# **Comprehensive Reference of Scratch 3.0 Blocks: Structure, Functionality, and Inputs**

This report provides a detailed technical reference for Scratch 3.0 blocks, focusing on their internal structure, programmatic identifiers (opcodes), functional descriptions, input parameters, and illustrative examples. It is designed for developers, educators, and advanced users seeking a deeper understanding of Scratch's block-based programming paradigm.

## **I. Introduction to Scratch 3.0 Blocks**

Scratch 3.0, officially released on January 2, 2019, represents a significant evolution in block-based programming environments. This version is a complete redesign and reimplementation of Scratch, built using JavaScript.1 Its core strength lies in simplifying the coding process for beginners by employing a visual, drag-and-drop interface where programming logic is constructed by snapping together puzzle-shaped blocks.2 This visual approach significantly reduces common hurdles associated with text-based programming, such as memorizing commands and encountering syntax errors.2

### **Overview of Scratch 3.0's Block-Based Programming Paradigm**

Within the Scratch environment, projects are built around interactive elements known as "sprites" and "backdrops." Sprites are characters or objects that perform actions, while backdrops serve as the background of the stage.1 The behavior of these elements is dictated by "scripts," which are sequences of interconnected blocks. The intuitive drag-and-drop mechanism ensures that blocks can only be connected if their shapes align, thereby enforcing syntactical correctness and making the programming process more accessible.3 Each individual block is designed to perform a specific action or provide a particular value, collectively forming the fundamental building blocks of any Scratch program.4

### **Key Concepts: Sprites, Backdrops, and Scripts**

To fully appreciate the block system, it is essential to understand the core components it controls:

* **Sprites:** These are the interactive characters or elements within a Scratch project. Each sprite can possess its own unique code, multiple "costumes" (different images that facilitate animation), and distinct sounds.1
* **Backdrops:** Representing the background of the stage, backdrops can also have their own scripts and can be changed to create diverse settings or levels within a project.4 While all blocks are applicable to sprites, only a subset can be used with backdrops.6
* **Scripts:** A script is a linear series of instructions, composed of interconnected blocks, that defines what a sprite or backdrop does. A crucial aspect of Scratch programming is that every script must be initiated by an "Event" block.4

### **Understanding Block Categories and Shapes**

Scratch 3.0 employs a color-coded system to organize its blocks into nine primary categories: Motion, Looks, Sound, Events, Control, Sensing, Operators, Variables, and My Blocks.1 This color-coding enhances visual distinction and improves ease of access for users navigating the block palette.7 In addition to these core categories, Scratch 3.0 offers eleven extension categories, such as Music, Pen, Video Sensing, and Text to Speech, which introduce specialized functionalities or enable interaction with external hardware devices.1

Blocks are also distinguished by their unique shapes, which indicate their function and how they connect to other blocks 2:

* **Hat Blocks:** These blocks initiate scripts and are characterized by a rounded top and a bump at the bottom, allowing other blocks to be placed only below them. An example is the when green flag clicked block.
* **Stack Blocks:** These blocks perform the main commands within a script. They feature a notch at the top and a bump at the bottom, enabling connections both above and below them. The move 10 steps block is a common example. Stack blocks are the most prevalent block shape in Scratch.2
* **Boolean Blocks:** Hexagonal in shape, these blocks represent conditions that evaluate to either true or false, such as <mouse down?>.
* **Reporter Blocks:** Characterized by rounded edges, these blocks report values, which can be numbers or strings. They are designed to fit into input slots of other blocks, for instance, (x position).
* **C Blocks:** These blocks are shaped like the letter "C" and are used to loop or conditionally execute blocks placed within their opening. Examples include forever and if then.
* **Cap Blocks:** These blocks signify the end of a script. They have a notch at the top and a flat bottom, which prevents any further blocks from being placed below them, such as stop all.

### **The Significance of Opcodes as Internal Block Identifiers**

An "opcode" serves as the internal, programmatic identifier for a Scratch block, distinct from its user-facing display text.11 This identifier is particularly important in extension development, where the

opcode directly corresponds to the name of the method executed to perform the block's designated task.11 Opcodes are critical for understanding how Scratch projects are structured and executed, especially when examining or manipulating the underlying project JSON files.13 A fundamental principle in Scratch's development is its commitment to full backward compatibility, which dictates that block definitions and opcodes must remain consistent to prevent previously saved projects from failing to load or behaving unexpectedly.12

The design philosophy underpinning Scratch's block system places a strong emphasis on accessibility and minimizing user frustration. The visual, puzzle-shaped blocks, coupled with the explicit principle that "Scratch has no concept of a runtime error!" and that blocks should be "forgiving in its interpretation of inputs" 12, underscores a deliberate choice to lower the barrier to entry for beginners. This means the system is engineered to handle "invalid" inputs gracefully—for example, converting a number to a string if expected, or simply performing no action, rather than causing a program crash.12 This design choice, while highly beneficial for novice users, may obscure the underlying data type conversions or implicit behaviors for more advanced users or those transitioning to text-based programming languages with stricter type systems. It suggests that the internal Scratch Virtual Machine (VM) is highly robust in its ability to process diverse inputs without abrupt failures.

The role of opcodes extends beyond mere internal naming; they form the stable, machine-readable contract for block functionality. The consistent use of opcodes, as highlighted by their presence in project JSON files 13 and their critical function in extension development 11, directly enables backward compatibility and the portability of projects across different Scratch versions. This stability allows for the reliable interaction of external tools or modifications, such as hidden blocks or custom extensions, with Scratch projects, provided they adhere to these established opcode contracts. This architectural decision makes the entire system highly extensible without compromising existing user-created content.

