DIDAC-PROG

CARTESIA

APPLICATION USER MANUAL

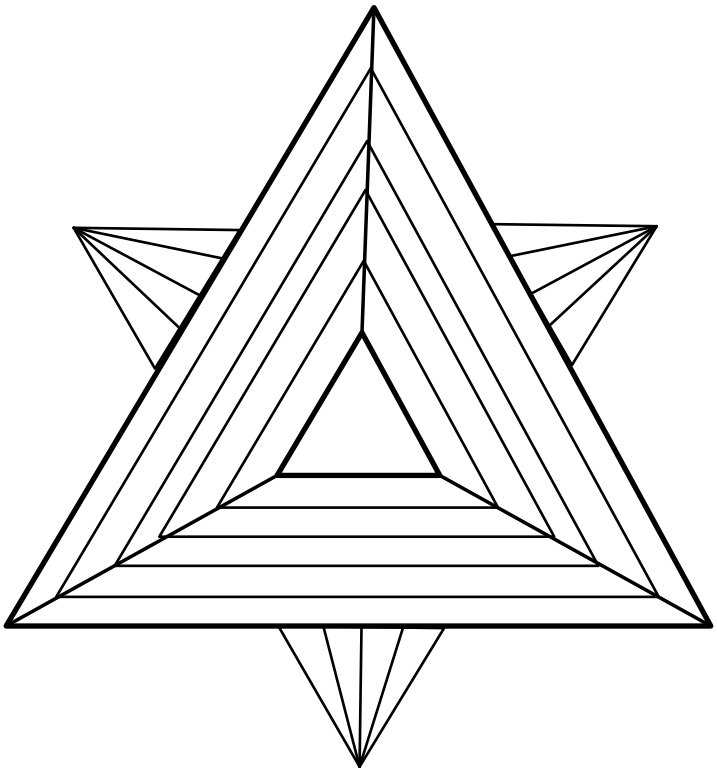
Learn to program with Didac-Prog Cartesia.

For children and not so children.

Written by Mario Rodríguez Rancel

2020 version update 001

What is this?



Find it out

Upon the subject of education... I can only say
that I view it as the most important subject
which we as a people may be engaged in.

Abraham Lincoln (1808-1865)

CONTENTS

1. INTRODUCTION

Didac-Prog Cartesia is an **educational and free** web application conceived for learning computer programming and / or mathematics for children from 10 years of age, although nothing prevents it from being used at younger ages (for example, from 7- 8 years) or later, and even adults.

This manual is prepared so that it can be read by any person, child or adult, and allows you to learn how to use the application since it's intended as a tutorial that covers all the possibilities of Didac-Prog Cartesia with explanations and examples of all of them. However, children may require specific adaptations of the manual or some additional explanations depending on their age. In the final part of the manual there is an annex specifically aimed for teachers (within teachers we will include trainers, parents or guardians, etc. That is, anyone who uses the application to teach another).

Didac-Prog Cartesia is prepared for **bilingual Spanish-English teaching** and can be configured at the user's wish to work entirely in Spanish or totally in English.

Starting to use Didac-Prog Cartesia is as simple as downloading the file (called cartesia2020_u01_en.zip or similar) with the application from the corresponding link at aprenderaprogramar.com, unzipping it on our computer or tablet and double-clicking on the index.html file. The space it occupies is minimal (<5 Mb). From then on you can use the application whenever you want without needing an internet connection. If you have problems opening the application, consult the installation manual.

When updates to the application are published, you can download them from this website. If you prefer, you can use the application online from aprenderaprogramar.com, without downloading it.

2. HOW TO GET HELP OR CONTACT THE CREATORS?

The application, associated resources (e. g. manuals), the frequent asked questions (FAQ) and the projects and contributions of the community that uses Didac-Prog Cartesia can be found in the section dedicated to the application on the web aprenderaprogramar.com, where you can upload your projects to share them and consult projects created by other users.

In the web forums there are questions and answers about the application. If with the indicated resources you cannot solve a problem, you can write your query in the forums, from where, as far as possible, expert or not so expert users, and even the creators of the application, try to answer.

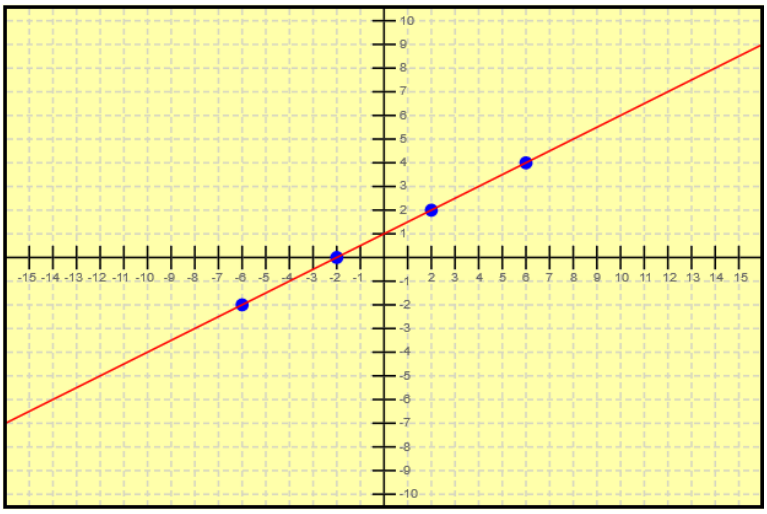
If you want to contact the creators of Didac-Prog Cartesia to make suggestions or comments, relate your experience, make a contribution in the form of an informative or research article to be published and available to the community, or for anything else, you can do it by writing to contacto@aprenderaprogramar.com

3. WHAT IS DIDAC-PROG-CARTESIA FOR?

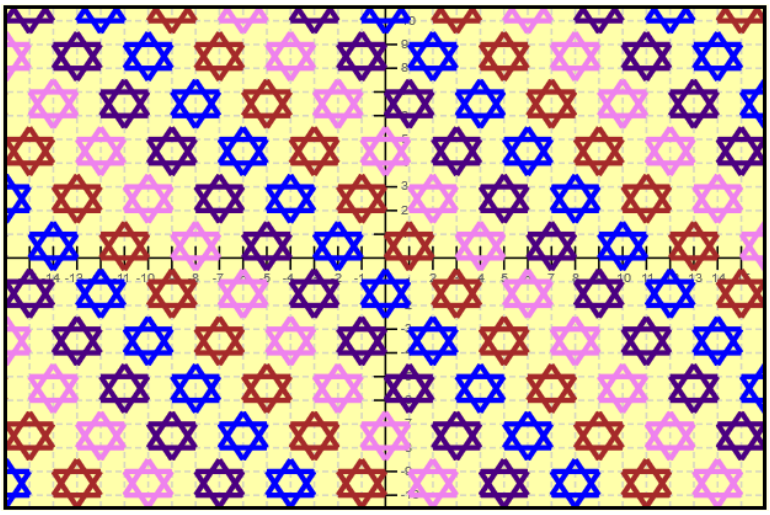
With Didac-Prog Cartesia we will learn how to define instructions to the computer so that it executes them. This is what we call *computer programming*, *software programming* or just *programming*. Sometimes it is also said to be "defining algorithms" where algorithm refers to **a sequence of instructions** or steps to be given. Programming can have a wide variety of purposes, such as making an online purchase, storing and consulting data, editing photographs, defining how a robot should act and almost anything we can think of. Smartphones and computers work because they have many instructions in their memory that define what to do. Today programming is used not only by computer professionals, but also engineers, scientists, economists, etc.

With Didac-Prog Cartesia we can order the computer to create drawings or graphic representations of what we want taking points and lines as basic shapes. In addition to points, line segments, rays and lines we can also represent geometric figures, mathematical functions, artistic drawings or anything we can imagine in a canvas.

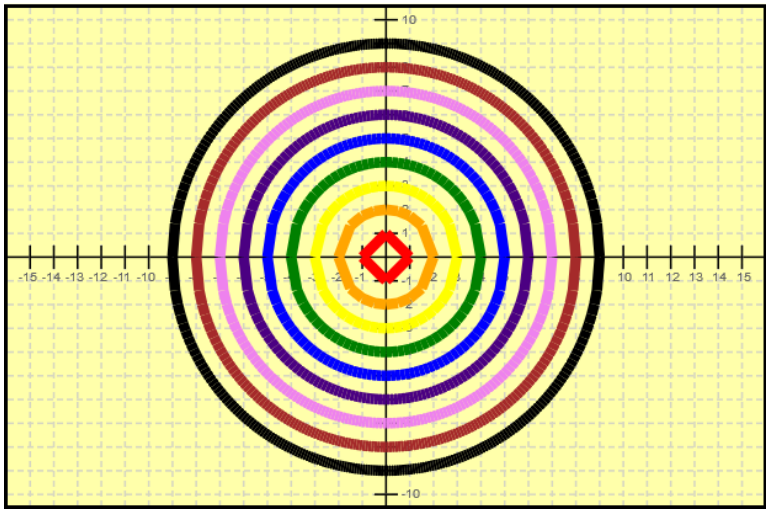
As Didac-Prog Cartesia approaches to programming based on mathematics, it can also be used to **learn mathematics**. Let's look at some examples of what Cartesia can do.



