

Meta Javascript Alpha 1.2

Programming for beginners

Meta_Javascript Alpha 1.2

Programming for beginners

Jose David Cuartas Correa



LOS LIBERTADORES
FUNDACIÓN UNIVERSITARIA

Meta_Javascript Alpha 1.2: Programming for beginners.

DRAFT EDITION 1.0: JULY, 2020

© Fundación Universitaria Los Libertadores

© Jose David Cuartas Correa

LOS LIBERTADORES, FUNDACIÓN UNIVERSITARIA

Bogotá D. C., Colombia

Cra 16 No. 63 A - 68 / Tel. 2 54 47 50

www.ulibertadores.edu.co

Juan Manuel Linares Venegas

President of the institution

Patricia Martínez Barrios

Chancellor

Ángela María Merchán Basabe

Academic vice chancellor

Jose David Cuartas Correa

Layout and cover design

Creative Commons License by-sa 4.0

<http://creativecommons.org/licenses/by-sa/4.0/deed.es>

INTRODUCTION

With this text I seek to introduce the reader to the basic aspects of the first version of Meta_Javascript, a meta-programming language that I developed for the beginner of web application programming. It is based on Meta_Processing and was designed to help you create interactive Javascript code. It is a personal initiative that is influenced by my work as director of the Hypermedia Laboratory (Hitec Lab) at the Fundación Universitaria Los Libertadores (Bogotá, Colombia).

What is sought with this derivation of Meta_Processing is to create an ecosystem of applications that allow the user to program different platforms using the same programming language. Today it is almost essential to learn a different language to program each platform.

This programming environment is based on the needs that I identified during the years that guided a basic programming course for graphic designers at the Fundación Universitaria Los Libertadores. It also responds to part of my research interests since when I was doing my PhD in Design and Creation, which ended with the thesis: "Program the world in the context of free technologies and Hacker-Maker cultures. Case study: Hitec Lab "(fourth, 2017).

Meta_Javascript is one of several initiatives that I have been leading in the quest to demystify technologies and to help fulfill one of the unfulfilled promises of free software: Allow anyone to modify and adapt the software they use, so that they can adjust it to their particular needs and personal likes. So making programming code easier to write and read is one of the purposes of Meta_Processing and Meta_Javascript.

Alpha Introduction 1.2.

In this new version of Meta_Javascript the following improvements are incorporated: The Configuration tab is added, it allows moving the lines of code with the mouse and the Punjabi, Kannada, Bengali, Tamil, Korean, Russian and German languages are supported. Now you can run the project on mobile devices (connected to the same network) by scanning a QR code. Support has been added to communicate with Arduino boards using Firmata and ESP boards using the IoTControllerAP library¹ (you do not need to install Node.js or Johnny-Five, just have Meta_Javascript open). The text instruction can now be used to display variables. The *multiply*, *divide* and *formula* instructions are added in the Mathematics category, and the *native* code instruction is added in the Advanced category. The *hyperlink* and *readparameter* instructions are added in

¹ <https://github.com/hiteclab/IoTControllerAP>

the HTML category and the instruction `for` is added in the Structures category. Additionally each time the run button is clicked, the meta code of the project is exported in three text files, which are saved inside the **meta** folder of the project. The code contained in each tab is saved in a text file with the same name as the tab and with **.meta** extension. And lastly is added support for the keyboard shortcuts: `ctrl+c`, `ctrl+x`, `ctrl+v`, `ctrl+z`.

Jose David Cuartas Correa
Bogota Colombia
2020

Thanks:

To the unconditional support received from the Fundación Universitaria Los Libertadores who have believed in every project we develop at Hitec lab.

Dedication:

To my wife Shahzadi and my daughters Helen and Megan, who fill my existence with love and joy.

CONTENT

1. META_JAVASCRIPT FIRST STEPS	13
1.1. How to open Meta_Javascript?.....	13
1.2. Windows that open when you start Meta_Javascript.....	16
1.3. Basic elements of the interface	17
1.4. How to select Languages?	18
1.5. How to open a project?.....	18
1.6. How to execute the code?.....	19
1.7. How to generate the project's QR code?	19
1.8. How to update the project without running it locally?	20
1.9. How to add a line of code?	21
1.10. How to delete a line of code?	21
1.11. How to add instructions?	22
1.12. What is the file and folder structure in Meta_Javascript?.....	24
1.13. How to open the data folder of the project?	26
1.14. How to create a new project?.....	26
1.15. How to save the current project?.....	27
1.16. How to export the current project as application?	27
1.17. Configuration options	28
1.18. Functions: Principal, Keyboard, Mouse and Configuration.....	30
1.18.1. Principal	30
1.18.2. Mouse	31
1.18.3. Keyboard	31
1.18.4. Configuration	32
1.19. Keyboard shortcuts: ctrl + c, ctrl + x, ctrl + v, ctrl + z.....	33
2. BASIC INSTRUCTIONS	35
2.1. Document the code	36

2.2.	Screen coordinates	37
2.3.	On-screen graphics instructions.....	39
2.3.1.	background.....	39
2.3.2.	line	40
2.3.3.	rectangle.....	40
2.3.4.	ellipse	41
2.3.5.	triangle	42
2.3.6.	texsize	42
2.3.7.	text	43
2.3.8.	image	44
2.3.9.	linecolor	44
2.3.10.	fill	45
2.3.11.	nofill.....	46
2.3.12.	noborder	47
2.4.	Multimedia instructions.....	47
2.4.1.	playnote.....	47
2.4.2.	sound	48
2.4.3.	video	48
2.5.	HTML instructions	49
2.5.1.	hyperlink	49
2.5.2.	read parameter.....	50
2.6.	Advanced Instructions	52
2.6.1.	native code	52
3.	VARIABLES, CONDITIONALS AND CYCLES	53
3.1.	Variables.....	53
3.1.1.	How do you look at the list of variables?.....	54
3.1.2.	How is a variable created?	54
3.1.3.	How is a variable removed?.....	55

3.1.4. How is a variable initialized?	56
3.1.5. How to assign a new value to a variable?	57
3.1.6. How to add a value to a variable?	58
3.1.7. How to subtract a value from a variable?	59
3.1.8. How to multiply a variable?	60
3.1.9. How to divide a variable?	61
3.1.10. How to assign random value to a variable?	62
3.1.11. How to add a mathematical formula?	63
3.2. Conditionals	65
3.1. Cycles.....	68
4. ARDUINO WITH META_JAVASCRIPT	72
4.1. Installation of the Firmata library on an Arduino board	73
4.2. Arduino instructions	77
4.2.1.digitaloutput	77
4.2.2.digitalinput	80
4.2.3.analoginput.....	83
4.2.4.servo	85
5. ESP BOARD WITH META_JAVASCRIPT	87
5.1. Installing the IoTControllerAP library on an ESP board	88
5.2. IoTControllerAP Instructions	92
5.2.1.digitaloutput	93
5.2.2.digitalinput	95
5.2.3.analoginput.....	96
5.2.4.servo	97
6. CODE EXAMPLES WITH META_JAVASCRIPT	99
6.1. Abstract line drawings	99
6.2. Basic example with the Keyboard	100
6.3. Basic example with the Mouse.....	101

6.4.	Animation	102
6.5.	Piano.....	103
6.5.1.	Simple piano	104
6.5.2.	Piano colors	104
6.5.3.	Piano colors and notes on screen	105
6.6.	Mini Game.....	107
	References	110

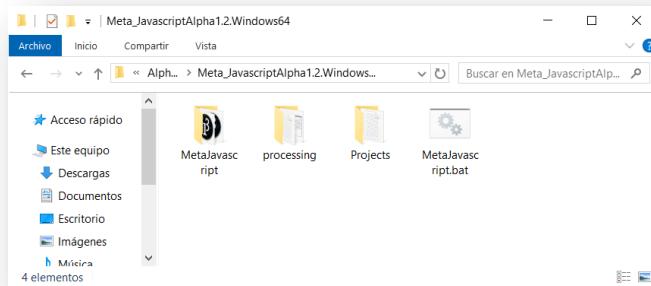
1. META_JAVASCRIPT FIRST STEPS

1.1. How to open Meta_Javascript?

Meta_Javascript was developed to work on the operating systems: Windows, Mac and GNU / Linux. The steps to open Meta_Javascript vary slightly depending on the operating system used, the steps for each system are described below:

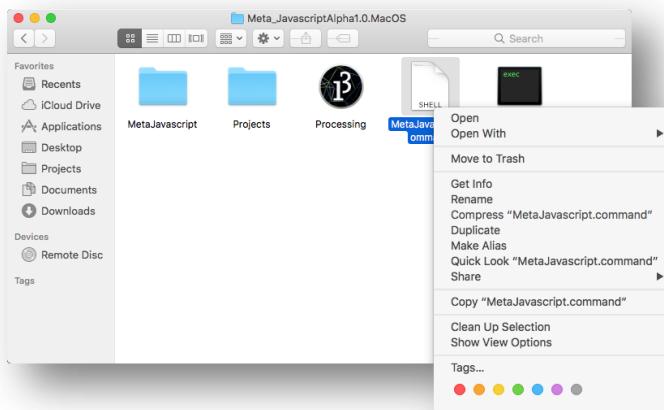
Windows

In Microsoft Windows, double-click on the file with the name: **MetaJavascript.bat**

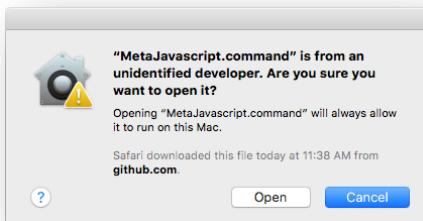


Mac OS

On Mac Os you must right-click on the file with the name **MetaJavascript.command** and select the option: **Open**

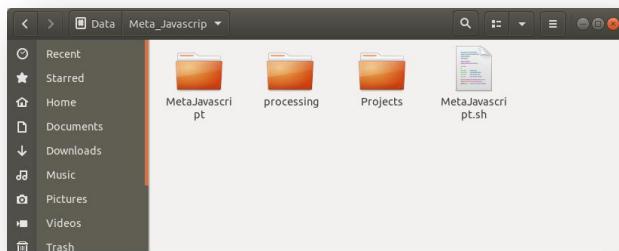


In the window that opens, select the option: **Open**

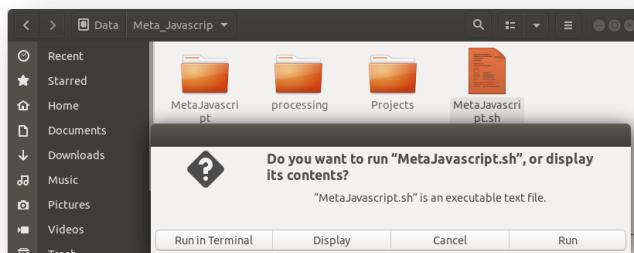


GNU / Linux

In GNU / Linux double click on the file with the name **MetaJavascript.sh**



In the window that opens you must select the option: **Run** (you can also use Run in Terminal if you want to open the **terminal window of Meta_Javascript**).

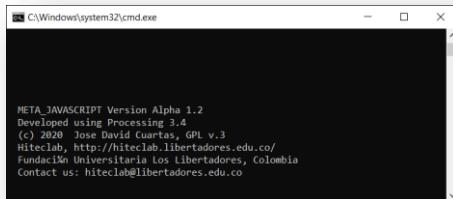


If the script does not open when you double click on it, you can try to run the following command in the Linux terminal to activate the previous dialog box.

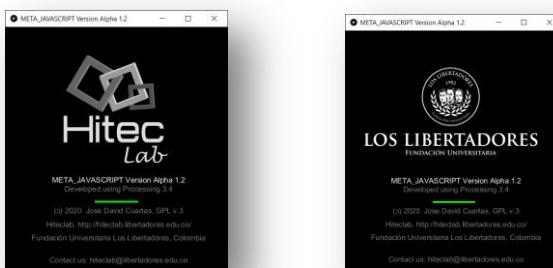
```
gsettings set org.gnome.nautilus.preferences executable-text-activation ask
```

1.2. Windows that open when you start Meta_Javascript

Once the MetaJavascript file is executed, regardless of the operating system you are working on, the terminal window opens where you can see messages that come from the main Meta_Javascript window.

