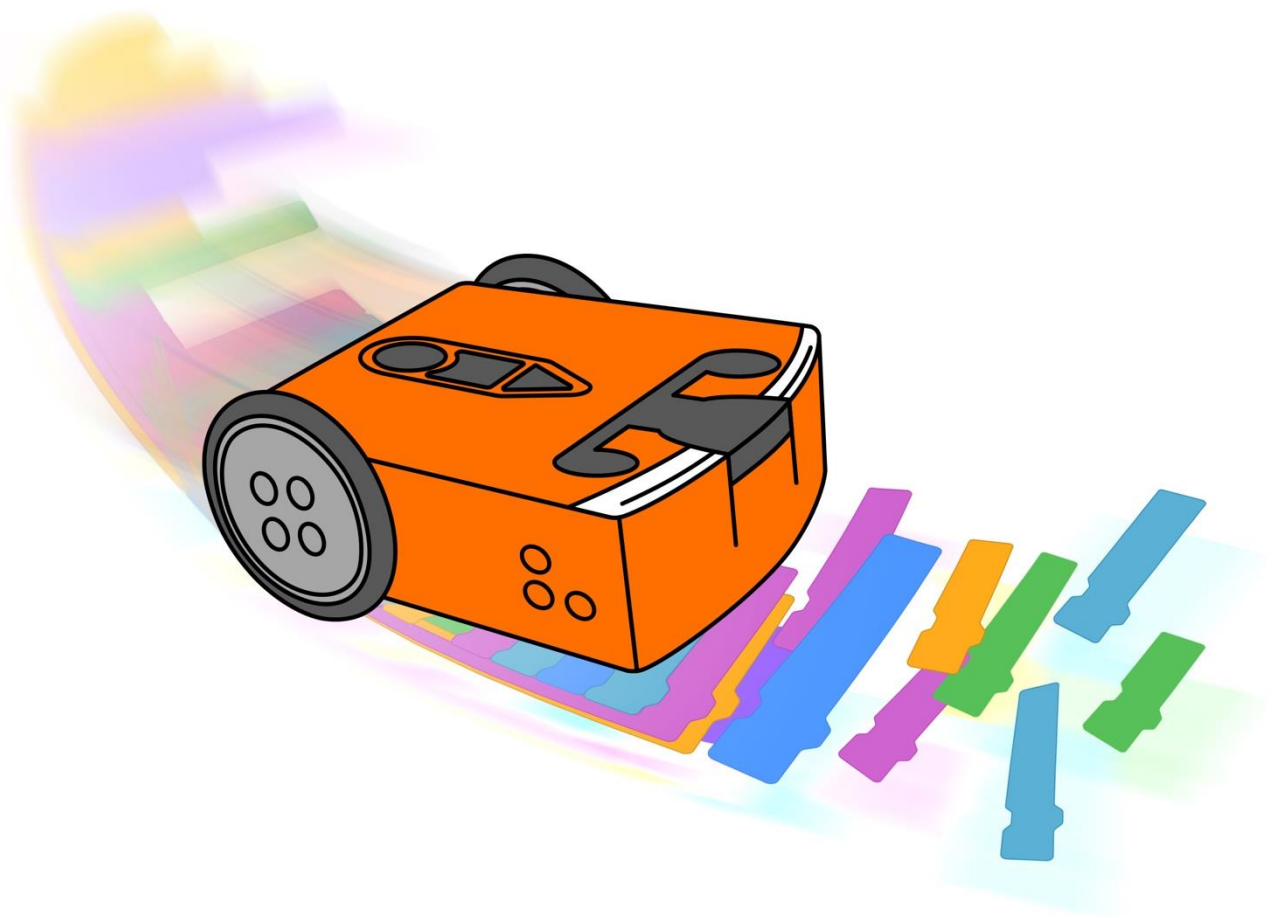




EdScratch lesson activities

Student worksheets and activity sheets

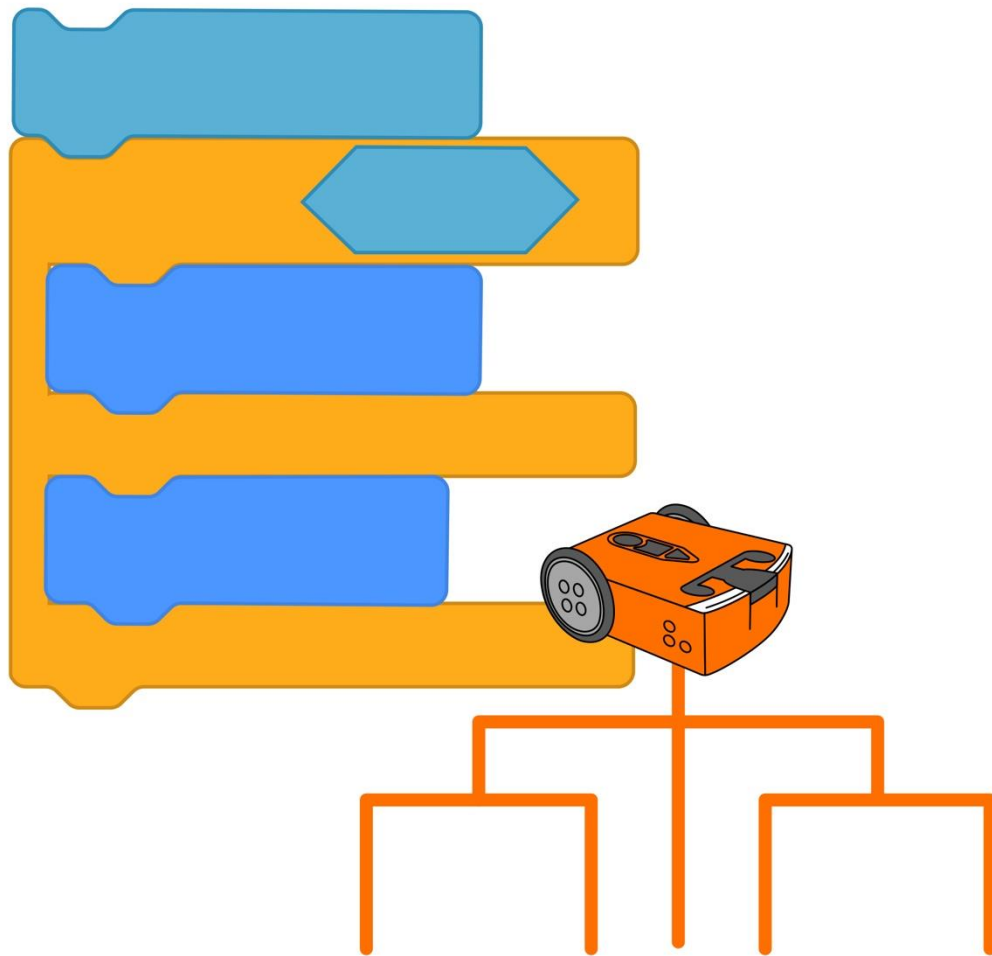


The EdScratch Lesson Plans Set by [Kat Kennewell](#) and [Jin Peng](#) is licensed under a [Creative Commons Attribution-ShareAlike 4.0 International License](#).

Contents

Unit 4: What if.....	3
U4-1.1 Let's explore using conditionals.....	4
U4-1.1a Change it up: Robot error or human error?	8
U4-1.2 Let's explore if statements.....	12
U4-1.3 Let's explore if statements and sequence	15
U4-1.4 Let's explore stacking and nesting if statements.....	18
U4-1.4a Challenge up: Build a pulley.....	22
U4-2.1 Let's explore pseudocode	23
U4-2.1a Change it up: Find the answer	26
U4-2.2 Let's explore Edison's line tracker	27
U4-2.2a Change it up: Drive inside a border	30
U4-2.3 Let's explore algorithms	31
U4-2.3a Challenge up: There's more than one way to follow a line	34
U4-2.4 Let's explore Edison's obstacle detection	36
U4-2.4a Change it up: Faster, faster, smash?	39
U4-2.4b Challenge up: If line, go right. If obstacle, go left	40
U4-2.4c Change it up: Where is the obstacle?.....	41
U4-2.4d Challenge up: 3D maze	42
U4-2.5 Let's explore messaging with Edison.....	43
U4-2.5a Change it up: Remote-controlled flag machine.....	46
U4-2.5b Challenge up: Build and control the EdCrane	48
U4-2.5c Challenge up: Firefighting water cannon.....	50
U4-2.5d Challenge up: Semi-automated digger.....	52
U4-2.5e Challenge up: Hazardous material removal.....	54
U4-2.5f Challenge up: Homing pigeons.....	56
Activity sheet U4-1: If-else maze	57
Activity sheet U4-2: Pseudocode step-by-step	58
Activity sheet U4-3: Line tracker test zone.....	59
Activity sheet U4-4: Non-reflective border.....	60
Activity sheet U4-5: If line, go right	61
Activity sheet U4-6: Programmable TV remote codes.....	62

Unit 4: What if...



U4-1.1 Let's explore using conditionals

To get a computer, like an Edison robot, to do what you want, you have to give it very specific instructions in the form of a computer program. The computer then follows your code step by step. It does whatever you told it to do.

What if you want the computer to make a decision on its own?

Most computers, including your Edison robot, cannot decide complicated things the way a person can, but you can get Edison robots to make simple decisions. The robot still needs you to give it exacting instructions to follow so that it knows what decision it is making and the rules, or **conditions**, for making that decision. To write this sort of program you need to use a type of coding structure known as a **conditional**.



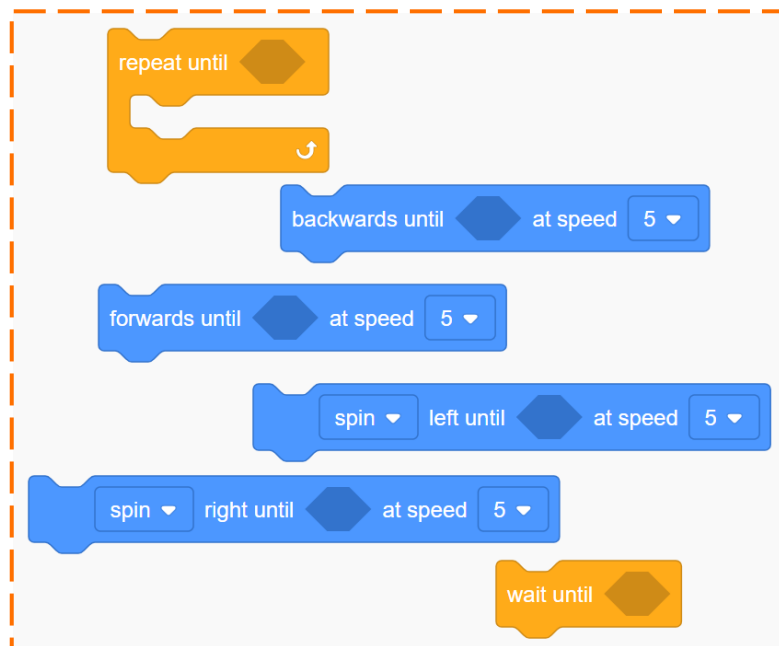
Jargon buster

A **conditional**, which is sometimes called a **conditional statement**, is an element of code that is dependent on something else. This bit of code will only happen if its **condition** is met.

In code, a **condition** is a predetermined circumstance or set of factors that need to be met in order for conditional code to run.

One way you can use conditionals with Edison in EdScratch is by writing a program telling the robot to do something **until** a condition has been met.

Look at the EdScratch blocks in this picture:



All of these blocks are conditionals that use the same **until condition** formula. Each block tells Edison to do an action until a specific condition is met. But what is the condition?

Look at the **until** blocks in the picture again. Do you see the diamond-shaped hole in each block? You give a condition to an **until** block by putting a special input parameter in that hole.



Don't forget

There are three styles of input parameters in EdScratch:

- numbers you type into a block using your keypad,
- drop-down menus where you choose an option from inside the block, and
- round or diamond-shaped holes which you fill with special blocks.

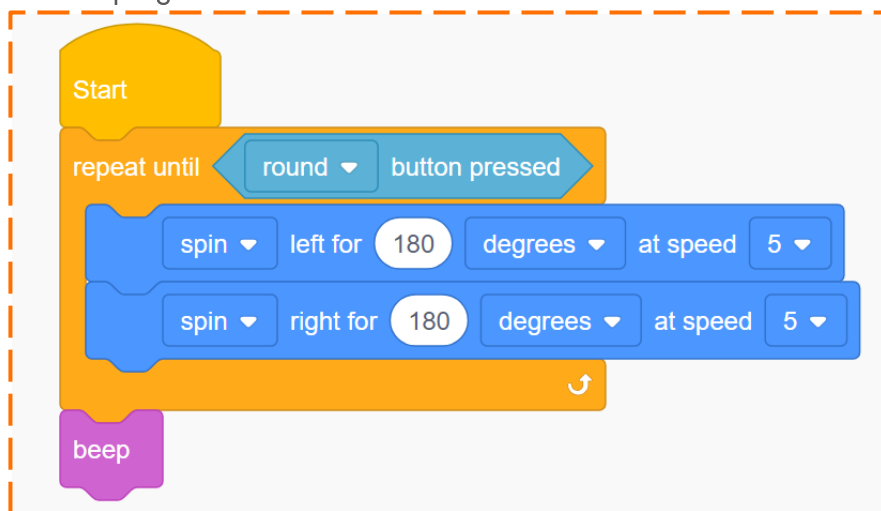
Each input parameter in a block gives a different piece of information to Edison that the robot will need in order to run that command. You can think of input parameters as the answers to questions the robot has about what you are asking it to do.

Just like any other code block, conditionals need all of their input parameters filled in to work correctly. When you use any of the **until** blocks in EdScratch, you need to give the robot the condition by using a diamond-shaped input parameter.

1. Open up the EdScratch programming environment and look at the different blocks. Which block categories contain blocks that you think you could use to give a condition input parameter to one of the **until** blocks? Why do you think that?

Task 1: Repeat until...

Look at this EdScratch program:



This program uses a **repeat until** loop, which is an indefinite loop.



Why is that?

