

Edison and the spiralling spider trap

Some spiders build webs which spiral into a central point in the middle. Can you program Edison to drive so that the robot spirals inward, like a spider laying a trap?

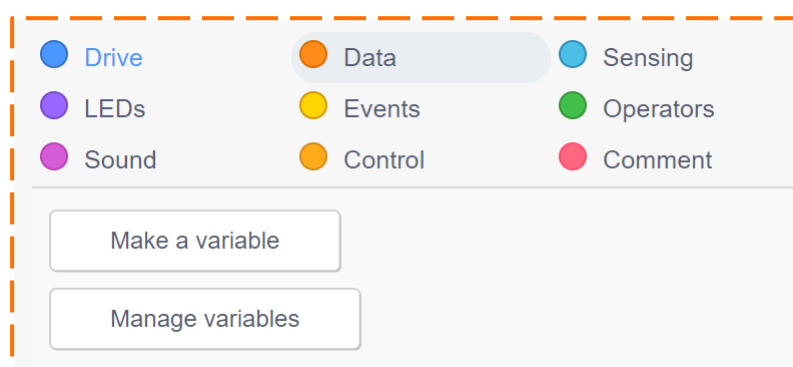
To program Edison to lay a spiralling spider trap, there are a few things we need to learn:

- Part 1: What are variables?
- Part 2: How do you program a spiralling spider trap?

Is this your first time using Edison robots or EdScratch? Start with the activity *Let's get started with Edison and EdScratch* first! Ask your teacher for a copy.

Part 1: What are variables?

In EdScratch, there is one category of blocks that looks very different to the other categories when you first open it: the **Data** category.



When you first click on the **Data** category, there are no blocks available for you to use in an EdScratch program. Instead, there are two buttons: the **Make a variable** button and the **Manage variables** button. By using the **Data** category, we can create **variables** to use in our EdScratch programs.



Jargon buster

A **variable** is a bit of memory that is used to store a value in a program. You can think of a variable like a container that you can use to store some bit of information in a way that will make sense to a computer.

In computer programming, we often use the same bit of information multiple times in a single program. Variables make it a lot easier to do this. Using variables in programs lets us tell the computer to store a specific bit of information inside a variable. We can then use that variable in

different places in the program. Any time the computer sees the variable, it will recall whatever information is stored inside the variable.

In EdScratch, when you click the **Make a variable** button, a pop-up window will open up asking you to give your variable a name. Giving variables good names is important: you want the name of your variable to make sense to you and tell you what bit of information is stored inside.

Your EdScratch variable names can only contain lowercase English letters, uppercase English letters, numbers, and underscores (_). Other symbols, like exclamation marks (!) or spaces, are not allowed.



Why is that?

Variable names need to make sense to you and to the computer. Computer languages often have rules about what characters can be used in variable names. The computer can only understand variables with names that follow these rules.

Give your variable a name that will help you identify what type of information is going to be stored inside the variable. If you want to change the name of a variable after you make it, you can do that using the **Manage variables** button.

Once you make and name a variable, it will appear in the **Data** category along with some other special blocks including the **set** block, the **increment** block and the **decrement** block. You can then use these blocks along with your variable in an EdScratch program.



Jargon buster

In computer programming, **increment** means to increase by 1 and **decrement** means to decrease by 1.

Trace the code

In EdScratch, we can use variables to store different values. Because these values are numbers, we can then do different things with these numbers, like compare them to other values using operators or do computations with them.

Look at the following program:

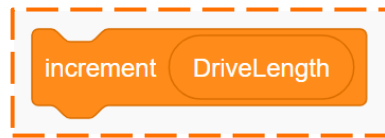
```

Start
set DriveLength to 1
repeat 10
  set both motors to drive forwards at speed 5
  wait DriveLength * 200 milliseconds
  increment DriveLength
  spin right for 90 degrees at speed 5
  
```

This program uses a variable named **DriveLength**.

It is important to note that a variable represents a value that is set somewhere in your program. That's why you always need to use code in your program to tell the computer what to set the variable to be.

This is what the **set** block in this program is doing. At the beginning of the program, the **set** block sets the variable **DriveLength** to be 1. The value of **DriveLength** isn't going to stay set to 1 forever, however. That's because this program uses another block from the **Data** category, the **increment** block:



The **increment** block is inside the **repeat** loop and changes the variable **DriveLength** to a new value. What is this block changing the variable **DriveLength** to be?

That will depend on where in the program you are up to. In other words, it will depend on how many times the **repeat** loop has run.



Why is that?

One of the main reasons we use variables in programs is that a variable can hold a piece of information, even when the value of that information changes. This might sound really confusing but think about it like this:

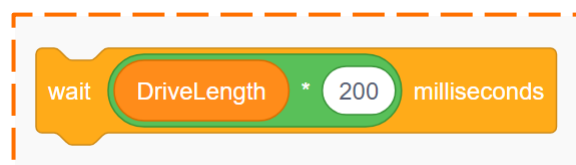
Let's pretend you have a variable called **YearsOld** and the bit of information that this variable holds is your age. At your first birthday, **YearsOld** was set to 1. After that, each year on your birthday, **YearsOld** is **incremented**, which changes it to a new value: (**YearsOld** + 1).