**Table 1: Scratch 3.0 Block Categories Overview**

| Category Name | Associated Color | General Purpose | Number of Blocks (Approx.) |
| --- | --- | --- | --- |
| Motion | Blue | Control sprite movement and orientation. | 17 2 |
| Looks | Purple | Control sprite/stage appearance (costumes, effects, visibility). | 23 2 |
| Sound | Pink/Magenta | Control audio playback and sound properties. | 16 2 |
| Events | Light Yellow | Trigger scripts based on user or system events. | 9 16 |
| Control | Amber/Tangerine | Manage script flow (loops, conditionals, cloning, waiting). | 11 2 |
| Sensing | Teal/Light Blue | Detect interactions with environment, users, or other sprites. | 21 2 |
| Operators | Light Green | Perform mathematical, comparison, and string operations. | 18 2 |
| Variables | Orange | Create, store, and manipulate single values (variables). | 5 2 |
| Lists | Orange | Create, store, and manipulate sequences of values (lists). | 11 2 |
| My Blocks | Red/Light Red | Create custom, reusable functions (procedures). | (User-defined) 4 |

## **II. The Anatomy of a Scratch 3.0 Block (JSON Structure)**

Scratch projects, along with their constituent sprites and scripts, are systematically stored using JavaScript Object Notation (JSON). This data format is typically found within project.json or sprite.json files.14 Although a default

project.json might show an empty blocks entry 14, the underlying structure for defining blocks, particularly for extensions, offers critical insight into their programmatic representation.11

### **Core Properties: opcode, blockType, text, and arguments**

When a block is defined, especially in the context of creating extensions, it is represented as a JSON object with several key fields 11:

* opcode: This field provides the unique internal identifier for the block's specific functionality.11 It acts as the direct reference to the method that is invoked to execute the block's designated task.11
* blockType: This property specifies the visual shape and general behavior of the block. Common types include "command" (corresponding to a Stack block), "reporter" (for a Reporter block), "Boolean" (for a Boolean block), and "hat" (for a Hat block).11
* text: This field contains the user-friendly display text of the block. It often incorporates placeholders for arguments, denoted by square brackets (e.g., "fetch data from [url]").11
* arguments: This is an object that meticulously defines each input parameter that the block is designed to accept. Each argument is represented by its own field within this object (e.g., "url").11

### **Detailed Breakdown of Argument Types**

The arguments object further specifies properties for each input, including its type and a defaultValue.11

* **Common type values for arguments include:** "string" for text input, "number" for numerical input, and "Boolean" for true/false conditions.11 Additional types identified from internal attributes include  
  angle, checkbox, colour, date, dropdown, iconmenu, and variable.19
* **defaultValue:** This property provides a value that is pre-filled into the input slot when the block is initially dragged into the scripting area.11
* **Menus:** For inputs that present a dropdown menu, the menu property can be specified. This property links to an items list (for static options) or a function (for dynamically generated options) that provides the available choices for the dropdown.12
* **Inline Images:** Blocks can also incorporate inline images, which are specified using a dataURI.12

The opcode is explicitly described as the method name responsible for the block's operation.11 This structural design indicates that Scratch blocks, despite their visual and intuitive interface, function as direct calls to underlying JavaScript functions within the Scratch VM.12 The

arguments field then precisely maps to the parameters passed to these functions. This architecture effectively transforms the visual blocks into a high-level, user-friendly Application Programming Interface (API) for the Scratch runtime. This modularity is fundamental for maintaining the system, enabling its extension through custom blocks, and optimizing performance, as exemplified by discussions concerning splice versus direct index assignment for list operations in the scratch3\_data.js file.24

A notable design principle is that blocks "should accept any value as input, even 'invalid' values" and exhibit a "forgiving in its interpretation of inputs" characteristic.12 For command blocks, this might translate to performing no action, while for reporter blocks, it could mean returning zero or an empty string.12 This robust input handling leads to a more resilient and less error-prone user experience, as users are not confronted with runtime errors for common mistakes like providing a number where a string is anticipated. For developers creating extensions or analyzing project files, this means that the actual runtime behavior for "invalid" inputs may differ from the strict type-checking found in other programming languages. The responsibility falls on the block's internal implementation to manage type coercion or provide graceful fallbacks, rather than on the user to supply perfectly typed inputs. This also suggests that the internal Scratch VM incorporates mechanisms for type handling or validation that are not exposed to the end-user.

**Table 2: Generic Scratch Block JSON Structure Example**

| JSON Key | Description | Example Value | Notes |
| --- | --- | --- | --- |
| opcode | Unique internal identifier for the block's functionality. | "myCustomBlock" | Name of the method called to execute the block's work. |
| blockType | Defines the visual shape and general behavior of the block. | "command" | Can be "command", "reporter", "Boolean", "hat". |
| text | User-friendly display text of the block, with argument placeholders. | "say for seconds" | Arguments are enclosed in square brackets. |
| arguments | An object defining each input parameter the block accepts. | {} | Each field within this object represents an argument. |
| arguments.ARG\_NAME.type | Data type expected for the argument. | "string" | Can be "string", "number", "Boolean", etc. |
| arguments.ARG\_NAME.defaultValue | Value pre-filled in the input slot. | "Hello" | Provides a default value for the argument. |
| arguments.ARG\_NAME.menu | (Optional) Specifies a dropdown menu for the argument. | "myMenu" | Links to a menu definition in the extension's menus section. |

## **III. Detailed Block Reference by Category**

This section provides a detailed breakdown of Scratch 3.0 blocks, organized by their respective categories. For each category, a narrative description of its purpose and characteristics is provided, followed by examples of key blocks, their functionality, expected inputs, and illustrative usage. While the research material provides display names for many blocks, explicit opcodes for all standard blocks are not consistently detailed. Where available, opcodes for specific examples or hidden blocks are included.

### **A. Motion Blocks**

Motion blocks, distinguished by their blue color, are central to controlling the movement and orientation of sprites on the stage.2 It is important to note that the Stage, being a static element, does not utilize Motion blocks.7 Scratch 3.0 incorporates 17 distinct Motion blocks.2 These blocks enable sprites to execute actions such as moving a specified number of steps, gliding smoothly to particular positions, turning by a certain degree, pointing in a specific direction, or bouncing off the edges of the stage.4 Their primary function involves manipulating the sprite's x and y coordinates and its directional orientation.7

Examples of Motion blocks and their typical usage include:

* move () steps: This block advances the sprite by the specified number of steps in its current direction. Input: a number (e.g., 10).
* turn right () degrees: Rotates the sprite clockwise by the specified angular value. Input: a number (e.g., 15).
* go to x: () y: (): Instantly repositions the sprite to the given X and Y coordinates. Inputs: a number for X, a number for Y (e.g., x:0 y:0).
* glide () secs to x: () y: (): Moves the sprite smoothly to the specified X and Y coordinates over a defined duration. Inputs: a number for seconds, a number for X, a number for Y (e.g., 1 secs to x:100 y:50).
* point in direction (): Sets the sprite's facing direction. Input: a number representing degrees (0=up, 90=right, 180=down, -90/270=left).25
* change x by (): Modifies the sprite's X position by the given value. Input: a number (e.g., 10).
* set x to (): Sets the sprite's X position to an absolute value. Input: a number (e.g., 0).
* if on edge, bounce: Reverses the sprite's direction if it encounters the edge of the stage. This block requires no inputs.