Any loop that repeats for an undefined number of times is an indefinite loop.

The **forever** block in EdScratch is one example of an indefinite loop because it repeats indefinitely. The **repeat until** block is another example because it will loop *until* its condition is met. The condition might be met after just one loop, or maybe it will be met after 20 loops, or it might never be met! Because we don't know exactly how many times the block will loop, it is an indefinite loop.

Write the program from the picture in EdScratch and download it to your Edison robot. Run the program and test it to see how it works.

2. When you run this program, what do you need to do to get Edison to beep? Why is that?
Hint: look at the program and follow each command in sequence.

Task 2: Event + condition = event conditions

One of the main ways to use conditionals in EdScratch is by having an event be the condition. This is called an **event condition**.



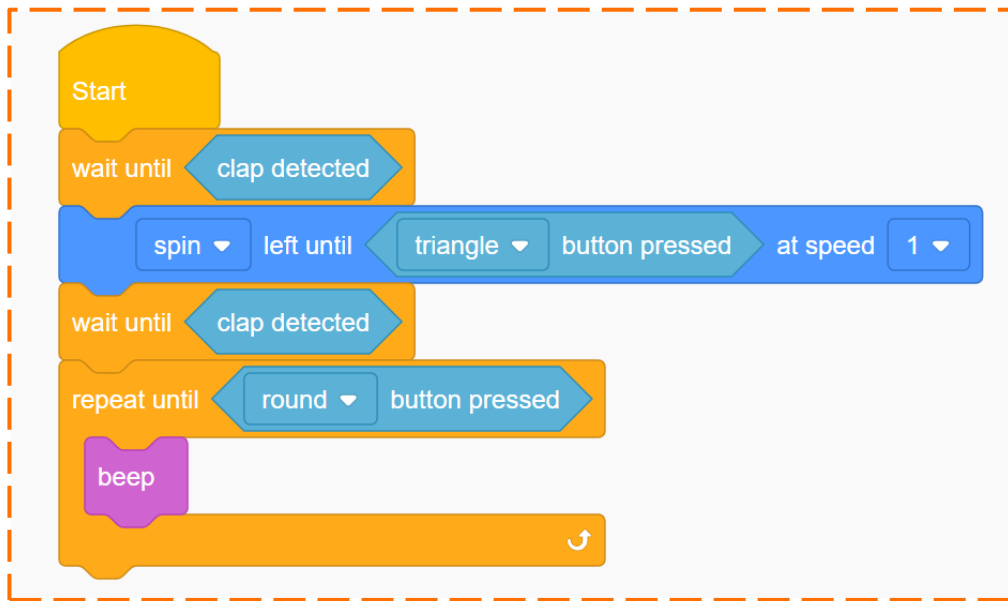
Jargon buster

Remember that in programming, an **event** is something that happens outside of the program code that affects how the program runs.

An **event condition** is a condition that requires a specific event, like a button press, to happen for the condition to be met, triggering the conditional code to run.

You can use event conditions with **until** blocks in EdScratch. Edison will keep doing the action of the **until** block until the event happens. Once the event occurs, Edison will move on to the next block in the program.

This program uses a lot of **until condition** blocks with event conditions:



Write this program in EdScratch and download it to your Edison robot. Run the program. Can you get the program to complete every step and end successfully, with the robot returning to standby mode?

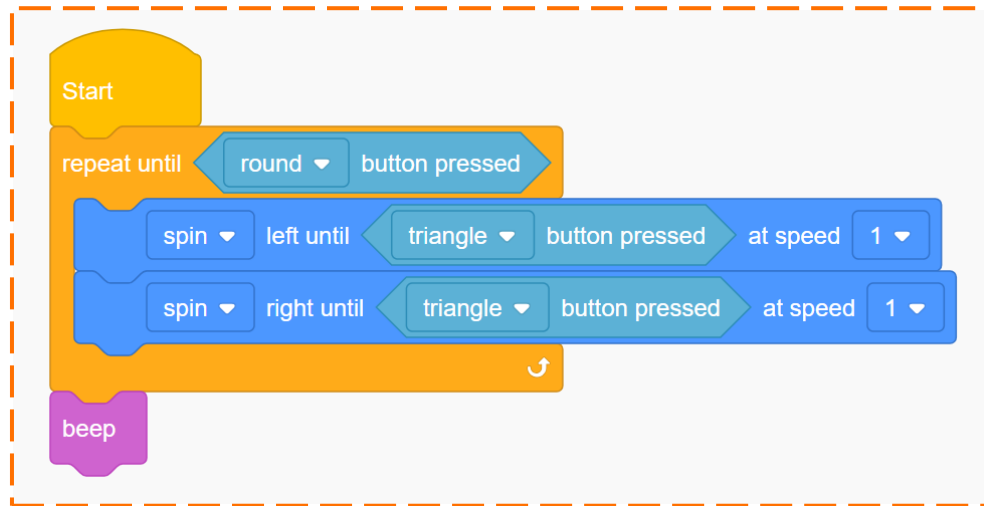
- Once the program is running, how many events need to occur for the entire program to complete successfully?

- The first code block in this program tells the robot to **wait until clap detected**. Can you think of an example of something in real-life that might use this type of **wait until condition** code? What sort of device might use a program with a **wait until condition** as the first action? What condition would trigger the conditional code? What would the program make the device do once the event condition occurred?

U4-1.1a Change it up: Robot error or human error?

Sometimes when we write a program for Edison, it seems like the robot just won't do what we want. Like all computers, there are limits to what Edison robots can do. But before you blame the robot for your program not working, ask yourself if the problem is with the robot... or if it is with the human.

Look at this program which uses three different conditional statements:



This program isn't getting Edison to behave the way the programmer wants:

"Once the program is running, if I press the round button, I think that the robot should stop moving, then beep, and then the program should end. That's not what's happening! I also tried pressing the triangle button twice, then pressing the round button, but the robot just keeps spinning. I think the robot must be broken."

Is the robot really broken? Are there bugs in the program? Has the programmer made a logical error? Or is it something else?



Don't forget

Logical errors are problems with the logic, or the way of thinking, in a program. If a program does not work the way you expect, you may have a logical error. Edison might be running the program exactly as you have written the code, but your way of thinking about the program makes it seem like it isn't working.

Remember, computers cannot think like a human. That's why you have to use **computational thinking** to plan, problem-solve and analyse information the same way a computer does.

What to do

Let's see if we can help figure out what's going on with the program and explain it to the programmer.

The first thing to do is run the program for yourself. That way you can see how it works and start checking for any bugs. Running the program will also let you start checking the programmer's thinking to see if they have made any logical errors.

Write the program in EdScratch and download it to your Edison robot. This program uses event conditions, so you need to start the program and then test it to see if the robot is behaving as you expect it should when each event occur.



Don't forget

To get the program running, press the 'play' (triangle) button one time. This just starts the program, it doesn't count as a button press inside the program.

1. Get the program running by pressing the 'play' (triangle) button. What does this get the robot to do?

2. Is that the behaviour you expected? Why or why not?

3. Now press the triangle button. The robot will start spinning right. Why does this happen?

The programmer said there were two problems with how this program works:

Problem #1:

"Once the program is running, if I press the round button, I think that the robot should stop moving, then beep, and then the program should end. That's not what's happening!"

Problem #2:

"I also tried pressing the triangle button twice, then pressing the round button, but the robot just keeps spinning."

Run the program in your Edison robot again. Follow the steps the programmer described to see if you can replicate the problems the programmer experienced. Do you have the same issues as the programmer? Do you think the problems are human problems or robot problems?

What's going on here?

Let's look at the two problems one at a time.

Problem #1:

"Once the program is running, if I press the round button, I think that the robot should stop moving, then beep, and then the program should end. That's not what's happening!"

This is a human problem. The programmer made a logical error when thinking about how the program should make Edison behave.

To understand the error, try running the program in your Edison again. This time, once the program is running, press the round button one time. Then press the triangle button two times.

The robot will spin left, then spin right, then beep, and the program will end.



Why is that?

Remember, the **repeat until condition** block is an indefinite loop that will loop until its condition is met.

The loop tells Edison to do each item of code inside the loop in order, then come back to the top of the loop. If the loop condition has NOT been met, then the loop tells Edison to start the code inside the loop again. You can think about it as the loop asking Edison the question, 'has the round button been pressed?' If the answer is 'no', then the loop sends Edison back into the loop. If the answer is 'yes', then the loop sends Edison to the code that comes after the loop instead.

A program will only check if the loop condition is met at the start of the loop, not while the code inside the loop is running. The robot needs to complete all of the code inside of the loop first, then it will go back to the top of the loop and check the condition.

Our programmer pushed the round button but didn't complete the conditions in the code blocks inside of the loop. That is what caused the error!

4. In your own words, explain the logical error that the programmer who wrote this program made.

Now, let's look at the second problem:

Problem #2:

"I also tried pressing the triangle button twice, then pressing the round button, but the robot just keeps spinning."

This is also a human problem, but it is not exactly a logical error. The problem here is that, compared to a robot, humans are a bit slow. That's because robots move through code really, *really* fast!

**Why is that?**

Remember, a program will only check if the loop condition is met at the **start** of the loop.

In this program, as soon as the triangle button is pushed the second time, the **spin right until triangle button pressed** block finishes, and the code moves to the top of the loop to check if the round button has been pressed.

This happens very fast. It takes less than 10 milliseconds before the code checks for the round button press. That's less than 1/100th of a second!

Our programmer did push the round button, but by then, the code had already checked the loop condition and sent the robot back into the loop. The programmer was too late!

5. Things not working the way we expect them to work happens all the time in coding. This can be very frustrating! Think about what you can do the next time you write a program that doesn't seem to work. Write a message to your future-self. Make some suggestions about what you can do to try to fix the problem.

U4-1.2 Let's explore if statements

Using conditionals in coding lets you write programs which ask the computer to decide something. Conditionals are how you give the computer exacting instructions to follow so it knows what decision it is making and the conditions for making that decision.

In coding, the most common way to set conditions for a computer is by using an **if statement**.



Jargon buster

An **if statement** is a conditional statement. It is an element of code that is dependent on something else. This bit of code will only happen **if** the condition is met. That's why it is called an 'if' statement!

You probably use 'if' statements to make decisions in life all the time, maybe without even realising it. Look at these examples:

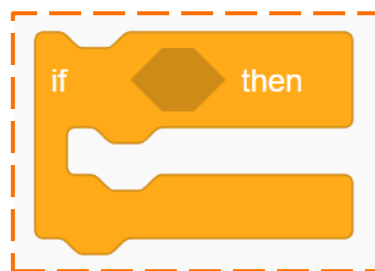
- ♦ **IF** it is cold outside, **THEN** I put on a jacket before I leave the house.
- ♦ **IF** I am hungry after school, **THEN** I eat a snack.

1. Think about a conditional you encounter in your daily life. Write it using the 'if___, then___' formula.

IF _____

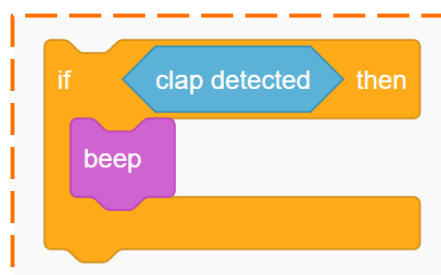
THEN _____

In code, 'if' statements follow this same formula. Look at the **if** block from EdScratch:



Do you see the 'if___, then___' formula in the block? When you use an 'if' statement in code, you are telling the computer that **IF condition** happens, **THEN** do the conditional action.

For example, **IF clap detected**, **THEN** beep:



You can also tell the computer what to do if the condition does NOT happen. To do this, you need to use a type of conditional called an **if-else statement**.



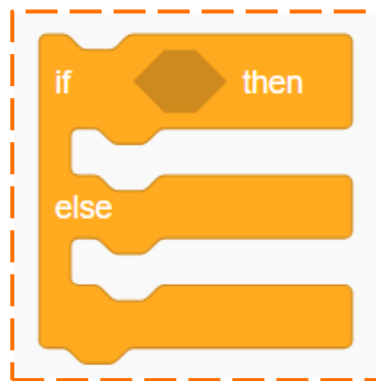
Jargon buster

An **if-else statement** is a conditional statement. Just like all conditionals, this is an element of code that is dependent on something else. An if-else statement tells the program what to do if the condition is met and also tells the program what to do if the condition is NOT met.

The 'else' part of the if-else statement tells the program what to do if the condition is not met.

You can think about an if-else statement like a decision point for the program. An if-else statement tells the program: "if the condition is met, do thing A. If the condition is not met, do thing B."

This is what the **if-else** block looks like in EdScratch:



Just like the **if** block, the **if-else** block uses the basic 'if __, then __' formula but also says what action to do when the condition is not met. An if-else statement lets you make a conditional choice:

- ◆ **IF** it is cold outside, **THEN** I put on a jacket before I leave the house. **ELSE** I go out in a tee-shirt.
- ◆ **IF** I am hungry after school, **THEN** I eat a snack. **ELSE** I wait until dinner time to eat.

Using an if-else statement forces a program to branch. It will either go down one path, or it will go down a different path.

Try it out!

Let's practice using the 'if __, then __ else __' formula to see how it makes programs branch. For this activity, you need to use activity sheet U4-1. This activity sheet has a special treasure map that can only be solved with if-else statements.

Follow each set of instructions to work out the location of each treasure.



Hint!

Did you know that some mathematics symbols are also used in coding? For this activity, you will need to use these symbols:

$A = B$ means 'A is the same as B'

$A > B$ means 'A is greater than B'

$A < B$ means 'A is less than B'

2. The Orb of Marshmallow

Begin at the start spot.

Repeat 3 times:

IF the number < 7 , THEN go left. ELSE go right.

Where is the Orb of Marshmallow? _____

3. The Cloak of Pancakes

Begin at the start spot.

Repeat 3 times:

IF the number > 2 , THEN go left. ELSE go right.

Where is the Cloak of Pancakes? _____

4. The Omelette of Space Eggs

Begin at the start spot.

