EdPy worksheets

Student worksheets and activity sheets

Contents

Lesson 1: Worksheet 1.1 – Meet Edison

Edison is a small programmable robot. Edison uses sensors and motors to interact with the world. You can also use Edison with LEGO bricks to build all kinds of things.

Look at the images below to become familiar with Edison's sensors, buttons and switches.
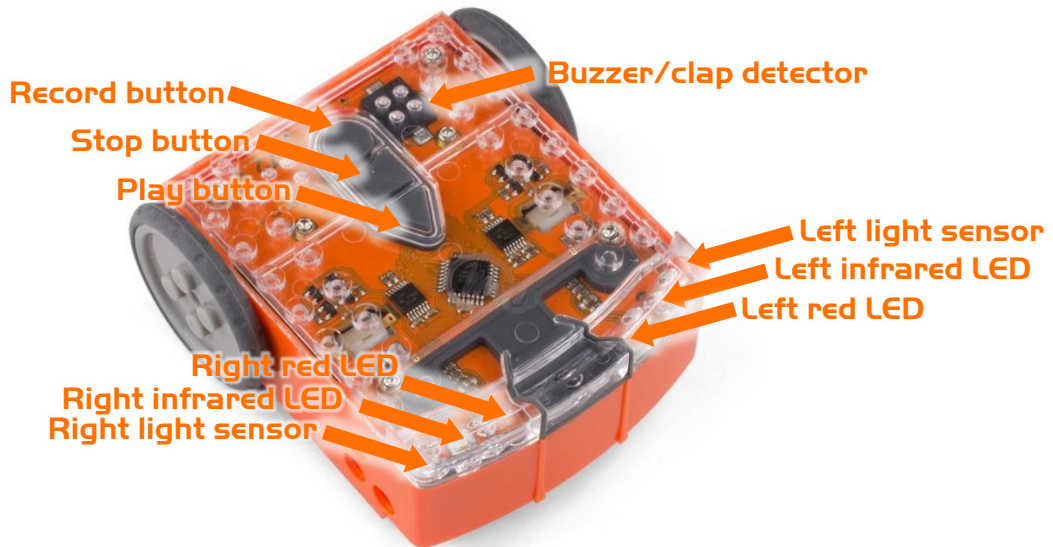
This is the top of Edison.

Play (triangle) button – Run program
Stop (square) button – Stop program
Record (round) button – 1 press = download a program, 3 presses = scan barcode

Edison's line tracking sensor is made up of two parts: a red LED light and a light sensor.

The line tracking sensor can also read special barcodes that activate pre-set programs.

This is the bottom of Edison.

You will also use the EdComm cable to download your programs to Edison. It connects to the headphone socket on your computer.

This is the EdComm programming cable.

Lesson 1: Worksheet 1.2 – Barcode programming

Edison comes with pre-loaded programs which are activated by driving over special barcodes.

Reading the barcode

1. Place Edison facing the barcode on the right side, above the arrow
2. Press the record (round) button three times
3. Edison will drive forward and scan the barcode
4. Press the play (triangle) button to run the program

Your turn:

For each barcode, describe what actions the robot takes and why it does so.

_____

_____

_____

_____

Barcode - Clap controlled driving

_____

_____

_____

_____

Barcode – Avoid obstacles

_____

_____

_____

_____

Barcode – Line tracking

_____

_____

_____

_____

Barcode – Follow torch

Lesson 1: Worksheet 1.3 – Meet the EdPy app

In this activity, you will get to know the EdPy app, the software we will use to program the Edison robot.

To get familiar with the EdPy app and programming, try opening up some example programs. Investigate how some of the functions work by searching in the 'Documentation' window. Everything you need to know about the EdPy app commands can be found in the documentation section. Also, try using the 'Line Help' to discover more about each

function.

Your turn:

1. What should you change the following line to if you are using an Edison Version 1?

   Ed.EdisonVersion = Ed.V2   _____

2. How many input parameters does each of the following commands have?

   Ed.PlayBeep()   _____

   Ed.TimeWait()   _____

   Ed.LeftLed()   _____

Ed.DriveRightMotor()   _____

3. You can change the Setup code to use inches instead of centimetres as your unit of measurement throughout the program. If you did this, how should that line of the Setup code be written?

_____

_____

4. If there are errors in your code after you have clicked the 'Check Code' button, in which window will the errors appear?

_____

_____

Lesson 1: Worksheet 1.4 – Downloading a test program

Open the program called 'Test_Program' from the 'Examples' window in EdPy.

This is what the Test Program looks like:

```
Test_Program                    ✕
1
2   #-------------Setup--------------
3
4   import Ed
5
6   Ed.EdisonVersion = Ed.V2
7
8   Ed.DistanceUnits = Ed.TIME
9   Ed.Tempo = Ed.TEMPO_MEDIUM
10
11  #--------Your code below-----------
12
13
14
15 ▾ while True:
16      Ed.PlayBeep()
17      Ed.LeftLed(Ed.OFF)
18      Ed.RightLed(Ed.ON)
19      Ed.Drive(Ed.SPIN_RIGHT, 5, 350)
20      Ed.TimeWait(20, Ed.TIME_MILLISECONDS)
21      Ed.PlayBeep()
22      Ed.LeftLed(Ed.ON)
23      Ed.RightLed(Ed.OFF)
24      Ed.Drive(Ed.SPIN_LEFT, 5, 350)
25      Ed.TimeWait(20, Ed.TIME_MILLISECONDS)
26
```

Edison looks at each line of the program one at a time, then does what the line says.

There are some lines Edison will skip, however.

Look at line 2, which starts with a '#' (hash) character. When a line starts with this character, it is called a 'comment line.' Edison will ignore any characters that come after the '#' on a line and move onto the next line. In programming, we use comment lines to document our code to help keep track of things and so that other people can understand the program.

Download the program to Edison

To download a program to Edison, connect the EdComm cable to the headphone socket on the computer and **turn up the volume to full**. Plug the other end of the EdComm cable into Edison as shown.

To download the test program, follow these steps:

1. Turn Edison on, then press Edison's record (round) button once
2. Connect Edison to the computer using the EdComm cable and confirm the volume is turned up to full
3. Press the 'Program Edison' button in the upper right corner of the EdPy app
4. Follow the steps in the pop-up window, then press 'Program Edison'

Once the program finishes downloading, unplug the EdComm cable. Press the play (triangle) button once to run the program.

Your turn:

1. What did the robot do when you pressed the play button?

_____

_____

_____

_____

2. Look at the Python commands in the program and think about what Edison did when you played the program. Describe how they relate to each other.

_____

_____

_____

_____

3. Explain how the program gets from the computer to the robot.

_____

_____

_____

_____

Lesson 2: Worksheet 2.1 – Drive the robot forward

In this activity, you need to write a program to drive your Edison robot forward.

Look at the following program:

```
1
2   #------------Setup---------------
3
4   import Ed
5
6   Ed.EdisonVersion = Ed.V2
7
8   Ed.DistanceUnits = Ed.CM
9   Ed.Tempo = Ed.TEMPO_MEDIUM
10
11  #--------Your code below----------
12
13  Ed.Drive(Ed.FORWARD,Ed.SPEED_5,8)
14
```

Step 1: Start writing the above program into the EdPy app by typing 'Ed' into line 13.

As you begin typing 'Ed' on line 13, you will see a prompt box pop up. The prompt box shows a list of possible commands for you to select. This is a feature of the EdPy app called 'command line completion', and it makes it quicker for you to program.

Step 2: Type 'Ed.Drive(' into line 13, and select the 'Ed.Drive()' function.

Ed.Drive() is a function in Python that's being imported from the Edison 'Ed' library module by the Setup code.

A function is a piece of code that performs a particular role or job, depending on which parameters are input. All the functions that are imported from the 'Ed' library must start with 'Ed.' This tells the program which library to go to in order to find that function.

Step 3: Fill in the input parameters.

When a function has input parameters, you need to enter a value for each one.

The Drive() function has three input parameters:
- **direction** – the direction that Edison will drive
- **speed** – the speed at which Edison will drive
- **distance** – the number of distance units Edison will travel

Different input parameters take different values. For example, 'speed' takes Ed.SPEED_ a number from 1 to 10 (10 is the maximum).

The type of distance units is controlled by the constant that 'Ed.DistanceUnits' is set to in the Setup code. There are three distance units that you can use:

- Centimetres, written as Ed.CM
- Inches, written as Ed.INCH
- Time, written as Ed.TIME

  Note: If you have an Edison V1 robot, you must use time, so make sure that Ed.DistanceUnits = Ed.TIME. The time distance unit is milliseconds, which means that to drive for 2 seconds, you need to set the distance input parameter to 2000 (since 2000 milliseconds = 2 seconds).

Step 4: Check your program.

Let's take a closer look at the drive function and parameters in this program.


```
13  Ed.Drive(Ed.FORWARD,Ed.SPEED_5,8)
```
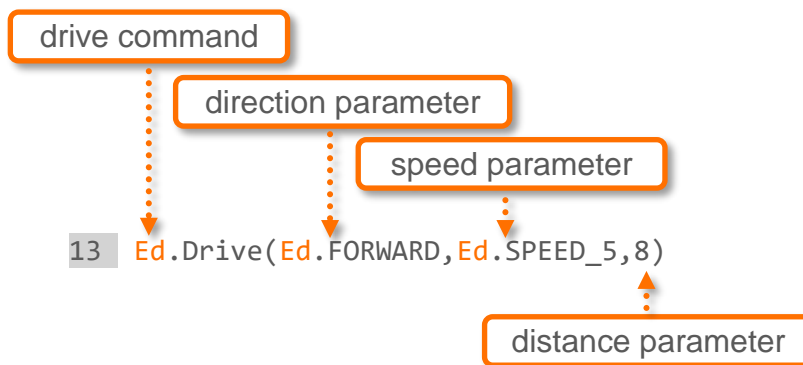
Once you have written the full program, click the 'Check Code' button and look at the 'Compiler Output' window to make sure you haven't made any errors while typing.

Typo-style errors are called 'syntax errors.' If you type a word that isn't a part of the EdPy program's language, also called syntax, then the EdPy compiler can't understand what it is meant to do. This creates a syntax error.

If there are any errors in your code, fix them, so your code matches the example.

Step 5: Download and test your program.

Once you check that your code has no errors, download your program to your Edison.

1. Turn Edison on, then press Edison's record (round) button once
2. Connect Edison to the computer using the EdComm cable and confirm the volume is turned up to full
3. Press the 'Program Edison' button in the upper right corner of the EdPy app
4. Follow the steps in the pop-up window, then press 'Program Edison'

Once the program downloads, unplug the EdComm cable. Use activity sheet 2.1 or coloured tape to mark 'start' and 'finish' lines on a desk or the floor as a test area for your

program. Press the play (triangle) button once to run the program and watch what happens.

Step 6: Experiment with your program.

Measure the distance between your start line and finish line. Try to modify your program to make your Edison robot finish driving just before the finish line. Experiment to see what works.

Your turn:

1. What constant did you have 'Ed.DistanceUnits' set to in the Setup code?

    _____

2. What number did you need to enter as the distance input parameter to make Edison drive from the start to the finish line?

    _____

3. Experiment with driving the Edison robot at different speeds. What does the robot do when you drive Edison at speed 10? Do you notice any changes in the accuracy of Edison when it is driving at speed 10?

_____

_____

_____

_____

Lesson 2: Worksheet 2.2 – Drive the robot backwards

In this activity, you need to write a program to drive your Edison robot backwards.

Whenever you write a program for Edison in EdPy, you always follow the same basic steps:

- **Step 1:** Check the Setup code is using the constants you want.
- **Step 2:** Write the program, selecting the functions you want to use, and filling in the input parameters with the values you want.
- **Step 3:** Check your program for errors using the 'Check Code' button.
- **Step 4:** Download and test your program using your Edison robot.

Remember, if you have Edison V1 robot, make sure that Ed.DistanceUnits = Ed.TIME. The distance parameter for Ed.TIME is in milliseconds.

Your turn:

Task 1: Drive backwards

Write the following program:

```
1
2   #------------Setup----------------
3
4   import Ed
5
6   Ed.EdisonVersion = Ed.V2
7
8   Ed.DistanceUnits = Ed.CM
9   Ed.Tempo = Ed.TEMPO_MEDIUM
10
11  #--------Your code below----------
12
13  Ed.Drive(Ed.BACKWARD,Ed.SPEED_5,8)
14
```

Use activity sheet 2.1 or coloured tape to mark 'start' and 'finish' lines on a desk or the floor as a test area for your program. Experiment to see if you can get your Edison to drive backwards from the start to the finish line.

Task 2: Use the constant 'Ed.DISTANCE_UNLIMITED'

There are multiple ways of programming your Edison to drive forward and backwards. Another way is to use 'Ed.DISTANCE_UNLIMITED' for the distance parameter. This constant turns both of Edison's drive motors on.

Unlike when you use a number value for the distance parameter, Ed.DISTANCE_UNLIMITED doesn't specify an exact value after which the motors will stop. Instead, it turns the motors on and then moves on to the next line of code. A stop of the motors will be needed later in the code.

Using Ed.DISTANCE_UNLIMITED can be useful when you want to write a program where the motors only stop once some other event occurs, for example, when an obstacle is detected.

Look at the following program:

```
1
2   #-------------Setup----------------
3
4   import Ed
5
6   Ed.EdisonVersion = Ed.V2
7
8   Ed.DistanceUnits = Ed.CM
9   Ed.Tempo = Ed.TEMPO_MEDIUM
10
11  #--------Your code below-----------
12
13  Ed.Drive(Ed.BACKWARD, Ed.Ed.SPEED_6, Ed.DISTANCE_UNLIMITED)
14  Ed.TimeWait(200, Ed.TIME_MILLISECONDS)
15  Ed.Drive(Ed.STOP, Ed.SPEED_10, 0)
16
```

This program turns on the Edison motors to drive backwards, then waits for 200 milliseconds before turning the motors off.

Write a program using the Ed.DISTANCE_UNLIMITED parameter to drive backwards. Use activity sheet 2.1 or coloured tape to mark 'start' and 'finish' lines on a desk or the floor as a test area for your program. Experiment with your program to see if you can get your Edison to drive backwards from the finish to the start line.

1. Set the speed in your program to Ed.SPEED_6. How many milliseconds do you need to input to the TimeWait() function to make Edison drive backwards from the finish to the start line?

   _____

2. Experiment with different speed and TimeWait() input parameters. What are the fastest and the slowest times you can make Edison drive backwards from the finish to the start line?

   Fastest: _____   Slowest: _____

Lesson 2: Worksheet 2.3 – Forwards, then backwards

In this activity, you need to write a program to drive your Edison robot forward, then backwards.

Remember to follow the four basic programming steps for Edison using EdPy:

- **Step 1:** Check the Setup code is using the constants you want.
- **Step 2:** Write the program, selecting the functions you want to use, and filling in the input parameters to the values you want.
- **Step 3:** Check your program for errors using the 'Check Code' button.
- **Step 4:** Download and test your program using your Edison robot.

Remember, if you have Edison V1 robot, make sure that Ed.DistanceUnits = Ed.TIME. The distance parameter for Ed.TIME is in milliseconds.

Your turn:

Task 1: Write the following program:

```
1
2   #------------Setup---------------
3
4   import Ed
5
6   Ed.EdisonVersion = Ed.V2
7
8   Ed.DistanceUnits = Ed.CM
9   Ed.Tempo = Ed.TEMPO_MEDIUM
10
11  #--------Your code below-----------
12
13  Ed.Drive(Ed.FORWARD,Ed.SPEED_5,8)
14  Ed.Drive(Ed.BACKWARD,Ed.SPEED_5,8)
15  |
```

Use activity sheet 2.1 or coloured tape to mark 'start' and 'finish' lines on a desk or the floor as a test area for your program.

1. What are the correct distance input parameter values so that Edison drives forward, then backwards between the start and the finish lines?

   Forward _____   Backwards _____

Task 2: Write a new program that makes Edison drive forward, then backwards between the start and the finish lines. This time, use the Ed.DISTANCE_UNLIMITED parameter and Ed.TimeWait() functions. Use Activity sheet 2.1 or coloured tape to mark 'start' and 'finish' lines on a desk or the floor as a test area for your program.

2. What does your new program look like? Write your code down below.

_____

_____

_____

_____

_____

_____

_____

_____

_____

Name_____

Lesson 2: Worksheet 2.4 – Expressions in Python

In this activity, you will learn about an important element of code we use when programming in Python: expressions.

What are expressions?

An expression is a question that can be resolved as being either 'true' or 'false.' For example: 'Is A less than B?' or 'Is A the same as B?'

In code, expressions are written using mathematical notations instead of words.

In the Setup code, you have seen the = notation being used. For example, Ed.DistanceUnits = Ed.CM. Using the 'A = B' notation means 'set A to the same value as B.'