Beyond the visible blocks, the Scratch system contains "hidden" Motion opcodes, which are remnants from older Scratch versions or internal components.13 These include

motion\_scroll\_right, motion\_scroll\_up, motion\_xscroll, and motion\_yscroll, which are now obsolete and non-functional in Scratch 3.0 but are retained for backward compatibility. Similarly, motion\_align\_scene, once used to align the backdrop in Scratch 2.0 alpha, is also non-functional in the current version.

The heavy reliance of Motion blocks on x and y positions and direction 7 implicitly introduces fundamental mathematical and geometric concepts. Blocks such as

go to x: () y: (), change x by (), set y to (), and the reporter blocks x position and y position directly map to a Cartesian coordinate system. The explicit definition of degrees (0=up, 90=right, 180=down, 270=left) for point in direction further solidifies this connection to geometry.25 This approach means that users are not merely moving a character; they are engaging with concepts of coordinates, angles, and relative versus absolute positioning, thereby establishing a foundational understanding for more advanced computational thinking and even game development.

The presence of "hidden blocks" like motion\_scroll\_right and motion\_align\_scene 13 that "do not do anything" or "do not function properly" 13 but are maintained "to maintain compatibility with older versions of Scratch" 13 reveals a significant architectural decision. This commitment to backward compatibility leads to the retention of non-functional or obsolete opcodes within the codebase. This ensures that older projects can still load, even if some of their original functionalities are no longer supported or are handled through different mechanisms in the current version. For developers, this implies that the internal opcode list is more extensive and contains references to legacy code, which can add complexity when attempting to understand the current, active set of blocks. This also highlights the engineering challenge involved in evolving a platform while simultaneously preserving a vast library of user-created content.

### **B. Looks Blocks**

Looks blocks, identified by their purple color, are responsible for controlling the visual appearance of sprites and the stage.2 Scratch 3.0 features 23 Looks blocks, some of which have specific counterparts for the Stage.2 These blocks facilitate the management of costumes for animation, the application of graphic effects (such as color, fisheye, or pixelate), adjustments to size, control over visibility, and manipulation of layering.4 Additionally, this category includes blocks for displaying speech and thought bubbles, enabling dialogue and internal monologues within projects.26

Examples of Looks blocks and their typical usage include:

* say for () seconds: Displays a speech bubble containing specified text for a set duration. Inputs: a string for the text, and a number for seconds (e.g., say [Hello!] for (2) seconds).
* switch costume to ( v): Alters the sprite's appearance to a designated costume. Input: a dropdown menu for costume name or a number for costume number (e.g., switch costume to (costume1)).
* next costume: Advances the sprite to the subsequent costume in its sequence. This block requires no inputs.
* change size by (): Modifies the sprite's size by a specified percentage. Input: a number (e.g., 10).
* set size to ()%: Sets the sprite's size to an absolute percentage. Input: a number (e.g., 100).
* change [ v] effect by (): Applies a graphic effect (e.g., color, fisheye, ghost) by a specified increment. Inputs: a dropdown menu for the effect type, and a number for the value (e.g., change [color v] effect by (25)).
* show: Makes the sprite visible on the stage. This block requires no inputs.
* hide: Makes the sprite invisible on the stage. This block requires no inputs.
* switch backdrop to ( v) and wait: Changes the stage backdrop and pauses the script until the backdrop transition is complete. This block is exclusively for the Stage. Input: a dropdown menu for the backdrop name.26

Similar to Motion blocks, there are "hidden" Looks opcodes, such as looks\_hideallsprites, which was designed to hide all visible sprites but is now obsolete or non-functional in Scratch 3.0.13 Additionally,

looks\_setstretchto and looks\_changestretchby, which were used for horizontal stretching in Scratch 1.x, are also non-functional in the current version.

Looks blocks are not merely about static appearance; they are integral to creating dynamic animations and fostering visual narratives. The switch costume to, next costume, switch backdrop to, and next backdrop blocks 26 directly enable frame-by-frame animation and seamless scene transitions. The ability to incorporate

say and think text 26 adds dialogue and internal monologues, which are essential components for storytelling. This category highlights Scratch's capacity as a tool for creative expression beyond pure logical programming. It empowers users to easily implement visual storytelling techniques, making it highly engaging for educational applications in areas such as digital art, animation, and interactive narratives.

The observation that looks\_setstretchto and looks\_changestretchby are "non-functional" in Scratch 3.0 but persist for compatibility 13 points to a conscious decision by the Scratch Team to remove or re-prioritize certain visual effects that were present in earlier versions (Scratch 1.x). Furthermore, discussions regarding the spacing of "say blocks for each sprite" to appear "closer to one another" 21 indicate continuous refinement of the visual presentation. This suggests an ongoing process of feature evaluation and user interface enhancement in Scratch development. Features that may have been complex to maintain or had limited utility for the target audience (beginners) can be deprecated, even if their opcodes are preserved for backward compatibility. The visual design, including the arrangement of speech bubbles, is actively optimized for clarity and an improved user experience.

### **C. Sound Blocks**

Sound blocks, identifiable by their pink or magenta color, are dedicated to controlling audio functions within Scratch projects, allowing users to play sounds, integrate music, and manipulate various sound properties.2 Scratch 3.0 includes 16 Sound blocks.2 These blocks enable the playback of specific sound files, drum sounds, and musical notes, as well as the setting of instruments and control over volume and various sound effects.4 It is worth noting that note blocks, which were present in earlier versions, have been relocated to the Music Extension in Scratch 3.0.2

Examples of Sound blocks and their typical usage include:

* play sound () until done: Plays a specified sound and pauses the script's execution until the sound has completed. Input: a dropdown menu for the sound name (e.g., play sound [Meow v] until done).
* start sound (): Initiates playback of a specified sound without pausing the script, allowing other actions to proceed concurrently. Input: a dropdown menu for the sound name (e.g., start sound [Pop v]).
* stop all sounds: Halts all sounds currently playing across all sprites and the Stage. This block requires no inputs.29
* change [ v] effect by (): Modifies a sound effect, such as pitch or pan (left/right), by a specified amount. Inputs: a dropdown menu for the effect type, and a number for the value (e.g., change [pitch v] effect by (10)).
* set [ v] effect to (): Sets a sound effect to a specific absolute value. Inputs: a dropdown menu for the effect type, and a number for the value (e.g., set [volume v] effect to (100)).
* clear sound effects: Removes all previously applied sound effects. This block requires no inputs.29
* change volume by (): Adjusts the sprite's volume by a specified increment. Input: a number (e.g., -10).
* set volume to ()%: Sets the sprite's volume to a specific percentage. Input: a number (e.g., 50).
* (volume): A reporter block that provides the current volume level of the sprite.

