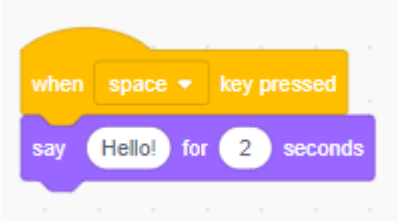
“Skills to Catch the Future”
Getting Started with Scratch:
A Beginner’s Guide

What is coding?

Coding is the process of creating computer programs by writing and testing lines of code. Code is a set of instructions that a computer can interpret and execute, and it is written in a programming language. There are many different programming languages, such as Python, C++, and Java, JavaScript, Scratch, that can be used to create code. Coding is an important skill in the digital age, as it allows people to build and create software, websites, and apps.

An example in scratch

Code	Output
	

This code will cause the sprite to display the text "Hello" on the screen and speak it out loud for 2 seconds when the space bar is pressed.

Why adding coding to curriculums?

- Because code is everywhere
- Coding is a valuable skill: Coding skills are in high demand, and the ability to code can open up a wide range of career opportunities.
- Coding teaches problem-solving skills: Coding requires you to break down complex problems into smaller, manageable pieces and then find solutions to those problems. This can help pupils develop problem-solving skills that are applicable to a wide range of fields.
- Coding helps pupils understand technology: Coding is a fundamental part of how technology works, and learning to code can help pupils understand and work with technology in a more meaningful way.
- Coding can be fun and rewarding: Many pupils find coding to be a fun and rewarding activity, and it can be a great way to encourage creativity and innovation.
- Coding allows students to create content, not just consume it.
- Coding allows them to be the creator of their own stories and stretches them to be able to make it fun for others to consume.
- Coding helps students to take risks and fail and develop their personality through this process. But the error here is not costly and has no negative effect because it can be corrected directly

- Coding develops teamwork and collaborative skills

- It is for everyone

What is computational thinking?

Computational thinking is a way of solving problems, designing systems, and understanding human behavior that draws on concepts and techniques from computer science. It involves breaking down complex problems into smaller, more manageable parts, and using logical reasoning and data analysis to find solutions.

Computational thinking skills are useful in a wide range of fields, including computer science, data science, education, and business. They are also important for understanding and participating in the digital world.

Some key principles of computational thinking include:

- **Decomposition:**
  - Breaking down complex problems or system into smaller parts that are more manageable and easier to understand.
  - The smaller parts can then be examined and solved, or designed individually, as they are simpler to work with.
  - The process of breaking down a complex problem into smaller, manageable parts is called **Analysis.** This involves identifying the major components of the problem, understanding their relationships, and finding solutions to the identified components.
  - Analysis should involve both qualitative and quantitative methods and involve looking at the problem from multiple perspectives.
  - In contrast, the process of combining the smaller components identified during the analysis stage into a unified solution is called **Synthesis**. This involves combining the different perspectives and solutions identified in the analysis stage to create an overall solution to the problem. Synthesis takes a holistic view of the problem and involves the integration of multiple solutions and perspectives. This process encourages creativity and innovation and can help to identify solutions that are more effective and efficient than those identified in the analysis stage.

    Analysis
    Examples of quantitative methods include:

    1. Cost-benefit analysis: This method involves assessing the relative costs and benefits associated with different solutions to a problem.
    2. Statistical analysis: This method uses data and statistics to analyze the problem and identify potential solutions.
    3. Simulation: This method uses computer models to simulate scenarios and predict potential outcomes.
    4. Optimization: This method uses mathematical algorithms to optimize a solution so that it is the most efficient and effective.

Examples of qualitative methods include:

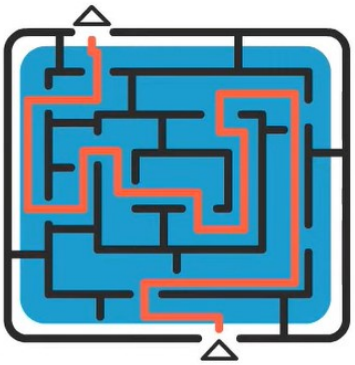
1. Interviews: This method involves speaking to stakeholders and gathering information to gain a better understanding of the problem.
2. Surveys: This method uses questionnaires to identify potential solutions.
3. Brainstorming: This method involves gathering a group of people together to brainstorm ideas and potential solutions.
4. Observation: This method involves observing people and processes to gain a better understanding of the problem.

Synthesis

Examples of synthesis in problem decomposition include:

1. Combining multiple solutions: This involves combining different solutions identified during the analysis stage to create a unified solution.
2. Creating a new solution: This involves using the components identified in the analysis stage to create a new solution to the problem.
3. Adapting existing solutions: This involves adapting existing solutions to fit the problem better.
4. Refining existing solutions: This involves making small changes to existing solutions to make them more effective.

- **Pattern recognition:**
  - Once you have decomposed a complex problem, it helps to examine the small problems for similarities or “patterns”.
  - These patterns can help you to solve complex problems more efficiently.
  - When you decompose a complex problem, you often find patterns among the smaller problems you create.
  - The patterns are similarities or characteristics that some of problems share.
  - It involves finding similarities or patterns among small, decomposed problems that can help us solve more complex problems more efficiently.
  - Identifying patterns and trends in data using them to make predictions or generalizations. “Lets one object stand for many”.
  - The more patterns you can find, the easier and quicker our overall task of problem solving will be.
  - To find patterns in problems you look for things that are the same (or very similar) in each problem.



- **Abstraction:**
  - Abstraction is a fundamental part of computational thinking, as it allows us to break down complex problems into simpler, more manageable tasks.
  - By abstracting away the details, we can identify patterns and recognize similarities across different problems. We can then develop efficient solutions by focusing on the most important aspects of the problem first.
  - Abstraction also helps us to identify patterns and recognize similarities across different situations, allowing us to generalize solutions and apply them to a variety of problems.
  - Finally, abstraction helps us to develop efficient solutions, by allowing us to identify the most important aspects of the problem and focus on those first.

- **Algorithm design:**
  - Algorithm design in computational thinking is the process of designing a step-by-step set of instructions that can be used to solve a problem. This involves breaking down a problem into smaller, manageable parts and then creating a sequence of instructions that can be used to solve it. Algorithm design requires the use of sound problem-solving methods, algorithmic logic, and computational thinking skills. Examples of algorithm design include:

    1. Search algorithms: These algorithms are used to search for and find a specific item or solution in a large data set.

2. Sort algorithms: These algorithms are used to sort items in a specific order.
3. Optimization algorithms: These algorithms are used to find the optimal solution to a problem.
4. Pathfinding algorithms: These algorithms are used to find the shortest path between two points.

- Evaluation and testing: Testing and evaluating an algorithm involves running it through a variety of tests to ensure that it works correctly and produces the desired result. This process includes testing the algorithm on different types of data, testing for edge cases, and testing for errors. It also involves evaluating the performance of the algorithm and making sure that it is efficient and effective. Examples of testing and evaluating an algorithm include:

  1. Sanity checking: This involves running the algorithm on simple test cases to make sure it works correctly.
  2. Performance testing: This involves running the algorithm on larger datasets to make sure it is efficient and effective.
  3. Edge case testing: This involves testing the algorithm with extreme inputs to make sure it handles them correctly.
  4. Error testing: This involves testing the algorithm for potential errors and making sure it handles them correctly.

The Definition of a problem

In computational thinking, a problem refers to a situation or challenge that requires the use of computational methods or techniques to solve or achieve a specific goal or outcome. A problem in computational thinking can be defined as a specific task or set of tasks that need to be completed using a computer or other information processing system, such as a phone or tablet.

Problems in computational thinking are characterized by their complexity, ambiguity, and the need for decomposition, pattern recognition,

abstraction, and algorithms to solve them. They can be found in a wide range of fields, including computer science, mathematics, engineering, science, and business.

Problems in computational thinking typically involve breaking down complex systems into smaller, manageable parts and then using algorithms, data structures, and other computational methods to solve them. The process of solving a problem in computational thinking often involves the use of logical reasoning, mathematical analysis, and other analytical techniques to understand and solve the problem.

It is possible for a problem in computational thinking to have none, one, or multiple solutions.

A problem may have no solutions if it is impossible to solve or if there are too many constraints that cannot be met. In this case, the problem is considered to be unsolvable.

A problem may have one solution if there is a clear and unique solution that can be found using computational methods. In this case, the problem is considered to be well-defined.

A problem may have multiple solutions if there are different ways to solve the problem and many possible outcomes. In this case, the problem is considered to be ill-defined.

It's important to note that, when a problem has multiple solutions, it's important to evaluate the trade-offs and the best approach to solve the problem. Sometimes, there will be a best solution which is the one that is more efficient, more reliable, or less expensive than other solutions. And other times, there will be a solution that is better suited for a specific context or use case.

Coding vs Computational Thinking

Coding and computational thinking are closely related concepts.

Computational thinking is a way of thinking that involves breaking down complex problems into smaller, manageable pieces and finding solutions to those problems using algorithms, data, and other tools.

Coding is the process of creating these algorithms and solutions using a programming language.

In other words, computational thinking involves thinking like a computer, and coding is the process of turning those thoughts into action.

Coding is one way to apply computational thinking, but it is not the only way. Computational thinking can also be applied in other fields, such as data analysis, engineering, and even the arts.

Overall, computational thinking and coding are important skills to have in the digital age, as they allow you to think critically and solve problems in a logical and systematic way.

Computational thinking can contribute to education in several ways:

Problem solving skills: Computational thinking helps students develop problem solving skills by breaking down complex problems into smaller, more manageable parts and using logical reasoning and data analysis to find solutions.

- **Creativity and innovation:** Computational thinking encourages creativity and innovation by giving students the tools to design and build their own systems and solutions.

- **Collaboration and communication:** Computational thinking promotes collaboration and communication as students work together to design and build solutions to problems.

- **Critical thinking and evaluation:** Computational thinking encourages critical thinking and evaluation as students test and refine their solutions to ensure they are effective and accurate.

- **Computational literacy**: Computational thinking helps students become computationally literate, which is important for understanding and participating in the digital world.

Overall, computational thinking can help students develop a wide range of skills that are important for success in the 21st century

The difference between an algorithm and a program
An algorithm is a step-by-step procedure for solving a problem, while a program is a specific implementation of an algorithm, written in a programming language, that can be executed by a computer.

Unplugging from Computational Thinking: Creative Activities for All!

Exploring Unplugged Computational Thinking Activities
Unplugged computational thinking activities are activities that help children develop problem solving and computational thinking skills without the use of technology. These activities can be used to teach concepts such as coding, debugging, and algorithms in a fun and interactive way. Some examples of unplugged activities include board games, puzzles, and role-playing games.

Unplugged activities can help children develop problem solving skills and learn to think logically. They can also help children understand the basics of coding and algorithms.

These activities can be used to teach children how to debug and troubleshoot problems, as well as how to use a systematic approach to solve challenges.

Unplugged activities are also great for teaching children to think creatively and develop their own unique solutions to problems.

Benefits of Unplugged Computational Thinking Activities

Unplugged activities can help children learn to think critically and solve problems. They can also help children develop logical thinking skills, as well as the ability to break down complex tasks into smaller, more manageable steps. Unplugged activities can also help children develop their creativity and ability to think outside the box.

Unplugged activities can also help children understand the concepts of coding and algorithms.

By playing board games and puzzles, children can learn how to debug and troubleshoot problems, as well as how to use a systematic approach to solve challenges.

Unplugged activities are also great for teaching children to think creatively and develop their own unique solutions to problems.

Examples of Unplugged Computational Thinking Activities

Some examples of unplugged activities include board games, puzzles, and role-playing games. Board games can help children learn to think logically and develop problem solving skills. Puzzles can help children understand the basics of coding and algorithms. Role-playing games can help children develop their creativity and ability to think outside the box.

Unplugged activities can also include activities such as treasure hunts, scavenger hunts, and mazes.

These activities can help children learn to debug and troubleshoot problems, as well as how to use a systematic approach to solve challenges.

Unplugged activities are also great for teaching children to think creatively and develop their own unique solutions to problems.

Using Unplugged Computational Thinking Activities in the Classroom

Unplugged activities can be used in the classroom to help children develop problem solving and computational thinking skills. These activities can be used to teach concepts such as coding, debugging, and algorithms in a fun and interactive way. Unplugged activities can also help children understand the basics of coding and algorithms, as well as how to debug and troubleshoot problems.

Unplugged activities can also be used to teach children how to use a systematic approach to solve challenges.

These activities can also help children develop their creativity and ability to think outside the box.

Unplugged activities are also great for teaching children to think creatively and develop their own unique solutions to problems.

Tips for Teaching Unplugged Computational Thinking Activities

When teaching unplugged activities, it is important to make sure that the activities are engaging and accessible to all children. It is also important to make sure that the activities are age appropriate and that they challenge the children to think critically. It is also important to provide enough time for the children to complete the activities and to provide feedback and support throughout the process.

It is also important to ensure that the activities are fun and interactive. This will help to keep the children engaged and motivated.

Finally, it is important to provide enough time for the children to reflect on their experiences and to discuss their learning.

This will help to reinforce the concepts that they have learned and to ensure that they have a deeper understanding of the material.

Plugged Coding Activities with Scratch MIT

Plugged coding activities are an innovative way to introduce computational thinking to students. Scratch MIT is a programming language that allows students to create interactive stories, animations, and games. It is an ideal platform for teaching basic coding concepts and problem-solving skills.

Coding is a critical skill for students of all ages. With the introduction of Plugged Coding Activities with Scratch MIT, secondary level schools can now access a wide range of coding activities that are both engaging and educational. Scratch MIT provides an easy-to-use coding platform that allows students to learn the fundamentals of coding in a fun and creative environment.

Benefits of Plugged Coding Activities with Scratch MIT

Scratch MIT provides a range of benefits for secondary level schools. The activities are designed to be engaging and educational, allowing students to learn the fundamentals of coding in a fun and creative environment. Furthermore, Scratch MIT is designed to be accessible to all levels of students, making it ideal for secondary level schools.

Scratch MIT also provides a variety of activities that can be used to teach coding concepts.

From basic concepts such as loops and conditionals to more advanced topics like game design, Scratch MIT offers activities that are both engaging and educational. This makes it ideal for secondary level schools who are looking to introduce coding to their students.

Introduction to Scratch

What is Scratch MIT?
Scratch is a free, open-source programming language and online community developed by the Lifelong Kindergarten group at the Massachusetts Institute of Technology (MIT). It is designed to be easy to use, even for people with no programming experience, and its block-based interface and visual programming language make it accessible to people of all ages and backgrounds.

Scratch allows you to create interactive stories, games, and animations, and it has a wide range of features, including support for sprites, graphics, sound, and basic programming concepts such as variables, loops, and conditionals. The projects made with Scratch can be shared online in the Scratch online community where users can share, collaborate, and learn together.

The Scratch project is also a research project that looks at the impact of new technologies on children's learning and creativity, the Scratch programming language and the online community provides a tool for children and young people to learn to think creatively, reason systematically and work collaboratively – essential skills for life in the 21st century.

Shall scratch be taught in school?
Many educators believe that Scratch should be taught in schools as it is an excellent tool for introducing students to coding and computational thinking.

Scratch has many benefits for education, including its:

- easy to use interface,
- visual block-based programming language,
- ability to create interactive projects,
- large and active online community, and
- wide range of applications in educational settings
- In addition to that, Learning Scratch can also be engaging and fun for students, which can help to motivate them to learn and retain the material.
- Furthermore, Learning Scratch can help students develop key skills such as problem-solving, logic, and critical thinking which are essential for success in the 21st century.

Overall, many educators believe that Scratch can be a valuable tool for teaching coding and computational thinking in schools, and that it can be used to help students develop a wide range of important skills.

The Scratch Interface

The Scratch interface is designed to be intuitive and user-friendly, even for people with no programming experience.

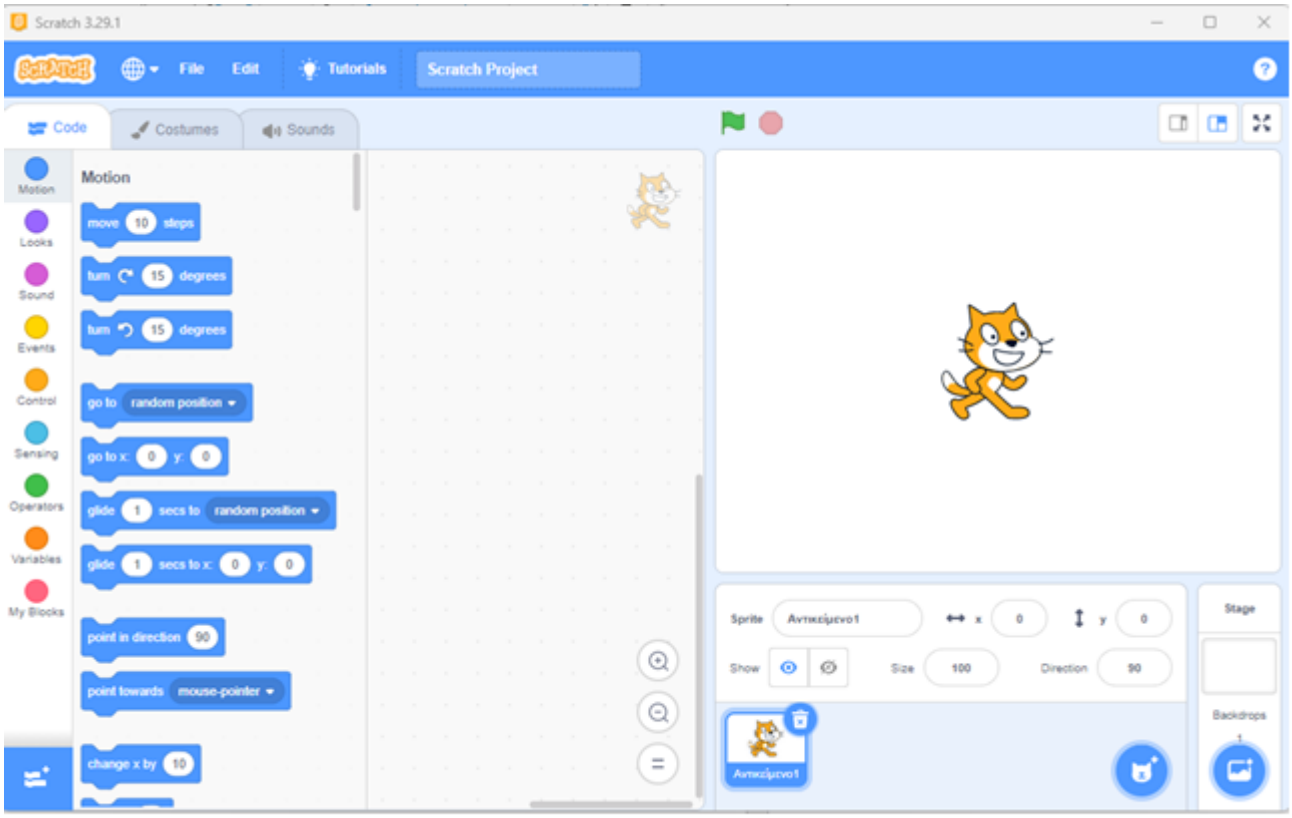


Figure 1: The Scratch Interface

Scratch is a optical programming language with many sprites, which act like different characters. Each sprite has unique characteristics, such as costumes, size, position, effects, and private variables.

