
 COMMENT When seconds=45, angle=270 points down, where as it should be left
(i.e. 180)

 set angle to (90-angle)


 COMMENT We could use the same calculations from earlier to
 COMMENT work out the end point of the second hand.
 COMMENT The cos and sin functions will also handle cases
 COMMENT where the number is negative, or greater than 360

 set x to centrex + (cos(angle)*80)
 set y to centrey - (sin(angle)*80)


 COMMENT We can now draw the hand

 colour drawing "red"
 position centrex centrey
 line x y
```

```
  COMMENT We can repeat the process for the minute hand

  set angle to 90-(minutes*360)/60


  COMMENT The computer always performs multiplications (*) before additions +
  COMMENT therefore, we don't need the brackets.

  set x to centrex + cos(angle)*90
  set y to centrey - sin(angle)*90
  colour drawing "blue"
  position centrex centrey
  line x y


  COMMENT We can repeat the process for the hours hand
  COMMENT The hours is in 24 hour clock, 0 to 23

  set angle to 90 - (hours*360)/12

  set x to centrex + cos(angle)*70
  set y to centrey - sin(angle)*70
  colour drawing "black"
  position centrex centrey
  line x y

 end
```

Hints

1. The hour hands doesn't need to use only the hour data.

More ideas #2

Task: Can you draw a dot in the centre of the hands. Or at the tip of the hands?

More ideas #3

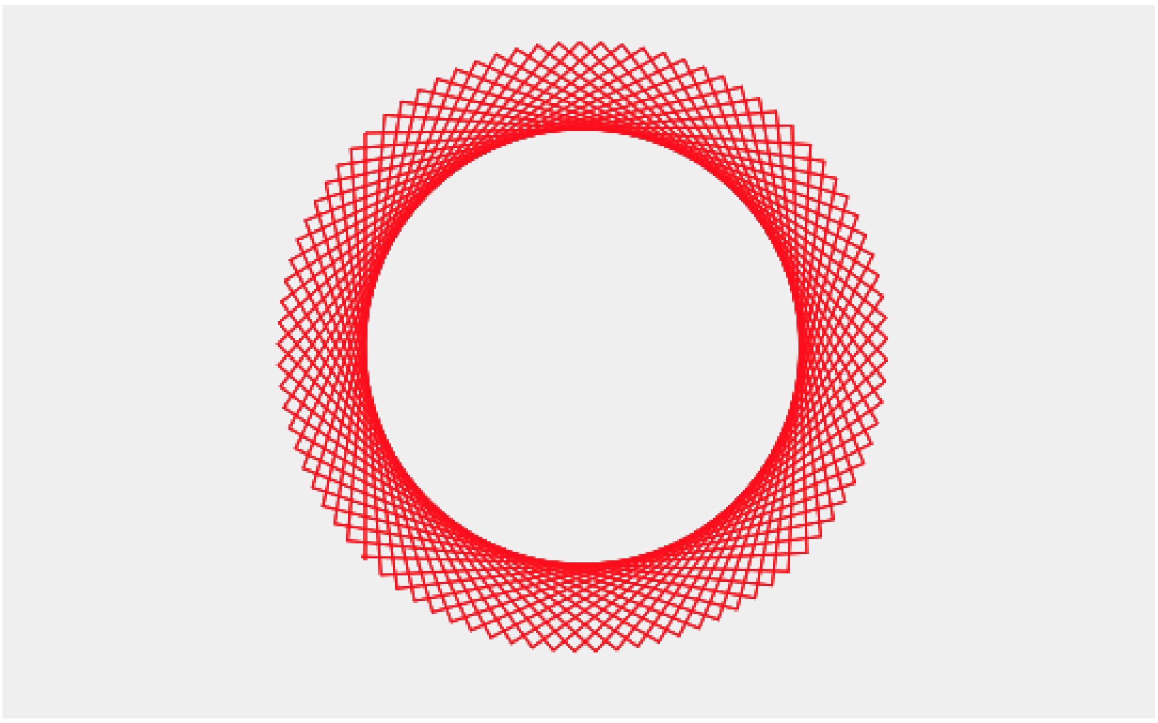
Task: Print the time in digital form on the screen. Include AM or PM, as appropriate.

Spirograph

Category: **Playtime** | ***Level 1***

Play with me!

Task: Try changing the values of the variables 'length', 'angle', and 'count' to see what shapes you can make!



Code

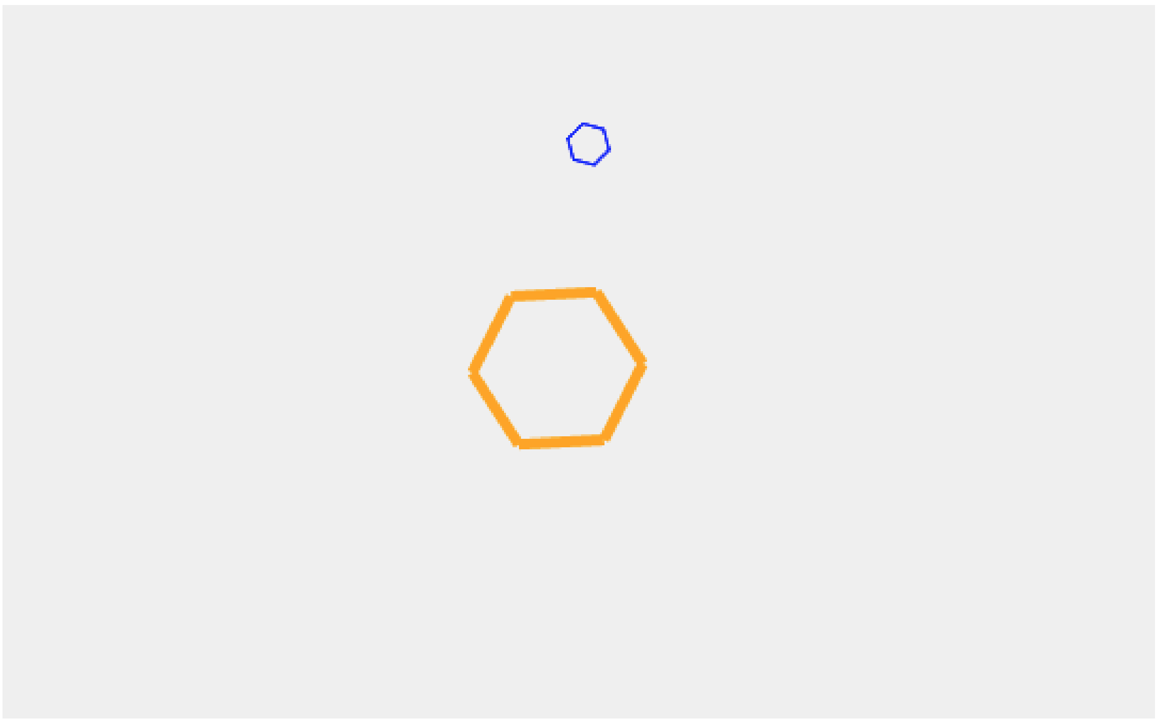
```
reset

