EdScratch warning messages

This guide includes all the warning messages that can be displayed in the bug box of the EdScratch language app, available at www.edscratchapp.com. Each message is included with an explanation of what the message means and examples of when you may encounter the message.

There are two types of warnings: red messages and yellow messages. When a red message is displayed, the program cannot be downloaded to the Edison robot. When a yellow message is displayed, the program can be downloaded to the Edison robot, but may or may not work as expected.

Red messages: Red warning messages are like 'stop' messages. These messages are EdScratch saying "Sorry! This program won't make sense to Edison." If there are any red warning messages in the bug box, you will not be able to download the program to Edison until the issue is resolved.

- See all red messages

Yellow messages: Yellow warning messages are caution messages. This is EdScratch saying "Heads up! This might not work the way you want it to work." You can download a program even if there are yellow messages in the bug box, but if your program does not work the way you expect it to in Edison, it is wise to review any yellow messages. These can often help you discover any logical errors in the program.

- See all yellow messages

Red messages

This section contains all of the red messages that can appear in EdScratch.

Red warning messages are like ‘stop’ messages. These messages are EdScratch saying “Sorry! This program won’t make sense to Edison.” If there are any red warning messages in the bug box, you will not be able to download the program to Edison until the issue is resolved.

'Backwards until' blocks need a condition input.

Meaning:

The Drive category ‘backwards until’ block requires a diamond-shaped input to be added to this block in the diamond-shaped hole. This input gives the block the ‘drive until’ condition.

Examples:

The ‘backwards until’ block is attached to the ‘start’ block but no condition (i.e. diamond-shaped input) has been put into the ‘backwards until’ block.

The ‘backwards until’ block is attached to the ‘any obstacle detected’ event block but no condition (i.e. diamond-shaped input) has been put into the ‘backwards until’ block.

'Bit shift left' blocks need a variable.

Meaning:

The Data category ‘bit shift left’ block requires a variable to be input into the greyed-out round hole in the block. This tells the block what variable holds the value that the program should bit shift left.

Examples:

The ‘bit shift left’ block is attached to the ‘start’ block but no variable has been put into the greyed-out round hole in the block.

A variable has been added to the round input telling the block by how much to bit shift left, but no variable has been put into the greyed-out round hole in the block.

'Bit shift right' blocks need a variable.

Meaning:

The Data category ‘bit shift right’ block requires a variable to be input into the greyed-out round hole in the block. This tells the block what variable holds the value that the program should bit shift right.

Examples:

The ‘bit shift right’ block is attached to the ‘start’ block but no variable has been put into the greyed-out round hole in the block.

A variable has been added to the round input telling the block by how much to bit shift right, but no variable has been put into the greyed-out round hole in the block.

'Decrement variable' blocks need a variable.

Meaning:

The Data category ‘decrement variable’ block requires a variable to be input into the greyed-out round hole in the block. This tells the block what variable holds the value that the program should decrement (reduce by one).

Examples:

The ‘decrement variable’ block is attached to the ‘start’ block but no variable has been put into the greyed-out round hole in the block.

The ‘decrement variable’ block is attached to the ‘any obstacle detected’ event block but no variable has been put into the greyed-out round hole in the block.

Edison cannot detect claps while driving because the motors are too noisy.

Meaning:

When the Edison robot’s motors are on, these motors make noise. The robot’s sound sensor cannot differentiate the noise caused by the motors with other sounds, such as claps. For this reason, the robot is unable to drive and detect claps simultaneously.

Examples:

The ‘forwards until’ block from the Drive category is in the program with the ‘clap detected’ diamond-shaped input block set as the ‘forwards until’ block’s input condition.

The ‘backwards until’ block from the Drive category is in the program with the ‘clap detected’ diamond-shaped input block set as the ‘backwards until’ block’s input condition.

'Forwards until' blocks need a condition input.

Meaning:

The Drive category ‘forwards until’ block requires a diamond-shaped input to be added to this block in the diamond-shaped hole. This input gives the block the ‘drive until’ condition.

Examples:

The ‘forwards until’ block is attached to the ‘start’ block but no condition (i.e. diamond-shaped input) has been put into the ‘forwards until’ block.

