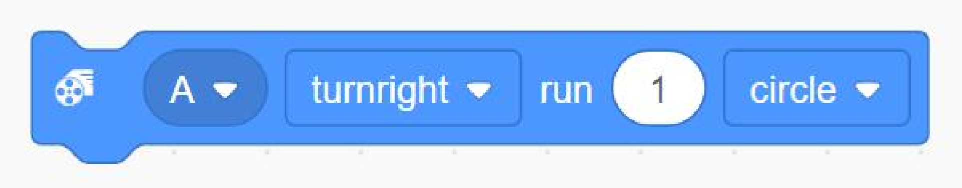
Programming Module Description

Motor Blocks

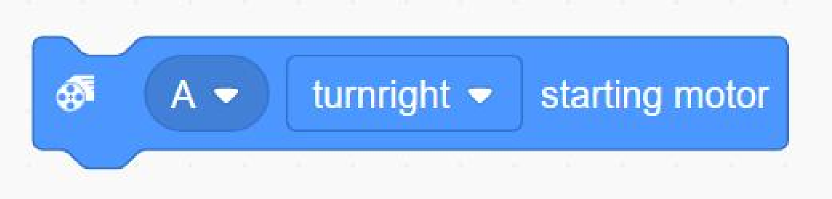
Motor Blocks not only has the function of driving motor operation but also get information from the motors. The Motor Blocks is the most common and general category that widely used in various motor control scenarios.

Run Motor



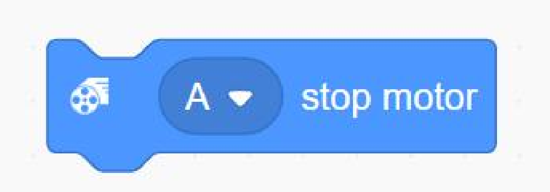
This block will run one or more motors clockwise or counterclockwise for a specified number of rotations, seconds, or degrees.

Start Motor



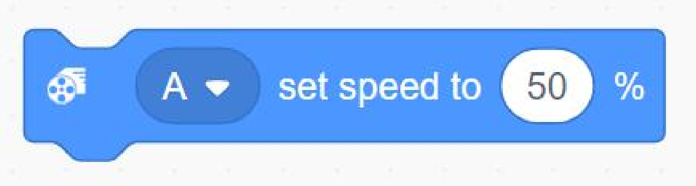
This block sets one or more motors to a specified position. The motor can be set to run clockwise or counterclockwise.

Stop Motor


A stop motor

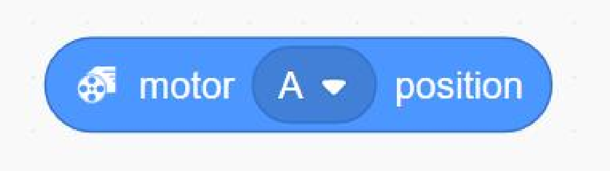
This block stops one or more motors from running. The motor will brake so that it quickly comes to a complete stop. It's important to note that after the motor stops, it won't maintain its current position; it might experience positional changes due to external factors or internal inertia.

Set Motor Speed


A set speed to 50 %

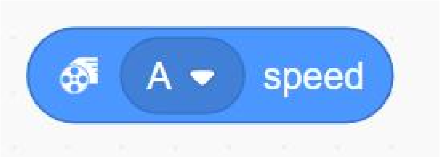
This block sets the speed of one or more motors. The speed range is -100 to 100. Negative values will reverse the direction of the motor. If the speed hasn't been specified, the default value is 50%.

Motor Position


motor A position

This block reports the current position of a motor. Through this module, users can obtain real-time precise positional information of the motor.

Motor Speed

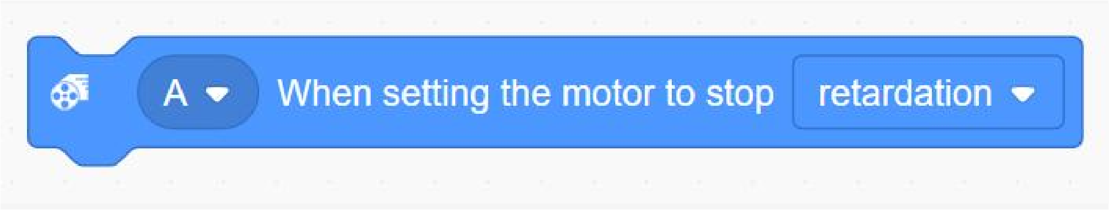


This block reports the current speed of a motor. The value given is the motor's actual speed, not the speed set by the Set Speed Block.

Movement Blocks

Movement Blocks enable users to run two motors in a synchronized motion. They're primarily used to move Driving Bases around. Only motors of the same type (e.g., two Medium Motors) can be synchronized.

Stop Motor with Inertia Glide



When using **Motor Blocks** or **Stop Motor** blocks, users can choose from three stopping modes:

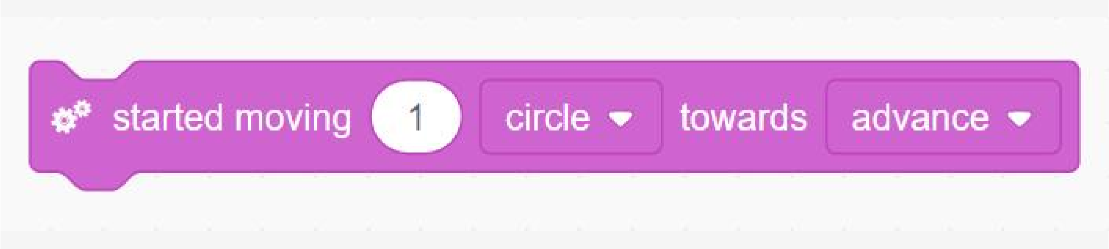
Brake Mode: Applies active braking then friction to ensure a full stop.

Position-Hold Mode: Uses braking and actively returns to the target stop position if pushed away.

Inertia Glide Mode: Cuts power immediately, letting the motor coast to a stop naturally.

Movement Blocks

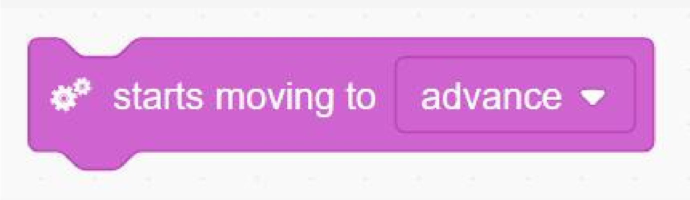
Move for a Specific Amount



started moving 1 circle towards advance

Controls the driving base to move forward, backward, or rotate clockwise/counterclockwise for a set time (seconds), angle (degrees), or number of rotations.

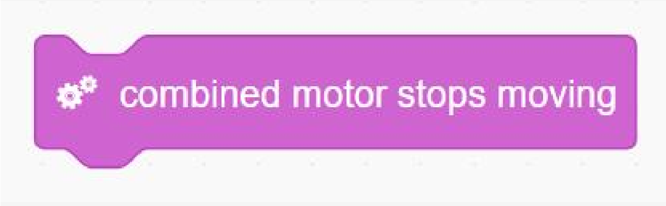
Start Moving



starts moving to advance

Continuously starts driving base movement in the selected direction.