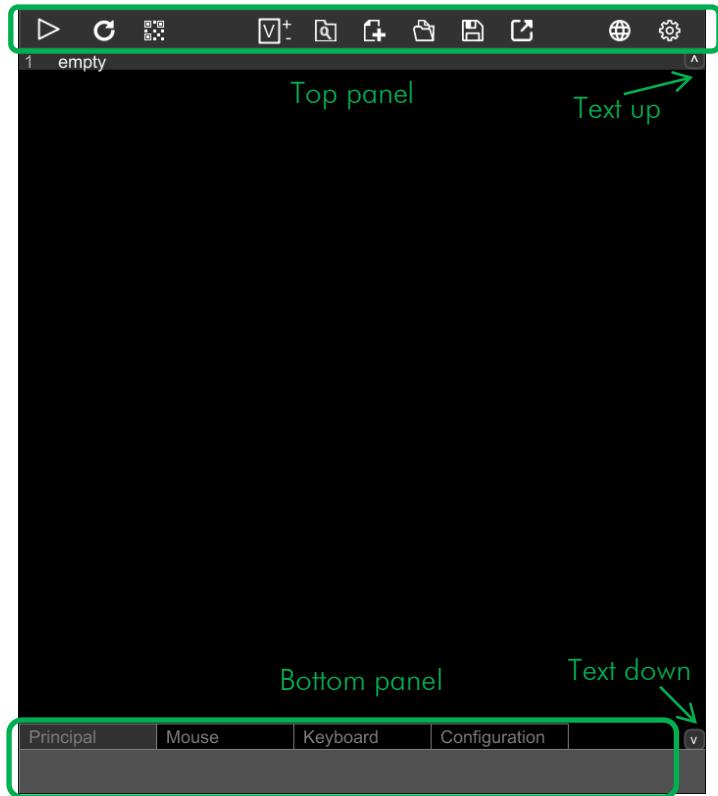


Then a second window opens with the animated splash of welcome to Meta_Javascript. Clicking on or closing this window opens the main Meta_Javascript window.



1.3. Basic elements of the interface

In the upper panel are the buttons: Run, Refresh, Generate QR, Variables, Data, New, Open, Save, Export, Languages and Configuration.



In the Bottom panel are the tabs: Principal, Mouse, Keyboard and Configuration. Also there is the description bar: where the names of the buttons and the prototype of the instructions are shown.

1.4. How to select Languages?

To change the Meta_Javascript language, click on the languages icon in the top bar.



Then in the window that opens, click on the desired language.

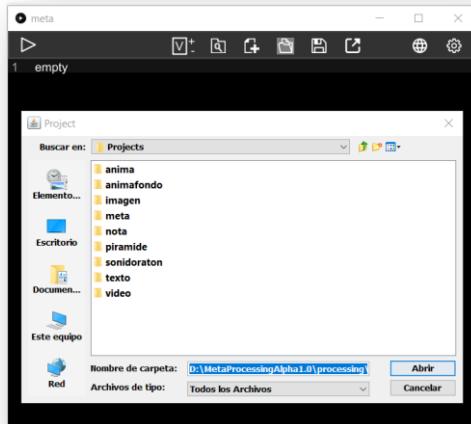


1.5. How to open a project?

To open a project, click on the **open** icon on the top bar.



Then a new window opens in which you can select the project folder you want to open and click the **open** button.



1.6. How to execute the code?

To execute the code, click on the run icon in the top bar.



Wait few seconds and a new window should appear in which the created code will be executed.

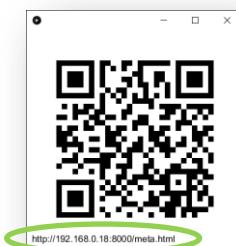
1.7. How to generate the project's QR code?

To run the project on a mobile device connected to the same network, a QR code can be generated that, when scanned, allows directing to the URL of the local server

where the project can be run in the web browser of the mobile device. For this you must click on the Generate QR icon in the top bar.



Wait one second and a new window should appear in which you can see the QR code that directs the project.



If the equipment does not have a software to scan QR codes, you can also use the URL that appears below the QR code and which must be entered in the address bar of the web browser of the mobile device.

1.8. How to update the project without running it locally?

For when a change is made in the code and you want to see these changes on the mobile device, without having to run it locally or generate the QR code again, you can update the local project server and then update the web browser on the device mobile. This allows the changes made to the code to be seen on the mobile

device that already has the project URL in the address bar of the Web browser. For this you only need to click on the Refresh icon in the top bar.



1.9. How to add a line of code?

To add a line of code you must move the mouse cursor until a green circle appears with the plus (+) character inside it.

```
1 background(px)
2 add(px, 0.5)
```

A screenshot of a code editor showing two lines of code. The first line is 'background(px)' and the second line is 'add(px, 0.5)'. A green circle with a plus sign is positioned next to the second line, indicating where a new line can be added.

A new line of code will appear once the green circle is clicked.

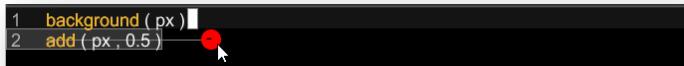
```
1 background(px)
2 add(px, 0.5)
3 empty
```

A screenshot of a code editor showing three lines of code. The first line is 'background(px)', the second line is 'add(px, 0.5)', and the third line is 'empty'. The 'empty' line is the result of clicking the green circle from the previous screenshot.

1.10. How to delete a line of code?

To remove a line of code, you must move the mouse cursor until a red circle appears with the minus character inside it.

ter (-) inside it, and you see a gray line crossing out the entire instruction.



The line disappears once it is clicked.



1.11. How to add instructions?

To add an instruction, you must click on the word that says **empty**.



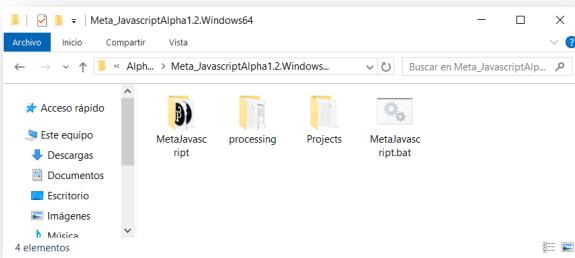
Once this is done, a new window will open where all the instructions available in Meta_Javascript will appear, organized by categories like this:



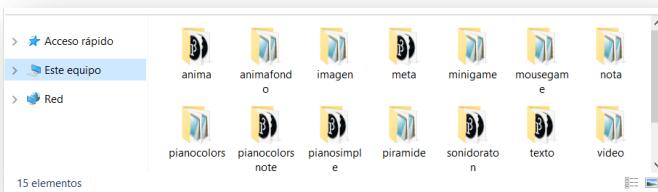
When the mouse cursor is placed on any of these instructions the instruction is highlighted, in this example you can see how the **rectangle** instruction is highlighted. Clicking will open a new window where you can enter the properties of each instruction. The description of each of these instructions will be made in Chapter 2.

1.12. What is the file and folder structure in Meta_Javascript?

Inside the **Meta_Javascript** folder is the file to run **Meta_Javascript** and three sub-folders. The **Meta_Javascript** folder contains the files that allow it to function. In the processing folder there is a distribution of this language that is used to execute **Meta_Javascript**.

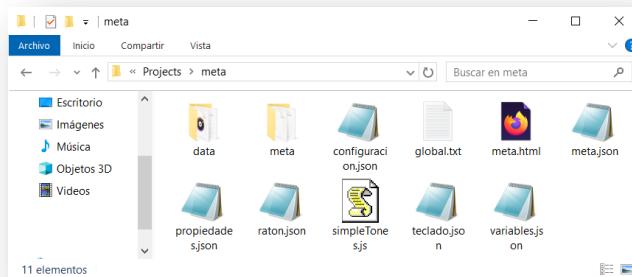


In the **Projects** folder it contains the folders of each of the program projects written using the **Meta_Javascript** programming environment.



In the folder of each project there are few **.json** files, one **.js** file and one **.html** file. The **.json** files contain the

instructions in Meta_Javascript language, the **.js** file contain a sound library necessary to play notes in the web browser and the **.html** file contain the JavaScript code of the project, and is generated every time the **run** icon is clicked.



Starting with this version, the **global.txt** file is added, in which instructions can be added using the notepad. It is useful for projects where you need to add global instructions or functions outside of the Principal, Mouse, Keyboard and Configuration functions. It is an option for advanced users.

And finally in the **data** folder of each project are stored the files that will be used in the execution of the program, such as images, sounds and videos.



1.13. How to open the data folder of the project?

To open the data folder of the current project, click on the **data** icon in the top bar.



Once clicked, the data folder of the current project opens in another window

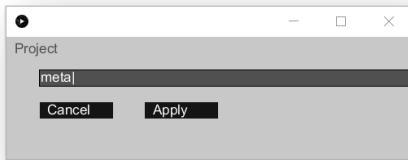


1.14. How to create a new project?

To create a new project, click on the **new** icon in the top bar.



In the window that opens, you must write the name that you want to give to the new project and click on the **Apply** button.



1.15. How to save the current project?

To save the current project, click on the **save** icon in the top bar.



1.16. How to export the current project as application?

To export the current project as an application, click on the **export** icon on the top bar.



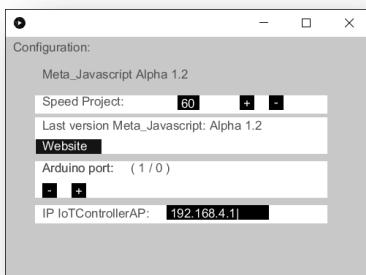
The application is saved in the subfolder called **web** inside the current project folder.

1.17. Configuration options

To change the configuration options, click on the **configuration** icon in the top bar.



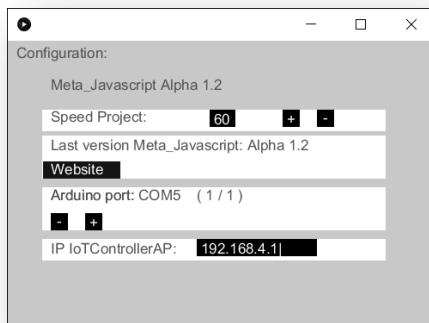
Then a new window opens offering four options: change project speed, check the latest version of Meta_Processing, select Arduino port and define IP address for IoTControllerAP.



The **Speed** option allows you to change the speed of the project (a rough measure of frames per second, but still not entirely accurate).

The **Last version** option allows you to check the latest version of Meta_Javascript published on the internet. The information that appears after the colon is the version available for download. If you want to download that new version you can click on the **Website** button that directs to the Meta_Javascript website.

The **Arduino Port** option shows the ports of the devices connected to the computer. In case of connecting an Arduino card through the USB port, the connection port can be selected by clicking on the - or + buttons.(For more information see Chap. 4)



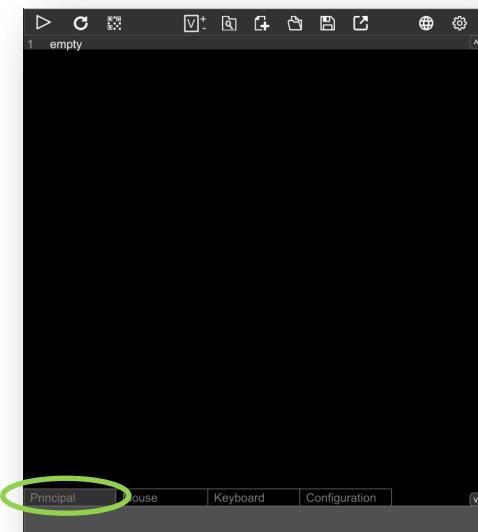
The **IP IoTControllerAP** option allows you to change the IP address of the ESP board by running IoTControllerAP.

1.18. Functions: Principal, Keyboard, Mouse and Configuration

To write the code in Meta_Javascript, four functions can be used: Principal, Mouse, Keyboard and Configuration. Each of these functions is selected by clicking on its corresponding tab.

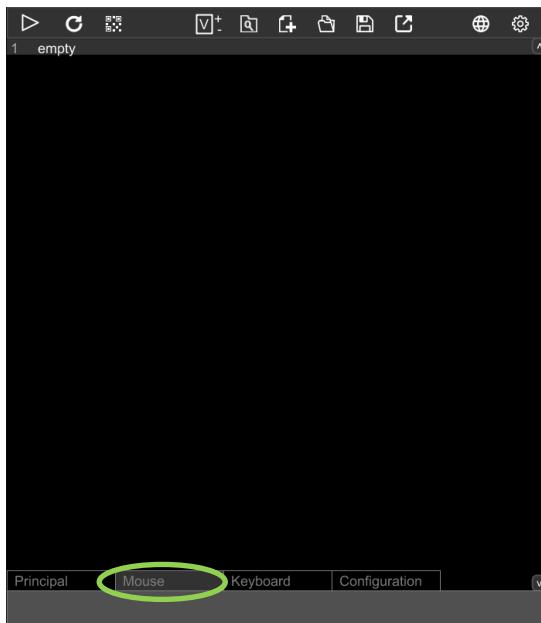
1.18.1. Principal

The code that is written to the **Principal** tab runs in an infinite loop, until the application window is closed.



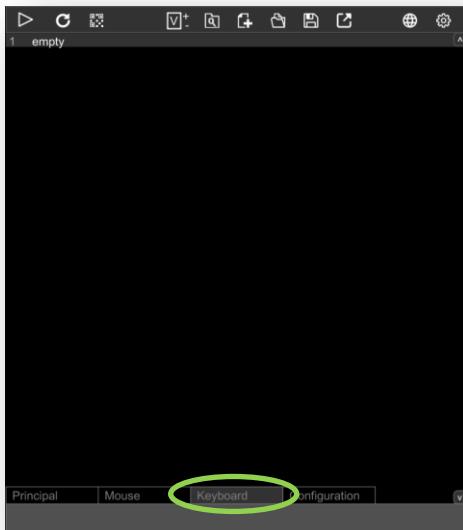
1.18.2. **Mouse**

The code that is written on the **Mouse** tab is executed the moment any mouse button is pressed.