The ‘forwards until’ block is attached to the ‘any obstacle detected’ event block but no condition (i.e. diamond-shaped input) has been put into the ‘forwards until’ block.

'If' blocks need a condition input.

Meaning:

The Control category ‘if’ block requires a diamond-shaped input to be added to this block in the diamond-shaped hole. This input gives the block the ‘if’ condition.

Examples:

The ‘if’ block is attached to the ‘start’ block but no condition (i.e. diamond-shaped input) has been put into the ‘if’ block.

The ‘if’ block is attached to the ‘any obstacle detected’ event block but no condition (i.e. diamond-shaped input) has been put into the ‘if’ block.

'If-else' blocks need a condition input.

Meaning:

The Control category ‘if-else’ block requires a diamond-shaped input to be added to this block in the diamond-shaped hole. This input gives the block the ‘if’ condition.

Examples:

The ‘if-else’ block is attached to the ‘start’ block but no condition (i.e. diamond-shaped input) has been put into the ‘if-else’ block.

The ‘if-else’ block is attached to the ‘any obstacle detected’ event block but no condition (i.e. diamond-shaped input) has been put into the ‘if-else’ block.

'Increment variable' blocks need a variable.

Meaning:

The Data category ‘increment variable’ block requires a variable to be input into the greyed-out round hole in the block. This tells the block what variable holds the value that the program should increment (increase by one).

Examples:

The ‘increment variable’ block is attached to the ‘start’ block but no variable has been put into the greyed-out round hole in the block.

The ‘increment variable’ block is attached to the ‘any obstacle detected’ event block but no variable has been put into the greyed-out round hole in the block.

'Left until' blocks need a condition input.

Meaning:

The Drive category ‘left until’ block requires a diamond-shaped input to be added to this block in the diamond-shaped hole. This input gives the block the ‘drive until’ condition.

Examples:

The ‘left until’ block is attached to the ‘start’ block but no condition (i.e. diamond-shaped input) has been put into the ‘left until’ block.

The ‘left until’ block is attached to the ‘any obstacle detected’ event block but no condition (i.e. diamond-shaped input) has been put into the ‘left until’ block.

'Line detection' blocks do not work unless the line detection LED is turned on using the 'line tracking LED' block from the 'Sensing' category.

Meaning:

The ‘line tracker on’ diamond-shaped input block from the Sensing category can be used to set a condition for another block, such as an ‘until’ block or an ‘if’ block. In order for Edison to use its line tracking sensor in this way, the LED in the line tracker needs to be turned on. Include the ‘turn line tracking LED’ block from the Sensing category set to ‘on’ before the ‘line tracker on’ diamond-shaped input block condition in your program.

Examples:

The ‘line tracker on’ diamond-shaped input block is set as the condition in a ‘if’ block but the ‘turn line tracking LED’ block is not included in the program.

The ‘line tracker on’ diamond-shaped input block is set as the condition in a ‘if’ block and there is a ‘turn line tracking LED’ block in the program, but not before the ‘if’ block in the program.

'Line detection' events do not work unless the line detection LED is turned on using the 'line tracking LED' block from the 'Sensing' category in the main program.

Meaning:

All of the Event category blocks related to the line tracker require that the LED in the Edison robot’s line tracker be turned on. Include the ‘turn line tracking LED’ block from the Sensing category set to ‘on’ in the main program (the stack attached to the yellow ‘start’ block) when using any of these events in your program.

Examples:

The ‘line tracker on reflective surface’ event block is in the programming space but the ‘turn line tracking LED’ block is not included in the main program.

The ‘line tracker on reflective surface’ event block is in the programming space and the ‘turn line tracking LED’ block is attached to the ‘line tracker on reflective surface’ event block but is not included in the main program.

'Obstacle detection' blocks do not work unless the obstacle detection beam is turned on using the 'obstacle detection beam' block from the 'Sensing' category.

Meaning:

The ‘obstacle detected’ diamond-shaped input block from the Sensing category can be used to set a condition for another block, such as an ‘until’ block or an ‘if’ block. In order for Edison to use its obstacle detection sensor in this way, the obstacle detection beam needs to be turned on. Include the ‘turn obstacle detection beam’ block from the Sensing category set to ‘on’ before the ‘obstacle detected’ diamond-shaped input block condition in your program.