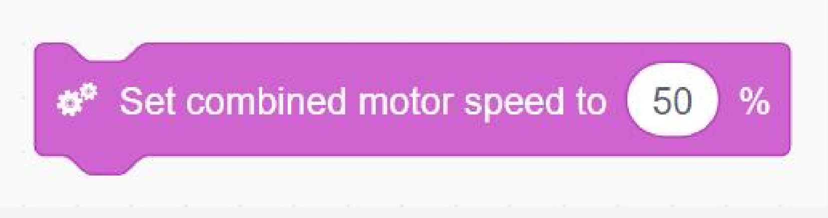
Stop Movement



combined motor stops moving

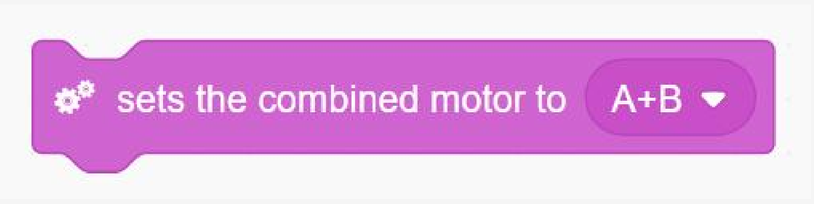
Immediately stops the driving base by turning off its motors.

Set Movement Speed


Set combined motor speed to 50 %

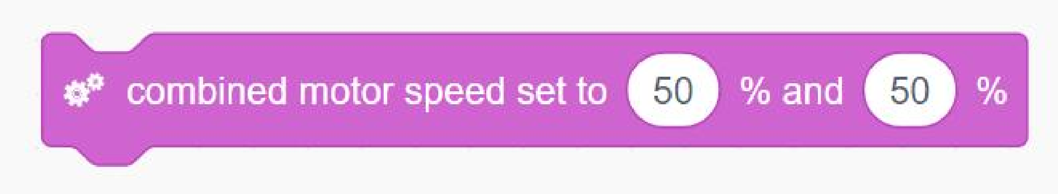
Sets speed from -100 to 100. Positive values move forward (or turn one way), negative values move backward (or turn the other way). Default is 50%.

Set Running Motors

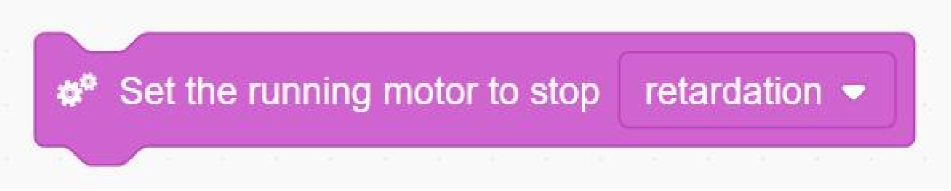

sets the combined motor to A+B

Assigns which two ports the driving motors are connected to.

Set Individual Motor Speeds


combined motor speed set to 50 % and 50 %

Independently sets the speed for the left and right driving motors.


Set the running motor to stop retardation

This module can set the stopping prevention of the motor of the driving base. Users can choose three different stopping mechanisms for the motor. The motor can stop working in the following three modes:

Braking mode: As the default stopping method, the motor will first start the dynamic brake after receiving the stop command, and then apply friction to ensure that the motor stops completely.

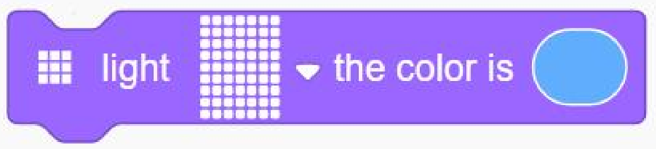
Position holding mode: In this mode, the motor also uses dynamic braking to stop. But what is special is that if the motor deviates from the set stop position due to external force, it will actively adjust and return to the original position.

Inertial glide mode: When this mode is selected, the motor will directly cut off the power when stopping, allowing the motor to naturally slow down and eventually stop by inertia.

Light Module Blocks

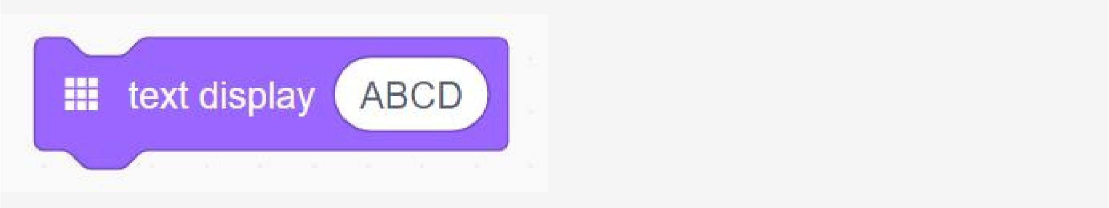
Controls a 7x9 LED matrix—turning it on/off, setting patterns, text, brightness, and color.

Turn On 7x9 Matrix Light



Displays user-defined patterns and colors on the matrix. Stays lit until new commands or program stop.

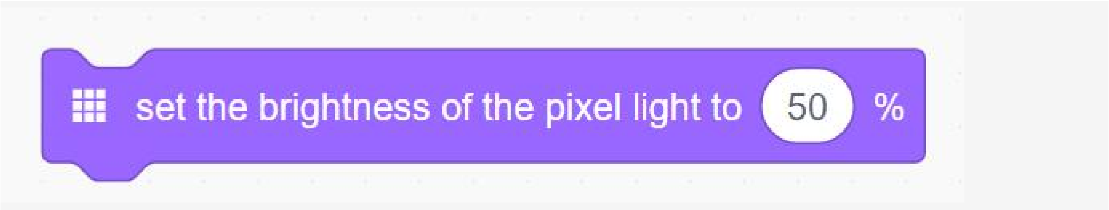
Scroll Text on 7x9 Matrix



text display ABCD

Displays text string letter by letter, scrolling across the display to fit the limited area.

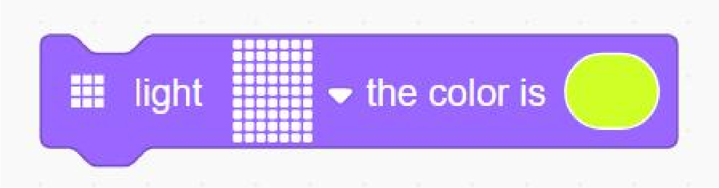
Set 7x9 Matrix Brightness



set the brightness of the pixel light to 50 %

Sets brightness for the next matrix command in the stack. Range: 0% (off) to 100% (max).

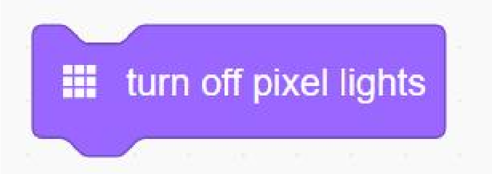
Set Matrix Color



light ▾ the color is

This module is specially used to set the color of 7x9 color matrix lights, providing users with flexible and powerful color control functions.

