

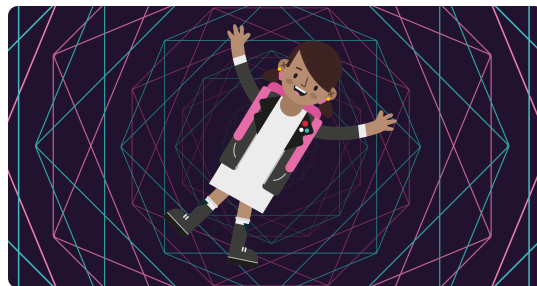


Projects

Intermediate Scratch Sushi Cards

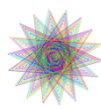
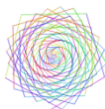
Make a cool pattern generator

Scratch



Step 1 Introduction

You've already learned the basics of Scratch (and if you haven't, check out the Beginner Scratch Sushi Cards) to make your first Scratch game. Here you're going to learn a few more cool tricks and make one of my favourite Scratch projects: it draws colourful patterns and, if you set it up right, can be really cool to watch.



What you will make

Here's a project I've prepared:



What you will learn

In the Intermediate Scratch Sushi Cards, you'll find out about:

- Using the Pen tool
- Using and updating variables in loops
- Using Repeat Until loops
- Getting input values with ask blocks
- Using multiple lists with properties related by index



What you will need

Hardware

- A computer capable of running Scratch 3

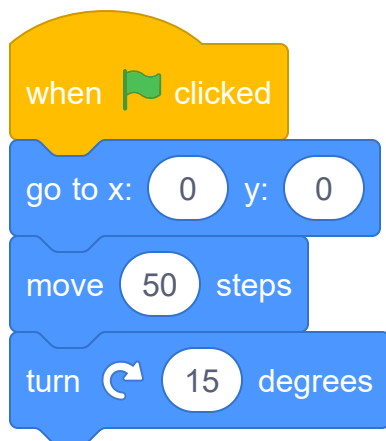
Software

- Scratch 3 (either **online** (<https://scratch.mit.edu/projects/edito>
[r/](https://scratch.mit.edu/projects/edito)) or **offline** (<https://scratch.mit.edu/download/>))

Step 2 Using the Pen tool

The project you're going to make relies on the **Pen** tool, which draws a line behind the centre of a sprite as it moves. You're going to learn to use it now!

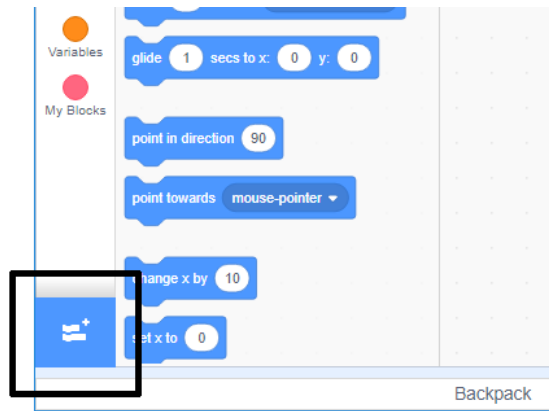
Open a new Scratch file, select the Scratch Cat sprite, and drag in a few blocks you may have already seen, until it looks like this:



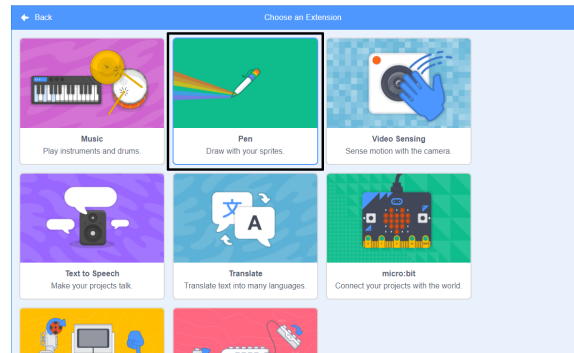
Now, time to test out the pen!

To use the Pen blocks in Scratch, you need add the **Pen extension**.

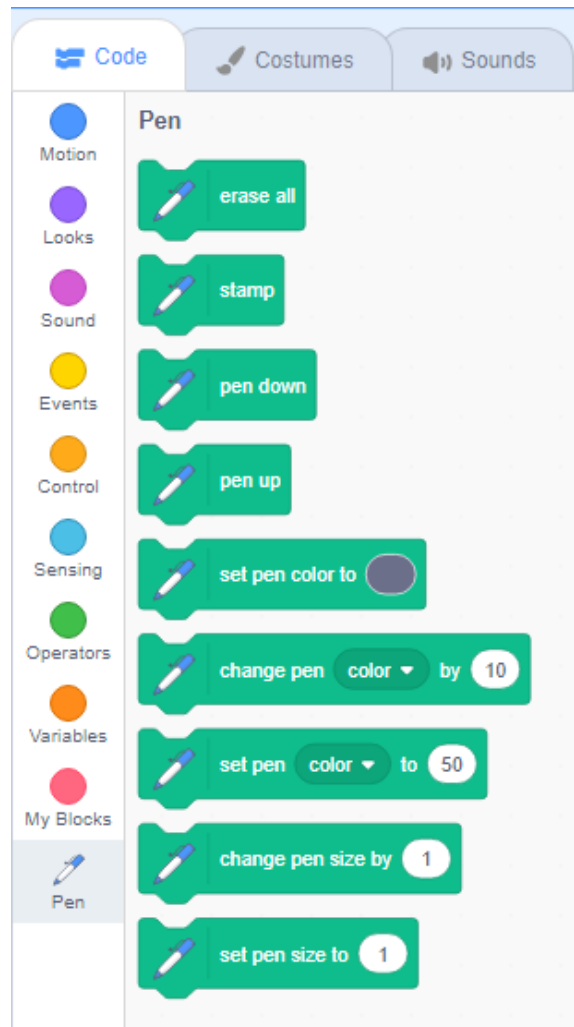
Click on the **Add extension** button in the bottom left-hand corner.



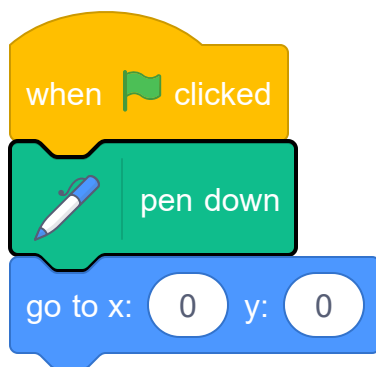
Click on the **Pen** extension to add it.