Examples:

The ‘obstacle detected’ diamond-shaped input block is set as the condition in a ‘if’ block but the ‘turn obstacle detection beam’ block is not included in the program.

The ‘obstacle detected’ diamond-shaped input block is set as the condition in a ‘if’ block and there is a ‘turn obstacle detection beam’ block in the program, but not before the ‘if’ block in the program.

'Obstacle detection' events do not work unless the obstacle detection beam is turned on using the 'obstacle detection beam' block from the 'Sensing' category in the main program.

Meaning:

All of the Event category blocks related to obstacle detection require that the Edison robot’s obstacle detection beam be turned on. Include the ‘turn obstacle detection beam’ block from the Sensing category set to ‘on’ in the main program (the stack attached to the yellow ‘start’ block) when using any of these events in your program.

Examples:

The ‘any obstacle detected’ event block is in the programming space but the ‘turn obstacle detection beam’ block is not included in the main program.

The ‘any obstacle detected’ event block is in the programming space and the ‘turn obstacle detection beam’ block is attached to the ‘any obstacle detected’ event block but is not included in the main program.

'Play music in background' grouping blocks can only accept 'note' blocks.

Meaning:

The Sound category ‘play music in background’ block allows musical notes to be played while other commands, such as driving commands, are carried out. The only blocks that can sit inside the ‘play music in background’ block are musical note blocks.

Examples:

There is a ‘beep’ block inside the ‘play music in background’ block.

There is a ‘set music tempo’ block inside the ‘play music in background’ block.

'Play music in background' grouping blocks need to contain at least one 'note' block.

Meaning:

The Sound category ‘play music in background’ block allows musical notes to be played while other commands, such as driving commands, are carried out. The ‘play music in background’ block must have at least one musical note block inside the grouping block to function.

Examples:

The ‘play music in background’ block is attached to the ‘start’ block but no blocks have been added inside of it.

The ‘play music in background’ block is attached to the ‘any obstacle detected’ event block but no blocks have been added inside of it.

Programs must have a 'start' event to work correctly. If you are seeing this error message, start a new program by going to the menu and selecting 'new'.

Meaning:

The yellow ‘start’ block is required in every EdScratch program, which is why it is a persistent block that is always in the programming area. In incredibly rare situations, the EdScratch app may not load properly and the ‘start’ block may not appear. The application should be reloaded.

Examples:

There was a critical local internet failure as the EdScratch app loaded, preventing the app from fully loading properly.

There was a critical local programming device failure as the EdScratch app loaded, preventing the app from fully loading properly.

'Repeat until' blocks need a condition input.

Meaning:

The Control category ‘repeat until’ block requires a diamond-shaped input to be added to this block in the diamond-shaped hole. This input gives the block the ‘repeat until’ condition.

Examples:

The ‘repeat until’ block is attached to the ‘start’ block but no condition (i.e. diamond-shaped input) has been put into the ‘repeat until’ block.

The ‘repeat until’ block is attached to the ‘any obstacle detected’ event block but no condition (i.e. diamond-shaped input) has been put into the ‘repeat until’ block.

'Right until' blocks need a condition input.

Meaning:

The Drive category ‘right until’ block requires a diamond-shaped input to be added to this block in the diamond-shaped hole. This input gives the block the ‘drive until’ condition.

Examples:

The ‘right until’ block is attached to the ‘start’ block but no condition (i.e. diamond-shaped input) has been put into the ‘right until’ block.

The ‘right until’ block is attached to the ‘any obstacle detected’ event block but no condition (i.e. diamond-shaped input) has been put into the ‘right until’ block.

'Set variable' blocks need a variable.

Meaning:

The Data category ‘set variable’ block requires a variable to be input into the greyed-out round hole in the block. This tells the block what variable to set to the value specified in the second input in the ‘set variable’ block.

Examples:

The ‘set variable’ block is attached to the ‘start’ block but no variable has been put into the greyed-out round hole in the block.