Blocks are the pieces of code that run the project, and each sprite has its own blocks that are independent of other sprites' blocks.

Sprites interact with each other on the Stage, and they can communicate with each other using broadcasts, variables, and sensing blocks.

Blocks from each sprite are displayed in the Code Area. They are structured into scripts, which are connected chains of blocks that run together.

The Block Palette contains a list of all blocks and is where blocks are dragged from.

1. Menu Bar

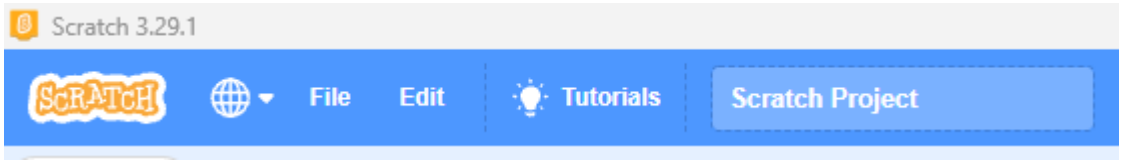


Figure 2: Menu Bar

In Scratch 3.0 offline editor, the Menu Bar is located at the top of the interface and provides access to various menus containing options and commands similar to the online editor. The menu bar typically includes menus such as "File", "Edit", "Tutorials", "Project".

2. Tabs

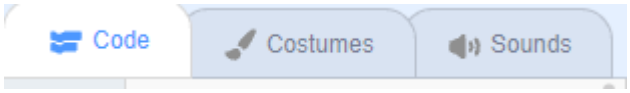


Figure 3: Tabs

In Scratch, the "tabs" refers to the different work areas or workspaces that you can access and work on while creating a project. These tabs are located at the top of the interface and allow you to switch between different areas, such as the script area, the costume area and the sound area. Each tab is associated with different set of blocks and tools, depending on what you want to do.

- The Code Tab: is where you write the code of your project, using the blocks to give instructions to the sprites and make them interact with the stage and with the user.

- The Costumes Tab: is where you manage the different costumes that a sprite can have and can create new costumes or import them from other projects or files.

- The Sounds Tab: is where you manage the different sounds that a sprite can use and can create new sounds or import them from other projects or files.

Switching between these tabs allows you to focus on different aspects of your project and use the different set of tools and blocks, so you can create animations, games, interactive stories, and many other types of programs.

3. Sprite Pane - List

The Sprite Pane refers to the area of the interface where you can manage and organize the sprites that you are using in your project. This area is typically located on the left side of the Scratch interface and it allows you to access information and properties of the sprites and also to create new sprites or import them from other projects or files.

The sprite pane typically contains several sections:

- The sprite list: where you can see the list of sprites that are currently in your project and you can select one of them to work on.

- The sprite thumbnail: where you can see a preview of the sprite you're working on, and you can also select the costume that you want to use.
- The sprite info: where you can access information about the sprite like its name, size, and position, and you can also set the visibility and effects of the sprite.

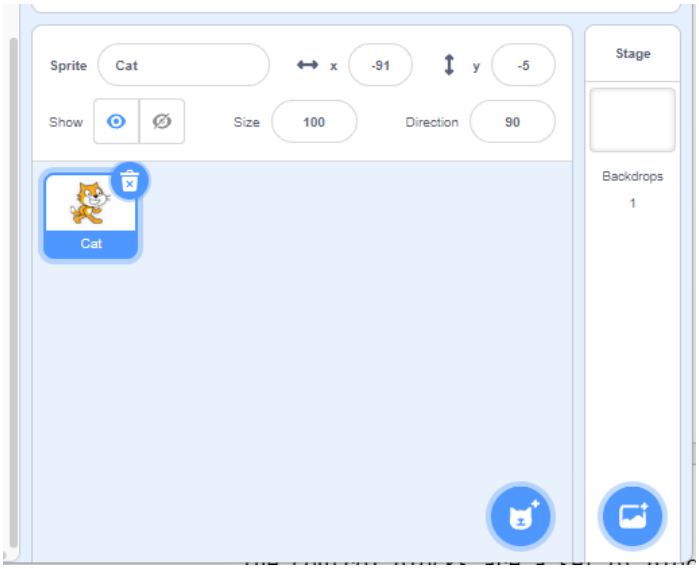


Figure 4: Sprite Pane

- The button to create a new sprite: where you can create a new sprite from scratch or import it from a file or from the library.

- The sprite pane allows you to easily manage and organize the sprites that you are using in your project and it also allows you to easily switch between the sprites and work on different aspects of your project.

Each sprite has its own scripts. When you click the sprite in the Sprite List, that sprite's scripts will display in the Code Area, and its properties will display in the Sprite Pane

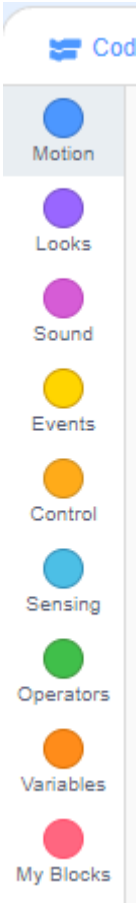
4. Code Palette

The Code Palette consists of the a) Code Categories and the b) Code Blocks

There are 9 Code Categories and each of them has several Code Blocks:

a) Code Categories

.



1. Motion,
2. Looks,
3. Sound,
4. Events,
5. Control,
6. Sensing,
7. Operators,
8. Variables,
9. My Blocks,

and other categories for any extensions.

Some blocks can move the sprite or change its looks, some interact with other sprites or the Stage, while others perform internal operations, such as mathematics or changing variables and lists.

Figure 5: Code Categories

b) Code Blocks

- **Motion**
  These are a set of blocks that are used to control the movement and position of sprites on the stage. These blocks are found in the "Motion" category of the blocks palette, and can be used to change the x and y coordinates of a sprite, rotate a sprite, and set the direction and size of a sprite

- **Looks**
  The looks blocks can be used to change the sprite's costume, to set the visibility and effect of sprites, and to show or hide the sprites on the stage

- **Sound**
  The sound blocks are a set of blocks that are used to control the sound of sprites and the stage
- **Events**

The event blocks are a set of blocks that are used to trigger scripts or sequences of blocks based on certain events or conditions. They can be used to respond to user input, such as keyboard presses or mouse clicks, or to create loops or animations

- **Control**
  The control blocks are a set of blocks that are used to control the flow of a program and to create complex behaviors and interactions. They can be used to create loops, conditionals, and functions

- **Sensing**
  The sensing blocks are a set of blocks that are used to detect and respond to the state of the sprites and the stage. They can be used to detect the position of the sprite, the position of the mouse, the color of the pixel under the sprite, and to detect if certain keys or the mouse button are being pressed

- **Operators**
  The operator blocks are a set of blocks that are used to perform mathematical or logical operations. They can be used to add, subtract, multiply, divide, and calculate the remainder of numbers, to compare numbers and strings, and to perform logical operations such as and, or, not, and others

- **Variables**
  The variable blocks are a set of blocks that are used to create, manipulate, and access variables. Variables are used to store and retrieve data that can be used to change the behavior of the program or to keep track of certain values. They and can be used to create new variables, to set the value of existing variables, to change the value of existing variables, to retrieve the value of existing variables, and to display the value of variables on the stage

- **My Blocks**
  "My Blocks" are a feature that allows you to group blocks together to create your own custom blocks. These custom blocks can be used just like the built-in blocks in Scratch and can make scripts more organized and easier to read. You can create a "My Block" by defining a stack of blocks that you want to group together, and then giving it a name and parameters. Once you've created a "My Block", you can use it in your scripts just like any other block, and you can also make it available to other sprites in your project.

5.Code (Scripts) Area

In Scratch, the "Code Area" refers to the area of the interface where you can create, edit, and organize the scripts or programs that control the behavior of the sprites and the stage.

This area is where you use blocks to give instructions to the sprites and make them interact with the stage and with the user.

The Code Area is typically located in the center of the Scratch interface, and it is where you write the code using blocks and can organize them using the blocks' categories and the scripts' groups.

In this area you can create new scripts by dragging blocks from the palette and connecting them together to form a script, you can also edit the existing scripts by dragging the blocks around, deleting blocks or adding new ones, you can also organize the scripts by grouping them together, giving them a name and a comment, and you can also use the "My Blocks" feature to create your own block.

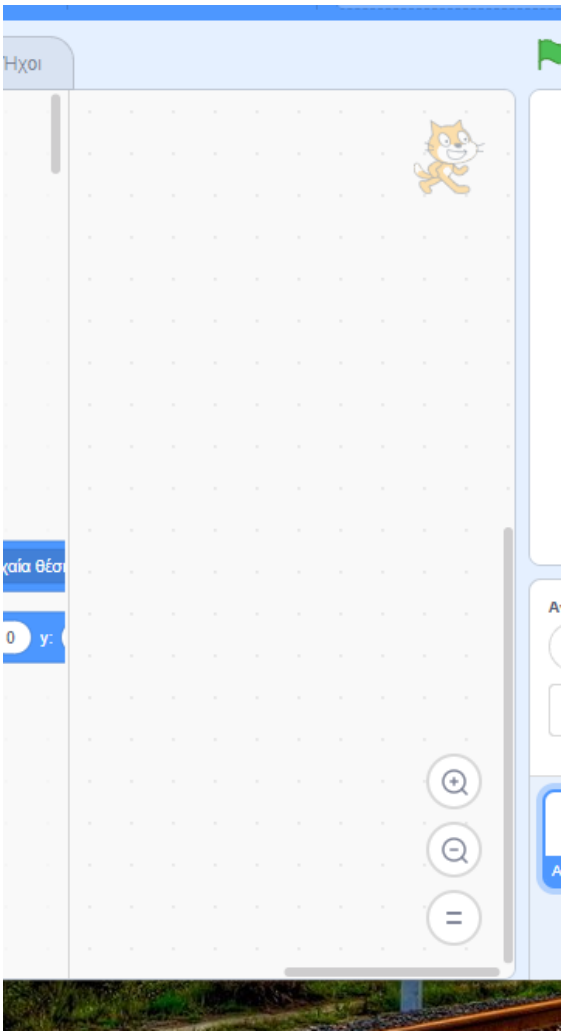


Figure 6: Code Area

6.Stage

In Scratch, the Stage Area" refers to the main visual area of the interface where the sprites and backdrops are displayed.

The stage area is where the sprites can move and interact with the user and with other sprites, and it's also where the backdrops or backgrounds are displayed and can be changed.

The stage area is typically located in the right of the Scratch interface and it shows the current background, the sprites that are on it and their current costumes, the position and size of the sprites, and the sound effects that are playing. It also includes the stage monitor that allows you to see the value of certain variables, the position of the mouse, and the loudness of the microphone.

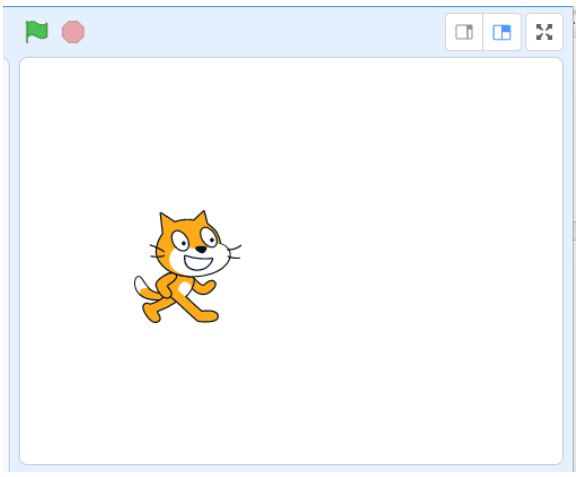


Figure 7: Stage

You can interact with the stage area by clicking on the sprites to select them and move them, by clicking on the backgrounds to change them, by using the zoom and pan buttons to zoom in and out or move the stage around, and by using the full-screen button to view the stage in full screen.

The stage area allows you to see the visual representation of your project and it's the place where the sprites' behaviors and interactions are shown.

The Paint Editor
the Paint Editor is a feature that allows you to create and edit the costumes of a sprite.

A costume is a representation of a sprite that can be changed in order to create animations, to change the appearance of the sprite, or to give the sprite different expressions.

The paint editor is a tool that allows you to create and edit these costumes by drawing and manipulating pixels. You can access the paint editor by clicking on the "Costumes" tab in the sprite pane and then clicking on the "Edit" button for a costume. Once you're in the paint editor, you can use a variety of tools to draw and edit the costume, such as a pencil, an eraser, a paint bucket, and a color picker.

You can also use the zoom and pan buttons to zoom in and out or move the costume around, and you can use the undo and redo buttons to undo and redo your changes.
The paint editor allows you to create and edit the costumes of a sprite in a very precise way, and it allows you to create animations and change the appearance of the sprite in very precise and detailed ways. It also allows you to import and export costumes, which can be useful for sharing costumes with other people or using costumes from other projects.

The main parts of the Paint Editor are

- The drawing tools, which you can select using the buttons on the left side
- The Canvas, where you draw images
- The Costume center, which indicates the center of the costume with the crosshairs symbol
- The Line width selector, which sets the width of the drawing tools
- The Color selectors, which change the color of the drawing tools
- The Zoom buttons for zooming into or out of the canvas
- The Undo and Redo buttons, which can help you correct mistakes

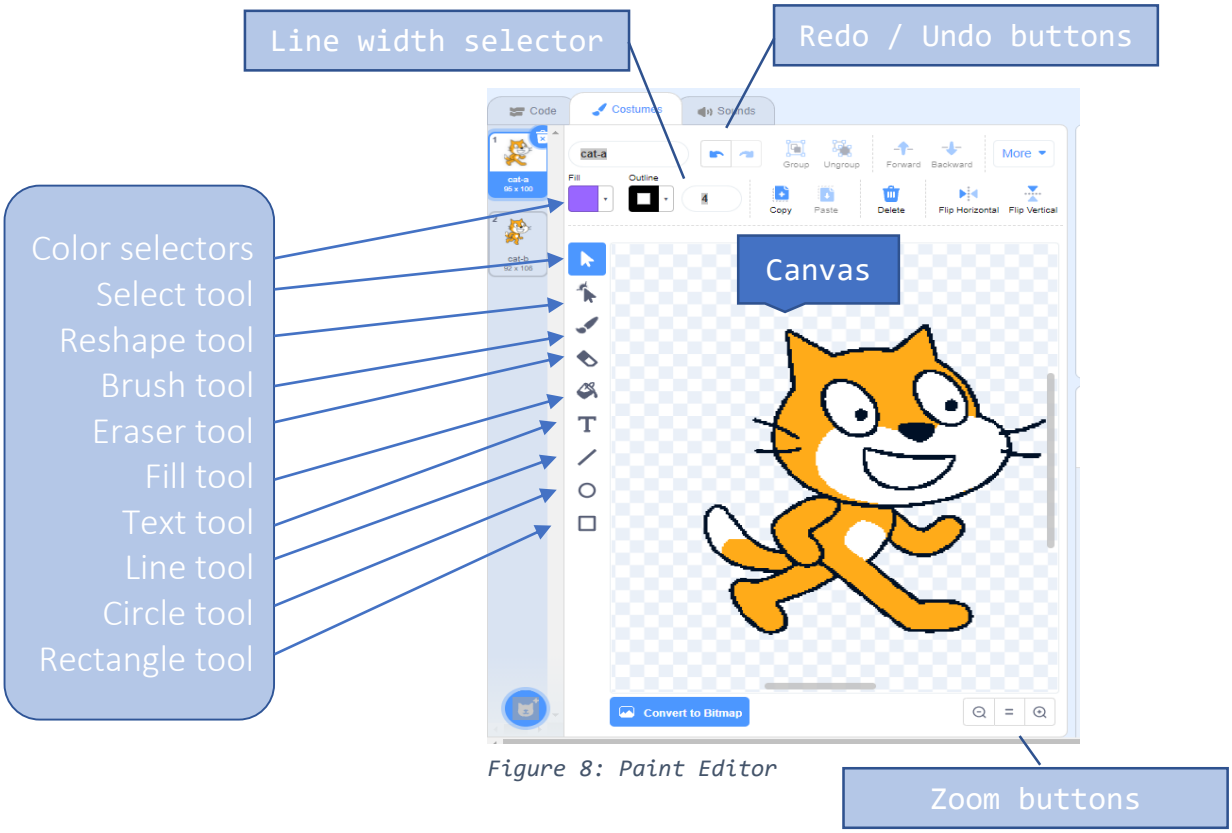


Figure 8: Paint Editor

The Sounds Tab

The Sounds tab is an area of the interface where you can manage and organize the sounds that you are using in your project. This tab allows you to access, create, edit, and import sounds that can be used by the sprites in your project.

The Sounds tab typically contains several sections:

The sound list: where you can see the list of sounds that are currently in your project and you can select one of them to work on.

Viewing Area: allows one to view the sound waves of the selected sound.

Sound Effects Tools: above and below the viewing area are many tools. These include making the sound louder, softer, have effects, fade in or out, copying and pasting, undo, and redo.
The sound info: where you can access information about the sound like its name, and its duration.

The buttons to create a new sound and to import sounds: where you can create a new sound from scratch or import it from a file or from the library.