Expressions also use notations. These are some of the basic notations in expressions we can use in Python:

Expression	Meaning
A == B	Is A the same as B?
A != B	Is A not equal to B?
A > B	Is A greater than B?
A >= B	Is A greater than or equal to B?
A < B	Is A less than B?
A <= B	Is A less than or equal to B?

Expressions compare the left side to the right side of the notation in the expression.

You can replace the 'A' and 'B' in the list of expressions above with any value or function that returns a value. You can also do maths to those values. For example, '(A + 2) > B' means 'Is A plus 2 greater than B?'

In code, expressions work in a specific order. When your expression includes maths to a value or calls a function, the expression will resolve the maths or function first. It will then compare the left side of the expression to the right side and resolve to either 'true' or 'false.'

What are expressions used for in Python?

Expressions are used along with other elements of code, such as 'while' loops and 'if' statements, to change the flow of code. These elements allow the code to move differently than just in the sequential flow of line 1 → line 2 → line 3.

Your turn:

Practice resolving expressions. First, write out what each expression means, then resolve it to either true or false.

If A = 2 and B = 4, what does each of the following expressions mean and what does each resolve to (true or false)?

1. (A*2) == B

    Meaning: _____

    Resolves to: _____

2. A >= B

    Meaning: _____

    Resolves to: _____

3. (A+A) != B

    Meaning: _____

    Resolves to: _____

4. (A-1) < (B-3)

    Meaning: _____

    Resolves to: _____

Lesson 2: Worksheet 2.5 – Keypad activated driving

In this activity, you need to write a program to drive your Edison robot forward only when either the round button or the triangle button is pressed. To do this, we will use expressions and the 'while' loop.

Look at the following program:

```
1
2   #------------Setup---------------
3
4   import Ed
5
6   Ed.EdisonVersion = Ed.V2
7
8   Ed.DistanceUnits = Ed.CM
9   Ed.Tempo = Ed.TEMPO_MEDIUM
10
11  #--------Your code below-----------
12  Ed.ReadKeypad()
13 ▾ while Ed.ReadKeypad() == Ed.KEYPAD_NONE:
14      pass
15  Ed.Drive(Ed.FORWARD, Ed.SPEED_6,8)
16
```

This program uses a 'while' loop with an expression.

In Python, a 'while' loop repeats a statement or group of statements while a given condition is TRUE. It tests the condition, which is written as an expression, before executing the loop body.

While the expression evaluates to TRUE, the program repeats the commands in the loop. When the expression evaluates to FALSE, the program moves on to the next line of code outside the loop.

Using indentation

Python uses indentation to group statements or commands together.

In Python, all the statements indented by the same number of character spaces are considered to be a single block of code.

Look at line 14 of the program. Because 'pass' is indented, it is inside of the loop. Line 15 in the program is not indented, however, so line 15 is not inside of the loop.

Functions and constants in this program

Ed.ReadKeyPad() – this function reads Edison's keypad state. In other words, it determines whether one of Edison's buttons has been pressed or not. Ed.ReadKeyPad() returns a value indicating which button has been pressed: Ed.KEYPAD_NONE, Ed.KEYPAD_TRIANGLE or Ed.KEYPAD_ROUND.

This function does not work for the square button. That's because the square button is designed only to be used to stop a program. The square button will always stop any program running when it is pressed.

Special note: using 'read' functions inside a loop

Some types of data get temporarily stored in Edison's memory. That's how the Ed.ReadKeyPad() function may read a button press from before the read function is called in your code.

In this program, we want to make sure the Ed.ReadKeyPad() in the while loop will wait until a button has been pressed and not consider any button presses that happen before the while loop. That's why we put Ed.ReadKeyPad() into the line above the while loop (line 12). This will clear any previous key presses before the loop.

You should always follow this process when using a 'read' function inside a loop.

Your turn:

Write the program.

Be sure that when you write the while loop, you include the colon and that you indent correctly:

while loop command		state of the keypad

	reads the state of the keypad	

	is the same as	

```
13    while Ed.ReadKeypad() == Ed.KEYPAD_NONE:
14        pass
```

colon (:) is required with a while loop

pass is required when a loop has no other statements
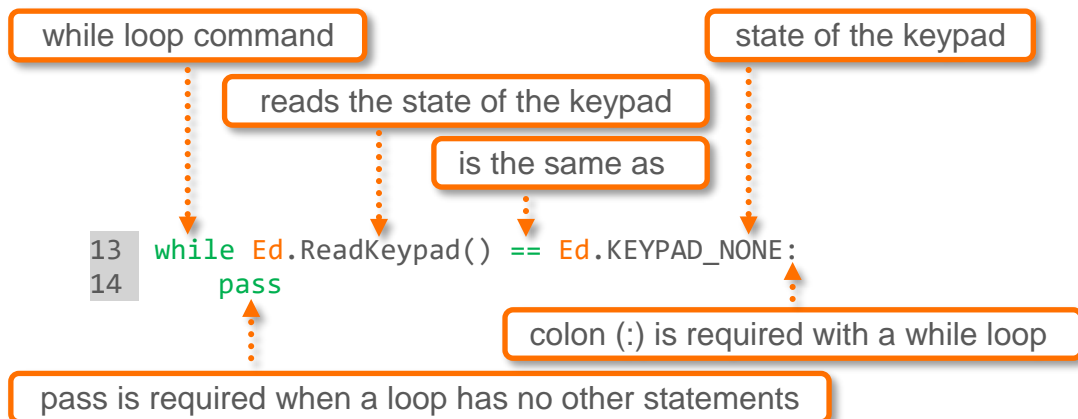
Download the program and press the triangle button to start the program. Wait a bit, then try pressing either the triangle or round button.

1. What did the robot do when you pressed the triangle or round button a few seconds after starting the program?

_____

_____

2. Run the program again and try pressing the square button instead of the round or triangle button. What happened? Why did this happen? Explain.

_____

_____

_____

_____

3. Now try adding some more code at the end of the program. The new code you write should make Edison drive backward after you press either the round or the triangle button again. In other words, your program should tell Edison to drive forward the first time a button is pressed, then backwards when a button is pressed again. Remember to include the colon and to indent your code inside the while loop. What does your new program look like? Write your code down below.

_____

_____

_____

_____

_____

_____

_____

_____

Lesson 2: Activity sheet 2.1

FINISH LINE

START LINE

Lesson 3: Worksheet 3.1 – Turn right

In this activity, you need to write a program that will make your Edison robot turn right.

Look at the following program:

```
1
2  #-------------Setup---------------
3
4  import Ed
5
6  Ed.EdisonVersion = Ed.V2
7
8  Ed.DistanceUnits = Ed.CM
9  Ed.Tempo = Ed.TEMPO_MEDIUM
10
11 #--------Your code below-----------
12 degreesToTurn = 90
13 Ed.Drive(Ed.SPIN_RIGHT,Ed.SPEED_6,degreesToTurn)
14
```

Look at line 13 of the program. Remember that the Ed.Drive function has three input parameters: direction, speed, and distance. In this program, the distance parameter is not a number but is 'degreesToTurn'.

This is a variable.

In Python, variables are reserved memory locations to store values. This means that when you create a variable, you reserve some space in the program's memory.

A variable represents a value that is set somewhere in your program.

Look at line 12 of the program. This is where the value of 'degreesToTurn' is being set. This is called assigning a value to a variable. In Python, the equal sign (=) is used to assign values to variables.

You can use variables to store values that are used in several places throughout a program. This can be very helpful, especially if the value of a variable changes. By using variables, you only need to make the change in one line of the code to adjust the value everywhere that variable is being used in your program.

Your turn:

Write the program, then download and run the program with your Edison. Use activity sheet 3.1 or coloured tape to mark 'start' and 'end' angle marks on a desk or the floor as a test area for your program.

1. Describe what the robot does and why it does this when you run the program.

_____

_____

_____

Add a new line to the end of your program (after line 13) and add the following code to your program:

```
Ed.Drive(Ed.SPIN_LEFT,Ed.SPEED_6,degreesToTurn)
```

Download and run the program with your Edison in your test area.

2. Describe what the robot does and why it does this when you run the updated program.

_____

_____

_____

Now edit your program so that your Edison will first turn 180 degrees to the right, then 180 degrees to the left.

Hint: Remember you can change the value of the degreesToTurn variable.

Download and run your updated program with your Edison in your test area to see if your change was successful.

3. What line or lines in your program did you change? Write down the updated line or lines.

_____

_____

_____

Variable names must follow certain rules in Python. For example, no special characters like # or $ are allowed. Try changing the name of the variable 'degreesToTurn'. Experiment with different possible names and use the 'Check Code' button to find which names are allowed and which are not.

4. Give two examples of illegal variable names you discovered.

_____

_____

Name_____

Lesson 3: Worksheet 3.2 – Turn left 180°

In this activity, you need to write two different programs to turn your Edison robot left exactly 180°.

Your turn:

Task 1: Turn left exactly 180°

Write a program that makes your Edison robot turn left exactly 180°.

Hint: Try using the program you used in worksheet 3.1 as a starting point.

Download your program and test it using activity sheet 3.1 or coloured tape to mark 'start' and 'end' angle marks on a desk or the floor as a test area for your program. Remember to experiment with your program. If your Edison doesn't turn exactly 180°, try adjusting your input parameters and test again.

1. What are the input parameters you used to make the robot turn exactly 180°? If you used a variable, include what value you assigned to that variable.

_____

_____

_____

Task 2: Turn exactly 180° using the Ed.DriveRightMotor() command

EdPy has a command called 'Ed.DriveRightMotor()' which makes only Edison's right motor move. If only the right motor moves, which way will Edison turn? Hold Edison in your hands and imitate what will happen if only the right motor moves.

Search for the Ed.DriveRightMotor() command in the Documentation window of the EdPy app to see how this function works.

Then write a program that makes your robot to turn left 180° using the Ed.DriveRightMotor() function.

2. What are the input parameters you used to make the robot turn left exactly 180° using the Ed.DriveRightMotor() command?

_____

_____

_____

Lesson 3: Worksheet 3.3 – Right turn, then left turn

In this activity, you need to write a program so that your Edison robot will turn when the triangle button is pressed.

Write the following program:

```
1
2    #------------Setup---------------
3
4    import Ed
5
6    Ed.EdisonVersion = Ed.V2
7
8    Ed.DistanceUnits = Ed.CM
9    Ed.Tempo = Ed.TEMPO_MEDIUM
10
11   #--------Your code below-----------
12   degreesToTurn = 90
13   Ed.ReadKeypad()
14   while Ed.ReadKeypad() != Ed.KEYPAD_TRIANGLE:
15       pass
16   Ed.Drive(Ed.SPIN_RIGHT,Ed.SPEED_6,degreesToTurn)
17
```

Download your program and test it using activity sheet 3.1 or coloured tape to mark 'start' and 'end' angle marks on a desk or the floor as a test area for your program.

Your turn:

Write a program to make the robot turn right exactly 90° when the triangle button is pressed once, then turn left exactly 270° when the triangle button is pressed a second time.

Remember to put Ed.ReadKeyPad() into the line above each 'while' loop to clear any previous key presses before the loop.

1. What does your program look like? Write your code down below.

_____

_____

_____

_____

_____

_____

Lesson 3: Worksheet 3.4 – Mini maze

In this activity, you need to write a program that will allow your Edison robot to successfully navigate through a maze.

Your turn:

Write a program so that your Edison robot will drive through the mini maze on activity sheet 3.2 when you hit the play (triangle) button.

To successfully complete the maze, you must:

- have Edison start from behind the 'start' line,
- have Edison stop after crossing the 'finish' line, and
- keep Edison inside the border lines of the maze.

Use the robot programming knowledge that you've gained so far to write a program which uses multiple functions to allow Edison to make it through the maze's turns.

Hints: | Ed.Drive() | Ed.SPIN_RIGHT | Ed.FORWARD | Ed.SPIN_LEFT |

1. Describe the sequence of moves your robot did to complete the maze.

_____

_____

_____

2. What did you find difficult about writing this program?

_____

_____

_____

_____

_____

Challenge 1: Race!

Who can get through the maze the fastest, without cheating?

Remember: your robot must start from behind the start line, stop after the finish line and must not drive over any border lines to win.

3. Who did you race? Who won the race?

Competitor: _____

Winner: _____

4. What was the winning robot's time through the maze?

_____

Challenge 2: Design your own maze

Design your own, more challenging maze with a few more turns for Edison to navigate. Write a program for Edison to complete the maze successfully. Or, exchange mazes with a partner and write a program to complete their maze successfully.

Remember: your robot must start from behind the start line, stop after the finish line and must not drive over any border lines to win.

5. Draw a small version of the maze you completed in the box.
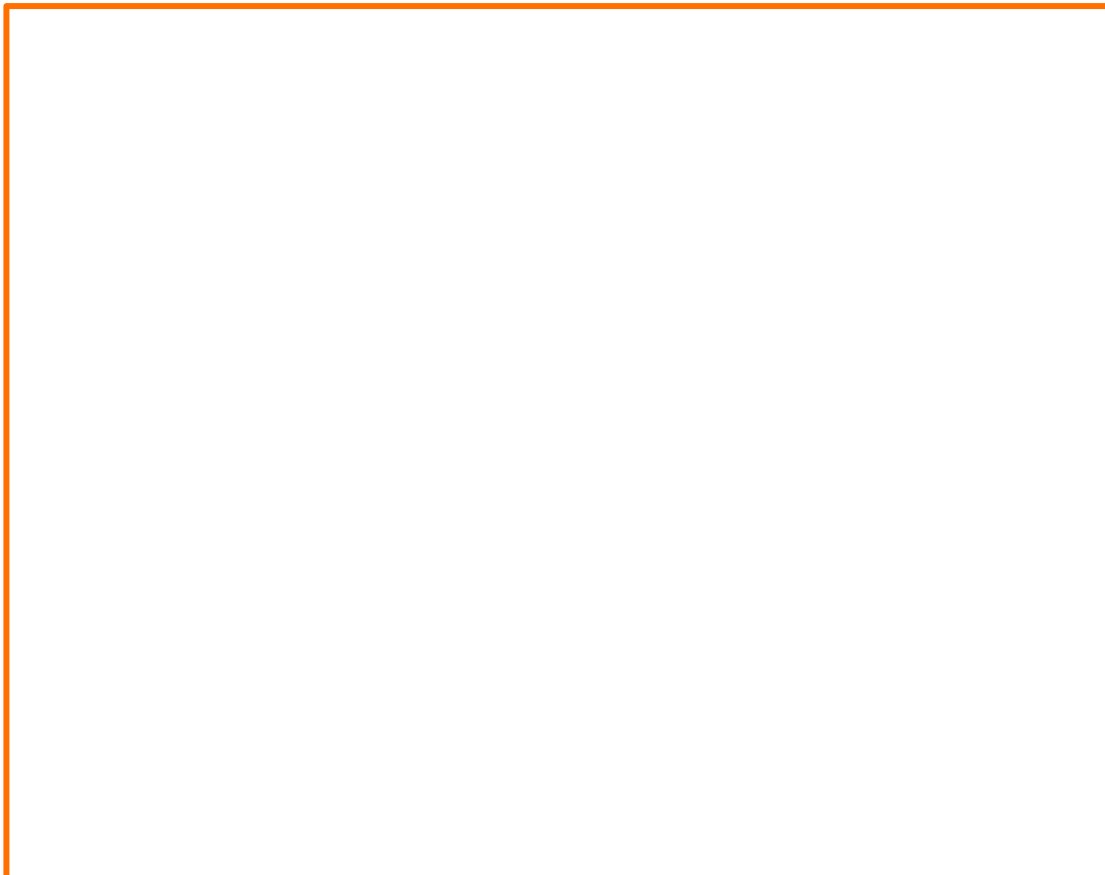
Lesson 3: Activity sheet 3.1 – Turning

Put your Edison robot onto the outline as shown. Always start your robot from the start (0°) marker.