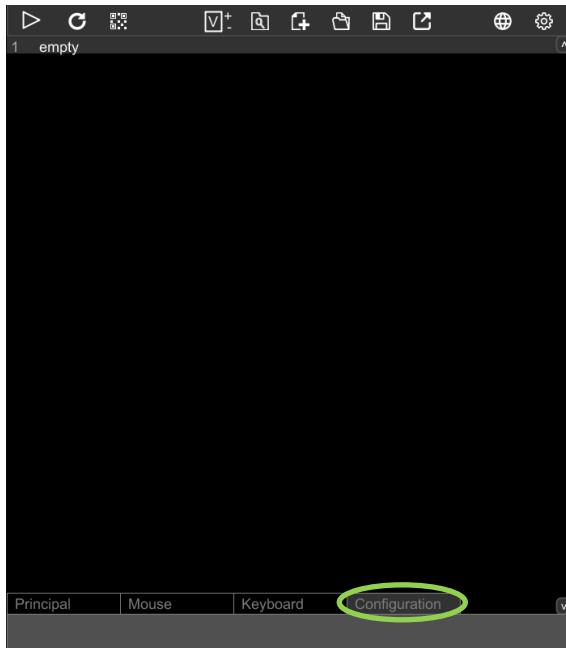
1.18.3. **Keyboard**

The code that is written on the **Keyboard** tab runs the moment any key is pressed.



1.18.4. Configuration

The code that is written in the Configuration tab is executed only once at the moment the application is opened, it is the first function that is executed, before Principal, Mouse or Keyboard. It is used to set the initial configuration of the application (such as initialize variables).



1.19. Keyboard shortcuts: **ctrl + c, ctrl + x, ctrl + v, ctrl + z**

From this version of Meta_Javascript is added support for the keyboard shortcuts: **ctrl+c, ctrl+x, ctrl+v, ctrl+z**.

The **ctrl+c** shortcut allows you to copy the line of code that is being pointed by the mouse cursor.

The **ctrl+x** shortcut allows you to cut the line of code that is being pointed by the mouse cursor.

The **ctrl+v** shortcut allows you to paste the line that has been copied or cut at the position of the line of code that is being pointed by the mouse cursor.

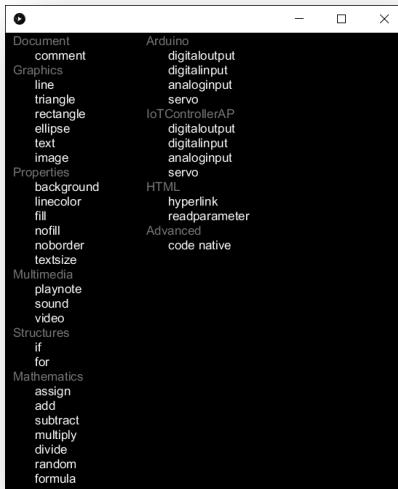
The **ctrl+z** shortcut allows you to undo delete line. If you delete a line by mistake, you can retrieve the deleted line by pressing **ctrl+z**. It only applies to the last line deleted.

2. BASIC INSTRUCTIONS

This section will cover the basic instructions for programming with Meta_Javascript. As explained in point 1.11, if you want to add an instruction you must click on the word that says **empty**.

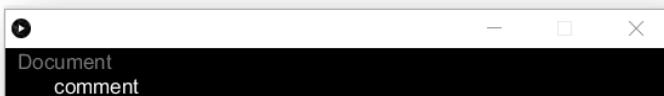


Then, a new window will open showing all the instructions available in Meta_Javascript organized by categories like this:

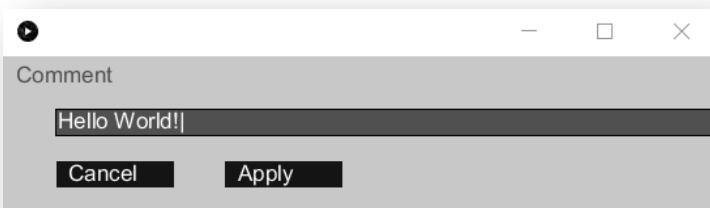


2.1. Document the code

Documenting the programming code is one of the first things anyone who wants to learn to program must learn. For this purpose, all programming languages allow adding comment lines. The main feature of this line of code is not executed; it's just there to give the developer information on how that part of the code works. To add a comment, click on the **comment** option in the **Document** category.



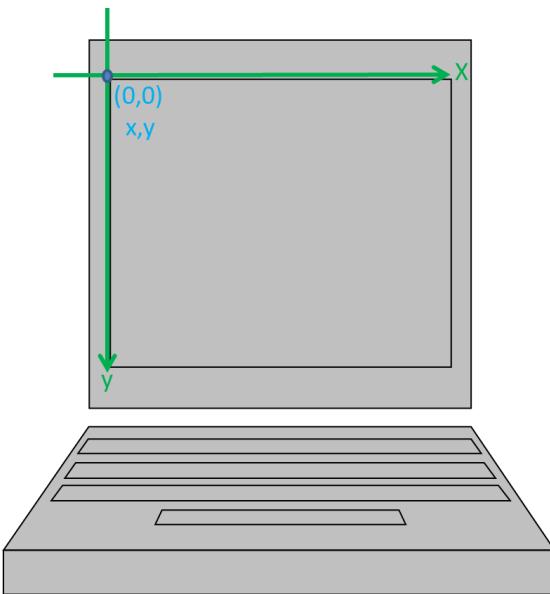
Then a new window will open in which you write the comment and click **Apply**.



2.2. Screen coordinates

In order to make graphics on the screen, it is necessary to first know how the screen coordinates work in Meta_Javascript. The positions on the screen are measured in pixels and each screen has a certain number of pixels on the X axis and on the Y axis.

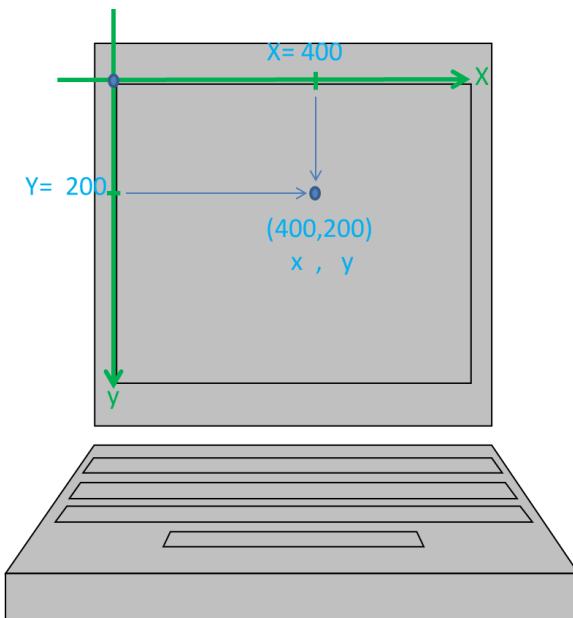
As can be seen in graph 1, the upper left corner of the screen is the point of origin of the coordinate system. This point is position 0 on the X axis and position 0 on the Y axis. Coordinates are always arranged first with value on X axis, then a comma and then the value on the Y axis, so this point is (0,0).



Graph 1 Origin point in the coordinate system

The positions on the X axis increase from left to right and the positions on the Y axis increase from top to bottom. Figure 2 shows an example to better illustrate this concept.

If you want to locate the point (400,200) on the screen, what you do is count 400 pixels to the right from the point (0,0) of the screen and count 200 pixels down from the point (0,0) of the screen. This is how the point (400,200) is located.

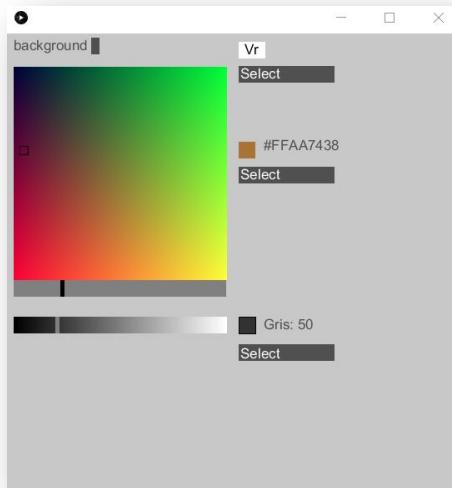


2.3. On-screen graphics instructions

Some of the instructions to display on the screen include: line, triangle, rectangle, ellipse, text, image. Next pages will explain how to use each of them.

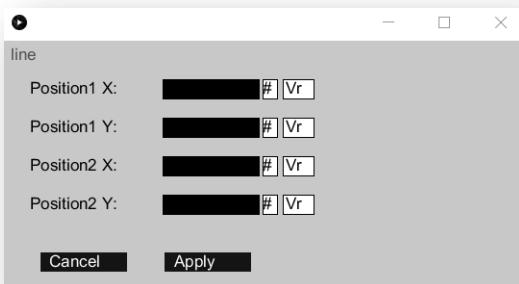
2.3.1. background

The **background** instruction is used to define the background color of the entire window of the application. This instruction erase everything that is being displayed on the screen and leaves the entire screen with the selected color.



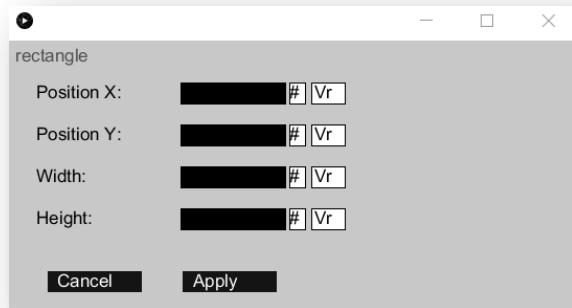
2.3.2. line

The **line** instruction is used to draw a line on the screen. Tracing a line requires defining the (x,y) position of the point where the line begins and the (x,y) position of the point where the line ends.



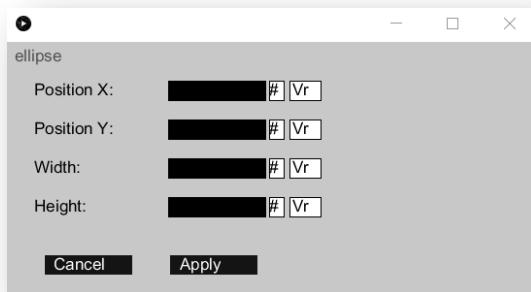
2.3.3. rectangle

The **rectangle** instruction is used to draw a rectangle on the screen. To use this instruction you must define in the first two boxes the position x,y of the upper left corner from where the square will be drawn, and in the next two boxes must be defined its width and height.



2.3.4. ellipse

The **ellipse** instruction is used to draw an ellipse on the screen. To use this instruction you must define in the first two boxes the center point x,y from where the ellipse will be drawn, and in the next two boxes define its width and height.



2.3.5. triangle

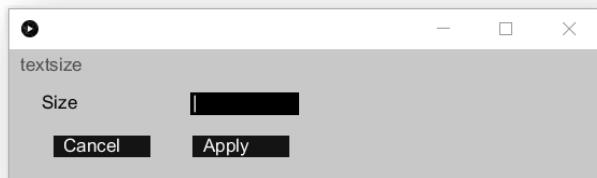
The **triangle** instruction is used to draw a triangle on the screen. To draw any triangle it is required to define the three points corresponding to each of its corners. To use this instruction, the position of the first point must be defined in the first two boxes, the position of the second point must be defined in the following two boxes, and the position of the third point must be defined in the last two boxes.



2.3.6. texsize

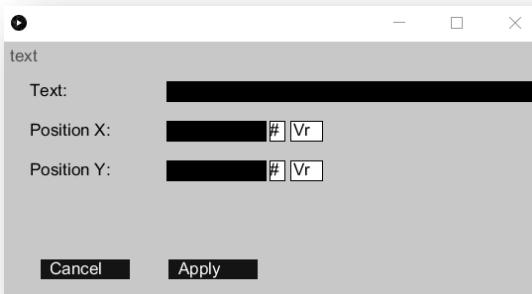
The **texsize** instruction is used to define the size of the font when displaying text on the screen. To use this instruction, the size box must be filled with the number corresponding to the font size to be applied.

For this statement to take effect, it must be added before the **text** instruction.



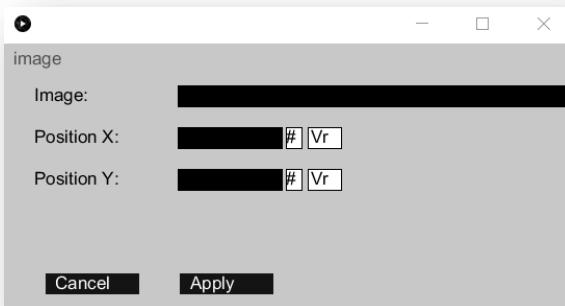
2.3.7. **text**

The **text** instruction is used to display text on the screen. To use this instruction, you must write in the first box the text you want to display, and define in the next boxes the x, y position of the lower left corner from where the text will start to be displayed on the screen.



2.3.8. **image**

The **image** instruction is used to display an image on the screen. To use this instruction, you must first save the image you want to use inside the project's **data** folder, then select the image that was saved in the data folder, and after that in the next two boxes define the x, y position of the upper left corner from where the image will begin to be displayed on the screen.