Clear Content

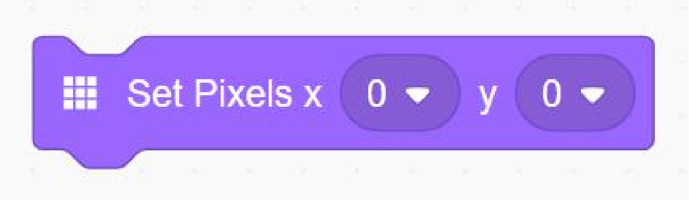


turn off pixel lights

This module has the function of clearing the display content of the 7×9 matrix light.

Users can use it to quickly clear all display information on the matrix light.

Set Pixels by Coordinates



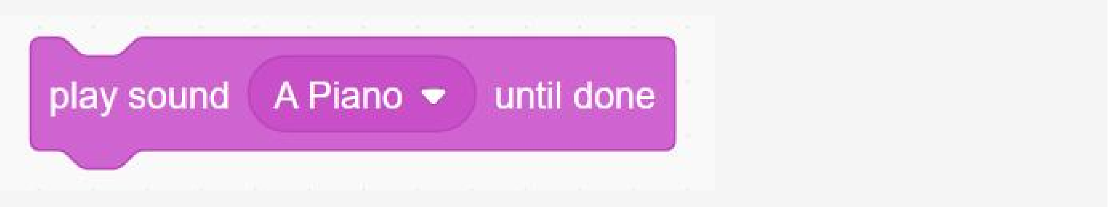
Set Pixels x 0 y 0

Controls individual or multiple pixels by specifying their X-Y coordinates to light them up precisely.

Sound Blocks

Play sounds on the device.

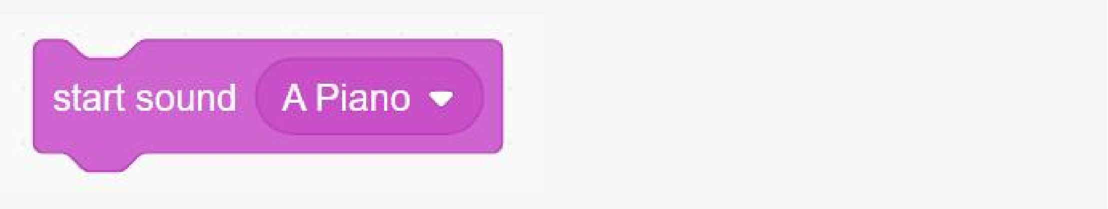
Play Sound Until Done



play sound A Piano until done

Plays the selected sound and pauses the program stack until playback finishes, ensuring complete uninterrupted playback.

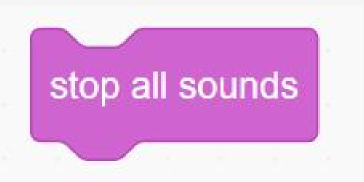
Start Playing Sound



start sound A Piano

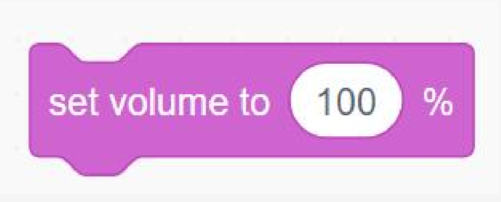
Starts playing sound immediately while the program stack continues running in parallel.

Stop All Sounds



Immediately stops all current sounds.

Set Volume



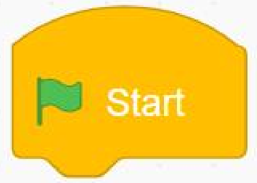
This module has the function of setting the volume, allowing users to adjust the volume of the sound as needed. The volume range is usually from 0% to 100%, where 0% means mute, that is, the sound is completely turned off; and 100% means the volume is at the maximum, and the sound is played at the maximum loudness.

Event Blocks

The event module is a unique component composed of hat modules. It occupies an indispensable and fixed position in the program stack - it always exists as the first module. All other modules must be spliced one by one under the hat module in a logical order. This structure ensures the clarity of the program and the order of execution.

The hat module plays a vital role in starting the entire program stack. Its core function is to serve as a trigger mechanism. When a specific event occurs, the hat module will be activated and start the execution of subsequent modules. This event-driven approach enables the program to flexibly respond to external or internal changes and achieve the expected functions.

When Program Starts

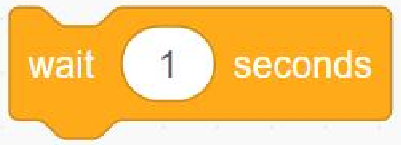


Once the program is started, this core module will automatically perform a series of operations. The user can choose to trigger the entire program through the Start button on the interface. In addition, if the system is not currently in Running Mode, the user can also start the program through physical operation.

Control Blocks

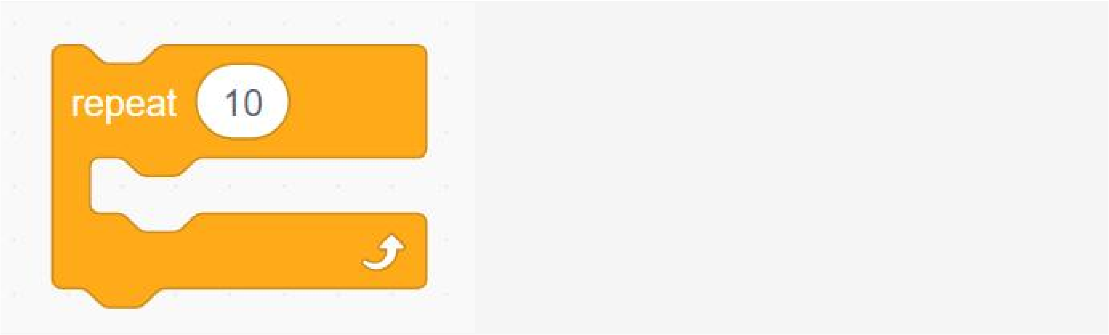
Control module category covers flow-control structures such as wait, loops, and conditionals.

Wait for Seconds



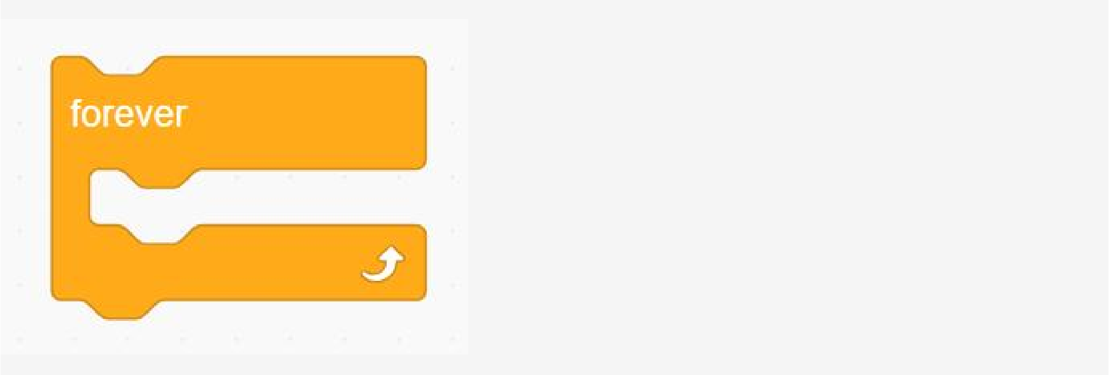
Pauses program execution for a specified number of seconds.