set length to 200
set angle to 89
set count to 89

position 170 260
colour draw "red"

repeat count

 forward length
 turn right angle

end
```

Planets

Category: Apps | ***Level 1***

Play with me!

Task: Try changing the values of the variables to see what you can do! Remember, there is no right or wrong answers here.



Code

```
reset
scale 100

define shape
 set angle to 360/sides

 COMMENT Move from the centre to the edge by
 COMMENT calculating the radius
 pen up
  back length/2
  turn right 90
  COMMENT 1.73 is square root(3) and only
  COMMENT works for hexagons
  back length/2*1.73
  turn left 90
 pen down

 COMMENT Now draw the shape itself
 repeat sides
  forward length
```

```
   turn right angle
  end
end

define update

  clear
  position 260 170

  set length 40
  width 5
  colour draw "orange"
  turn to sun
  call shape

  pen up
   forward length*3
  pen down

  set length 10
  width 1
  colour draw "blue"
  turn to earth
  call shape

  set sun to sun+1
  set earth to earth+5

end


set sides to 6

set sun to 0
set earth to 30
```

Glossary

This example introduces the new concepts of:

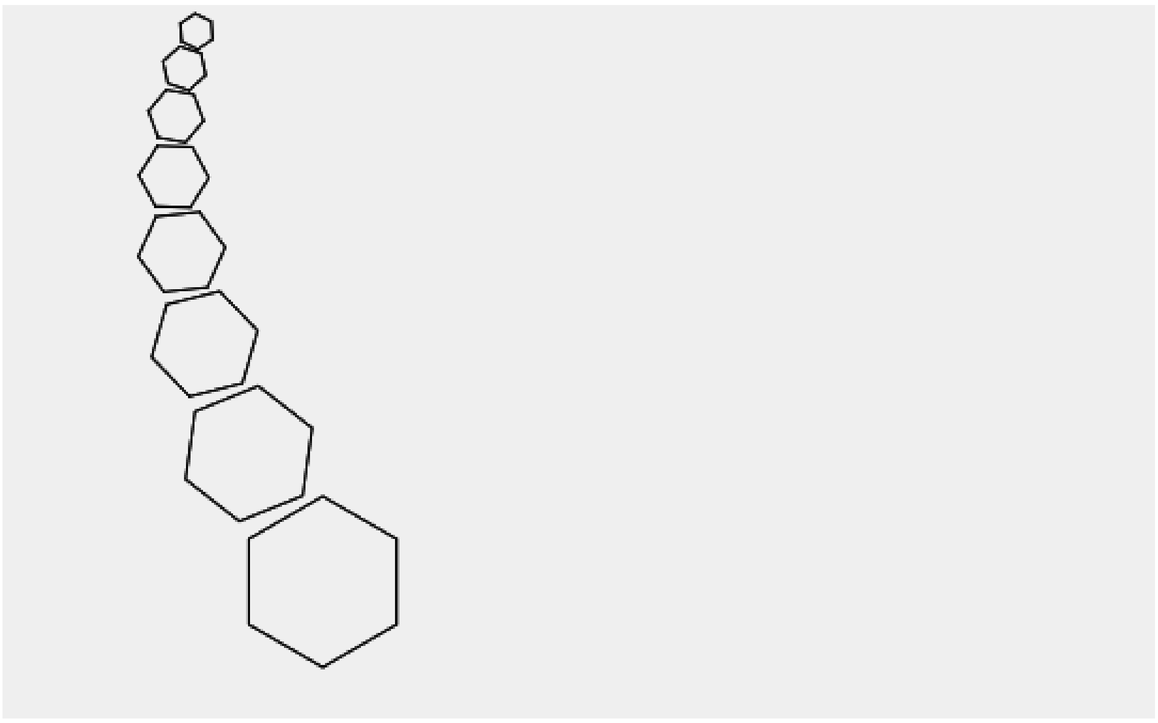
- back

Oct-a-Hex

Category: Apps | ***Level 1***

Play with me!

Task: Try changing the values of the variables to see what you can do! Remember, there is no right or wrong answers here.



Code

```
reset
scale 100

define shape
 set angle to 360/sides

 COMMENT Move from the centre to the edge by
 COMMENT calculating the radius
 pen up
  back length/2
  turn right 90
  COMMENT 1.73 is square root(3) and only
  COMMENT works for hexagons
  back length/2*1.73
  turn left 90
 pen down

 COMMENT Now draw the shape itself
 repeat sides
  forward length
  turn right angle
 end
end