Repeat 2 times:

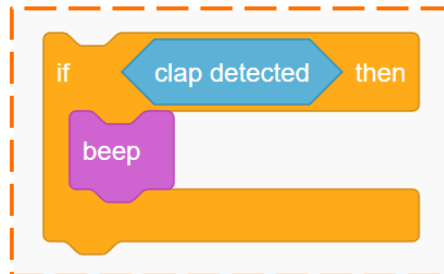
IF the number $= 9$, THEN go left. ELSE go right.

IF the number < 7 , THEN go left. ELSE go right.

Where is the Omelette of Space Eggs? _____

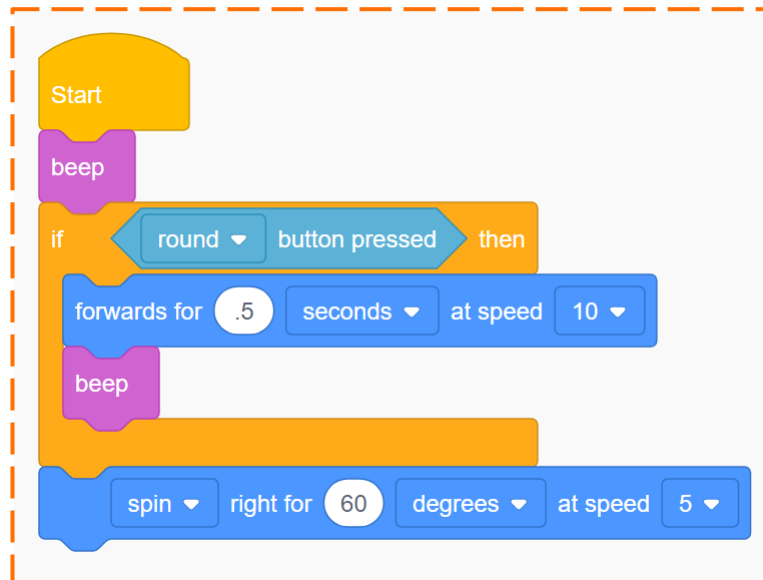
U4-1.3 Let's explore if statements and sequence

When you use an 'if' statement in code, you are telling the computer what to do if that condition happens. In EdScratch, the **if** block needs you to give the condition by using a diamond-shaped input parameter. You also tell the robot what the conditional action is by putting a block or blocks inside the 'mouth' of the **if** block:



What happens in a program that uses an **if** block when the condition is **not** met?

Look at the following program:



1. In this program, what needs to happen for the condition in the **if** block to be met?

Write the program in EdScratch. Download the program and run it in your Edison robot.

2. What happened when you ran this program? Did the conditional code (the code inside the **if** block) run?

3. From what you found, what do you think happens in a program that uses an **if** block when the condition is NOT met?



Why is that?

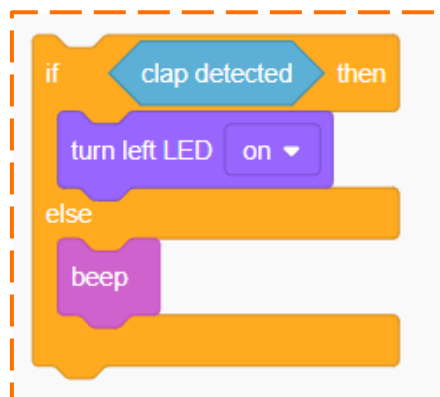
Remember, all programs move through the code step-by-step in sequential order. When the program gets to an **if** block, it checks to see if the condition has been met. If it has, the program runs the code inside the block. If the condition has not been met, then the program skips the code in the **if** block and moves on to the next line of code in the program.

Edison moves from code block to code block very fast. It takes less than 10 milliseconds before the robot is already at the **if** block checking for the round button press. That's less than 1/100th of a second! It's almost impossible to press the round button in time.

If you want to see the conditional code run, what extra block could you add to the program to give yourself some more time to press the round button?

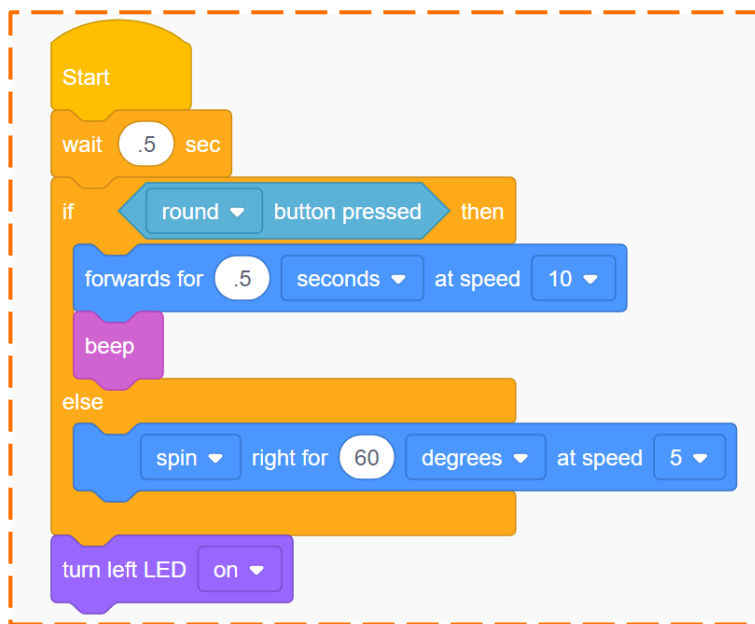
Just like an **if** block, when an EdScratch program gets to an **if-else** block, it checks to see if the condition has been met. An **if-else** block tells the robot both what to do if the condition is met and what to do if the condition is not met, so the robot has an action to take no matter what.

If the condition is met, the robot will do the code inside the 'if' part of the block. If the condition is not met, the robot will do the code inside the 'else' part of the block:



Using an if-else statement forces a program to branch because the robot can only do the 'if' or the 'else' code, but not both. Once the robot finishes either the 'if' or the 'else' code, it moves on to the next line of code in the program.

Look at this EdScratch program:



4. If you ran this program in Edison and the robot did NOT detect a round button press, would the robot beep? Why or why not?

5. When this program runs, what actions will always happen whether or not the robot detects a round button press? *Hint:* Follow the program in sequential order. What three things happen no matter what?

U4-1.4 Let's explore stacking and nesting if statements

Conditionals are powerful code elements in any computer language, including EdScratch. Using conditionals like 'if' statements in EdScratch lets you write all types of interesting programs for your Edison robot.

All the conditional blocks, including both the **if** block and the **if-else** block, are in the **Control** category in EdScratch. Loops are also in the **Control** category. This is because both conditionals and loops allow you to control the flow of your program. The **if** block and the **if-else** block have something else in common with loops too: you can stack or nest them in programs.



Why is that?

In block-based programming languages like EdScratch, adding blocks together is sometimes called stacking blocks. When you use multiple loops together in a program one after another, you can say you are stacking the loops. You can also stack **if** and **if-else** blocks with each other and with loops.

Likewise, just like you can nest loops by putting one loop block inside another loop block, you can nest **if** and **if-else** blocks with each other and with loops too!

Task 1: What's going to happen this time?

When a program has multiple loops or conditional blocks stacked or nested together, it can be a bit confusing to follow the flow of the program. To understand what the program is going to do, you need to think about each action that is going to happen in sequence.



Don't forget

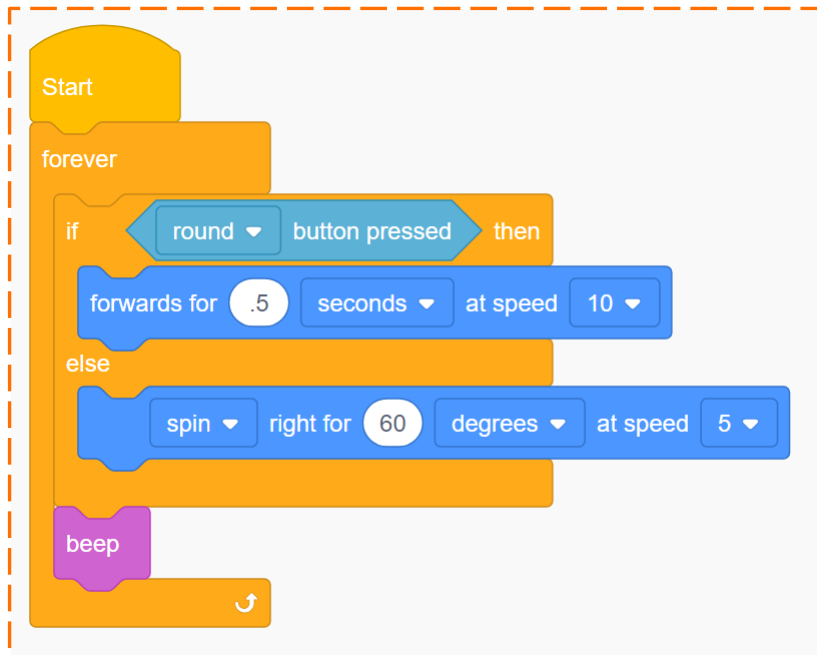
All EdScratch programs you make work in the same basic way. The program tells the robot to start with the top block and then do each action one-by-one. Once a block is executed, the program moves on to the next block.

If a block contains conditional code, the program first checks if that condition has been met. The result of the condition check determines what actions the program does next.

While loops and conditionals do control the flow of a program, all programs still follow sequential order. When you look at a program with nested loops and conditionals, keep in mind that this step-by-step flow is always happening. Remembering this will help you be able to follow what is going on in a program.

Look at the following programs and answer the questions.

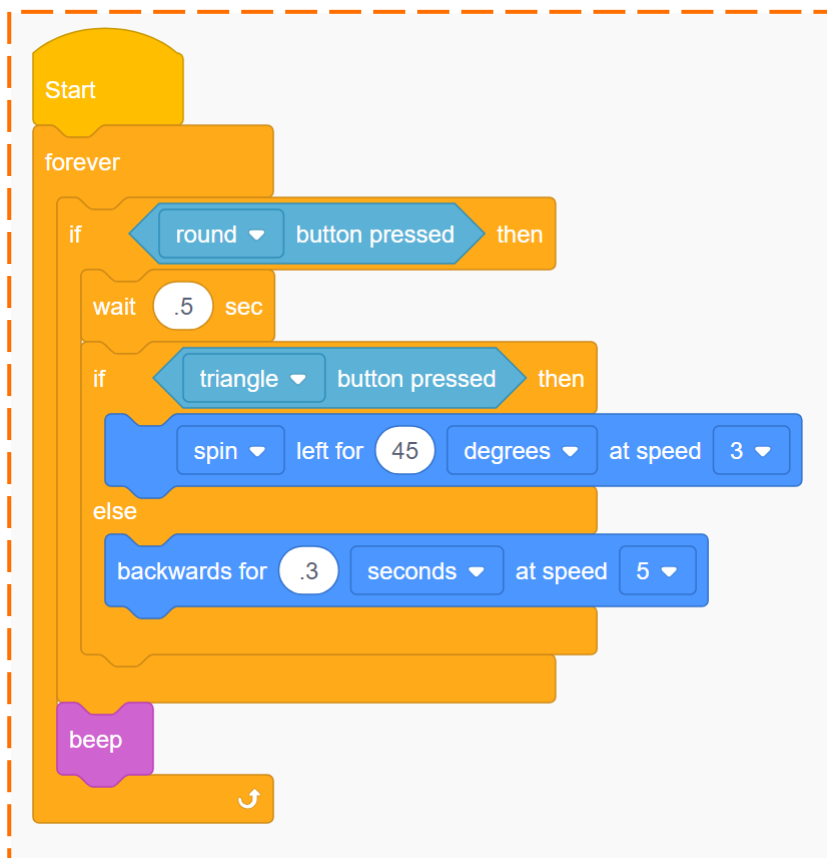
Program 1:



1. If you run program 1 but never press the round button, what will happen? Why?

2. If you run program 2 but never press the round button, what will happen? Why?

Program 2:



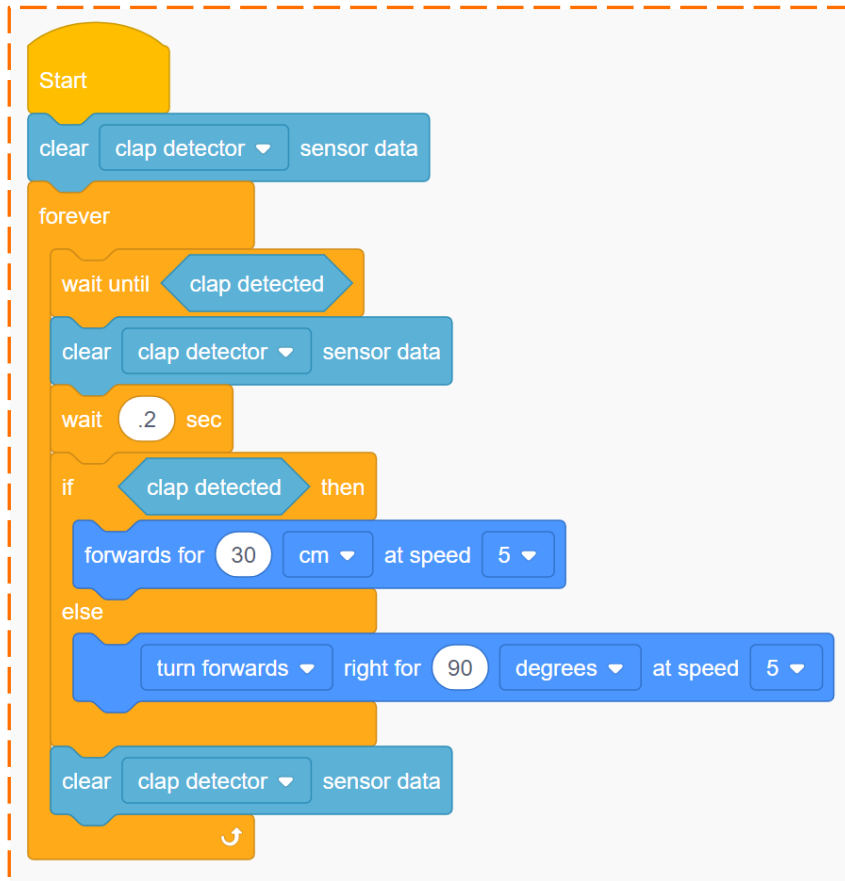
3. If you run program 2, what do you need to do to get the robot to drive backwards? Why?