The Sounds tab allows you to easily manage and organize the sounds that you are using in your project, you can easily switch between the sounds and work on different aspects of your project. It also allows you to add sound effects and background music to your project, to make it more interactive and engaging

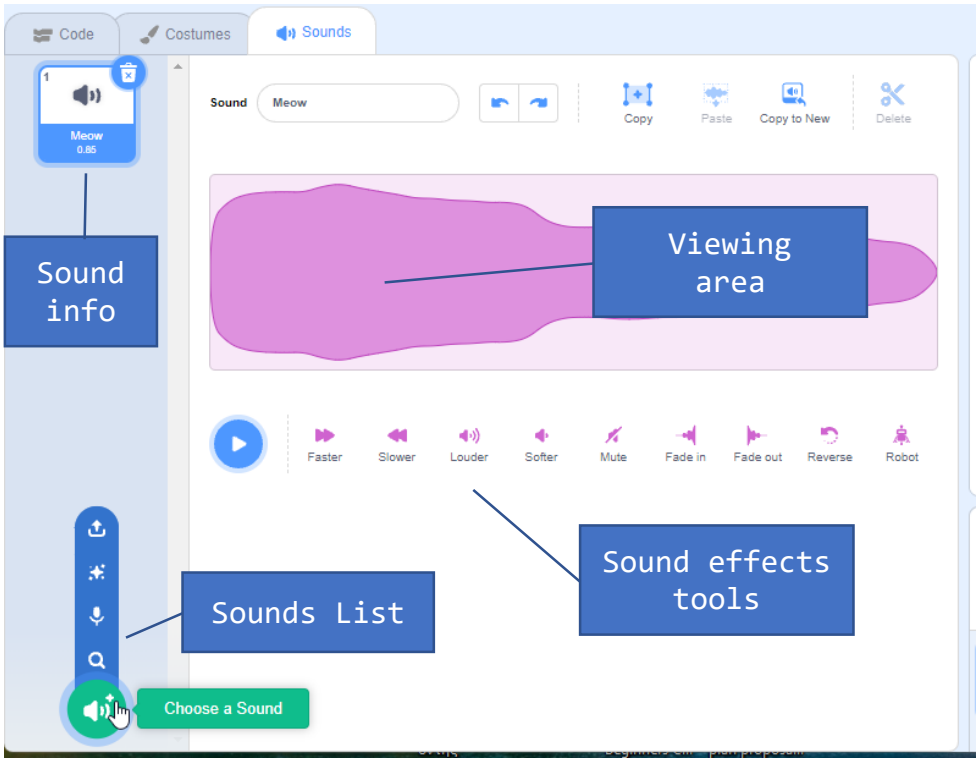


Figure 9: Sounds tab

Let’s start coding with Scratch!!!

The stage is represented as a rectangular area with an X and Y axis. The X-axis extends horizontally across the stage from left to right, and the Y-axis extends vertically from top to bottom. The origin (0,0) of the coordinate system is located in the center of the stage. The coordinates of each point on the stage can be represented by an X-Y pair, with the X-coordinate indicating the position on the horizontal axis and the Y-coordinate indicating the position on the vertical axis.

Sprites, which are the visual objects in a Scratch project, can be moved and positioned on the stage using the Motion blocks, which include commands to change the X and Y coordinates of a sprite. This allows you to create animations and interactive programs where sprites move and respond to user input.

Additionally, the stage also have a background that can be change and you can also add different backgrounds to different scenes.

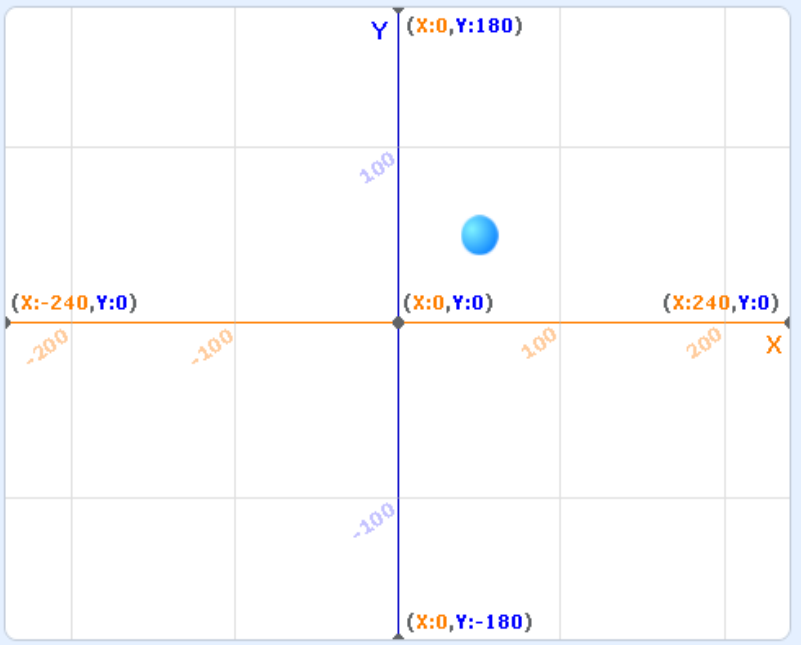


Figure 10: Rectangular Area

How to insert a sprite

The most common way is to use the built-in library of sprites that you can use in your projects. To insert a sprite from the library, click on the "Choose a Sprite from the library" button in the Scratch editor, and then select the sprite you want to use from the library.

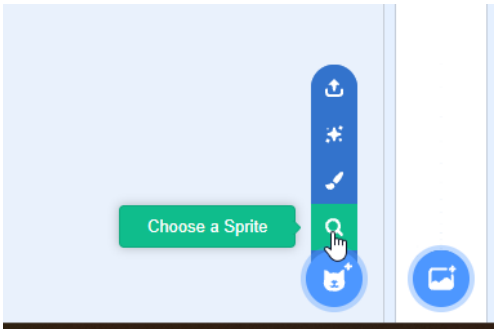


Figure 11: Insert Sprite

Go right down to the Sprite Pane and click on the “Choose a Sprite” button.
Then click on the “Search icon” and a window with several sprites will appear. Choose on which is appropriate for your project

Once you have inserted a sprite, it will appear in the Sprites Pane on the right-hand side of the Scratch editor. You can then use the Motion blocks to control the movement and position of the sprite on the stage.

How to insert a backdrop (background image)

Scratch comes with a built-in library of backdrops that you can use in your projects. To insert a backdrop from the library, click on the "Choose a backdrop from the library" button in the Scratch editor, and then select the backdrop you want to use from the library.

Once you have inserted a backdrop, it will appear in the Backdrop Pane on the right-hand side of the Scratch editor. You can then use the Backdrop block to change the background when you want.

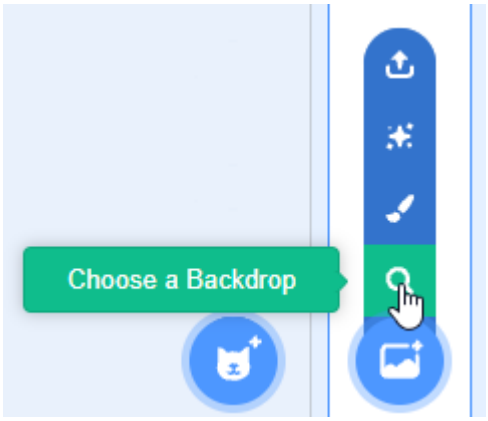


Figure 12: Insert Backdrop

https://youtu.be/WH9FOwQaNio

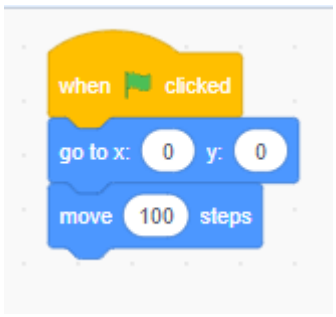
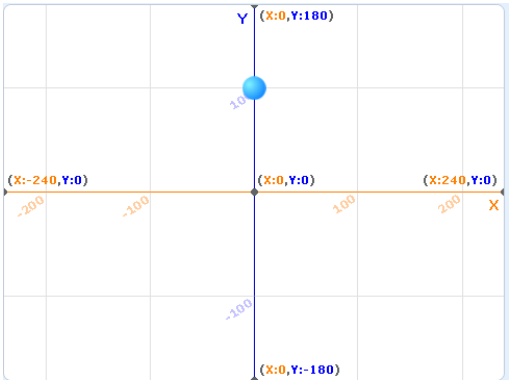
I. The code blocks of the Motion category

The move block.

Let us move the sprite
You can move the sprite in the Stage area

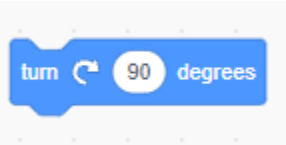
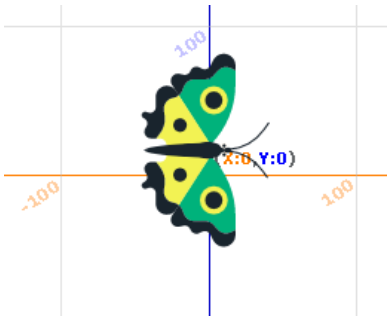
- x-axis from -240 to 240
- y-axis from -180 to 180

For this you can use the move block.

Code	Output
	

https://youtu.be/0IGk3I4MAHo

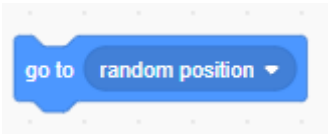
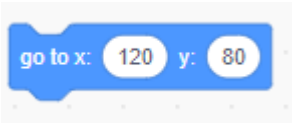
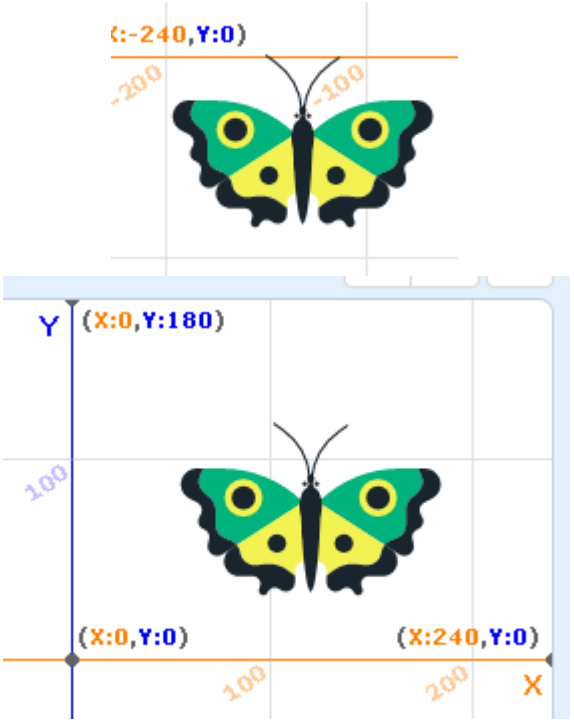
The turn block
You can turn the sprite to the left or right in a range of 0 to 360 degrees Code Output.

Code	Output
	

https://youtu.be/Nlkg26GjnB0

The goto block
It that makes the sprite move to a specific x and y coordinate on the stage. This block can be found in the "Motion" section of the Scratch blocks palette. The x and y coordinates can be input as specific numbers

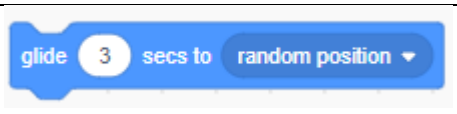
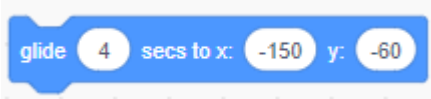
or can be set to variables, allowing for more dynamic movement in a program. The sprite will move to the specified location with a smooth animation.

Code	Output
 The sprite moves to a random position  The sprite moves to the specific x and y coordinates	

https://youtu.be/hmXK0QVovNU

The glide block

It is a motion block that makes the sprite move to a specific x and y coordinate on the stage over a certain amount of time. The x and y coordinates can be input as specific numbers or can be set to variables, allowing for more dynamic movement in a program. The sprite will smoothly glide to the specified location over the specified amount of time. This block allows for more control over the animation of a sprite's movement compared to the "goto" block.

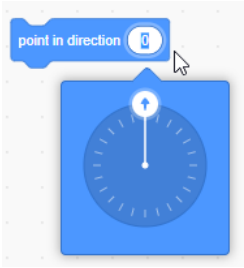
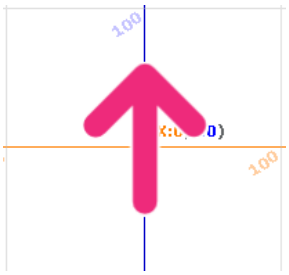
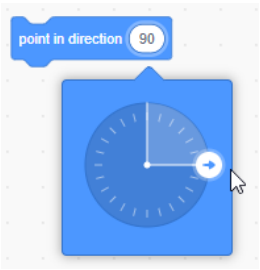
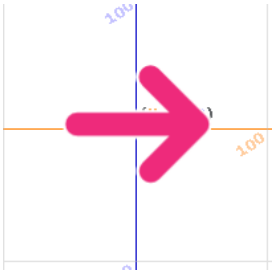
Code
 The sprite glides for 3 seconds to random position   

The sprite glides for 4 seconds to the specific x and y coordinates

https://youtu.be/4fzSQE1Ov4o

The point direction block

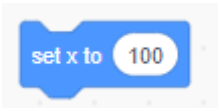
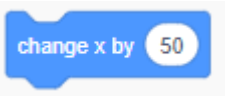
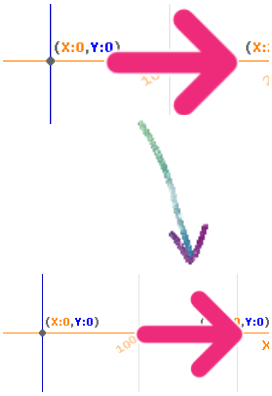
It makes the sprite rotate to a certain direction. The direction is input as a number in degrees, with 0 being to the right, 90 being up, 180 being to the left, and 270 being down. The sprite will rotate to the specified direction instantly. It can be used to set a direction for a sprite to move, or to make a sprite look at a certain point. This block can be used in conjunction with the "point towards" block which makes the sprite rotate to point towards a specific sprite or the mouse pointer.

Code	Output
 The sprites points to the 0 degrees of the circle	
 The sprites points to the 90 degrees of the circle	

https://youtu.be/_XLrOc3EaWQ

Set and Change block

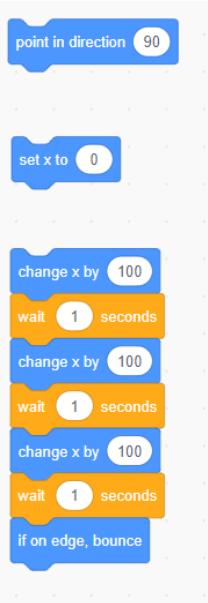
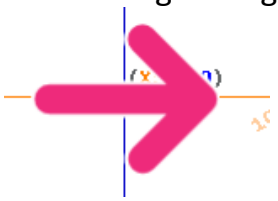
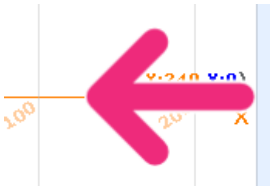
The "set x to" and "set y to" blocks are motion blocks that change the x or y position of a sprite respectively. They allow you to set the x or y position of the sprite to a specific value or to a value stored in a variable. Unlike the "goto" block, it does not animate the movement, it instantly moves the sprite to the position. These blocks can be useful for precise positioning of sprites, or for creating motion that is not smooth or linear.

Code	Output
 The sprite shifts to the position x = 100  The sprite changes its position by 50 and goes to the position x = 150	

https://youtu.be/KJQTrObCm_U

If on edge, bounce block

In Scratch, the "if on edge, bounce" block is a control block that causes a sprite to bounce off the edges of the stage when it reaches them. When the sprite reaches any of the edges of the stage, its direction of motion will be reversed, allowing it to move back into the stage. This block is particularly useful for creating games or simulations where objects move around the screen and need to be contained within the boundaries of the stage. It can also be used to make a sprite move in a specific pattern.

Code	Output
	At the beginning  Afterwards 

https://youtu.be/sJjqkxwthyU

Set Rotation Style block

It is a motion block that sets the rotation style of a sprite. The sprite can be set to have a "left-right" rotation style or a "don't rotate" style or the "all-around" style. When a sprite is set to "left-right" rotation style, it will rotate on the y-axis and only between the leftmost and rightmost positions when turning. When a sprite is set to "don't rotate" style, it will not rotate when turning. Finally, if it is set to "all-around" it will rotate on its x-axis. This block allows for more control over the behavior of a sprite and can be useful for creating certain types of games or simulations.

https://youtu.be/eZZyiML6tG0

The difference between If On Edge Bounce and Set Rotation Style block

The "if on edge, bounce" block, is related to edges, specifically the edges of the stage. It causes a sprite to bounce off the edges of the stage when it reaches them. So, it's used to handle the sprite's behavior when it

reaches the edges of the stage, while the "set rotation style" block is used to control the way a sprite rotates when it moves.

II. The code blocks of the Looks category

Say block

It causes a sprite to display a specified message in a speech bubble above its head. The message can be input as a string of text or as a variable. The sprite will display a speech bubble with the specified message, but it will not speak it out loud. This block can be useful for creating interactive stories, games, or simulations where characters need to communicate with the user.

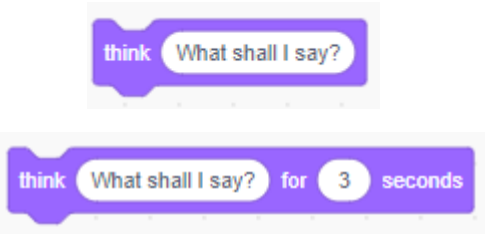
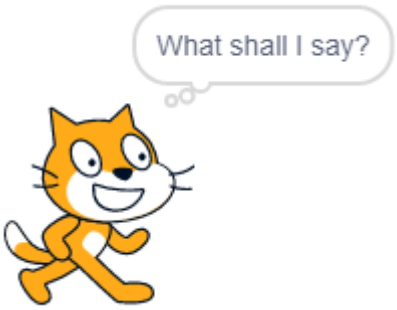
Code	Output
	

If you press the say block the message will be displayed all the time
Pressing the say "message" for x seconds will display the message for this specific x seconds

https://youtu.be/JTZ-XUkVNJM

Think block

You can also use the “Think” block instead of “Say” block, which makes the sprite display a thought bubble instead of a speech bubble.