The system also contains "hidden" Sound opcodes, such as music\_midiPlayDrumForBeats and music\_midiSetInstrument, which are obsolete blocks related to the Music Extension.13

Sound blocks offer two primary modes of playback: play sound () until done, which pauses the script, and start sound (), which allows concurrent execution.29 This distinction is vital for controlling the flow of a program in response to auditory events. Furthermore, the capability to apply effects like pitch and pan 29 enables creative sound design within projects. This demonstrates how Scratch integrates multimedia elements with precise script control. The choice between blocking and non-blocking sound playback introduces users to fundamental concepts of synchronous versus asynchronous operations in a visual manner. The sound effects also provide an introduction to basic audio engineering principles.

A notable observation from development discussions highlights a particular performance characteristic: using the "set volume to" block can introduce "imposed delays" and cause the game to "yield one frame," even when used within a "wait without screen refresh block".22 This behavior is attributed to the internal implementation, specifically the use of

Promise.resolve() in the scratch3\_sound.js file, which indicates an intentional yielding mechanism. This suggests that the internal implementation of certain sound blocks can introduce unexpected performance characteristics, such as temporarily pausing the main thread. This might be a consequence of the nature of audio processing or a deliberate choice to prevent the user interface from becoming unresponsive for extended periods. This reveals that even in a high-level, block-based language, underlying technical decisions, such as using Promises for yielding, can influence script performance. For advanced users or those developing complex projects, understanding these subtle performance considerations, for example, that setting volume is not always an instantaneous, non-blocking operation, becomes important for debugging and optimizing their code. This also suggests a trade-off between simplicity of use and absolute performance optimization within the Scratch VM.

### **D. Events Blocks**

Events blocks, distinguished by their light yellow color, are foundational elements in Scratch as they are responsible for triggering scripts in response to various occurrences.2 This category is the smallest in terms of the number of blocks.16 These blocks function as "hats" for scripts, initiating their execution when a specific event takes place, such as a mouse click, a key press, or the reception of a broadcast message.2 Without the hat blocks from this category, projects would be unable to begin automatically.16

Examples of Events blocks and their typical usage include:

* when green flag clicked: This Hat block starts the script when the green flag, which serves as the project's "start" button, is clicked. It requires no inputs.16
* when () key pressed: This Hat block initiates the script when a specified keyboard key is pressed. Input: a dropdown menu for the key name (e.g., space, up arrow, a).16
* when this sprite clicked: This Hat block starts the script when the sprite itself is clicked. It requires no inputs.16
* when backdrop switches to (): This Hat block triggers the script when the stage backdrop changes to a specified backdrop. Input: a dropdown menu for the backdrop name.16
* when () > (): This Hat block starts the script when a certain value (e.g., loudness from a microphone, or the timer) exceeds a defined threshold. Inputs: a dropdown menu for the value type (e.g., loudness, timer), and a number for the threshold.16
* when I receive (): This Hat block initiates the script upon the reception of a specific broadcast message. Input: a dropdown menu for the message name.16
* broadcast (): This Stack block sends a broadcast message throughout the Scratch program, activating any when I receive () blocks that are set to listen for that message. Input: a string or dropdown menu for the message name.2
* broadcast () and wait: This Stack block sends a broadcast message and pauses the current script until all other scripts activated by that broadcast have completed their execution. Input: a string or dropdown menu for the message name.2

The system also contains "hidden" Events opcodes, such as event\_whentouchingobject and event\_touchingobjectmenu, which are obsolete blocks related to sensing touch events and were likely removed due to redundancy with existing sensing blocks.13

The entire Events category 16 exemplifies the event-driven programming paradigm. Scripts are not executed in a strict linear fashion but are triggered by specific occurrences. While the

when green flag clicked block serves as the most common entry point, blocks like when key pressed, when this sprite clicked, and when I receive () demonstrate the system's responsiveness to user input and its capability for inter-sprite communication. This design teaches fundamental concepts of reactive programming and concurrency, particularly through the distinction between broadcast and broadcast and wait. It enables the creation of interactive and dynamic projects where actions are directly linked to user or system events, a core concept in modern software development.

The broadcast () and broadcast () and wait blocks 2 are specifically designed to allow one sprite to "send a message to another sprite".30 This mechanism facilitates indirect communication, enabling different parts of a project (individual sprites or the stage) to coordinate their actions without direct dependencies. The

broadcast mechanism enables complex interactions and synchronization among multiple independent scripts and sprites. The and wait variant provides a straightforward method to implement sequential coordination across sprites, ensuring that one action is completed before a subsequent action begins. This feature is crucial for developing more sophisticated games and animations that involve multiple interacting characters or distinct phases. It introduces the concept of message queues and asynchronous communication in a simplified, visual format, which is a key pattern in various software systems, including user interface programming and distributed computing.

### **E. Control Blocks**

Control blocks, characterized by their amber or tangerine color, are essential for managing the flow of scripts within a Scratch project. They enable functionalities such as looping, conditional execution, pausing, and cloning.2 Scratch 3.0 includes 11 Control blocks.2 These blocks provide the logical constructs for repetition (

repeat, forever, repeat until), decision-making (if then, if then else), temporal control (wait () seconds, wait until), and the dynamic management of sprite copies (create clone of, when I start as a clone, delete this clone).17

Examples of Control blocks and their typical usage include:

* wait () seconds: Pauses the script for a specified duration. Input: a number representing seconds, which can be a decimal value (e.g., wait (0.5) seconds).17
* repeat (): Executes the blocks contained within its loop a specified number of times. Input: a number (e.g., repeat (10)).17
* forever: Continuously loops the blocks inside until the project is manually stopped. This is a C block and requires no inputs.17
* if <> then: Executes the blocks contained within its conditional branch only if a Boolean condition evaluates to true. Input: a Boolean block (e.g., if <touching [mouse-pointer]?> then).17
* if <> then else: Executes one set of blocks if a condition is true, and an alternative set of blocks if the condition is false. Input: a Boolean block (e.g., if <score > 10> then else).17
* repeat until <>: Loops the blocks contained within its structure until a Boolean condition becomes true. Input: a Boolean block (e.g., repeat until <key [space v] pressed?>).17
* stop [ v]: Halts all scripts, only the current script, or other scripts within the same sprite. Input: a dropdown menu (e.g., all, this script, other scripts in sprite).17 This block's shape can dynamically change based on the selected option.31
* create clone of (): Generates a copy, or clone, of a specified sprite. Input: a dropdown menu for the sprite name, or myself.17
* when I start as a clone: This Hat block triggers scripts specifically for newly created clones. It requires no inputs.17
* delete this clone: This Cap block removes the clone that is executing it from the stage. It requires no inputs.17

The system also includes "hidden" Control opcodes, such as control\_for\_each, which iterates through a list, assigning each item to a variable (similar to a for loop).13 Another is

control\_while, which loops as long as a condition remains true (the inverse of repeat until). Additionally, control\_get\_counter, control\_incr\_counter, and control\_clear\_counter are blocks for managing a simple counter. control\_all\_at\_once is an obsolete C block that had no functional effect in Scratch 3.0 but was retained for backward compatibility.

The Control category directly implements core programming constructs found in text-based languages, including loops (repeat, forever, repeat until), conditionals (if then, if then else), and mechanisms for concurrency and process management (create clone of, delete this clone, when I start as a clone). The wait blocks introduce the concept of timing and delays in script execution. Scratch effectively abstracts complex programming logic into intuitive visual blocks, making concepts like iteration, branching, and parallelism accessible to beginners. This forms a robust conceptual foundation for users who may later transition to more advanced programming languages. The stop block's capability to stop "other scripts in sprite" 17 also introduces nuanced control over concurrent execution within a single sprite.

While cloning provides a powerful mechanism for dynamic object creation, enabling complex visual effects and game mechanics 32, discussions among advanced users indicate concerns about rapidly reaching "the clone limit" and suggest "list stamping" as a more performant alternative for "higher levels" of game development.32 This highlights that while cloning simplifies many tasks for beginners, it introduces performance considerations and resource limitations for more ambitious projects. This reveals a natural progression in Scratch programming: from straightforward visual solutions to more optimized, though potentially more complex, techniques as projects scale. This also implicitly teaches about resource management and optimization, even within a high-level programming environment.

### **F. Operators Blocks**

Operators blocks, identifiable by their light green color, are designed to perform mathematical calculations, comparisons, and string manipulations within Scratch projects.2 Scratch 3.0 includes 18 Operators blocks.2 This category encompasses standard arithmetic operations (

+, -, \*, /), comparison operators (<, =, >), logical operators (and, or, not), and functions for handling strings (join, letter of, length of, contains?).18 Additionally, it features a

pick random block for generating random numbers and a versatile () of () block that supports a wide range of advanced mathematical functions.18

Examples of Operators blocks and their typical usage include:

* () + (): Adds two numerical values. Inputs: a number, a number (e.g., (5) + (3)).
* () - (): Subtracts the second numerical value from the first. Inputs: a number, a number (e.g., (10) - (4)).
* () \* (): Multiplies two numerical values. Inputs: a number, a number (e.g., (6) \* (7)).
* () / (): Divides the first numerical value by the second. Inputs: a number, a number (e.g., (20) / (5)).
* pick random () to (): Generates a random integer within a specified inclusive range. Inputs: a number for the minimum, a number for the maximum (e.g., pick random (1) to (10)).33
* <() < ()>: A Boolean block that checks if the first value is less than the second. Inputs: any type, any type (e.g., <(score) < (100)>).2
* <() = ()>: A Boolean block that checks if two values are equal. Inputs: any type, any type (e.g., <(answer) = >).2
* <<> and <>>: A Boolean block that returns true if both provided Boolean conditions are true. Inputs: a Boolean, a Boolean (e.g., <<key [space v] pressed?> and <touching [color red]?> >).2
* join ()(): Concatenates two strings or values into a single string. Inputs: a string or number, a string or number (e.g., join [Hello ][World!]).27
* letter () of (): Reports the character at a specific numerical position within a string. Inputs: a number for the index, a string for the text (e.g., letter (1) of [apple]).18
* length of (): Reports the total number of characters in a given string. Input: a string for the text (e.g., length of [banana]).18
* () contains ()?: A Boolean block that checks if one string contains another string. Inputs: a string, a string (e.g., [apple] contains [a]?).2
* () mod (): Reports the remainder when the first number is divided by the second. Inputs: a number, a number (e.g., (10) mod (3)).
* round (): Rounds a numerical value to the nearest integer. Input: a number (e.g., round (3.7)).
* ([abs v] of ()): A highly versatile block that performs various mathematical functions such as absolute value (abs), square root (sqrt), trigonometric functions (sin, cos, tan), inverse trigonometric functions (asin, acos, atan), logarithms (ln, log), and exponential functions (e^, 10^). Inputs: a dropdown menu for the function type, and a number for the value.18

Operators blocks provide the essential tools for performing calculations, making comparisons, and manipulating text.18 These are not merely "math blocks" but form the core of any computational logic, enabling programs to respond dynamically to data, calculate scores, process user input, and manage game states. The

join block, for example, allows for the dynamic construction of strings.33 This category is fundamental for teaching core computer science concepts such as arithmetic operations, Boolean logic, and string processing. It empowers users to build programs that are not only animated but also intelligent and data-driven, capable of complex decision-making and dynamic content generation.

A notable observation from a GitHub issue highlights a RangeError: Invalid string length that can occur in scratch3\_operators.js when concatenating strings, potentially leading to "a variable containing a string almost 300 million characters long!".23 This points to a potential performance bottleneck and memory issue associated with extreme string manipulation. The solution to this issue involved addressing a

join operation. This reveals a tension between Scratch's ease of use and the practical limitations of the underlying JavaScript environment. While the system strives to be forgiving, there are still boundaries to resource consumption. For advanced users, this underscores the importance of understanding algorithmic complexity and memory management, even in a block-based environment, particularly when dealing with large datasets or repetitive string operations.