START 0°

← Left

Right →

END 90°

END 180°

Lesson 3: Activity sheet 3.2 – Mini maze

FINISH LINE

START LINE
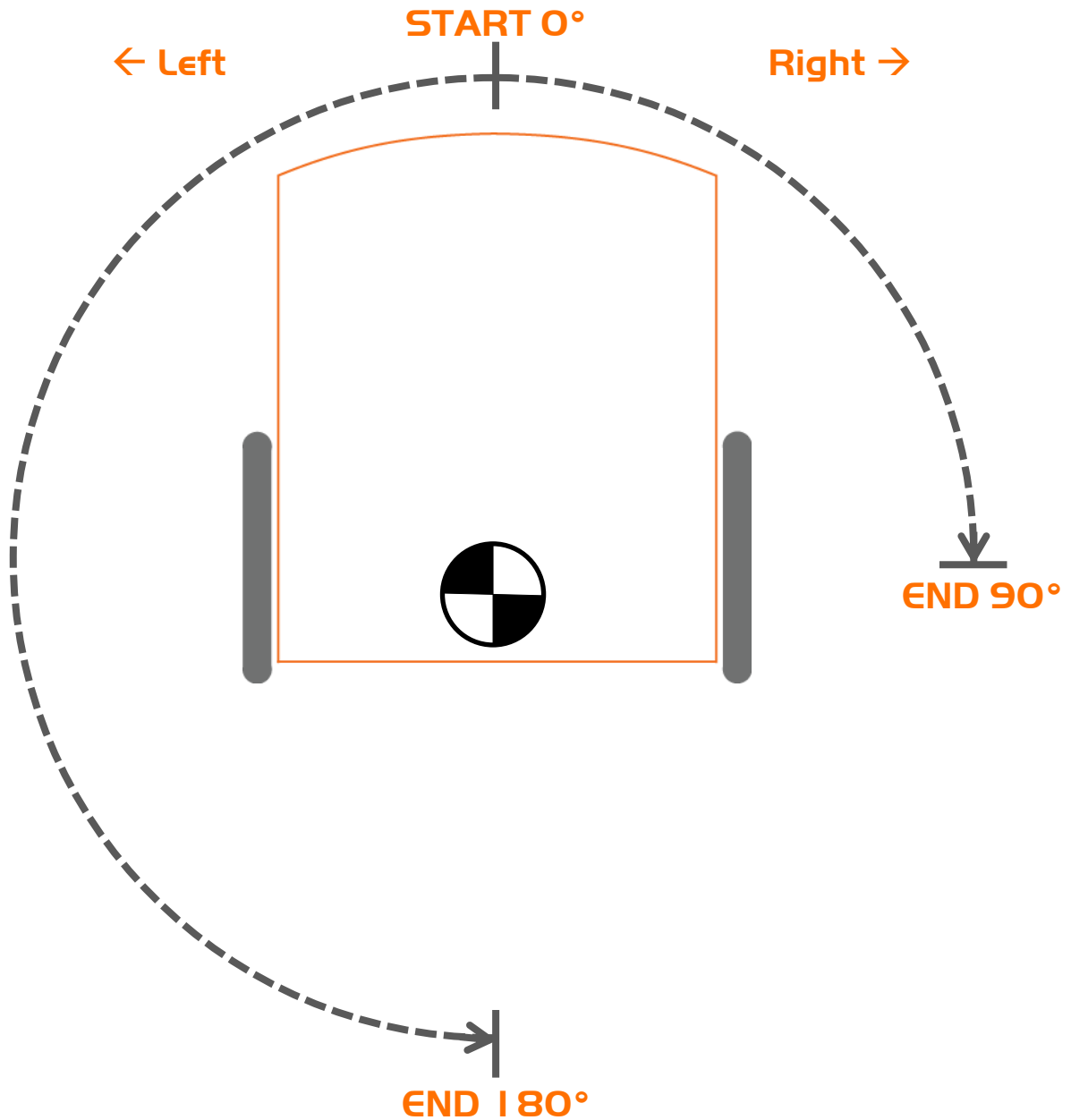
Lesson 4: Worksheet 4.1 – Drive in a square

In this activity, you need to write a program that will allow your Edison robot to drive in the shape of a square.

Your turn:

Write a program so that when your Edison robot drives, it makes a square. Use the commands you have already learned, including Ed.Drive() and a variable. Be sure your program ends with your Edison in the same spot it started.

Download your program and test it using activity sheet 4.1, placing your Edison at the 'start' point and following the lines. You can also make a larger square using coloured tape to mark the lines and a 'start' point on a desk or the floor.

1. What does your program look like? Write your code down below.

_____

_____

_____

_____

_____

_____

_____

2. How many function calls did you use in this program?

_____

3. Do you have any duplicate lines of code in your program? If so, what are they and how many times did you use each?

_____

_____

_____

_____

Lesson 4: Worksheet 4.2 – Use a loop to drive in a square

In this activity, you need to write a different program that will allow your Edison robot to drive in the shape of a square.

Using worksheet 4.1, you wrote a program that used the same commands multiple times. You needed to use Ed.Drive() with a direction parameter of Ed.FORWARD four times, once for each side of the square. You also needed to use Ed.Drive() with a direction parameter of Ed.SPIN_LEFT four times, to turn each corner.

Did you find writing the same commands many times a bit boring?

Repeating the same commands over and over is no problem for a computer, but writing out a program this way isn't very efficient. Instead, it is better to use a loop structure.

Watch Mark Zuckerberg, who created Facebook, explain the concept of loops when programming:
https://www.youtube.com/watch?v=hYvcoRkAkOU

Who knew being a great coder could make you one of the world's youngest billionaires?
Mark Zuckerberg's net worth is estimated to be more than 70.5 billion US dollars!

We can write a program to make Edison drive in a square with less code by using a 'for' loop. This will make writing the program more efficient. Since we will need to use fewer lines of code, using the 'for' loop will also help reduce the likelihood of mistyping and having a syntax error in the program.

The 'for' loop and 'range()' function in Python

In Python, a 'for' loop is a control structure which can be used to repeat sets of commands or statements any number of times.

Using a 'for' loop allows you to repeat (also called 'iterate over') a block of statements as many times as you like.

The 'for' loop often goes together with the 'range()' function in Python.

The range() function returns a set of values within a certain range.

In EdPy, range() only has one input parameter. That input parameter determines the upper limit of the set and the lower limit is always 0.

The range() function returns values from 0 to (input parameter – 1).

Example:

range(4) → there are 4 values in the set: 0, 1, 2, 3.

Let's look at an example:

'for' loop command

variable for the range value

range function and number of values

play a beep

```
13  for i in range(4):
14      Ed.PlayBeep()
15      Ed.TimeWait(1,Ed.TIME_SECONDS)
```

wait for 1 second
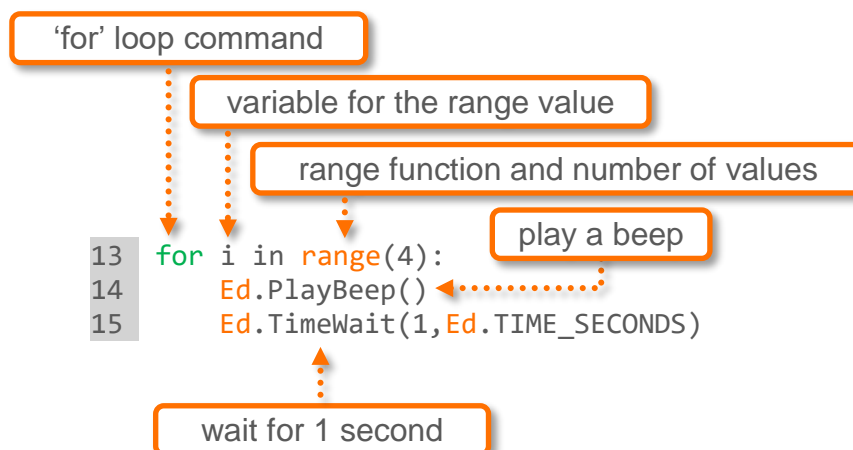
In this example, the 'for' loop iterates four times, causing the variable 'i' to have the values 0, 1, 2 and 3. Each time it iterates, the loop executes the statement block consisting of Ed.PlayBeep() and Ed.TimeWait().

The result? The beep is played four times with a one-second delay in between each beep.

Your turn:

Write a program using the 'for' loop and the 'range()' function so that when your Edison robot drives, it makes a square. You should be able to complete the program using just two Ed.Drive() functions, one for forward and one for spin.

Don't forget to include a colon and proper indentation inside your loop.

Download your program and test it using activity sheet 4.1, placing your Edison at the 'start' point and following the lines. Be sure your program ends with your Edison in the same spot it started. You can also make a larger square using coloured tape to mark the lines and a 'start' point on a desk or the floor.

1. What does your program look like? Write your code down below.

_____

_____

_____

_____

_____

_____

Lesson 4: Worksheet 4.3 – Drive in a triangle and a hexagon

In this activity, you need to write two different programs to get your Edison robot to drive in the shape of a triangle, and then in a hexagon.

Your turn:

Task 1: Drive in a triangle

Write a program so that when your Edison robot drives, it makes a triangle.

Download your program and test it using activity sheet 4.2, placing your Edison at the 'start' point and following the lines. You can also make a larger triangle using coloured tape to mark the lines and a 'start' point on a desk or the floor.

1. How many times did your 'for' loop execute for your triangle shape?

    _____

Task 2: Drive in a hexagon

Write a program so that when your Edison robot drives, it makes a hexagon.

Download your program and test it using activity sheet 4.3, placing your Edison at the 'start' point and following the lines. You can also make a larger hexagon using coloured tape to mark the lines and a 'start' point on a desk or the floor.

2. How many times did your 'for' loop execute for your hexagon shape?

    _____

3. You should see a pattern emerging between the number of sides of the shape and the number of times the 'for' loop executes. Describe that pattern.

_____

_____

4. How many times you would need the 'for' loop to execute to draw a regular (meaning that all sides are equal) 12-sided shape?

    _____

Lesson 4: Worksheet 4.4 – Challenge! Drive in a circle

In this activity, you need to write a program to get your Edison robot to drive in a circle.

Your turn:

Write a program where your Edison robot drives in a circle. Your Edison needs to drive in the shape of a circle, not just spin in one spot.

Download your program and test it using activity sheet 4.4, placing your Edison at the 'start' point and following the line. You can also make your robot drive around any circular object, like a round rubbish bin or a round table.

Hint: A shape with many hundreds of very small sides can closely approximate a circle.

1. How many times does your loop execute to make your shape a circle?

_____

_____

_____

2. How far does your robot travel each time you execute your loop?

_____

_____

3. Does your robot drive in a perfect circle? If not, can you think of a reason why not?

_____

_____

_____

Optional challenge: Draw your shape

Attach a crayon or coloured marker to your robot using some combination of LEGO block pieces or masking tape. Place your Edison on a piece of paper and run your circle program. Watch as the coloured marker draws your shape as the robot moves. See if your Edison draws a true circle or not.

Lesson 4: Activity sheet 4.1

Starting point

Lesson 4: Activity sheet 4.2

Starting point

Lesson 4: Activity sheet 4.3

Starting point
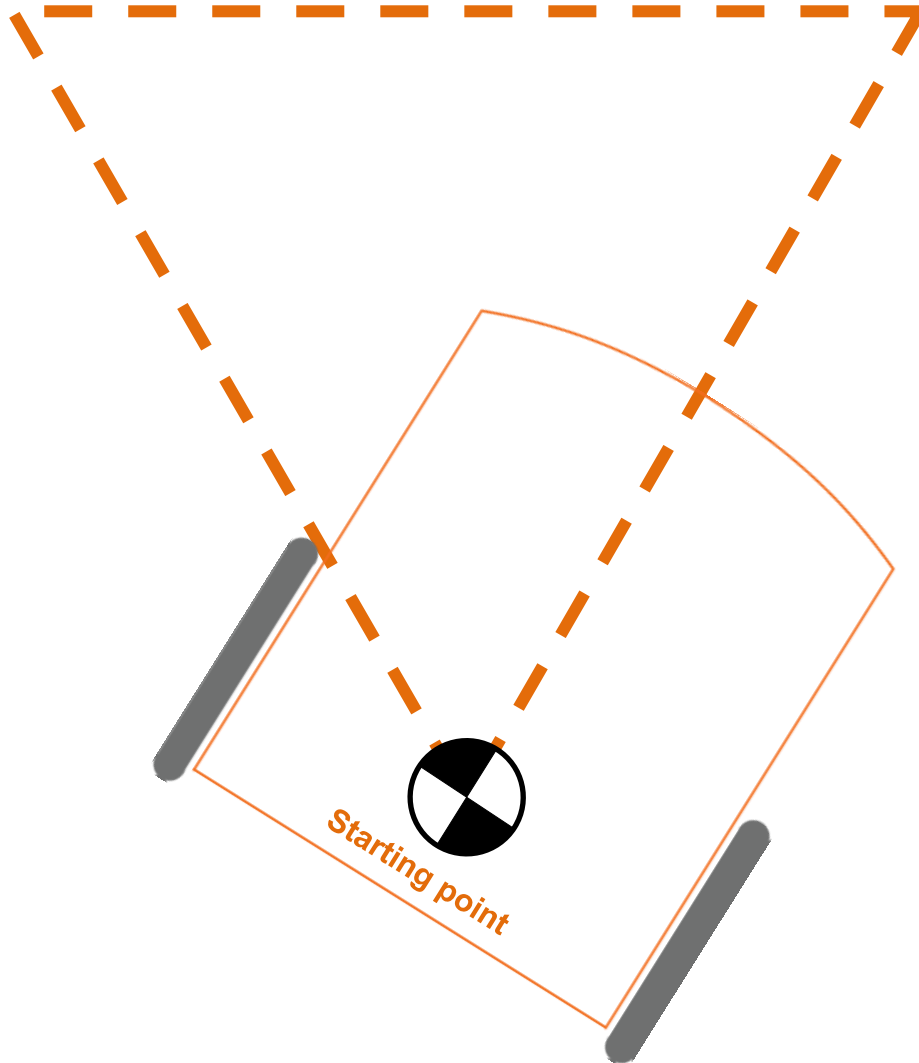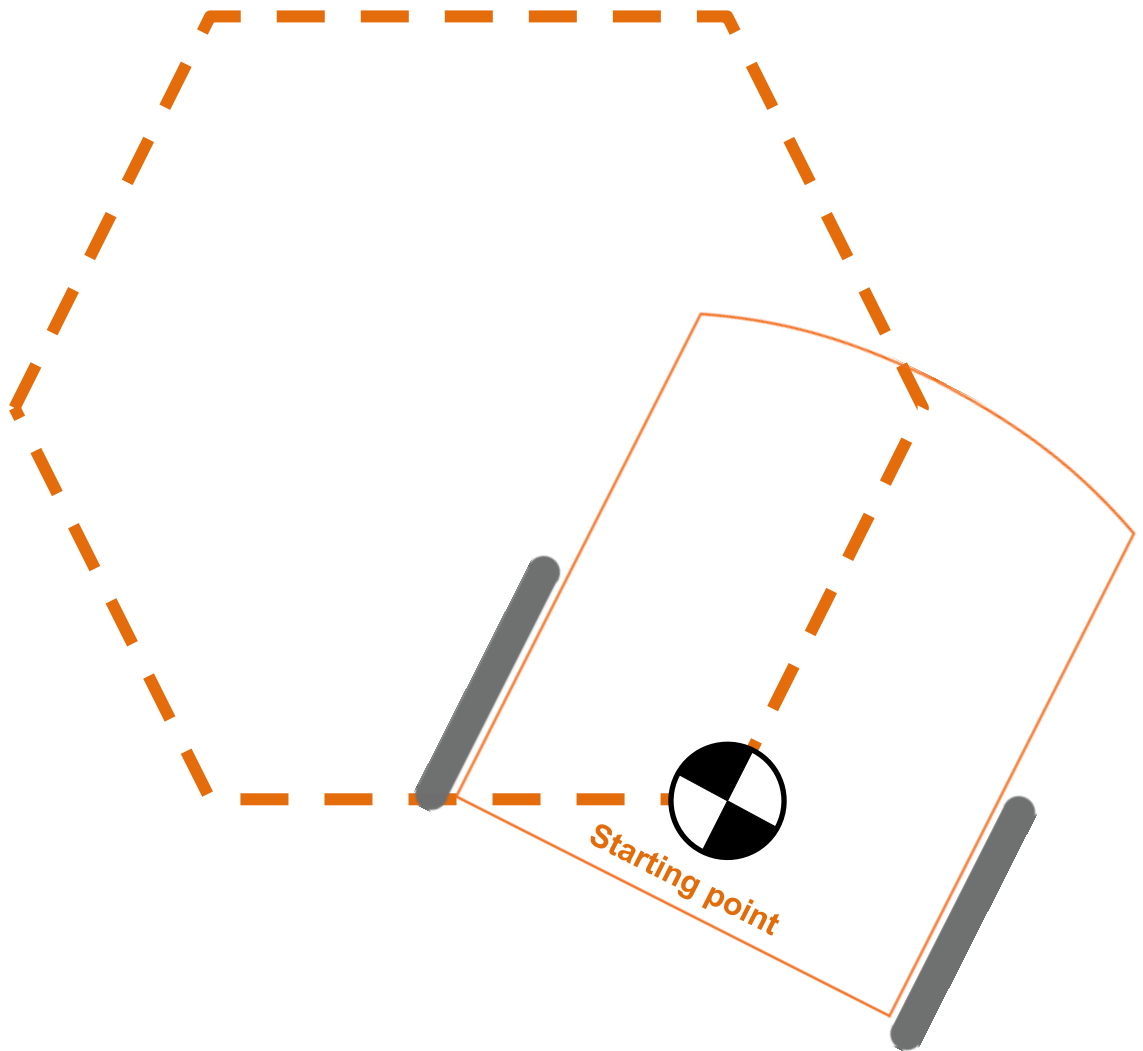
Lesson 4: Activity sheet 4.4