Choose one of the two programs to write in EdScratch. Download the program and test it out in your Edison robot. Experiment to see how these programs with nested 'if' statements work.

Task 2: Clap-controlled driving

By nesting an **if-else** block inside a **forever** loop, we can build a clap-controlled driving program for Edison. The program needs to make Edison wait until a clap is detected, then either drive forwards or turn 90 degrees, depending on if the robot detects one clap or two claps.

Look at this clap-controlled driving program:



This program uses a special block from the **Sensing** category called the **clear sensor data** block.



Why is that?

Remember that Edison has different sensors, including the bit of tech that lets the robot detect sounds like claps. These sensors generate data when they detect specific events. Some of this sensor data is stored in Edison's memory. This stored data can sometimes be a problem, making the robot react to an old event because the robot still 'remembers' the old event.

When Edison checks if a condition has been met, if there is stored data, the robot will think that the condition has been met, even if it has not! That's why it's good coding practice to clear the sensor data. This is especially important when you use sensor events in conditionals nested inside loops. You don't want the data from a previous loop to affect the next loop!

It is also best to clear the data at the start of a program, just in case the robot has old data stored from a previous program.

Name _____

Take another look at the clap-controlled driving program. Can you follow how the code flows?
Write the clap-controlled driving program in EdScratch.



Hint!

Don't forget to use comments! Adding good comments makes keeping track of what's meant to happen in a program a whole lot easier. This is especially helpful in programs with nested loops and conditionals!

Download the program to your Edison robot and run it. Experiment to see how the program makes Edison respond to claps.

4. The clap-controlled driving program nests an **if-else** block inside a **forever** loop. Why do you think this is the case? What would happen if you didn't use the **forever** block?

U4-1.4a Challenge up: Build a pulley

A pulley is a device which consists of a wheel with a grooved rim over which a rope or chain is pulled in order to lift heavy objects. You can use Edison to create a programmable pulley and write a program in EdScratch to control how the pulley operates!

What to do

Build a pulley system using an Edison robot. You can use EdCreate parts or any other materials you like. You will also need to create a program to operate the pulley using EdScratch.

Your pulley system should use Edison's motor outputs to control the pulley. The motor should move the pulley in one direction (up or down) if the round button is pushed and in the other direction if the triangle button is pushed.

Write your program in EdScratch and test it out with your pulley design.

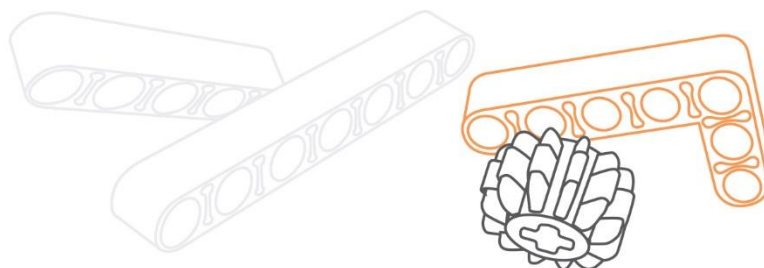
You may need to change your design, your program or both to get your pulley to work. That's okay! Experiment to see what works.



Hint!

Try stacking or nesting 'if' statements in your pulley program to get the pulley to respond to the different button pushes.

1. How did it go? Write about what happened in your pulley project. What went wrong? How did you overcome any problems you encountered? What was the best part of the project? Why was it the best part?



U4-2.1 Let's explore pseudocode

Making good computer programs takes more than just writing code. You also need to be able to problem-solve when things go wrong with a program. Planning your programs before you begin coding is another important and useful skill in programming.

Just like you can plan out a story using a storyboard or plan out an essay using an outline, **pseudocode** is a tool that programmers use to help plan their programs before they start coding.



Jargon buster

Pseudocode is a way of writing out a program in a simple, easy-to-read format. Instead of worrying about syntax, pseudocode uses normal words to describe what the program will do.

Pseudocode looks a bit like a simplified programming language, but it isn't based on any specific programming language. That's why pseudocode can be used to plan programs in any coding language.

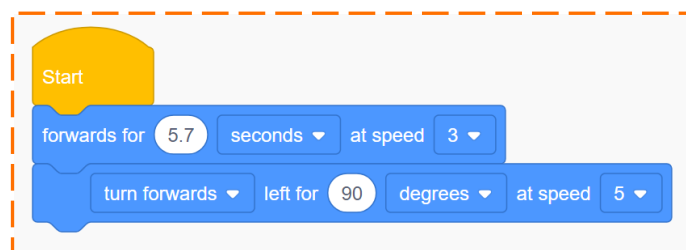
When you plan out your program using pseudocode, you don't need to write everything out in detail or worry about exactly how it will end up looking when you code it. You just need to write a simple version of your plan that makes it easy to follow the flow of the program.

Here's an example of some pseudocode for a driving program for Edison:

```
drive forwards  
turn left
```

The pseudocode outlines the basic plan but doesn't include all of the details. Those get worked out later when you code the program.

Here's the program that the pseudocode translated into:



Do you see how the basic plan from the pseudocode translated into the program?

Writing out a plan in pseudocode first makes it easy to get the structure of your program worked out. You can then adjust it and fill in the details when you write the code.

To make your pseudocode easy to read, you should keep it neat. Write your pseudocode so that it flows the same way the code will when you program it in EdScratch. That means you should write each step one after another, line-by-line.

Using pseudocode is especially helpful to plan programs that use control structures like loops or conditionals. You should indent actions that are inside of loops or conditionals to show that this code is inside of the loop or 'if' statement. Organising your pseudocode in this way makes it a lot easier to understand.

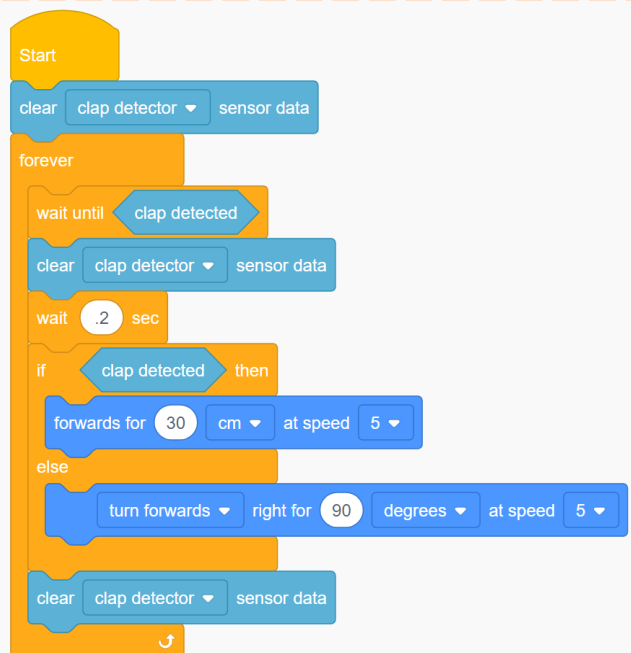
Here is an example of some pseudocode describing a clap-controlled driving program and the corresponding program in EdScratch:

Pseudocode:

```

clear clap data
forever
    wait until clap
    clear clap data
    wait
    if clap
        drive forward
    else
        turn right
    clear clap data
  
```

EdScratch program:



See how the actions line up in both the pseudocode and the program?

Try it out!

Let's try reading some pseudocode instructions. Use activity sheet U4-2 and follow the pseudocode instructions to find the answers to the questions.



Don't forget

Pseudocode should always outline the basic plan, but doesn't need to include all of the details. How much detail you include is up to you and can vary depending on the program.

1. Follow the pseudocode. Where will the program end? _____

```
Start on C facing east
forwards until food
left 90 degrees
repeat 6 times
    forwards 1
    if animal
        left 90 degrees
```

2. Follow the pseudocode. Where will the program end? _____

```
Start on H facing east
repeat 3 times
    forwards 1
    if living thing
        right 90 degrees
    else
        left 90 degrees
backwards 1
```

3. Follow the pseudocode. Where will the program end? _____

```
Start on D facing west
repeat 3 times
    forwards 3
    if animal
        left 90 degrees
    else
        if food
            right 180 degrees
repeat 3 times
    forward until letter
    right 90 degrees
backwards until number
```



Hint!

Think about how **until**, **if** and **if-else** conditional code works.

Will the program run the conditional code in an **if** block if the condition is **not** met? What happens instead? What about in an **if-else** block?

Make sure you follow the pseudocode just like a computer would!

U4-2.1a Change it up: Find the answer

When you first use pseudocode, it might seem a bit awkward. Once you get familiar with pseudocode, however, planning out your programs in pseudocode will save you a lot of time!

What to do

For this activity, you will need to work with a partner to practice writing and following pseudocode instructions. Write some pseudocode instructions using activity sheet U4-2 for your partner to follow. The first line of your pseudocode should tell your partner where to start, including which direction to face.



Don't forget

Make sure your pseudocode is nice and neat and easy to read. Write each step one after another, line-by-line. You should indent actions that are inside loops or conditionals to show that this code is inside the loop or 'if' statement.

Your pseudocode doesn't need to include all of the details, but should clearly outline the plan so that your partner can follow it to find the answers.

1. Write your pseudocode. Be sure and test it out before you exchange with your partner.

Mini challenge!

Control structures, like loops and conditionals, make programs more powerful and coding more fun! Can you include at least two different control structures in your pseudocode instructions?



Hint!

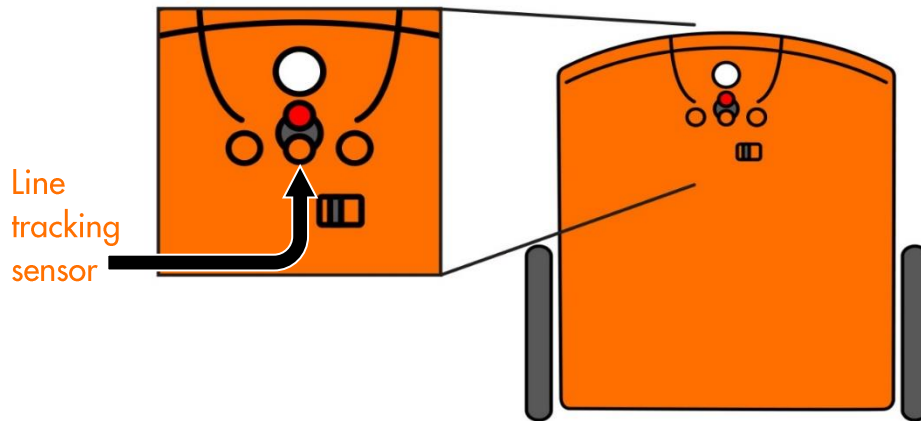
Look at the **Control** category in EdScratch for some ideas on what control structures to use.

U4-2.2 Let's explore Edison's line tracker

Edison robots have different sensors that can detect different things. One of these sensors is the line tracking sensor.

Task 1: Meet Edison's line tracking sensor

The line tracking sensor is the sensor that lets Edison see the difference between dark and light surfaces. The sensor is located on the bottom of Edison, near the power switch.



The line tracking sensor is made up of two parts: a red LED and a light sensor. Look at your Edison robot's line tracker. Do you see the two parts of the sensor?

The line tracking sensor works by shining light from the red LED onto the surface below the robot. The light sensor then measures how much of that light bounces up from the surface. Edison stores the value of the reflected light as a light reading. The more light that is reflected back to Edison, the higher the light reading.

Will a white surface or a black surface reflect more light back to Edison? Use activity sheet U4-3 to test whether a white or a black surface is more reflective to Edison.

Turn Edison on and press the round button twice so that the red line tracking LED comes on. Lift Edison up from the paper slightly and have a close look at the round spot of light that the LED shines onto the surface. Compare how bright the spot of light appears when placed on a black surface and then on a white surface.



Hint!

The more light that is being reflected, the brighter the spot will appear on the surface below.

1. Which surface reflects more light back to Edison, a white or a black surface? Why do you think that?

By measuring how much reflected light is coming from the surface below the robot, the line tracking sensor lets the robot 'see' the difference between dark and light surfaces. Edison doesn't see colours like a human does, however.

The robot can only tell if a surface is **reflective** or **non-reflective**. A reflective surface will shine back a lot of light from the red LED, and a non-reflective surface will shine back very little light.

Edison sees white surfaces as reflective and black surfaces as non-reflective. What about other colours?

2. Will Edison see a red surface as reflective or non-reflective? What about a blue surface? Or green? Use activity sheet U4-3 to test all three colours using the red line tracking LED.
Hint: if there is a bright spot similar to what you see on a white surface, a lot of light is being reflected, and the robot will see that colour as 'reflective'.

Colour	Reflective or non-reflective?
Red	
Blue	
Green	

Task 2: Drive until a black line

We can use Edison's sensors to create inputs in EdScratch programs, telling Edison to look for different types of events and instructing the robot what to do when those events occur.



Don't forget

Inputs are the information and instructions that you give a computer. When you write a program for your Edison robot, you are telling the robot what you want it to do by giving it inputs. Edison's microchip then processes the information to tell the robot what to output as part of the input-process-output cycle.

An **event** is something that happens outside of the program code that affects how the program runs. An event might be a button being pressed or information being relayed from a sensor.

Let's try using the line tracking sensor in a program which tells Edison to drive until it detects a black line. To write this program, you will need to use blocks from the **Sensing** category in EdScratch.

Look at this program:



The first code block in this program turns the line tracking LED on. Whenever you want to use the line tracking sensor in a program, you need to turn it on.



Why is that?

Some of Edison's sensors are always on and checking for events. The sound sensor that can detect claps is an example of this 'always on' type of sensor.

Other sensors, like Edison's line tracker, are off by default. You need to include code in your program to turn these sensors on. Just turning the line tracking LED on isn't enough, however. You also need code to tell the sensor what event to check for (reflective surface or non-reflective surface) and what to do if that event is detected.