2.3.9. **linecolor**

The **linecolor** instruction is used to define the color of the lines and the border color of the rectangular, ellipse and triangle figures. To assign the line color using this instruction you can, use the color picker, or the grayscale picker and click the **Apply** button. You can also use a variable to dynamically change the color. For this instruction to take effect, it must be added before

the instructions with which to draw lines or figures on the screen.



2.3.10. fill

The **fill** instruction is used to define the fill color of the rectangle, ellipse and triangle figures. To assign the fill color using this instruction you can, use the color picker, or the grayscale picker and click the **Apply** button. You can also use a variable to dynamically change the color. For this instruction to take effect, it must be added before the instructions with which to draw figures on the screen.



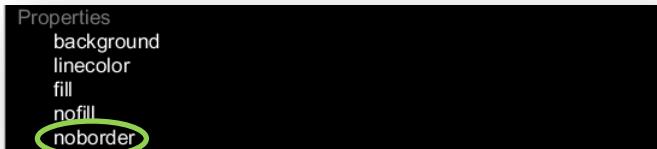
2.3.11. **nofill**

The **nofill** instruction is used to remove the filling of the rectangle, ellipse and triangle figures. This will look transparent and only its edge will be seen. To use this instruction, it is only necessary that in the add instruction window, you click on the **nofill** instruction within the **Properties** category and this line will automatically be added to the project code.



2.3.12. **noborder**

The **noborder** instruction is used to remove the border of the rectangle, ellipse and triangle figures. If this statement is added before the **line** instruction then the line will not be displayed. To use this instruction, it is only necessary that in the add instruction window, you click on the instruction **noborder** within the **Properties** category and this line will automatically be added to the project code.

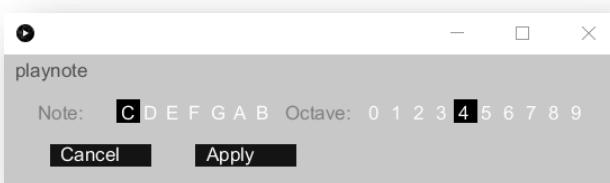


2.4. Multimedia instructions

Some of the multimedia instructions that can be used with Meta_Javascript are: **playnote**, **sound** and **video**. The next pages will explain how to use each of them.

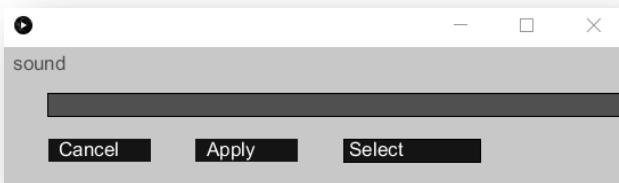
2.4.1. **playnote**

The **playnote** instruction is used to reproduce the sound of a note on the musical scale. To use this instruction, you must select the note that you want to play and then select the octave that you want the note to sound. Finally, click on the **Apply** button.



2.4.2. sound

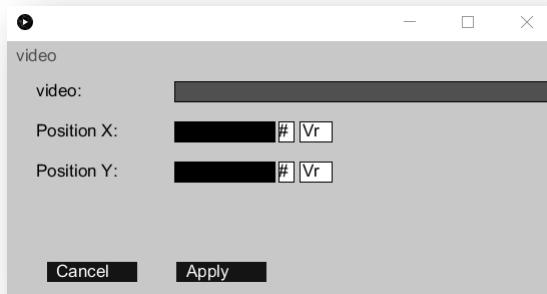
The **sound** instruction is used to play a sound file in wav or mp3 format. To use this instruction you must first save the sound file you want to use inside the project's **data** folder, then click on the **Select** button to choose the file that was previously saved in the data folder and finally you must click the **Apply** button.



2.4.3. video

The **video** instruction is used to play on the screen a video file in compatible format with the web browser. To use this instruction you must first save inside the project's **data** folder the video file you want to use, then select

the file that was saved in the data folder and finally click the **Apply** button.

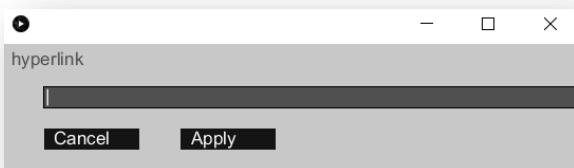


2.5. HTML instructions

The HTML instructions that can now be used with Meta_Javascript are: **hyperlink** and **readparameter**. The following will explain how to use each of them.

2.5.1. hyperlink

The **hyperlink** instruction is used to jump to another web page, either local or on the Internet. The path opens in the same browser tab, which means that the Meta_Javascript code execution is interrupted to open the web page defined in the hyperlink.

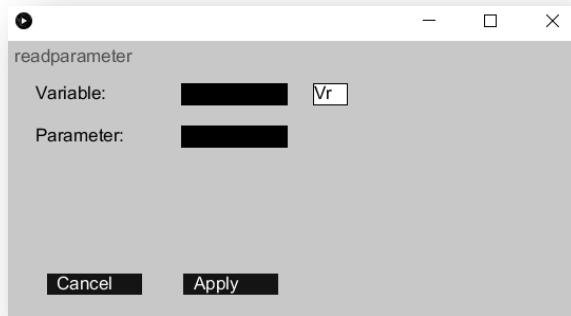


An example could be a project that opens the google web page when any mouse button is clicked. For this, the following code must be added in the Mouse tab.

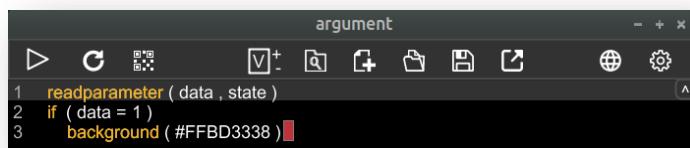


2.5.2. **read parameter**

The **readparameter** statement is used to read the parameter that can be sent to the project's .html file when it is called from the web address bar or by clicking on a hyperlink within it or from another project. To use this instruction in the first box, you must select the variable in which the value to read will be stored and in the second box, you must write the name of the parameter you want to read.



An example could be creating a project that allows reading the state parameter and storing it in the variable data. For this it would be necessary to add the following code in the main tab:



If you wanted a parameter to be sent by clicking any mouse button via URL parameters, you could use the hyperlink statement as follows:

1 hyperlink (localhost:8000/argument.html?state=1)

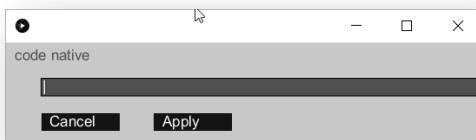
In this case the value 1 would be sent to the state parameter to the argument.html page.

2.6. Advanced Instructions

At the moment the only advanced instruction available for Meta_Javascript is: *native code*. How to use it will be explained below.

2.6.1. native code

The native code instruction is used to add lines of code that are not available with Meta_Javascript, but are native for JavaScript language. To use this instruction you must write in the first box, the line of code that you want to add and then click on the **Apply** button.



An example could be using the JavaScript instruction called **alert** that allows displaying text in a popup window. If the following line of code is added in the Mouse tab:

```
alert ("Hello");
```

Then every time any mouse button is pressed, would show in a popup window the text: Hello.

3. VARIABLES, CONDITIONALS AND CYCLES

This section will address the concepts of variables, conditionals and cycles, which are fundamental when you are learning to program.

3.1. Variables

A variable is a memory space reserved for storing a value that changes while the program is on execution. In Meta_Javascript there are two types of variables: System variables and variables created by the user.

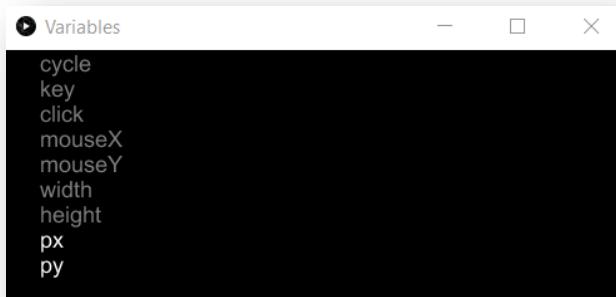
The system variables are: **tecla**, **click**, **ratonX**, **ratonY**, **ancho** and **alto**. The variable **tecla** stores the value of the last key pressed on the keyboard. The variable **click** stores the value of the last button pressed on the mouse. The variable **ratonX** stores the current mouse position on the X axis. The variable **ratonY** stores the current mouse position on the Y axis. The variable **ancho** stores the value of the width of the screen in which the code is executing. And the variable **alto** stores the value of the height of the screen in which the code is running.

3.1.1. How do you look at the list of variables?

To see the variables list of the project, click on the **variable** icon.



In the window that opens you will see the variables that are being used. The ones shown in gray are the system variables and the ones shown in white are the variables created by the user.

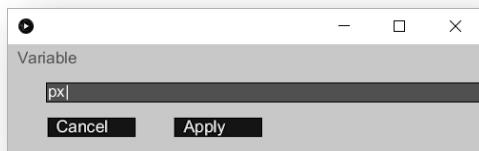


3.1.2. How is a variable created?

To create a variable, click on the plus (+) icon next to the **variable** icon.



In the window that opens, you must write the name that you want to give of the variable to be created, in this example it is given the name px.

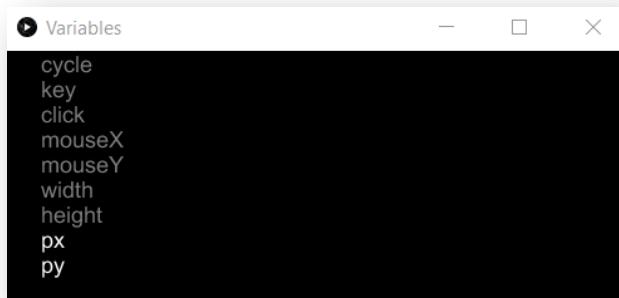


3.1.3. How is a variable removed?

To remove a variable, click on the minus icon (-) next to the **variable** icon.



In the window that opens, click on the name of the variable to be removed. Only variables that have been created by the user can be removed, that is, only variables with white color can be removed. Variables in gray are system variables and cannot be removed.

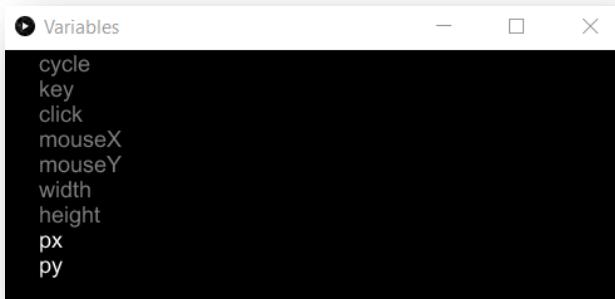


3.1.4. How is a variable initialized?

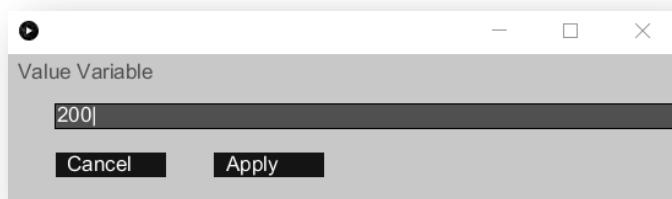
To initialize a user-created variable, click on the **variable** icon.



In the window that opens, click on the variable that you want to initialize.



Once this is done, another window will appear in which you must write the value with which you want to initialize the variable. In this example, the variable is initialized with the value 200.

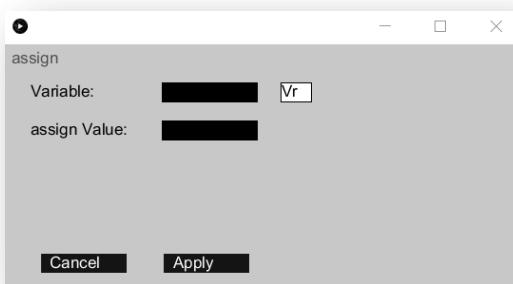


3.1.5. How to assign a new value to a variable?

Within the code you can assign a new value to a variable, for this you must add the **assign** instruction found within the **Mathematics** category in the add instruction window.



Inside the window that opens, in the first box select the variable you want to assign a new value and in the second box write the value to be assigned.

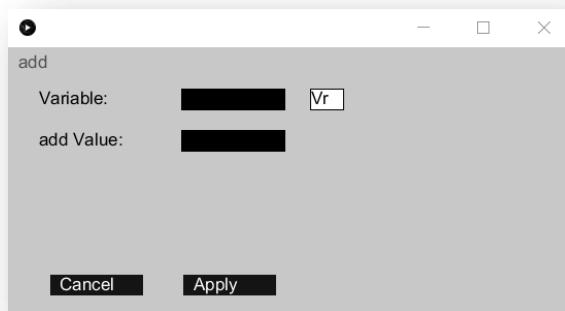


3.1.6. How to add a value to a variable?

Within the code you can add a value to a variable, for this you must aggregate the **add** instruction found within the **Mathematics** category in the add instruction window.



Inside the window that opens, in the first box select the variable you want to add the value and in the second box write the value to be added.