Example 1 of drawing made with Didac-Prog Cartesia. In this case, several points of a line are represented in blue and the line is drawn in red. Cartesia does not have a specific instruction to draw infinite lines, but if we take two very distant points and join them the appearance is to have drawn the infinite line.



Example 2 of drawing made with Didac-Prog Cartesia. In this case stars of David are represented formed by two triangles each and are drawn repeatedly throughout the screen alternating different colours (pattern of shapes and colours).



Example 3 of drawing made with Didac-Prog Cartesia. Although the application does not have any instructions to draw curves explicitly, they can be created by joining points or joining small segments.

Unlike applications that allow the user to draw as Paint, where we can choose tools such as a brush and draw, Didac-Prog Cartesia does not allow drawing directly. What it allows is to define a series of written instructions (the **code** or **program**) about what the computer must do, for the latter to build the drawing.

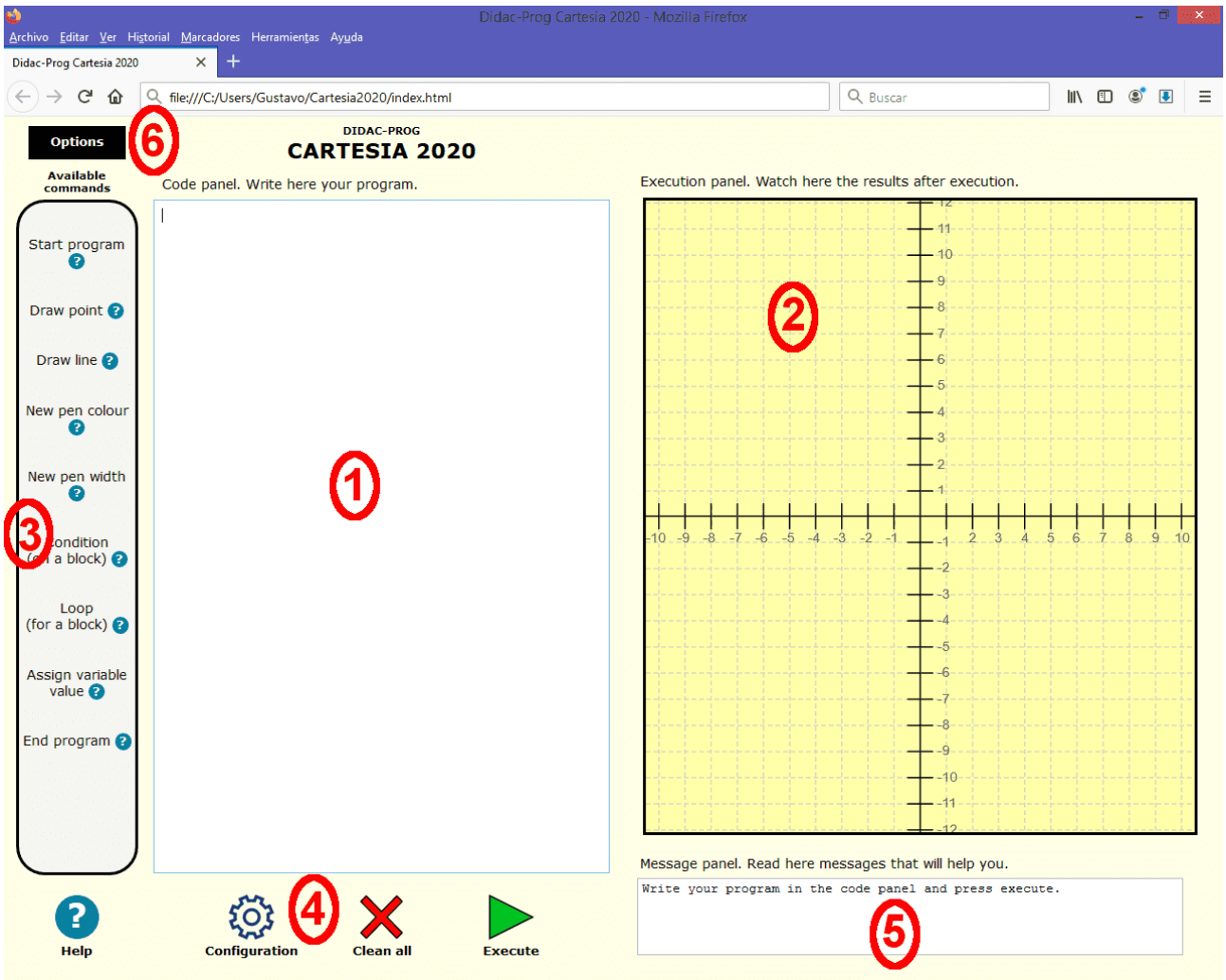
Didac-Prog Cartesia is an educational environment but can serve as a preliminary step to the use of more advanced or professional programming environments.

Once the instructions (the code) have been defined, we can tell the computer to execute it and see the result of that execution. As the application serves to give instructions on what to draw, many times we will talk about the **drawing pen** as if the computer were a drawing robot using a drawing pen. Thus, we can define where to draw, whether to draw a thick or thin line, or whether to use one colour or another. If we change the colour in which to draw from red to blue, we will say that we have changed the colour of the drawing pencil.

If we save the code, we can execute it later at any time we want, always obtaining the same result if we do not make modifications.

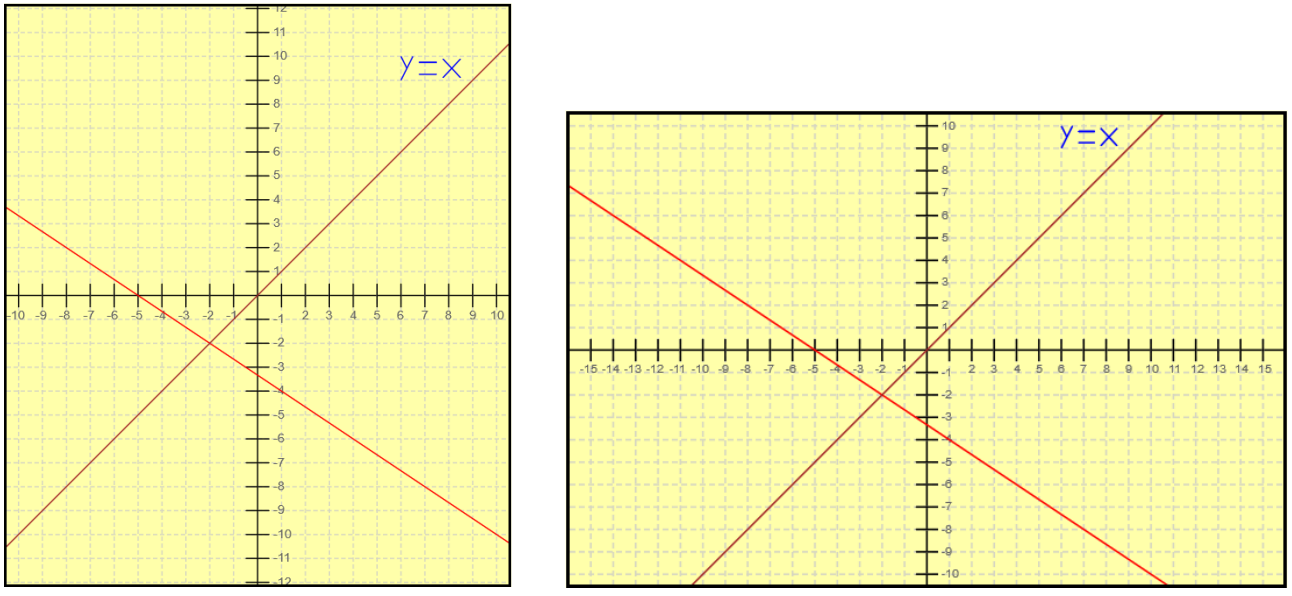
4. STARTING WITH DIDAC-PROG CARTESIA: MENU, PANELS AND EXECUTE

If you have opened the application correctly you will have found a screen similar to the following (on the screen we have indicated with numbers the different parts that can be differentiated and that we will list below):



Nº	Name	What is it for
1	**Code panel**	Here we will write our program (the instructions that we want the computer to run). When we open from a file a project created by us or by someone else, it will be here where we will see the code.
2	**Drawing panel**	Contains coordinate axes. Here we will see the graphic result after executing the code after pressing the "Execute" button. Note that you may not see anything for several reasons: that our code has errors, that the drawing is outside the visible area, or other reasons.
3	**Command panel**	Here is a list of the 9 commands or instructions available in the Didac-Prog Cartesia programming language. By double-clicking on any of them, they will be inserted into the code panel (in the absence of completing with specific data).
4	**Control panel**	In this area we have a few buttons: Configuration, Clean All, Execute and Help.
5	**Message panel**	When we press the Execute button it is advisable to pay attention to the indications that appear in this panel, where we will be informed if the execution has been correct, if there are errors, warnings, etc.
6	**Options menu**	Hovering over this menu displays different options: Open Project, Save Project, Open Example, and Undo Clean.