The Pen section then appears at the bottom of the blocks menu.



From the **Pen** section, select the **pen down** block and add it to the start of your program, like this:



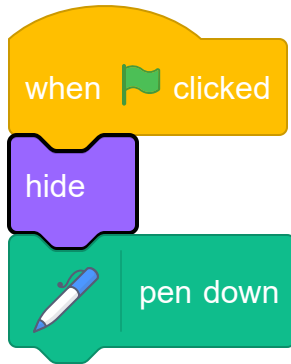
Now click the green flag a few times and watch what happens.



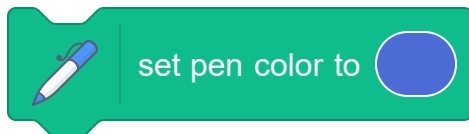
If you can see the lines behind the cat sprite, then the pen is working and you can start making it draw really cool patterns.

First, you should get rid of the sprite. It's getting in the way of the drawing!

Add a **hide** block from **Looks** to the start of the program and it'll disappear.

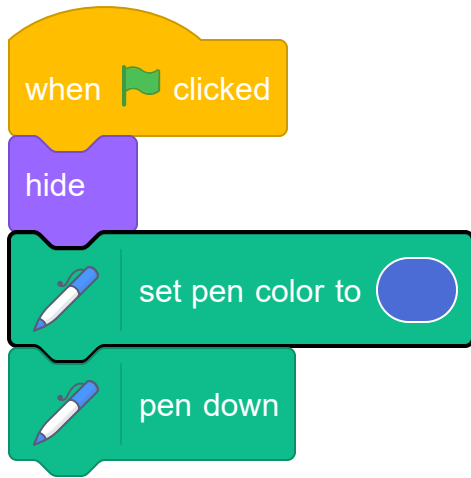


Now, you can change the colour of the pen with another block from the **Pen** section, but the block is a little different to the others you've seen. It's the **set pen color to** block and looks like this:



Drag a **set pen color to** block into your sprite panel, and snap it in above the **pen down** block.

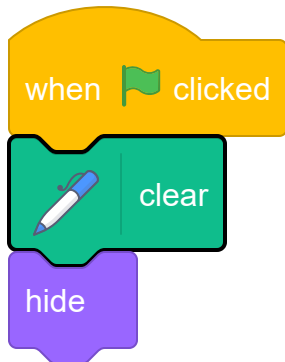




Now, click on the box of colour (in the code above it's the blue one), and choose a colour.

If you've been clicking on the green flag to test your code, you'll have noticed that the drawings the pen makes don't go away.

Add a **clear** block from the **Pen** section to the start of your code to take care of that:

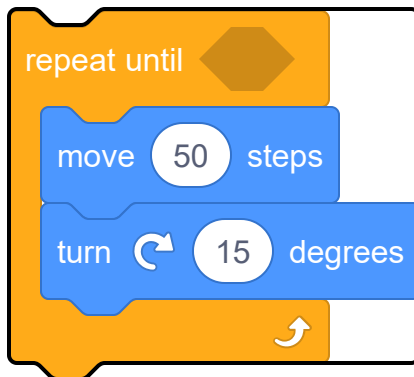


Step 3 Drawing patterns

Now you've got a program that draws a line, but it only draws one line. That's a bit dull! You could use **forever** loop to draw something over and over again, but then you'll get drawings that go off the Stage!

So you need use a different type of loop called **repeat until**, which you'll also find in the **Control** section. This type of loop will do something over and over again, **until** a True/False condition is met.

Take a **repeat until** block from the **Control** section, and put the **move** and **turn** blocks inside it, like so:



Now click the green flag to run the program a few times and see what happens. You'll notice two things: the pen always starts by drawing a line towards the middle of the Stage, and it doesn't stop at the edge.



Why does the pen do this?

The pen always starts drawing in the direction of the middle, because the first **Motion** block that runs after the **pen down** block is **go to x: 0 y: 0**. So the pen will draw a line as it moves to the centre of the Stage.

The pen doesn't stop at the edge of the Stage, because you haven't yet told the **repeat until** loop what condition it's checking. This means the condition can never be met, so the loop will run on and on. This

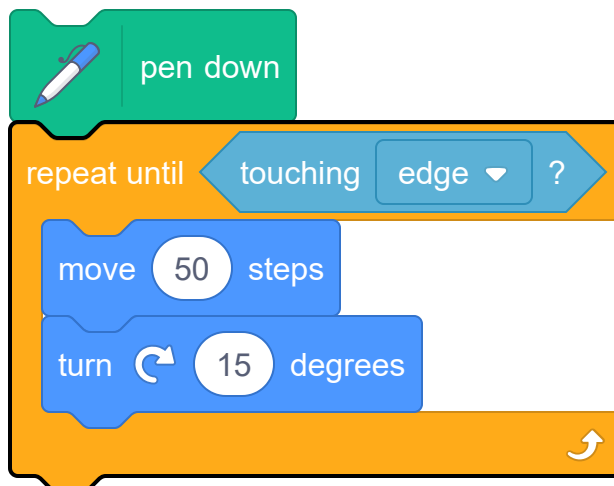
means that right now, the loop is working like a **forever** loop.

Move the **go to x: 0 y: 0** block to before the **pen down** block and add, from the **Pen** section, a **pen up** block right at the start of your code.