A variable has been added to the round input telling the block the value to set the variable to be, but no variable has been put into the greyed-out round hole in the block.

There are no blocks connected to the 'start' event, so there is no main program for Edison to run.

Meaning:

There must be a main program in order to run an EdScratch program in the Edison robot. Any blocks attached to the yellow ‘start’ block in EdScratch form the main program that Edison will run when the program is compiled and sent to Edison. When there are no blocks attached to the ‘start’ block, there is no main program.

Examples:

A program has been built, compiled (i.e. the ‘program Edison’ button has been pressed) and then all of the blocks that were attached to ‘start’ block have been removed, leaving nothing attached to the ‘start’ block in the programming area.

A ‘clap detected’ event block with a subroutine has been built but no blocks have been attached to the ‘start’ block and the ‘program Edison’ button has been pressed.

'Wait milliseconds' blocks need an input value.

Meaning:

The Control category ‘wait milliseconds’ block requires an oval-shaped input to be added to this block in the oval-shaped hole. This input gives the block the value for which to wait.

Examples:

The ‘wait milliseconds’ block is attached to the ‘start’ block but no value (i.e. oval-shaped input) has been put into the ‘wait milliseconds’ block.

The ‘wait milliseconds’ block is attached to the ‘any obstacle detected’ event block but no value (i.e. oval-shaped input) has been put into the ‘wait milliseconds’ block.

'Wait until' blocks need a condition input.

Meaning:

The Control category ‘wait until’ block requires a diamond-shaped input to be added to this block in the diamond-shaped hole. This input gives the block the ‘wait until’ condition.

Examples:

The ‘wait until’ block is attached to the ‘start’ block but no condition (i.e. diamond-shaped input) has been put into the ‘wait until’ block.

The ‘wait until’ block is attached to the ‘any obstacle detected’ event block but no condition (i.e. diamond-shaped input) has been put into the ‘wait until’ block.

Yellow messages

This section contains all of the yellow messages that can appear in EdScratch.

Yellow warning messages are caution messages. This is EdScratch saying “Heads up! This might not work the way you want it to work.” You can download a program even if there are yellow messages in the bug box, but if your program does not work the way you expect it to in Edison, it is wise to review any yellow messages. These can often help you discover any logical errors in the program.

A 'forever' loop will continue looping until you press the square button on Edison.

Meaning:

The Control category ‘forever’ loop block is an indefinite loop with no in-built end point. When run in the Edison robot, this loop will repeat indefinitely until either the ‘stop’ (square) button on the robot is pressed or the batteries in the robot run flat. The program will not move on to any commands sequentially after the ‘forever’ loop.

Examples:

The ‘forever’ loop block is inside of a ‘repeat’ loop block in a program.

The ‘forever’ loop block is inside of an ‘if’ block in a program.

An 'operator' block inside a 'drive backwards until' block may cause Edison to drive forever or skip the drive block.

Meaning:

The ‘backwards until’ block from the Drive category requires a diamond-shaped input block to set the ‘until’ condition and can accept a diamond-shaped input block from the Operators category. All of the diamond-shaped input blocks from the Operators category are expressions that will resolve to be either ‘true’ or ‘false’. Depending on the inputs in the ‘operator’ block, the expression may always be ‘true’ or always be ‘false’ (if, for example, the inputs of the operator don’t change while the motors are moving). This perpetual state can cause the ‘drive’ block to continue driving indefinitely or to never satisfy the required condition resulting in the drive block being skipped.

Examples:

The ‘greater than’ operator block is in the conditional input of a ‘backwards until’ block in a program.

The ‘is not the same as’ operator block is in the conditional input of a ‘backwards until’ block in a program.

An 'operator' block inside a 'drive forwards until' block may cause Edison to drive forever or skip the drive block.

Meaning:

The ‘forwards until’ block from the Drive category requires a diamond-shaped input block to set the ‘until’ condition and can accept a diamond-shaped input block from the Operators category. All of the diamond-shaped input blocks from the Operators category are expressions that will resolve to be either ‘true’ or ‘false’. Depending on the inputs in the ‘operator’ block, the expression may always be ‘true’ or always be ‘false’ (if, for example, the inputs of the operator don’t change while the motors are moving). This perpetual state can cause the ‘drive’ block to continue driving indefinitely or to never satisfy the required condition resulting in the drive block being skipped.