position 150 270

set sides to 6
set length 40


repeat 8

 call shape

 pen up
  forward length*2
 pen down

 set length to length*0.8
 turn right 8

end
```

Glossary

Instruction

This is a command you give to Bolt to tell him, and the program, to do something. It is the first thing on a line of code, and includes things like print and draw.

Concept

This is word that describes one of the underlying principles of coding, like expression or function. Although it's usually an English word, it meaning it slightly different to its real world counterpart as it is a piece of computer jargon.

System variable

A normal variable is an area - with a name - for storing numbers so that they may be used later. As a coder you can create variables to hold the current score or number of lives in a game, for example. A system variable is one created by Bolt, and include things like the position of the mouse, or the number of hours since midnight.

Operator

A maths symbol which performs computations, such as +, - and *.

Function

These are similar to instructions, insomuch as they tell Bolt to do something. The difference is that they produce an answer which you can then use. The most common function is random, to produce a random number between two given values.

B

breakpoint *instruction*

Forces the program into step mode, so you can see what it's doing. You can optionally include an expression that indicates when the program is to stop

Code	Output
`breakpoint`	Stop the program after this line, and get ready to execute the next instruction.
`breakpoint length>100`	Stop the program after this line, but only when the variable length is greater than 100

C

comment *instruction*

A way for the programmer to remember how their code works. Bolt ignores the word comment, and the rest of the text on that line

call *instruction*

Calls a method in any other part of the program. Remember that you must 'define' the method, before you call it, however. And remember to set any variables you plan to use in the method, before you call it.

clear *instruction*

Clear the screen of all existing graphics, so you can start again. This is often called at the start of the update method.

colour *instruction*

Changes the colour of the line, or graphic, drawn. The colour affects all graphics until another colour instruction is given. Bolt understands many colours including: beige, black, blue, brown, chocolate, crimson, cyan, darkblue, darkcyan, darkgray, darkgreen, darkorange, darkred, firebrick, gold, gray, green, indigo, lightblue, lightgrey, lightgreen, orange, pink, purple, red, silver, violet, white, and yellow.

Code	Output
`colour drawing "red"`	Colour all lines (drawn with 'forward' or 'line' to red
`colour text "blue"`	Colour all text (drawn with print) to blue
`colour images "yellow"`	Colour all images and backgrounds (drawn with 'draw') to yellow

canvas *concept*

The area of the screen where your code draws things. It is divided into units called pixels. There are 540 pixels on the width, and 334 pixels on the height. The centre of the canvas is at 270, 167.

D

data *instruction*

Declare some information which may be loaded into variables, via the 'read' instruction. Is it usually numbers or strings.

Code	Output
`data 100 0 200 0 200 50 200 100`	Prepare 8 pieces of data

define *instruction*

Define a method so that it may be called later.

draw *instruction*

Draws a graphic on the screen. If the 'at' word is included the image is drawn at the position specified. If it omitted, then it is drawn at the current cursor position. However, if the image is a background, and 'at' is omitted, then it is drawn in the centre of the screen.

CODE	OUTPUT
`draw image smalldot at 200 10`	Draws a dot at x=200, y=10
`draw image smalldot`	Draws a dot at wherever the cursor position is
`draw background city`	Draws the city to cover the whole screen

E

expression *concept*

A formula to which Bolt calculates the answer. It can include all the standard mathematical operations, along with variables, numbers, and functions

CODE	OUTPUT
`set result 80+10`	Computes 80+10 and assigns it to the variable result
`set result result+1`	Increase the result variable by 1
`set score score-random(10,15)`	Decrease the score by a random value, between 10 and 15

end *instruction*

Defines the end of a block of code that has been started with 'if', 'define', 'repeat', or 'while'. When ending an 'if' block, for example, it is considered good manners to write 'end if' (to remind you that it relates to an 'if' instructions, and not a 'repeat' instruction) but it is not necessary.

F

forward *instruction*

Move the cursor forward by the number of units given, from the current position, in the current direction. The number is converted in pixels, according to the current scale. If the 'pen' is down a line is drawn in the current colour. Note that moving forward a negative number of pixels is identical to going backwards by a positive number.

CODE	OUTPUT
`forward 100`	Move forward 100 units.

false *system variable*

False is 0 (zero). Using false, instead of 0, within logic statements makes the code easier to read.

function *concept*

Functions are like methods, insomuch as they are pieces of code that you can call several times within a program. Functions, however, return a value that you can use in an expression.

H

hours *system variable*

Returns the hour, of the current time. It ranges between 0 and 23

I

if *instruction*

This is a conditional, which allows Bolt to decide whether he should execute the next instruction, or not. The instruction must be partnered with and 'end' instruction to indicate the end of the block. You can also add an 'else' instruction into the block, to indicate what happens in all other cases, when the 'if' part is not true.

CODE	OUTPUT
`if mousedown > 100 draw image smalldot at mousex,mousey end`	Draw a dot, but only when the mouse button is down

L

label *instruction*

Indicate the start of a block of code, or data.

line *instruction*

Draw a line from the previously specified position to the co-ordinates given.

CODE	OUTPUT
`position 100 0 line 100 100`	Draw a vertical line down the screen

M

method *concept*

A piece of code that can be called at any time, to save typing the same code in again. Each method should do one small job, such as drawing a shape, or a game character.

minutes *system variable*

Returns the minute, of the current time. It ranges between 0 and 59

mousepressed *system variable*

Returns 1 if the mouse button was pressed during the last update

mousereleased *system variable*