Repeat Loop



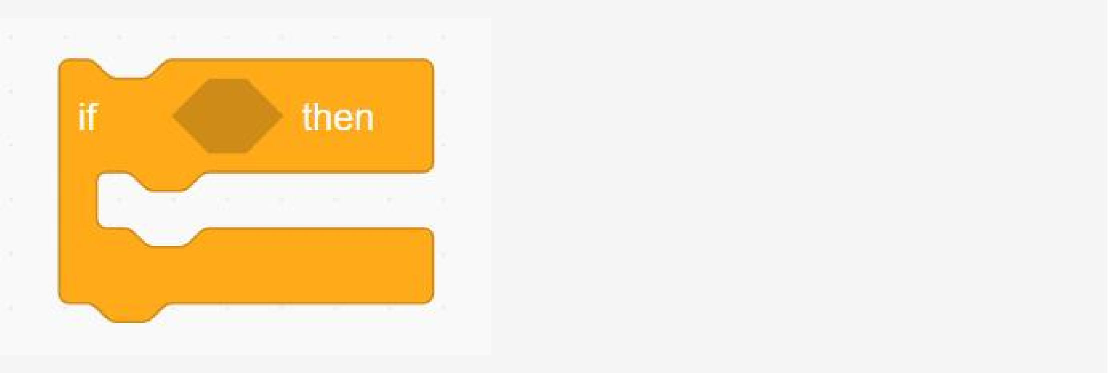
This module has the function of loop execution, which allows all the modules it contains to be executed repeatedly according to the specified number of times. After reaching the set number of loops, the program will jump out of the loop and continue to execute the subsequent modules or operations. This loop mechanism is very practical in program design, especially when a certain section of code or a series of operations needs to be repeatedly executed.

Forever Loop



This module has an infinite loop function, which will continuously execute all the modules it contains, forming a permanent loop process. This loop will continue until the user takes active measures to stop it.

If Then



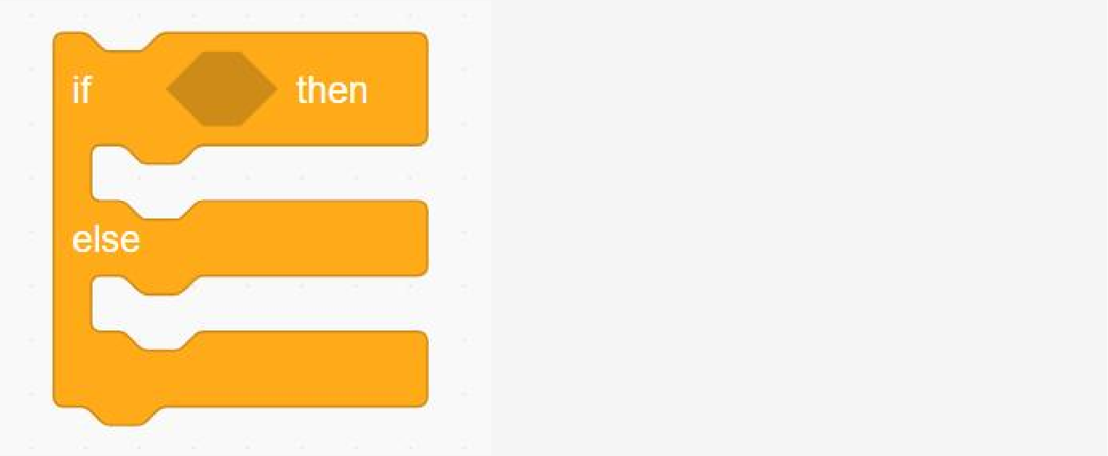
This module is responsible for checking whether a specific Boolean condition is true. Depending on the result of the check, the module behaves differently:

If the Boolean condition is true, this module will execute all the submodules contained in it in order. These submodules will run in the order in which they are arranged in the module to complete their respective tasks.

If the Boolean condition is false , all the submodules contained in this module will be skipped and no action will be performed.

-

If Then Else

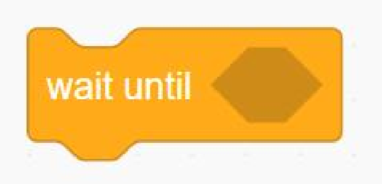


This module has the function of conditional judgment. It will decide which module in which block to execute based on the specified Boolean condition.

Specifically, when the condition is true, this module will execute all modules in its first block and run them in order. After completing the execution of the modules in the first block, the program will continue to run the subsequent program stack, that is, continue to execute the modules after this module.

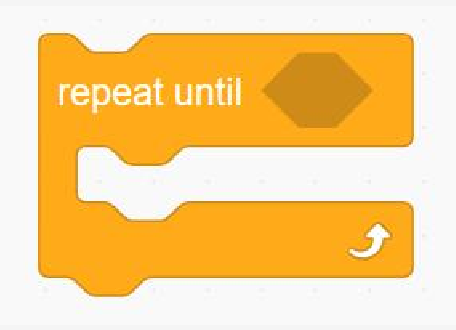
On the contrary, if the condition is false, this module will skip the modules in the first block and execute all modules in its second block instead. If the condition is not met, the program will execute a different code path from the first block.

Wait Until



Pauses execution until a specified boolean condition becomes true.

Repeat Until Loop

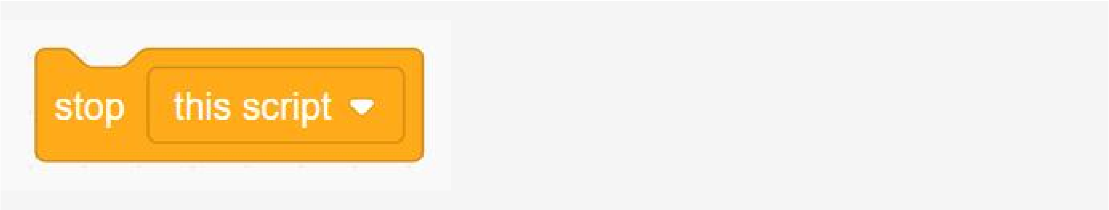


The function of this module is to create a loop structure that will continue to execute all the modules contained in it until the specified Boolean condition becomes yes.

During the loop, these modules will run repeatedly until the termination condition is met.

Once the specified Boolean condition becomes yes, the loop will terminate and the program will jump out of this module and continue to execute the modules spliced below this module.

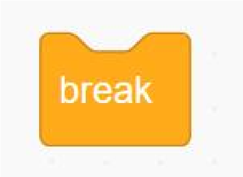
Stop



Stop all programs: When this function is selected, the module will immediately interrupt all currently executing program stacks to ensure that they stop running.

Stop this: If this option is selected, the module will only stop the current program stack, without affecting other program stacks running in parallel.