Starting point
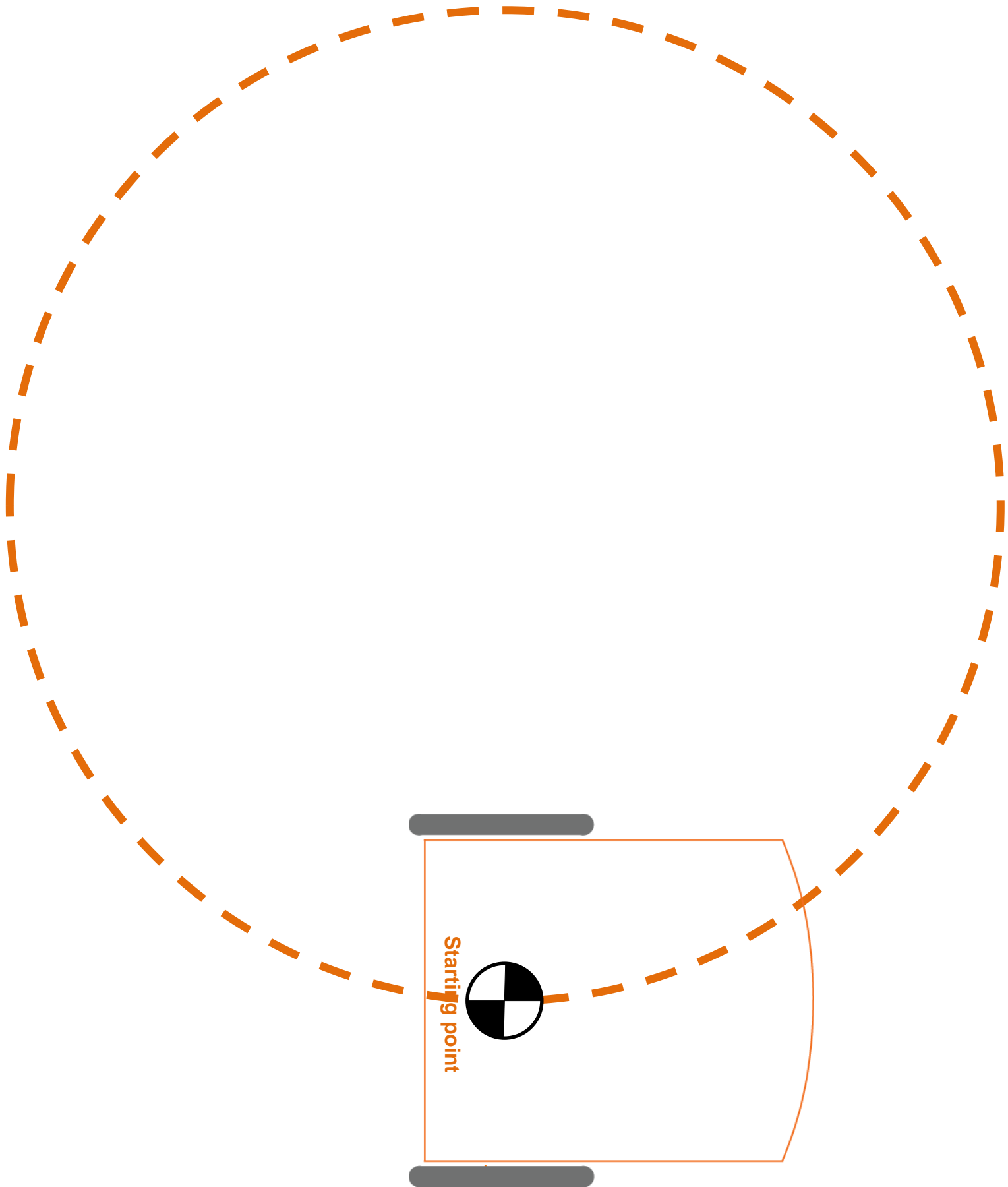
Lesson 5: Worksheet 5.1 – Play tones

In this activity, you need to write a program to make Edison play a musical note and learn how Edison plays sounds in a program.

You can play individual musical notes through Edison's small speaker using the Ed.PlayTone() function in EdPy.

The Ed.PlayTone() function takes two input parameters: the note and the duration. The note determines what note to play and the duration determines the given length of time the note should be played.

This list includes the possible parameter values:

note

Parameter input options	Plays musical note
Ed.NOTE_A_6	low A
Ed.NOTE_A_SHARP_6	low A sharp
Ed.NOTE_B_6	low B
Ed.NOTE_C_7	C
Ed.NOTE_C_SHARP_7	C sharp
Ed.NOTE_D_7	D
Ed.NOTE_D_SHARP_7	D sharp
Ed.NOTE_E_7	E
Ed.NOTE_F_7	F
Ed.NOTE_F_SHARP_7	F sharp
Ed.NOTE_G_7	G
Ed.NOTE_G_SHARP_7	G sharp
Ed.NOTE_A_7	A
Ed.NOTE_A_SHARP_7	A sharp
Ed.NOTE_B_7	B
Ed.NOTE_C_8	high C
Ed.NOTE_REST	rest

duration

Parameter input options	Plays note for
Ed.NOTE_SIXTEENTH	125 milliseconds
Ed.NOTE_EIGHTH	250 milliseconds
Ed.NOTE_QUARTER	500 milliseconds
Ed.NOTE_HALF	1,000 milliseconds
Ed.NOTE_WHOLE	2,000 milliseconds

Let's take a closer look at the play tone function in a program:

play tone command

note parameter

duration parameter

```
13  Ed.PlayTone(Ed.NOTE_B_6,Ed.NOTE_HALF)
```
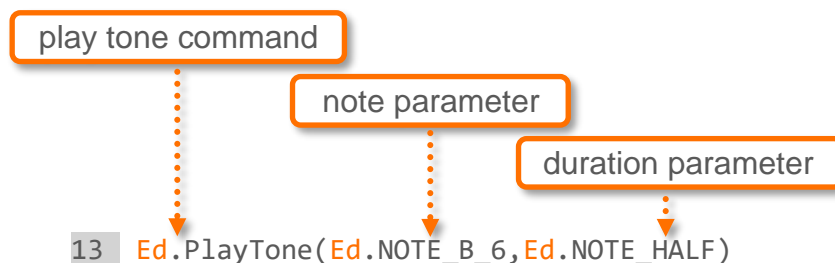
Using the parameter values tables as a reference, can you work out what the program will do?

This program will play a low B note for a duration of 1 second.

Your turn:

Task 1: Play a note

Write the following program:

```
1
2   #-------------Setup----------------
3
4   import Ed
5
6   Ed.EdisonVersion = Ed.V2
7
8   Ed.DistanceUnits = Ed.CM
9   Ed.Tempo = Ed.TEMPO_MEDIUM
10
11  #--------Your code below-----------
12
13  Ed.PlayTone(Ed.NOTE_A_SHARP_7,Ed.NOTE_HALF)
14
```

Download and test the program to see what it sounds like.

Task 2: Play a note, then drive? Or play a note while driving?

When Edison plays sounds, it does this in the background. This means that as soon as Edison starts playing the sound, the program will move onto the next line of code. The sound will keep playing 'in the background' while Edison continues on with the program.

If you want Edison to wait for the sound to finish, you need to use the Ed.ReadMusicEnd() function in a 'while' loop.

Write the following program:

```
1
2   #-------------Setup----------------
3
4   import Ed
5
6   Ed.EdisonVersion = Ed.V2
7
8   Ed.DistanceUnits = Ed.CM
9   Ed.Tempo = Ed.TEMPO_MEDIUM
10
11  #--------Your code below-----------
12  Ed.PlayTone(Ed.NOTE_C_8, Ed.NOTE_WHOLE)
13  while Ed.ReadMusicEnd()==Ed.MUSIC_NOT_FINISHED:
14      pass
15  Ed.Drive(Ed.FORWARD, Ed.SPEED_6, 5)
16
```

Download and test the program.

1. Describe what happened when you ran this program.

_____

_____

_____

2. Look at line 13 and 14 of the program. Remember that expressions compare the left side to the right side of the notation in the expression. What is this loop doing?

_____

_____

_____

Write the following program:

```
1
2   #-------------Setup----------------
3
4   import Ed
5
6   Ed.EdisonVersion = Ed.V2
7
8   Ed.DistanceUnits = Ed.CM
9   Ed.Tempo = Ed.TEMPO_MEDIUM
10
11  #--------Your code below----------
12  Ed.PlayTone(Ed.NOTE_C_8, Ed.NOTE_WHOLE)
13  Ed.Drive(Ed.FORWARD, Ed.SPEED_6, 5)
14
```

Download and test the program.

3. Describe what happened when you ran this program.

_____

_____

_____

4. Why did this program behave differently than the last program?

_____

_____

_____

Lesson 5: Worksheet 5.2 – Make an alarm

In this activity, you need to write a program to make Edison play an alarm at a frequency you specify.

Using the Ed.PlayTone() function, you can customise the exact frequency of the sound that Edison's speaker produces by using numbers and variables.

Frequency in acoustics

As you may know, sound travels in waves called sound waves. Acoustics, the branch of physics that deals with sound and sound waves, looks at everything to do with sound, including how to measure it.

One way to measure sound is by measuring frequency. Frequency is the number of waves passing a point in a certain period of time.

Frequency is most often measured in cycles per second (cycle/sec). The base unit for frequency is hertz, abbreviated Hz.

One hertz is equal to one complete wave per second.

> ***Did you know?*** The human hearing range is 20 Hz ~ 20000 Hz.

Frequency and period

In addition to the musical notes that are pre-set in EdPy, we can also program Edison to play sounds with different frequencies.

To do this, we convert frequencies into periods, which Edison can understand.

A period is how long it takes an acoustic wave to complete a full cycle. Since we are using hertz, we measure frequency in cycles per second.

In acoustics, when period increases, frequency decreases.

Let's look at some examples of how frequency and period relate:
- If a wave has a period of 0.5 seconds, it has a frequency of 2Hz because it can complete 2 cycles in 1 second.
- If a wave has a period of 2 seconds, it has a frequency of 0.5Hz because it can only complete half of a cycle in 1 second.

Converting frequency to period for your program

To get Edison to play a custom frequency, we need to work out the value of the period. This is the number we input into the 'note' parameter in Ed.PlayTone().

To convert a frequency into a period, divide the number 8,000,000 by the desired frequency. For example, to play a 1kHz (1000 cycles per second) sound:

$$\frac{8000000}{1000} = 8000$$

Your turn:

Task 1: Play a custom tone

Write the following program:

play tone command

note parameter as a fixed value

duration parameter in milliseconds

```
13  Ed.PlayTone(8000,2000)
```
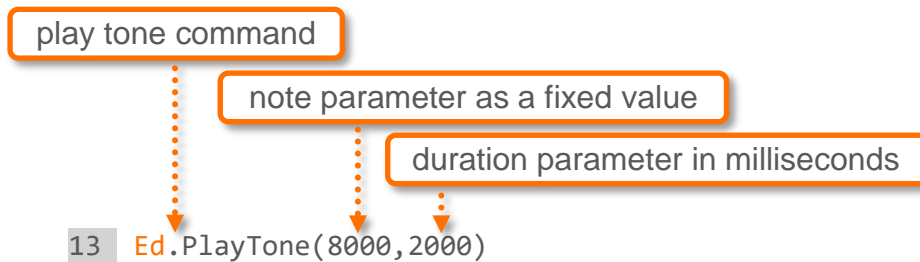
Download and test it to hear what this program sounds like.

Task 2: Play an alarm

In this program, we want Edison to play notes of increasing period.

To make the alarm program, you will need to use a 'for' loop, variables and the range() function. You also need to nest a 'while' loop into the program.

Write the following program:

```
1
2   #------------Setup---------------
3
4   import Ed
5
6   Ed.EdisonVersion = Ed.V2
7
8   Ed.DistanceUnits = Ed.CM
9   Ed.Tempo = Ed.TEMPO_MEDIUM
10
11  #--------Your code below-----------
12
13  for i in range(33):
14      Ed.PlayTone(100+(i*100), 1000)
15      while Ed.ReadMusicEnd()==Ed.MUSIC_NOT_FINISHED:
16          pass
17
```

Download it and test what this program sounds like.

Name_____

1. What do you hear from the robot? Why is this happening?

_____

_____

_____

An important skill in programming is being able to 'trace' through a program to understand what is happening. Programmers perform a code trace as a method for hand simulating the execution of their code to verify that it works correctly before compiling it manually.

Tracing involves stepping through the program line by line, recording important values. It is often done to help find errors or 'bugs' in code, but it is also useful when you just need to understand what is happening in a program.

Try to 'trace' through what is happening in the program and answer the following questions about the program.

2. Fill in the following table by calculating the period parameter for each given value of 'i' in the above code. The first value is filled in for you.

Value of i	Period parameter [the 1st input parameter to PlayTone()]
0	100
1	
2	

3. What is the maximum value of i?

_____

4. What is the maximum value of the period parameter input to the PlayTone() function?

_____

5. How many tones are played?

_____

Try it!

The application of acoustics in technology is called acoustical engineering.

Try some acoustical engineering of your own. Experiment with modifying the parameters to PlayTone() to make the program play a different combination of sounds.

Lesson 5: Worksheet 5.3 – Play a tune

In this activity, you need to write a program to make Edison play a musical tune.

You can get Edison to play a tune using the Ed.PlayTune() function and a special type of input called a 'string.'

Using a string to play a tune

In Python, a 'string' is a list of characters in order. A 'character' is anything you can type on the keyboard like a letter, a number, or a special character like $ or #. For example, 'Meet Edison' is a string, 11 characters long (10 letters and 1 space).

In the EdPy app, we need to use a string to play a musical tune. We call this a 'tune string.'

Tune strings are a special string of characters that represent particular tunes. Tune strings are made up of notes and duration inputs, which are represented by single characters.

A tune string looks like this: "ndndndndnd…ndz" where n is a note from the notes table and d is duration from the duration table:

Notes Table

String character	Plays musical note
m	low A
M	low A sharp
n	low B
c	C
C	C sharp
d	D
D	D sharp
e	E
f	F
F	F sharp
g	G
G	G sharp
a	A
A	A sharp
b	B
o	high C
R	rest
z	end of tune

Duration Table

String character	Plays
1	whole note
2	half note
4	quarter note
8	eighth note
6	sixteenth note

All tune strings must end with the 'z' character to end correctly.

To create a tune string, you need to call the function Ed.TuneString(), which has two input parameters. The size of the string (in other words, the number of characters in the string) is the first parameter, and the actual string you want to play is the second parameter.

You can change the speed your tune plays by changing the Ed.Tempo variable in the Setup code.

Your turn:

Write the following code to play the tune 'Mary Had a Little Lamb':

```
 1
 2   #-------------Setup----------------
 3
 4   import Ed
 5
 6   Ed.EdisonVersion = Ed.V2
 7
 8   Ed.DistanceUnits = Ed.CM
 9   Ed.Tempo = Ed.TEMPO_MEDIUM
10
11   #--------Your code below-----------
12   maryLamb = Ed.TuneString(53,"e4d4c4d4e4e4e2d4d4d2e4g4g4e4d4c4d4e4e4e4e4d4d4e4d4c1z")
13
14   Ed.PlayTune(maryLamb)
15 ▾ while Ed.ReadMusicEnd() == Ed.MUSIC_NOT_FINISHED:
16       pass
17
```

This is the tune string in the program:

"e4d4c4d4e4e4e2d4d4d2e4g4g4e4d4c4d4e4e4e4e4d4d4e4d4c1z"

Experiment with changing the Ed.Tempo value in the Setup code.

1. What are the different values that Ed.Tempo can take?

Hint: Remember you can use the autocomplete feature in the EdPy. Try typing 'Ed.TEMPO' and see all the possible values for Ed.TEMPO the autocomplete brings up.

_____

_____

2. Which Ed.TEMPO value will make the tune play the fastest?

_____

3. Modify your program to only play some part of the tune. Describe the changes you had to make to your program to only play a part of the tune.

_____

_____

_____

_____

_____

Lesson 5: Worksheet 5.4 – Make your robot dance

In this activity, you will write a program to make your robot dance.

In most good dance performances, there are some moves or actions which are repeated. You can make your Edison repeat actions in a dance routine by using the 'for' loop.

The 'shimmy' is a dance move where you hold your body still and quickly move your shoulders back and forth.

Look at the following program which will make your Edison robot do a version of a shimmy:

```
1
2   #------------Setup---------------
3
4   import Ed
5
6   Ed.EdisonVersion = Ed.V2
7
8   Ed.DistanceUnits = Ed.CM
9   Ed.Tempo = Ed.TEMPO_MEDIUM
10
11  #--------Your code below-----------
12
13  #Set up some variables
14  turnSpeed = Ed.SPEED_9
15  degreesToTurn = 20
16  numberOfTwists = 3
17
18  #Now shimmy!
19  Ed.Drive(Ed.SPIN_RIGHT,turnSpeed,degreesToTurn/2)
20▾ for i in range(numberOfTwists):
21      Ed.Drive(Ed.SPIN_LEFT,turnSpeed,degreesToTurn)
22      Ed.Drive(Ed.SPIN_RIGHT,turnSpeed,degreesToTurn)
23  Ed.Drive(Ed.SPIN_LEFT,turnSpeed,degreesToTurn/2)
24
```

This program uses variables so that it will be easy to change the turning speed, the number of twists in the dance and the degrees Edison will turn.