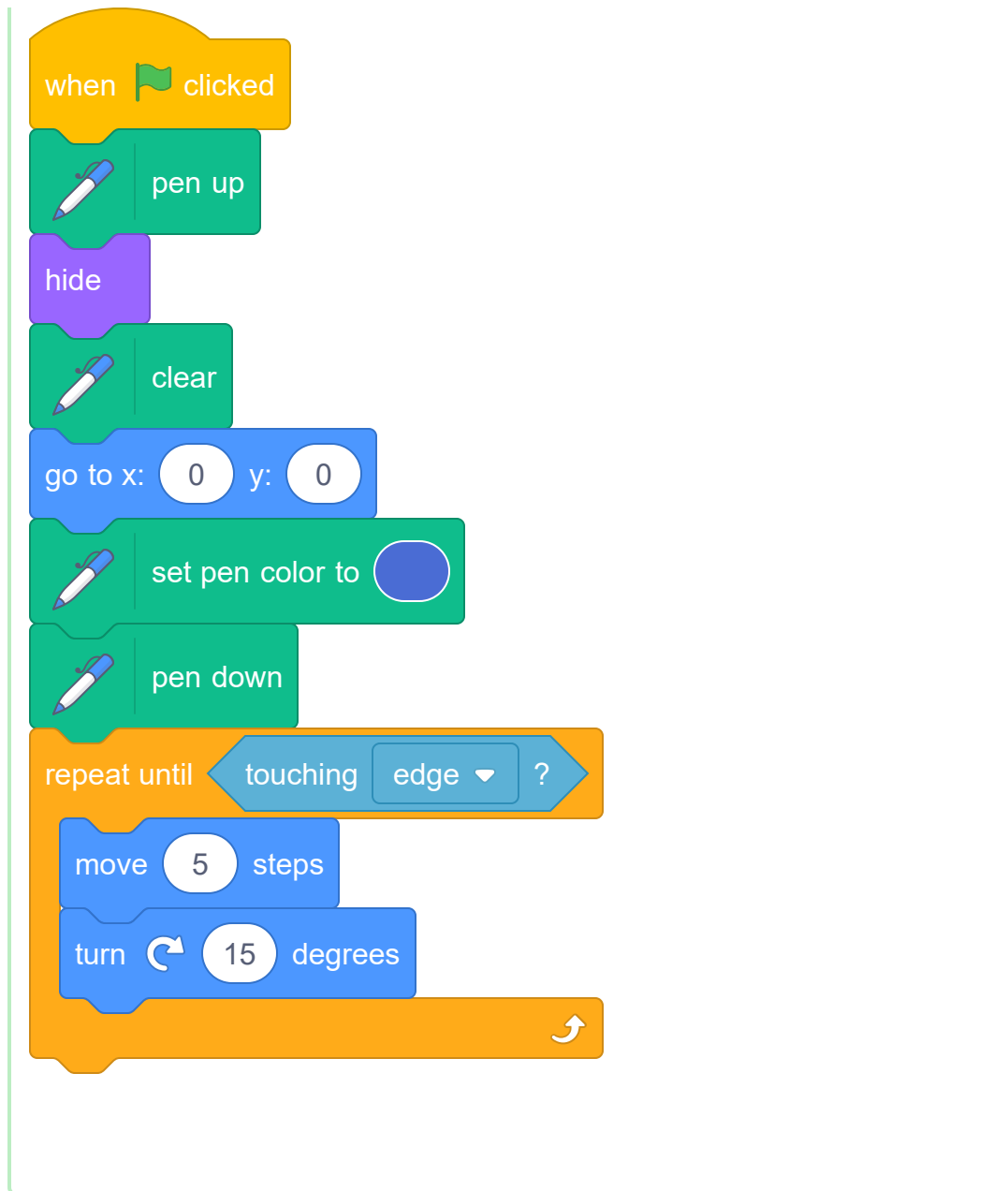
Time to fix your **repeat until** loop so that it stops when you want it to. You're looking to figure out if the (invisible) sprite is touching the edge of the Stage, so you need a **Sensing** block — in this case, the **touching ?** block.

Add a **touching ?** block into your **repeat until** loop, and select **edge**. Then the loop will run **until** the (invisible) sprite touches the edge of the Stage.



Change the number of steps in the **move** block to **5**, and check that your program matches this one before you test it:



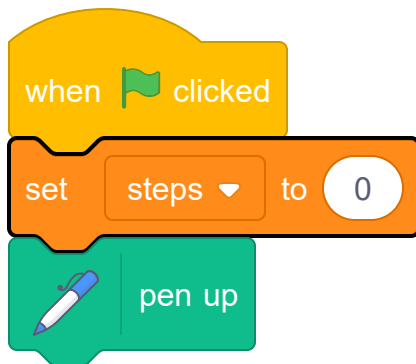


If you run the code now, you'll see that the drawing the pen makes stays on the Stage.

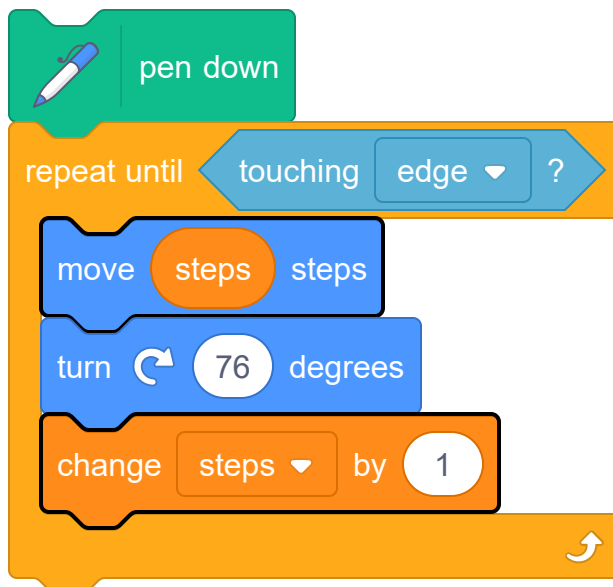
Not only that, but your program has turned into a circle-drawing program! What's happening here is that those 15-degree turns eventually add up to 360 degrees, and so your pen turns in a full circle. You're going to change the way it moves slightly to make it take slightly a longer step each time, so it spirals out. For this, you're going to need a **variable**.

Variables are basically labeled places to store numbers or other information that you care about. You can create them in the **Variables** blocks section.

Make a variable called **steps**, and then add a **set steps to 0** block at the start of your program.



Then use the value of **steps** instead of the 5 in the **move** block, and add **change steps by 1** as part of your loop:



Do you think it matters where in the loop you put the **change steps by 1** block?



Putting code in the right order

When you're deciding which order to put blocks in, think about what each block does and what you want your code to do.

In this case, you want the pen to move, then turn, over and over. Each time it does this, you want to move one extra step.

So it makes sense to put the **change steps by 1** block **after** the **move** block. However, after moving, it doesn't really matter if the pen turns first and then the number of steps changes, or if the number of steps changes first and then the pen turns instead.

Now run the program, and also try changing the number of degrees around (try **76** and **120**)!