A year after your first birthday, **YearsOld** is incremented. Because the value of **YearsOld** was 1, and since $1 + 1 = 2$, the new value of **YearsOld** became 2. This repeats again on your next birthday, where the value of **YearsOld** is incremented and becomes 3.

The value of **YearsOld** changes each birthday, but the type of information doesn't change. **YearsOld** is your age, no matter how many birthdays you have!

When our program first starts, **DriveLength** will be set to 1, but because there is an **increment** block inside the looped code, the value will change each loop.

DriveLength is also being used in another place inside the looped code in this program:



The input value of this **wait** block will depend on the value of **DriveLength** and will also change each loop.

Be careful though!

While the value of the **wait** block will change depending on the value of **DriveLength**, this **wait** block will not change the value of **DriveLength**. Only a block from the **Data** category can change the value of a variable.

You might also notice that this **wait** block is set to milliseconds, not seconds.



Why is that?

Whenever you want to use a variable or a block from the **Operators** category as an input in a **wait** block in EdScratch, you need to use this special **millisecond wait** block. Edison actually 'thinks' in milliseconds, not seconds, so using this block makes it possible to do computations which Edison will be able to understand.

Remember, 1 second = 1000 milliseconds.

Let's trace the program to work out what the value of the variable **DriveLength** and the input value of the **wait** block are going to be at different points when the code in this program runs.



Jargon buster

Tracing code means working through a program line by line, recording important values.

Trace through the program to work out the values. Fill out the table with what the starting value of variable **DriveLength** will be at the beginning of each loop repetition, what the input value of the **wait** block will be in that loop, and what the new value of **DriveLength** will be after the **increment** block inside the loop runs. The first two rows have already been filled in for you.

| In loop # | Starting value of DriveLength | Wait block input value (in milliseconds) | New value of DriveLength |
|-----------|-------------------------------|--|--------------------------|
| 1 | 1 | 200 | 2 |
| 2 | 2 | 400 | 3 |
| 3 | | | |
| 5 | | | |
| 7 | | | |
| 10 | | | |

Write and run the program

What is the program going to make the robot do when you run it in Edison?

Write the program in EdScratch. Download it and run it in your Edison robot to see what it does.

Part 2: How do you program a spiralling spider trap?

Now it is your turn! Using what you have learned about variables, can you write a program in EdScratch that will get Edison to drive in an inwards-spiral shape? Edison should drive, then turn, then repeat over and over, spiralling inwards, just like a spider laying a trap.

Your program should use a variable to help you control how far Edison drives each time.

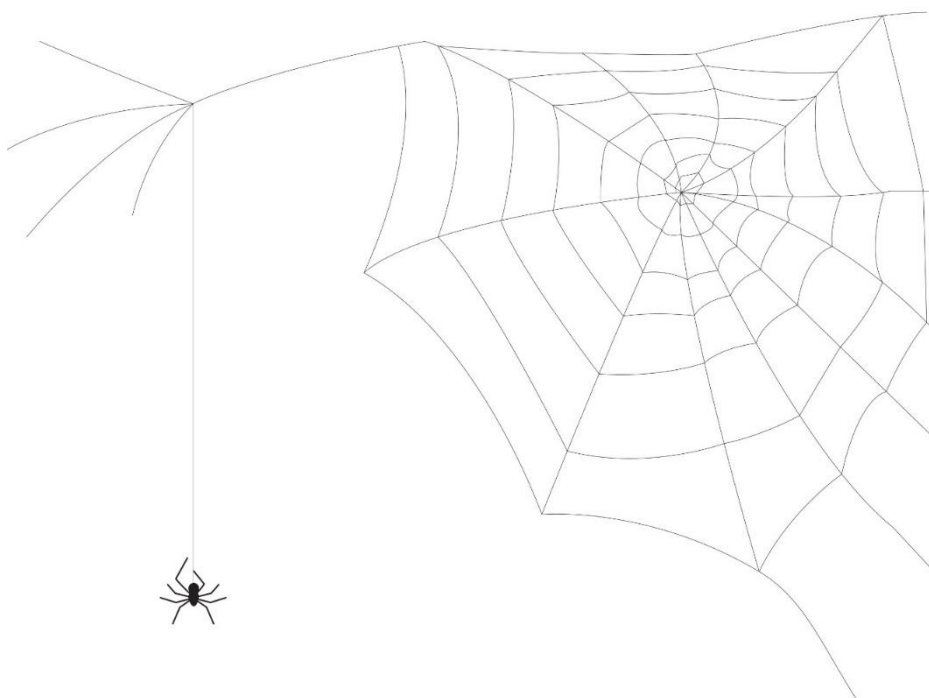
Think about how far you want Edison to drive each section compared to the previous section. How many sections will Edison need to drive in total? You will also need to decide how far Edison should turn between each driving section.

Download your program and test it using your Edison robot. Experiment using different input values to see what works best.



Hint!

You might want to look at the program from part 3 for some inspiration to help you write your program.



Bonus challenge!

Can you make Edison leave a 'sticky spider thread' trail behind the robot as it drives? Use something, like some string, to represent a spider's silk. See if you can engineer a solution so that Edison leaves a trail behind it as it drives.