Both lines 13 and 18 start with '#' which means these lines are comment code lines added to make it easier for us to read the program. Remember, Edison will skip any line that starts with '#'.

Look at lines 19 and 23. In these lines, we are doing a mathematical calculation in our code to make Edison turn only half the number of degrees.

Your turn:

Write the program.

Download the program to your Edison and run it to see the dance in action.

1. How many times does the robot turn to the left?

    _____

2. How many times does the robot turn to the right?

    _____

3. The first turn to the right is only half the distance of all the turns inside the 'for' loop because this line has the input parameter 'degreesToTurn/2'. Why do want this line in the program? Try removing the maths (the /2) and run the program again. What do you notice? (*Hint:* look at how far Edison moves left compared to the start point.)

_____

_____

_____

_____

Try it!

Experiment with the program. Try changing the variables to change the way Edison dances. Change the number of degrees Edison will turn, the speed Edison will turn, the number of twists in the dance or all three!

Lesson 5: Worksheet 5.5 – Challenge! Dance to music

Dancing is more fun with music! In this activity, you will write a program combining dance moves with some tones or a tune.

Your turn:

Write and run the following program that combines a 'shimmy' dance with some tones:

```
1
2    #------------Setup---------------
3
4    import Ed
5
6    Ed.EdisonVersion = Ed.V2
7
8    Ed.DistanceUnits = Ed.CM
9    Ed.Tempo = Ed.TEMPO_MEDIUM
10
11   #--------Your code below----------
12
13   #Set up my variables
14   turnSpeed = Ed.SPEED_9
15   degreesToTurn = 60
16   numberOfTwists = 3
17
18   #Now dance to the music!
19   Ed.Drive(Ed.SPIN_RIGHT, turnSpeed, degreesToTurn/2)
20   Ed.PlayTone(Ed.NOTE_C_7, Ed.NOTE_SIXTEENTH)
21 ▾ for i in range(numberOfTwists):
22       Ed.Drive(Ed.SPIN_LEFT, turnSpeed, degreesToTurn)
23       Ed.PlayTone(Ed.NOTE_A_7, Ed.NOTE_SIXTEENTH)
24       Ed.Drive(Ed.SPIN_RIGHT, turnSpeed, degreesToTurn)
25       Ed.PlayTone(Ed.NOTE_C_7, Ed.NOTE_SIXTEENTH)
26   Ed.Drive(Ed.SPIN_LEFT, turnSpeed, degreesToTurn/2)
27   Ed.PlayTone(Ed.NOTE_A_7, Ed.NOTE_SIXTEENTH)
28
```

Now design your own dance for your Edison, adding some tones or using a tune string. Can you synchronise it so Edison dances in time with the music?

1. Describe your robot's dance moves. Is there anything in your program you really liked? If so, describe it.

_____

_____

_____

_____

_____

2. What combination of tones or notes did you play along with your dance?

_____

_____

_____

_____

Lesson 6: Worksheet 6.1 – Flash the LED in response to a clap

In this activity, you need to write a program using Edison's clap-detecting sensor to get the robot to flash one LED light whenever it detects a clap.

The first thing to do is to plan out the program.
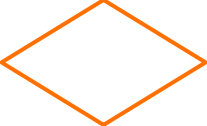
Flowcharts in programming

Professional programmers usually plan out their program before they start writing their code. Using flowchart diagrams is a way that programmers can organise and plan out their programs.

The idea of a flowchart is to graphically summarise what happens in the code without needing to go into all of the detail. Flowcharts allow a programmer to visualise and communicate how the 'flow' of the program will work.

In a flowchart, a program is represented using different shapes and arrows. Each shape represents a different element in the program, and the arrows show how the elements work together.

There are five main symbols used in flowcharts:

Symbol	Name	Function
	Terminator (start/end)	An oval represents a start or end point
	Arrow	A line which acts as a connector, showing the relationships between representative shapes
	Input/output	A parallelogram represents an input or an output
	Process	A rectangle represents a process or action
	Decision	A diamond represents a decision

More complicated flowcharts may also use additional shapes with different meanings.

When making a flowchart to plan a program, words are often added inside the shapes or next to the arrows. These words are short summaries of the process or decision.

Let's look at an example flowchart summarising the program we want to make.

Here is a flowchart for a program that will tell your Edison robot to wait for a clap, then flash the left LED:

```
                    ┌──────────────┐
                    │    Start     │
                    └──────┬───────┘
                           │
          ┌────────────────┤
          │           ╱─────────╲
          │          ╱   Clap     ╲        No
          │          ╲ detected?  ╱──────────────┐
          │           ╲─────────╱                │
          │             │ Yes                     │
          │      ┌──────┴───────┐                 │
          │     ╱  Left LED on  ╱                  │
          │    └──────┬───────┘                    │
          │           │                            │
          │      ┌────┴─────┐                      │
          │      │   Wait   │                      │
          │      └────┬─────┘                      │
          │           │                            │
          │      ┌────┴─────┐                      │
          │     ╱ Left LED off                     │
          │    └──────┬───────┘                    │
          │           │                            │
          │      ┌────┴─────┐                      │
          │      │   Wait   │                      │
          │      └────┬─────┘                      │
          └───────────┘────────────────────────────┘
```
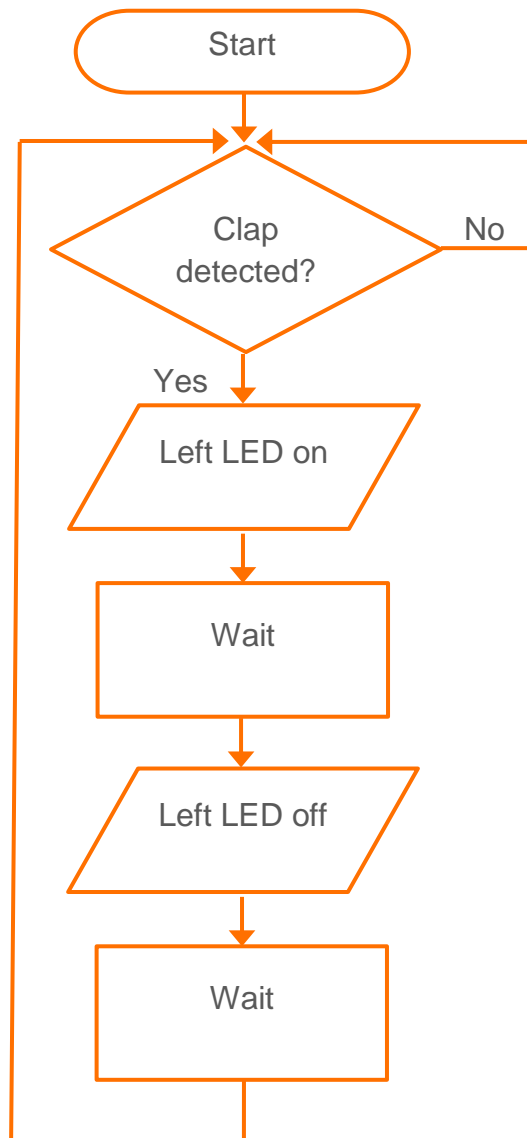
This program will use Edison's sound-detecting sensor to determine whether or not a clap has occurred. The result determines how the code flows next.

When you look at this flowchart, you may notice that it doesn't have an 'end' terminator. That is because this program uses a 'while' loop set up in a way to make the program continue indefinitely.

Making an infinite loop
Sometimes you may want to write a program that doesn't have an end, but loops forever. In programming, this is often referred to as an infinite loop.

You can use an infinite loop to make the program planned out in the flowchart.

Look at the following program:

```
1
2    #-------------Setup----------------
3
4    import Ed
5
6    Ed.EdisonVersion = Ed.V2
7
8    Ed.DistanceUnits = Ed.CM
9    Ed.Tempo = Ed.TEMPO_MEDIUM
10
11   #--------Your code below-----------
12   while True:
13       Ed.ReadClapSensor()
14       while Ed.ReadClapSensor() != Ed.CLAP_DETECTED:
15           pass
16       Ed.LeftLed(Ed.ON)
17       Ed.TimeWait(50, Ed.TIME_MILLISECONDS)
18       Ed.LeftLed(Ed.OFF)
19       Ed.TimeWait(50, Ed.TIME_MILLISECONDS)
20
```

This program is represented by our flowchart and includes an infinite loop.

Look at line 12 of the program, which uses a 'while' loop.

'While' loops always need a condition. The loop will repeat any indented code while that condition resolves to 'true'.

If we want the 'while' loop to repeat infinitely, instead of giving a condition the program must evaluate, we replace the condition with 'True'.

'True' always resolves to 'true'. By setting the condition to 'True', we have hardcoded the condition of our 'while' loop to be 'true'.

In programming, hardcoding is where you force something to be a specific way by explicitly typing it out.

By using 'True' as the condition for the 'while' loop in our program, the condition of the loop can never be false and will repeat infinitely.

Your turn:

Write the above code to program your Edison robot to make the left LED flash when you clap. Download it and test to see how it works.

1. What is the furthest distance away from your Edison that you can be and still have the robot sense when you clap?

   _____

2. What is the purpose of having the TimeWait() function calls in the code? What would happen if we didn't have them? *Hint:* Try running the program without the TimeWait() function calls.

_____

_____

_____

3. Look at the program and the flowchart to compare how the code in the program and the flowchart relate to each other. Explain what happens in the code when the outcome of the decision represented in the flowchart is 'no'.

_____

_____

_____

Try it!
Can you change the program so both the LEDs come on when you clap?

Lesson 6: Worksheet 6.2 – Drive in response to a clap

In this activity, you need to write a program to make your Edison robot drive forward in response to a clap.

Your turn:

Task 1: Drive forward when a clap is detected

Write and run the following program:

```
1
2   #------------Setup----------------
3
4   import Ed
5
6   Ed.EdisonVersion = Ed.V2
7
8   Ed.DistanceUnits = Ed.CM
9   Ed.Tempo = Ed.TEMPO_MEDIUM
10
11  #--------Your code below-----------
12  Ed.ReadClapSensor()
13 ▾ while Ed.ReadClapSensor() == Ed.CLAP_NOT_DETECTED:
14      pass
15  Ed.Drive(Ed.FORWARD,Ed.SPEED_8,10)
16
```

1. Why is it necessary to perform an initial read of the clap sensor in line 12? What is this doing? *Hint:* Look back at worksheet 2.5.

_____

_____

_____

Task 2: Drive forward, then backwards when a clap is detected

The Edison robot's sound sensor is not just sensitive to claps. The sensors can respond to any loud sound detected, which is why you can tap near the speaker on the robot to trigger the sound sensor.

Edison's motors, gears and wheels all make sounds as they turn, which can trigger the sound sensor. To prevent the sound of the robot driving from triggering the sound sensor, you need to alter the program.

You will need to add a TimeWait() function call with an input parameter of about 350 milliseconds to give the robot's motors time to stop.

You also need to use a ReadClapSensor() to clear the clap sensor.

Write the following program:

```
1
2    #-------------Setup----------------
3
4    import Ed
5
6    Ed.EdisonVersion = Ed.V2
7
8    Ed.DistanceUnits = Ed.CM
9    Ed.Tempo = Ed.TEMPO_MEDIUM
10
11   #--------Your code below-----------
12   Ed.Drive(Ed.FORWARD,Ed.SPEED_8,10)
13
14   Ed.TimeWait(350,Ed.TIME_MILLISECONDS)
15   Ed.ReadClapSensor()
16 - while Ed.ReadClapSensor() == Ed.CLAP_NOT_DETECTED:
17       pass
18   Ed.Drive(Ed.BACKWARD,Ed.SPEED_8,10)
19   |
```

Download and test the program.

2. Usually, we write a flowchart to help us plan out our code. For practice, write a flowchart to match the code you just wrote. Draw your flowchart below or try to use a program like Google Slides to make your flowchart.

Lesson 6: Worksheet 6.3 – Design your own function

In this activity, you will design your own function and use it to write a program for Edison.

What, exactly, are functions?

Now that you have been programming for a while with Edison and EdPy, you have used a range of different functions from the EdPy library.

As you know, a function is a piece of code that performs a particular role or job in the program depending on which input parameters are used.

But you might not have realised that all of the functions you have used so far have actually executed multiple lines of code when run by the program.

That's because a function is a block of organised, reusable code that is used to perform a single, related action.

Functions are very helpful because they allow us to program is a more modular way, using the same block of code at various points throughout the program. To get your program to run all of those lines of code you only need to type one line: calling that function.

Functions make reusing code easier and more efficient when we program.

> Watch a short video by code.org explaining why functions are so useful in programming:
> https://youtu.be/8T5acEwfJbw

Making your own function

So far, every time you have used a function in EdPy you have called that function from the Ed library. You can also make your own functions.

In Python, functions look like this:

```
       define the function
                 name of the function
                        first parameter        second parameter

13  def function_name (parameter_1, parameter_2):
14      {this is the code in the function}
15      {more code in the function}
16      return {value to return to the program}
17  {this code isn't in the function – no indent}
```
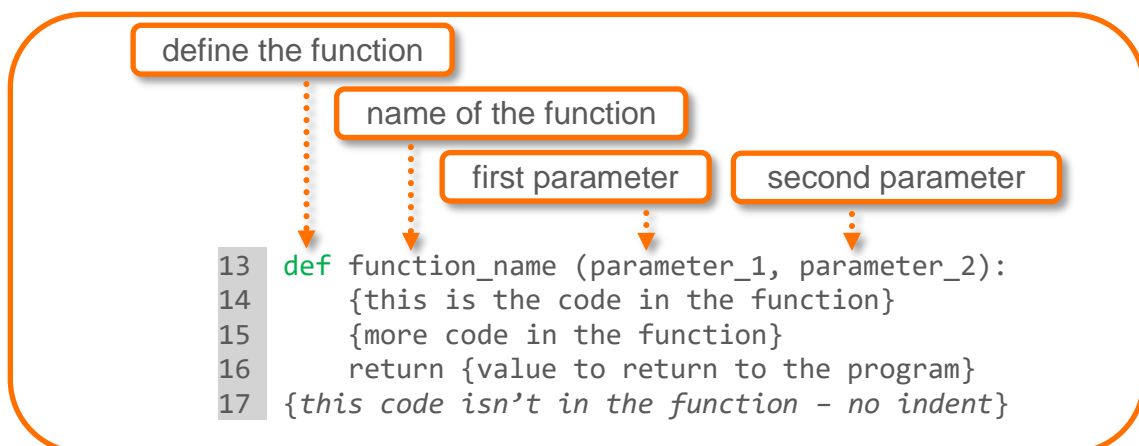
Anything indented is a part of the function. Anything not indented is not a part of the function, but the next line of code in the program.

It's important to note that the functions you create are no different from the functions you've used from the Ed library so far.

When you call your function (with or without parameters), the program jumps from the call to the code in the function (the indented code). The program then runs this code before returning to the line where you made the call.

If you set your function to return a value, when the program returns to the line where you made the call, the function call gets replaced by the value that the function has returned.

This is important because the program will always resolve functions, then maths orders then expressions in this order.

Organising your code

The convention in EdPy is to write the definition of your functions at the end of your code after you have already used your function in your program.

This is so your code is nice and neat. By organising your code this way, all your functions, whether you are using your own or calling functions from a library, can be written in the main part of your program in a visually clean, organised way. Your function definitions can sit outside of this, further down at the bottom of the program.

Your turn:

Task 1: Practice defining a function

Look at the following program:

```
1
2    #------------Setup---------------
3
4    import Ed
5
6    Ed.EdisonVersion = Ed.V2
7
8    Ed.DistanceUnits = Ed.CM
9    Ed.Tempo = Ed.TEMPO_MEDIUM
10
11   #--------Your code below-----------
12   directionToMove=Ed.SPIN_LEFT
13   speedToMoveAt=Ed.SPEED_8
14   distanceToMove=360
15
16   #call the function
17   moveOnClap(directionToMove, speedToMoveAt, distanceToMove)
18
19
20   #user defined functions
21 ▾ def moveOnClap(direction, speed, distance):
22       Ed.ReadClapSensor()
23 ▾     while Ed.ReadClapSensor()==Ed.CLAP_NOT_DETECTED:
24           pass
25       Ed.Drive(direction, speed, distance)
26
27
```

Line 17 in the program is the function call. Lines 21 to 25 are defining the function. Remember, the convention is to define your function at the bottom of your program to keep everything neat and organised.

Write the program and download it to your Edison robot. Run the program to see how it works.