Write the program in EdScratch and use activity sheet U4-3 to test it with your Edison robot. Line your robot up on the outline facing the black line on the activity sheet and run your program. Does Edison stop at the black line?

Mini challenge!

Will your program make Edison stop at the coloured lines on activity sheet U4-3 as well as the black line? Why or why not?

Think about whether or not the program will make Edison stop at each of the colours, then test to see if you predicted correctly!

U4-2.2a Change it up: Drive inside a border

You can use Edison's line tracking sensor to write a program that keeps Edison driving inside a black border.

What to do

The first thing you need to do is plan your program using pseudocode.



Don't forget

Make sure your pseudocode is nice and neat and easy to read. Write each step one after another, line-by-line. You should indent actions that are inside loops or conditionals to show that this code is inside the loop or 'if' statement.

Your pseudocode doesn't need to include all of the details, but should clearly outline your plan so that you can use it to write the code later.

Your program should have Edison drive until it detects a black line. If the robot detects a black line, it should back up, then turn away from the line, and then start driving again until it detects a black line.

1. Write your pseudocode.

Use your pseudocode as a guide to help write your program in EdScratch. Download it to your Edison robot and test it using activity sheet U4-4.



Hint!

If your program doesn't work the first time, that's okay! Check the logic of your pseudocode to see if you can spot any issues. Don't forget to check for messages in the bug box too!

U4-2.3 Let's explore algorithms

Edison's line tracking sensor can detect if the surface below the robot is reflective or non-reflective. You can use this sensor to get Edison to behave in different ways, such as driving until it detects a black line, then stopping. You can also use this sensor to program Edison to follow a black line, even if you don't know what that black line looks like. To do this, you first need to create an **algorithm**.



Jargon buster

An **algorithm** is a broad set of instructions to solve a set of problems. An algorithm lays out a process or a set of rules to be followed in order to solve any problem in the set.

Computer programs often use algorithms, but programs and algorithms are not the same thing. Remember, a computer program is a collection of instructions that tell a computer to perform a specific task. An algorithm lays out the logic for how to solve a whole set of problems, not just one specific task. You can write a computer program that uses an algorithm, but not all computer programs are algorithms.

Algorithms are really helpful because using an algorithm lets a person, or a computer, solve a whole set of problems, even if you don't know every detail.



Why is that?

Let's say you want to teach your friends how to make fruit pies. If you know that all of your friends have apples, you can just write down one recipe for apple pie.

Not all of your friends might have apples, however. What if one of your friends has blueberries, another has cherries, and a third has apples? They cannot all follow the apple pie recipe. You would need to write a separate recipe for each different fruit.

What if you don't know what fruit each of your friends has? How could you teach them to make fruit pies?

No matter what fruit they have, all of your friends need to follow the same basic instructions: make the dough, fill the pie with the fruit, then bake the pie.

This new set of instructions is an example of an algorithm.

In computer programming, we often want to create instructions for a computer to follow in order to solve a whole set of problems. By using an algorithm, we can write a program that will let the computer solve any problem in the set. Without an algorithm, we would need to write a new program for every single problem individually.

Task 1: Follow a black line

You know that you can use Edison's line tracking sensor to detect black (non-reflective) and white (reflective) surfaces. We can create an algorithm that uses this sensor to get Edison to follow any black line.



Why is that?

Let's say you draw a black line for Edison to follow. To get Edison to follow your line, you could write a program which makes Edison drive the exact path of the line. If you make a new line, however, you will need to write a whole new program for that new line.

Instead, you can create an algorithm.

This algorithm will solve a set of problems: 'follow any black line'. Any specific line you make for Edison to follow is a new problem inside this set.

Using the algorithm to guide the logic, you can then write a program which will work for all of the problems in the set. This way, a whole new program isn't needed for each new problem.

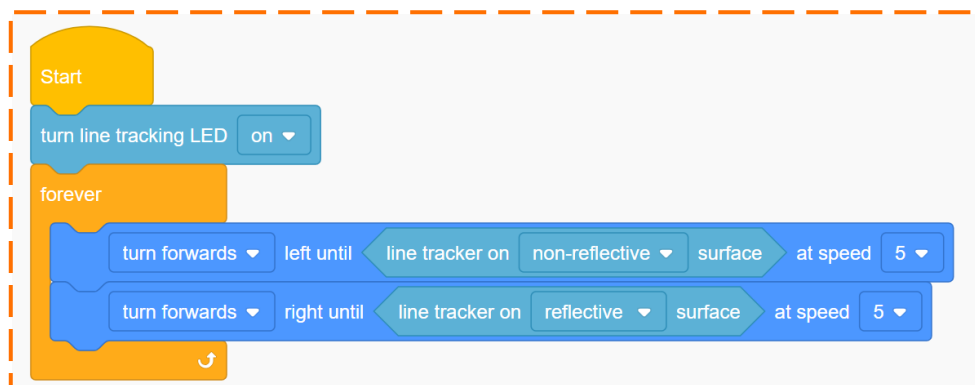
You can plan out an algorithm using pseudocode, just like you do when you plan out a program. Here is an algorithm in pseudocode that will allow Edison to follow any black line:

```

turn line tracker on
loop forever
    drive forwards left until the robot detects a black surface
    drive forwards right until the robot detects a white surface
  
```

This algorithm says that the robot should drive forwards to the left until the line tracking sensor is on a non-reflective (black) surface. Then the robot should drive forwards to the right until the line tracking sensor is on a reflective (white) surface. The robot should keep doing this behaviour forever.

This algorithm can be used to write a program in EdScratch:



Can you see how the algorithm's logic is inside the program?

Name _____

Write the line-following program in EdScratch and download it to your Edison robot. Use activity sheet U4-4 to test the program. Remember to start Edison with the line tracker on a white surface, not directly on the black line.

1. How does the robot move when you run the program? Look at the program and think about the logic in the algorithm. Why does the robot move that way?

Task 2: Follow a different black line

The program you wrote uses an algorithm that is designed to solve any problem in the set of problems: 'follow any black line'. That means this same program should let Edison follow any black line you make!

Make your own line to test. Use a black marker on white paper or make a line using black tape on the floor or a desk. Run the program in Edison to test out your line. Can Edison follow your line?

2. Was Edison able to follow your line? If you had any problems, describe them. What do you think caused the problems?

U4-2.3a Challenge up: There's more than one way to follow a line

To get your Edison robot to follow any line, you need to use an algorithm.



Don't forget

An **algorithm** is a broad set of instructions to solve a set of problems. An algorithm lays out the logic for how to solve a whole set of problems, not just one specific task.

Think about the logic in the algorithm that will allow Edison to follow any black line. At its simplest, what is it saying?

The broad instructions to solve the 'follow any black line' set of problems are: using the line tracker, while moving forward, go one way if on black and the other way if on white. Here's what this very basic algorithm looks like in pseudocode:

```
use the line tracker
forever
  if the robot detects black, drive forward one way (left or right)
  if the robot detects white, drive forward the other way (right or left)
```

If you were planning out an algorithm to then code, you might write this same logic slightly differently, and you would probably add in some of the details, like in this example:

```
turn line tracker on
loop forever
  drive forwards left until the robot detects a black surface
  drive forwards right until the robot detects a white surface
```

Because the logic is the same in both examples, any programs you make using either example will follow the same logic. Even if the programs look different, they will still solve the 'follow any black line' set of problems.

Just how many different ways could you write a program that will solve the 'follow any black line' set of problems? More than you might think!

What to do

Your challenge is to write at least two different programs that use the basic logic of the 'follow any black line' algorithm.

You will need to plan each program using pseudocode, then code it in EdScratch.



Hint!

Pseudocode helps us plan, comments help us debug. Adding comments as you code will help you check your logic and keep track of your thinking, so you can find and fix any issues you encounter when you test your program.

Download each of your programs one at a time. Test each program using activity sheet U4-4 or make your own line to use as a test space. Both of your programs need to use Edison's line tracking sensor and should be able to follow any black line.



Hint!

Think about how you can apply the logic of the algorithm in EdScratch. Look at different conditionals, including **until** blocks, **if** blocks and **if-else** blocks. Don't forget about the **Events** category too!

What do your two programs look like? Either write pseudocode, download and share your program files or write an explanation of each of your programs.

Program 1:

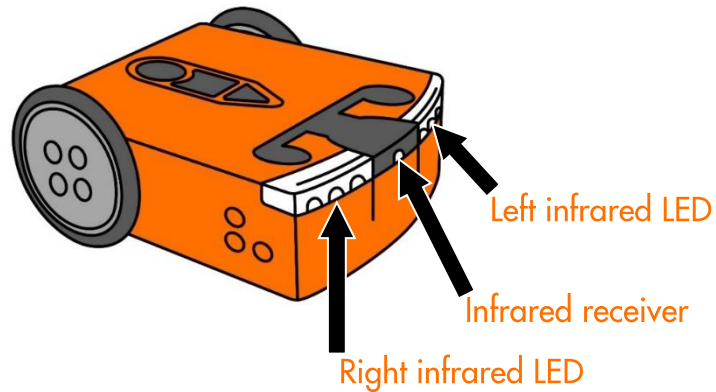
Program 2:

U4-2.4 Let's explore Edison's obstacle detection

Edison robots have different sensors that can detect different things. One of these sensors is the infrared light sensor which we can use to detect obstacles.

Task 1: Meet Edison's infrared light sensor

Edison's infrared light sensor is the sensor that lets Edison emit and detect infrared light. The sensor is made up of three parts all located across the front of Edison: two infrared LEDs (one on the right and one on the left) plus an infrared receiver in the middle. Look at your Edison robot. Do you see the different parts of the sensor?



Just like Edison's red LEDs, the two infrared LEDs can emit light. However, because it's infrared, you won't be able to see it.



Why is that?

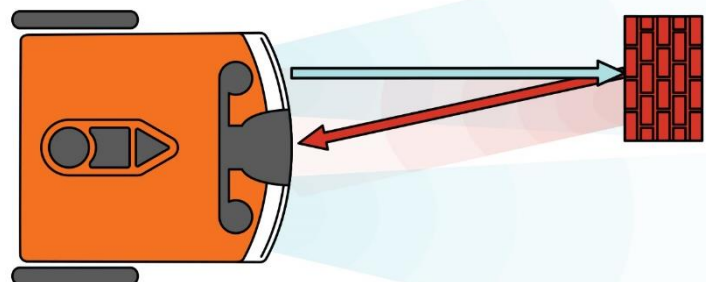
There is a wide range of light. People can see some of this range, but not all of it. Infrared light (which is also called IR light) is not visible to humans.

Even though we cannot see it, people use infrared a lot. For example, infrared light is used in TV remote controls. It's how the remote tells the TV to change the channel or turn up the volume!

Unlike us, Edison can detect infrared light using the infrared receiver on the front of the robot. One way we can use Edison's infrared light sensor is to detect obstacles.

Edison can emit infrared light from the two infrared LEDs. If that infrared encounters an obstacle, like a wall, the light is reflected back towards Edison. Edison's infrared receiver detects the reflected light, telling Edison that there is an obstacle.

Depending on where the obstacle is, infrared light will bounce back from the left LED, the right LED or both. The reflected light tells the robot where the obstacle is located.



Task 2: Forward until obstacle

We can use Edison's sensors to create inputs in EdScratch programs, telling Edison to look for different types of events and instructing the robot what to do when those events occur.



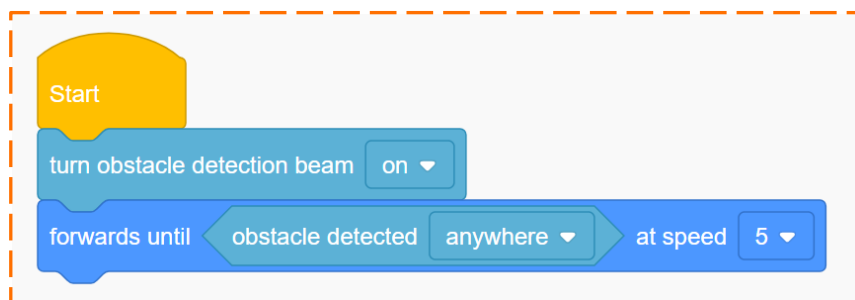
Don't forget

Inputs are the information and instructions that you give a computer. When you write a program for your Edison robot, you are telling the robot what you want it to do by giving it inputs. Edison's microchip then processes the information to tell the robot what to output as part of the input-process-output cycle.

An **event** is something that happens outside of the program code that affects how the program runs. An event might be a button being pressed or information being relayed from a sensor.

Let's try using the infrared sensor in a program which tells Edison to drive until it detects an obstacle. To write this program, you will need to use blocks from the **Sensing** category in EdScratch.

Look at this program:



The first code block in this program turns on the obstacle detection beam.



Why is that?

Some of Edison's sensors are always on and checking for events. The infrared receiver is actually always on, but the infrared LEDs are not.

For Edison to detect obstacles, you need to include code in your program to turn on the obstacle detection beam. This turns on the infrared LEDs and tells the infrared receiver to look for reflected infrared light bouncing back to the robot. Just turning the obstacle detection beam on isn't enough, however. You also need code to tell the sensor what event to check for (in other words, where it should be looking for an obstacle) and what to do if that event is detected.

Write the program in EdScratch. Download the program and test it using your Edison robot. You will need an obstacle to use in your test as well.

Some obstacles will work better with Edison than others. If an obstacle is too small or doesn't reflect enough infrared light, Edison cannot detect it. Select an object that is opaque but not too dark (don't choose black objects) and at least as tall as Edison. For this program, the wall of the room would be a good obstacle.

Test the program to see how it works.

**Hint!**

If your robot isn't detecting your obstacle, try a different object. If it still isn't detecting the obstacle, you might need to calibrate the obstacle detection. Ask your teacher for the special barcode to calibrate obstacle detection.

Task 3: Detect and avoid

Instead of just stopping when it detects an obstacle, you can get Edison to avoid any obstacle it detects and keep driving. To do this, you need to create an algorithm to solve the set of problems: 'detect and avoid any obstacle'.