Break Out of Loop

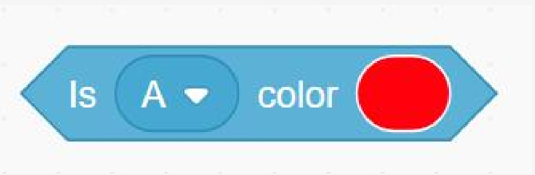


Exits the current loop and continues with the program below.

Sensor Blocks

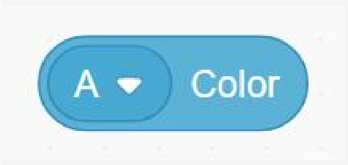
Read data from sensors like color, distance, force, and gyroscope sensors.

Is Color Equal To?



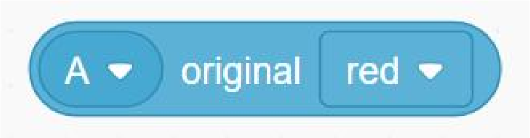
When the color sensor detects the specified color, the module will return a signal according to its pre-set logic or programming instructions. This signal may be an electronic signal, digital code or text message.

Color



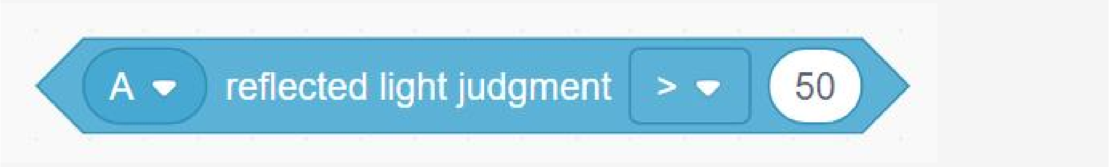
If a color sensor module is able to return the current value of the detected color, it is usually equipped to read the sensor data and convert it into a readable format.

Recognize Original Colors



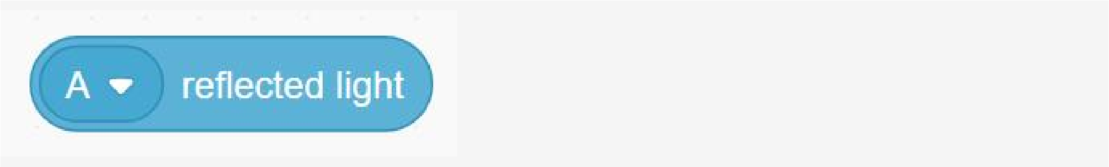
Reads and recognizes RGB values.

Is Reflected Light Condition Met?


A ▾ reflected light judgment > ▾ 50

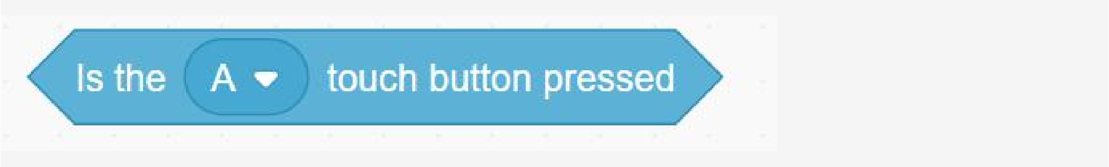
Returns true if reflected light is greater than, equal to, or less than a set percentage.

Reflected Light


A ▾ reflected light

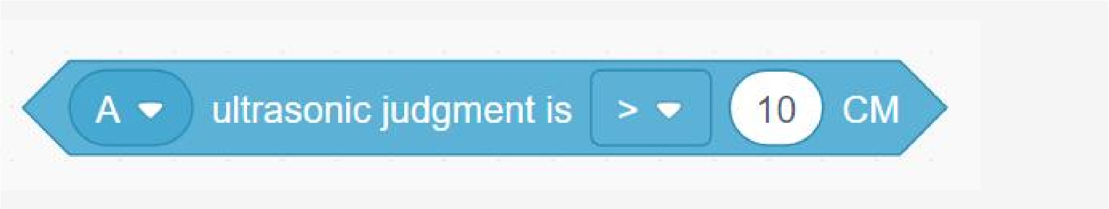
The color sensor module is able to report the current value of the light reflected back to the color sensor.

Is Pressed?


Is the A ▾ touch button pressed

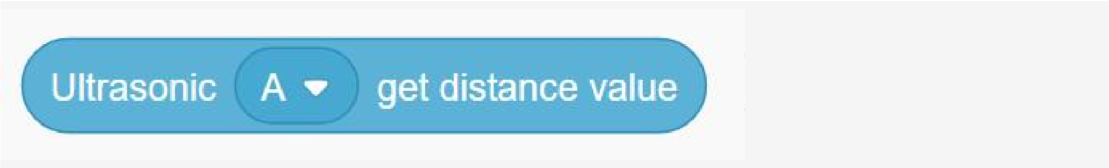
Returns true if a force sensor is pressed or released.

Is Distance Condition Met?


A ▾ ultrasonic judgment is > ▾ 10 CM

Returns true if the detected distance meets a relational condition (>, <, =).

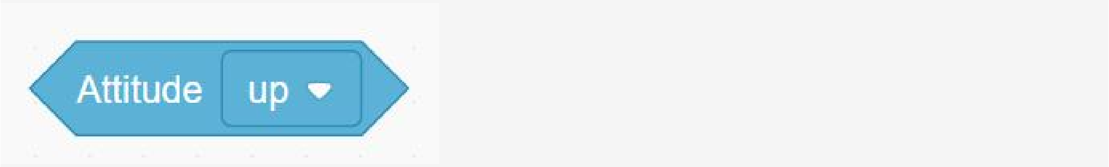
Distance



Ultrasonic A ▾ get distance value

Reports the current detected distance in real time.

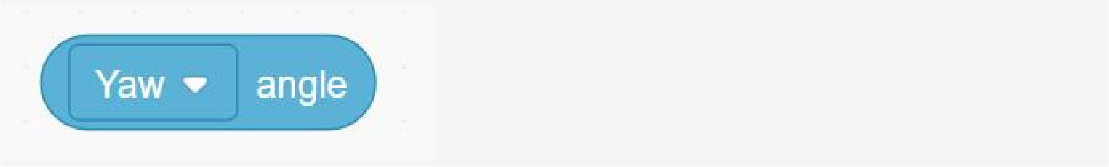
Is Tilted?



Attitude up ▾

When the main controller starts to tilt in the specified direction from a horizontal state, the module will return a logical true.

Hub Pitch, Roll, Yaw



Yaw ▾ angle

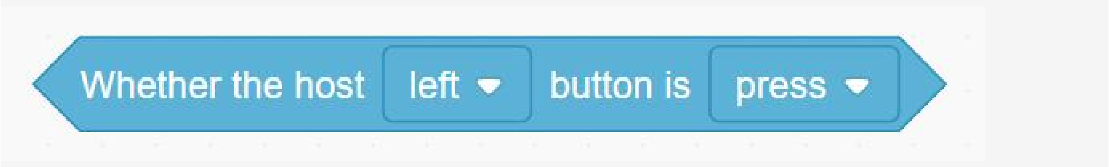
This module is responsible for providing the main controller's current pitch, roll, and yaw data. These terms - pitch, roll, and yaw - were originally used to describe the motion of an aircraft in the air, but they are also applicable to any object that rotates in three-dimensional space. In the case of an aircraft, these angles can accurately reflect the changes in its flight attitude.