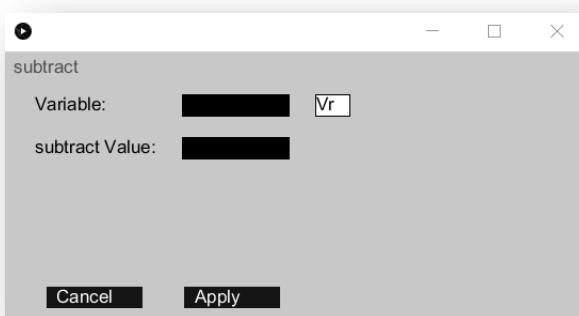


3.1.7. How to subtract a value from a variable?

Within the code you can subtract a value from a variable, for this you must add the **subtract** instruction that is within the **Mathematics** category in the add instruction window.



Inside the window that opens, in the first box select the variable you want to subtract the value and in the second box write the value to be subtracted.

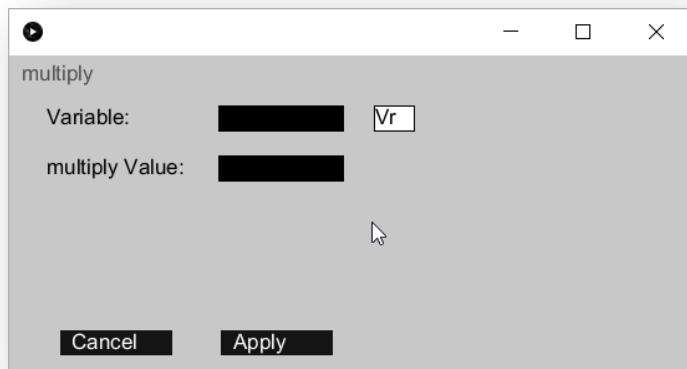


3.1.8. How to multiply a variable?

Within the code you can multiply a variable by a value, for this you must add the **multiply** instruction found within the **Mathematics** category in the add instruction window.

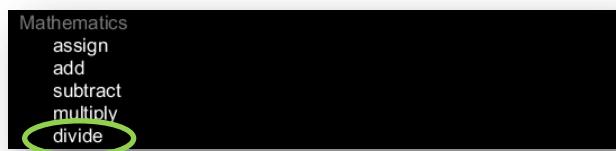


Then in the window that opens, in the first box select the variable you want to be multiplied by the value and in the second box enter the value by which the variable is to be multiplied.

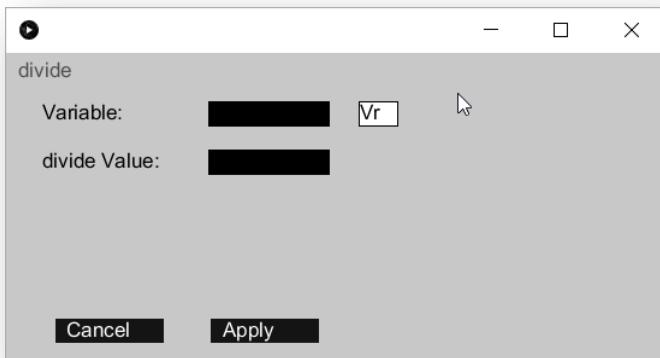


3.1.9. How to divide a variable?

Within the code, you can divide a variable by a value. To do this, you must add the **divide** instruction found within the **Mathematics** category in the add instruction window.



Then in the window that opens, in the first box select the variable you want to be divided by the value and in the second box enter the value by which the variable is to be divided.

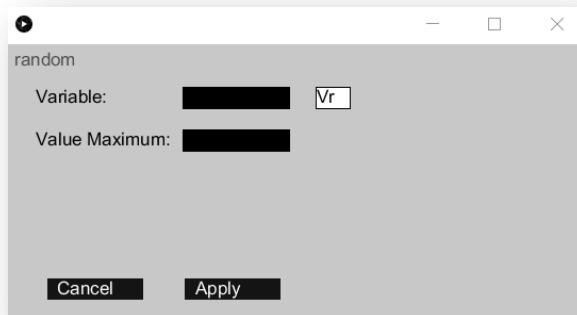


3.1.10. How to assign random value to a variable?

Within the code you can assign a random value to a variable, for this you must add the **random** instruction that is within the **Mathematics** category in the add instruction window.

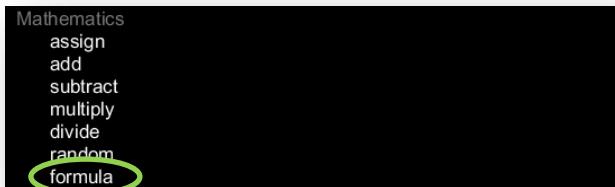


Inside the window that opens, in the first box select the variable you want to assign the random value and in the second box write the maximum value that would be generated randomly. A random value would be generated between 0 and the maximum value defined inside the instruction.

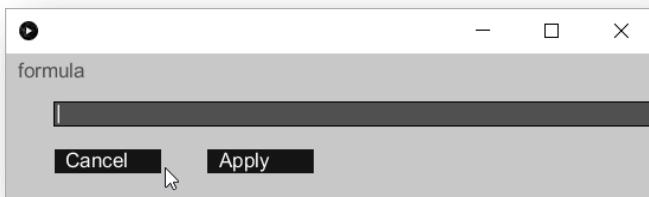


3.1.11. How to add a mathematical formula?

Within the code you can add a mathematical formula to do more advanced operations, for this you must add the **formula** instruction that is within the **Mathematics** category in the add instruction window.



Then in the window that opens, in the first box you must write the formula that you want to add and then click on the **Apply** button.



An example could be calculating the average between three values:

```
average = (value1 + value2 + value3) / 3;
```

Then the result of this mathematical operation would be stored in the variable called: average.

3.2. Conditionals

Conditions are a type of algorithmic structures that allow the program to make decisions as a condition is met.

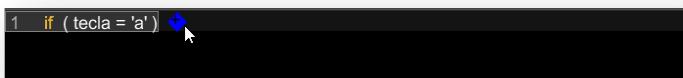
To add a condition, you need add the **if** instruction that is within the **Structures** category in the add instruction window.



Then in the window that opens in the first box you can write a value or select a variable. Then an operator must be selected, which can be equal (=), less than (<), greater than (>) or different (i=). And in the second box you can write a value, select a variable, choose a key or choose one of the mouse buttons.



Once the **Apply** button is clicked, the condition can be seen in the Meta_Javascript code window. To add an instruction inside the condition, you must move the mouse cursor until a blue diamond appears with the plus (+) character inside it.



Once clicked the new empty line of code will appear and click to assign the desired instruction. If you want to create one more instruction inside the condition, then you must move the mouse cursor until a blue diamond appears with the plus (+) character inside it.

```
1 if ( tecla = 'a' )
2 background ( #FF1ED538 )
```

If once you finish adding the instructions inside the condition, what you want is to add another line of code but outside the condition, then you must move the mouse cursor until a green circle appears with the plus character (+) inside.

```
1 if ( tecla = 'a' )
2 background ( #FF1ED538 )
```

If, on the contrary, what you want is to add a line for the case when said condition is not met, then you can add lines within the Else. For this you must move the mouse cursor until a yellow diamond appears with the plus (+) character inside it.

```
1 if ( tecla = 'a' )
2 background ( #FF1ED538 )
```

Once the instruction you want to use is selected, you would see that the word **Else** appears before the instruction. For when you want to add more instructions inside the **Else**, you must move the mouse cursor until a blue diamond appears with the plus (+) character inside it.

```
1 if ( tecla = 'a' )
2 background ( #FF1ED538 )■
3 Else background ( 50 )■
```

If, on the other hand, you want to add another line of code but outside the **Else**, then you must move the mouse cursor until a green circle appears with the plus (+) character inside it.

```
1 if ( tecla = 'a' )
2 background ( #FF1ED538 )■
3 Else background ( 50 )■
```



When you do this then a new empty line will appear outside the condition's instructions.

```
1 if ( tecla = 'a' )
2 background ( #FF1ED538 )■
3 Else background ( 50 )■
4 empty
```

3.1. Cycles

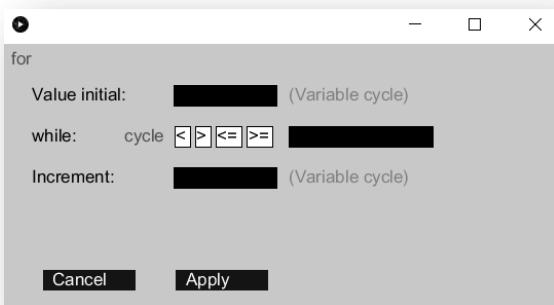
Cycles in programming are a type of algorithmic structures that execute a set of instructions multiple times as long as a condition is met. Cycles are very useful for

repetitive tasks, an example of use could be when you want to draw a grid, you could do it by writing the code to draw line by line, or you could do it by creating a cycle to draw all the horizontal lines and creating another cycle to draw all the vertical lines. If the grid is 3 lines x 3 lines, it is not difficult to draw line by line, but if the grid is 1000 x 1000, then with two cycles the task can be performed, and makes the code look optimized and simplified.

To add a cycle, the instruction `for` must be selected within the Structures category in the add instruction window.



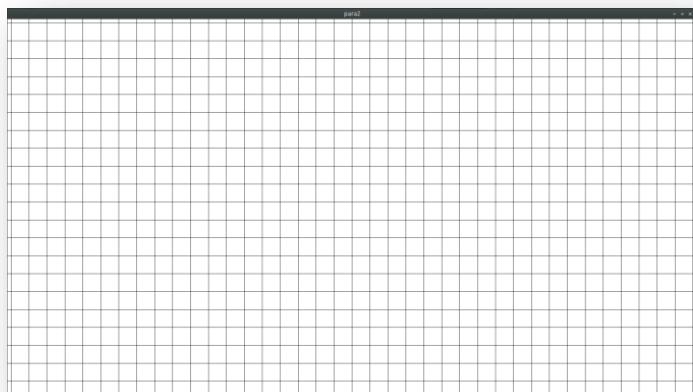
Then in the window that opens you can write in the first box the initial value for the system variable called **cycle**. Below this box, the operator can be selected to compare the value of cycle variable with a value that must be added in the second box. Finally, in the third box it is defined how much the cycle variable is going to be increased every time the cycle `for` is restarted.



An example of using the structure for can be drawing a grid. The following code can be written in the mouse tab, so when you press any of its buttons draw the grid.

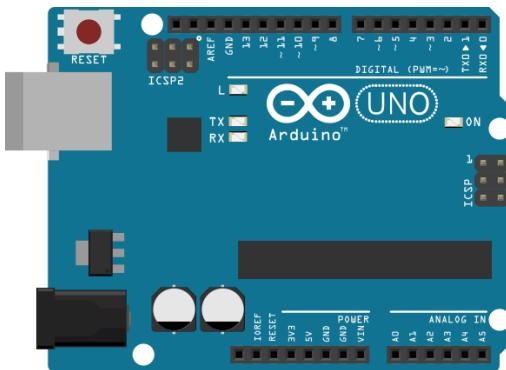
```
1 background ( 255 )
2 for ( desde 10 ; while cycle < 1920 ; Increment 50 )
3   line ( ciclo , 0 , ciclo , alto )
4 for ( desde 10 ; while cycle < 1080 ; Increment 50 )
5   line ( 0 , ciclo , ancho , ciclo )
```

The output you get from running this 5-line code should be similar to this:



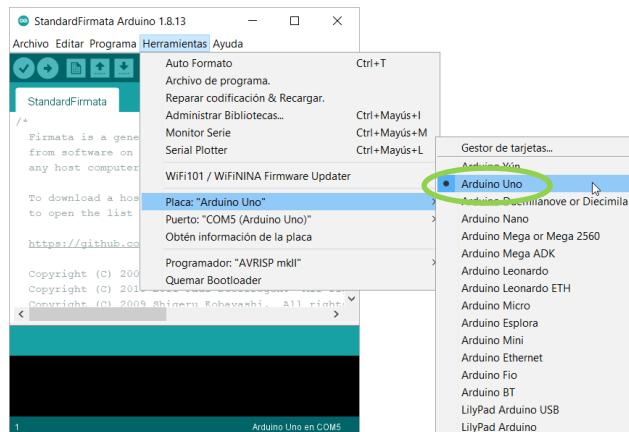
4. ARDUINO WITH META_JAVASCRIPT

To use an Arduino board with Meta_Javascript, the Arduino requires the Firmata library to be loaded. In this way you can establish communication through the USB port with the board and control several of its main functions. Turn pins on and off, digital readout of a pin, analog readout of a pin, and control servos. You do not need to install Node.js or Johnny-Five, just have Meta_Javascript open.