### **G. Data Blocks (Variables & Lists)**

Data blocks, color-coded orange, are designed to allow users to define, store, and manipulate data within their Scratch projects.2 This category encompasses blocks for creating, setting, changing, showing, and hiding individual variables, as well as managing collections of data through lists.4 Variables can store either numbers or strings.36 Lists, which are sequences of values 14, can be conceptualized as a type of variable capable of holding multiple items.38

Examples of Data blocks for Variables and their typical usage include:

* make a variable: This is a user interface action to create a new variable, not a block itself.
* set [my variable v] to (): Assigns a specific value to a variable. Inputs: a dropdown menu for the variable name, and any type for the value (e.g., set [score v] to (0)).27
* change [my variable v] by (): Increases or decreases a variable's numerical value by a specified amount. Inputs: a dropdown menu for the variable name, and a number for the value (e.g., change [score v] by (1)).6
* show variable [my variable v]: Makes a variable's monitor visible on the stage. Input: a dropdown menu for the variable name.6
* hide variable [my variable v]: Hides a variable's monitor from the stage. Input: a dropdown menu for the variable name.6
* (my variable): A reporter block that provides the current value stored in a variable. Input: a dropdown menu for the variable name.27

Examples of Data blocks for Lists and their typical usage include:

* make a list: This is a user interface action to create a new list, not a block itself.
* add () to [my list v]: Appends an item to the end of a list. Inputs: any type for the item, and a dropdown menu for the list name (e.g., add [apple] to [my list v]).27
* delete () of [my list v]: Removes an item from a list by its index or by selecting "all" items. Inputs: a number for the index or a dropdown menu (all, last, random), and a dropdown menu for the list name.27
* insert () at () of [my list v]: Inserts an item at a specific position within a list. Inputs: any type for the item, a number for the index, and a dropdown menu for the list name.40
* replace item () of [my list v] with (): Replaces an item at a specific position in a list with a new value. Inputs: a number for the index, a dropdown menu for the list name, and any type for the new item.27
* item () of [my list v]: Reports the item located at a specific position in a list. Inputs: a number for the index or a dropdown menu (last, random), and a dropdown menu for the list name.35
* length of [my list v]: A reporter block that provides the total number of items contained in a list. Input: a dropdown menu for the list name.27
* [my list v] contains ()?: A Boolean block that checks if a list includes a specific item. Inputs: a dropdown menu for the list name, and any type for the item.27
* show list [my list v]: Makes a list's monitor visible on the stage. Input: a dropdown menu for the list name.40
* hide list [my list v]: Hides a list's monitor from the stage. Input: a dropdown menu for the list name.40

The system also contains "hidden" Data opcodes, such as data\_listindexall and data\_listindexrandom, which were used as internal inputs to list blocks in early Scratch 3.0 versions and are not directly exposed to users.13

Variables and lists serve as the primary mechanisms for storing and manipulating data that changes during program execution.4 Variables allow for the storage of single values, such as scores or names, while lists enable the organization of structured collections of data, such as inventories or sequences of actions. The ability to

set, change, show, and hide these data elements provides comprehensive control over a project's state. This category introduces fundamental computer science concepts of data storage, retrieval, and manipulation. It teaches users how to create dynamic, responsive programs that can retain information, track progress, and adapt their behavior based on stored values, moving beyond purely sequential or event-driven logic.

A critical observation from a GitHub issue 24 revealed that the

replace item () of () with () block in scratch3\_data.js exhibited significantly slower performance in Scratch 3.0 compared to Scratch 2.0. This was identified as an O(N) runtime complexity issue, attributed to the use of JavaScript's splice() method. The proposed solution, utilizing direct index assignment (list.value[index - 1] = item), which is an O(1) operation, resulted in a "huge performance impact" for projects involving large lists. This highlights that the choice of underlying JavaScript array manipulation methods directly influences the performance of list operations in Scratch. An inefficient underlying implementation can lead to substantial slowdowns in user projects, particularly when dealing with larger datasets. This is a profound observation regarding the technical challenges inherent in building a high-level visual programming environment. Even though users interact with simple blocks, the efficiency of the underlying JavaScript runtime and its standard library functions directly affects the real-world performance of Scratch projects. It underscores that "block-based" does not imply "performance-agnostic" and that the Scratch Team actively optimizes the VM for common use cases. For advanced users, this knowledge can guide decisions on how to structure data and select list operations for optimal performance in large-scale projects.

### **H. My Blocks (Custom Blocks)**

My Blocks, typically color-coded red, light red, or pink, provide users with the capability to create their own custom blocks, also known as functions or procedures.4 This category initially appears blank in the block palette.4

* **Purpose and Characteristics:** My Blocks enable users to define reusable sequences of code, effectively encapsulating complex logic into a single, custom-named block.4 This feature promotes key programming principles such as modularity, abstraction, and code reusability.37 Custom blocks can be designed to accept inputs (arguments) and can function as commands (performing an action), reporters (returning a value), or predicates (returning a Boolean true/false value).38
* **Examples of Functionality and Inputs:**
  + define [my custom block]: This is the Hat block that serves as the definition header for a custom block's script. It can include placeholders for inputs, for example, define jump (height).
  + [my custom block]: This represents the callable instance of the custom block. Its inputs correspond to those defined in the define block.
  + **Inputs for Custom Blocks:** When a user creates a custom block, they can add input parameters. These inputs are visually represented as purple ovals containing the input's name and function as local variables within the custom block's definition.38

The system also contains "hidden" opcodes specifically related to Custom Blocks, such as procedures\_prototype, argument\_editor\_boolean, argument\_editor\_string\_number, and procedures\_declaration.13 These are internal opcodes used in the underlying creation and definition process of custom blocks and their arguments, rather than being directly exposed as user-facing blocks.

My Blocks are explicitly designed for "more advanced applications of coding" 4 and serve to "take a collection of blocks and put them all together in its own reusable coding block".37 This directly supports the programming principles of abstraction, which involves hiding complexity, and modularity, which entails breaking down problems into smaller, manageable units. This category acts as a critical bridge for users transitioning from simple scripts to more complex, organized projects. It teaches the value of functions and procedures, which are fundamental components of efficient and maintainable code in any programming language. It empowers users to effectively manage complexity as their projects expand, thereby enabling them to build more ambitious creations.

The presence of hidden opcodes like procedures\_prototype and procedures\_declaration 13 specifically for custom blocks indicates that even user-defined abstraction in Scratch relies on a sophisticated internal representation. These custom blocks are not merely simple groupings of existing blocks but involve a structured definition within the project's JSON. This demonstrates that while the user experience of creating a custom block is straightforward, the underlying system must meticulously manage the block's definition, its inputs, and its relationship to the scripts it encapsulates. This internal complexity is necessary to enable the powerful reusability and modularity that custom blocks provide, reflecting how even high-level abstractions are built upon lower-level, structured components.