1. Use pseudocode to write a detect-and-avoid algorithm.

Once you have your algorithm, write an EdScratch program for Edison that uses the logic from that algorithm. Download your program and test it out using Edison and some obstacles.

2. Even when professional computer programmers write programs, they run into problems. Describe one problem you had creating your detect-and-avoid algorithm or program. What did you do to solve the problem?

U4-2.4a Change it up: Faster, faster, smash?

Edison's obstacle detection works by sending out infrared light from the robot's two infrared LEDs. If there is an obstacle, the light bounces off of the obstacle and reflects back to Edison where the robot's infrared receiver detects it. We can program Edison to detect the reflected IR light and react to that event.

Robots can process information incredibly fast, but the task of receiving inputs, processing the information and generating an output still takes some amount of time. There are multiple steps in the Edison robot's obstacle detection process: Edison emits IR light, the light encounters an object, the light bounces off of the object, the robot's infrared receiver detects the reflected light, and, finally, the robot reacts to that event. Just how long does this obstacle detection process take?

What to do

Write a program to get Edison to drive forward until the robot detects an object. Edison should stop driving as soon as it detects the object so that the robot doesn't hit the obstacle.



Don't forget

Whenever you want to use the infrared sensor to detect obstacles, you need to turn on the obstacle detection beam.

Download your program and test it using Edison and an object that Edison should be able to detect. Then adjust the speed input parameter in your program. Test different speeds with Edison to see what happens with faster and slower speeds.

1. What happens when you run your obstacle detection program with a very fast speed?

2. In programming, we sometimes have to balance different features to achieve the best result. This is known as a trade-off. Describe the trade-off between Edison's speed and the robot's ability to detect obstacles.

U4-2.4b Challenge up: If line, go right. If obstacle, go left

Edison robots have different sensors that can detect different things. You can use multiple sensors in a single program, getting the robot to react to different types of events.

What to do

Write a program so that your Edison robot moves through a grid, checking each new section of the grid for any non-reflective (black) surfaces or obstacles. Your program should tell Edison that if the robot detects a black surface, it needs to turn right, but if it detects an obstacle, the robot should turn left instead. Test your program out using activity sheet U4-5. See if you can write a program so that your Edison robot starts on the outline, then uses its sensors to get to the parking zone goal.



Hint!

- Sensors like the obstacle detection beam and line tracker need to be turned on to work. Your program also needs to tell Edison what event the sensors should look for and how to react to that event.
- Some of Edison's sensors, including obstacle detection, generate and store data when they detect specific events. It is good coding practice to clear the sensor data, especially when you use sensor events in conditionals nested inside loops. You don't want the data from a previous loop to affect the next loop!
- It is also wise to clear the sensor data at the start of a program, just in case the robot has old data stored from a previous program.
- Pseudocode helps us plan, comments help us debug. Use them to help you plan and test your program!

Mini challenge!

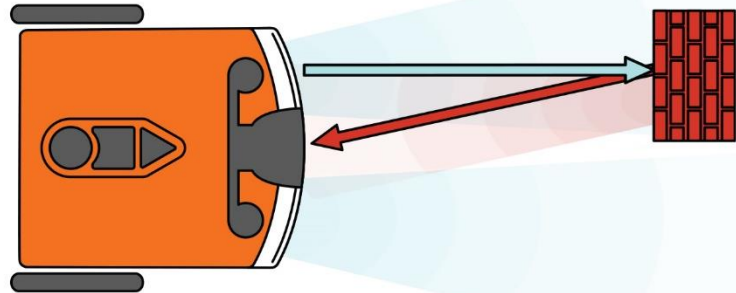
Can you make your own 'grid maze' for Edison to solve using its sensors? Design a layout that has a start, a goal, and both black surfaces and obstacles to detect. Write a program that uses the robot's sensors to get Edison to the goal.

U4-2.4c Change it up: Where is the obstacle?

Edison can emit infrared light from the two infrared LEDs on the right and left of the robot. If that infrared light encounters an obstacle, like a wall or your hand, the light is reflected back towards Edison. Edison's infrared receiver detects the reflected light, telling Edison that there is an obstacle.

Depending on where the obstacle is, infrared light will bounce back from the left LED, the right LED, or both. The reflected light tells the robot where the obstacle is located.

You can program Edison to react with different outputs depending on where the robot detects the obstacle.



What to do

Create a program in EdScratch to get Edison to detect obstacles but respond with a different output depending on where the obstacle is located: on the right, on the left, or straight ahead.

In this challenge, you need to use blocks from the **Events** category. Your program should contain all of the blocks in this picture:



What the robot outputs for each of the different events is up to you.



Hint!

You probably don't want to use Edison's motors as outputs in this program. Why? If the robot moves, it will change position relative to the obstacle. That could become really confusing!

What outputs could you use instead? What could you do to make your outputs help communicate where the robot detected the obstacle?

Download and test your program by putting objects to the right, then the left and then straight in front of Edison.

U4-2.4d Challenge up: 3D maze

You can use Edison's obstacle detection beam to program the robot to detect and respond to obstacles. Can you use obstacle detection to get Edison to drive autonomously through a maze?

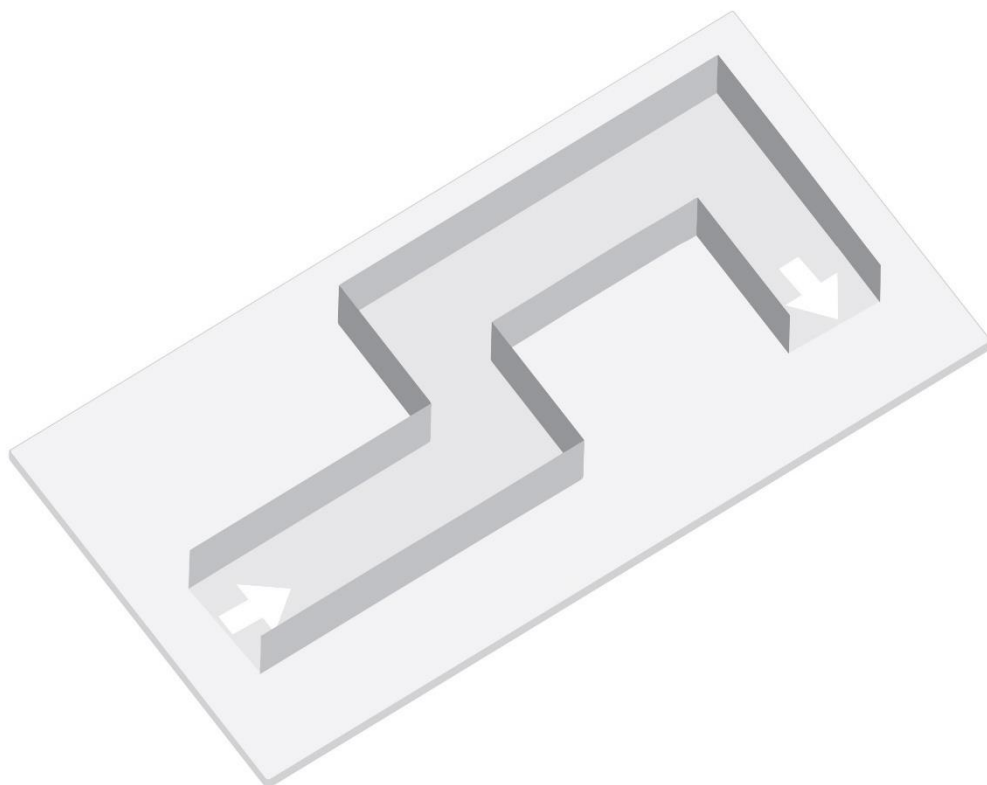
What to do

In this challenge, you will need to build a 3D maze for Edison to navigate using obstacle detection. The walls of your maze will need to be high enough so that Edison can detect and react to them. You also need to write a program to get Edison through the maze using obstacle detection.



Hint!

- Some of Edison's sensors, including obstacle detection, generate and store data when they detect specific events. It is good coding practice to clear the sensor data, especially when you use sensor events in conditionals nested inside loops, plus at the start of a program.
- Remember that Edison can detect where an obstacle is located: to the right, to the left, or straight ahead. This may help you create a successful program.
- Pseudocode helps us plan, comments help us debug. Use them to help you plan and test your program!

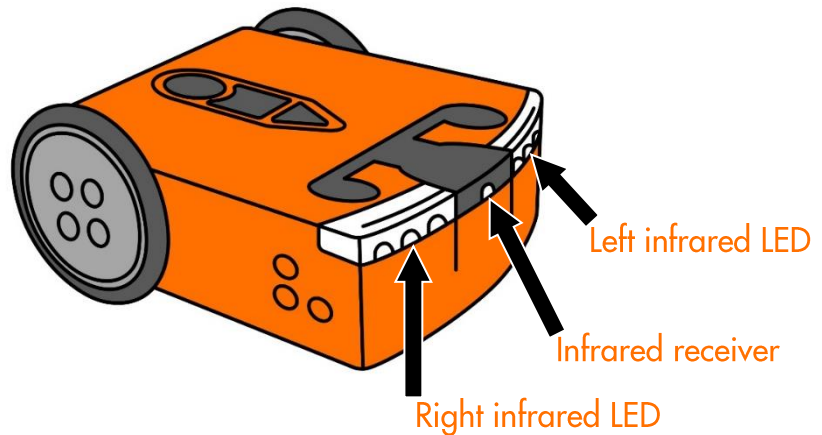


U4-2.5 Let's explore messaging with Edison

Edison robots have an infrared light sensor which can be used to detect obstacles. We can also use this sensor in another way: to send and receive infrared messages.

Task 1: Edison and infrared messages

Edison's infrared (IR) light sensor is the sensor that lets Edison emit and detect infrared light. The sensor is made up of three parts all located across the front of Edison: two infrared LEDs (one on the right and one on the left) plus an infrared receiver in the middle.



In obstacle detection, we use the infrared LEDs to emit infrared light and the infrared receiver to check for any of that light that has been reflected off of objects back to Edison. We can also use the infrared receiver to detect infrared light from other sources, like a TV or DVD remote control, or another Edison robot. Using the infrared receiver this way lets you send or receive messages using your robot.



Why is that?

TV remote controls use infrared light to send 'messages' to the television set. These messages aren't messages like you might send to a friend, however. The message is a pre-set signal that activates a specific function. For example, one message tells the TV to turn up the volume and a different message tells the TV to turn the volume down.

Each button on a remote sends a different message using infrared light!

We can also use infrared light to send and receive messages using Edison robots. One robot can send out an infrared message using its two IR LEDs which another robot's IR receiver can detect.



Don't forget

Infrared is sometimes abbreviated as IR. In EdScratch, **IR message** blocks are blocks that relate to Edison's infrared messaging using the infrared LEDs and infrared receiver.

Task 2: Message received

To use messaging with Edison robots, you always need at least two robots, one to send out the IR messages and one to detect and react to the IR messages.

For this activity, you need a partner or a group. One robot is going to send out a message. All the other robots need to wait until they detect a message. Once the receiving robots detect a message, each robot should react by dancing!

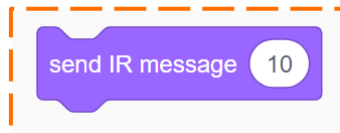


Why is that?

You can use Edison's infrared sensor in a program which tells Edison to detect an IR message event. You also need another Edison robot to run a program which sends out an IR message, however, or your first robot will have nothing to detect!

The 'sending IR message' program

To send an infrared message with your Edison robot in EdScratch, you need to use this block:



1. The **send IR message** block is in the **LEDs** category of blocks in EdScratch. Why is that?

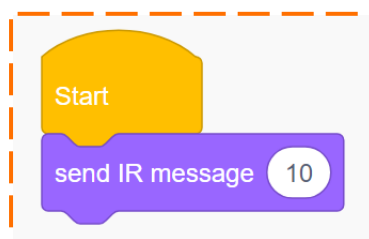
The **send IR message** block has an input parameter which you can set to send a specific message.



Why is that?

Just like a remote control can send different messages to a television, you can send different messages from your Edison robot. In EdScratch, you can change the message by changing the value of the input parameter in the **send IR message** block. The **send IR message** block has an input range of 0 to 255. In other words, Edison can send and receive 256 different 'messages'. Imagine if a remote control could send that many messages. That would be a LOT of buttons!

Sometimes we want to create programs telling Edison to react a specific way to a specific message. Other times, like in this activity, we just want Edison to react if any IR message is detected. The robot that is sending out the IR message for the other robots to detect needs to use this base program:



For this activity, you can set the input parameter in the **send IR message** block to whatever value you like. You also need to write more code for this robot to do after it sends out the IR message. Use at least two different types of outputs including blocks from the **Drive** category to make your robot dance after it has sent out the IR message.

The 'receiving IR message' program

All of the robots which are going to detect the IR message need to have the same base program:



This program uses a block from the **Sensing** category in EdScratch with the **wait until** control block. You don't need a block to turn on the IR message detection.



Why is that?

Some of Edison's sensors are always on and checking for events. The infrared receiver is one of these 'always on' sensors, so you don't need code to turn it on. You do need to tell it what kind of IR event to detect, however, and what to do if that event is detected.

In the base program example, the code tells Edison to look for an **IR message detected** event. This event will be triggered if the robot detects an IR message, no matter which message it detects. That's why it doesn't matter what value the 'sending IR message' program uses!

Use the base program and write more code to tell the robot what to do once it receives an IR message. Use at least two different types of outputs including blocks from the **Drive** category to make your robot dance once it receives the IR message.

Once all of the robots have their programs downloaded, it's time to party!



Hint!

Press the play (triangle) button on all of the receiving robots first, then on the sending robot. If you run the sending robot's program before the receiving robots are ready, the receiving robots will miss the message!

U4-2.5a Change it up: Remote-controlled flag machine

Do you remember Edison's special barcodes? Edison comes with some programs already loaded which we can get the robot to access by using different barcodes. There is also a set of barcodes we can use along with Edison's infrared sensor to pair the robot with a TV or DVD remote control. We can then write an EdScratch program telling Edison what to do when it detects a specific remote control code.