Among the buttons on the control panel is one that is especially important: the **Execute button**, because it is the one that will allow us to see the results generated by a code that we have written in the code panel. Throughout this manual we will see with examples the use of all these elements. Cartesia elements adapt to the screen size of the device where they are displayed (computer, tablet, etc.) as long as the screen has a minimal size. If the screen is too small the application may not display correctly. One thing to keep in mind is that the Cartesia code panel is not always equal horizontally and vertically as it is able to adapt to the screen size. This can make the drawings created look slightly different depending on the device you're using. To understand this, we show an example of a drawing created with Cartesia seen on two different devices: the code is the same and the drawing is the same, but the display is slightly different.



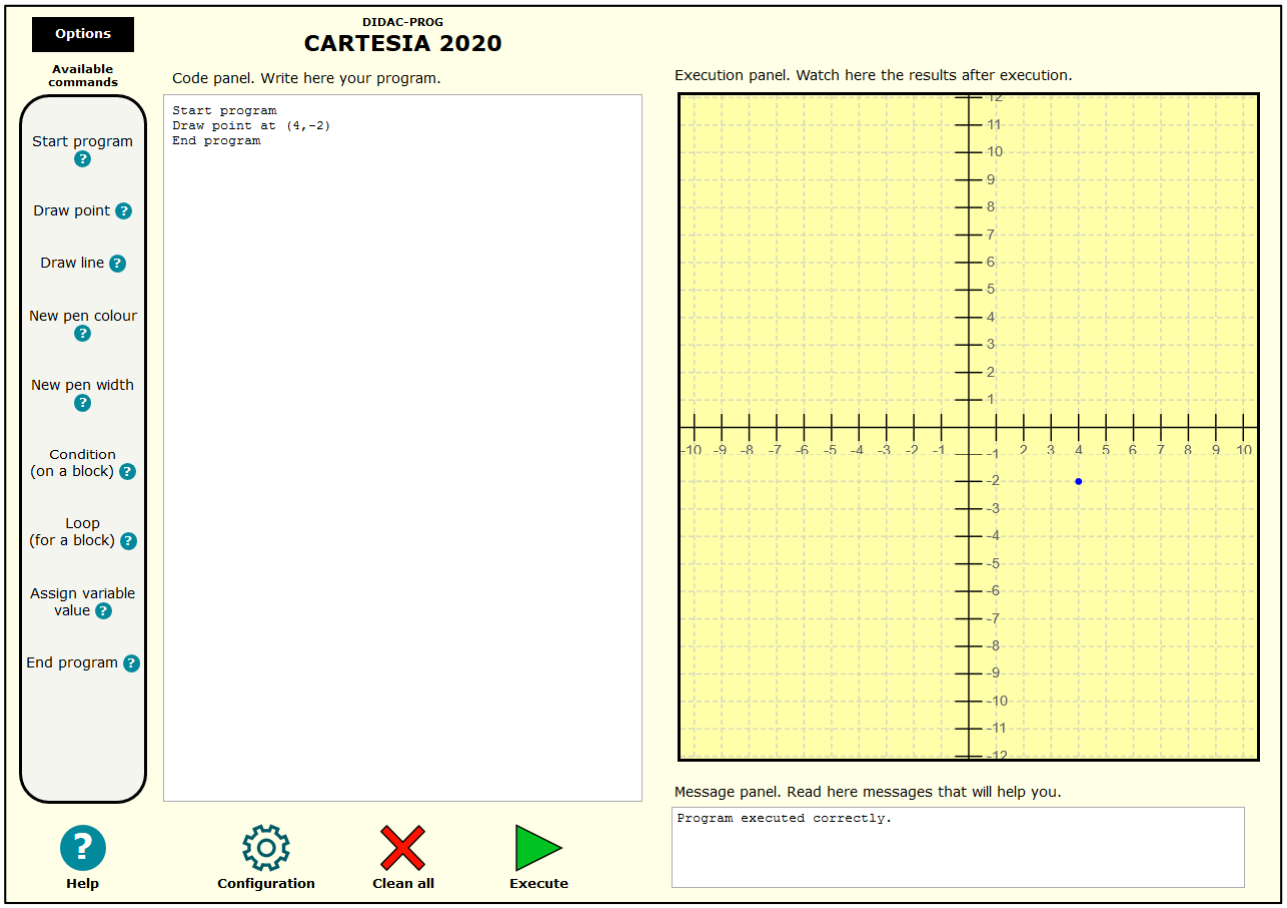
Visualization on different devices of the result of the same program created with Cartesia. It can be seen that in one case the axes have approximately the same number of divisions, while in the other there are more horizontal than vertical divisions. The visible area is different, hence y=x appears separate in one case from the top edge and in another nearly pasted to it.

5. START PROGRAM, DRAW POINT AND END PROGRAM COMMANDS

Let's create our first program with Cartesia. To do this, type the following in the **code panel**:

```
Start program
Draw point at (4,-2)
End program
```

Notice how the content of the message panel initially is "Write your program in the code panel and press Execute." Press the Execute button now. The result on the execution panel (which we also call drawing panel) should look like the following:



As you'll see, a point has appeared on the execution panel and the message "Program executed correctly" appeared in the message panel.

Didac-Prog Cartesia works with a Cartesian coordinate system that you may be familiar with. If you're not, let's briefly explain what it's all about.

Above the drawing panel we see a horizontal axis, also called an **abscissa axis** or simply **x-axis** divided into portions that allow us to position ourselves horizontally. So if a point has an x = 4 coordinate, it means that we have to move four units from the center to the right. If the value was negative, for example x = -4, we would have to move in the opposite direction: to the left.

On the other hand we have a vertical axis, also called **ordinate axis** or simply **y-axis** also divided and that allows us to position ourselves vertically. So if a point has coordinate y = -2 it means that we have to move two units down. If the value was positive, for example y = 2, we would have to move in the opposite direction: upwards.

The plane we draw on is called the Cartesian plane or **Cartesian coordinate system** and allows us to accurately position the location of any element we draw. These coordinate systems are very important in many aspects of daily life: in engineering (e.g. civil engineering plans), in architecture (e.g. building plans), in physics (e.g. to represent displacements), in computer applications (e.g. the screen of your computer or tablet), to represent a city in the form of a map (e.g. Google Maps), etc.

A point is defined by 2 values that we call x-coordinate or horizontal coordinate and y-coordinate or vertical coordinate. In the example we are looking at x = 4 and y = -2, and the point is (4, -2).

To place the point on the Cartesian plane we place in the center, we move 4 units to the right and then 2 units down, and in that place we draw the point.

In this example we have seen 3 Didac-Prog Cartesia commands:

- **Start program:** it must be the first command to appear in any program you type in Didac-Prog Cartesia. This command tells the computer to start executing the instructions. It cannot appear more than once in a program.
- **Draw point at (?,?):** it's the command to draw a point. You should replace the question marks with the values of the coordinates of the point, for example, draw point at (4, -2)
- **End program:** it must be the last command to appear in any program you type in Didac-Prog Cartesia. This command is used to instruct the computer to complete the execution of the instructions. It cannot appear more than once in a program.

Each Cartesia command must be written in one line (mandatory) and a program can have as many lines as you want.

6. MESSAGE PANEL. ERROR MESSAGES AND WARNING MESSAGES.

Errors in programs

The **message panel** is very useful as it informs us if we have made mistakes or alerts us of possible problems with our program. Write the following in the code panel (note that we have written *Stat* instead of *Start* and that we have not written the coordinates of the point in parentheses):

```
Stat program
Draw point at 4,-2
End program
```

Press the Execute button. Nothing appears on the drawing panel. Why? Because the program has **errors**. In this case we know that the errors are to have written *Stat* instead of *Start* and to have written the coordinates of the point without parentheses. But imagine that we have made these mistakes without realizing it and that we do not know why the program does not work.

To find out why nothing is displayed in the drawing panel, it will generally be useful to check the message panel to see if Cartesia informs us of what is happening.