The pitch angle describes the angle at which the aircraft's nose is tilted up or down relative to the horizontal line.

The roll angle refers to the angle at which the wing is tilted up or down relative to the horizontal line.

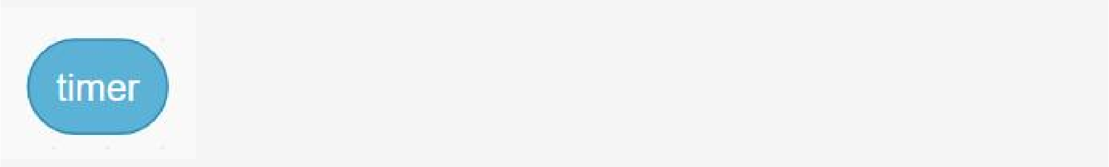
The yaw angle is used to indicate the direction of the aircraft relative to the ground, that is, the angle between the direction pointed by the aircraft's nose and a fixed direction on the ground, which determines the aircraft's heading in the horizontal plane.

Is Main Controller Button Pressed?



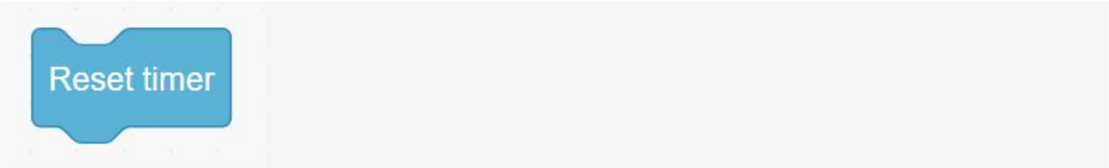
Returns true if either the Left or Right button is pressed or released.

Timer



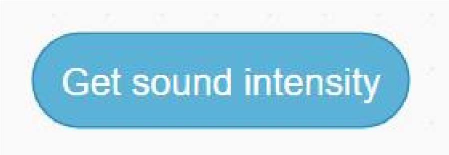
After the program starts, this module will start a timer to record the elapsed time in seconds. Every time the program is restarted, the timer will be reset accordingly and a new round of timing will begin.

Reset Timer



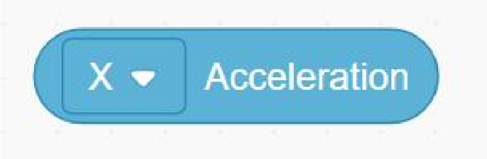
This module can be used to reset the timer.

Get Sound Level



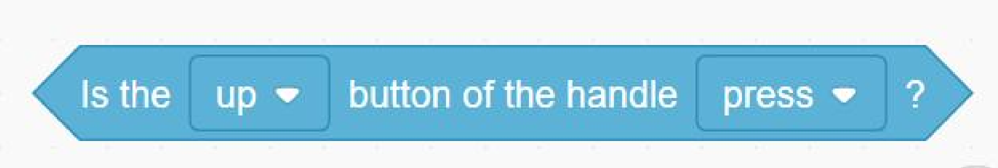
This module can capture and obtain the received sound signal and convert it into corresponding numerical data.

XYZ Acceleration



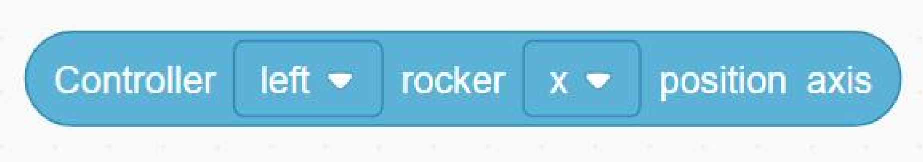
This module has the function of obtaining acceleration data on the three-dimensional coordinate axes XYZ, and can detect and output the acceleration values on the X-axis, Y-axis and Z-axis in real time.

Controller Button Command



This module can detect the state of the joystick button. When the button is pressed or released, it can judge and return the corresponding logical value. If the state of the button is detected to have changed (whether it is pressed or released), the module will return the logic true.

Controller Joystick Command

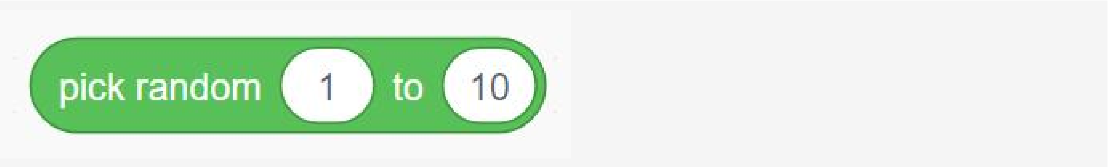


Captures joystick X and Y coordinates for precise position reading.

Operator Blocks

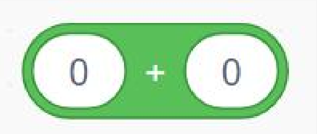
Performs all mathematical operations.

Pick Random Number



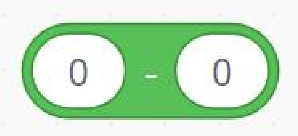
This module has the function of randomly selecting numbers in a specified range, including the two endpoints of the range. Both integers and decimals can be used as the minimum and maximum values. If the setting value contains a decimal, the module will be able to return the corresponding decimal as the random result.

Addition



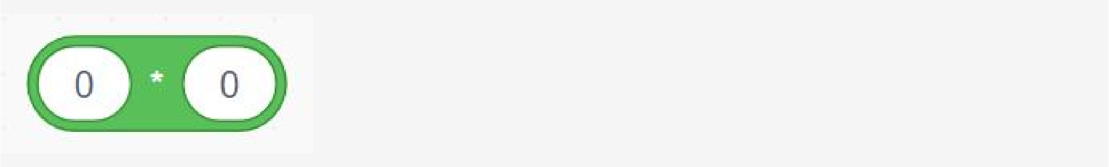
This module has an addition function that can add two values and return their sum as the result. Both integers and decimals can be used as input values for addition.

Subtraction



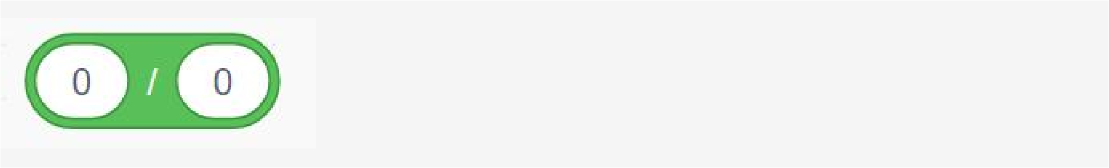
This block has a subtraction function that accepts two values as input, subtracts the first value from the second value, and returns the result. Whether the input is an integer or a decimal, the block can correctly perform the subtraction operation and return the corresponding difference.