1. We could write a function to make the Edison robot drive in a square shape. Fill in the missing words in the function below to finish writing this function.

```
def driveInaSquare():

    for i in range(___):

        Ed.Drive(_____,_____,_____)

        Ed.Drive(_____,_____,_____)
```

2. Write the syntax for the code to call this function. In other words, what would you type in your program to call this function?

_____

Write a program that will drive Edison in multiple squares. You will need to define the 'driveInaSquare' function, and your program will need to call this function more than once. Download and run the program to see how it works.

3. Does the program do what you expected it to do? If not, describe what you expected and what the program actually did. Describe any issues you had getting your program to work.

_____

_____

_____

_____

_____

Task 2: Design your own function

Write your own function which will have Edison do something when you clap. You could make Edison dance, flash an LED, turn in a circle or anything else you like!

Step 1: Design

First, write a flowchart which summarises your program graphically. Either make your flowchart by drawing it out on paper or use a program such as Google Slides. Be sure to use the correct shapes in your flowchart to represent the program's start, processes, decision points and flow.

The shapes you will need will depend on your flowchart. At a minimum, you will need the oval start shape, the rectangle process shape and the diamond decision shape in your flowchart.

Step 2: Code

Translate your idea into code in the EdPy app. Use your flowchart as your guide and code your function to include each step you laid out in your flowchart.

Step 3: Test

Write a test program that includes your function and additional code to 'exercise' your function. (Exercising a function means calling it in your code.) See if your function works as you planned and expected it to work. If not, revisit your flowchart and code, adjusting as needed. Experiment to see what works!

4. Describe what your function did.

_____

_____

_____

_____

_____

_____

5. Describe any problems you had.

_____

_____

_____

_____

_____

Lesson 7: Activity sheet 7.1 – Calibrate obstacle detection

You can regulate the sensitivity of Edison's obstacle detection system. By making the obstacle detections system more sensitive, Edison can detect obstacles further away. By making the system less sensitive, Edison will only detect very close obstacles. Use this activity sheet to adjust your Edison's obstacle detection system.

Step 1: read the barcode

1. Place Edison on the right side, facing the barcode
2. Press the record (round) button three times
3. Edison will drive forward and scan the barcode

Barcode – Calibrate obstacle detection

Step 2: set maximum sensitivity

After scanning the barcode, set Edison down on a table or desk and remove any obstacles in front of Edison. Then press the play (triangle) button. Edison is now in calibration mode.

The left sensitivity is calibrated first.

1. Repeatedly press the play (triangle) button, which increases sensitivity, until the red LED on the left is flickering.
2. Repeatedly press the record (round) button, which decreases the sensitivity, until the LED completely stops flickering.
3. Press the stop (square) button to switch over to calibrate the right side.
4. Repeatedly press the play (triangle) button until the right red LED is flickering. Now repeatedly press the record (round) button until the LED completely stops flickering.
5. Press the stop button to complete the calibration.

Special note: custom sensitivity

It is possible to set the distance that obstacles are detected. To do this, scan the 'calibrate obstacle detection' barcode, place an obstacle in front of Edison at the distance you want Edison to detect obstacles, press the play button and then repeat steps 1 through 5.

Lesson 7: Worksheet 7.1 – Infrared obstacle detection

In this activity, you will learn more about infrared (IR) light and how Edison can use IR to detect obstacles.

What is infrared (IR) light?

There is a wide range of light, some of which is visible to the human eye and some of which is not. Infrared, also called IR, is not visible to humans.

> *Did you know?* Even though people cannot see it, infrared is a type of light. Therefore, it will work in the dark. That's why you can turn on a TV set using a remote control even if there are no lights on in the room!

Edison and infrared

The Edison robot is equipped with an infrared system that gives the robot 'vision' of a sort. This infrared system allows Edison to detect obstacles around the robot.

Edison's infrared system is made up of two IR light emitter diodes (or LEDs) on the front. One is on the left, and one is on the right. Edison also has an IR sensor on the front, directly in the middle.

For Edison to detect obstacles, infrared light is emitted forward from the left and right IR LEDs. If the IR light encounters an obstacle, such as a wall, it is reflected back towards Edison. Edison's IR sensor then detects the reflected light.

Look at the illustration below:

IR light emitter

IR light detector

IR light emitter
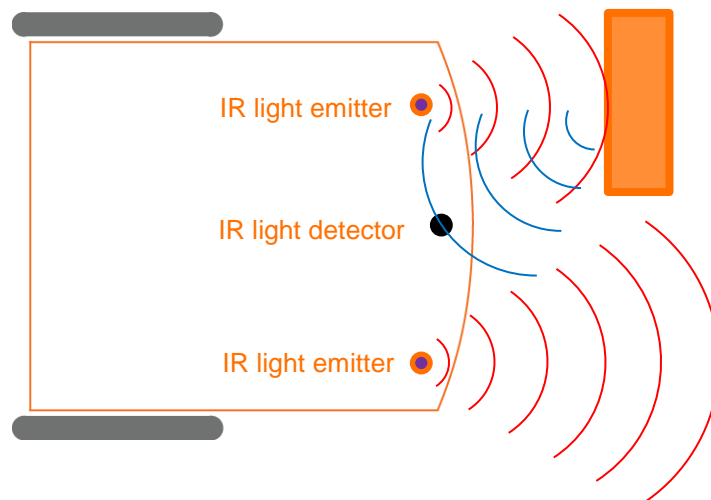
Edison emits IR light (shown in red) from both the robot's left and right IR LEDs. Reflected IR light (shown in blue) bounces off obstacles and is detected by Edison's IR sensor.

In this picture, there is an obstacle in front of Edison's left side, but not the right side. That's why only IR light from the left emitter is reflected.

From the received signal Edison can determine that there is an obstacle to the left, but no obstacle to the right.

Your turn:

1. Draw the emitted IR light and reflected IR light for this obstacle.

IR light emitter

IR light detector

IR light emitter
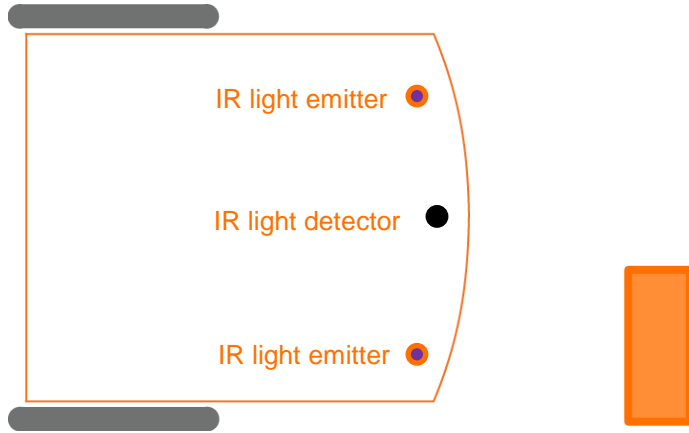
Lesson 7: Worksheet 7.2 – Detect an obstacle and stop

In this activity, you will need to write a program to make your Edison robot drive until it encounters an obstacle and then stop.

Look at the following program:

```
1
2    #------------Setup---------------
3
4    import Ed
5
6    Ed.EdisonVersion = Ed.V2
7
8    Ed.DistanceUnits = Ed.CM
9    Ed.Tempo = Ed.TEMPO_MEDIUM
10
11   #--------Your code below-----------
12
13   Ed.ObstacleDetectionBeam(Ed.ON)
14
15   Ed.Drive(Ed.FORWARD,Ed.SPEED_5,Ed.DISTANCE_UNLIMITED)
16
17 - while Ed.ReadObstacleDetection() != Ed.OBSTACLE_AHEAD:
18       pass
19   Ed.Drive(Ed.STOP,1,1)
20
```

This program tells Edison to drive until it encounters an obstacle.

There are a couple of important things to notice about this program.

Look at line 13 of the program. This line turns Edison's obstacle detection beam to 'on'. Whenever you want to use Edison's obstacle detection beam in an EdPy program, you always need to turn the beam to 'on' before the beam is used in the program.

Now, look at line 15 of the program. This line sets the speed to 5 in this program. When using obstacle detection, you need to use a slightly lower speed to allow the robot to detect an obstacle before colliding with it. If the speed is too fast, the robot will crash into obstacles before being able to detect them.

Your turn:

Task 1: Select your obstacles

You need to choose good obstacles to use with Edison. 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 (e.g. not black) and at least as tall as Edison.

For this program, the wall of the room would be a good obstacle.

Task 2: Get your Edison ready

When you want to run a program using obstacle detection, it is a good idea to check that your Edison robot's obstacle detection is calibrated to the distance you want. Use activity sheet 7.1 to calibrate your Edison. You may need to calibrate your robot with custom sensitivity to make sure your robot can detect and stop at your obstacle.

Task 3: Write and run the program

Write the program using the EdPy app and download it to your Edison robot. Then run the program to see how it works.

Experiment using different obstacles to see what Edison can and cannot detect. You can also try setting Edison's obstacle detection calibration to different distances to see what happens.

1. Delete the 'Ed.ObstacleDetectionBeam(Ed.ON)' line out of your program. Try to download and run the adjusted program. What happened? Why does that happen?

_____

_____

_____

2. Think about where you have seen this type of invisible detection before in the real world. Describe an example.

_____

_____

_____

3. Where else do you think this type of detection technology could be used? Write down at least one idea of how you could use this technology.

_____

_____

_____

_____

Lesson 7: Worksheet 7.3 – Obstacle avoidance

In this activity, you will write a program to make your Edison robot drive until it encounters an obstacle, turn and drive away.

Look at the following program:

```
1
2   #------------Setup---------------
3
4   import Ed
5
6   Ed.EdisonVersion = Ed.V2
7
8   Ed.DistanceUnits = Ed.CM
9   Ed.Tempo = Ed.TEMPO_MEDIUM
10
11  #--------Your code below-----------
12
13  Ed.ObstacleDetectionBeam(Ed.ON)
14
15  Ed.Drive(Ed.FORWARD,Ed.SPEED_5,Ed.DISTANCE_UNLIMITED)
16
17  while Ed.ReadObstacleDetection() != Ed.OBSTACLE_AHEAD:
18      pass
19  Ed.Drive(Ed.SPIN_RIGHT,Ed.SPEED_5,180)
20  Ed.Drive(Ed.FORWARD,Ed.SPEED_5,10)
21  |
```

This program tells Edison to drive forward until an obstacle is detected. Once Edison detects an obstacle, the program tells Edison to turn 180° and drive away for 10 cm.

Your turn:

Task 1: Write and run the program

Write the program using the EdPy app and download it to your Edison robot. Then run the program to see how it works.

After you have run the program, look at it again and think about how you could change the program. Try to modify the code so that the robot behaves differently when it detects an obstacle ahead. (*Hint*: try using sound and lights.)

1. Think about how you could improve the original program. What could you change so that the program does more than just turn and drive away after encountering one obstacle?

_____

_____

_____

Task 2: Syntax errors and logical errors

Programmers often make 'typo' like errors, called syntax errors. It is important to be good at spotting your own syntax errors so you can correct your code.

You can also get another type of error, called a logical error, when you program. In coding, any error that is not a syntax error is a logical error.

When there is a logical error in a program, the code doesn't behave the way that the programmer expects it to. In EdPy, if you have a logical error in your program, the program will usually still download to Edison. When you run the program, however, it won't behave the way you think it should.

A logical error can be as simple as using the wrong function or leaving out a function. An example of a logical error would be writing a program that uses obstacle detection but not turning the obstacle detection beam on.

Look at the following program:

```
1
2    #------------Setup---------------
3
4    import Ed
5
6    Ed.EdisonVersion = Ed.V2
7
8    Ed.DistanceUnits = Ed.CM
9    Ed.Tempo = Ed.TEMPO_MEDIUM
10
11   #--------Your code below----------
12
13   Ed.ObstacleDetectionBeam(Ed.ON)
14
15   Ed.Drive(Ed.FORWARD,Ed.SPEED_5, Ed.DISTANCE_UNLIMITED)
16
17   While Ed.ReadObstacleDetection() != Ed.OBSTACLE_NONE
18   pass
19   Ed.Drive(Ed.SPIN_RIGHT,Ed.SPEED_5,145)
20   Ed.Drive(Ed.FORWARD, Ed.SPEED_5,20)
21
```

The program is meant to make the Edison robot drive until it encounters an obstacle, then turn 135° and drive away from the obstacle for 20 cm.

There are five errors in the program. Can you identify all five errors in the program and determine if they are syntax errors or logical errors? Fill in your answers in the table on the next page.

Name_____

Hint: You can write this program in EdPy and use the 'Check Code' button to help you find the syntax errors.

Error #	Line #	Error type (syntax or logical)	Error description
1			
2			
3			
4			
5			

Lesson 7: Worksheet 7.4 – Detect an obstacle as an event

In this activity, you will write an event driven program to make your Edison robot continuously drive forward while avoiding obstacles.

Event driven programming

In programming, an event is something which occurs 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.

Many programming languages, including Python, allow programmers to create code which can respond to some set of events. This type of programming is called 'event driven programming'.

Whenever you write a program which is event driven, you will also need to write code which handles these events. There are two main ways to do this, either by having the program wait in a loop or by using interrupts.

We interrupt this lesson to talk about … interrupts

As you already know, programs usually move sequentially through the code, line by line. There are ways to allow the code to move differently, however, such as by using loops.

You can also affect the way a program runs by using an 'interrupt'.

An interrupt is a section of code that pauses the main program and runs itself. Once the interrupt code is complete, the program returns to wherever it left off in the main program.

Interrupts are always functions defined somewhere in the program. They are generally short sections of code designed to perform a specific task without causing a major disruption to the flow of the main program.

Programmers use interrupts because interrupts allow a program to react to an event at any time while the program is running. In other words, by using interrupts, a programmer doesn't have to predict exactly when during a program the event will occur.

Event handlers and interrupts

When using interrupts in event driven programming, you will need to use 'event handlers'.

An event handler is a way of tying an interrupt to a specific event.

To use an event handler, you first need to set up, or 'register', the event handler in your code. Once registered, the event handler constantly monitors for its given event. Whenever that event occurs, the event handler triggers the interrupt, which calls and runs the function, then returns to the main program.

Look at the following program:

```
 1
 2   #-------------Setup---------------
 3
 4   import Ed
 5
 6   Ed.EdisonVersion = Ed.V2
 7
 8   Ed.DistanceUnits = Ed.CM
 9   Ed.Tempo = Ed.TEMPO_MEDIUM
10
11   #--------Your code below-----------
12   Ed.ObstacleDetectionBeam(Ed.ON)
13   Ed.RegisterEventHandler(Ed.EVENT_OBSTACLE_AHEAD, "avoidObstacle")
14
15 ▾ while True:
16       Ed.Drive(Ed.FORWARD,Ed.SPEED_5,Ed.DISTANCE_UNLIMITED)
17
18 ▾ def avoidObstacle():
19       Ed.Drive(Ed.SPIN_RIGHT,Ed.SPEED_5,180)
20       Ed.ReadObstacleDetection()
21
```

In EdPy, we use the 'Ed.RegisterEventHandler' function to register an event handler.

The 'Ed.RegisterEventHandler' function has two parameters. The first parameter is the event that is going to occur and the second parameter is the function that will be called whenever that event occurs.

Look at line 13 of the program. This line is registering an event handler so that whenever 'EVENT_OBSTACLE_AHEAD' occurs, the 'avoidObstacle' function will be called.

Your turn:

Write the program using the EdPy app and download it to your Edison robot. Then run the program to see how it works.

1. Describe what the robot does.

_____

_____

_____

_____

2. In this program, when the robot detects an obstacle ahead, which lines of code will be executed? Why does the program run these lines?

_____

_____

_____

_____

3. Why is Ed.ReadObstacleDetection included in this program? In other words, what is line 20 doing? *Hint:* Look back at worksheet 2.5 for a clue or try removing line 20 and running the program.

_____

_____

_____

_____

Try it!

What else could Edison do when it detects an obstacle ahead? Try to modify the code so that Edison performs a different behaviour when it detects an obstacle ahead. Experiment with the program to see what works and what does not.

Lesson 7: Worksheet 7.5 – Right and left obstacle detection

In this activity, you will write a program for Edison to react to obstacles on the left or right of the robot. To do this, we will use 'if statements'.

If statements

An important part of coding is making decisions. The most common way to do this is to use an 'if statement'.

An 'if statement' asks whether a condition is true or false. If the result is true, then the program executes the block of statements following the if statement. If the result is false, the program ignores the statements inside the if statement and moves to the next line of code outside of the if statement.