If after pressing Execute we consult the message panel we will see a message similar to the following:

```
ERROR: First valid line in the program must begin with <<Start program>>. Please check code.
```

Now fix the first error: type *Start* correctly and press the Execute button again. As we still have an error because we have not added parentheses, we will see another message similar to the following:

```
ERROR: draw point statement needs a valid point in format (a,b). Please check code. Not valid line.
"draw point at 4,-2"
```

Cartesia is warning us that there is an error so you cannot run the program, and it is also indicating where the problem is.

Now type the parentheses and press the Execute button again. You will see that the point drawn now appears on the drawing panel, and that the message “Program executed correctly” appears in the message panel.

It is therefore important to consult the message panel to know if our code contains errors and to be able to correct them.

Cartesia warnings

Cartesia is not only able to detect and tell us if we have errors in our program, but also can give us useful **warnings**. Type this program in the code panel and click Execute:

```
Start program
Draw point at (400, -200)
End program
```

Nothing appears in the drawing panel. Why? If we consult the message panel we will verify that something similar to the following appears:

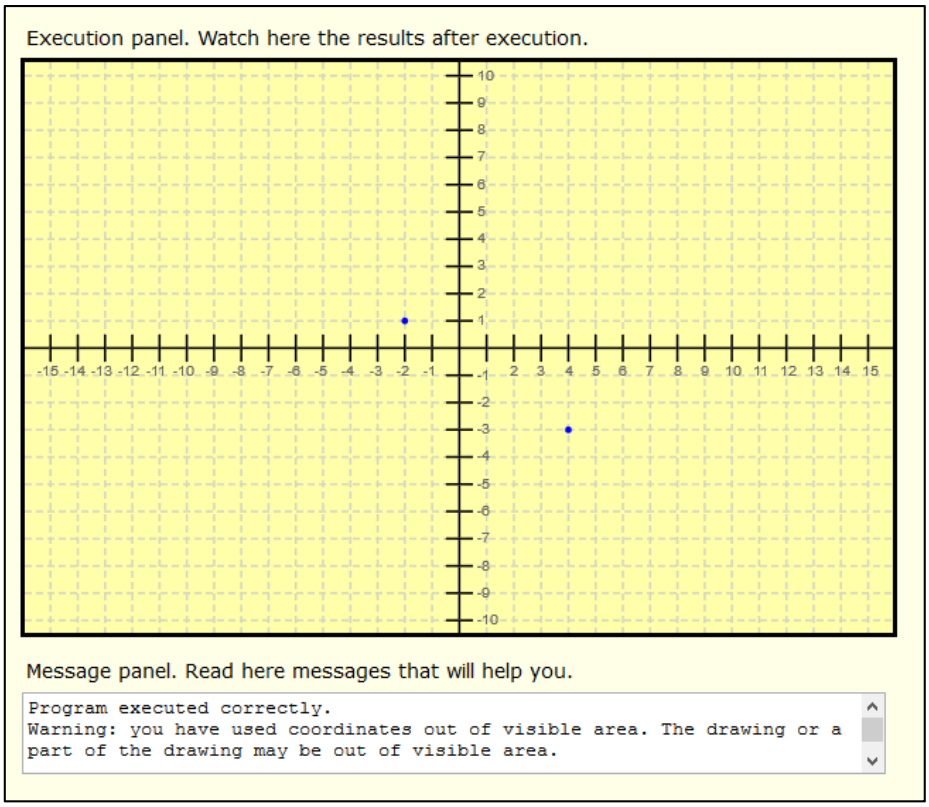
```
Program executed correctly.
Warning: you have used coordinates out of visible area. The drawing or a part of the drawing may be out of visible area.
```

Cartesia tells us that the program has been executed correctly, that is, we have not violated the required rules and writing form, and the computer has executed the program. However, it shows us a warning: we have used coordinates outside the visible area and the drawing may be **outside the visible area** in the drawing panel. That's right: if we start counting to the right to see where the value of x=400 is located we will check that that value is not visible. That is, it may have been drawn, but we do not see it because the coordinates are not within the coordinates shown in the drawing panel. In some cases there may be one part of the drawing visible and another part not visible.

Type this program in the code panel and click Execute:

```
Start program
Draw point at (4,-3)
Draw point at (400,-200)
Draw point at (-2,1)
End program
```

The result will look something like this:



Why are only 2 points displayed if we have indicated that 3 should be drawn? The message panel will give you the answer: there are coordinates outside the visible area and the drawing or part of it may not be seen. In this case of the 3 points that we have ordered to draw, 2 are seen in the drawing panel and 1 is outside the visible area.

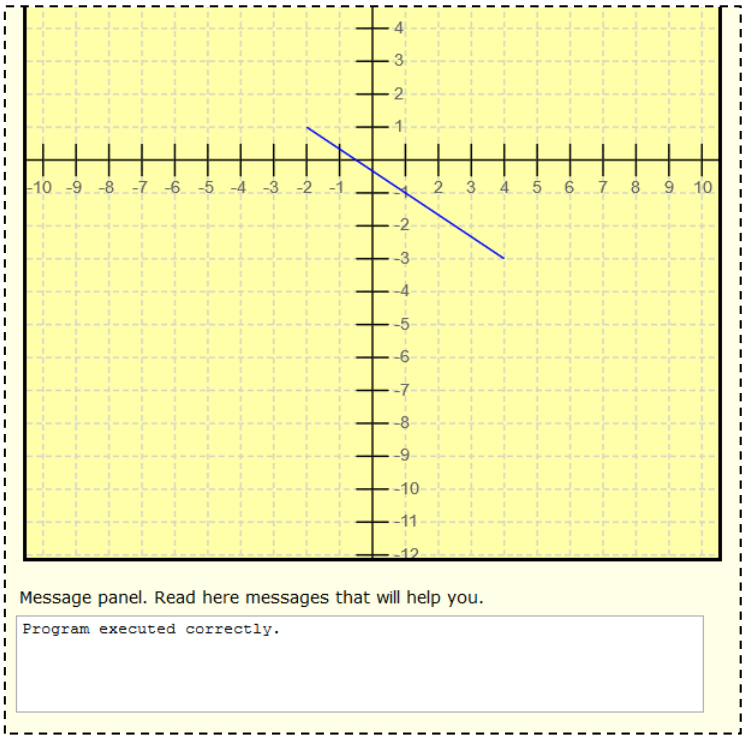
Keep in mind that a warning is not a mistake, and sometimes it may not even be a problem. In fact, we can create programs in Cartesia whose execution is accompanied by warnings and this is not a bad thing.

7. DRAW LINE COMMAND. USE OF DECIMALS AND FRACTIONS.

The command **Draw line from (?, ?) to (?, ?)** allow us to draw lines between two points that we must specify. Write in the code panel the following:

```
Start program
Draw line from (-2,1) to (4,-3)
End program
```

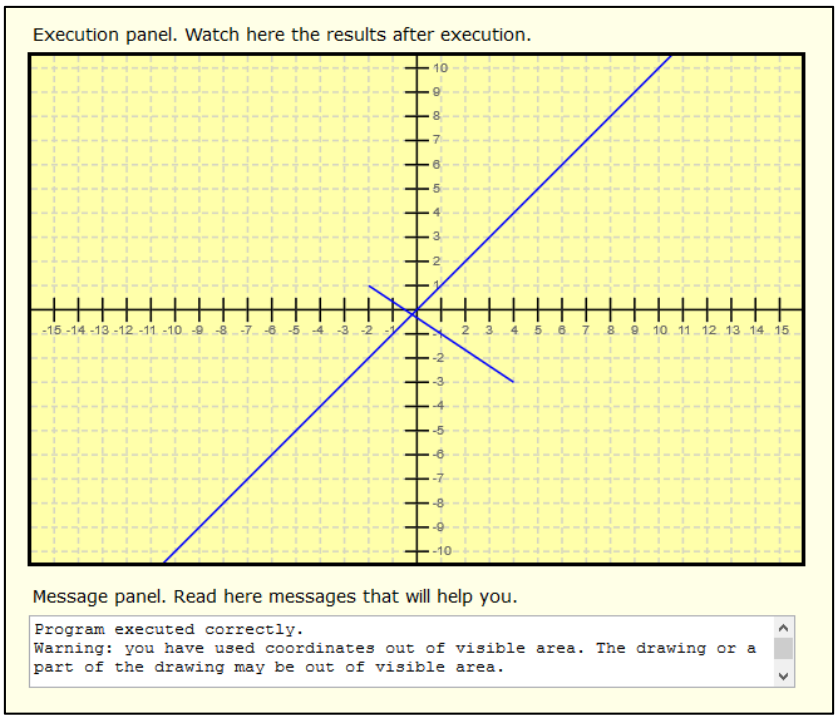
The result will look like the following:



Now write this other code:

```
Start program
Draw line from (-2,1) to (4,-3)
Draw line from (-100,-100) to (100,100)
End program
```

Whose result will look something like this:



Notice how one of the lines has its origins at a point outside the drawing panel, passes through the panel, and continues to finish outside the drawing panel as well. We could consider that these types of lines represent **straight** or "infinite lines". A line that starts and ends inside the drawing panel we might consider it as a **line segment** or portion of a line. Finally, if a line began at a point inside the drawing panel and ended outside it, we could consider it a **ray**.