Multiplication



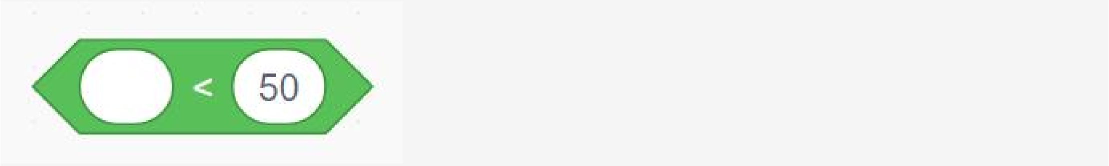
This module has a multiplication function that can receive two values as input, multiply them, and then return the result. Both integers and decimals can be used as input values for multiplication.

Division



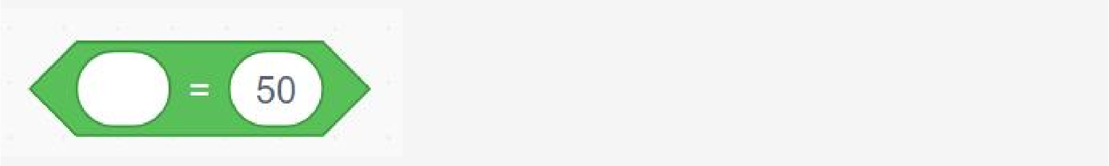
This module has a division function that can receive two values as input, divide the first value by the second value, and return the result. Whether the input is an integer or a decimal, the module can correctly perform the division operation and return the corresponding quotient.

Less Than



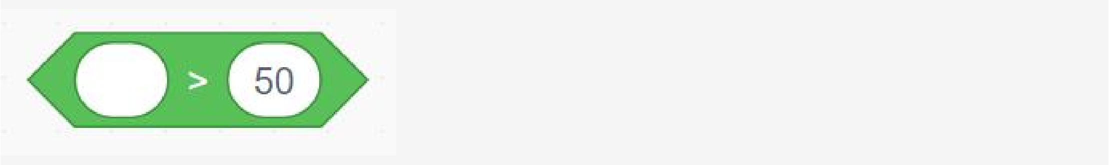
This module is responsible for comparing the size of two values. Specifically, it checks whether the first value is less than the second value. If the first value is indeed less than the second value, the module will return yes as confirmation; if the first value is not less than the second value, whether it is equal to or greater than the second value, the module will return no to indicate the comparison result.

Equal To



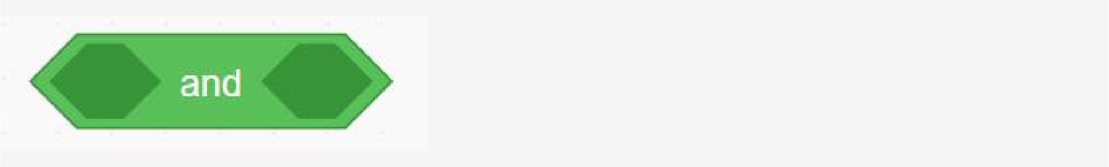
This module is responsible for comparing two values for equality. It takes two input values and checks if they are exactly the same. If the first value is equal to the second value, whether they are integers or decimals, the module returns yes, indicating that the two are equal. If they are not equal, the module returns no, indicating that the two are not equal.

Greater Than



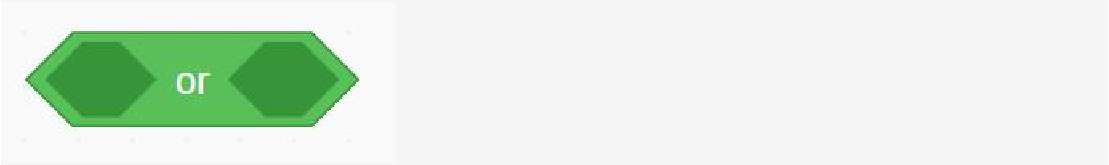
The function of this module is to compare the size of two values. Specifically, it receives two input values and checks whether the first value is greater than the second value. If the first value is indeed greater than the second value, the module will return yes, indicating that the first value is larger. Conversely, if the first value is not greater than the second value, the module will return no, indicating that the first value is not the larger one.

And



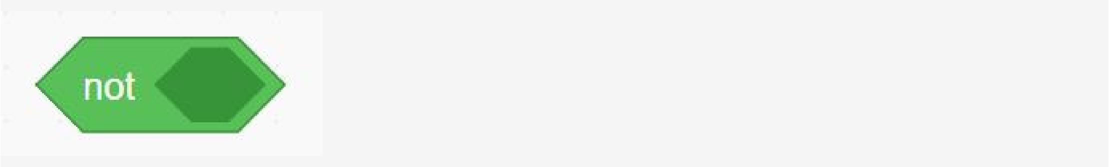
This module functions as a logical operator, connecting two Boolean modules using the AND condition. Specifically, it receives two input signals from the Boolean modules, and only when both input signals are true will the module return true as output. If any input signal is false, the output is false.

Or



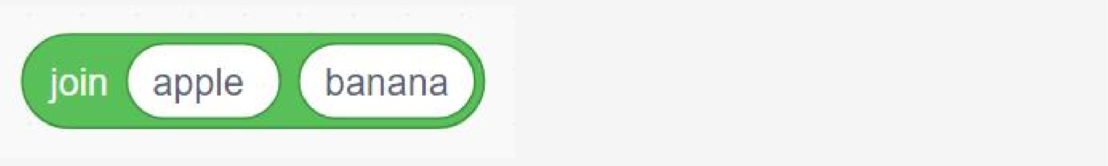
This module functions as a logical operator to connect two Boolean modules using the OR condition. It receives two input signals from the Boolean modules and checks if at least one of the two signals is True. As long as one of the input signals is True, the module returns True as output. The output will be False only when both input signals are False.

Not



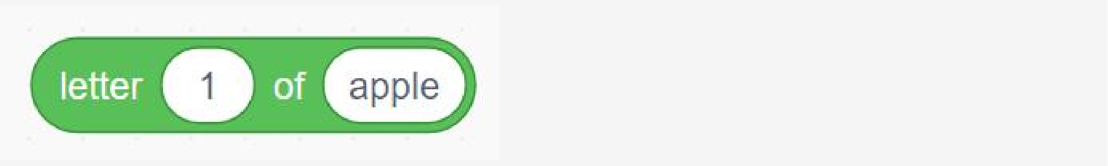
The function of this module is to reverse or negate the Boolean value of its internal condition. Specifically, it receives a Boolean value as input and returns the opposite of that value. If the input is true, the module reverses it to false; if the input is false, the module reverses it to true.

Join Strings



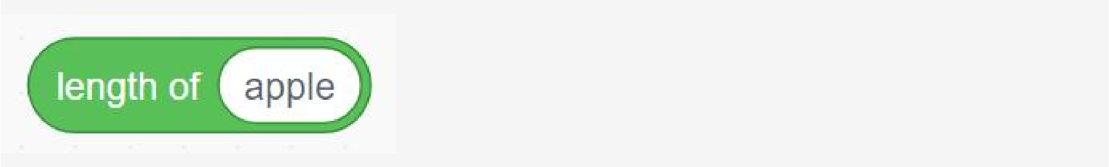
The function of this module is to merge or concatenate two text values and combine them into a new text string. When the user enters two text values, such as Hello and World, into the module, the module will splice them together, remove any spaces or separators in the middle, and return a new string "Hello World".