Examples:

The ‘greater than’ operator block is in the conditional input of a ‘forwards until’ block in a program.

The ‘is not the same as’ operator block is in the conditional input of a ‘forwards until’ block in a program.

An 'operator' block inside a 'drive left until' block may cause Edison to drive forever or skip the drive block.

Meaning:

The ‘left until’ block from the Drive category requires a diamond-shaped input block to set the ‘until’ condition and can accept a diamond-shaped input block from the Operators category. All of the diamond-shaped input blocks from the Operators category are expressions that will resolve to be either ‘true’ or ‘false’. Depending on the inputs in the ‘operator’ block, the expression may always be ‘true’ or always be ‘false’ (if, for example, the inputs of the operator don’t change while the motors are moving). This perpetual state can cause the ‘drive’ block to continue driving

indefinitely or to never satisfy the required condition resulting in the drive block being skipped.

Examples:

The ‘greater than’ operator block is in the conditional input of a ‘left until’ block in a program.

The ‘is not the same as’ operator block is in the conditional input of a ‘left until’ block in a program.

An 'operator' block inside a 'drive right until' block may cause Edison to drive forever or skip the drive block.

Meaning:

The ‘right until’ block from the Drive category requires a diamond-shaped input block to set the ‘until’ condition and can accept a diamond-shaped input block from the Operators category. All of the diamond-shaped input blocks from the Operators category are expressions that will resolve to be either ‘true’ or ‘false’. Depending on the inputs in the ‘operator’ block, the expression may always be ‘true’ or always be ‘false’ (if, for example, the inputs of the operator don’t change while the motors are moving). This perpetual state can cause the ‘drive’ block to continue driving indefinitely or to never satisfy the required condition resulting in the drive block being skipped.

Examples:

The ‘greater than’ operator block is in the conditional input of a ‘right until’ block in a program.

The ‘is not the same as’ operator block is in the conditional input of a ‘right until’ block in a program.

An 'operator' block inside a 'repeat until' block may cause Edison to loop forever or skip the 'repeat' block.

Meaning:

The ‘repeat until’ block from the Control category requires a diamond-shaped input block to set the ‘until’ condition and can accept a diamond-shaped input block from the Operators category. All of the diamond-shaped input blocks from the Operators category are expressions that will resolve to be either ‘true’ or ‘false’. Depending on the inputs in the ‘operator’ block, the expression may always be ‘true’ or always be

‘false’ (if, for example, the inputs of the operator don’t change while the blocks inside the repeat loop are running). This perpetual state can cause the ‘repeat’ block to continue looping indefinitely or to never satisfy the required condition resulting in the repeat loop and all blocks it contains to be skipped.

Examples:

The ‘greater than’ operator block is in the conditional input of a ‘repeat until’ block in a program.

The ‘is not the same as’ operator block is in the conditional input of a ‘repeat until’ block in a program.

An 'operator' block inside a 'repeat' block may cause Edison to skip the 'repeat' block.

Meaning:

The ‘repeat’ block from the Control category requires a value input and can accept an oval-shaped input block from the Operators category. All of the oval-shaped input blocks from the Operators category resolve to a numeric value. Depending on the inputs in the ‘operator’ block, the value can be zero, in which case the repeat loop and all blocks it contains will be skipped.

Examples:

The ‘divided by’ operator block is in the value input of a ‘repeat’ block in a program.

The ‘minus’ operator block is in the value input of a ‘repeat’ block in a program.

An 'operator' block inside a 'wait until' block may cause Edison to wait forever or skip the 'wait' block.

Meaning:

The ‘wait until’ block from the Control category requires a diamond-shaped input block to set the ‘until’ condition and can accept a diamond-shaped input block from the Operators category. All of the diamond-shaped input blocks from the Operators category are expressions that will resolve to be either ‘true’ or ‘false’. Depending on the inputs in the ‘operator’ block, the expression may always be ‘true’ or always be ‘false’ (if, for example, the inputs of the operator don’t change while the wait block is running). This perpetual state can cause the ‘wait’ block to continue waiting indefinitely or to never satisfy the required condition resulting in the wait block being skipped.