Code	Ouput
	

If you press the think block the bubble will be displayed all the time. Pressing the think "message" for x seconds will display the bubble for this specific x seconds

https://youtu.be/RnrUrSS3TlQ

Switch Costume To block

It changes the current costume of a sprite to a different one. The sprite can have multiple costumes that can be switched between to create the illusion of animation or different appearances. The block allows you to specify which costume to switch to by selecting it from a drop-down menu or by using a variable. This block can be used to create simple animations or to change the appearance of a sprite based on certain conditions in a program.

In example the cat sprite has the cat-a and cat-b costumes.

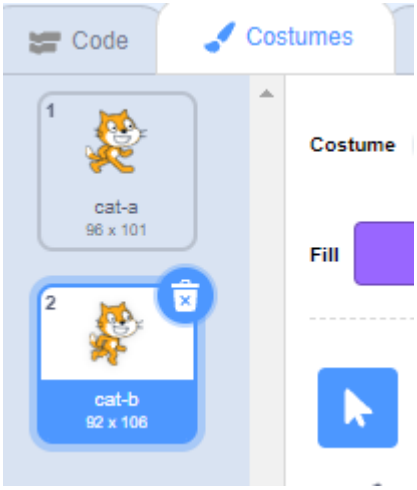
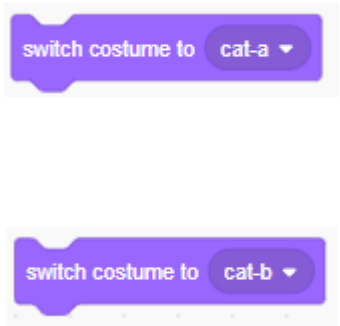
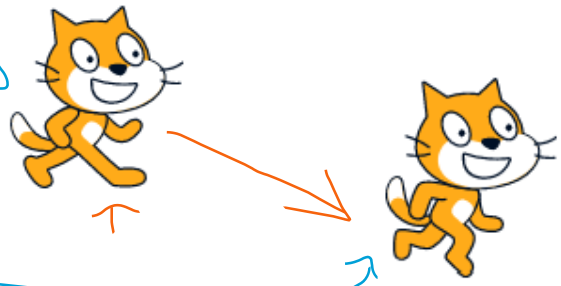


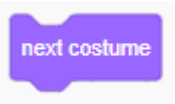
Figure 13: Cat Costumes

Code	Output
	

https://youtu.be/UHZJq2reGp0

Next Costume block

You can change the current costume of a sprite to the next one in the costume list. The sprite can have multiple costumes, so this block will change the sprite to the next costume in the list. If the current costume is the last one in the list, the block will switch to the first costume in the list, allowing you to create a looping animation. This block can be used in conjunction with other blocks such as "if-then" or "repeat" to create more complex animations or to change the appearance of a sprite based on certain conditions in a program.

Code	Output
Each time you press the next costume block the next costume of the sprite will be selected and displayed 	

https://youtu.be/1CAM_PKyEbI

Switch Backdrop To block

In Scratch, the "switch backdrop to [backdrop v]" block is a looks block that changes the current backdrop of the stage to a different one. A project can have multiple backdrops, so this block allows you to switch between them. The block allows you to specify which backdrop to switch to by selecting it from a drop-down menu or by using a variable. This block can be used to create animations or to change the backdrop of the stage based on certain conditions in a program. It allows you to change the background of the stage, creating the illusion of different scenes or locations in your program.

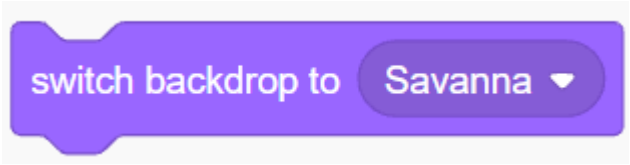


Figure 14: Switch Backdrop To block

Next Backdrop block

This block changes the current backdrop of the stage to the next one in the backdrop list. A project can have multiple backdrops, so this block allows you to switch to the next one in the list. If the current backdrop is the last one in the list, the block will switch to the first backdrop in the list, allowing you to create a looping animation. This block can be used in conjunction with other blocks such as "if-then" or "repeat" to create more complex animations or to change the backdrop of the stage based on certain conditions in a program. It allows you to change the background of the stage, creating the illusion of different scenes or locations in your program.

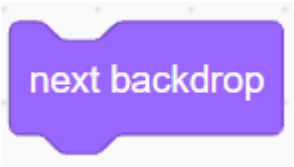
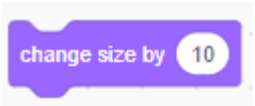
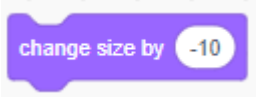
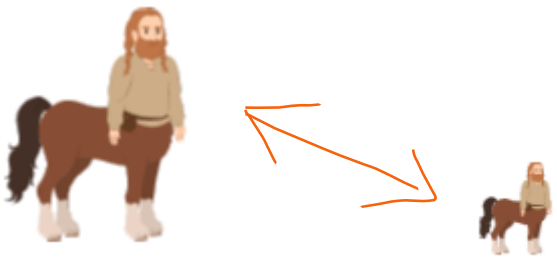


Figure 15: Next Backdrop block

https://youtu.be/49w-2gyMbC4

Change Size By block

The "change size by" block is a block that increases or decreases the size of a sprite by a specific value. The block takes a numeric value as an input, which can be positive to increase the size of the sprite or negative to decrease the size of the sprite. The block can be used to create animations or to change the size of a sprite based on certain conditions in a program. It can also be used in conjunction with other blocks such as "if-then" or "repeat" to create more complex interactions.

Code	Output
This will increase the size by 10%  This will decrease the size by 10%  	

https://youtu.be/3yqLgs6PZ1s

Set Size To block

In Scratch, the "set size to" block is a block that sets the size of a sprite to a specific value. The block takes a numeric value as an input, which represents the percentage size of the sprite, where 100% is the original size of the sprite and smaller value will decrease the size and bigger value will increase the size. The block can be used to create animations or to change the size of a sprite based on certain conditions in a program. It can also be used in conjunction with other blocks such as "if-then" or "repeat" to create more complex interactions.

https://youtu.be/qK92eHjaY7A

Chage Effects By block

Figure 16: Change Effects By block

In Scratch, the "change effect by" block is a Looks block that changes the value of a visual effect of a sprite by a specific amount. The visual effect options include "color", "fisheye", "whirl", "pixelate", "mosaic", "brightness" and "ghost". Each effect has a default value of 0. The block takes a numeric value as an input, which can be positive to increase the effect or negative to decrease the effect. The block can be used to create animations or to change the visual effects of a sprite based on certain conditions in a program. It can also be used in conjunction with other blocks such as "if-then" or "repeat" to create more complex interactions.

https://youtu.be/U_iqezjqeDM

Set Effect To block

The "set effect to" block is a Looks block that sets the value of a visual effect of a sprite to a specific amount. The visual effect options include "color", "fisheye", "whirl", "pixelate", "mosaic", "brightness" and "ghost". Each effect has a default value of 0. The block takes a numeric value as an input, which represents the level of the effect. The block can be used to create animations or to change the visual effects of a sprite based on certain conditions in a program. It can also be used in conjunction with other blocks such as "if-then" or "repeat" to create more complex interactions. The effects can be used to add special visual effects to a sprite, like making it look brighter or more colorful.

Figure 17: Set Effect To block

https://youtu.be/dE_p_0pkPdk

Clear Graphic Effects block

The "clear graphic effects" block is a block that clears all the visual effects applied to a sprite. It resets the visual effects of a sprite to their default values, effectively removing any changes made by the "change effect by" or "set effect to" blocks. This block can be used to remove the visual effects of a sprite at a specific point in a program, or to reset the effects to their default state after they have been changed. It can also be used in conjunction with other blocks such as "if-then" or "repeat" to create more complex interactions.

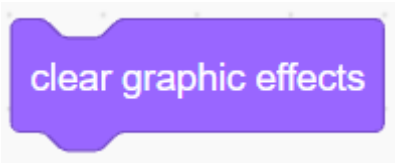


Figure 18: Clear Graphics Effects block

Show and Hide blocks

In Scratch, the "show" and "hide" blocks are looks blocks that control the visibility of a sprite. The "show" block makes a sprite visible on the stage, while the "hide" block makes a sprite invisible on the stage. They allow you to control the visibility of a sprite based on certain conditions in a program. For example, you can use the "hide" block to make a sprite disappear when the user clicks on it, and the "show" block to make the sprite reappear when a certain condition is met. These blocks can also be used to create animations or to create more complex interactions in a program.

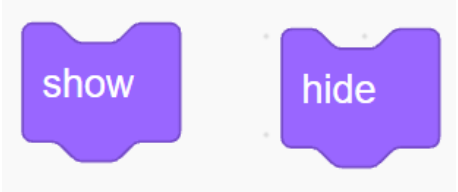


Figure 19: Show and Hide blocks

https://youtu.be/g1OoNBg4uV4

Go To Layer block

In Scratch, the "go to [layer v]" block is a looks block that changes the layer of a sprite on the stage. The sprite can be moved to the front or back layer of the stage, or moved up or down one layer at a time. The block allows you to specify which layer to switch to by selecting it from a drop-down menu or by using a variable. This block can be used to create animations or to change the layer of a sprite based on certain conditions in a program. It allows you to control the depth of the sprite on the stage, making it appear as if it is in front of or behind other sprites.

Figure 21: Go To front/Back Layer block

Go Forward/Backward layer block

This block allows the sprite to move on the stage in front or behind other sprites by a certain number of layers. The block takes two inputs, the first one being the direction in which the sprite should move, either forward or backward, and the second one is the number of layers the sprite should move by. This block can be used in conjunction with other blocks such as "if-then" or "repeat" to create more complex animations or to change the layer of a sprite based on certain conditions in a program. This block is useful for controlling the depth of the sprite on the stage, making it appear as if it is in front of or behind other sprites.

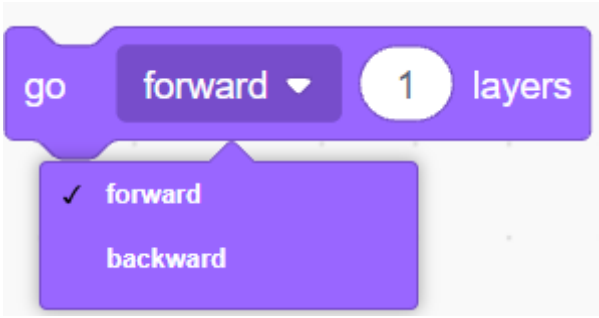


Figure 20: Go Forward/Backwards Layer block

III. The code blocks of the Sound Category

Play Sound Until Done
In Scratch, the "play sound [sound v] until done" block is a sound block that plays a sound from the project's library and waits for the sound to finish before continuing with the program. The block allows you to specify which sound to play by selecting it from a drop-down menu or by using a variable. Once the block is executed, the sound will start playing and the program will **not** continue until the sound has finished playing.

This block is useful when you want the program to wait for a sound to finish before proceeding with other instructions, for example, if you want to play a sound effect and then show a message on the screen, you would use the "play sound until done" block before the block that shows the message.

As with "start sound" the sound can be added to the project's library by clicking on the "Sounds" tab and then on the "Import" button, you can upload sounds from your computer or record them.

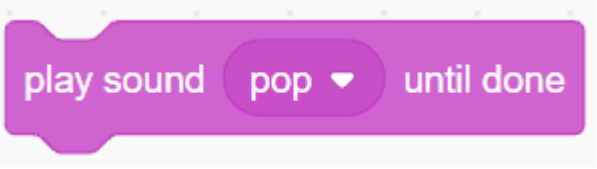


Figure 21: Play Sound Until Done block

Start Sound
In Scratch, the "start sound [sound v]" block is a sound block that plays a sound from the project's library. The block allows you to specify which sound to play by selecting it from a drop-down menu or by using a variable. Once the block is executed, the sound will start playing. It can be used to add sound effects to a game, or to play background music in an interactive story.

The sound can be added to the project's library by clicking on the "Sounds" tab and then on the "Import" button, you can upload sounds from your computer or record them.

It is important to note that the sound will play simultaneously with any other sounds that are currently playing, if you want to stop a sound you can use the "stop all sounds" block or the "stop sound [sound v]" block.

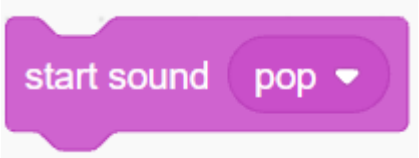


Figure 22: Start Sound block

https://youtu.be/iSRWlEMcK9k

The difference between Play Sound Until and Start Sound
The difference between the "play sound [sound v] until done" and "start sound [sound v]" blocks in Scratch is that the "play sound until done" block will wait for the sound to finish playing before continuing with the program, while the "start sound" block will start playing the sound and continue with the program immediately.

The "play sound until done" block is useful when you want the program to wait for a specific sound to finish before proceeding with other instructions, for example, if you want to play a sound effect and then show a message on the screen, you would use the "play sound until done" block before the block that shows the message, this way the program will wait for the sound to finish before showing the message.

On the other hand, the "start sound" block is useful when you want the sound to play simultaneously with other actions in the program. For example, if you want to play background music while the sprite is moving, you would use the "start sound" block at the beginning of the script and the sound will play while the sprite is moving.

Stop All Sounds block
In Scratch, the "stop all sounds" block is a sound block that stops all currently playing sounds. Once the block is executed, all sounds that are currently playing will be stopped immediately.

This block is useful when you want to stop all sounds that are currently playing in a program, for example, if you want to stop background music or sound effects when a game is over or when a user clicks a button, you would use the "stop all sounds" block in the script that is triggered by the button click or the end of the game.

It's important to note that the "stop all sounds" block will stop all sounds, including those that were started with the "start sound" and "play sound until done" blocks, so use it carefully. If you only want to stop a specific sound you can use the "stop sound [sound v]" block.

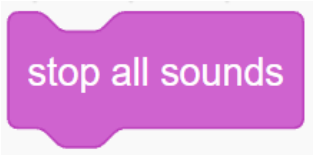


Figure 22: Stop All Sounds block

https://youtu.be/lEb5VWOwKT0

What is the pitch of sound (increasing and decreasing frequencies)
The pitch of a sound is a measure of how high or low it sounds. It is determined by the frequency of the sound waves that make up the sound. Higher-pitched sounds have a higher frequency, while lower-pitched sounds have a lower frequency.

The perception of pitch is closely related to the frequency of a sound wave. Humans can perceive a wide range of pitches, from very low to very high. The unit of measurement for frequency is hertz (Hz) which represents the number of oscillations or cycles per second of a sound wave.

Musical notes also have a pitch, they are associated with a specific frequency. For example, the note A4 (also known as A above middle C) has a frequency of 440 Hz.

In addition, the pitch can be affected by other factors such as the amplitude, the duration and the timbre of the sound.

In Scratch, the "change pitch by [value]" and "set pitch to [value]" blocks allow you to adjust the pitch of a sound, this can be useful to create different variations of a sound, to create special effects or to match the pitch with other elements in a project.

Change Pitch Effect By block

In Scratch, the "change pitch by [value]" block is a sound block that changes the pitch of a sound by a specified amount. This block can be found in the "Sound" category of the Scratch blocks palette. The block takes a numeric value as an input, which can be positive to increase the pitch of the sound or negative to decrease the pitch of the sound. This block can be used to create animations or to change the pitch of a sound based on certain conditions in a program.

The pitch of a sound is a measure of how high or low it sounds. Changing the pitch of a sound can make it sound higher or lower and can be used to create different variations of a sound or to create special effects.

It is important to note that this block will change the pitch of all sounds that are currently playing, so use it carefully. If you only want to change the pitch of a specific sound you can use the "set pitch to [value]" block.

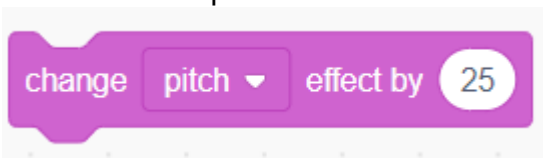


Figure 23: Change Pitch Effect By block

Change Pan Left/Right Effect By block

It that changes the stereo panning of a sound by a specified amount. The block takes a numeric value as an input, which can be positive to increase the panning towards the right or negative to decrease the panning towards the left. The range of this value is from -100 (fully left) to 100 (fully right). This block can be used to create animations or to change the stereo panning of a sound based on certain conditions in a program.

Stereo panning refers to the placement of a sound in the stereo field. It allows you to control where a sound is heard in the stereo field, and it can be used to create a sense of movement, depth, or direction in a mix. Panning a sound fully left or fully right will make it sound as if it's coming from the corresponding speaker, while panning it to the center will make it sound as if it's coming from both speakers.

It is important to note that this block will change the stereo panning of all sounds that are currently playing, so use it carefully. If you only want to change the stereo panning of a specific sound you can use the "set pan to [value]" block.

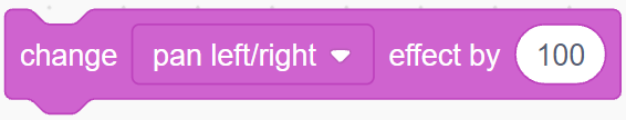


Figure 24: Change Pan Left/Right Effect by block

Set Pitch Effect To block

In Scratch, the "set pitch to [value]" block is a sound block that sets the pitch of a sound to a specific valueThe block takes a numeric value as an input, which can be any value between -100 and 100. This block can be used to create animations or to set the pitch of a sound based on certain conditions in a program.

The pitch of a sound is a measure of how high or low it sounds. Setting the pitch of a sound to a specific value can make it sound higher or lower and can be used to create different variations of a sound or to create special effects.

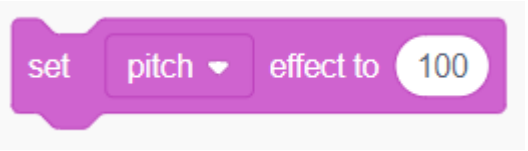


Figure 25: Set Pitch Effect To block

It is important to note that this block will set the pitch of all sounds that are currently playing, so use it carefully. If you only want to set the pitch of a specific sound you can use the "set pitch to [value]" block.