Didac-Prog Cartesia allows the use of decimals and fractions. For example you can type something like: draw line from (-3.75, 2) to (4.25, -6)

But it is also supported and you get the same result if you write something like: Draw line from (-15/4, 2) to (8.50/2, -6) as -15/4 = -3.75 and 8.50/2 = 4.25

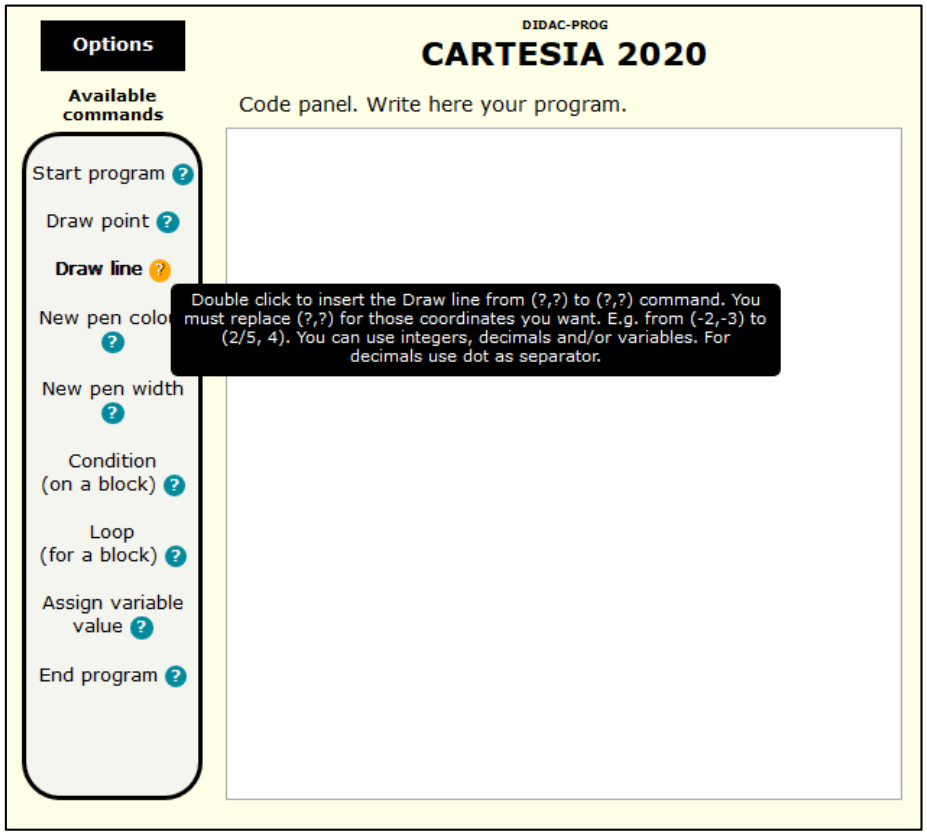
Let's try it testing with different programs that draw lines and interpret the results.

8. CLEAN ALL AND INSERT COMMANDS WITH THE COMMAND PANEL

Writing programs may make you tired or you are often mistaken in writing. Cartesia has a **command panel** on the left side of the screen that will make it easy to insert commands and write as little as possible if you prefer.

Let's start by deleting any drawing that exists and any code. To do this write a code, execute it and view the result, and then press the **Clear all** button. You will see that the code, drawing and messages in the message panel disappear. The code panel will be empty, the drawing panel will not contain any drawing, and in the message panel there will be a message similar to "Write your program in the code panel and press execute", that is, everything has been cleaned and the application is waiting for us to enter a new code.

Now look at the left side of the command panel where the 9 commands allowed by Didac-Prog Cartesia appear, each followed by a help symbol ? By placing the mouse pointer over a command symbol ? it will be displayed a help text on how to use that command. For example if we stand on the symbol ? next to *Draw line* we'll see something similar to this:



The help text will indicate something like "Double click to insert the command draw line from (?,?) to (?,?). You must replace (?,?) for those coordinates you want. E.g. from (-2,-3) to (2/5, 4). You can use integers, decimals and / or variables. For decimals use dot as a separator".

Let's now create a program that we have already written earlier, but now using the command insertion from the command panel. Our program will consist of 4 instructions: Start, Draw line, Draw line and Finish. We will then double click on the command panel on Start program, Draw line twice, and End program. On the code panel this should appear:

```
Start program
Draw line from (?,?) to (?,?)
Draw line from (?,?) to (?,?)
End program
```

Now we replace the question marks with the values that interest us, for example write:

```
Start program
Draw line from (-2,1) to (4,-3)
Draw line from (-100,-100) to (100,100)
End program
```

Press the Execute button and you will see that the result is the same as what we got before when we wrote the code.

From now on, remember that if you insert a command with a double click from the command panel, you must replace the places where question marks appear with those values, expressions or commands that are necessary.

To create your code in Didac-Prog Cartesia you can:

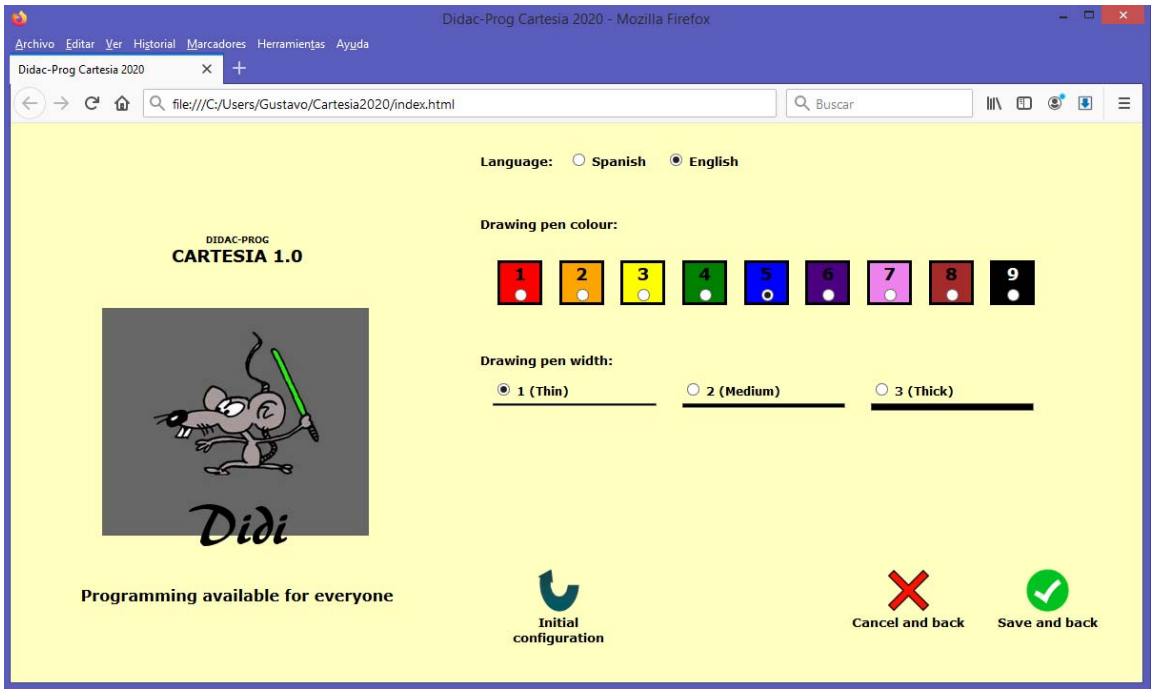
- **Write directly** from the code panel.
- **Copy and paste** (as is usually done in other applications) something that you have written, that someone else has written, obtained from a web page, etc.
- **Use the command panel** to insert a command by double clicking on it.
- **Open a program** that you have previously saved to run it or to modify it (later we will explain how).