Examples:

The ‘greater than’ operator block is in the conditional input of a ‘wait until’ block in a program.

The ‘is not the same as’ operator block is in the conditional input of a ‘wait until’ block in a program.

An 'operator' block inside a 'wait' block may cause Edison to skip the 'wait' block.

Meaning:

The ‘wait milliseconds’ block from the Control category requires a value input and can accept an oval-shaped input block from the Operators category. All of the oval-shaped input blocks from the Operators category resolve to a numeric value. Depending on the inputs in the ‘operator’ block, the value can be zero, in which case the wait loop will be skipped.

Examples:

The ‘divided by’ operator block is in the value input of a ‘wait milliseconds’ block in a program.

The ‘minus’ operator block is in the value input of a ‘wait milliseconds’ block in a program.

Blocks not connected to a yellow 'event' block will not be programmed into Edison.

Meaning:

Blocks from the block pallet can be added into the programming area in EdScratch but not attached to any other blocks. Blocks that are not attached to the yellow ‘start’ block or a yellow block from the Events category will be ignored by the complier when the ‘program Edison’ button is pressed and will not be sent to the robot.

Examples:

A ‘beep’ block is in the programming area of EdScratch but not attached to any other blocks.

A stack of blocks, all connected, is in the programming area but is not attached to the ‘start’ block nor a yellow block from the Events category.

Comments are notes to help keep track of things. Comment blocks will not be programmed into Edison.

Meaning:

The pink blocks added from the Comment category into a program in EdScratch do not compile and will not be sent to the Edison robot when the ‘program Edison’ button is pressed. Comments are like sticky-notes: little memos that are jotted down to help the programmer. These blocks serve as a way for programmers using EdScratch to add annotation notes to their code and will always be ignored by the complier. Comments will not affect the program in the robot in any way.

Examples:

A ‘comment’ block is attached to the ‘start’ block in a program.

A ‘comment’ block is attached to the ‘any obstacle detected’ event block in a program.

Consider turning the line detection LED on using the 'line tracking LED' block from the 'Sensing' category if measuring reflectiveness of the driving surface.

Meaning:

The line tracking sensor on the Edison robot is comprised of multiple parts, including a phototransistor and a LED. The phototransistor is always on and capable of detecting visible light, but when the bottom of the robot is very close to an opaque surface (for example, while driving on a desk) this phototransistor will detect very little light. In order to best measure the reflectiveness of the driving surface, it may be necessary to turn on the line tracking LED using the ‘turn line tracking LED’ block from the Sensing category.

Examples:

An ‘if’ block from the Control category contains a diamond-shaped ‘operator’ block as its conditional input. One of the input values of that ‘operator’ block is a ‘light level of sensor’ oval-shaped value input block from the Sensing category which is set to ‘line tracking’.

A ‘wait until’ block from the Control category contains a diamond-shaped ‘operator’ block as its conditional input. One of the input values of that ‘operator’ block is a ‘light level of sensor’ oval-shaped value input block from the Sensing category which is set to ‘line tracking’.

Driving the motors creates noise which may cause the 'clap' event to trigger. This may cause the 'clap event' blocks to trigger repeatedly while Edison is driving.

Meaning:

When the Edison robot's motors are on, these motors make noise. The robot's sound sensor cannot differentiate the noise caused by the motors with other sounds, such as claps. For this reason, if a 'clap detected' event from the Events category in EdScratch is being used and there are also 'drive' commands in the program, the 'clap detected' event may repeatedly trigger while the 'drive' commands are being executed.

Examples:

The main program contains 'drive' blocks and there is also a subroutine triggered by the 'clap detected' Event category block.

There is a subroutine triggered by the 'any obstacle detected' Event category block which contains 'drive' blocks and there is also a 'clap detected' Event category block.

Edison cannot receive messages while 'obstacle detection' is on. Make sure to turn obstacle detection off in the program before attempting to receive a message.