Look at the following program:

```
1
2   #------------Setup----------------
3
4   import Ed
5
6   Ed.EdisonVersion = Ed.V2
7
8   Ed.DistanceUnits = Ed.CM
9   Ed.Tempo = Ed.TEMPO_MEDIUM
10
11  #--------Your code below-----------
12  Ed.ObstacleDetectionBeam(Ed.ON)
13
14 ▾ while True:
15 ▾     if Ed.ReadObstacleDetection() != Ed.OBSTACLE_NONE:
16            Ed.PlayBeep()
17            Ed.ReadObstacleDetection()
18
```

This program uses an if statement to give the robot the ability to make decisions without human guidance. When a robot can make decisions on its own in this way, it is called an autonomous robot.

Your turn:

Task 1: Beep if there's an obstacle

Write the above program using the EdPy app, download it to your Edison robot and run the program. Then try moving an obstacle, like your hand, in and out of Edison's obstacle detection beam to see what happens.

1. Edison can now react differently to different stimuli, making a 'decision' as to what to do. Does this mean Edison has intelligence? *Hint:* You may want to do a bit of research on artificial intelligence to help you decide.

_____

_____

_____

_____

_____

Task 2: Beep if there's an obstacle, else turn on the light

In addition to telling a program what to do when an if statement condition is true, you can also tell a program what to do if that condition is false.

If, else
Using if statements with 'else' allows us to write programs making more complicated decisions. If/else statements are basically a way of making a decision between two things.

Watch Bill Gates, founder of Microsoft, explain if/else statements and decision making in programming: https://youtu.be/fVUL-vzrIcM

In Python, the syntax for if/else is:

```
if expression:
    statement(s)
else:
    statement(s)
```

The program moves sequentially from the top down, starting with the if condition. If the if statement is true, the program runs the indented code for the if expression and skips the else. If the if statement is false, however, the program skips that section of indented code and runs the 'else' indented code instead.

Write the following program:

```
1
2   #-------------Setup----------------
3
4   import Ed
5
6   Ed.EdisonVersion = Ed.V2
7
8   Ed.DistanceUnits = Ed.CM
9   Ed.Tempo = Ed.TEMPO_MEDIUM
10
11  #--------Your code below-----------
12  Ed.ObstacleDetectionBeam(Ed.ON)
13
14 ▾ while True:
15 ▾     if Ed.ReadObstacleDetection() != Ed.OBSTACLE_NONE:
16            Ed.LeftLed(Ed.OFF)
17            Ed.PlayBeep()
18            Ed.TimeWait(100, Ed.TIME_MILLISECONDS)
19            Ed.ReadObstacleDetection()
20 ▾     else:
21            Ed.LeftLed(Ed.ON)
22
```

Download and run the program. Try moving an obstacle in and out of Edison's obstacle detection beam to see what happens.

This program has two pathways: one for if an obstacle is detected and one for if no obstacle is detected.

If, elif, else

You can also make a program which makes a decision using more than two conditions. To do this, you use another Python syntax structure:

```
if expression:
    statement(s)
elif expression:
    statement(s)
else:
    statement(s)
```

'Elif' is how you say 'else if' in Python. You can use elif to write a program with multiple if conditions.

A program using if/elif/else still moves sequentially from the top down. Once the program runs any indented code inside any part of the if statement structure, it will skip the rest of the structure and move on to the next line of code outside the structure.

This means that if the if statement at the top is true, the program runs the indented code for the if expression and skips any elif sections as well as the else section if there is one. If the if statement is false, however, the program skips that section of indented code and moves to the first elif section.

Again, if the first elif condition is true, the program runs its indented code and skips everything below it in the if statement structure (any other elifs and the else condition if there is one). If this elif condition is false, the program moves to the next part of the if statement structure and so on.

Task 3: Detect an obstacle on the left or the right

Look at the following program:

```
1
2   #-------------Setup----------------
3
4   import Ed
5
6   Ed.EdisonVersion = Ed.V2
7
8   Ed.DistanceUnits = Ed.CM
9   Ed.Tempo = Ed.TEMPO_MEDIUM
10
11  #--------Your code below-----------
12  Ed.ObstacleDetectionBeam(Ed.ON) #turn on obstacle detection
13
14  while True:
15      Ed.Drive(Ed.FORWARD, Ed.SPEED_1, Ed.DISTANCE_UNLIMITED)
16
17      obstacle=Ed.ReadObstacleDetection()
18
19      if obstacle>Ed.OBSTACLE_NONE: #there is an obstacle
20          Ed.Drive(Ed.BACKWARD, Ed.SPEED_5, 7)
21          if obstacle==Ed.OBSTACLE_LEFT:
22              Ed.Drive(Ed.SPIN_RIGHT, Ed.SPEED_5, 90)
23          elif obstacle==Ed.OBSTACLE_RIGHT:
24              Ed.Drive(Ed.SPIN_LEFT, Ed.SPEED_5, 90)
25          elif obstacle==Ed.OBSTACLE_AHEAD:
26              Ed.Drive(Ed.SPIN_RIGHT, Ed.SPEED_5, 180)
27          Ed.ReadObstacleDetection() #clear any unwanted detections
28
```

This program has three different paths that it can take when an obstacle is detected based on where the detected obstacle is in relation to Edison.

Write the program using the EdPy app and download it to your Edison robot. Then run the program to see how it works.

2. When running this program, explain in your own words what the robot does when:

Obstacle detected ahead: _____

Obstacle detected on right: _____

Obstacle detected on left: _____

Lesson 8: Worksheet 8.1 – Line tracking sensor

In this activity, you will learn more the Edison robot's line tracking sensor and how Edison can use this sensor to determine if it is on a reflective or non-reflective surface.

How does Edison's line tracking sensor work?

Your Edison robot is equipped with a line tracking sensor, located near the power switch on the bottom of the robot. This sensor is made up of two main electronic components:

1. a red light emitting diode (LED), and
2. a phototransistor (light sensor).

This image represents a cross-section of Edison's line tracking sensor:
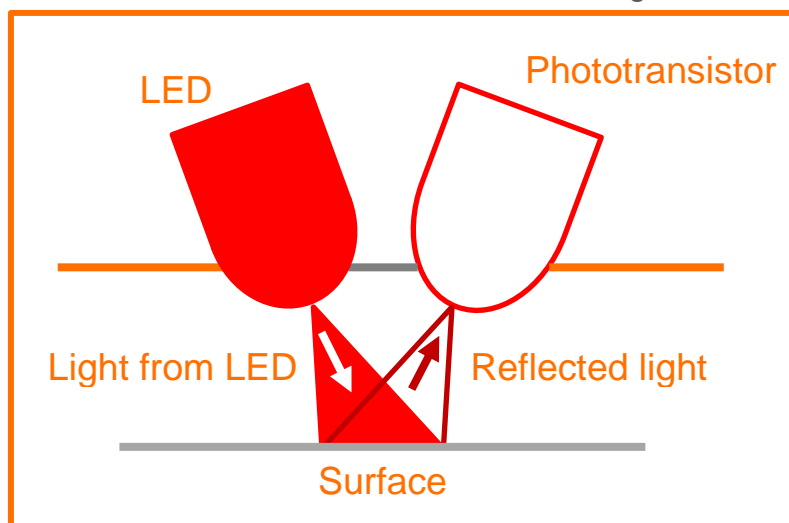
The line tracking sensor's LED shines light onto the surface that the Edison robot is driving on.

The phototransistor component is a light sensor. The phototransistor measures the amount of light that is reflected back up from the surface beneath Edison.

Your turn:

Task 1: Black versus white

When more light is reflected back to Edison's phototransistor, it gives a higher light reading.

Experiment to see whether a white or a black surface will reflect back more light.

Use activity sheet 8.1 or one piece of white and one piece of black paper. Turn Edison on and press the round button twice so that the 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 on the surface. Compare how bright the spot of light appears when placed on a black surface and then on a white surface.

1. Does the LED light spot appear brighter when placed on a black or a white surface?

_____

Task 2: Red, green and blue

As you saw in task 1, there is more light reflected from a white surface than from a black surface. That's why the light appears brighter on a white surface.

When the phototransistor is on a white surface, it gives a higher light reading than when it is on a black surface. A black surface is, therefore, considered to be 'non-reflective' and a white surface is considered to be 'reflective'.

The phototransistor's ability to determine if Edison is on a reflective or non-reflective surface is what allows the robot to be programmed to respond to the surface that it is driving on.

2. Think about how the line tracker would respond to each of the following surface colours. Would the line tracker see each colour as reflective or non-reflective? Remember, Edison's line tracker LED emits red light. (*Hint*: you can test your answers using activity sheet 8.1.)

Red surface _____

Green surface _____

Blue surface _____

Video – Humans need not apply

Line tracking is a very basic robotic behaviour that is used in many automated factories and warehouses today. Will it be the same in the future? What will the workplace of the future look like with more robots? Will it even exist?

Watch the video *Humans Need Not Apply* to see more about what the widespread uptake of robots might mean for the future of humans at work: https://www.youtube.com/watch?v=7Pq-S557XQU (15 minutes)

Lesson 8: Worksheet 8.2 – Drive until a black line

In this activity, you will write a program so that your Edison robot will drive forward on a white (reflective) surface until a black (non-reflective) line is crossed.

Look at the following program:

```
1
2    #-------------Setup---------------
3
4    import Ed
5
6    Ed.EdisonVersion = Ed.V2
7
8    Ed.DistanceUnits = Ed.CM
9    Ed.Tempo = Ed.TEMPO_MEDIUM
10
11   #-------Your code below-----------
12
13   Ed.LineTrackerLed(Ed.ON)
14
15   Ed.Drive(Ed.FORWARD,Ed.SPEED_6,Ed.DISTANCE_UNLIMITED)
16
17 ▾ while True:
18 ▾     if Ed.ReadLineState() == Ed.LINE_ON_BLACK:
19           Ed.PlayBeep()
20           Ed.Drive(Ed.STOP,Ed.SPEED_6,0)
```

Look at line 13. This line calls the function Ed.LineTrackerLed() and turns the state to 'on'.

Just like with Edison's obstacle detection beam, to use the line tracking sensor in a program, you must first turn the sensor on. Turning the line tracking sensor on will also activate the line tracker's red LED.

Now, look at line 19. This line calls the Ed.PlayBeep() function. This line doesn't affect the way the line tracking program works. Instead, this line's purpose is for debugging.

Debugging

Debugging is the process of finding 'bugs' or errors in your program. Often, programmers will put lines like line 19 in this program into their code to keep track of the program's flow.

Let's say you run your program, but the robot does not stop at the black line. There are two possible reasons: (1) the robot may not be detecting the black line or (2) there could be a mistake in the final Ed.Drive() command.

If we hear the beep played, we would know that the black line was detected. Therefore, we know that the error was in the next command. This extra debugging code helps us to determine the error more easily.

Other functions could also be used for debugging, such as the Ed.LeftLed() command. For example, you could use this command to turn the left LED on to indicate that a certain point in the program has been reached.

Name_____

Your turn:

Write the program using the EdPy app and download it to your Edison robot. Use the black line on activity sheet 8.1 to test the program. You can also draw a black line on a piece of white paper or use black electrical tape on a white desk.

> **Note:** Whenever you use Edison's line tracking sensor in a program, always start the robot on the white (reflective) surface – never the black (non-reflective) surface.

Put Edison on the white surface and drive the robot towards the black line.

Then try running the program again using each of the three coloured lines on activity sheet 8.1 one at a time. Drive Edison towards each coloured line to test whether or not the robot will detect the line and stop.

1. Are there colours that Edison can't detect very well? If so, which colour(s)?

    _____

2. Why do you think you got the answer you did for question number 1? Why can't Edison detect that colour(s)?

_____

_____

_____

3. Pretend you were programming your Edison robot to drive a slalom course with three slalom flags. Describe how you could use the Ed.PlayBeep(), the Ed.LeftLed() or Ed.RightLed() functions in a program for debugging purposes and what they would do.

_____

_____

_____

_____

_____

_____

_____

Lesson 8: Worksheet 8.3 – Drive inside a border

In this activity, you need to write a program that will keep Edison inside a black border using the robot's line tracking sensor.

First, we need to plan out the program using pseudocode.

Pseudocode

Planning out your program before you begin coding is an important and useful skill in programming.

One way to do this is by using a flowchart to summarise your program flow, which you learned about in lesson 6. Pseudocode is another way to represent your program before you begin coding.

Pseudocode is a way of writing out a program in a simplified, easy to read way. Pseudocode resembles a simplified programming language, but it isn't based on any specific programming language, so it is syntax-free. Instead, pseudocode uses English language to describe what the program will do. That's why pseudocode is also called 'structured English'.

When you plan out your program using pseudocode, you should indent in the same way you would in the programming language so that the pseudocode is easy to read and understand.

Here is an example of some pseudocode describing a program which makes the Edison robot stay within a non-reflective border using the line tracking sensor:

```
Turn the line tracker on
Loop forever
    Drive Forward
    If the robot detects a black line then
        Stop
        Play a beep
        Spin right 135°
```

Here is the corresponding code in EdPy:

```
11  #--------Your code below-----------
12
13  Ed.LineTrackerLed(Ed.ON)
14
15  while True:
16      Ed.Drive(Ed.FORWARD, Ed.SPEED_3, Ed.DISTANCE_UNLIMITED)
17      if Ed.ReadLineState() == Ed.LINE_ON_BLACK:
18          Ed.Drive(Ed.STOP, Ed.SPEED_3, Ed.DISTANCE_UNLIMITED)
19          Ed.PlayBeep()
20          Ed.Drive(Ed.SPIN_RIGHT, Ed.SPEED_5, 135)
21
```

Compare the pseudocode to the program. Do you see how the pseudocode translated into the Python program?

Your turn:

Write the following program:

```
1
2   #-------------Setup----------------
3
4   import Ed
5
6   Ed.EdisonVersion = Ed.V2
7
8   Ed.DistanceUnits = Ed.CM
9   Ed.Tempo = Ed.TEMPO_MEDIUM
10
11  #--------Your code below-----------
12
13  Ed.LineTrackerLed(Ed.ON)
14
15 ▾ while True:
16      Ed.Drive(Ed.FORWARD, Ed.SPEED_3, Ed.DISTANCE_UNLIMITED)
17 ▾    if Ed.ReadLineState() == Ed.LINE_ON_BLACK:
18          Ed.Drive(Ed.STOP, Ed.SPEED_3, Ed.DISTANCE_UNLIMITED)
19          Ed.PlayBeep()
20          Ed.Drive(Ed.SPIN_RIGHT, Ed.SPEED_5, 135)
21
```

Download the program to your Edison robot and run it to see how it works.

Use activity sheet 8.2 as a border to test the program. You can also create your own border using a large piece of paper and a thick black marker or use black electrical tape on a white desk or the floor to create a large border.

Modify the program:
Try removing the stop by removing line 18 of the program. Then experiment with changing the program to use different speeds. Test your modified programs to see what happens.

1. How fast can the robot go before there are problems?

   _____

2. What happens when the robot drives too fast?

_____

_____

_____

Lesson 8: Worksheet 8.4 – Follow a line

In this activity, you will use the Edison robot's line tracker to write a program in EdPy which makes Edison follow any black line.

To do this, we first need to create an algorithm for following a black line.

What is an algorithm?

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.

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

Algorithms in programming

In programming, we often have sets of problems we want to solve.

In this activity, for example, we want to be able to have Edison follow any black line. Our set of problems, therefore, is 'follow any black line'. Any specific line you make for Edison to follow is a new problem inside this set.

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.

The algorithm produces 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.

To solve our set of problems, we need the algorithm to produce a set of instructions that will work for any black line.