Step 4 Asking for input

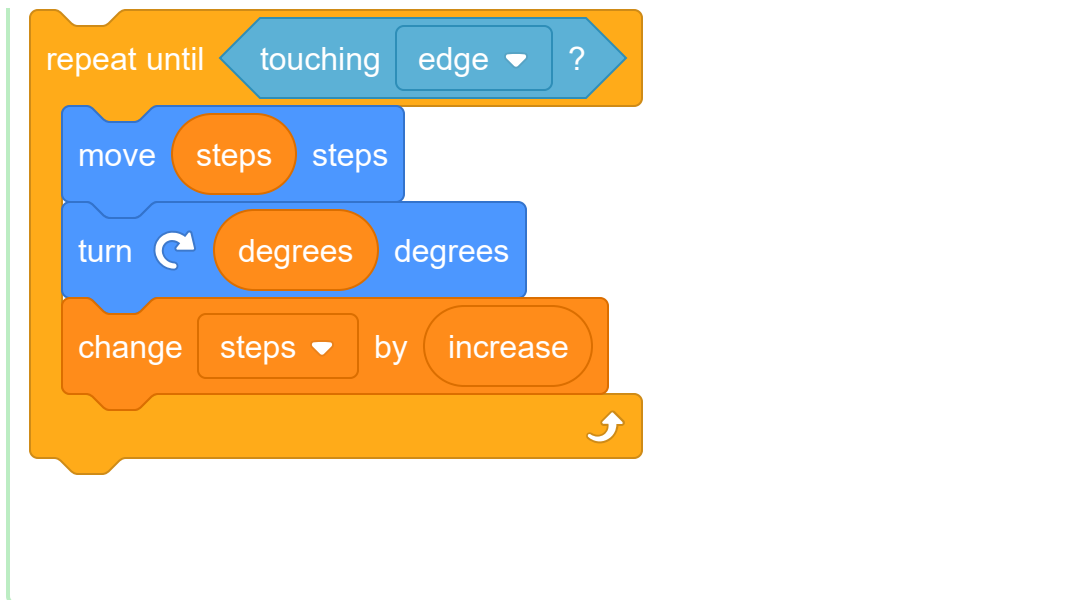
Ok, this is getting pretty cool, but it's a bit boring to have to edit your code every time you want to draw a different pattern. Wouldn't it be good to get the program to ask you for values to use? You can do that!

First, go to the **Variables** section and create variables called **degrees** and **increase**.



Now add the new variables to your code like this:





Now you need to ask for values for these two variables and store them. You do this using a **Sensing** block called `Ask and wait`, which you can type a question into.

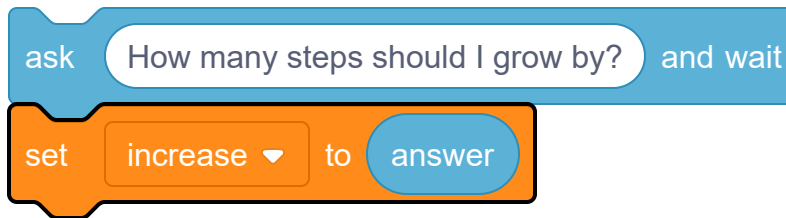
Pull the `Ask and wait` block into your sprite panel and change the question to `How many steps should I grow by?` ☒

Then add it to your program, just after you set `steps` to `0`, like this:

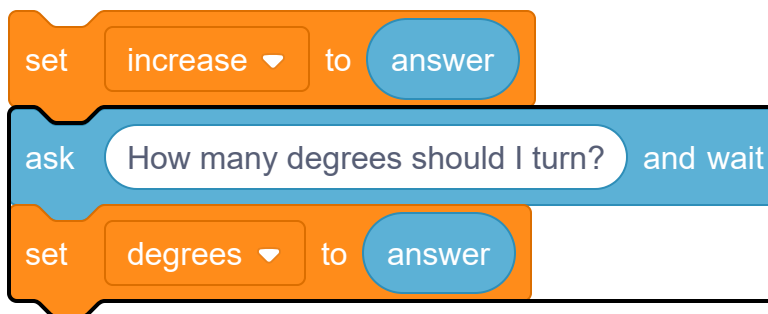
A Scratch code block starting with a 'when green flag clicked' event block. This is followed by a 'set steps to 0' block, then an 'ask How many steps should I grow by? and wait' block, and finally a 'pen up' block.

Now you've got your program asking a question, you need it to remember the answer! It turns out that Scratch has a special variable called `answer`, where it stores the most recent answer it has received. You can find this variable among the **Sensing** blocks.

Using a **set to** block from **Variables**, take the value of **answer** and store it in the **increase** variable like so:

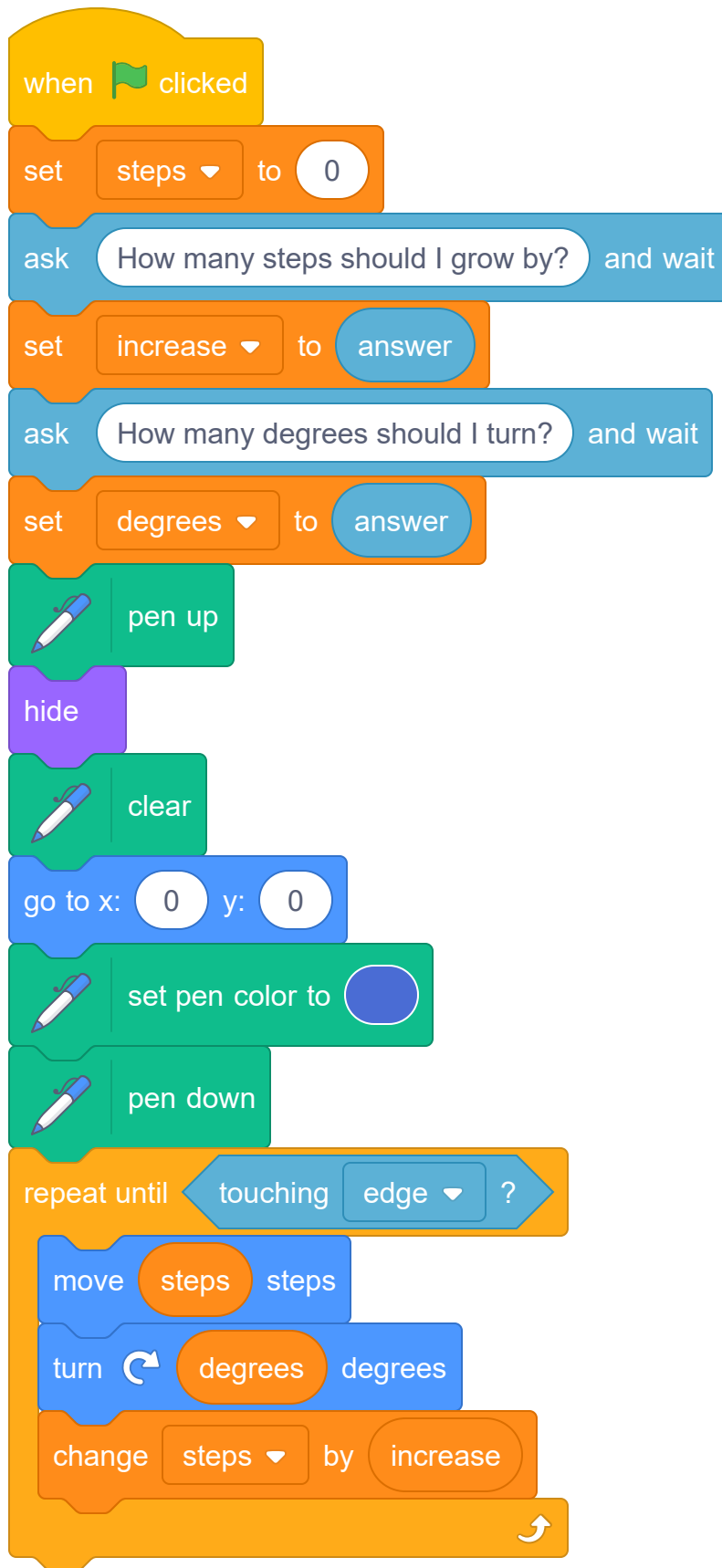


Now, do the same thing with **degrees**, asking **How many degrees should I turn?** and storing the value of **answer** in **degrees**:



Check your program now looks like the one below, and run it a few times with different numbers. Write down the answers that make the coolest pictures. You'll need them in a later step!

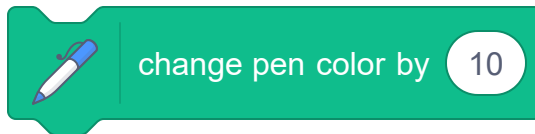




Step 5 Cooler lines

Time to add colour! Right now, your line is one colour, but the **Pen** has blocks that can change its colour. And with the right **Operator** block, you can even change the colour randomly!

The block for changing the **Pen** colour is **change pen color by**:

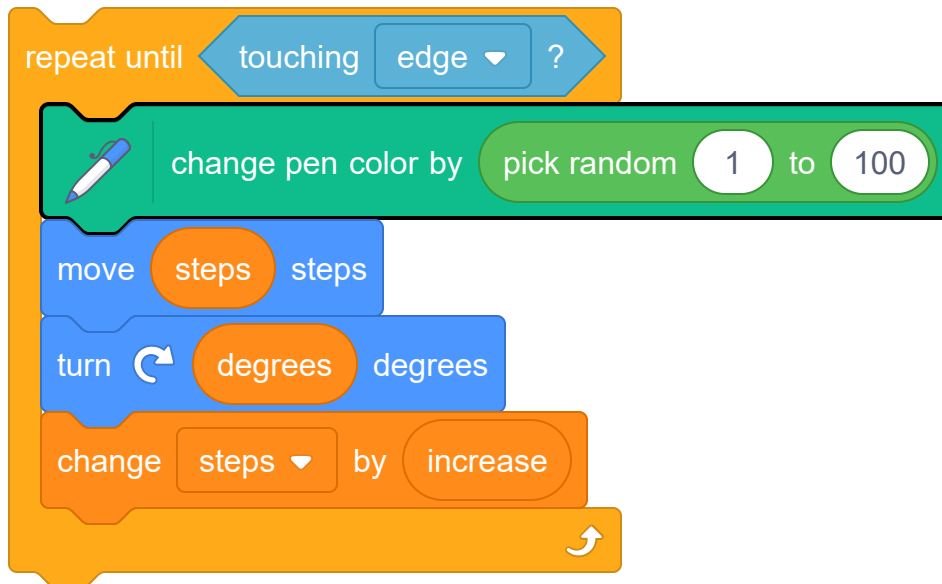


Grab one of those blocks and put it into your **repeat until** loop, like this:



That's cool, but a bit predictable. You can make it a bit more fun if you add a random number into it, so the colour changes randomly.

Put the random number **Operator** block into the **change pen color by** block and pick some values to go in it. I'd try **1** and **100** to start.




Try running it again, and watch the random rainbow!

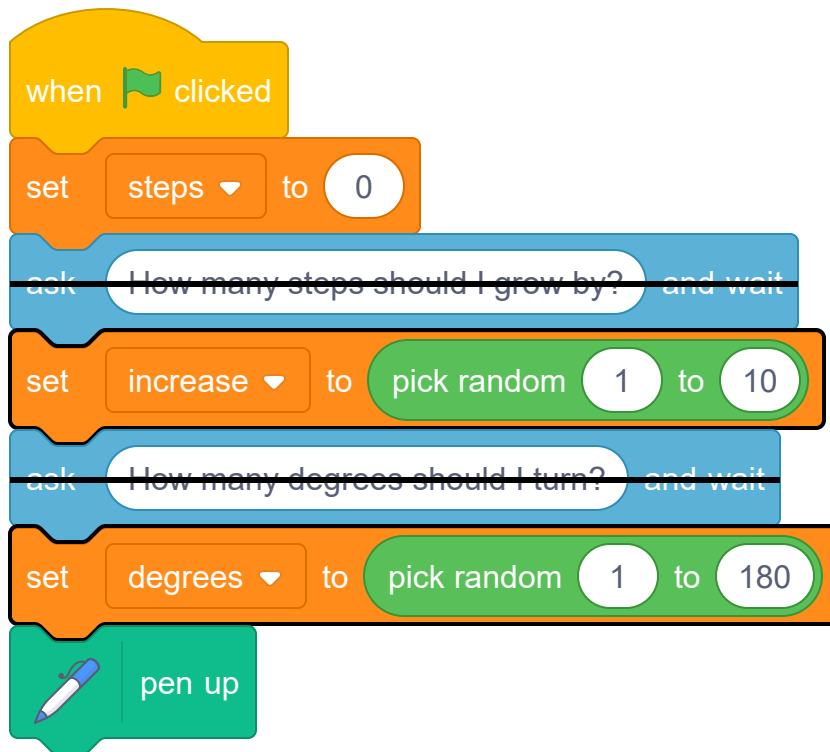


Step 6 Randomise the whole thing

You can actually use random numbers to make the whole program run over and over, changing the pattern each time! It'll look a bit like screen savers did in the 1990s...which you probably won't remember, but ask one of your parents!