Why is that?

The programmable TV remote codes are a special type of barcode. These barcodes let Edison store a specific code that the robot can reference later.

Unlike other Edison barcodes, these barcodes don't activate a program in Edison. Instead, the barcodes tell Edison to store the next remote-control button press that the robot detects as a specific remote-control code.

To use a programmable TV remote code, you first need to scan the barcode with your Edison and pair it to a button of your choice on a remote control. For example, you might scan remote code #1 and match that to the 'volume up' button on your remote. Once paired, whenever you push that button on the remote, Edison will reference the IR signal as 'remote code #1'.

You can then write a program in EdScratch using that same code as an input. Your EdScratch program will need to tell Edison what remote codes to detect and what to do if the robot detects a specific code.

By using remote codes as events in EdScratch programs, you can build robotic creations that you can control with a TV remote control! Let's try building a flag machine using Edison's motor outputs which you can control using a TV remote.

What to do

The goal of this activity is to build and program a flag machine that you can use to help make studying for a test more fun.

Your creation should let you control a two-sided flag which has the word 'Yes' on one side and the word 'No' on the other side of the flag. You will need to write a program in EdScratch that will let you show the 'Yes' side of the flag when you press one button on a TV remote and the 'No' side when you press a different button. When your flag machine is finished, work with a partner and take turns quizzing each other. Use your flag machine to signal if your partner answered your question correctly or not!

You need to design how your flag machine will work using Edison's outputs and write a program in EdScratch to control it using remote codes.

The first thing to do is pair your robot to a remote control. Look at the programmable TV remote codes on activity sheet U4-6. For this activity, you will probably need to use two of these remote codes.

To pair your robot with one of these codes, you need to follow these steps:

1. Place Edison facing the barcode on the **right side** of the barcode.
2. Press the record (round) button **three times**.
3. Wait while Edison drives forward and scans the barcode.
4. Choose a button on your remote control that you want to match to that remote code. Point the remote at your robot and press your selected button. Be sure and use different remote-control buttons for each different remote code.



Hint!

You need Edison to be able to drive in order to scan a barcode, so be sure to scan whichever barcodes you need before you start building your robotic creation!

You also need to write a program in EdScratch that tells Edison what to do if it detects the different remote codes. Your program should tell Edison to show one side of the flag if you press the first remote-control button your paired and the other side of the flag if you press the other button.

Once you have built and programmed your creation, try it out!



U4-2.5b Challenge up: Build and control the EdCrane

Edison has a set of barcodes we can use along with Edison's infrared sensor to pair the robot with a TV or DVD remote control. We can then write an EdScratch program telling Edison what to do when it detects a specific remote control code.



Why is that?

The programmable TV remote codes are a special type of barcode. These barcodes let Edison store a specific code that the robot can reference later.

Unlike other Edison barcodes, these barcodes don't activate a program in Edison. Instead, the barcodes tell Edison to store the next remote-control button press that the robot detects as a specific remote-control code.

To use a programmable TV remote code, you first need to scan the barcode with your Edison and pair it to a button of your choice on a remote control. For example, you might scan remote code #1 and match that to the 'volume up' button on your remote. Once paired, whenever you push that button on the remote, Edison will reference the IR signal as 'remote code #1'.

You can then write a program in EdScratch using that same code as an input. Your EdScratch program will need to tell Edison what remote codes to detect and what to do if the robot detects a specific code.

By using remote codes as events in EdScratch programs, you can build robotic creations that you can control with a TV remote control!

What to do

In this activity, you will build and program the EdCrane. The EdCrane is a remote-controlled crane with a magnetic 'hook' that you can use to lift and move small metal objects.

The first thing to do is pair your robot to a remote control using the programmable TV remote codes on activity sheet U4-6. For this activity, you will need to use four remote codes to tell the EdCrane to do one of four different actions.

1. Plan out your remote codes and remote-control buttons:

EdCrane action	Remote code	Remote control button
Spin the magnetic spool clockwise		
Spin the magnetic spool counter-clockwise		
Spin the crane clockwise		
Spin the crane counter-clockwise		

Using your plan as a reference, pair your robot to each code. To pair your robot with one of these codes, you need to follow these steps:

1. Place Edison facing the barcode on the **right side** of the barcode.
2. Press the record (round) button **three times**.
3. Wait while Edison drives forward and scans the barcode.
4. Choose a button on your remote control that you want to match to that remote code. Point the remote at your robot and press your selected button. Be sure and use different remote-control buttons for each different remote code.



Hint!

You need Edison to be able to drive in order to scan a barcode, so be sure to scan whichever barcodes you need before you start building your robotic creation!

Once the robot is paired to the remote, you can build the EdCrane and write a program in EdScratch that tells Edison what to do if it detects each of the four different remote codes.



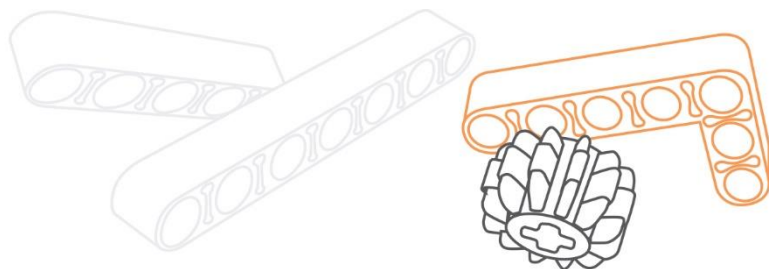
Use this link

Go to meetedison.com/content/EdCreate/EdBuild-EdCrane-instructions.pdf

This link will take you to the step-by-step instructions for building the EdCrane.

Using your plan as a reference, write your EdScratch program so that the EdCrane will react to each remote code with the outputs it needs to complete the desired action.

Once you have built and programmed your creation, try it out!



U4-2.5c Challenge up: Firefighting water cannon

Fighting fires, including major disasters like wildfires, is a dangerous but necessary task. When terrible fires break out, the conditions can become extremely dangerous for humans. One way to help fight these fires is by using firefighting robots.

For this challenge, you will need to build and program the EdTank to use its cannon to shoot a jet of water (represented by the rubber band) to help fight a fire.

What to do

To complete this project, you will need to build the EdTank, create your test space, and program your EdTank using EdScratch.

Test-space set-up

You need to create the test-space you will use to run your firefighting water cannon program. Create two parallel black lines on a white surface, such as a large poster, table, or the floor. You can put the lines as far apart from each other as you like.

One black line represents the firefighting team's base, and the other represents the 'ash line' beyond which the fire is burning.



Hint!

The firefighting EdTank robots will fire their rubber bands out from the 'ash line'. Make sure that this line is facing somewhere where no one will get hit, such as a wall.

The EdScratch program

You will need to write two different programs, one for the top Edison and one for the bottom Edison in the EdTank.

Your programs need to control the EdTank so that the robot will drive from the firefighting team's base until it encounters the 'ash line' (but no further), shoot out its water payload (in other words, fire the cannon) and then return to the firefighting team's base.



Hint!

- You can use messaging in your programs to have the two Edison robots communicate with each other.
- Pseudocode helps us plan, comments help us debug. Use them to help you plan and test your program!

The build

You will need to build the EdTank to use as the firefighting water cannon.



Use this link

Go to meetedison.com/content/EdCreate/EdBuild-EdTank-instructions.pdf

This link will take you to the step-by-step instructions for building the EdTank.

Remember that each of the two robots will need to be programmed with one of your EdScratch programs. Be sure to get the right program into the right robot!

Try it out!

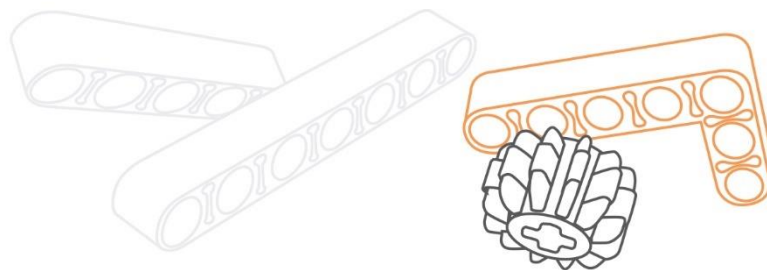
Test out your solution on the test space.

Did your firefighting robot drive up to the 'ash line', fire the water cannon and then come back to the base to be loaded again?



Hint!

If your firefighting robot isn't working quite right, don't give up! Keep experimenting, testing and problem solving until you get your firefighting robot working like a hero!



U4-2.5d Challenge up: Semi-automated digger

Remote-controlled construction machines allow people to handle potentially hazardous tasks or environments more safely. In construction projects which require demolition, it is especially important to keep the machine operators away from flying debris. Many tasks on a construction site need to be performed again and again. Pre-setting, or automating, a portion of this type of task limits the amount of active operation required by the controller. Automating helps save time and limits the chances of human errors occurring, all while keeping the operators safer.

For this challenge, you will need to build and program the EdDigger to perform a rubble clean-up task semi-autonomously.

What to do

To complete this project, you will need to build the EdDigger, create your test space, and program your EdDigger using EdScratch.

Test-space set-up

You need to create the test-space you will use to run your rubble clean-up program. Create a 'worksite' with a pile of 'rubble' in one area and a drop-off zone in a different area where the rubble needs to be delivered so it can be hauled away.



Hint!

You can use any small objects to represent the rubble, even extra bits of EdCreate! Just make sure the EdDigger will be able to scope up the objects you choose.

The EdScratch program

You will need to write two different programs, one for the top Edison and one for the bottom Edison in the EdDigger. The robot should only start moving when the remote-control operator presses the right button on a remote control.

Once the remote code is detected, the robot should be able to perform the rubble clean-up task autonomously. Your programs need to control the EdDigger so that the robot will drive from the drop-off zone to the rubble pile, scoop up some rubble, then return to the drop-off zone and deliver the rubble.



Hint!

- You will need to use one programmable TV remote code from activity sheet U4-6.
- You can use messaging in your programs to have the two Edison robots communicate with each other.
- Pseudocode helps us plan, comments help us debug. Use them to help you plan and test your program!

The build

You will need to build the EdDigger to use in your rubble clean-up task.



Use this link

Go to meetedison.com/content/EdCreate/EdBuild-EdDigger-instructions.pdf

This link will take you to the step-by-step instructions for building the EdDigger.

Remember that each of the two robots will need to be programmed with one of your EdScratch programs. Be sure to get the right program into the right robot!

Try it out!

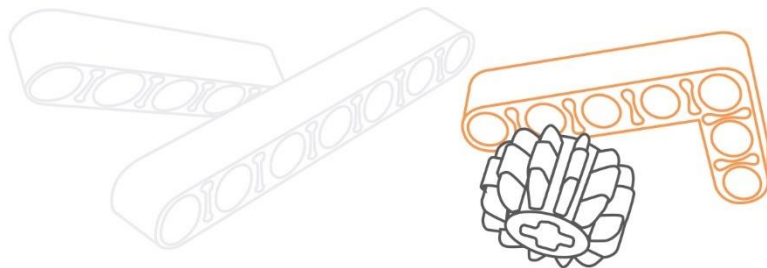
Test out your solution on the test space.

Did your robot wait for a remote code, then drive up to the rubble pile, scoop up some rubble, then bring it back to the drop-off zone and deliver it?



Hint!

If your rubble clean-up digger isn't working quite right, don't give up! Keep experimenting, testing and problem solving until you get your robot working like the #1 employee!



U4-2.5e Challenge up: Hazardous material removal

Many types of materials can be hazardous to people, such as biomedical waste, by-products from manufacturing or radioactive material. To keep people safe, these types of materials need to be safely moved and disposed of away from populated areas. Remotely operated vehicles with robotic arms are one way to handle this task.

For this challenge, you will need to build and program the EdRoboClaw to use its articulated arm to carry away some hazardous material, placing it into a pre-designated area.

What to do

To complete this project, you will need to build the EdRoboClaw, create your test space, and program your EdRoboClaw using EdScratch.

Test-space set-up

You need to create the test-space you will use to run your hazardous waste removal program. Create a 'disposal zone' by marking out an area, such as a square, with black lines on a white background surface. The inside of the 'disposal zone' is where the hazardous material will need to be dropped off. You will also need something to represent the hazardous waste.



Hint!

Make sure the robot's arm can pick up and carry whatever is representing the hazardous waste.

The EdScratch program

You will need to write two different programs, one for the top Edison and one for the bottom Edison in the EdRoboClaw.

Your programs need to control the EdRoboClaw so that the robot will pick up the hazardous waste and carry it to the disposal zone where it will drop off the waste. Your robot should not enter the zone, but its arm can reach into the disposal area. Once the material is dropped off, the robot should retreat away from the zone.



Hint!

- You may want to start the robot with the claw already hovering open over the 'waste'.
- You can use messaging in your programs to have the two Edison robots communicate with each other.
- Pseudocode helps us plan, comments help us debug. Use them to help you plan and test your program!

The build

You will need to build the EdRoboClaw to use in your hazardous waste removal task.



Use this link

Go to meetedison.com/content/EdCreate/EdBuild-EdRoboClaw-instructions.pdf

This link will take you to the step-by-step instructions for building the EdRoboClaw.

Remember that each of the two robots will need to be programmed with one of your EdScratch programs. Be sure to get the right program into the right robot!

Try it out!

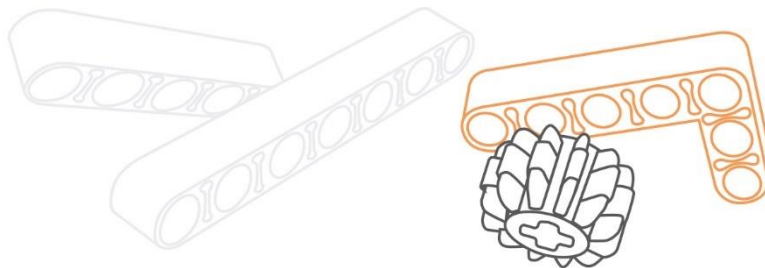
Test out your solution on the test space.

Did your robot pick up the hazardous waste, carry it to the disposal zone, drop the waste off inside the designated area, then retreat away from the disposal zone?