9. CONFIGURATION SCREEN. LANGUAGE, COLOUR AND WIDTH OF THE DRAWING PEN.

In the programs that we have executed so far, the points and lines are drawn in a certain colour and a certain thickness. But we can change this in two ways:

- **By code** (will be explained later).
- **Modifying the application configuration.**

To change the application configuration press the "Configuration" button. You'll access a screen similar to the one shown below, with options for **Language**, **Drawing Pen Colour**, and **Drawing pen width**:



Language

On this screen you have the option to modify the language. Currently Didac-Prog Cartesia can be configured in two languages: **Spanish and English.** If you change the language and choose Spanish, you should be careful, because all texts will change to English and also the programs will have to be written in English. So, instead of *Start program* you would have to type *Iniciar programa*, instead of *Draw point at (*?, ?) the command would be *Dibujar punto en* (?, ?) and so on. It's easy to find out the command in Spanish. Once you've set up the Spanish language, double-click each command in the command panel to see how it's written in Spanish.

Drawing pen colour

Cartesia allows you to draw with 9 colours. Each colour is designated by a number, so the colours we can choose are 1, 2, 3, 4, 5, 6, 7, 8, or 9. The colour equivalence of each number is as follows:

Colour in Cartesia	Colour you will see in the drawing panel
1	Red
2	Orange
3	Yellow
4	Green
5	Blue
6	Indigo
7	Pink
8	Brown
9	Black

If on the configuration screen we choose a colour and press the “Save and back” button, all the elements (points and lines) that appear in the drawing from now on will do so with this chosen colour, except if we indicate through code the use of another colour.

Drawing pen width

Cartesia allows you to draw with 3 pen widthnesses. Each widthness is designated by a number, so the widthnesses we can choose are 1, 2, or 3. The widthness equivalence of each number is as follows:

Width in Cartesia	Points and lines widthness in the drawing panel
1	Thin
2	Intermediate
3	Thick

If on the configuration screen we choose a widthness and press the "Save and back" button, all the elements (points and lines) that appear in the drawing from now on will do so with this chosen widthness, except if we indicate through code the use of another widthness.

Save configuration

If after changing any configuration option we press "Save and back", we will return to the main screen and all the drawings we make will be governed by the chosen settings. For example, try changing the colour to 1 (red) and widthness to 3 (thick) and draw some drawing over the drawing panel and you'll see that the drawings appear in red and thick stroke.

Caution: if you press "Cancel and back" instead of pressing "Save and back" no option chosen will be saved and the settings that previously existed will continue.

Recover the initial configuration

Clicking on the "Initial configuration" button (in Spanish, “Configuración inicial”) will return to the initial application settings.

10. NEW PEN COLOUR AND NEW PEN WIDTH COMMANDS. STROKE TEXT.

So far we have not mentioned the possibility of text appearing inside the drawing panel. Is it possible? In the current version of Cartesia, it is not possible to insert text directly into the drawing panel. However, if you have patience you can draw or create small programs where we can store the code that generates text in the form of drawing (for example the x would be two lines that cross) for use in later programs.

In the programs we have run so far the points and lines are drawn in a certain colour and a certain widthness. We have already seen that colour and widthness can be changed through the configuration of the application. But this has one limitation: it assumes that all points and lines will be drawn **in the same way.**

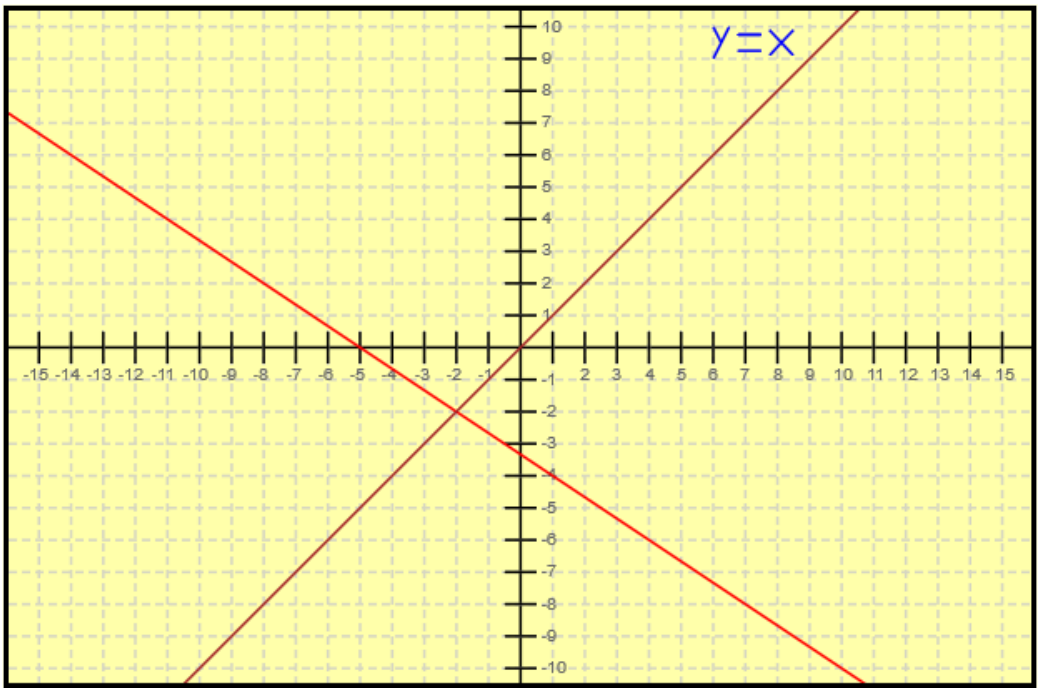
How can we draw lines and points of different colours and widthnesses within a drawing? The solution is to use specific commands that introduced in our code allow us to modify the colour and thickness with which we stroke the drawing. Note that if we change the colour (or widthness) at any point in our code, everything that is drawn afterwards will be in that colour (or widthness), except if we set a statement that modifies the current values.

New pen colour and stroke text in the drawing panel

Let's create a program similar to the ones we've used before to plot with different colours.

The command to use is **New pen colour (?)** replacing the ? by a whole number between 1 and 9. Write this code in the code panel and check the result, which should look similar to the one shown below:

```
Start program
New pen colour (1)
Draw line from(-20,10) to (40,-30)
New pen colour (8)
Draw line from(-100,-100) to (100,100)
New pen colour (5)
Draw line from(6,10) to (6.25,9.5)
Draw line from(6.50,10) to (6,9)
Draw line from(6.75,9.75) to (7.5,9.75)
Draw line from(6.75,9.75) to (7.5,9.75)
Draw line from(6.75,9.25) to (7.5,9.25)
Draw line from(7.75,9.9) to (8.5,9.1)
Draw line from(8.5,9.9) to (7.75,9.1)
End program
```

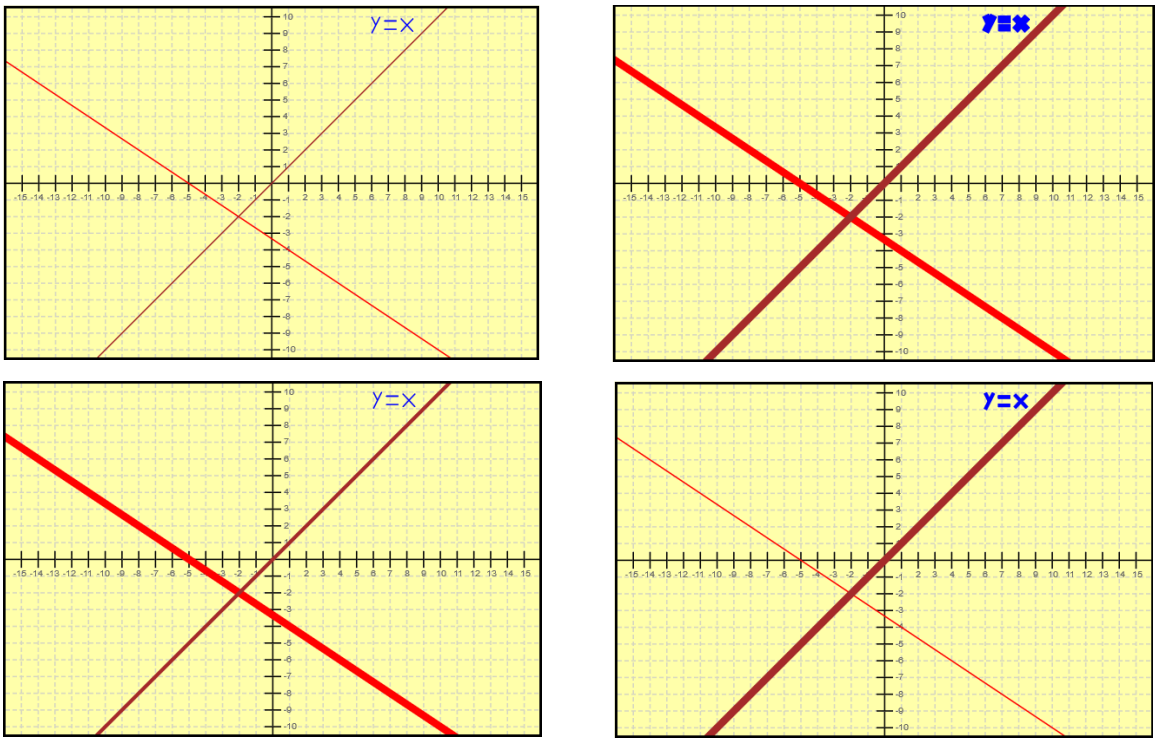


Clicking *Execute* causes the following: colour 1 is set as the new drawing pencil colour (that is, the stroke colour will be red). Next a red line is drawn (because we have set red as the pencil colour) that passes through the points (-20, 10) to (40, -30). These points will surely be outside the visible area, however the line that joins them passes through the visible area, and we can see it in red. Then it is set as the colour of the drawing pencil the colour 8 (that is, the stroke colour will be brown) and the same way a new line that we can see in brown colour is plotted. Finally it is set as the colour of the drawing pencil 5 (blue) and several lines are drawn that when viewed on the screen they look like the text y = x, which is seen in blue. Make tests by drawing dots and lines of different colours.