Letter of String



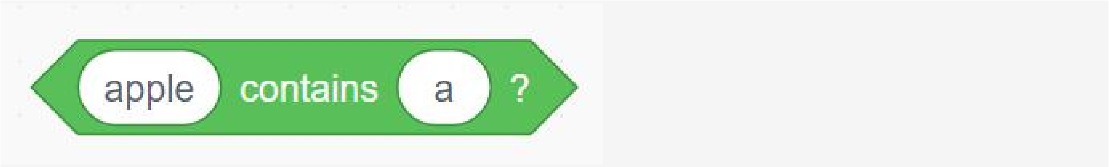
The function of this module is to extract the character at a specified position in a given string. The user enters a string and specifies a position (usually an index starting at 0 or 1), and the module returns the character at that position. For example, if the input string is EST and the specified position is 1, the module will return the first character of the string E.

String Length



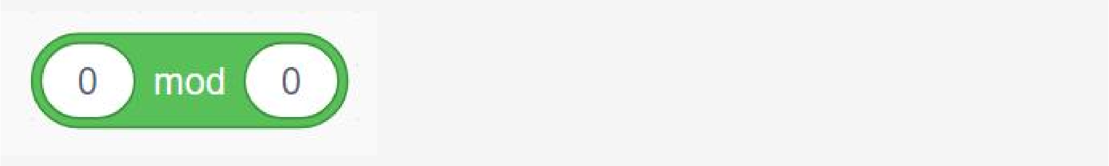
The function of this module is to count and return the number of characters in a given string. The user enters a string, and the module counts the number of characters contained in the string and returns the corresponding integer value. For example, if the input string is EST, the module will calculate that the string contains 3 characters and return the number 3.

String Contains



The function of this module is to check whether the specified character is contained in a specific string. The user needs to enter a string and a character, and the module will iterate over each character in the string to check whether there is a character that is the same as the specified character. If a matching character is found, the module will return yes, indicating that the character does exist in the string. If the string does not contain the specified character, the module will return no.

Modulo

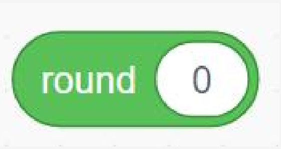


This module performs a remainder calculation, which returns the remainder after dividing the first value by the second value. For division of positive numbers, this is straightforward. However, when working with negative numbers, it is important to note that the remainder must always be positive, so the calculation is slightly different.

For example, when the first input is 10 and the second input is 3, the remainder of 10 divided by 3 is 1, which is a standard remainder calculation. However, when working with negative numbers, such as the first input is -10 and the second input is 3, the quotient of -10 divided by 3 is -3 and the remainder is -1 if we follow the normal division calculation. But in this module, in order to keep the remainder positive, we use a different calculation method. In fact, we add multiples of 3 to -10 until the result is a non-negative number that is less than 3. In this case, we need to add 4 times 3 (that is, 12), so that -10 becomes 2, and the remainder of 2 divided by 3 is 2. Therefore, this module returns 2 as the remainder of -10 divided by 3.

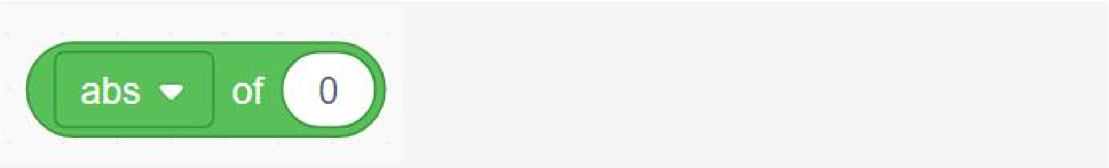
This is designed to ensure that the remainder result of negative number operations is always positive.

Round



The function of this module is to round a given number to the nearest integer, which follows the standard rounding rules. Specifically, when the decimal part of the given number is equal to or greater than 0.5, the module will round it up to the next higher integer; when the decimal part is less than 0.5, the module will round it down to the previous lower integer.

Math Function



The function of this module is to substitute a given number as an input value into a specified mathematical function and return the result calculated by the function. The user needs to provide the number to be substituted and the definition or description of the mathematical function.

Variable Blocks

The variable module category is indeed a broad and important category, covering all functional modules related to variables, lists (arrays), and user-defined "My Modules". Such modules play a vital role in programming and automation processes, allowing users to store, manage, and manipulate data, as well as create and reuse customized functions.

Variables

The variable module allows users to create and store data values, which can be numbers, text, Boolean values, etc. The variable module allows users to easily pass and share data between different parts of the program. For example, a variable module might allow users to set the value of a variable and then retrieve and use that value in a subsequent part of the program.

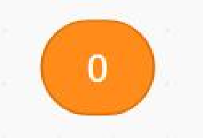
Lists (Arrays)

List (array) modules are powerful tools for working with ordered collections of data. These modules allow users to create lists and store multiple values in lists. Users can perform various operations on lists, such as adding, deleting, and modifying elements, as well as performing complex operations such as searching and sorting. List modules are particularly useful in scenarios where you are working with large amounts of data or need to organize the data for further analysis.

My Blocks

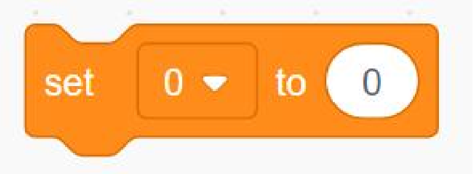
The "My Modules" category allows users to create and save custom modules. These modules can contain user-written code and logic to perform specific tasks or implement specific functions. By creating and using "My Modules", users can simplify complex programming tasks and improve the reusability and maintainability of code. In addition, this also provides users with a platform to learn and practice programming skills.

Variable



This block is specifically designed to report or display the current value of a variable. When a user creates or updates a variable in the system, it will automatically generate a “Report Variable Value” block associated with that variable. This block will display the given variable name so that users can clearly identify which variable it represents.

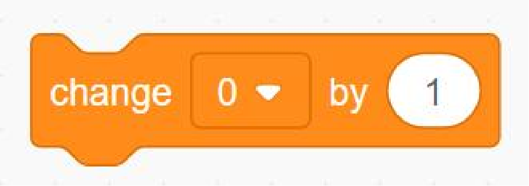
Set Variable To



This block sets or modifies the specified variable to a given value. Whether the variable originally holds a string or a number, this block can assign a new value to it.

When using this block, the user needs to specify two key pieces of information: the name of the variable to modify, and the new value. The block will accept these inputs and update the variable with the specified new value.

Change Variable By



This block changes the current value of the specified variable rather than simply assigning it a new value. Specifically, it adds a given amount to the variable’s existing value.

When the block is called with a variable and an increment amount, it finds the current value of that variable, adds the specified amount to it, and then assigns this new value back to the variable.

For example, if a variable currently holds the value 4, and you use the “Change Variable By 3” block, the variable’s value will become 7 (i.e., 4 + 3). This demonstrates that the block performs an addition operation to increase the variable’s value.