Returns 1 if the mouse button was released during the last update

mousedown *system variable*

Returns 1 if the mouse button is currently down

mousex *system variable*

Returns the x position of the mouse cursor

mousey *system variable*

Returns the y position of the mouse cursor

N

now_date *system variable*

The current day of the month, from 1 to 31 (if appropriate for that month)

now_day *system variable*

The current day of the week. 0=Sunday, 1=Monday, and so on.

now_dayname *system variable*

The name for the current day of the week. e.g. Sunday, Monday, and so on.

now_month *system variable*

The current month, as a number from 0 (January) to 11 (December)

now_monthname *system variable*

The name of the current month. e.g. January, February, and so on.

now_year *system variable*

The current year, written as 4 digits. e.g. 2015

O

origin *instruction*

When used on its own, the origin instruction moves the cursor back to the origin of the X-Y co-ordinate system. This is usually the centre of the screen. However, when used with the two parameters - X and Y - you can move the origin to be anywhere on the screen.

operator *concept*

An operator is something that produces a result, based on one or more terms. For example, the '+' sign is the addition operator that adds two numbers together. The minus sign can be used with one value, to make it negative, or with two to performance a subtraction operation.

P

program *concept*

A computer program is the code that you write to make it do stuff. It consists of a number of lines which contain instructions. Each instruction executes one at a time. Most programs can be thought of having two parts - a setup part, and an update part. In setup, you prepare everything you need (such as setting the number of lives) for the program. The update is then executed 30 times a second to move and draw the output for your program.

pixel *concept*

Each dot on the screen is called a pixel. It is the smallest thing than can be drawn, as all other shapes and images are drawn as a combination of pixels. A pixel can be any colour, and each each is given a position on-screen as X and Y co-ordinates.

pen *instruction*

When you call 'line' to draw a line, it will only leave a mark on the canvas if the pen is 'down'. The pen is down whenever the program starts, or 'reset' is called. You can lift the pen with 'pen up', or put it down with 'pen down'.

Code	Output
`pen up`	Stop drawing, until further notice.
`pen down`	Draw, until further notice.

print *instruction*

This writes text on the screen, at the position you specify. You can include strings and variables in the list of things to print.

Code	Output
`print at 10 50 "Score:" score`	Writes the score at the top left of the screen.

power (^) *operator*

The power operator is presented by '^', the 'caret' or 'hat'. To calculate the square use x^2, instead of x*x, and x^3 for the cube, for example.

position *instruction*

Set the graphics cursor to a specific position on the canvas. It should include both X and Y co-ordinates, and be between 0 and 539 (inclusive) for X and 0 and 333 (inclusive) for Y.

random *function*

A way of producing random numbers.

CODE	OUTPUT
`set x to random()`	A random number between 0 and 100
`set x to random(10)`	A random number between 0 and 10
`set x to random(4,8)`	A random number between 4 and 8

read *instruction*

Handle the information stored in the 'data'. This can be used to indicate the start position of the data, by referencing a label, or to load the data into variables.

CODE	OUTPUT
`read from square`	
`read into x y`	

reset *instruction*

Clear the screen, and clear all variables from the memory, so the program can begin clear of all old data.

repeat *instruction*

Initiates a loop from the next instruction, until the matching 'end'. You can loop any number of times, as specified by a number or a variable. There is also the 'as' option which writes the loop counter into the variable.

Code	Output
`repeat 4`	Execute the loop 4 times
`repeat 4 as side`	Execute the loop 4 times, storing the loop counter into the variable 'side' each time. It starts with 1, 2, 3, and finally 4.

rem *instruction*

See: "comment"

S

set *instruction*

Assigns a number to a variable, so that the variable can be used in place of the number. This is always better than using a number directly, since it can be changed while the program runs. Variables can also be a list of different items, called an array.

Code	Output
`set position to 80`	Assign the variable called position to the number 80
`set position to random(80)`	Assign the variable called position to a random number between 0 and 80

seconds *system variable*

Returns the seconds, of the current time. It ranges between 0 and 59

screenwidth *system variable*

Returns the width of the working area of your screen, or 540 pixels. It is always best to use variables instead of numbers

screenheight *system variable*

Returns the height of the working area of your screen, or 334 pixels. It is always best to use variables instead of numbers

stop *instruction*

Stops the program from running. This is good when the player has run out of lives, for example.

scale *instruction*

Determines how big everything is drawn on the canvas. The scale is given as a percentage, although you don't need to use the '%' sign. By default everything is 100%, meaning 'forward 20' moves by 20 pixels. If scale was set to 200, then it would move by 40 pixels. Note than only drawing commands are affected, 'turn' commands aren't.

screenshot *instruction*

Opens a brand new window containing an image of the cavnas. Be careful to not include this command inside an update loop, as it will open 100s of windows that you won't be able to close!

system *instruction*

This lets you change the way that Bolt operates! Be very careful with this as it can break everything!

Code	Output
`system warnings off`	Stops the 'warnings' message from appearing when you make a mistake

T

true *system variable*

True is same as the numeric value of '1'. Using true, instead of 1, within logic statements makes the code easier to read

turn *instruction*

Rotate the cursor relative to its left, or its right, or to an absolute angle.

Code	Output
`turn left 90`	Turn 90 degrees to the left
`turn right 45`	Turn 90 degrees to the right
`turn to 180`	Force the cursor to 180 degrees. i.e. pointing left

V

variable *concept*

A variable is an area for storing numbers so that it may be used later. Each variable is given a name so that it can be referenced later. A variable can be used anywhere a number can.