You need a few changes to make this happen. The first one is that you need to set the **increase** and **degrees** variables randomly rather than asking for them from the user. So you need to change some of your code blocks.

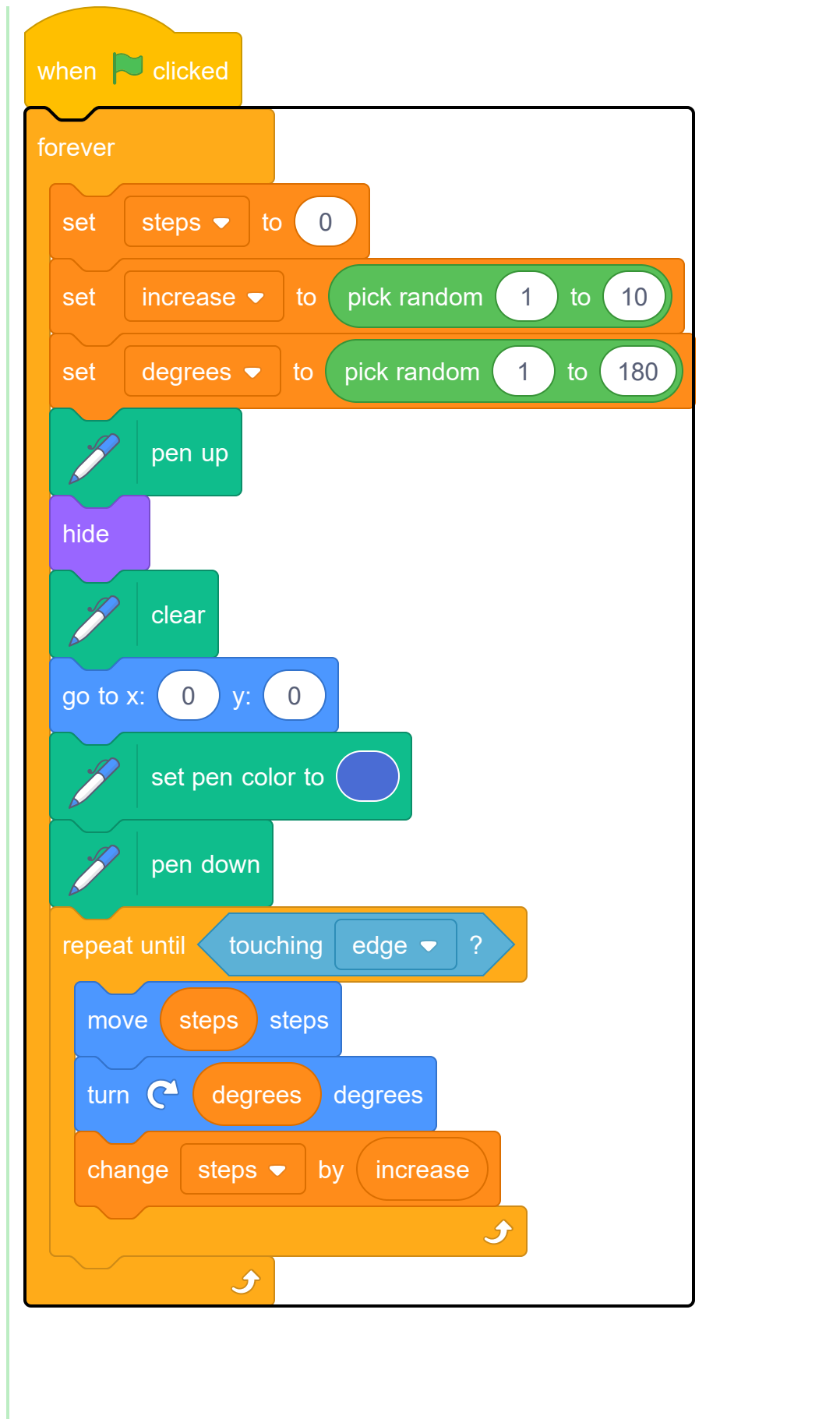
Remove the questions from your code, and update it to use random numbers instead. 



If you run your program now, you'll find that it does draw a random pattern, but only once. Why do you think that is?

It's because the loop only runs until it reaches the edge of the Stage.

You need another loop that runs forever (so a **forever** block then!) outside the current one to keep it going over and over. Just drag one out of the **Control** section, and add all your other code into it. 



Now you've really got something awesome to look at!

However, you may notice that, every now and then, the computer draws something that looks pretty...bad. This is because some numbers for some of those variables are just bad choices, and some **combinations of those numbers** are also bad choices.

On the next card, you'll help the computer to pick only good combinations!

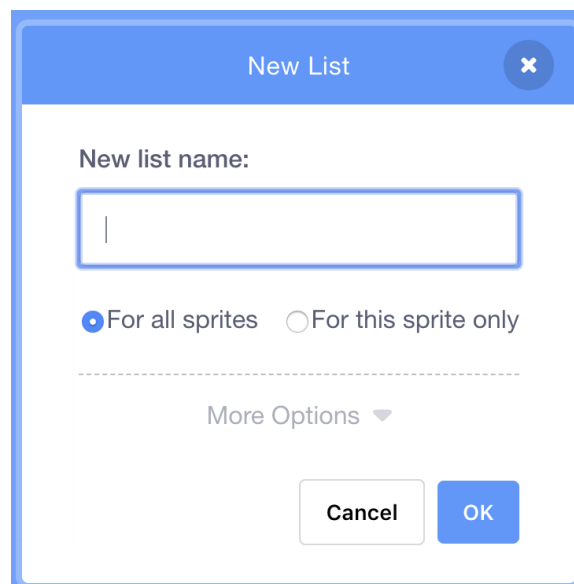
Step 7 Helping the computer

Do you remember a few steps back, where I told you to write down some of your favourite values for **increase** and **degrees**, the ones that gave the best-looking patterns? If you didn't do this, don't worry: you can just watch the random program run for a while now and write down the combinations that give great results.