Clear Sound Effects

The "clear sound effects" block is a sound block that resets all the sound effects applied to a sound to their default values, effectively removing any changes made by the sound effects blocks such as "change pitch by" or "set pitch to" and "change pan left/right by". It can be used to remove the sound effects of a sound at a specific point in a program, or to reset the effects to their default state after they have been changed.

This block is useful when you want to remove all the sound effects that have been applied to a sound and restore it to its original state. It can be used in conjunction with other blocks such as "if-then" or "repeat" to create more complex interactions.

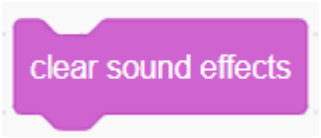


Figure 26: Clear Sound Effects block

Change Volume By block

In Scratch, the "change volume by [value]" block is a sound block that changes the volume of a sound by a specified amount. The block takes a numeric value as an input, which can be positive to increase the volume of the sound or negative to decrease the volume of the sound. The range of this value is from -100 (silent) to 100 (maximum volume). This block can be used to create animations or to change the volume of a sound based on certain conditions in a program.

The volume of a sound is a measure of how loud or quiet it sounds. Changing the volume of a sound can make it louder or quieter and can be used to create different variations of a sound or to create special effects.

It is important to note that this block will change the volume of all sounds that are currently playing, so use it carefully. If you only want to change the volume of a specific sound you can use the "set volume to [value]" block.

Set Volume To block

The "set volume to [value]" block is a sound block that sets the volume of a sound to a specific value. This block can be found in the "Sound" category of the Scratch blocks palette. The block takes a numeric value as an input, which can be any value between 0 and 100. This block can be used to create animations or to set the volume of a sound based on certain conditions in a program.

The volume of a sound is a measure of how loud or quiet it sounds. Setting the volume of a sound to a specific value can make it louder or quieter and can be used to create different variations of a sound or to create special effects.

It is important to note that this block will set the volume of all sounds that are currently playing, so use it carefully. If you only want to set the volume of a specific sound you can use the "set volume to [value]" block.

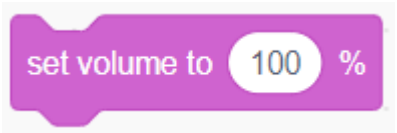


Figure 27: Set Volume To block

Difference between Pitch and Volume of a sound

Pitch and volume are two separate properties of a sound, and they are related to different aspects of the sound wave:

Pitch refers to the perceived highness or lowness of a sound and is determined by the frequency of the sound wave. Higher-pitched sounds have a higher frequency, while lower-pitched sounds have a lower frequency. Changing the pitch of a sound can make it sound higher or lower.

Volume, on the other hand, refers to the perceived loudness or softness of a sound and is determined by the amplitude of the sound wave. The amplitude is the height of the wave which is related to the energy of the wave and it determines the loudness of the sound. Higher amplitude means a louder sound, and lower amplitude means a softer sound. Changing the volume of a sound can make it louder or quieter.

In summary, pitch is related to how high or low a sound is, while volume is related to how loud or soft a sound is.

https://youtu.be/J3Dvm3RY8y8

IV. The code blocks of the Events Category

When Clicked block

In Scratch, the "when clicked" block is a control block that is used to specify a script or a set of commands that should be executed when the sprite is clicked on. When the script that contains this block is run, the script will wait until the sprite is clicked on before executing the commands inside the block. This block is used to create interactive programs where the user can interact with the sprite by clicking on it.

You can add blocks inside the "when clicked" block to specify the actions that should be performed when the sprite is clicked on, for example, you can make the sprite move, change its appearance, play a sound or display a message.

It's important to note that the "when clicked" block only responds to a left mouse button click, if you want the block to respond to a right mouse button click, you can use the "when right-clicked" block

Figure 28: When Clicke block

https://youtu.be/zXDiqacltjU

When <key> Key Pressed block

The "when [key] key pressed" block is a control block that is used to specify a script or a set of commands that should be executed when a specific key on the keyboard is pressed. When the script that contains this block is run, the script will wait until the specified key is pressed before executing the commands inside the block. This block is used to create interactive programs where the user can interact with the sprite by pressing a key on the keyboard.

You can add blocks inside the "when [key] key pressed" block to specify the actions that should be performed when the key is pressed, for example, you can make the sprite move, change its appearance, play a sound or display a message.

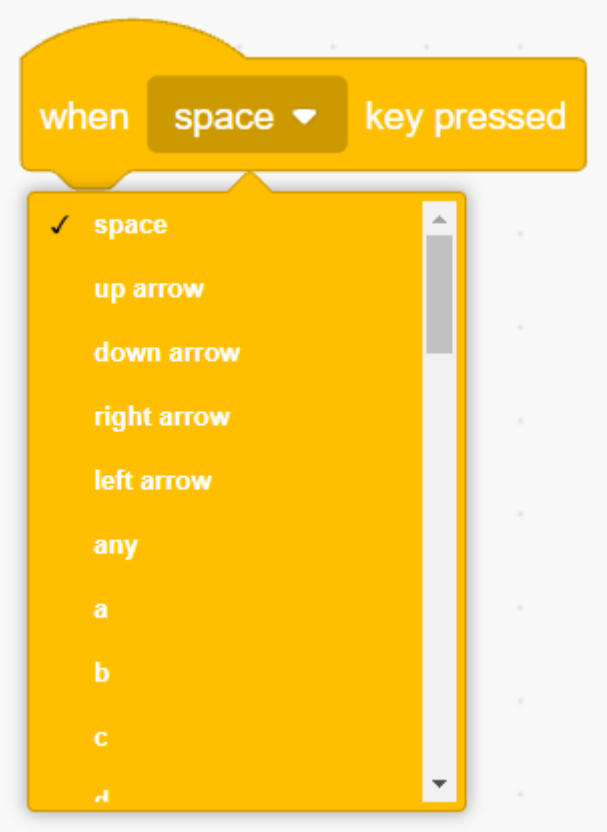


Figure 29: When Key Pressed block

This block allows you to specify which key you want to respond to by

clicking on the key button, this will open a drop-down menu where you can choose the key you want to use, you can choose from standard keys such as letters, numbers, space, arrow keys and special keys like shift, control and alt.

https://youtu.be/YuKXZaA9ok8

When This Sprite Clicked

It is used to specify a script or a set of commands that should be executed when the sprite that the script is attached to is clicked on. When the script that contains this block is run, the script will wait until the sprite is clicked on before executing the commands inside the block. This block is used to create interactive programs where the user can interact with the sprite by clicking on it.

You can add blocks inside the "when this sprite clicked" block to specify the actions that should be performed when the sprite is clicked on, for example, you can make the sprite move, change its appearance, play a sound or display a message.

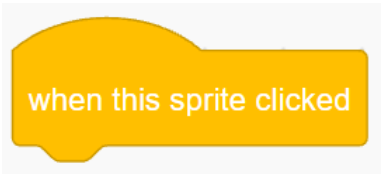


Figure 30: When This Script Clicked block

It's important to note that the "when this sprite clicked" block only responds to a left mouse button click, if you want the block to respond to a right mouse button click, you can use the "when this sprite right-clicked" block.

https://youtu.be/6_Gmgx7syjs

When Backdrop Switches To block

In Scratch, the "when backdrop switches to [backdrop v]" block is a control block that is used to specify a script or a set of commands that should be executed when the current backdrop changes to a specific backdrop. This block can be found in the "Events" category of the Scratch blocks palette. When the script that contains this block is run, the script will wait until the current backdrop changes to the specified backdrop before executing the commands inside the block.

You can add blocks inside the "when backdrop switches to [backdrop v]" block to specify the actions that should be performed when the specific

backdrop is shown, for example, you can make a sprite move, change its appearance, play a sound or display a message.

The backdrop can be selected from a drop-down menu which shows all the backdrops present in the project, this allows you to specify which backdrop should trigger the script, this way you can create different actions for different backdrops in your project.

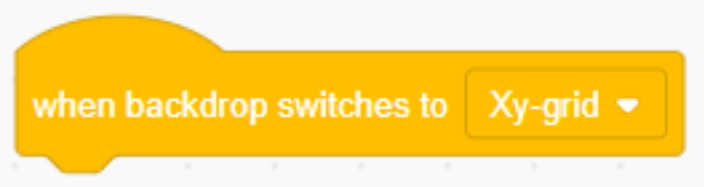


Figure 31: When Backdrop Switches To block

https://youtu.be/pMCf8pItgIM

When Loudness/Timer block

The "loudness" block is used to specify a script or a set of commands that should be executed when the loudness of a **microphone** input exceeds a specified threshold. When the script that contains this block is run, the script will constantly check the loudness of the microphone input, and when the loudness exceeds the specified threshold, the commands inside the block will be executed.

You can add blocks inside the "when loudness > [value]" block to specify the actions that should be performed when the loudness exceeds the threshold, for example, you can make a sprite move, change its appearance, play a sound or display a message.

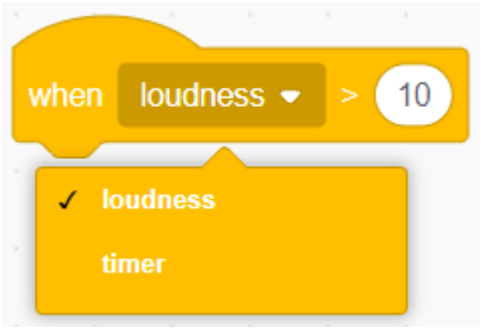


Figure 32: When Loudness/Timer > block

It's important to note that for using this block, you will need a microphone connected to your device and you should allow Scratch to access the microphone in your browser's settings. Also, the loudness value is a decimal number between 0 and 100.

The "when timer > [value]" block is a control block that is used to specify a script or a set of commands that should be executed when a specified amount of time has passed. When the script that contains this block is run, the script will wait for the specified amount of time to pass before executing the commands inside the block.

You can add blocks inside the "when timer > [value]" block to specify the actions that should be performed when the timer reaches the specified

value. For example, you can make a sprite move, change its appearance, play a sound or display a message.

It's important to note that the timer starts counting as soon as the script containing the "when timer > [value]" block starts running. You can start or reset the timer using the "reset timer" block in the control category. You can also use the "wait [value] secs" block to pause the script for a certain amount of time before executing the next block.

https://youtu.be/8Ew0_s24VVE

When I Receive Message block

In Scratch, the "when I receive [message]" block is a control block that is used to specify a script or a set of commands that should be executed when the sprite receives a specific message. When the script that contains this block is run, the script will wait until the sprite receives the specified message before executing the commands inside the block.

You can add blocks inside the "when I receive [message]" block to specify the actions that should be performed when the message is received, for example, you can make a sprite move, change its appearance, play a sound or display a message.

You can send a message to a sprite by using the "broadcast [message]" block or "broadcast [message] and wait" block, which can be found in the "Control" category of the Scratch blocks palette. These blocks allow you to send a message to all the sprites or a specific sprite in the project, you can use this feature to create more complex interactions between sprites.

Figure 33: When I receive Message block

https://youtu.be/yJTgDxZOBbQ

Broadcast Message block

This is a control block that is used to send a message to all sprites in a project. When this block is executed, it sends the specified message to all sprites in the project, and any scripts that are running on those sprites and contain a "when I receive [message]" block with the same message will be triggered.

The message that is sent is specified by the user and can be any text, it can be a variable, a string, or a predefined message. For example, you

could use the message "start" to signal the start of a game, or "score" to update the score of a game.

This feature allows you to create more complex interactions between sprites, for example, you can use the broadcast block to make one sprite control another sprite's behavior. You can also use the "broadcast [message] and wait" block which allows the script to wait until the message is received and acted upon before executing the next block.

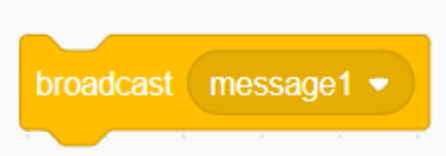


Figure 34: Broadcast Message block

Broadcast Message And Wait block

The "broadcast [message] and wait" block is a control block that is used to send a message to all the sprites in a project and wait for the scripts associated with that message to complete before continuing with the next block in the script. This block can be found in the "Control" category of the Scratch blocks palette.

When the script that contains this block is run, the message specified in the block will be broadcasted to all the sprites in the project and the script will pause execution until all the scripts associated with that message have completed their execution. This allows you to coordinate the actions of multiple sprites, by ensuring that all sprites have finished executing the script associated with the message before continuing with the next block.

You can use this block in combination with the "when I receive [message]" block to create more complex interactions between sprites, for example, you can use it to make all the sprites in the project move or change their appearance at the same time, or to play a sound simultaneously.

It's important to note that the "broadcast [message] and wait" block will pause the execution of the script containing it until all the other scripts associated with the message have completed, so use it carefully to avoid any performance issues.

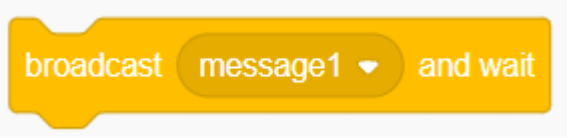


Figure 35: Broadcast Message And Wait block

V. The code blocks of the Control Category

Wait block
In Scratch, the "wait [value] secs" block is a control block that is used to pause the execution of a script for a specified amount of time. When the script that contains this block is run, the script will pause execution for the amount of time specified in the block before continuing with the next block.

You can use this block to create animations, delays, or to synchronize the actions of multiple sprites. For example, you can use it to make a sprite wait before moving to its next position or to make a sound wait before playing. You can also use it in conjunction with other blocks such as "repeat" or "if-then" to create more complex interactions.

The "wait [value] secs" block takes a numeric value as input which represents the time in seconds, this value can be a decimal number that allows you to specify a time with more precision. The time input can be a variable or a mathematical expression, you can also use the "pick random [a] to [b]" block to generate a random wait time.

https://youtu.be/OwgO32c1blE

https://youtu.be/pUMOqnBoAaM

Repeat block
The "repeat [value]" block is a control block that is used to repeat a set of commands a specified number of times. This block can be found in the "Control" category of the Scratch blocks palette. When the script that contains this block is run, the commands inside the block will be executed the number of times specified in the block before continuing with the next block.

You can add blocks inside the "repeat [value]" block to specify the actions that should be performed during each iteration of the loop, for example, you can make a sprite move, change its appearance, play a sound or display a message. The value input can be a variable or a mathematical expression.

The repeat block is useful for creating repetitive actions, for example, making a sprite move in a pattern, playing a sound multiple times, or displaying a message multiple times. You can also use it in conjunction with other blocks such as "wait" or "if-then" to create more complex interactions.

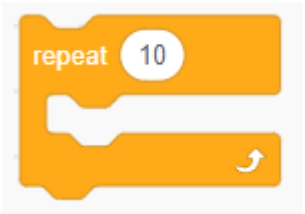


Figure 36: Repeat block

It's important to note that the repeat block will repeat the script inside it the number of times

specified in the input value, so use it carefully to avoid any performance issues.

https://youtu.be/wWBgrSVJ4WQ

Forever block
It is a control block that is used to repeat a set of commands indefinitely. When the script that contains this block is run, the commands inside the block will be executed repeatedly until the script is stopped or another control block such as "stop script" is used to exit the loop.

You can add blocks inside the "forever" block to specify the actions that should be performed during each iteration of the loop, for example, you can make a sprite move, change its appearance, play a sound or display a message.

The "forever" block is useful for creating actions that should repeat indefinitely, for example, making a sprite move in a pattern, playing a background music, or displaying a message continuously. You can also use it in conjunction with other blocks such as "wait" or "if-then" to create more complex interactions.

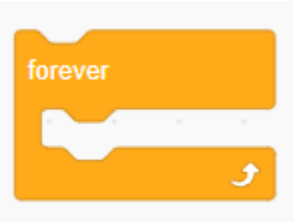


Figure 37: Forever block

It's important to use the "forever" block with caution as it will run indefinitely, it's a good practice to include a way to exit the loop, for example, using "if-then" block or a "stop script" block, otherwise the script will keep running even when you no longer need it.

https://youtu.be/q5ItzoeZ4xw

If Then block
In Scratch, the "if [condition] then" block is a control block that is used to specify a script or a set of commands that should be executed only if a certain condition is met. When the script that contains this block is run, the condition specified in the block will be checked and if the condition is true, the commands inside the block will be executed. If the condition is false, the commands inside the block will be skipped.

You can use the "if [condition] then" block in combination with other blocks to specify the condition you want to check, for example:

"touching [object]" block to check if the sprite is touching a specific object.
"key [key] pressed" block to check if a specific key is pressed
"loudness > [value]" block to check the loudness of the microphone input.
You can add blocks inside the "if [condition] then" block to specify the actions that should be performed when the condition is true, for example,

you can make a sprite move, change its appearance, play a sound or display a message.

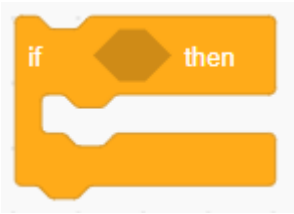


Figure 37: If Then block

It's important to note that the "if [condition] then" block only checks the condition once and if the condition is not met, the script inside the block will not be executed, so it's a good practice to use it in combination with a "forever" block or a "repeat" block to check the condition multiple times.

If Then Else block

The "if [condition] then else" block is a control block that is used to specify a script or a set of commands that should be executed only if a certain condition is met, and another script or set of commands that should be executed if the condition is not met.

When the script that contains this block is run, the condition specified in the "if [condition] then" block will be checked and if the condition is true, the commands inside the "if [condition] then" block will be executed. If the condition is false, the commands inside the "else" block will be executed.

The "if [condition] then else" block works in a similar way as the "if [condition] then" block, but allows you to specify an alternative action that should be taken in case the condition is not met.

You can use the "if [condition] then else" block in combination with other blocks to specify the condition you want to check, for example, you can use the "key [key] pressed" block to check if a specific key is pressed, the "loudness > [value]" block to check the loudness of the microphone input or "touching [object]" block to check if the sprite is touching a specific object

You can add blocks inside the "if [condition] then" block and the "else" block to specify the actions that should be performed when the condition is true or false respectively, for example, you can make a sprite move, change its appearance, play a sound or display a message.