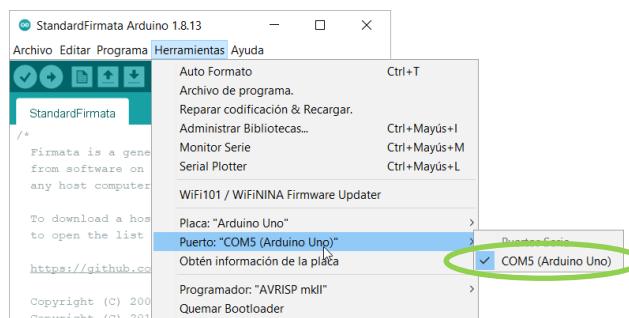


4.1. Installation of the Firmata library on an Arduino board

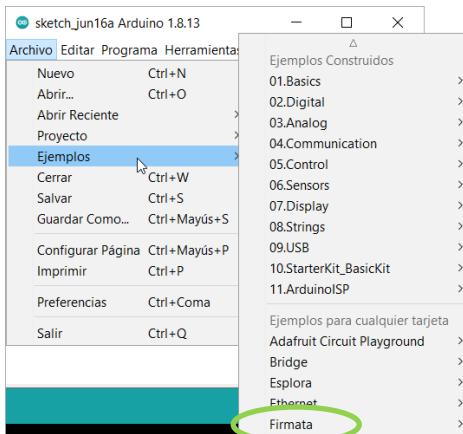
To load the Firmata library on the board the first step is to open the Arduino IDE and choose the Arduino board to which the Firmata library will be loaded.



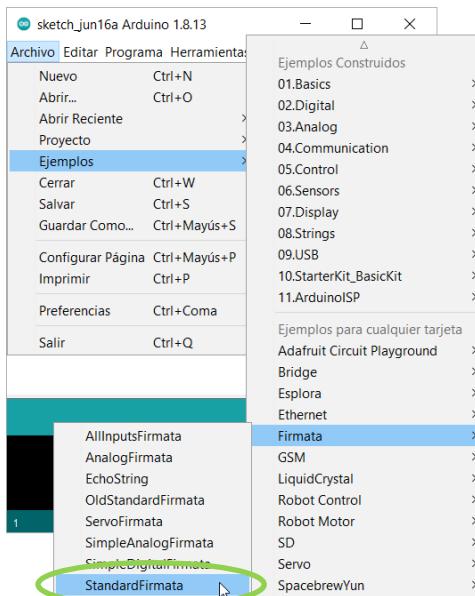
Then you must choose the port to which the Arduino board is connected.



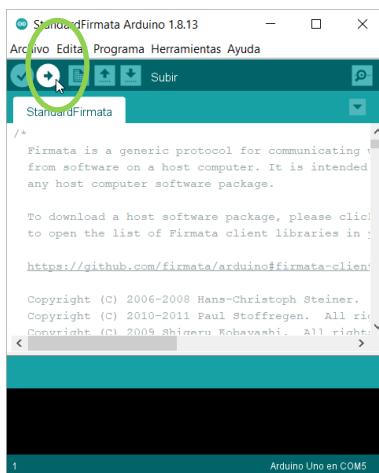
Then you need to open the Firmata library **examples**, which are accessed from the **file** menu.



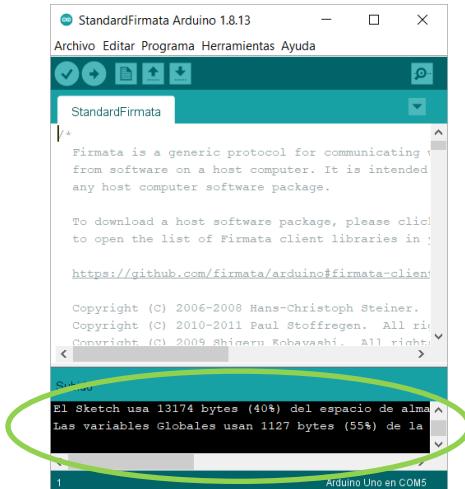
Then you must click on the **Firmata** option to see all the sample files of this library, and open the example: **StandardFirmata**



With the example open, click on the upload icon.



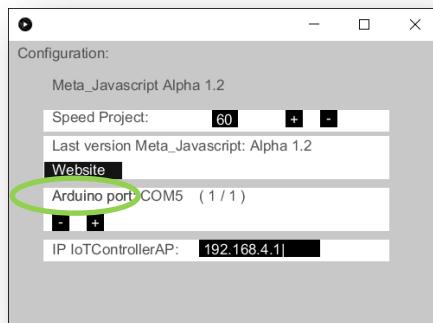
When you finish uploading the library to the Arduino board you should see something like this:



The next step is to open Meta_Javascript and select the port to which the Arduino board is connected. For this, you must click on the configuration icon to open the configuration window.



In the window that opens you can see that the fourth option is: Arduino port.



To select the connection port with the Arduino board, click on the - or + buttons. When the Arduino is connected to the USB port, the first value displayed is the port name, and right next to it is a parenthesis with two values. The first value indicates the current port number and the value followed by / is the total ports. If there is no connected device, the port name is not shown, and the port numbers would say: (0/0).

4.2. Arduino instructions

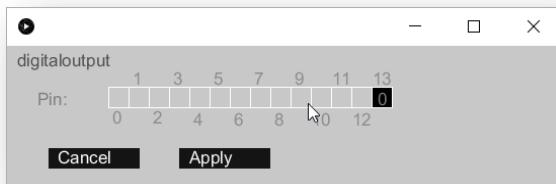
The Arduino instructions that can be used with Meta_Processing Alpha 1.1 are: **digitaloutput**, **digitalinput**, **analoginput**, and **servo**. The following will explain how to use each of them.

4.2.1. **digitaloutput**

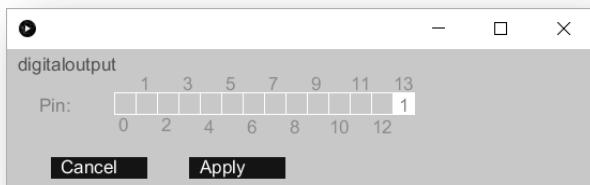
The **digitaloutput** instruction is used to turn on or off a specific pin on the Arduino board. If a pin is turn on it

means that this pin will supply a voltage greater than 3 volts to the device that is connected to that pin. On the contrary, turning off a pin means that this pin will supply a voltage less than 1.5 volts to the device that is connected to that pin.

To use this instruction, click on the box corresponding to the pin of the Arduino that you want to turn on or off. When the background of the box is black that pin is turned off, if the color is white that pin is turned on. Example pin 13 off:



Example pin 13 on:



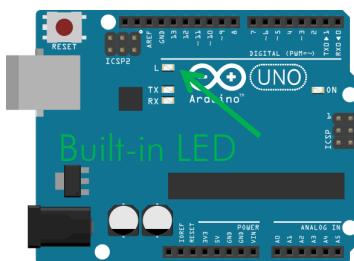
If you wanted to create a project that turns pin 13 on when the mouse is clicked and turns off when any key is pressed, the following code would need to be added in the **Mouse** tab:

```
1 digitaloutput( 13, 1 )
2 servo( 7, 0 )
```

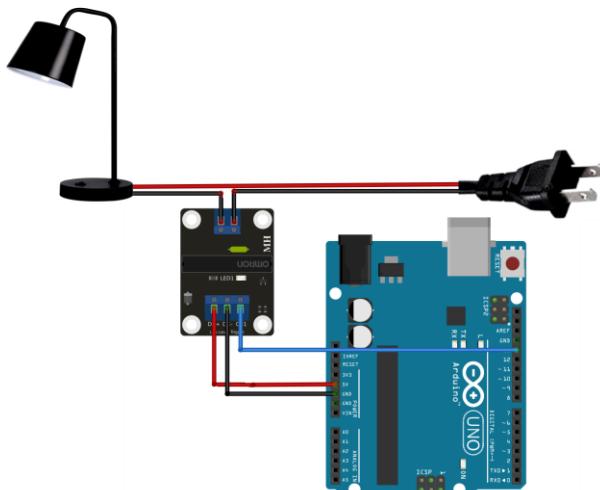
And in the **Keyboard** tab the following code would need to be added:

```
1 digitaloutput( 13, 0 )
2 servo( 7, 180 )
```

To use the previous code it would not be necessary to build any circuit with the Arduino, since there is a Led that is built-in in all Arduino Uno boards and by default is connected to pin 13.

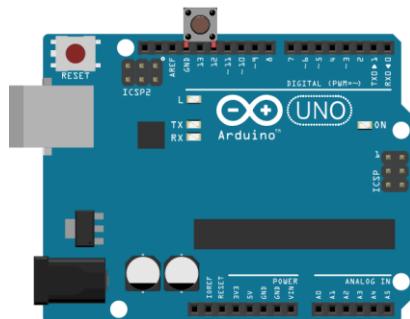


Below is a circuit that could be built to turn on and off a lamp connected to 110 volts, using the same code. This would require a solid state relay module for the Arduino.



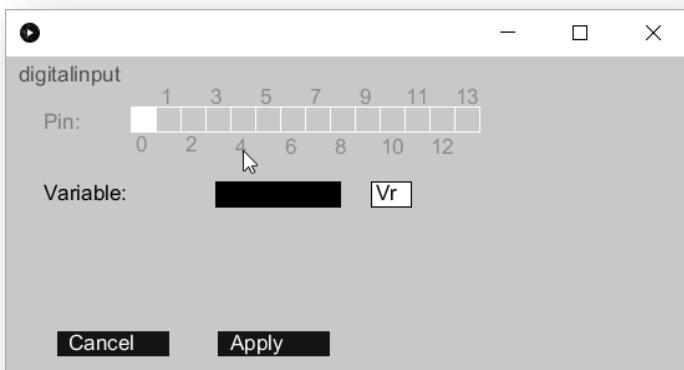
4.2.2. **digitalinput**

The **digitalinput** instruction is used to read the status of a digital pin on the Arduino board. This instruction was created to be able to connect a push-button directly to any digital pin, without needing to build an additional circuit (Pull Up). One of the push-button terminals must be connected to the GND pin and the other to the digital pin that you want to use. In this example pin 12:



This instruction stores in a variable the state of the digital pin, if it is 1 it means that the button is pressed, if on the contrary it is 0 it means that the button is not being pressed.

To use this instruction, you must click on the box corresponding to the pin of the Arduino that you want to read and in the second box, you must select the variable that will store the state of the pin.



If you wanted to create a project with a button that changes the background color to green each time the button is pressed, and when the button is not pressed the background color turns red, the following code would need to be added in the **Principal** tab:

```
1 digitalinput ( 12, x )
2 if ( x = 1 )
3   background ( #FF15ED38 ) █
4 Else background ( #FFDC3338 ) █
```

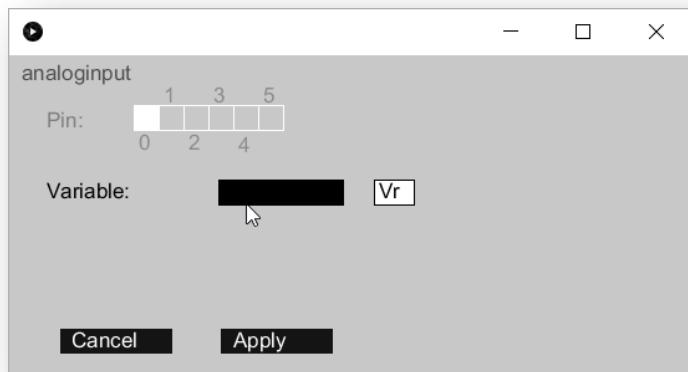
To use this example code, you must have one button terminal connected to the GND pin of the Arduino and the other terminal to pin 12 of the Arduino.

4.2.3. analoginput

The **analoginput** instruction is used to read the status of an analog pin on the Arduino board. The range of values that can be obtained from this reading is between 0 and 1023.

This instruction stores the state of the analog pin in a variable, where 0 would be equivalent to a reading of a voltage less than 0.0049 volts and 1023 to a reading of a voltage of 5 volts.

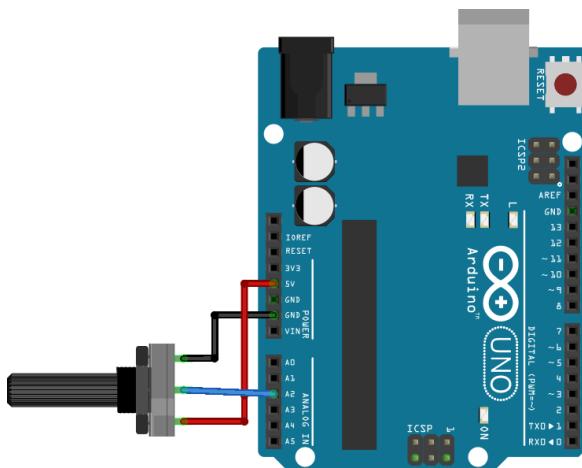
To use this instruction, you must click on the box corresponding to the analog pin of the Arduino that you want to read and in the second box, you must select the variable that will store the state of the pin.