You're going to teach Scratch those combinations of values, so it can use them to make nothing but awesome pictures!

To do this, you'll need a **list**. You'll find lists with the variables in the **Variables** section. Just like you did with your variables, you'll need to create your list first!

Click **Make a List**, and enter **Degrees List** as the name.

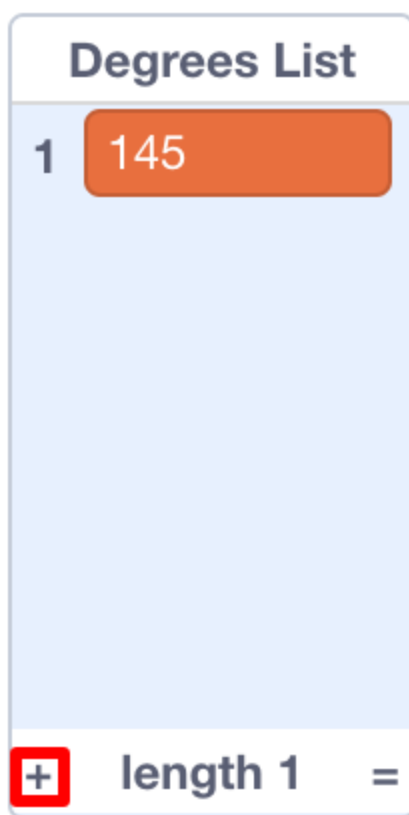


Your list, which is empty at the moment, will appear on the Stage, and you'll see a bunch of blocks for it in **Variables**.

Make another list called **Increase List**



Now, by clicking on the little plus sign (+) at the bottom of the lists, add in the first pair of values of **increase** and **degrees** you liked, each value into the right list. Do this again to add the second pair of values. This will be enough for now – you'll add the rest of the value pairs you like later!



Make sure that the **degrees** value and the **increase** value that worked well together are at the same position in the **Degrees List** and the **Increase List**. They need to be there so your program can match them up again using their position!

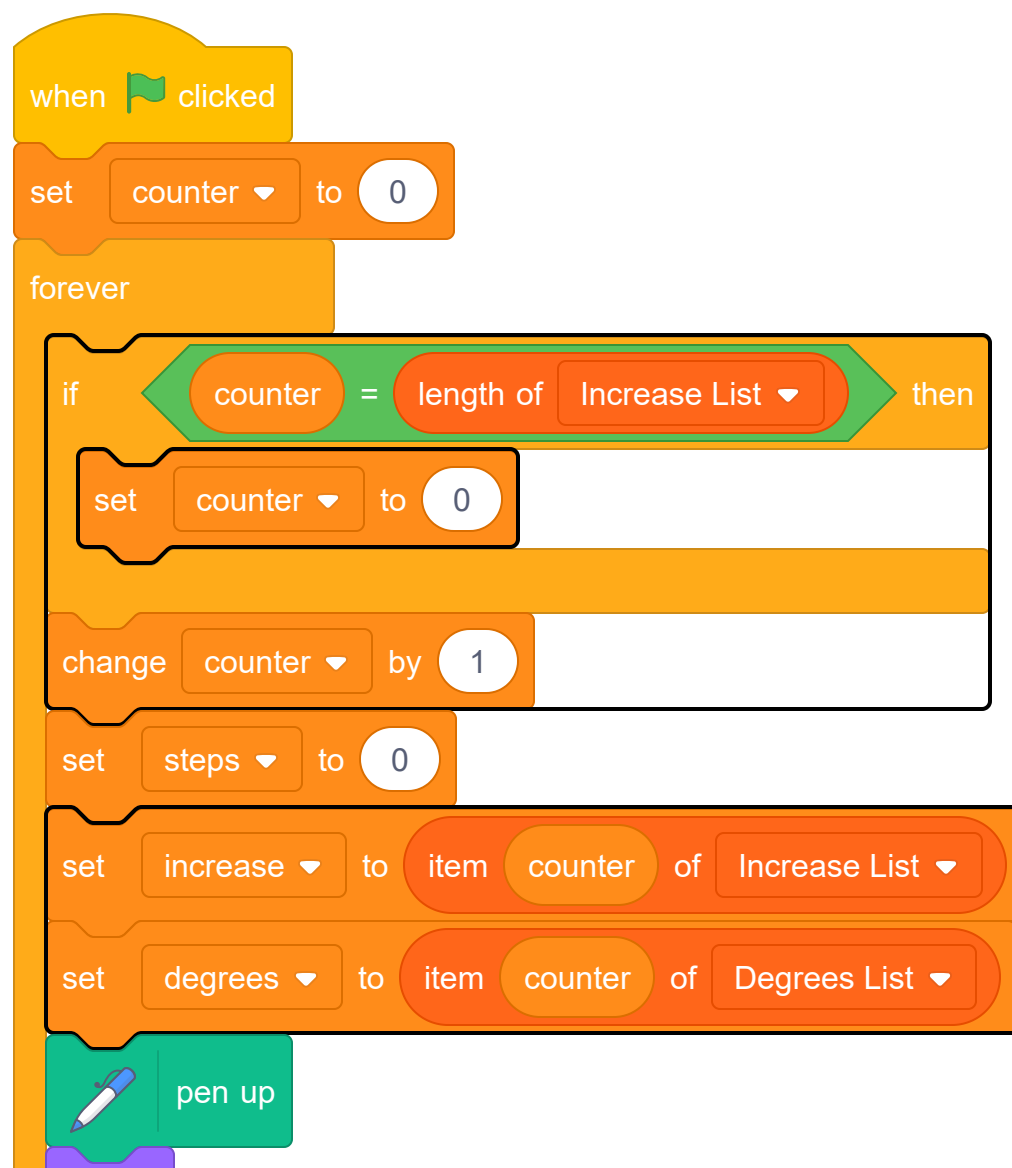
Now you have the lists, you just need to get your code to read them and loop over them! To do this, you're going to use a new variable to act as a counter, some **incrementing**, and an **if then Control** block.