Figure 35: If Then Else block

It's important to note that the "if [condition] then else" block only checks the condition once and if the condition is met the script inside the "if [condition] then" block will be executed and if the condition is not met the script inside the "else" block will be executed, so it's a good practice to use it in combination with a "forever" block or a "repeat" block to check the condition multiple times.

https://youtu.be/YoIp9gjiNBo

Wait until block
The "wait until [condition]" block is a control block in Scratch that is used to pause the execution of a script until a certain condition is met.

The condition specified in the block will be checked repeatedly until it becomes **true**, at which point the script will continue to execute the next block. This block is useful when you want to wait for a specific event or sensor reading before continuing with the script.

You can use the "wait until [condition]" block in combination with other blocks to specify the condition you want to check, for example:

"touching [object]" block to check if the sprite is touching a specific object.
"key [key] pressed" block to check if a specific key is pressed
"loudness > [value]" block to check the loudness of the microphone input.
You can add blocks after the "wait until [condition]" block to specify the actions that should be performed after the condition is met.

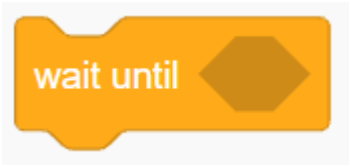


Figure 36: Wait Until block

https://youtu.be/msoTERozv8o

Repeat until block
The "repeat until [condition]" block is a control block in Scratch that is used to repeat a set of commands until a certain condition is met.

When the script that contains this block is run, the commands inside the block will be executed repeatedly until the condition specified in the block is met. Once the condition is met, the script will continue to execute the next block.

You can use the "repeat until [condition]" block in combination with other blocks to specify the condition you want to check, for example:

"touching [object]" block to check if the sprite is touching a specific object.
"key [key] pressed" block to check if a specific key is pressed
"loudness > [value]" block to check the loudness of the microphone input.

You can add blocks inside the "repeat until [condition]" block to specify the actions that should be performed while the condition is not met.

It's important to note that the "repeat until [condition]" block will repeat the commands inside the block repeatedly until the condition is met, so it's a good practice to use it in combination with a

Figure 37: Repeat Until block

"forever" block or a "repeat" block to limit the number of iterations

https://youtu.be/JiaLznNsReU

Stop all block

The "stop all" block is a control block in Scratch that is used to stop all scripts that are currently running in a project. This block can be found in the "Control" category of the Scratch blocks palette.

When the script that contains this block is run, all other scripts that are currently running in the project will be stopped immediately, regardless of where they are in the script or which sprite they belong to. This block can be used to stop all sounds, animations, and movements that are currently happening in the project.

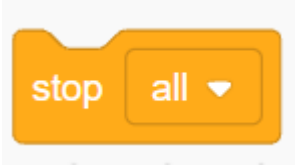


Figure 38: Stop All block

It's important to note that the "stop all" block will stop all running scripts in the project, so it's a good practice to use it only when necessary, as it will stop all other running scripts even the ones that you want to keep running.

https://youtu.be/J157zC7Cq4Q

When I Start As A Clone block

The "when I start as a clone" block is a control block in Scratch that is used to specify a script or a set of commands that should be executed when a clone of a sprite is created.

The "when I start as a clone" block is useful when you want to create multiple copies of a sprite and give them unique behavior. In Scratch, you can create a clone of a sprite by clicking on the sprite and then clicking the "duplicate" button or by using the "create clone of [myself]" block.

When a clone is created, the script that is associated with the "when I start as a clone" block will be executed.

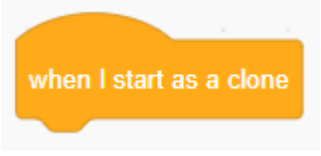


Figure 39: When I start As A block

It's important to note that when a clone is created, all the blocks inside the "when I start as a clone" block are executed, so it's a good practice to use it

only when necessary, as it will duplicate the script for each clone created.

Create Clone Of block

The "create clone of [myself]" block is a control block in Scratch that is used to create a copy of the sprite that the script is running on. This block can be found in the "Control" category of the Scratch blocks palette.

When the script that contains this block is run, a new sprite, identical to the original, will be created and added to the stage. Each clone can have its own scripts and behavior and can interact independently with other sprites and the stage.

It's important to note that when a clone is created, all the blocks inside the "when I start as a clone" block are executed, so it's a good practice to use it only when necessary, as it will duplicate the script for each clone created.

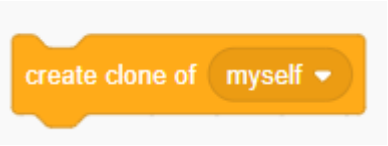


Figure 40: Create Clone Of block

Additionally, too many clones can slow down the project, so be mindful of how many clones you are creating.

Delete This Clone block

The "delete this clone" block is a control block in Scratch that is used to delete the clone of the sprite that the script is running on.

When the script that contains this block is run, the clone of the sprite that the script is running on will be deleted. The deletion is permanent, meaning that the sprite and its associated data will be removed and cannot be recovered.

It's important to note that when a clone is deleted, all the associated data such as variables and lists will also be deleted. Additionally, deleting too many clones too quickly can slow down the project, so be mindful of how many clones you are deleting and how often you are doing it.

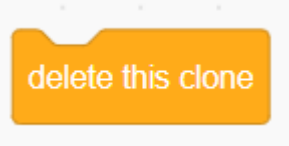


Figure 41: Delete This Clone block

https://youtu.be/98mkjS3Phak

VI. The code blocks of the Sensing Category

Scratch has several "sensing" blocks that can be used to gather information about the stage, other sprites, and the user's input.

Touching Mouse-Pointer/Edge block?

The "touching pointer" block is a sensing block in Scratch that checks if the sprite that the script is running on is currently touching the mouse pointer. It returns a Boolean value (true or false) depending on whether the sprite is touching the mouse pointer.

The "touching edge" block is another sensing block in Scratch that checks if the sprite that the script is running on is currently touching the edge of the stage. It returns a Boolean value (true or false) depending on whether the sprite is touching the edge of the stage.

You can use these blocks in combination with other blocks to create interactive programs. For example, you can use the "touching pointer" block in combination with the "say [message]" block to make a sprite say a specific message when it touches the mouse pointer. Or use the "touching edge" block in combination with the "bounce" block to make a sprite bounce when it touches the edge of the stage.

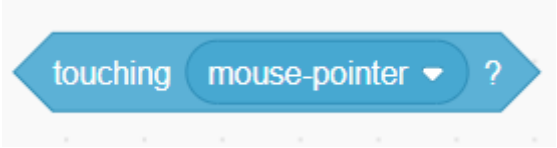


Figure 42: Touching Edge block

It's important to note that, unlike the "touching [object]" block, the "touching pointer" and "touching edge" blocks only check if the sprite is touching the mouse pointer or the edge of the stage respectively and don't take other sprites or objects into account.

https://youtu.be/XP2EiKB9BsE

Difference between If on Edge, Bounce block and Touching Edge block

The "if on edge, bounce" block and the "touching edge" block are both sensing blocks in Scratch, but they serve different purposes.

The "if on edge, bounce" block is a control block that checks if the sprite that the script is running on is currently touching the edge of the stage, and if so, it makes the sprite bounce off the edge. This block can be found in the "Control" category of the Scratch blocks palette. It contains a condition that is checked and based on the outcome, an action is taken.

On the other hand, the "touching edge" block is a sensing block that checks if the sprite that the script is running on is currently touching the edge of the stage. This block can be found in the "Sensing" category of the Scratch blocks palette. It returns a Boolean value (true or false) depending on whether the sprite is touching the edge of the stage.

In summary, the "if on edge, bounce" block checks if a sprite is touching the edge of the stage and if so, makes it bounce, while the "touching edge"

block simply checks if the sprite is touching the edge of the stage and returns a Boolean value, and it doesn't take any action on its own.

Both blocks can be useful in different situations and can be used in combination with other blocks to create interactive programs. For example, you can use the "touching edge" block in combination with the "if [condition]" block and "change costume" block to make a sprite change costume when it touches the edge of the stage.

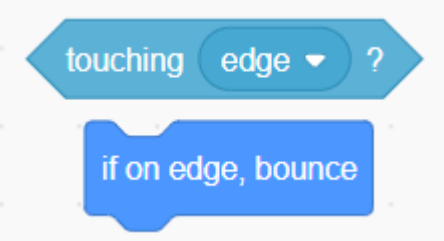


Figure 43: Touching Edge and If on Edge, Bounce block

https://youtu.be/1FTtyZjeTs0

Touching Color? block

The "touching color [color]" block is a sensing block in Scratch that checks if the sprite that the script is running on is currently touching a specific color. It returns a Boolean value (true or false) depending on whether the sprite is touching the specified color.

This block can be useful in different situations, for example, you can use it to detect when a sprite touches a specific colored object on the stage or to create interactive programs that respond to different colored backgrounds.

You can use the "touching color [color]" block in combination with other blocks to create interactive programs and games. For example, you can use the "touching color [color]" block in combination with the "say [message]" block to make a sprite say a specific message when it touches a certain color. Or use the "touching color [color]" block in combination with the "change costume" block to make a sprite change costume when it touches a certain color.

It's important to note that the "touching color" block only checks if the sprite is touching a certain color and doesn't take other sprites or objects into account. Additionally, the color that you specify in the block must be a color that is visible on the stage or background, otherwise, the block will always return false.

https://youtu.be/OSF8o18ItRE

Color is touching Color? block

The "color [color] is touching color [color]" block is a sensing block in Scratch that checks if a specific color is touching a certain other color. This block can be found in the "Sensing" category of the Scratch blocks palette. It returns a Boolean value (true or false) depending on whether the specified color is touching the other specified color.

This block can be useful in different situations, for example, you can use it to detect if a specific color on a sprite is touching a certain color on the background, or to check if a specific-colored object is touching another colored object on the stage.

You can use the "color [color] is touching color [color]" block in combination with other blocks to create interactive programs and games. For example, you can use the "color [color] is touching color [color]" block in combination with the "say [message]" block to make a sprite say a specific message when a certain color on a sprite is touching a certain color on the background. Or use the "color [color] is touching color [color]" block in combination with the "change costume" block to make a sprite change costume when a certain color on a sprite is touching a certain color on the stage.

It's important to note that the "color [color] is touching color [color]" block only checks if the specified color is touching the other specified color and doesn't take other sprites or objects into account. Additionally, the color that you specify in the block must be a color that is visible on the stage or background, otherwise, the block will always return false.

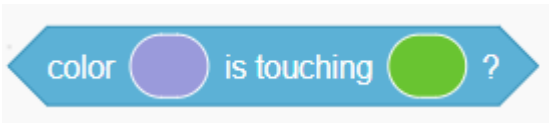


Figure 44: Color Is Touching Color block

https://youtu.be/Dbq19ZURrf4

Distance To block

The "distance to [object]" block is a sensing block in Scratch that returns the distance (in pixels) between the sprite that the script is running on and a specific object or sprite.

This block can be useful in different situations, for example, you can use it to:

- measure the distance between a sprite and a specific object or sprite make a sprite move towards or away from a specific object or sprite
- create interactive programs that respond to the distance between sprites

You can use the "distance to [object]" block in combination with other blocks to create interactive programs and games. For example, you can use the "distance to [object]" block in combination with the "move [value] steps" block to make a sprite move towards or away from a specific object or sprite. Or use the "distance to [object]" block in combination with the "if [condition]" block to make a sprite change costume when it gets close to a specific object or sprite.

It's important to note that the "distance to [object]" block only calculates the distance between the sprite and the specified object or sprite and doesn't take other sprites or objects into account.

Additionally, the object or sprite that you specify in the block must be on the stage, otherwise, the block will return an error.

https://youtu.be/liZZcxj8i58

Ask And Wait block

The "ask [question] and wait" block is a control block in Scratch that prompts the user to enter a response to a question, and then pauses the script until the user enters a response.

This block can be useful in different situations, for example, you can use it to:

- gather information from the user, such as a name or a choice
- create interactive programs that respond to user input
- create a dialogue system in your game or program

You can use the "ask [question] and wait" block in combination with other blocks to create interactive programs and games. For example, you can use the "ask [question] and wait" block in combination with the "say [message]" block to create a dialogue system in your game or program. Or use the "ask [question] and wait" block in combination with the "if [condition]" block to make a sprite change costume based on the user's input.

It's important to note that the script will pause and wait for the user to enter a response. The user's response will be stored in a special variable called "answer" that can be used in the script.

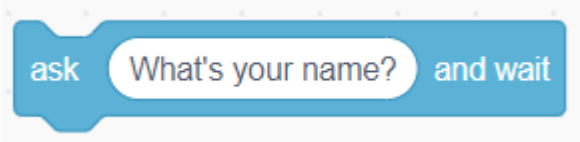


Figure 45: Ask And Wait block

https://youtu.be/04RZSWsxMcA

Answer block

The "answer" variable is a special variable in Scratch that holds the value of the user's response when the "ask [question] and wait" block is used. The value of the "answer" variable can be accessed using the "answer" block which can be found in the "Sensing" category of the Scratch blocks palette.

The "answer" variable can be used to:

- store user's response to a question
- use the stored response in further calculations or operations
- create interactive programs that respond to user input
- For example, you can use the "answer" block in combination with the "if [condition]" block to make a sprite change costume based on the user's input. Or use the "answer" block in combination with the "say [message]" block to make a sprite say a message that includes the user's response.

It's important to note that the value of the "answer" variable will only be set when the "ask [question] and wait" block is used and that the variable will be reset to empty after the script completes or when the program stops.

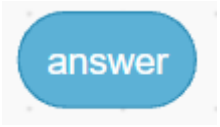


Figure 46: Answer block

Key Pressed? block

The "key [key] pressed?" block is a sensing block in Scratch that checks whether a specific key on the keyboard is currently being pressed. It returns a Boolean value (true or false) depending on whether the specified key is being pressed.

This block can be useful in different situations, for example, you can use it to:

- create interactive programs that respond to key presses
- create games that are controlled by the keyboard
- create interactive programs that require user input

You can use the "key [key] pressed?" block in combination with other blocks to create interactive programs and games. For example, you can use the "key [key] pressed?" block in combination with the "move [value] steps" block to make a sprite move when a specific key is pressed. Or use the "key [key] pressed?" block in combination with the "if [condition]" block to make a sprite change costume when a specific key is pressed.

Figure 47: Key <key> Pressed? block

It's important to note that the key pressed can be a letter or a symbol, depending on what you want to use it for.

Difference between When <key> Key Pressed and Key <key> Pressed? block

The "when key [key] pressed" block and the "key [key] pressed?" block are both sensing blocks in Scratch, but they serve different purposes.

The "when key [key] pressed" block is a control block that runs a script when a specific key on the keyboard is pressed. This block can be found in the "Control" category of the Scratch blocks palette. It contains a trigger event that runs the script when the key is pressed.

On the other hand, the "key [key] pressed?" block is a sensing block that checks if a specific key on the keyboard is currently being pressed. This block can be found in the "Sensing" category of the Scratch blocks palette. It returns a Boolean value (true or false) depending on whether the specified key is being pressed.

In summary, the "when key [key] pressed" block runs a script when a specific key is pressed and doesn't return a value, while the "key [key] pressed?" block simply checks if a specific key is pressed and returns a Boolean value (true or false)

Both blocks can be useful in different situations, depending on your needs. The "when key [key] pressed" block is useful for creating interactive programs that respond to key presses, while the "key [key] pressed?" block is useful for creating programs that require user input or for creating games that are controlled by the keyboard.

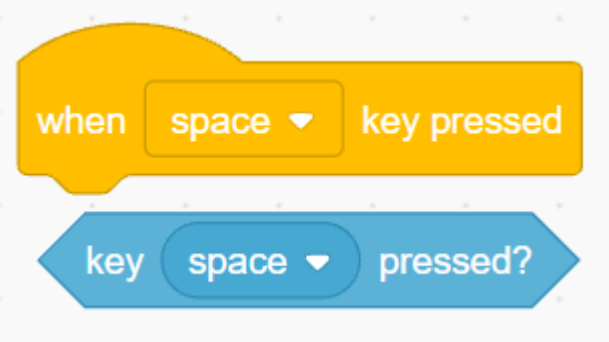


Figure 48: When <key> Key Pressed and Key <key> Pressed? block

Mouse Down? block

The "mouse down?" block is a sensing block in Scratch that checks if the left mouse button is currently being pressed. This block can be found in the "Sensing" category of the Scratch blocks palette. It returns a Boolean value (true or false) depending on whether the mouse button is being pressed.

This block can be useful in different situations, for example, you can use it to:

- create interactive programs that respond to mouse clicks
- create games that are controlled by the mouse
- create interactive programs that require user input

You can use the "mouse down?" block in combination with other blocks to create interactive programs and games. For example, you can use the "mouse down?" block in combination with the "move [value] steps" block to make a sprite move when the mouse button is pressed. Or use the "mouse down?" block in combination with the "if [condition]" block to make a sprite change costume when the mouse button is pressed.

It's important to note that the "mouse down?" block only checks for the left mouse button, if you want to check for the right mouse button you should use the "mouse right down?" block.

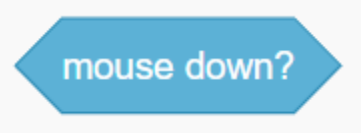


Figure 49: Mouse Down? block

Mouse X and Y blocks

The "mouse x" and "mouse y" blocks are sensing blocks in Scratch that return the current x and y position of the mouse pointer respectively. These blocks can be found in the "Sensing" category of the Scratch blocks palette.

The "mouse x" block returns the horizontal position of the mouse pointer on the stage, where the left edge of the stage has an x-coordinate of 0 and the right edge has an x-coordinate of the width of the stage.

The "mouse y" block returns the vertical position of the mouse pointer on the stage, where the top edge of the stage has a y-coordinate of 0 and the bottom edge has a y-coordinate of the height of the stage.

These blocks can be useful in different situations, for example, you can use it to:

- create interactive programs that respond to the position of the mouse pointer
- create games that are controlled by the mouse
- create interactive programs that require user input

You can use the "mouse x" and "mouse y" blocks in combination with other blocks to create interactive programs and games. For example, you can use the "mouse x" and "mouse y" blocks in combination with the "move [value] steps" block to make a sprite move towards the mouse pointer. Or use the "mouse x" and "mouse y" blocks in combination with the "if [condition]" block to make a sprite change costume when the mouse pointer is in a specific area of the stage.