Hint!

If your robot isn't working quite right, don't give up! Keep experimenting, testing and problem solving until you get your robot to remove the hazardous waste and come back to safety!



U4-2.5f Challenge up: Homing pigeons

Homing pigeons are a type of domestic pigeon descended from the rock pigeon. In the wild, rock pigeons have an incredible innate ability: they can find their way home even from very long distances. People have bred homing pigeons for this special talent and use the birds to fly back to their homes, carrying messages along with them!

Can you get your Edison robot to act like a homing pigeon?

What to do

To complete this project, you will need to design a way to get Edison to act as much like a homing pigeon as you can, returning to a designated 'home base' from somewhere else. You will need to create a test space to use and a program to run in your Edison robot.

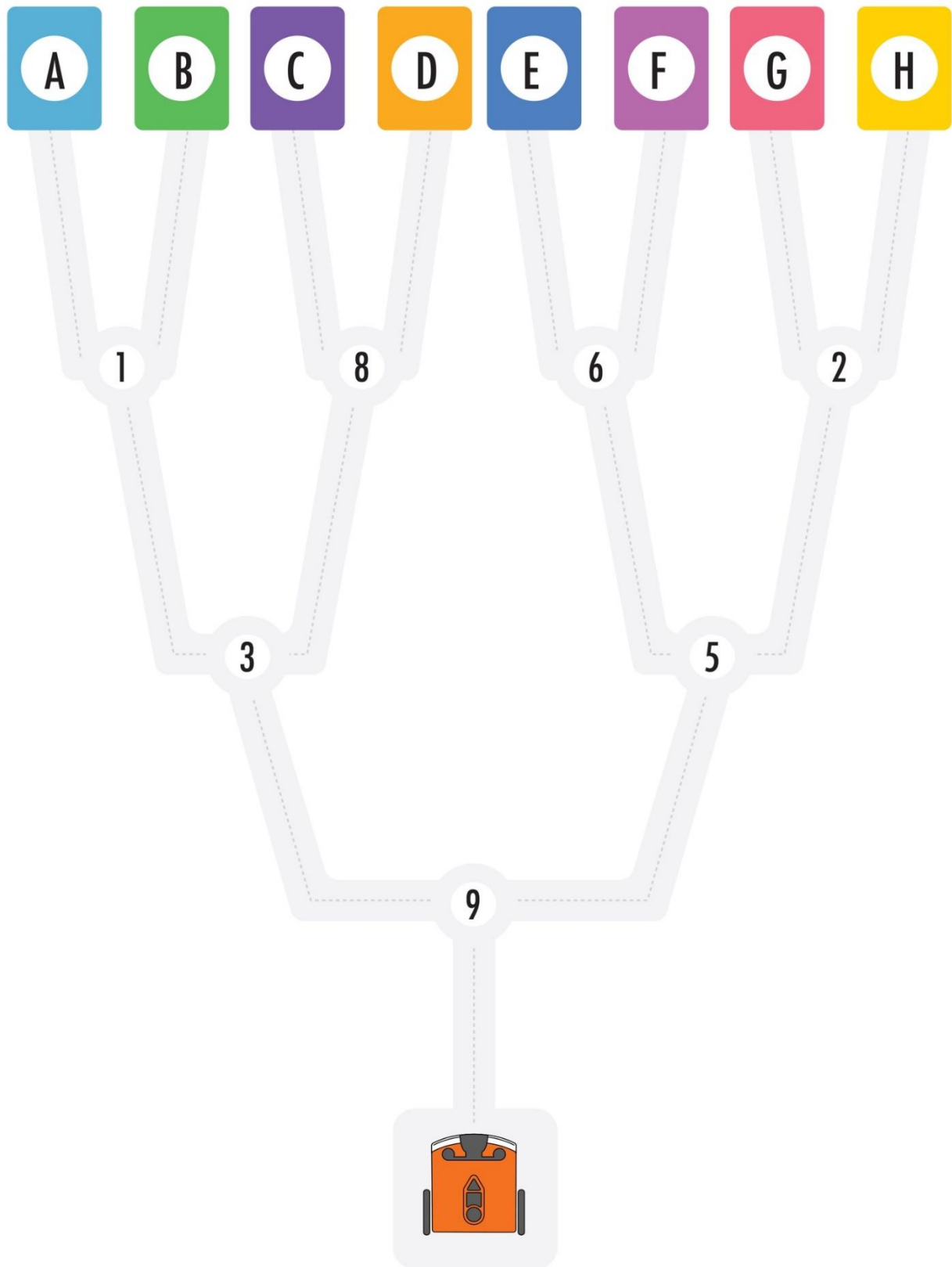
How you design your test space and how your program works is up to you. You can use any of Edison's sensors in any way you like, including line tracking, obstacle detection, and IR messaging.

































Hint!

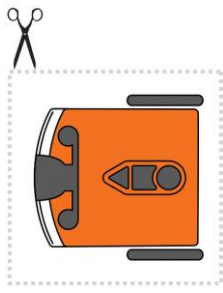
- Sensors like the obstacle detection beam and line tracker need to be turned on to work. The program also needs to tell Edison what event the sensors should look for and how to react to that event.
- Some of Edison's sensors, including obstacle detection, generate and store data when they detect specific events. It is good coding practice to clear the sensor data, especially when you use sensor events in conditionals which are nested inside loops. You don't want the data from a previous loop to affect the next loop! It's also wise to clear the sensor data at the start of a program, just in case the robot had old data stored from a previous program.
- Some of Edison's sensors, like IR messaging and obstacle detection, cannot be used at the same time. However, you can turn sensors on and off again in different spots in your program if you want.
- Use your test space design to your advantage. What could you build that would help Edison find and get back home?
- How could you signal the robots that it is time to start finding their way back home?
- Pseudocode helps us plan, comments help us debug. Use them to help you plan and test your program!

Activity sheet U4-1: If-else maze

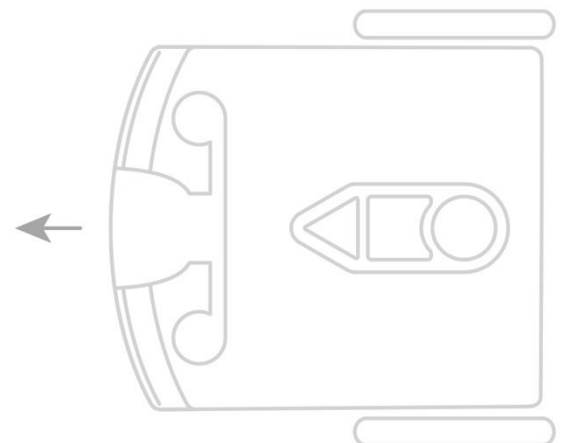
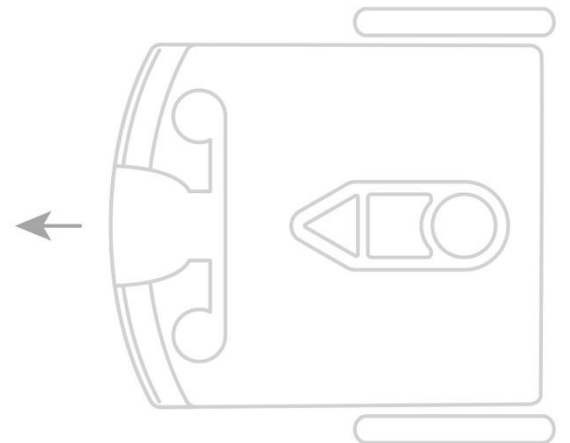
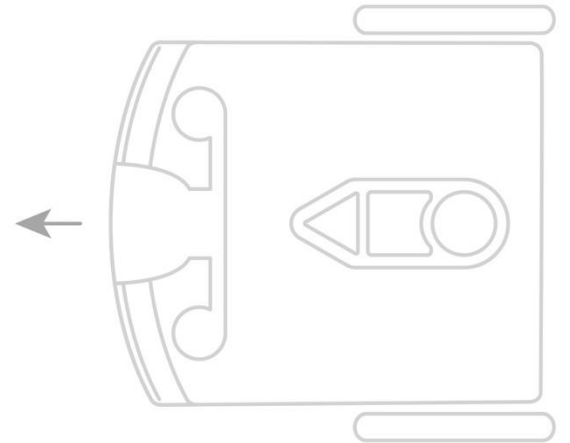
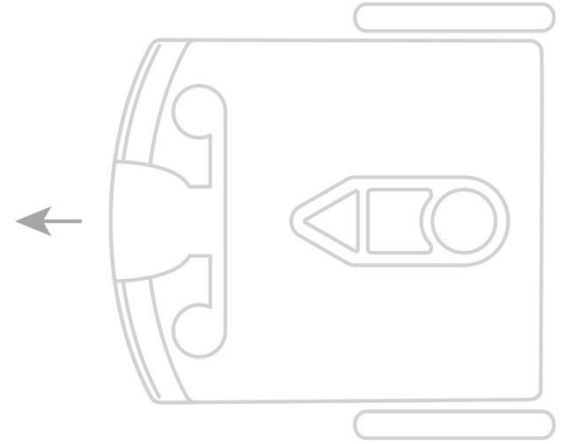


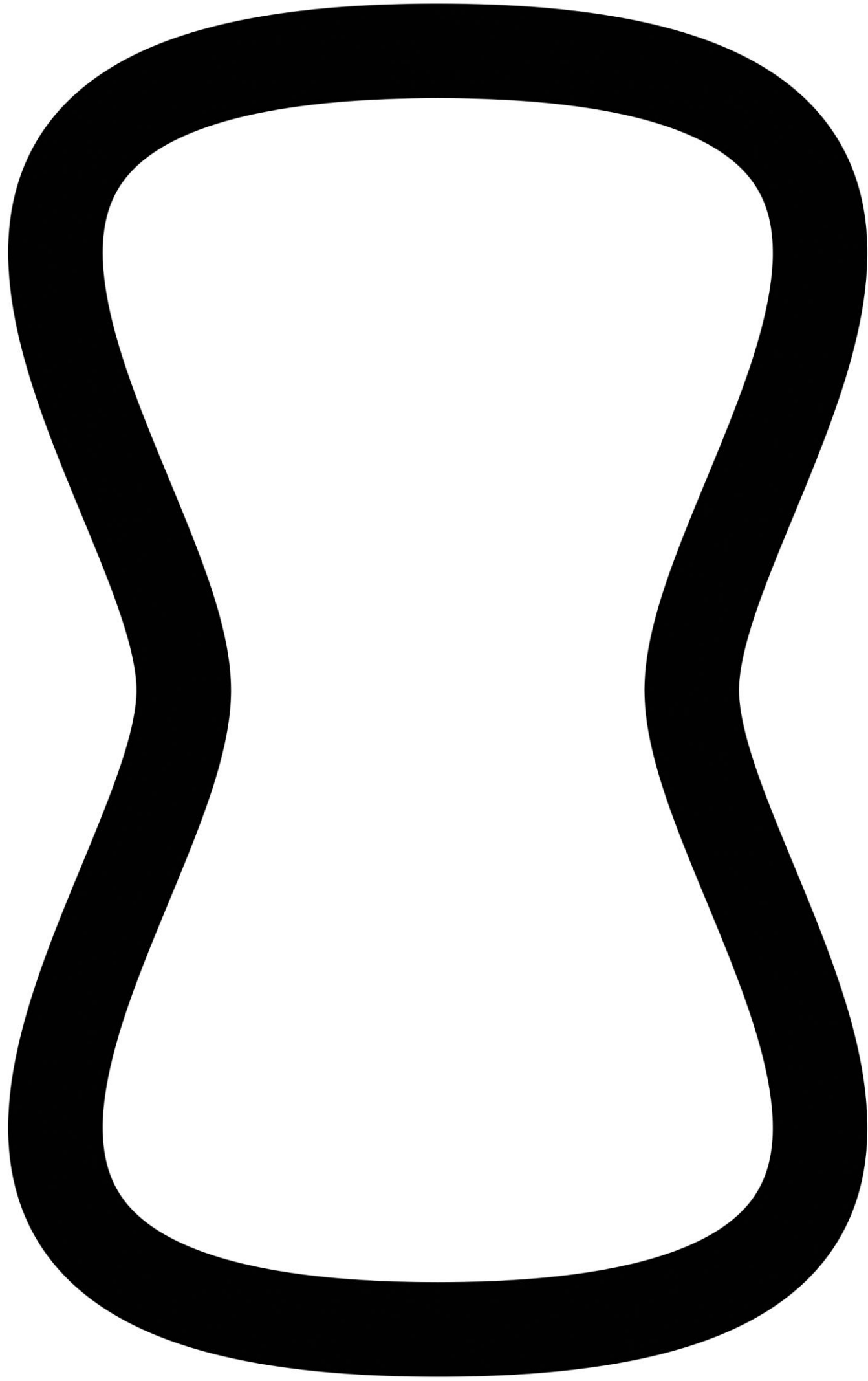
Activity sheet U4-2: Pseudocode step-by-step



Activity sheet U4-3: Line tracker test zone





Activity sheet U4-5: If line, go right

Place obstacle over here.

The diagram shows a robot at the bottom, represented by a rectangle with a triangle on top and two vertical bars on the sides. An arrow points upwards from the robot towards a large black square obstacle. Above the black square is a dashed rectangular area containing a dashed line that starts with a semi-circle, then goes right, then up, then left, and finally right again. A dashed arrow points left from the end of this path. Above the dashed area is a horizontal line with a black and orange diagonal pattern. To the right of the dashed area, five upward-pointing arrows are shown, with the text 'Place obstacle over here.' above them. To the right of the black square, five rightward-pointing arrows are shown, with the text 'Place obstacle over here.' to their right. The entire scene is framed by orange corner markers at the top-left, top-right, bottom-left, and bottom-right.

Activity sheet U4-6: Programmable TV remote codes



TV/DVD remote control code # 0



TV/DVD remote control code # 1



TV/DVD remote control code # 2



TV/DVD remote control code # 3



TV/DVD remote control code # 4



TV/DVD remote control code # 5



TV/DVD remote control code # 6



TV/DVD remote control code # 7