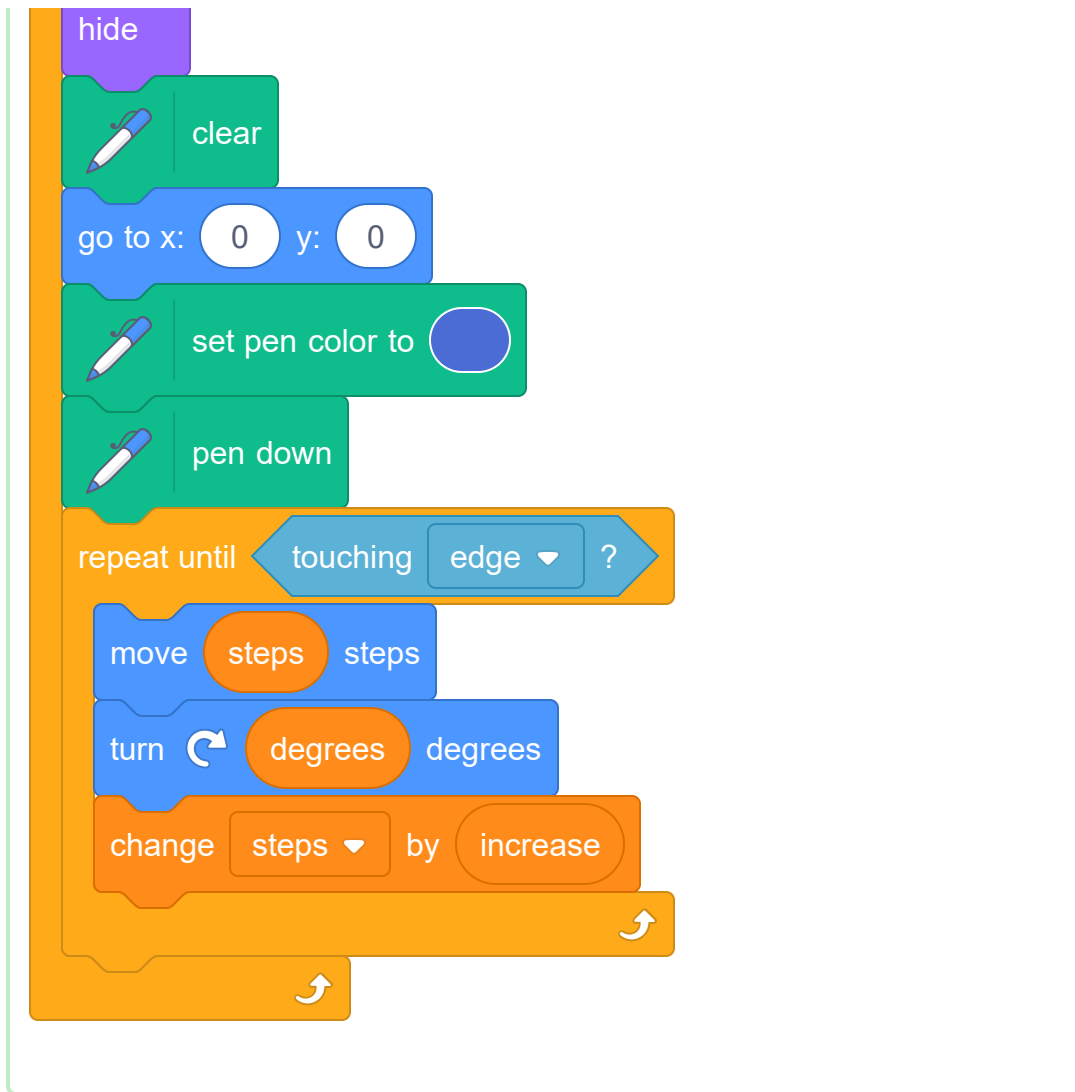
What does incrementing mean?

To increment something means to add something to it.

You will use a variable to act as a counter to keep track of what position you're at in your lists. To move through the lists, you'll keep incrementing the counter by **1** (so, adding **1** to it) until you get to the end of the list.

Create a new variable called **counter**, and update your code to look like this:





Notice the new blocks that:

1. Set **counter** to 0, outside all the loops.
2. Check if the number stored in **counter** is the length of the list, and if so, set **counter** to 0. This means that this variable will always be the number of a position in the lists, and won't get any bigger than that.
3. Add 1 to **counter**.
4. Pick the item from **Increase List** that is at the position described by **counter**, and put it in the **increase** variable. Do the same for the **Degrees List** and **degrees** variable.



How does the code work?

This is what happens when you run your program:

1. Set **counter** to 0.

2. Start the **forever** loop.
3. Check if **counter** (0) is the same as the length of **Increase List** (2). It isn't.
4. Change **counter** by 1. Now **counter** = 1.
5. Set **steps** to 0.
6. Get the item at the position named by **counter** (1) in the **Increase List**, and put it in **increase**.
7. Get the item at the position named by **counter** (1) in the **Degrees List**, and put it in **degrees**.
8. Do all the stuff related to drawing the patterns.
9. Restart the **forever** loop:
10. Check if **counter** (1) is the same as the length of **Increase List** (2). It isn't.
11. Change **counter** by 1. Now **counter** = 2.
12. Set **steps** to 0.
13. Get the item at the position named by **counter** (2) in the **Increase List**, and put it in **increase**.
14. Get the item at the position named by **counter** (2) in the **Degrees List**, and put it in **degrees**.
15. Do all the stuff related to drawing the patterns.
16. Restart the **forever** loop:
17. Check if **counter** (2) is the same as the length of the **Increase List** (2). It is!
18. Set **counter** to 0.
19. Continue from **step 4** of this list, in a never-ending loop!

Once you're happy with the code, go ahead and add the rest of the pairs of values you noted down to the **Degrees List** and the **Increase List**.



That's it! Sit back and watch your program keep drawing lovely patterns in a never-ending loop! If you want to add more patterns, you can: just add more pairs of numbers to the two lists and restart the program.

Published by **Raspberry Pi Foundation** (<https://www.raspberrypi.org>) under a **Creative Commons** license (<https://creativecommons.org/licenses/by-sa/4.0/>).

View project & license on GitHub (<https://github.com/RaspberryPiLearning/cd-intermediate-scratch-sushi>)