Figure 50: Mouse X and Mouse Y blocks

Set Drag Mode Draggable/Not Draggable block
The "set drag mode [drag mode]" block is a block in Scratch that allows you to control how a sprite behaves when it is dragged by the mouse in the full screen mode. In the editing mode you can move it around as much as you want.

The "set drag mode" block has two options: "draggable" and "not draggable".

- "draggable" : when you set a sprite to be draggable, you can drag it around the stage with the mouse.
- "not draggable" : when you set a sprite to be not draggable, you can't drag it around the stage with the mouse.

For example, you can use the "set drag mode [draggable]" block to make a sprite draggable, so that the user can move it around the stage with the mouse. Or use the "set drag mode [not draggable]" block to make a sprite not draggable, so that the user can't move it around the stage with the mouse.

You can use this block to make specific sprites interactive and let your user to interact with them in a way that suits your project.
You can also use it in combination with other blocks such as "if [condition]" or "when [event]" to create more sophisticated programs that respond to the user's interactions with the sprites.

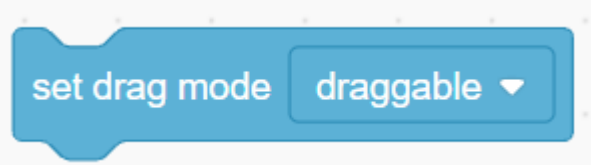


Figure 51: Set Drag Mode block

Loudness block
The "loudness" block is a sensing block in Scratch that returns the current loudness (volume) of the microphone.

The loudness block returns a value that represents the loudness of the sound currently being picked up by the microphone, measured in decibels (dB). The loudness value ranges from 0 to 100, where 0 represents no sound and 100 represents the maximum loudness that the microphone can pick up.

You can use the "loudness" block to create interactive programs that respond to the loudness of the sound.

For example, you can use the "loudness" block in combination with the "if [condition]" block to make a sprite change costume when the loudness exceeds a certain threshold.
Or use the "loudness" block in combination with the "set size to [value]" block to make a sprite change size based on the loudness of the sound.

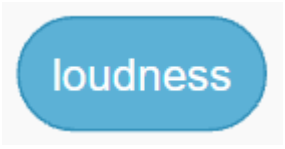


Figure 52: Loudness block

It's important to note that, in order to use the "loudness" block, the user's device must have a microphone and it must be enabled in the Scratch project settings.

Also, you should be aware of the sensitivity of the microphone and adjust your program accordingly to avoid false positives.

https://youtu.be/FaoOYmzKZPg

Timer block
The "timer" block is a sensing block in Scratch that returns the number of seconds that have elapsed since the project was started or since the timer was reset.

When a Scratch project is started, the timer is set to 0 and starts counting up in seconds. The timer block returns the current value of the timer.

You can use the "timer" block to create programs that are timed, such as games that have a time limit, or animations that play for a specific duration.

For example, you can use the "timer" block in combination with the "if [condition]" block to make a sprite change costume when the timer reaches a certain threshold.
Or use the "timer" block in combination with the "wait [seconds]" block to make a sprite wait for a specific amount of time before continuing with the script.

You can also use the "reset timer" block to reset the timer to 0, so the timer starts counting again from 0.

It's important to note that, the timer block only counts the time elapsed since the project has been started, if you want to measure time intervals or durations within the project, you may want to use the "wait" or "wait until" block in combination with the "current [time v]" or "time and date" block, or you can use variables to store the time at the start of an interval and compare it to the current time later on.

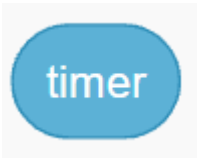


Figure 53: Timer block

Reset Timer block
The "reset timer" block is a control block in Scratch that sets the timer to 0, it can be found in the "Control" category of the Scratch blocks palette.
When you use the "reset timer" block, it sets the timer back to 0, and starts counting up again.

You can use the "reset timer" block to reset the timer at a specific point in your program, for example, to start counting the time again after a specific event occurs.

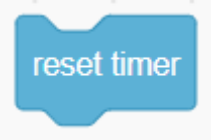


Figure 54: Reset Timer block

Backdrop#, Backdrop Name, Volume and My Variable of Stage block

The **"backdrop of [object]"** block is a sensing block in Scratch that returns the current backdrop of a specified object. This block can be found in the "Sensing" category of the Scratch blocks palette.

The "backdrop of [object]" block allows you to access information about the current backdrop of a specific object, such as a sprite or the stage. The block takes an object as an input, such as a sprite or the stage, and returns the number of the current backdrop.

You can use this block to create programs that respond to the current backdrop of a specific object. For example, you can use the "backdrop of [object]" block in combination with the "if [condition]" block to make a sprite change costume when the backdrop of the stage is a specific number.

Or use the "backdrop of [object]" block in combination with the "when [event]" block to make the sprite react to a change in the backdrop of the stage or another sprite.

It's important to note that, the "backdrop of [object]" block can only be used with objects that have backdrops, such as the stage or sprites that have backdrops assigned to them.

Also, the number of the backdrop starts at 1 and increases as you add more backdrops to your project, so the first backdrop you add will be number 1, the second one will be number 2 and so on.

The **"backdrop name of [stage]"** block is a sensing block in Scratch that returns the name of the current backdrop of the stage. This block can be found in the "Sensing" category of the Scratch blocks palette.

The "backdrop name of [stage]" block allows you to access the name of the current backdrop of the stage. The block takes no inputs and returns a string with the name of the current backdrop.

You can use this block to create programs that respond to the current backdrop of the stage. For example, you can use the "backdrop name of [stage]" block in combination with the "if [condition]" block to make a sprite change costume when the backdrop of the stage is a specific named backdrop.

Or use the "backdrop name of [stage]" block in combination with the "when [event]" block to make the sprite react to a change in the backdrop of the stage.

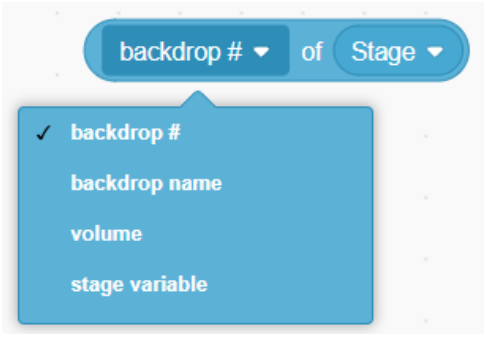


Figure 55: Properties Of Stage block

It's important to note that, when you first start a project, the name of the backdrop is set to "backdrop1" but you can change the name of the backdrop in the backdrops editor by clicking on the backdrop, and then

clicking on the "info" button and then changing the name to whatever you want.

Also, The backdrop name block is case-sensitive, so "backdrop1" and "Backdrop1" are considered different names.
It's a good practice to give meaningful names to your backdrops to make your code more readable and easy to maintain.

The **"volume of [stage]"** block displays the volume level of the backdrop.

The **"stage variable of [stage]"** block displays the value of a specific variable of the backdrop.

Current Date and Time block
In Scratch, there is a specific "current year" block that returns the current year.

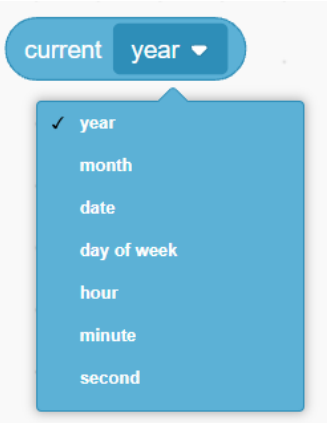


Figure 56: Current Date and Time block

Days since 2000 block
It displays the days past from 1-1-2000 until today

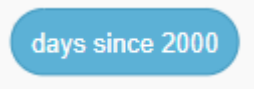


Figure 57: Days Since 2000 block

Username block
It displays the username only of the online editor

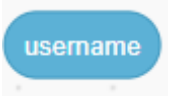


Figure 58: Username block

VII. The code blocks of the Operators Category

The "Operators" category in Scratch is a group of blocks that are used to perform mathematical and logical operations on input values. These blocks can be used to compare values, perform calculations, and make decisions based on the results of those operations.

The blocks in the "Operators" category include:

Mathematical Operators blocks
"() + ()", "-", "*", "/" : These blocks are used for mathematical operations such as addition, subtraction, multiplication, and division.

Figure 59: Mathematical Operators

https://youtu.be/xh3NLdWHSw0

Pick Random block
It is used to select a random number between a given range. The block takes two inputs, a starting and an ending number. It outputs a random number between the starting and ending number inputs.

For example, if the starting number is 1 and ending number is 6, the output will be a random number between 1 and 6 (1, 2, 3, 4, 5, or 6). This block can be used to generate random numbers for various purposes, such as generating random numbers for games, simulations, or to create randomness in the program. It can be used to control the loops, conditions, or make certain decisions based on the random number generated.

"pick random () to ()" : This block is used for picking a random number between two numbers

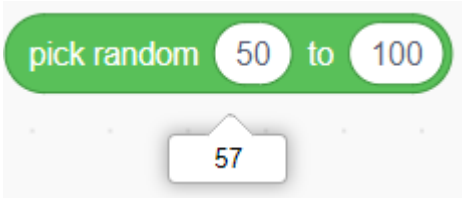


Figure 60: Pick Random block

https://youtu.be/pYhpubRrnf4

Relational Operators
The operator allows you to compare the relationship between two values or variables. It will compare the two and decide whether it is true or false.

"() > ()", "<", "=": These blocks are used for comparison operations such as greater than, less than and equal to.

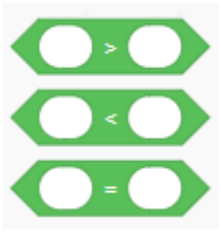


Figure 61: Relational Operator

https://youtu.be/Pf__RBamQ-I

Logical Operators
You can combine two or more relational operators to produce a single true/false result. Using these blocks allow you to further refine your comparison of values.

"and", "or" : These blocks are used for logical operations such as "and" and "or".

"not" : This block is used for negating a logical statement.

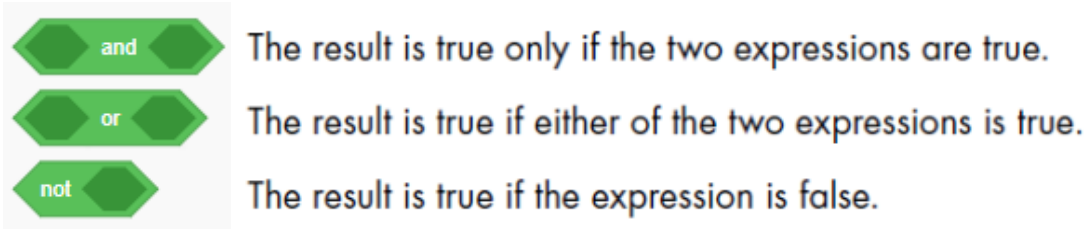


Figure 62: Logical Operators

https://youtu.be/UWjHNv7hwTY

Join block
It is used to combine two text inputs into one. It takes two text inputs as input, and outputs a single text input that is the combination of the two input texts.
"join () ()" : This block is used for concatenating two text inputs

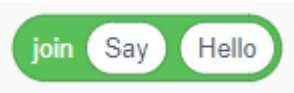


Figure 62: Join block

Letter Of block
It is used to extract a specific letter from a given string of text. The block takes two inputs: the string of text, and the position of the letter you want to extract (for example, 1 for the first letter, 2 for the second letter, and so on). The block will output the letter at the specified position in the string of text.

"letter () of ()" : This block is used for finding the letter at a specific position of a text input

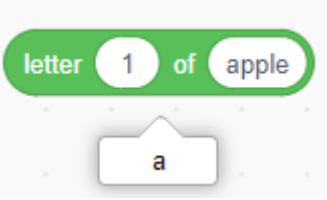


Figure 63: Letter Of block

Length Of block
It is used to determine the number of characters in each string of text. The block takes one input, which is the string of text you want to find the length of. It outputs a number, which represents the number of characters in the input string. This block can be used to determine the number of characters in a word, sentence, or a string of text. It can be used to control the loops, conditions, or make certain decisions based on the length of the text.

"length of ()" : This block is used for finding the length of a text input

Figure 64: Length Of block

Contains? block
It checks whether a given string of text contains a specified substring or not. It takes two inputs: the string of text to check, and the substring to look for. It outputs a Boolean value, either "true" if the substring is found in the text or "false" if it is not.

For example, if the input text is "Hello World" and the substring to look for is "orld", the output will be "true" because the text "Hello World" contains the substring "orld". This block can be used to check if a specific word or phrase is present in a given text, and can be used to control the loops, conditions, or make certain decisions based on the outcome.

Figure 64: Contains? block

https://youtu.be/UySrcyjqROc

Mod block

It is used to find the remainder of a division of two numbers. The block takes two inputs, which are the numbers you want to use in the division. It outputs a number, which is the remainder of the division of the two input numbers.

For example, if the input numbers are 10 and 3, the output will be 1 because 10 divided by 3 is 3 with a remainder of 1. This block can be used to perform various mathematical operations, such as determining if a number is even or odd, or to control the loops and conditions of the program.

"() mod ()" : This block is used for finding the remainder of a division between two numbers

Figure 65: Mod block

Round block

It is used to round a number to the nearest whole number. The block takes one input, which is the number you want to round. It outputs a number, which is the nearest whole number to the input number.

For example, if the input number is 3.6, the output will be 4 because 4 is the nearest whole number to 3.6. If the input number is 3.4, the output will be 3 because 3 is the nearest whole number to 3.4. This block can be used to perform various mathematical operations, such as rounding decimal numbers to whole numbers or make certain decisions based on rounding the input number.

"round ()" : This block is used for rounding a decimal number to the nearest whole number

Figure 66: Round block

Abs of block

It is used to find the absolute value of a number. The block takes one input, which is the number you want to find the absolute value of. It outputs a number, which is the absolute value of the input number. The absolute value of a number is its distance from 0 on the number line, so it is always a positive number.

For example, if the input number is -3, the output will be 3 because the absolute value of -3 is 3. If the input number is 5, the output will be 5 because the absolute value of 5 is still 5. This block can be used to perform various mathematical operations, such as finding the absolute distance between two numbers

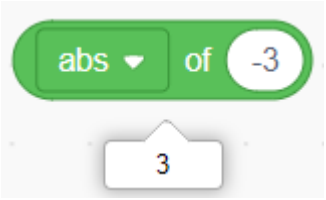


Figure 67: Abs Of block

or to make certain decisions based on the absolute value of the input number.

Floor of block

It is used to round a number down to the nearest whole number. The block takes one input, which is the number you want to round down. It outputs a number, which is the nearest whole number that is less than or equal to the input number.

For example, if the input number is 3.6, the output will be 3 because 3 is the nearest whole number that is less than or equal to 3.6. If the input number is 3.4, the output will also be 3 because 3 is the nearest whole number that is less than or equal to 3.4. This block can be used to perform various mathematical operations, such as rounding decimal numbers down to whole numbers or make certain decisions based on the rounded down number.

Figure 68: Floor Of block

Ceiling Of block

It is used to round a number up to the nearest whole number. The block takes one input, which is the number you want to round up. It outputs a number, which is the nearest whole number that is greater than or equal to the input number.

For example, if the input number is 3.6, the output will be 4 because 4 is the nearest whole number that is greater than or equal to 3.6. If the input number is 3.4, the output will also be 4 because 4 is the nearest whole number that is greater than or equal to 3.4. This block can be used to perform various mathematical operations, such as rounding decimal numbers up to whole numbers or make certain decisions based on the rounded up number.

Figure 69: Ceiling Of block

https://youtu.be/wbqSs9yAOeg

VIII. The code blocks of the Variables Category

In Scratch 3.0, the "Variables" category includes blocks that are used to create, store, and manipulate variables. Variables are used to store values or data in a program, and can be used to keep track of important information or to make decisions based on the values stored in the variables.

The blocks in the "Variables" category include:

Make a variable

You can create your own variables to store and manipulate data in your program. You can create a variable by using the "Make a Variable" option in the "Variable" category and name it according to your needs, Once you have created your variable, you can use the various blocks in the "Variables" category to manipulate the value of your variable.

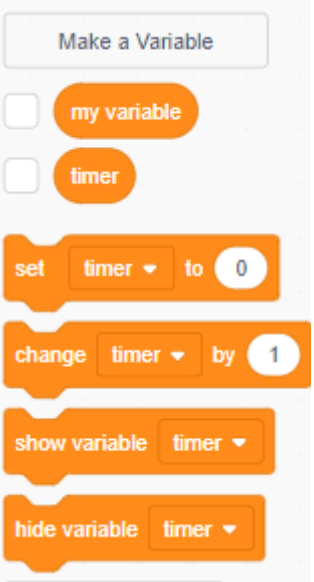


Figure 70: Various Variables blocks

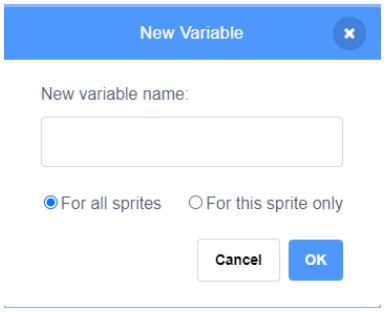


Figure 71: Make A Variable Name choice

For example, you can use the "set [variable] to [value]" block to assign a value to your variable, and then use the "change [variable] by [value]" block to increase or decrease the value of your variable. You can also use the "[variable]" block to reference the value stored in your variable in other blocks or operations.

You can also use the "show variable [variable]" block to display the value of your variable on the stage, and "hide variable [variable]" block to hide the value of your variable from the stage.

Set <variable> To <value> block

It is used to create and assign a value to a variable. The block takes two inputs, the first input is the variable you want to create or change, and the second input is the value you want to assign to the variable.

For example, you can use the block to set the variable "score" to 0 at the start of a game, and then use the "change [variable] by [value]" block to increase the value of "score" as the player progresses through the game. Or you can set the variable "age" to 10, and then use that variable in the conditions or loops.

It's important to note that when you use this block to assign a value to a variable that already exists, it will overwrite the previous value of the variable with the new value.