Meaning:

Edison robots use the same sensor components to receive messages and detect obstacles, and the sensor components cannot do both simultaneously. In order to receive an IR message from another Edison robot, obstacle detection must be off. This can be controlled using the 'turn obstacle detection beam' block from the Sensing category.

Examples:

The main program contains a 'turn obstacle detection beam' block from the Sensing category and also contains a diamond-shaped 'IR message detected' block as a conditional input in an 'if' block from the Control category later in the program.

The main program contains a 'turn obstacle detection beam' block from the Sensing category. There is also a subroutine triggered by a 'clap detected' Event category block which contains a diamond-shaped 'IR message detected' block as a conditional input in an 'if' block from the Control category.

Edison cannot receive messages while 'obstacle detection' is on. Make sure to turn obstacle detection off in the main program before attempting to receive a message.

Meaning:

Edison robots use the same sensor components to receive messages and detect obstacles, and the sensor components cannot do both simultaneously. In order to receive an IR message from another Edison robot, obstacle detection must be off. This can be controlled using the ‘turn obstacle detection beam’ block from the Sensing category.

Examples:

The main program contains a ‘turn obstacle detection beam’ block from the Sensing category. There is also a subroutine triggered by a ‘IR message received’ Event category block.

The main program contains multiple ‘turn obstacle detection beam’ blocks from the Sensing category. There is also a subroutine triggered by a ‘IR message received’ Event category block.

Edison cannot receive remote control codes while 'obstacle detection' is on. Make sure to turn obstacle detection off in the program before attempting to receive a remote code.

Meaning:

Edison robots use the same sensor components to receive IR signals from remote controls and detect obstacles, and the sensor components cannot do both simultaneously. In order to receive an IR message from a TV or DVD remote control, obstacle detection must be off. This can be controlled using the ‘turn obstacle detection beam’ block from the Sensing category.

Examples:

The main program contains a ‘turn obstacle detection beam’ block from the Sensing category and also contains a diamond-shaped ‘remote code received’ block as a conditional input in an ‘if’ block from the Control category later in the program.

The main program contains a ‘turn obstacle detection beam’ block from the Sensing category. There is also a subroutine triggered by a ‘clap detected’ Event category block which contains a diamond-shaped ‘remote code received’ block as a conditional input in an ‘if’ block from the Control category.

Edison cannot receive remote control codes while 'obstacle detection' is on. Make sure to turn obstacle detection off in the main program before attempting to receive a remote code.

Meaning:

Edison robots use the same sensor components to receive IR signals from remote controls and detect obstacles, and the sensor components cannot do both simultaneously. In order to receive an IR message from a TV or DVD remote control, obstacle detection must be off. This can be controlled using the ‘turn obstacle detection beam’ block from the Sensing category.

Examples:

The main program contains a ‘turn obstacle detection beam’ block from the Sensing category. There is also a subroutine triggered by a ‘remote code received’ Event category block.

The main program contains multiple ‘turn obstacle detection beam’ blocks from the Sensing category. There is also a subroutine triggered by a ‘remote code received’ Event category block.

Light level will return a very high value which may cause Edison to loop for a long time.

Meaning:

The phototransistors in Edison robots measure detected visible light and return the value of this measurement to the robot. Light levels in most environments, such as in a room with lights on in the day time, will return very high values. When this value is used as the input in a ‘repeat’ loop it can cause the blocks in the ‘repeat’ loop to be repeated a large number of times.

Examples:

The main program contains a ‘repeat’ loop block with an oval-shaped ‘light level of’ block from the Sensing category as the value input in the loop.

The program contains a subroutine triggered by a ‘clap detected’ Event category block which contains a ‘repeat’ loop block with an oval-shaped ‘light level of’ block from the Sensing category as the value input in the loop.

Some sensor data is stored in Edison's memory. You may need to clear the sensor data for your program to work correctly.

Meaning:

All of Edison's sensors generate data when they detect specific events. Some of this sensor data is stored in Edison's memory. This stored data can sometimes be a problem, making the robot react to an old event because the robot still 'remembers' the old event.