## **IV. Advanced Considerations and Special Block Types**

Beyond the commonly used blocks readily available in the Scratch 3.0 palette, the system incorporates internal and historical block types that are not typically visible. Understanding these elements provides a more comprehensive view of the Scratch VM's architecture and its evolutionary path.

### **Hidden Blocks: Internal Opcodes and Their Historical Context**

Hidden blocks are programmatic entities that reside within Scratch's codebase but are intentionally not displayed in the standard user interface's Block Palette.13 These blocks can be inserted into Scratch projects by directly editing the project's underlying

.json file.13 A significant number of these hidden blocks exist primarily for backward compatibility with older versions of Scratch.13 This means that even if they no longer function properly or have been superseded by new implementations, their opcodes are retained to ensure that older projects can still load without errors. Examples include

motion\_scroll\_right, looks\_hideallsprites, control\_for\_each, and data\_listindexall.13 Some hidden opcodes are also internal components of the system, such as those related to custom block creation like

procedures\_prototype and argument\_editor\_boolean.13 A unique "Undefined Hat Block," often appearing as a red block, is generated by errors and possesses no functional purpose, serving instead as a visual indicator of a problem within the project file.13

### **Obsolete Blocks: Brief Overview**

Throughout Scratch's development, certain blocks from previous versions have been removed or replaced in Scratch 3.0. Examples include the "Forever If ()" block from Scratch 1.4 and the "Stop Script" and "Stop All" blocks, which were later merged into the more versatile stop () block.17 The functionality of these obsolete blocks can often be replicated using alternative methods or combinations of current blocks.17 The decision to remove such blocks is typically driven by efforts to reduce complexity, eliminate redundancy, or streamline the user interface.17

### **Extensions: How New Blocks are Integrated**

Scratch extensions are distinct pieces of JavaScript code designed to introduce new blocks into the Scratch environment.11 These extensions define their new blocks using the same JSON structure—including

opcode, blockType, text, and arguments—as the core blocks.11 Extensions significantly expand Scratch's capabilities, allowing it to interact with external hardware devices such as micro:bit or LEGO MINDSTORMS EV3, or to provide specialized features like Music, Pen, Video Sensing, Text to Speech, and Translate functionalities.1 Extensions are categorized into Core, Team, Official, or Unofficial types, each with varying levels of maintenance by the Scratch Team and visibility within the block library.12 It is important to note that unofficial extensions are not officially supported for direct integration into the main Scratch editor.11

The presence of hidden and obsolete blocks 13 illustrates a continuous balancing act in software development. Features may be removed for reasons of simplicity or improved design, yet their internal opcodes are retained to ensure that older projects remain functional.12 This approach means that while the public-facing user interface is kept clean and streamlined, the underlying Virtual Machine carries a historical record of past functionalities. This highlights a common challenge in evolving software platforms: how to introduce new features and enhance user experience without compromising existing content. Scratch's solution of retaining non-functional opcodes for compatibility is a pragmatic engineering decision, even though it adds a layer of complexity to the internal codebase. This demonstrates that software evolution involves not only the addition of new features but also the careful management of legacy components.

The detailed explanation of how extensions are constructed 11, utilizing a defined JSON structure for blocks and JavaScript methods for their logic, reveals the inherent extensibility of Scratch. The fact that extensions can facilitate interaction with "external devices" 1 such as micro:bit or LEGO Mindstorms 7 represents a significant capability. This transforms Scratch from a purely on-screen programming tool into a platform capable of physical computing and interactions akin to the Internet of Things (IoT). For educators, this means Scratch can be employed to introduce concepts of hardware programming and real-world control. For developers, it indicates that the Scratch VM is designed as a robust, extensible runtime, enabling a broad spectrum of custom integrations beyond the core set of blocks. The precise definitions of

opcode and blockType within extensions serve as the blueprint for this extensibility.

## **V. Conclusion**

This report has provided a comprehensive, expert-level analysis of Scratch 3.0 blocks, detailing their internal JSON structure, opcodes, functionality, inputs, and illustrative examples across key categories. The analysis has demonstrated how Scratch's design prioritizes accessibility and user experience through its visual, forgiving block system, while simultaneously maintaining a robust underlying architecture that supports backward compatibility and extensibility.

The importance of opcodes as stable, programmatic identifiers is paramount; they establish the contract that enables the Scratch Virtual Machine to execute block functionality, irrespective of the visual representation or historical context of a project. The detailed examination of block types—Hat, Stack, Boolean, Reporter, C, and Cap—clarifies the mechanisms by which scripts are initiated, executed, and controlled. Furthermore, the analysis of inputs and their specified types underscores the structured nature of block parameters, which is essential for precise programming.

Beyond surface-level descriptions, this report has uncovered deeper observations regarding Scratch's design philosophy. These include its "forgiving" input handling, which is designed to prevent runtime errors; its effective implementation of event-driven programming for dynamic interactions; and the critical role of data structures in managing project state. The report also highlights the performance considerations related to underlying JavaScript implementations, particularly concerning string and list manipulations, and how the commitment to backward compatibility necessitates the retention of "hidden" or obsolete opcodes within the system.

The structured information presented here is invaluable for advanced Scratch users, educators, and software developers. It serves not only as a detailed reference but also illuminates the sophisticated engineering decisions that underpin a platform designed to make programming accessible to a broad audience. A thorough understanding of these technical underpinnings empowers users to create more complex and optimized projects, effectively bridging the conceptual gap between visual programming and core computer science principles.