Keep in mind that the value you want to assign to the variable must be of the same type as the variable. For example if the variable is a number, the value must be a number and so on.

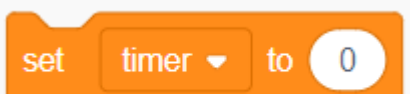


Figure 72: Set <variable> To <value> block

Change <variable> By <value> block
It is a block that is used in the "Variables" category. It is used to increase or decrease the value of a variable by a specified amount. The block takes two inputs, the first input is the variable you want to change and the second input is the value you want to add or subtract from the current value of the variable.

For example, you can use this block to increase the value of a "score" variable by 10 for each time the player answers a question correctly in a game. Or you can use this block to decrease the value of a "lives" variable by 1 each time the player loses a life.

If the input value is positive, it will add the value to the current value of the variable, if the input value is negative, it will subtract the value from the current value of the variable.

It's important to note that this block only works on variables that have already been created and have a value assigned to them. It will not create a new variable or assign a value to a non-existing variable.

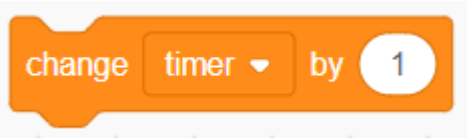


Figure 73: Change <variable> By <value> block

Show Variable block
The "show variable [variable]" block is a block that is used in the "Variables" category. It is used to display the value of a variable on the stage. The block takes one input, which is the variable you want to display.

When the block is executed, the value of the specified variable will be shown on the stage, typically as a text label. You can also customize the appearance of the variable value using the "Looks" blocks. You can change the color, size, font, etc.

For example, you can use this block to display the value of a "score" variable on the stage, so that players can see their current score as they play a game. Or you can use this block to display the value of a "time" variable so that players can see how much time they have left in a game.

It's important to note that this block only works on variables that have already been created and have a value assigned to them. It will not create a new variable or assign a value to a non-existing variable.

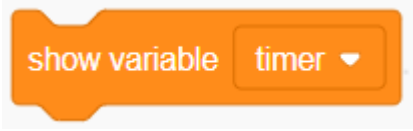


Figure 74: Show Variable block

Hide Variable block
The "hide variable [variable]" block is a block that is used in the "Variables" category. It is used to hide the value of a variable from the stage. The block takes one input, which is the variable you want to hide.

When the block is executed, the value of the specified variable will be hidden from the stage, typically as a text label. This block can be used to hide the variable value when it is no longer needed or when the game or program logic requires it.

For example, you can use this block to hide the value of a "lives" variable from the stage once the player runs out of lives in a game, or you can use this block to hide the value of a "password" variable after it's been entered correctly.

It's important to note that this block only works on variables that have already been created and have a value assigned to them. It will not create a new variable or assign a value to a non-existing variable. Also, once the variable is hidden, it will no longer be visible on the stage and the value will not be updated until it's shown again.

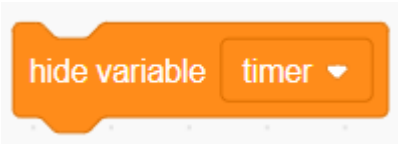


Figure 75: Hide Variable block

https://youtu.be/UySrcyjqROc

List

Make A List

In Scratch 3.0, the "Make a List" block is a block that is used in the "Variables" category. It is used to create a new list and give it a name. A list is a collection of items, similar to an array in other programming languages. These items can be numbers, text, or other variables.

Once you have created a list, you can use the various blocks in the "List" category to manipulate the items in the list. For example, you can use the "add [item] to [list]" block to add an item to the list, or the "delete [item] of [list]" block to remove an item from the list. You can also use the "item [number] of [list]" block to reference a specific item in the list.

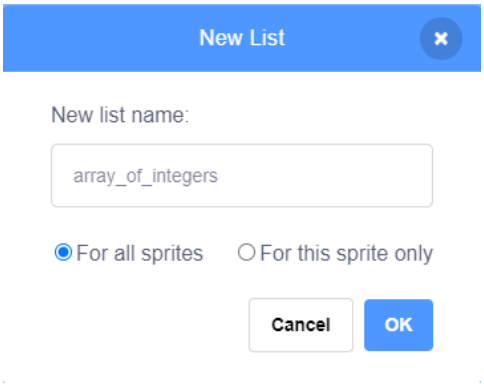


Figure 76: Make A list choice

You can use lists to store collections of data, such as a list of high scores for a game, a list of students in a class, or a list of items in a shopping cart.

When you create a list, it will be empty, you can add items to it later on. Keep in mind that it's important to choose meaningful and descriptive names for your lists, so it's easy to understand the purpose and usage of the list throughout your program.

- List items are indexed, the first item has index [1], the second item has index [2] etc.
- You can access them by referring to the index number.
- To change the value of a specific item, refer to the index number

Figure 77: Empty List

Add <item> To List

In Scratch 3.0, the "add [item] to [list]" block is a block that is used in the "Variables" category. It is used to add an item to an existing list. The block takes two inputs, the first input is the item you want to add to the list and the second input is the list where you want to add the item.

When the block is executed, the item you specified will be added to the end of the list. Keep in mind that the list can contain different types of items, like numbers, text, or variables.

For example, you can use this block to add a new high score to a list of high scores in a game, or you can use this block to add a new student to a list of students in a class, or you can use this block to add a new item to a shopping cart.

It's important to note that this block only works on lists that have already been created. It will not create a new list or add an item to a non-existing list.

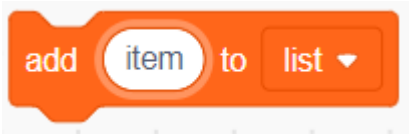


Figure 78: Add <item> To List block

Delete <index> From List block
It is used to remove an item from an existing list. The block takes two inputs, the first input is the index which points to the item in the list you want to remove and the second input is the list from which you want to remove the item.

When the block is executed, the item you specified will be removed from the list. Keep in mind that the list can contain different types of items, like numbers, text, or variables.

For example, you can use this block to remove an item from a shopping cart, or you can use this block to remove a student from a list of students in a class, or you can use this block to remove a high score from a list of high scores in a game.

It's important to note that this block only works on lists that have already been created and have items in it. It will not remove an item from a non-existing list or an empty list.

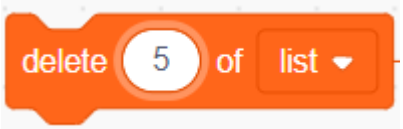


Figure 79: Delete <index> From List block

Delete All Of List block
It is used to remove all items from an existing list. The block takes one input, which is the list from which you want to remove all items.

When the block is executed, all items in the list will be removed, effectively making the list empty.

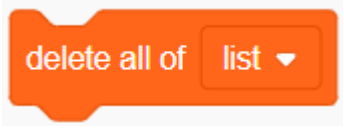


Figure 80: Delete All Of List block

https://youtu.be/JSh2SAZCJTQ

Insert <item> at <index> Of List block
It is used to insert an item at a specific position in an existing list. The block takes three inputs, the first input is the item you want to insert, the second input is the index (position) at which you want to insert the item, and the third input is the list where you want to insert the item.

When the block is executed, the item you specified will be inserted at the specified position in the list. Keep in mind that the list can contain different types of items, like numbers, text, or variables.

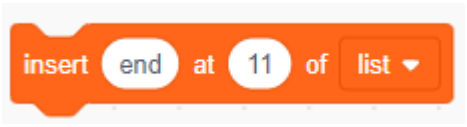


Figure 81: Insert <item> At <index> Of List block

It's important to note that this block only works on lists that have already been created, it will not insert an item in a non-existing list or at an invalid position. Also, when you insert an item at a position, all the items that were in that position and after it will be shifted one position to the right.

https://youtu.be/3oUgORRN5VA

Replace Item <index> Of List with <item> block

It is used to replace an item at a specific index position in a existing list with a new value. The block takes two inputs, the first input is the index number of the item you want to access, and the second input is the item which you want to insert in the list and replace the previous one at the same position.

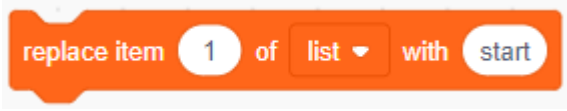


Figure 82: Replace Item <index> of List with <item> block

Item <index> Of List block

It is used to access a specific index in an existing list. The block takes two inputs, the first input is the index number of the item you want to access, and the second input is the list from which you want to access the item.

When the block is executed, it will return the item at the specified index position in the list. Keep in mind that the list can contain different types of items, like numbers, text, or variables.

It's important to note that this block only works on lists that have already been created and have items in it. It will not return an item from a non-existing list or an empty list. Also, the index positions starts from 1, so if you want to access the first item in a list, you would use index 1, the second item would be index 2 and so on.

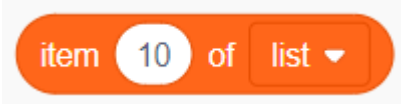


Figure 83: Item <index> Of List block

https://youtu.be/9lNqGpN1lqc

Item # of <item> in List block

It is used to access a specific item in an existing list. The block takes two inputs, the first input is the item you want to access, and the second input is the list from which you want to access the item.

When the block is executed, it will return the index position of the item in the list. Keep in mind that the list can contain different types of items, like numbers, text, or variables.

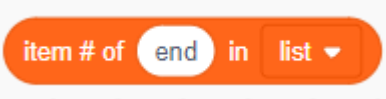


Figure 84: Item # of <item> in List block

It's important to note that this block only works on lists that have already been created and have items in it. It will not return an index from a non-existing list or an empty list.

https://youtu.be/DI4cV8Tlpd4

https://youtu.be/jVY03HVjNLg

Length Of List block

In Scratch 3.0, the "length of [list]" block is a block that is used in the "Variables" category. It is used to determine the number of items in a specific list. The block takes one input, the list of which you want to determine the length.

When the block is executed, it will return the number of items in the list. It's important to note that the list can contain different types of items, like numbers, text, or variables.

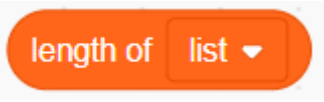


Figure 85: Length Of List block

Also, you can use this block to check if a list is empty or not, if the length of the list is equal to 0, then it's empty.

List Contains <item>? Block

It is used to check if a specific item is present in a list. The block takes one input, the item you want to check if it's present in the list.

When the block is executed, it will return a Boolean value, either "true" if the item is present in the list or "false" if the item is not present in the list.

It's important to note that this block only works on lists that have already been created and have items in it. It will not return any value for a non-existing list or an empty list. Also, it's important to keep in mind that this block is case-sensitive, so if the list contains "Apple" and you check for "apple" it will return false.

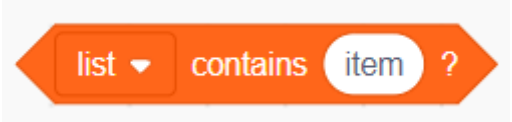


Figure 86: List Contains item

https://youtu.be/Ka464g2T1Ho

Show List <list> block
It is used to display the contents of a specific list on the screen. The block takes one input, the list that you want to display.

When the block is executed, it will display the items in the list on the screen. The items will be displayed in the order they appear in the list, separated by commas.

It's important to note that this block only works on lists that have already been created and have items in it. It will not display anything for a non-existing list or an empty list. Also, keep in mind that this block will only display the list on the screen, it will not return any value.

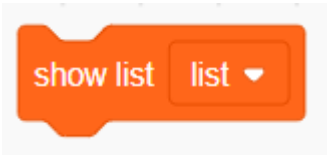


Figure 87: Show List <list> block

Hide List <list> block
It is a block which allows you to hide the contents of a specific list on the stage.

The block takes one input, the list that you want to hide. When the block is executed, it will hide the items in the list on the stage.

It's important to note that this block will only hide the list on the stage, it will not delete or clear its contents.

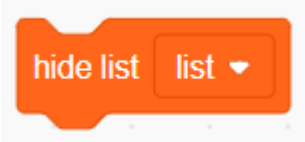


Figure 88: Hide List <list> block

https://youtu.be/kDrdG1oywSg

Rename – Delete List
In Scratch 3.x, there is no specific block called "rename list" or "delete list" in the "Variables" category. However, you can achieve the same effect by using any list block that contains a dropdown menu. Click on the dropdown menu and following menu will appear

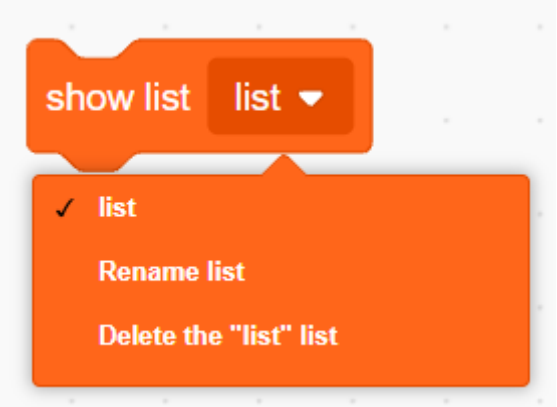


Figure 89: Rename / Delete <list>

Select "Rename list" to rename a list
Select "Delete the “list” list" to delete permanently the list

https://youtu.be/kBCu24bjLgA

IX. The code blocks of the MyBlocks Category

In Scratch 3.x, the "My Blocks" category is a special category that contains user-defined blocks, also known as custom blocks or procedure blocks. These blocks allow you to group together a set of blocks and give them a name, making it easy to reuse that set of blocks in multiple places in your program.

Myblocks are like functions in other programming languages.

When clicking on the "Make a Block" button following window appears

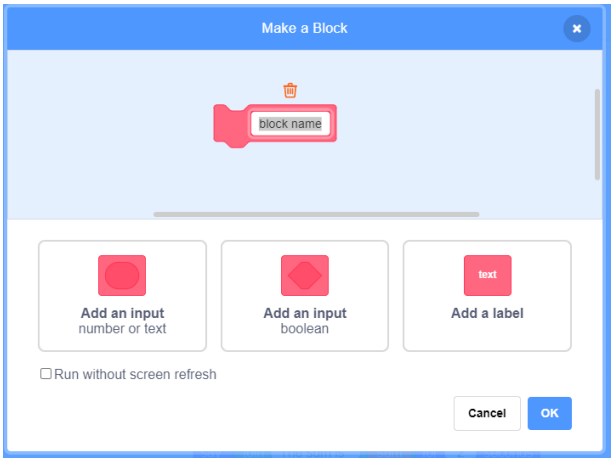
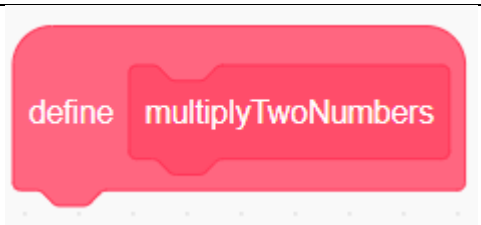
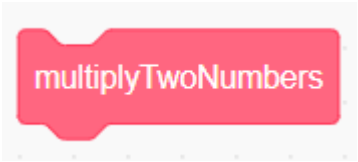


Figure 90: Make A Block block

You give the block a name and it is preferable to describe this name with a verb. Camel notation is a good practice when giving a name to a function or a variable.

When you create a function also a stack block is created with the name of the function.

Function	Stack block
	

https://youtu.be/2lsvD4_ICNo

The code blocks of the Pen Category

In Scratch, the "Pen" category is a collection of blocks that are used to control the pen's behavior of the selected sprite, such as drawing lines and shapes on the stage, controlling the pen's color and size, and clearing the stage. These blocks are located in the "Pen" section of the Blocks palette in the Scratch editor.

The Pen category includes blocks that allow you to control the following aspects of the pen:

- Positioning: You can use the "pen down" block to set the pen to the down position, meaning that when the sprite moves, it will leave a trail behind it on the stage. You can also use the "pen up" block to

set the pen to the up position, meaning that when the sprite moves, it will not leave a trail behind it on the stage.

- Color: You can use the "set pen color to [color]" block to set the color of the pen to a specified color, and "change pen color by [color]" block to change the current color of the pen by a specified amount.
- Size: You can use the "set pen size to [size]" block to set the size of the pen to a specified size, and "change pen size by [size]" block to change the current size of the pen by a specified amount.
- Erase: You can use the "clear" block to clear the trails that the sprite has drawn on the stage.

Overall, the Pen category in Scratch provides a set of blocks that allows you to easily control the pen's behavior of the selected sprite and make it draw lines and shapes, change color, size and erase trails. This makes it easy to create interactive and artistic projects.

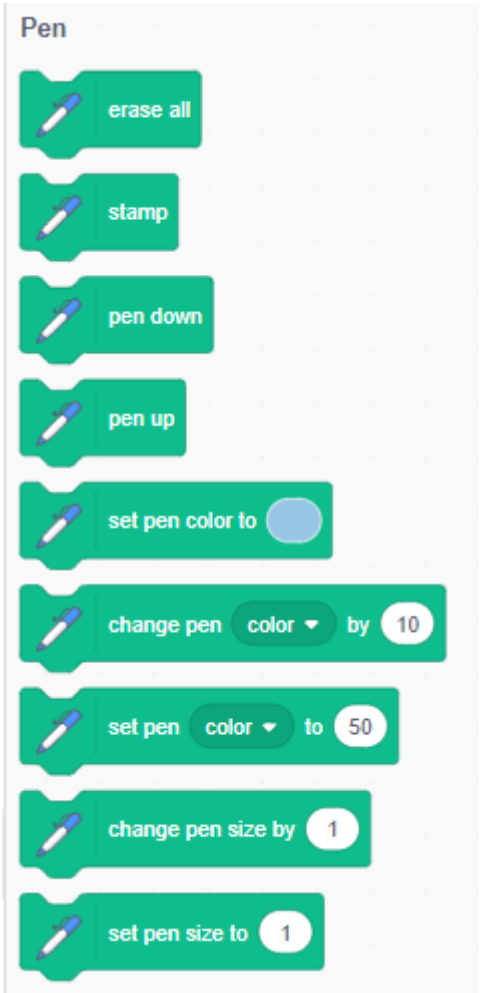


Figure 91: The Pen Category and its blocks

Here is an example how to use the "Pen" Category

https://youtu.be/JBYepH3S-zo

https://youtu.be/Wu1iwF_i-kM