If you wanted to create a project with a potentiometer connected to analog pin 2 so when it turns, the background color changes, the following code would need to be added in the **Principal** tab

```
1  analoginput( 2, giro )
2  if ( giro > 100 )
3    background( 218 )
4  if ( giro > 400 )
5    background( 158 )
6  if ( giro > 600 )
7    background( 87 )
```

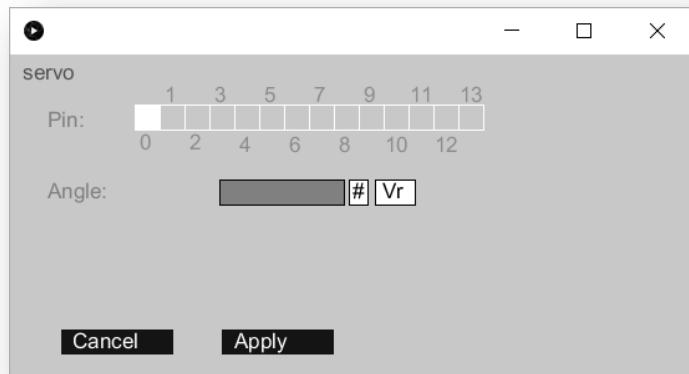
The circuit necessary to use the previous example code would be the following:



4.2.4. servo

The **servo** instruction is used to control the angle at which you want to rotate a servo motor connected to one of the digital pins of the Arduino board.

To use this instruction, click on the box corresponding to the pin of the Arduino to which the servomotor is connected, and in the second box, select the value of the angle to which you want to rotate the servomotor.



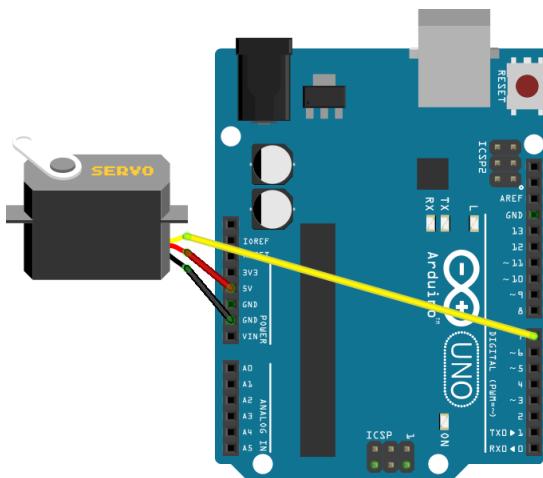
If you want to create a project with a servo motor connected to digital pin 7 so that it rotates in one direction when the mouse is clicked and it rotates in the opposite direction when any key is pressed, the following code would need to be added in the **Mouse** tab:

```
1 servo(7,0)
```

And in the **Keyboard** tab the following code would need to be added:

```
1 servo(7,180)
```

The circuit necessary to use the previous example code would be the following:



5. ESP BOARD WITH META_JAVASCRIPT

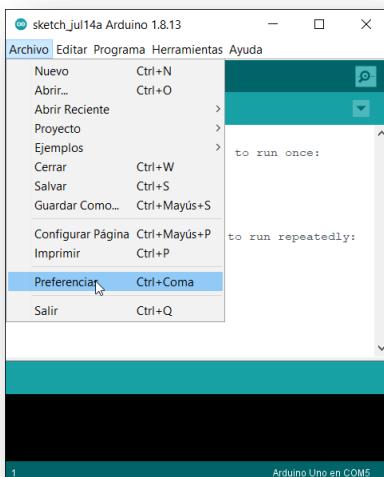
To use an ESP board with Meta_Javascript requires that the ESP be loaded with the IoTControllerAP library². This library allows the ESP board to function as an Access Point named IoTController (no password required). In this way, direct WiFi communication with the board can be established and several of its main functions can be controlled: turning pins on and off, digital and analog reading of a pin, and control of servos. You do not need to install Node.js or Johnny-Five, just have Meta_Javascript open.



² <https://github.com/hiteclab/IoTControllerAP>

5.1. Installing the IoTControllerAP library on an ESP board

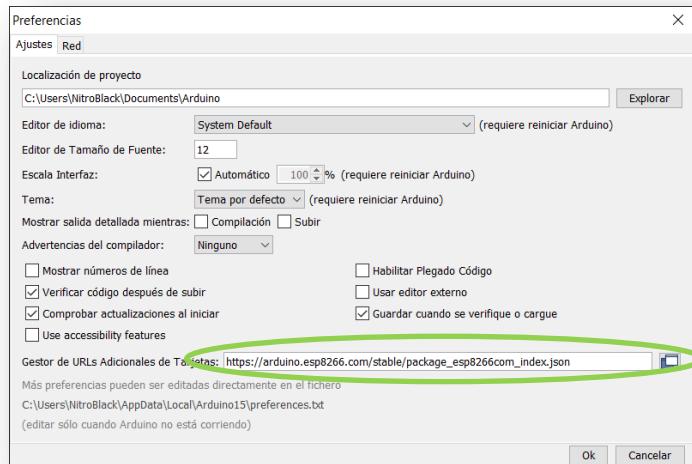
To load the IoTControllerAP library on the ESP board, the first step is to open the Arduino IDE and install the driver for these boards. For this it is necessary to click on the **Preferences** option within the **File** menu.



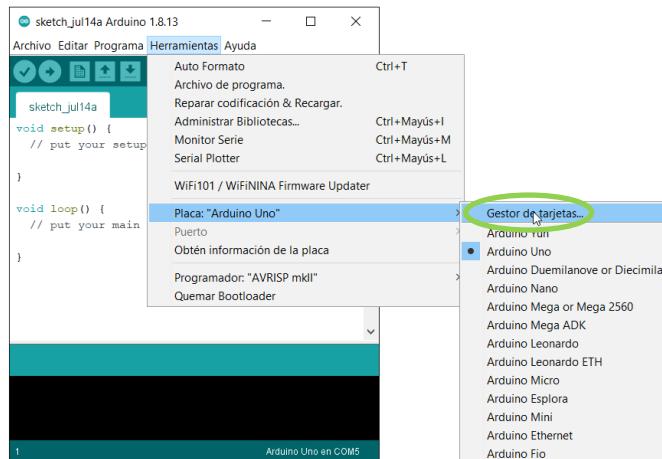
In the window that opens, you must add in the option **Additional Board URL Manager**, the line:

https://arduino.esp8266.com/stable/package_esp8266com_index.json

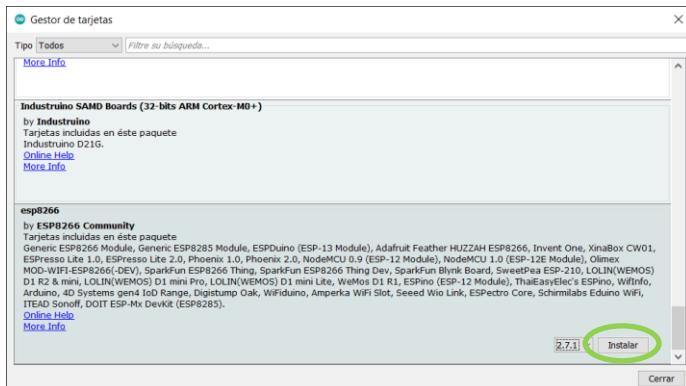
As it's shown in the following:



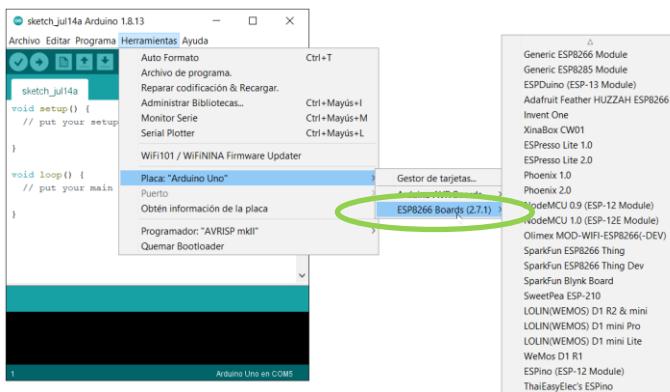
Then you must click on the **Board** option in the **Tools** menu, in the menu that opens you must click on the **Board manager** option:



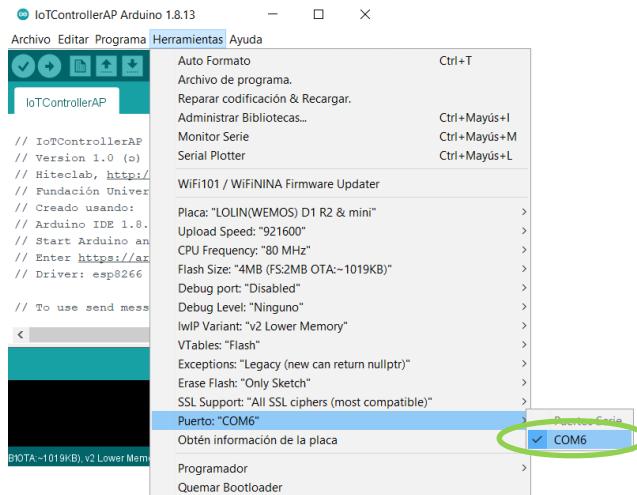
In the window that opens, search for the driver for the **esp8266** board, select version **2.7.1** and click over the **Install** button.



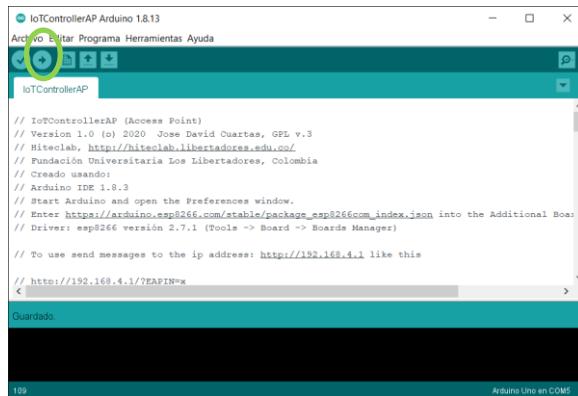
Once this is done, you can choose the ESP board to which the **IoTControllerAP** library will be loaded.



Then you must choose the port to which the ESP board is connected.



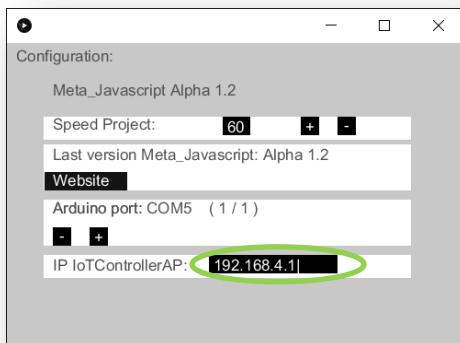
Next, download the **IoTControllerAP** library from <https://github.com/hiteclab/IoTControllerAP>, then open the **IoTControllerAP.ino** code, and click on the **upload** icon.



Once the IoTControllerAP controller is loaded on the ESP board, your computer must be connected via WiFi to the ESP board. Search the Access Point **IoTController** and click connect (No password required). Then you must open Meta_Javascript and click on the **configuration** icon and in the configuration window confirm that the IP address of the ESP board is 192.168.4.1.



In the window that opens you can see that the fifth option is: **IP IoTControllerAP**.



There you can change the IP address of the board with the IoTControllerAP library, by default it is: 192.168.4.1

5.2. IoTControllerAP Instructions

The IoTControllerAP instructions that can be used with Meta_Processing Alpha 1.2 are: digitaloutput, digi-

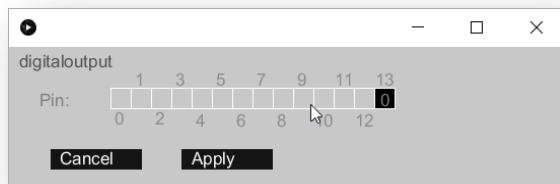
talinput, analoginput, and servo. The following will explain how to use each of them.

5.2.1. digitaloutput

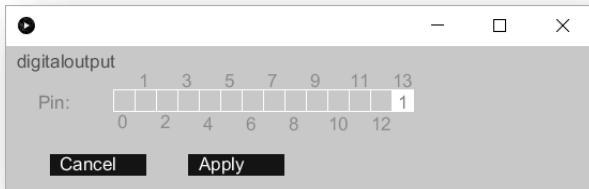
The **digitaloutput** instruction is used to turn on or off a specific pin on the ESP board. If a pin lights up, this pin will supply a voltage greater than 3 volts to the device that is connected to that pin. Conversely, if a pin is turned off, it means that this pin will supply a voltage less than 1.5 volts to the device that is connected to that pin.

To use this instruction, click on the box corresponding to the pin of the ESP you want to turn on or off. When the background of the box is black then that pin is turned off, if the color is white then that pin is turned on.

Example pin 13 off:



Example pin 13 on:



If you wanted to create a project that turns on pin 12 when the mouse is clicked and turns off when any key is pressed, the following code should be added in the **Mouse** tab:

```
1 digitaloutput ( 12, 1 ) IoTControllerAP
```

And in the **Keyboard** tab the following code should be added:

```
1 digitaloutput ( 12, 0 ) IoTControllerAP
```

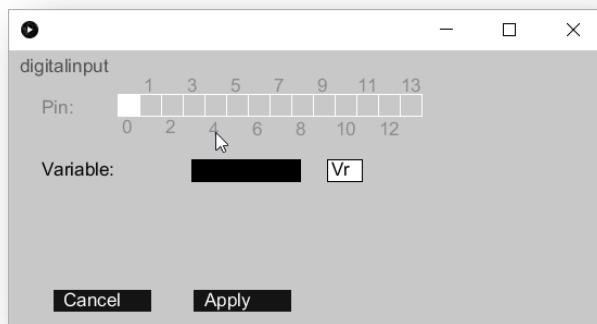
To use the previous code it would be necessary to connect a LED to pin 12 of the ESP board.