New pen width command

The change of widthness is done in a very similar way as we have seen for the colour. The command we should use is **New pen width (?)** replacing the ? by 1, 2 or 3. Setting widthness 1 is equivalent to a thin pen, widthness 2 is equivalent to an intermediate-widthness pen, and widthness 3 is equivalent to a thick pen.

You can set a widthness at the beginning of your program to always draw with that widthness, or change the widthness in different lines of your program. In the following images you can see how the appearance of the drawing seen in the previous example changes by changing the widthnesses:



Results that can be displayed by changing widthnesses to drawing elements. Top left: everything is plotted with widthness 1 (thin). Top right: everything is plotted with widthness 3 (width). Bottom left: the red line has thick stroke, the brown has intermediate widthness, and the y=x text has thin widthness. In the lower-right image, the brown line has a thick stroke, the red line a thin one, and the y=x text has intermediate widthness.

Warnings for execution without display possible

It could happen that by mistake we write a program where nothing is drawn. For example, a program that changes the colour or widthness of the drawing pen. In this case, after pressing the Execute button we will not see anything in the drawing panel because there's nothing to draw. As always, the message panel will be of great help to interpret what is happening, as it will show us a warning. Type this code and execute it:

```
Start program
New pen colour (8)
New pen width (2)
End program
```

You'll see that nothing drawn appears in the drawing panel, because we haven't entered any statements that draw something. A message similar to the following will appear in the message panel:

```
Program executed correctly.
Warning: your code doesn't have any command allowing to draw anything on the execution panel, therefore you will not be able to see any result.
```

11. COMMENTS IN CARTESIA CODE

As our programs become more extensive and complicated, we may no longer remember what certain lines of our code are for or even don't remember what we created that program for. It's also possible that if we have to modify a saved program, we don't know what each part of the code is for.

To avoid these problems (and for many more things), Cartesia allows you to enter **comments** in the code. These comments will not be executed, as they are not instructions to execute, but simply a text that helps us to know and explain the program, reflect the date of creation, author, or whatever we want.

A comment is inserted into Cartesia by inserting two middle hyphens (--) and then the free text you want to enter. Because comments are not executed, the same code could be commented out in different ways and not alter the result. To check this, let's comment on a code seen above in two different ways. Write both programs and see how the results are the same (we've boldly pointed out the comments to make them easy to identify):

```
-- PROGRAM THAT DRAWS TWO LINES
Start program
New pen colour (1) -- Red colour
Draw line from(-20,10) to (40,-30)
New pen colour (8) -- Brown colour
Draw line from(-100,-100) to (100,100)
New pen colour (5) -- Blue colour
Draw line from(6,10) to (6.25,9.5)
Draw line from(6.50,10) to (6,9)
Draw line from(6.75,9.75) to (7.5,9.75)
Draw line from(6.75,9.75) to (7.5,9.75)
Draw line from(6.75,9.25) to (7.5,9.25)
Draw line from(7.75,9.9) to (8.5,9.1)
Draw line from(8.5,9.9) to (7.75,9.1)
End program
-- Created on 16/05/2048 by John Smith for the subject of Math of 5th grade of Elementary school in Anne Frank Elementary School, Philadelphia.
```

```
-- PROGRAM THAT DETERMINES THE CUTOFF POINT BETWEEN TWO LINES
Start program
-- We set as drawing colour the red colour (1) and we stroke the red line
New pen colour (1)
Draw line from(-20,10) to (40,-30)
-- We set as drawing colour the brown colour (8) and we stroke the brown line
New pen colour (8)
Draw line from(-100,-100) to (100,100)
-- We set as drawing colour the blue colour (5) and we stroke the red line
New pen colour (5)
Draw line from(6,10) to (6.25,9.5)
Draw line from(6.50,10) to (6,9)
Draw line from(6.75,9.75) to (7.5,9.75)
Draw line from(6.75,9.75) to (7.5,9.75)
Draw line from(6.75,9.25) to (7.5,9.25)
Draw line from(7.75,9.9) to (8.5,9.1)
Draw line from(8.5,9.9) to (7.75,9.1)
End program
-- Created on 16/05/2049 by John Smith for the subject of Math of 6th grade of Elementary school in Anne Frank Elementary School, Philadelphia.
```

Please note that comments must meet certain rules:

a) The two middle hyphens **have to be together.** If they are separated, an error message appears in the message panel.

b) A comment can occupy **multiple rows** in the code panel as long as we don't press the Enter key, which generates a new line. If we pressed the Enter key, each line should start with the two middle hyphens. If we do not do so, an error message of this type will appear: "ERROR: Please check the code. Invalid line", next to the line text, for example "Math subject of 5th grade of elementary school in Anne Frank Elementary School, Philadelphia."

As you can see, a comment can be at the beginning of a line occupying the entire line, or after a command, remaining as the final part of a line. What is not allowed is to write a comment and then on that same line a command, because since the command is behind the two hyphens, it will not be executed precisely because it will be considered as a comment and not as a statement.

12. VARIABLES IN CARTESIA. ASSIGN VARIABLE VALUE COMMAND.

Introduction to variables in programming

In mathematics it is common to use variables like x, y, a, b, c and others. For example, if someone tells us that x = 3 and that a = 4, and ask us how much x + a is, we will say 7 since 4 + 3 = 7. But then in the same problem we could not say that a is worth 9, because we have previously said that a is worth 4.

Variables are used in programming similar to how they are used in mathematics, although there are some differences. In Cartesia we can see a variable as a memory, or as a box containing a piece of paper where a number is written. For example if we say that x is worth 3, inside the box "x" is the paper with the number 3. If in box "a" we have a 4 and they ask us how much x + a is worth, we will say 7 since 4 + 3 = 7.

If we now say that the new value of "a" is 1, it means that we take out the paper where 4 was written, we throw it away, and inside the box we put the paper with 1. If they ask us how much x + a is, we will say that 5 since 4 + 1 = 5. Here as we see "a" has changed in value. First it was 4 and then it has become 1. This is used in programming (but not in mathematics, where in the same problem, "a" only can have one value).

Available variables in Cartesia

In Cartesia you can use seven variables: **x, y, a, b, c, d, e**

Numeric values can be stored in each of these seven variables, which can be both integers and decimals. Numeric values can be expressed directly, for example 3 and 0.20, or as fractions, for example 6/2 which is equivalent to 3 or 1/5, which is equivalent to 0.20. All variables have a leading value of zero, until they are assigned another value with the assignment command.

In Cartesia it is not possible to use or create other variables than the 7 variables we have quoted.

The command to set the value of a variable is:

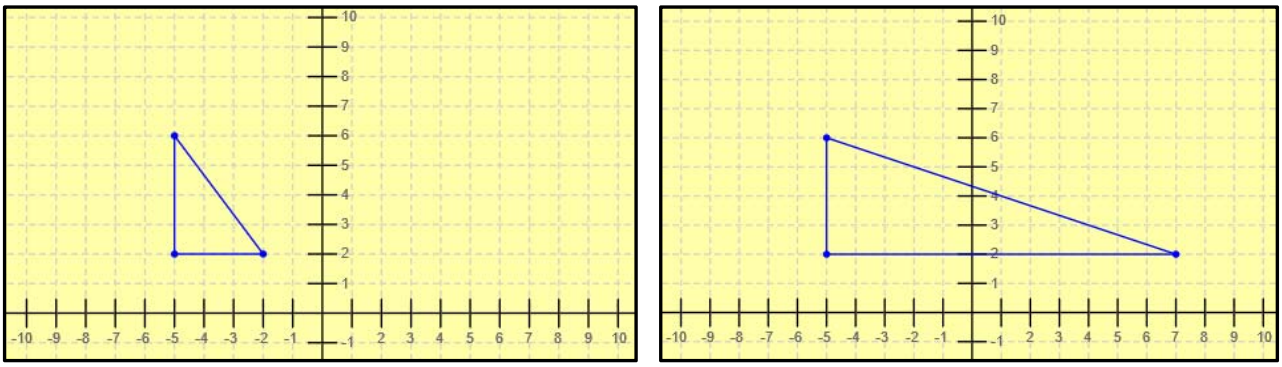
? new value is (??)