For example, when Edison checks if a condition has been met, if there is stored data, the robot will think that the condition has been met, even if it has not been met. For this reason, it is good coding practice to clear the sensor data. In EdScratch, you can clear sensor data using the 'clear sensor data' block from the 'Sensing' category in the block pallet.

Clearing the sensing data is especially important when you use sensor events in conditionals (such as 'if' or 'until' blocks) which are nested inside loops. This prevents data from a previous loop from affecting the next loop. Add the 'clear sensor data' block with the sensor you are using selected from the drop-down in the 'clear sensor data' block into your program after checking for events in such instances.

It is also best to clear the data at the start of a program, just in case the robot has old data stored from a previous program. Do this by using a 'clear sensor data' block with the relevant sensor selected as the first block in your program.

You can learn more about using the 'clear sensor data' block in EdScratch in activity U4-1.4 *Let's explore stacking and nesting if statements* from Unit 4 of the EdScratch lesson activities.

Examples:

The program contains an 'if' block from the Control category with a diamond-shaped 'clap detected' block from the Sensing category set as the conditional input.

The program contains an 'if' block from the Control category with a diamond-shaped 'obstacle detected' block from the Sensing category set as the conditional input.

The 'drive strain detected' block will only detect strain while the motors are running.

Meaning:

The diamond-shaped ‘drive strain detected’ block in the Sensing category can be used to detect if Edison’s motors stop moving while Edison is carrying out drive commands. If the robot’s motors are not moving, the robot cannot detect strain.

Examples:

The program contains an ‘if’ block from the Control category with a diamond-shaped ‘drive strain detected’ block from the Sensing category set as the conditional input.

The program contains a subroutine triggered by a ‘clap detected’ Event block with an ‘if’ block from the Control category with a diamond-shaped ‘drive strain detected’ block from the Sensing category set as the conditional input.

The 'drive strained' event can only trigger while the motors are running.

Meaning:

The ‘drive strained’ block from the Event category can be used to trigger a subroutine if Edison’s motors stop moving while Edison is carrying out drive commands. If the robot’s motors are not moving, the robot cannot detect strain and the event will not trigger.

Examples:

The program contains a subroutine triggered by a ‘drive strained’ Event block with drive blocks in the main program.

The program contains a subroutine triggered by a ‘drive strained’ Event block with no drive blocks in the main program.

The 'set both motors' block will only turn the motors on. Make sure there are additional blocks used in the program to control the motor's duration.

Meaning:

The ‘set both motors’ block from the Drive category turns the Edison robot’s motors on, but does not provide a distance input. As such, the robot will turn the motors on and then move on to the next block in the program. If there is no block after a ‘set both motors’ block, the program will end without Edison moving. If there are blocks

after the ‘set both motors’ block, Edison will execute these blocks with the motors moving.

Examples:

The main program contains a ‘set both motors’ block from the Drive category.

The program contains a subroutine triggered by a ‘clap detected’ Event block with a ‘set both motors’ block from the Drive category.

The 'set left motor' block will only turn the motors on. Make sure there are additional blocks used in the program to control the motor's duration.

Meaning:

The ‘set left motor' block from the Drive category turns the Edison robot’s left motor on, but does not provide a distance input. As such, the robot will turn the motor on and then move on to the next block in the program. If there is no block after a ‘set left motor' block, the program will end without Edison moving. If there are blocks after the ‘set left motor' block, Edison will execute these blocks with the motor moving.

Examples:

The main program contains a ‘set left motor' block from the Drive category.

The program contains a subroutine triggered by a ‘clap detected’ Event block with a ‘set left motor' block from the Drive category.

The 'set right motor' block will only turn the motors on. Make sure there are additional blocks used in the program to control the motor's duration.

Meaning:

The ‘set right motor' block from the Drive category turns the Edison robot’s right motor on, but does not provide a distance input. As such, the robot will turn the motor on and then move on to the next block in the program. If there is no block after a ‘set right motor' block, the program will end without Edison moving. If there are blocks after the ‘set right motor' block, Edison will execute these blocks with the motor moving.

Examples:

The main program contains a ‘set right motor' block from the Drive category.

The program contains a subroutine triggered by a ‘clap detected’ Event block with a ‘set right motor' block from the Drive category.