5.2.2. digitalinput

The **digitalinput** instruction is used to read the status of a digital pin on the ESP board. This instruction was created to be able to connect a button directly to any digital pin, without needing to build an additional circuit (Pull Up). One of the push-button terminals must be connected to the GND pin and the other to the digital pin that you want to use.

This instruction stores in a variable the state value of the digital pin, if is 1 it means that the button is pressed, if on the contrary is 0 it means that the button is not being pressed.

To use this instruction, you must click on the box corresponding to the ESP pin you want to read and in the second box, you must select the variable that will store the state value of the pin.



If you wanted to create a project with a button that changes the background color to green each time the button is pressed, and when the button is not pressed, the background color turns red, the following code should be added in the **Principal** tab:

```
1 digitalinput ( 4, esp ) IoTControllerAP
2 if ( esp > 0 )
3   background ( #FF52D938 )█
4 Else background ( #FFD24638 )█
```

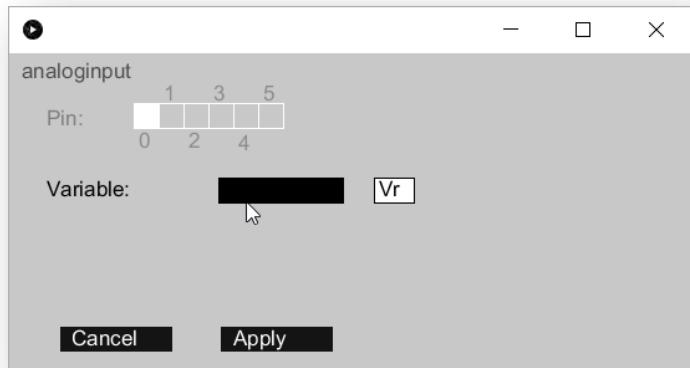
To use this example code, you must have one terminal of the button connected to the GND pin of the ESP and the other terminal to pin 4 of the ESP board.

5.2.3. analoginput

The **analoginput** instruction is used to read the status of an analog pin on the ESP board. The range of values that can be obtained from this reading is between 0 and 1023.

This instruction stores the state of the analog pin in a variable, where 0 would be equivalent to a reading of a voltage less than 0.0049 volts and 1023 to a reading of a voltage of 5 volts.

To use this instruction, you must click on the box corresponding to the analog pin of the ESP that you want to read and in the second box, you must select the variable that will store the state of the pin.



If you want to create a project with a potentiometer connected to the analog pin 0 and that when it turns, the background color changes, in the **Principal** tab you should add the following code:

```
1  analoginput( 0, giro ) IoTControllerAP
2  if ( giro > 100 )
3    background( 218 )■
4  if ( giro > 400 )
5    background( 158 )■
6  if ( giro > 600 )
7    background( 87 )■
```

5.2.4. servo

The **servo** instruction is used to control the angle at which you want to rotate a servo motor connected to any of the digital pins on the ESP board.

To use this instruction, click on the box corresponding to the pin of the ESP to which the servo motor is connected, and in the second box, select the value of the angle at which you want to rotate the servo motor.



If you wanted to create a project with a servo motor connected to digital pin 4 so that it rotates in one direction when the mouse is clicked and it rotates in the opposite direction when any key is pressed, the following code could be used.

In the **Mouse** tab should be added:

```
1 servo( 4, 0 ) IoTControllerAP
```

And in the **Keyboard** tab should be added:

```
1 servo( 4, 180 ) IoTControllerAP
```

6. CODE EXAMPLES WITH META_JAVASCRIPT

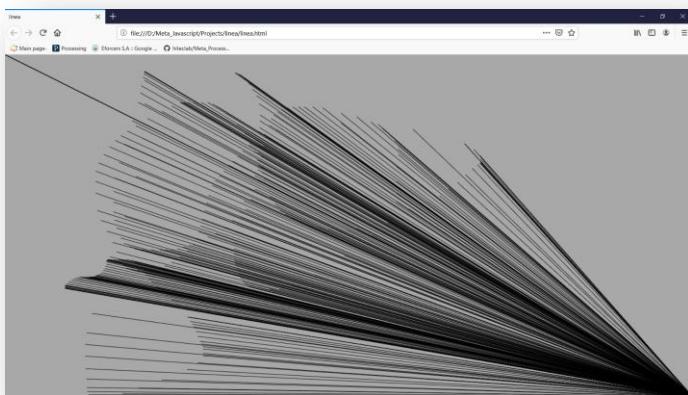
Below are a number of examples of how to use Meta_Javascript. The first of them shows how to make abstract line drawings. The second allows us to experiment with the keyboard function. The third allows us to draw circles on the screen when pressing the mouse. The fourth is an example of how an animation can be created. The fifth shows us three ways to program a piano. And the last one shows us how to create a simple mini game.

6.1. Abstract line drawings

The following code must be written on the **Principal** tab, and when the code is executed, the mouse cursor must be moved to generate the abstract drawings with lines.

```
1 line ( ancho , alto , ratonX , ratonY )
```

When executed, it will look like this in the browser:



6.2. Basic example with the Keyboard

The following code must be written on the **Keyboard** tab, so it will be executed at the moment any key is pressed.

```
1 background (#FF0B3B99)
2 textSize ( 180 )
3 text ( Hello World! , 500 , 500 )
```

The idea is that when you press any key, the screen turns blue and a white text appears on the screen.



Hello World!

6.3. Basic example with the Mouse

The following code should be written on the **Mouse** tab so it will be executed at the moment any of the mouse buttons are pressed.

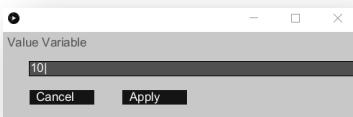
```
1 noborder
2 fill ( #FFFDD738 ) █
3 ellipse ( ratonX , ratonY , 80 , 80 )
```

The idea is that after pressing any of the mouse buttons yellow circles will be drawn on the screen.

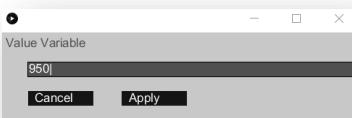


6.4. Animation

This is an example of how you can make an animation in Meta_Javascript. First, you need to create two variables, one called **px** and the other called **py**. The **px** variable must be initialized, with the value 10.



And the variable **py** must be initialized with the value 950.



Then the following code should be added in the **Principal** tab:

```
1 background ( px )
2 add ( px , 0.5 )
3 subtract ( py , 0.5 )
4 fill ( #FFF2E13B )
5 ellipse ( 200 , py , 200 , 200 )
6 fill ( #FF0B259A )
7 rectangle ( 0 , 800 , ancho , 400 )
```

After clicking on the **run** icon you will see that the animation begins by recreating a night scene in the sea, and it will slowly dawn until the firmament is fully illuminated.



6.5. Piano

Below are three examples of how to make a piano in Meta_Javascript. The first is a simple piano, the second is a piano that also changes the color of the screen, and the third is a piano that changes the color of the screen and displays text with the note being played.

6.5.1. Simple piano

To program this piano, the following code must be added in the **Keyboard** tab:

```
1 if ( tecla = 'a' )
2   playnote (C4)
3 if ( tecla = 's' )
4   playnote (D4)
5 if ( tecla = 'd' )
6   playnote (E4)
7 if ( tecla = 'f' )
8   playnote (F4)
9 if ( tecla = 'g' )
10  playnote (G4)
11 if ( tecla = 'h' )
12  playnote (A4)
13 if ( tecla = 'j' )
14  playnote (B4)
```

With this code you can play the 7 musical notes using the keys a, s, d, f, g, h, j. To make it work properly the keyboard cannot be in uppercase mode.

6.5.2. Piano colors

To program this piano, the following code must be added in the **Keyboard** tab:

```
1 if ( tecla = 'a' )
2   playnote ( C4 )
3   background ( #FF2AD538 )■
4 if ( tecla = 's' )
5   playnote ( D4 )
6   background ( #FFC72538 )■
7 if ( tecla = 'd' )
8   playnote ( E4 )
9   background ( #FF3348AE )■
10 if ( tecla = 'f' )
11   playnote ( F4 )
12   background ( #FFFCD138 )■
13 if ( tecla = 'g' )
14   playnote ( G4 )
   background ( #FFF48C0E )■
if ( tecla = 'h' )
  playnote ( A4 )
  background ( #FFEF1EC8 )■
if ( tecla = 'j' )
  playnote ( B4 )
  background ( #FF57938D )■
```

With this code you can play the 7 musical notes using the keys a, s, d, f, g, h, j. To make it work properly the keyboard cannot be in uppercase mode.

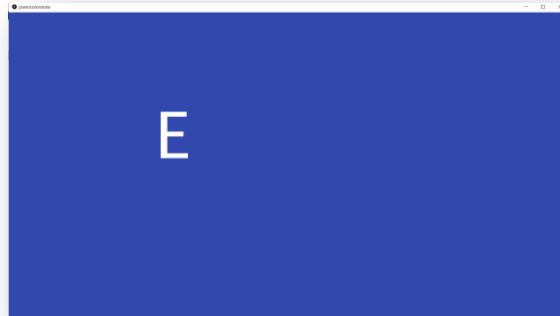
6.5.3. Piano colors and notes on screen

To program this piano, the following code must be added in the **Keyboard** tab:

```
1  textsize ( 220 )
2  if ( tecla = 'a' )
3    playnote ( C4 )
4    background ( #FF2AD538 )■
5    text ( C , 500 , 500 )
6  if ( tecla = 's' )
7    playnote ( D4 )
8    background ( #FFC72538 )■
9    text ( D , 500 , 500 )
10 if ( tecla = 'd' )
11   playnote ( E4 )
12   background ( #FF3348AE )■
13   text ( E , 500 , 500 )
14 if ( tecla = 'f' )
15   playnote ( F4 )
16   background ( #FFFCD138 )■
17   text ( F , 500 , 500 )
18 if ( tecla = 'g' )
19   playnote ( G4 )
20   background ( #FFF48C0E )■
21   text ( G , 500 , 500 )
22 if ( tecla = 'h' )
23   playnote ( A4 )
24   background ( #FFEF1EC8 )■
25   text ( A , 500 , 500 )
26 if ( tecla = 'j' )
27   playnote ( B4 )
28   background ( #FF68B0AO )■
29   text ( B , 500 , 500 )
```

With this code you can play the 7 musical notes using the keys a, s, d, f, g, h, j. To make it work properly the keyboard cannot be in uppercase mode.

When the piano is executed would look like this:



6.6. Mini Game

This is an example of how you can make a mini game in Meta_Javascript. The following code must be added in the **Principal** tab:

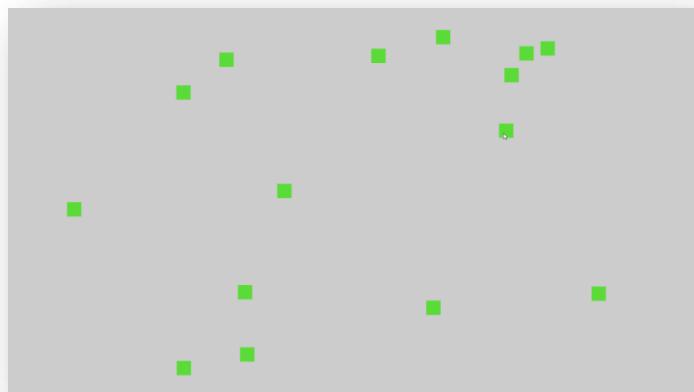
```
1 noborder
2 fill ( #FF5ADB38 )■
3 add ( contador , 0.1 )
4 if ( contador > 5 )
5   random ( px , 1900 )
6   random ( py , 1040 )
7   rectangle ( px , py , 40 , 40 )
8   assign ( contador , 0 )
```

And in the **Mouse** tab you must add the following code:

```
1 fill ( 205 )■
2 ellipse ( ratonX , ratonY , 100 , 100 )
3 playnote ( C4 )
```

The game consists of a series of green squares that appear randomly on the screen. The purpose of the game is to try to make disappear all the green squares by clicking on them.

When the game is executed would look like this:



References

Cuartas, J.D. (2014). Digitópolis I: Diseño de Aplicaciones Interactivas para Creativos y Comunicadores. Bogotá: Fundación Universitaria Los Libertadores.

Cuartas, J.D. (2017). Programar el mundo en el contexto de las tecnologías libres y las culturas Hacker-Maker. Caso de estudio: Hitec Lab. (Tesis doctoral). Doctorado en Diseño y Creación. Universidad de Caldas, Manizales.

Hitec Lab (2020). Hitec Lab Homepage. Retrieved from <http://hiteclab.libertadores.edu.co/>

Hitec Lab (2020). Meta_Javascript. Retrieved from https://github.com/hiteclab/Meta_Javascript

Hitec Lab (2020). Meta_Javascript. Retrieved from https://github.com/hiteclab/Meta_Javascript

Libertadores, L. (2019). Institución Universitaria Los Libertadores. Retrieved from <http://www.ulibertadores.edu.co/>

Processing. (2019). Processing. Retrieved from <http://www.processing.org/>