You can plan out an algorithm using pseudocode or a flowchart, 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 the line tracker on
Loop forever
    If the robot detects a black line then
        Drive Forward Right
    Else
        Drive Forward Left
```

This algorithm says that if the line tracking sensor is on a non-reflective surface (black), then the Edison robot should drive forwards to the right, moving the sensor towards the white surface. If the line tracking sensor is not on a black surface, then the robot should drive forwards to the left, moving the sensor towards the black surface. In this way, the robot continuously moves forward and tracks the edge of the line.

Notice that no speeds or distances are mentioned in the pseudocode. Those sorts of details are usually left to the coding stage.

Your turn:

Translate the pseudocode algorithm into a Python program to make your Edison robot follow any black line. Experiment with different speeds and distances to achieve the smoothest line tracking. Download your code into Edison and test it using the track on activity sheet 8.2.

1. What was the best combination of speed and distance to achieve smooth line tracking you found?

_____

2. What did your Python program look like? Write your code here.

_____

_____

_____

_____

_____

_____

_____

_____

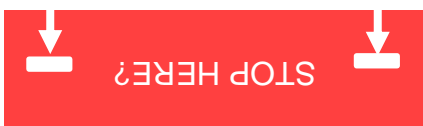### Try it!

Make your own line with a black marker on white paper or black tape. Can Edison follow your line?

Lesson 8: Activity sheet 8.1

STOP HERE

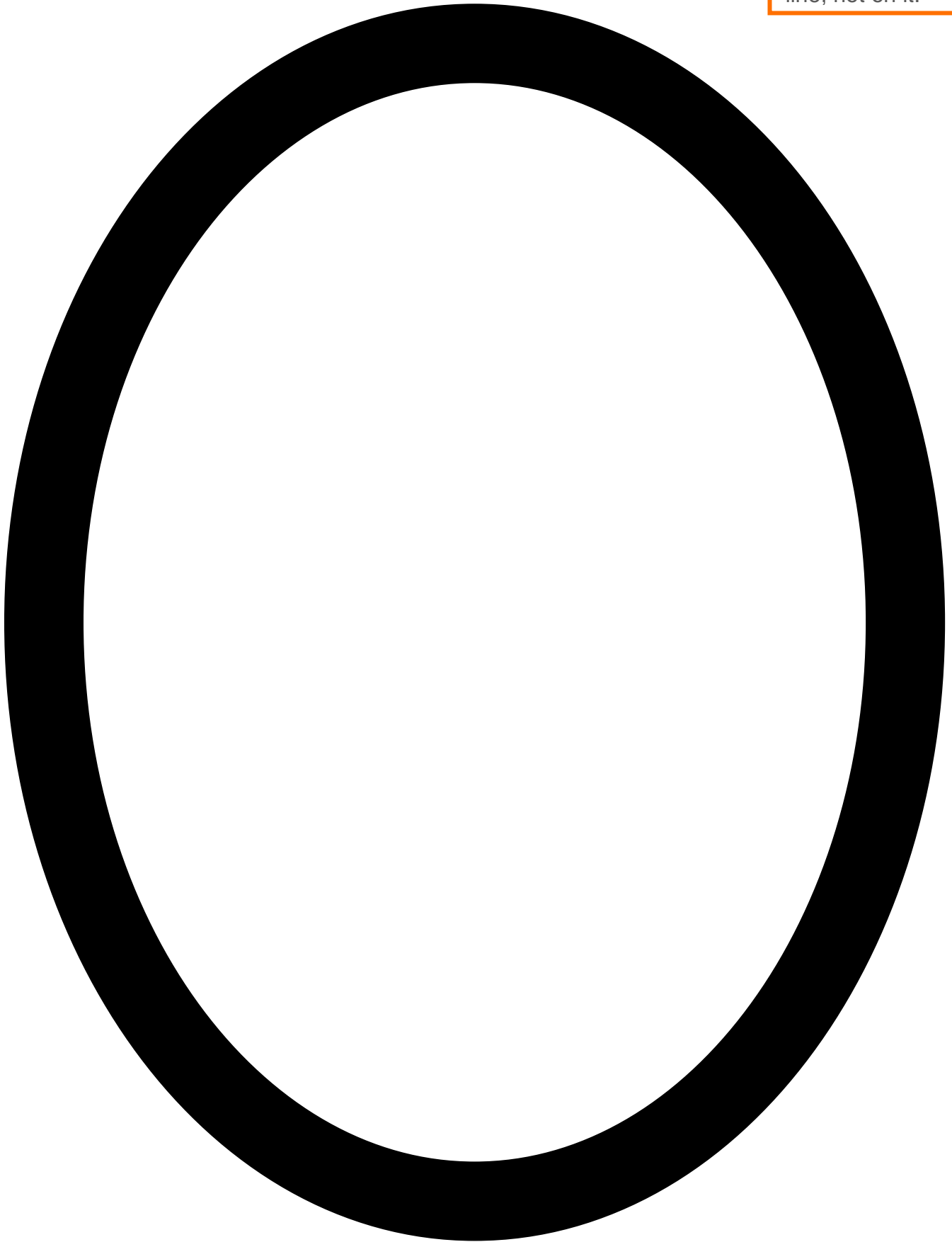STOP HERE?

STOP HERE?

STOP HERE?

Lesson 8: Activity sheet 8.2

Lesson 9: Worksheet 9.1 – Light alarm

In this activity, you will write a program to make your Edison robot sound an alarm when the lights in the room are turned on.

Using Edison's light sensors in programs

Your Edison robot has two light sensors, one on the left and one on the right, which can detect visible light. We can use these sensors to program Edison to respond to light.

In a program, we can have these sensors read the amount of light detected and return this as a digital value.

Look at the following program:

```
1
2   #------------Setup---------------
3
4   import Ed
5
6   Ed.EdisonVersion = Ed.V2
7
8   Ed.DistanceUnits = Ed.CM
9   Ed.Tempo = Ed.TEMPO_MEDIUM
10
11  #-------Your code below----------
12
13 ▾ while Ed.ReadLeftLightLevel() < 100:
14      pass
15 ▾ while True:
16      Ed.PlayBeep()
17
```

This program uses the Ed.ReadLeftLightLevel() function in line 13. This function tells the program to read the light level from the left light sensor on Edison and return a value between 0 and 1023.

Because the Ed.ReadLeftLightLevel() function returns a value, we can do maths to that value. The first loop in this program uses mathematics to determine what to do. This first loop says to pass (in other words, to do nothing) while the value returned by the Ed.ReadLeftLightLevel() function is 'less than' (<) 100.

When the value returned is greater than (>) 100, the program exits the first loop and goes to the next loop, which sounds an alarm continuously.

Your turn:

Write the program and download it to your Edison robot. Place the Edison robot in the dark before you press the play button.

Once you have the lights off or have blocked out Edison's left light sensor, press the play button. When the lights are turned on, or whatever is blocking Edison's left light sensor is removed, the robot will raise the alarm.

1. Think of a real-life situation where this type of alarm would be useful. Describe the situation and how the alarm could be used.

_____

_____

_____

_____

2. What changes need to be made to the program to make it a dark alarm?

_____

_____

_____

_____

Lesson 9: Worksheet 9.2 – Automatic lights

In this activity, you will write a program to have your Edison robot turn the two LED lights on when it gets dark.

Look at the following program:

```
1
2   #------------Setup----------------
3
4   import Ed
5
6   Ed.EdisonVersion = Ed.V2
7
8   Ed.DistanceUnits = Ed.CM
9   Ed.Tempo = Ed.TEMPO_MEDIUM
10
11  #--------Your code below-----------
12  Ed.Drive(Ed.FORWARD, Ed.SPEED_5, Ed.DISTANCE_UNLIMITED)
13
14  while True:
15      if Ed.ReadLeftLightLevel() < 100:
16          activateBothLights(Ed.ON)
17      else:
18          activateBothLights(Ed.OFF)
19
20
21  def activateBothLights(stateOfLed):
22      Ed.LeftLed(stateOfLed)
23      Ed.RightLed(stateOfLed)
24
```

In this program, we are using the 'less than' (<) symbol to determine the path that the program will take.

If the value returned from the Ed.ReadLeftLightLevel() function is less than 100, then the activateBothLights() function is called with the input parameter of Ed.ON. Otherwise, the program moves to the 'else' part of the if statement, which also calls the activateBothLights() function, but with the input parameter of Ed.OFF.

Your turn:

Write the program and download it to your Edison robot. You will need to create some 'tunnels' for Edison to drive through which will block out the light in order to see Edison's front lights come on.

Run the program with Edison driving in and out of tunnels to see how it works.

Then try experimenting with the value (100) in the if statement in line 15 to see what happens when you run the program with a larger or smaller number.

1. What happens when you make the value in the if statement higher?

_____

2. What happens if you make the value in the if statement lower?

_____

Try it!

Will it make a difference if you use the ReadRightLightLevel() function instead of the ReadLeftLightLevel()? Modify your program with this change and test it to see.

Lesson 9: Worksheet 9.3 – Light following

In this activity, you will write a program so that your Edison robot will follow the light from a torch (flashlight).

Look at the following program:

```
1
2   #------------Setup----------------
3
4   import Ed
5
6   Ed.EdisonVersion = Ed.V2
7
8   Ed.DistanceUnits = Ed.CM
9   Ed.Tempo = Ed.TEMPO_MEDIUM
10
11  #--------Your code below-----------
12
13 ▾ while True:
14 ▾     if Ed.ReadRightLightLevel() - Ed.ReadLeftLightLevel() < 0:
15             Ed.Drive(Ed.FORWARD_LEFT, Ed.SPEED_5, Ed.DISTANCE_UNLIMITED)
16 ▾     else:
17             Ed.Drive(Ed.FORWARD_RIGHT, Ed.SPEED_5, Ed.DISTANCE_UNLIMITED)
18
```

This program compares the light level between the right light sensor and the left light sensor to determine the flow of the program.

The presence of the torch on either the left or right of the robot will cause the robot to read a higher light level on that side of the robot. The logic of this program says that when the right light level minus the left light level is less than zero, the robot drives left towards the higher source of light, else the robot drives to the right.

Your turn:

Task 1: Trace the program

In worksheet 5.2, you learned about 'tracing' a program. When there are lots of calculation and different possible values in a program, it can be useful to trace the program. This allows you to understand the different values that can occur and predict the associated behaviour.

1. Fill out the following tracing table for this program. The expected behaviour should either be 'drive forward right' or 'drive forward left'.

	Right_Light	Left_Light	Expected behaviour
Torch on right	200	100	
Torch on left	100	200	
No torch	100	100	

Task 2: Write and run the program

Write the program and download it to your Edison robot. After you press the play button, shine a torch at Edison. The robot will drive towards the light. Use the torch to control where Edison drives.

2. What happens if you change the 'less than' symbol (<) to a greater than symbol (>) in this program?

_____

_____

What do you think?

The light-following program above demonstrates a robotic behaviour that is very similar to that of a moth attracted to a street light on a warm night.

Is a moth which is attracted to light intelligent? What about a robot with the same behaviour?

Why is an insect that is attracted to light alive, but not a robot?

Lesson 10: Worksheet 10.1 – Vampire robot

In this challenge-based activity, you will apply everything you have learned so far to create a program with lots of interesting behaviours for your Edison robot to perform while transformed into a vampire robot.

To turn your Edison into a vampire robot, we will use object-oriented code.

Objects and classes in Python

Python is an object-oriented programming language. Object-oriented code is a powerful way of reducing complexity inside a program. That's why many programmers use object-oriented code in large, complicated programs.

In an object-oriented programming language, rather than just using standalone variables and functions, programs can also use 'objects'.

An object is a collection of functions and variables which are related to each other. Objects are defined by a 'class'.

A class (also called a class definition) is like a blueprint that describes how to make an object.

In a program, you can create lots of different objects from a class and then manipulate these objects individually in your program for different purposes. All of the objects created from the class will have the same 'blueprint' when they are created, but you can then do different things to different objects throughout your program.

Example: class 'Shape'

The most common example of a class is a Shape.

All shapes, like squares, triangles and hexagons, have common attributes. For example, all shapes have some number of sides and some number of vertices. There are also common things you do with shapes, like find the area or the perimeter of the shape.

We can use these common elements to create a class for Shape in Python. By creating the class, we are setting up a blueprint with just the known, common elements. We will then be able to create many different objects from that class, such as 'square' and 'triangle'.

The syntax for defining a class in Python is 'class' and then the name you want to give that class, followed by a colon (:). For example:

```python
class Shape:
```

Anything you include as indented code in the lines following this is inside the class definition.

Look at the following example of a class for Shape in Python:

```python
class Shape:
    def __init__(self,x,y):
        self.x = x
        self.y = y

    def area(self):
        return self.x * self.y

    def perimeter(self):
        return 2 * self.x + 2 * self.y
```

This class says that the blueprint for any object created from class Shape will have two input parameters (x and y). The class also defines three functions: 'init', 'area' and 'perimeter'.

The __init__ function and 'self'
Each class must always have an __init__ function.

The __init__ function is start-up code. The term 'init' stands for 'initialisation'.

When the program needs to create an object, the program goes into the class definition and runs the __init__ function. This function contains code which sets up the object, which typically means setting the start values of the object variables.

Functions in a class must always have 'self' as their first parameter. This tells the created object to use its own functions and variables.

Example: object 'rectangle'
Once we have created a class definition, we can use it to create objects in our program.

Let's say we want to create an object 'rectangle' in class Shape.

In Python, when you want to create an object from a class, you use the syntax object_name = class_name(parameters).

Look at the example of creating object 'rectangle':

```python
rectangle = Shape(100,45)
```

In a program, this calls the __init__ function, which runs, creating an object 'rectangle' using the blueprint from the Shape class definition. The object 'rectangle' is set to have 100 be the 'x' input value and 45 as its 'y' input value. (You don't need to put in 'self' – that's only a reference for the program to know where to look.)

Once the program creates the object, it can access that object's functions as defined by the class.

To call one of these functions in Python, you use the syntax object_name.function_name(parameters).

For example:

```
rectangle.area()
```

In this example, the rectangle.area() function will return the value of the rectangle object's 'x' input parameter multiplied by its 'y' input parameter.

One way to use this particular function would be to store this value in a variable in your program. For example:

```
AreaOfRectangle = rectangle.area()
```

Setting up a class in EdPy

In EdPy, we can use a class to create different objects in a program for Edison.

Here is a simple class for a Vampire robot with an __init__ function and two other functions: one for daytime behaviour for the robot and one for night time behaviour. The Vampire class also contains a variable for the age of the robot:

```python
class Vampire:

    def __init__(self,age):
        self.age = age

    def doDayTimeBehaviour(self):
        Ed.LeftLed(Ed.OFF)
        Ed.RightLed(Ed.OFF)
        Ed.PlayTone(Ed.NOTE_A_7, self.age)
        while Ed.ReadMusicEnd() == Ed.MUSIC_NOT_FINISHED:
            pass
        Ed.TimeWait(1, Ed.TIME_SECONDS) # for debugging purposes and as a placeholder

    def doNightTimeBehaviour(self):
        Ed.RightLed(Ed.ON)
        Ed.LeftLed(Ed.ON) # for debugging purposes and as a placeholder
```

This class definition will allow you to create different Vampire objects.

You can then use these Vampire object in a program for your Edison robot.

Write the following code and check that you understand what is happening in the program:

```
1
2   #-------------Setup----------------
3
4   import Ed
5
6   Ed.EdisonVersion = Ed.V2
7
8   Ed.DistanceUnits = Ed.CM
9   Ed.Tempo = Ed.TEMPO_MEDIUM
10
11  #--------Your code below-----------
12
13  myEdisonVampire = Vampire(21)
14
15 ▾ while True:
16      # if it is the daytime
17 ▾    if Ed.ReadLeftLightLevel() > 100:
18          myEdisonVampire.doDayTimeBehaviour()
19      #it is night time
20 ▾    else:
21          myEdisonVampire.doNightTimeBehaviour()
22
23 ▾ class Vampire:
24
25 ▾    def __init__(self,age):
26          self.age = age
27
28 ▾    def doDayTimeBehaviour(self):
29          Ed.LeftLed(Ed.OFF)
30          Ed.RightLed(Ed.OFF)
31          Ed.PlayTone(Ed.NOTE_A_7, self.age)
32 ▾        while Ed.ReadMusicEnd() == Ed.MUSIC_NOT_FINISHED:
33              pass
34          Ed.TimeWait(1, Ed.TIME_SECONDS) # for debugging purposes and as a placeholder
35
36 ▾    def doNightTimeBehaviour(self):
37          Ed.RightLed(Ed.ON)
38          Ed.LeftLed(Ed.ON) # for debugging purposes and as a placeholder
39
```

Your turn:

Task 1: Design the Vampire behaviours

Now it is time to design some more interesting behaviours for your Vampire robot. Be creative! You can make your robot react to button pushes, claps and obstacles. Consider using event handling in your program. You can make multiple vampire objects with whatever ages you want. You can also modify the base program so that your objects have multiple input parameters.

Think about what you want to include in your program, then work through your program design by first creating flowcharts and then pseudocode based off of the flowcharts.

Step 1: Flowcharts

Draw flowcharts for the daytime and nighttime behaviours, either by hand or using a program like Google Slides or a similar drawing program. Include your flowcharts here. Use extra paper if you need.

Name_____

Step 2: Pseudocode

Translate your flowcharts into pseudocode. Write your pseudocode here. Use extra paper if you need.

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

Task 2: Write your program

Translate your pseudocode into Python code using the EdPy app. Download it to your Edison robot and run your program to test it out.

Task 3: Problem-solving

In programming, it is very common not to have everything work out the way you think it will the first time. No sweat! Remember this: Thomas Edison famously failed 10,000 times before he successfully invented the light bulb. So, *keep trying!*

We learn from the problems we encounter, so it is a good habit to take time to think about what went wrong. Describe any problems you had, what ideas you came up with to fix them, and the solutions you found in later attempts. Use extra paper, if you need.

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

Task 4: Describe some of the programming structures in your program

Choose three different programming structures you used in your program. Describe what each one does below.

1. Name of the programming structure:
   _____

   What does it do?

   _____

   _____

   _____

   _____

2. Name of the programming structure:
   _____

   What does it do?

   _____

   _____

   _____

   _____

3. Name of the programming structure:
   _____

   What does it do?

   _____

   _____

   _____

   _____

Task 5: Present your program

Demonstrate your Vampire robot to your partner, group or class. Talk about your ideas, the program, the problems that you encountered and how you solved them.