#### Works cited

1. en.scratch-wiki.info, accessed on July 1, 2025, <https://en.scratch-wiki.info/wiki/Scratch_3.0>
2. Blocks, accessed on July 1, 2025, <https://sjbdigital.com/classesPDF/scratchBlockList.pdf>
3. What are blocks | Introduction to Scratch 3.0 - YouTube, accessed on July 1, 2025, <https://www.youtube.com/watch?v=LK6ISj_r2ds>
4. How Do You Code in Scratch? Fun Blocks - Create & Learn, accessed on July 1, 2025, <https://www.create-learn.us/blog/how-do-you-code-on-scratch/>
5. Scratch Basics Episode 4: Adding Sound - Pinnguaq, accessed on July 1, 2025, <https://pinnguaq.com/learn/scratch-basics-series/scratch-basics-episode-4/>
6. All Blocks of Scratch 83, accessed on July 1, 2025, <https://scratched.gse.harvard.edu/sites/default/files/a.all_blocks_of_scratch_0.pdf>
7. Eng:Block Categories - Test-Scratch-Wiki, accessed on July 1, 2025, <https://test.scratch-wiki.info/wiki/Eng:Block_Categories>
8. Block Categories - Snap! Wiki, accessed on July 1, 2025, <https://snapwiki.miraheze.org/wiki/Block_Categories>
9. Blocks, accessed on July 1, 2025, <https://sjbdigital.com/class/lessonsPDF/scratchBlockList.pdf>
10. Block types | Scratch 3 Walkthrough and Demo - Adafruit Learning System, accessed on July 1, 2025, <https://learn.adafruit.com/guide-to-scratch-3/block-types>
11. Scratch 3.0 Extensions : 8 Steps - Instructables, accessed on July 1, 2025, <https://www.instructables.com/Making-Scratch-30-Extensions/>
12. scratch-vm/docs/extensions.md at develop - GitHub, accessed on July 1, 2025, <https://github.com/scratchfoundation/scratch-vm/blob/develop/docs/extensions.md>
13. Hidden Blocks - Scratch Wiki, accessed on July 1, 2025, <https://en.scratch-wiki.info/wiki/Hidden_Blocks>
14. JSON - Scratch Wiki, accessed on July 1, 2025, <https://en.scratch-wiki.info/wiki/JSON>
15. Hidden Blocks - Scratch Wiki, accessed on July 1, 2025, <https://www.en.scratch-wiki.info/wiki/Hidden_Blocks>
16. Events Blocks - Scratch Wiki, accessed on July 1, 2025, <https://en.scratch-wiki.info/wiki/Events_Blocks>
17. Control Blocks - Scratch Wiki, accessed on July 1, 2025, <https://en.scratch-wiki.info/wiki/Control_Blocks>
18. Operators Blocks - Scratch Wiki, accessed on July 1, 2025, <https://en.scratch-wiki.info/wiki/Operators_Blocks>
19. A Guide to Scratch-Blocks Themes - Discuss Scratch, accessed on July 1, 2025, <https://scratch.mit.edu/discuss/post/3015698/>
20. When I add a custom block to the source code I get a red hat without text · Issue #7997 · scratchfoundation/scratch-gui - GitHub, accessed on July 1, 2025, <https://github.com/LLK/scratch-gui/issues/7997>
21. Say blocks appear further a part so the conversation between sprites looks weird · Issue #1591 · scratchfoundation/scratch-gui - GitHub, accessed on July 1, 2025, <https://github.com/LLK/scratch-gui/issues/1591>
22. Altering sound effects (volume, pitch, etc) causes script pause · Issue #2079 · scratchfoundation/scratch-vm - GitHub, accessed on July 1, 2025, <https://github.com/LLK/scratch-vm/issues/2079>
23. 10240 limit for variables · Issue #922 · scratchfoundation/scratch-vm - GitHub, accessed on July 1, 2025, <https://github.com/LLK/scratch-vm/issues/922>
24. Replace-Item-of-List Block is very slow on large lists (Scratch 3.0 regression) #1934 - GitHub, accessed on July 1, 2025, <https://github.com/LLK/scratch-vm/issues/1934>
25. Motion Blocks in Scratch - Robotique Tech, accessed on July 1, 2025, <https://www.robotique.tech/robotics/motion-blocks-in-scratch/>
26. Looks Blocks | Scratch Wiki | Fandom, accessed on July 1, 2025, <https://en-scratch.fandom.com/wiki/Looks_Blocks>
27. Scratch Blocks Explained | PDF | Scratch (Programming Language ..., accessed on July 1, 2025, <https://www.scribd.com/document/614595778/Scratch-Blocks-Explained>
28. Scratch Basics Episode 4: Adding Sound - Pinnguaq, accessed on July 1, 2025, <https://pinnguaq.com/wp-content/uploads/2020/06/pinnguaq-video-tutorial-scratch-basics-episode-4.pdf>
29. Sound blocks in Scratch - Robotique Tech, accessed on July 1, 2025, <https://www.robotique.tech/robotics/sound-blocks-in-scratch/>
30. Event Blocks in Scratch Programming - UnicMinds, accessed on July 1, 2025, <https://unicminds.com/events-scratch-programming/>
31. Control Blocks – List - Scratch (Coding) - SARASWATI ENGLISH HIGH SCHOOL, accessed on July 1, 2025, <https://sehschool.com/courses/scratch-coding/lessons/sensing-blocks-list/>
32. block tier list :D : r/scratch - Reddit, accessed on July 1, 2025, <https://www.reddit.com/r/scratch/comments/1g40abd/block_tier_list_d/>
33. Operators | Getting started with Scratch - Code Club Projects, accessed on July 1, 2025, <https://projects.raspberrypi.org/en/projects/getting-started-scratch/11>
34. () of () (Operators block) - Scratch Wiki, accessed on July 1, 2025, <https://en.scratch-wiki.info/wiki/()_of_()_(Operators_block)>
35. Reporter Block | Scratch Wiki | Fandom, accessed on July 1, 2025, <https://en-scratch.fandom.com/wiki/Reporter_Block>
36. How Do You Use Variables? - Discuss Scratch, accessed on July 1, 2025, <https://scratch.mit.edu/discuss/post/8407299>
37. Scratch Tutorial: Explore Variable Blocks - YouTube, accessed on July 1, 2025, <https://www.youtube.com/watch?v=RjXLqrIZz4k>
38. This page uses materials from Scratch Wiki's page Snap! (programming language) which is licensed under CC BY-SA 4.0., accessed on July 1, 2025, <https://snapwiki.miraheze.org/wiki/Snap!>
39. How to Use Variables in Scratch 3.0 Tutorial - YouTube, accessed on July 1, 2025, <https://www.youtube.com/watch?v=wS1QETPfgGE>
40. Length of () (List block) - Scratch Wiki, accessed on July 1, 2025, <https://scratchwiki-loader.futuresight.org/wiki/Length_of_()_(List_block)>
41. How to Develop Your Own Block for Scratch 3.0 | by Roy Hiroyuki OSAKI | Medium, accessed on July 1, 2025, <https://medium.com/@hiroyuki.osaki/how-to-develop-your-own-block-for-scratch-3-0-1b5892026421>