Here ? should be replaced by the variable name and ?? for the value it takes. So we can assign the variable x the value 3 in many ways: x new value is (3), or x new value is (6/2) or x new value is (2+1), etc.

Type this code in the code panel and press Execute:

```
Start program
x new value is (-5)
y new value is (6)
a new value is (-2)
b new value is (2)
c new value is (-5)
d new value is (2)
Draw point at (x,y)
Draw point at (a,b)
Draw point at (c,d)
Draw line from(x,y) to (a,b)
Draw line from(a,b) to (c,d)
Draw line from(c,d) to (x,y)
End program
```

We've drawn a triangle. Now change the value assigned to "a" so that instead of -2 it takes value 7, that is, the fourth line should look like this: *a new value is (7)*. Press Execute.



Appearance of the two triangles drawn using variables.

You will see that by modifying the value of "a" the triangle has changed. This is possible because we have used variables. If we hadn't used them, we should have made more changes to our code to draw the new triangle, so using variables make us easier to code.

Use of variables

In Cartesia some uses of variables similar to those of professional and academic programming languages are allowed, such as:

a) A variable always **maintains** its value from the program line where it is assigned a value with the assignment command. That is why we say that variables "have memory". For many lines of code we write, the last value of the variable is always remembered. If no value is assigned, a variable will be zero throughout the program.

b) A variable can take its new value **from its previous value**. Suppose “a” is worth 3 because we have written "a new value is (3)". If at a later point in our program we write "a new value is (a + 1)" this means that we first take the paper that was inside the box that had a 3, then we calculate 3 + 1 which is 4, we write the value resulting in a new paper that we put in the box, and throw the previous paper. In summary, inside the box we have a 4, which has been obtained by adding 1 to the previous value that was in the box.

c) A variable can be used as the equivalent of its numerical value **anywhere in a program** where a number is required, for example to determine the coordinates of a point, to set a colour or widthness, or to assign a new value to another variable. For example *Draw point at (a, b)* is valid. It is also valid *Draw point at (a + 3, b-1)* and any expression where we make correct use of the mathematical operators.

c new value is (a + b) is also valid and means that c changes to take the result of adding a and b.

Suppose b is worth 3 because we have written *b new value is (3)*. If we now write *New pen colour (b)* that implies that from that line the drawing colour becomes 3 (yellow), since Cartesia is in charge of substituting the variable for its numerical value. Furthermore, we could have written *New pen colour (b-1)* and that would mean that the new drawing colour would become 2 (orange), or *New pen colour (b-2)* and that would mean that the new drawing colour would become 1 (red). On the other hand, if we wrote *New pen colour (b-3)* we would get an error message in the message panel of the type “*Error: invoked colour 0; colours only accept integer values between 1 and 9. Not valid value*”, since zero is not a valid value for a colour in Cartesia.

d) A variable **can be compared** with another numeric value to obtain a true or false result. For example if the variable c is 34 because we have written *c new value is (34)*, the comparison (c < 55) will result in true because 34 is less than 55. Instead the comparison (c > 55) will result in false as 34 is not greater than 55.

As a variable can be written anywhere in a program where a number is required, we can write comparisons such as (a >= b), the result of which will be true if a is greater than or equal to b, or FALSE if a is less than b.

Comparisons are used in the condition commands that are explained below.

13. REPEAT (LOOP) COMMAND. EXECUTION BLOCKS.

Introduction to looping (repeating) in programming

We have already described how you can draw a triangle or other geometric shapes. But what if we want to draw 20 triangles? Perhaps we could think about entering all the coordinates necessary to define where each triangle is located, which would take us a while. What if we wanted to draw 200 triangles? If we had to define the coordinates of all their vertices one by one, it could take several hours or a day. What if we wanted to draw 2000 triangles? It would probably get so tired we'd never do it. However, computers, tablets, smartphones, etc. have the ability to draw hundreds or thousands of triangles in less than a second thanks to the powerful built-in processors, as they are able to perform calculations and execute instructions at high speed.

In Cartesia it is possible to draw 20, 200, 2000 or more triangles using the repeat command, similar to that of professional and academic programming languages.

The idea will be: we define the variables necessary to draw the triangle. We enter a repeat command and draw the triangle. We modify the values of the variables that define where the triangle should be located, and repeat as many times as we want.

Let's take as starting code the next one. Type it in your code panel, press Execute, and make sure a triangle drawn appears in the drawing panel.

```
Start program
x new value is (-10)
y new value is (4)
a new value is (-8)
b new value is (2)
c new value is (-10)
d new value is (2)
Draw line from(x,y) to (a,b)
Draw line from(a,b) to (c,d)
Draw line from(c,d) to (x,y)
End program
```

Repeat command

The repeat or loop command is written as follows:

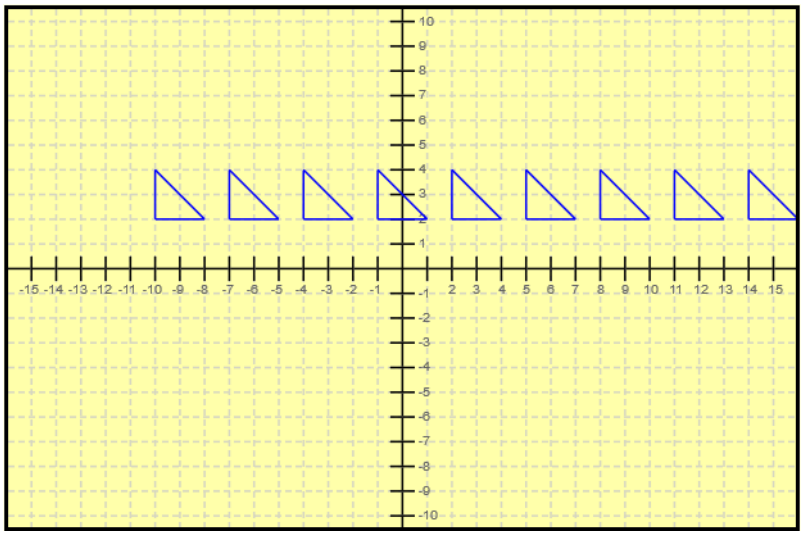
```
Repeat (?) times execute block
Block start
???
Block end
```

Where should we replace the ? by the number of repetitions we want to perform and the ??? by the instructions that we want to be executed on each repetition (for example the instructions to draw a triangle with each repetition). The code that will run on each repetition is called the execution block and can include as many commands as you want.

From the code above, type the following code and click Execute:

```
Start program
-- We define the initial coordinates for the triangle
x new value is (-10)
y new value is (4)
a new value is (-8)
b new value is (2)
c new value is (-10)
d new value is (2)
-- We will draw 25 triangles
Repeat (25) times execute block
Block start
--Statements to draw one triangle
Draw line from(x,y) to (a,b)
Draw line from(a,b) to (c,d)
Draw line from(c,d) to (x,y)
-- We prepare the next repetition
--We define coordinates for the next triangle
x new value is (x+3)
a new value is (a+3)
c new value is (c+3)
Block end
End program
```

Note that after drawing the triangle, we move 3 units to the right its vertices changing the values of **x**, **a** and **c**. the values of **y**, **b** and **d** do not change because we are not modifying the situation of the triangle vertically. The result we will get will be similar to the following:



Why if we have created a program to draw 25 triangles we do not see 25 triangles? Whenever a result is not what you expect you can:

- Check **the code** to see if you've done something wrong.
- Check the **message panel** to see if there is an error message or warning.

In this case, in the message panel we will find a message similar to: "*Program executed correctly. Warning: you have used coordinates out of visible area. The drawing or a part of the drawing may be out of visible area.*"

The 25 triangles **have actually been drawn**, the thing is that some of them are out of the visible area of the drawing panel and we don't see them.

Try writing the same program but making the values of **y**, **b** and **c** also increase 3 units in each repetition. To do this you will need to add three lines:

y new value is (y+3)
b new value is (b+3)
d new value is (d+3)

With this we will make the triangle draw not only to the right but also upwards. If instead of (y+3), (b+3) and (d+3) we write (y-3), (b-3) and (d-3) we will get the triangle to be drawn downwards.

Test modifying the horizontal and vertical offset by giving it different values and check the results. You will see that if you set very large gaps (for example 40 units) you will only see one triangle because the rest is far away. If you set very small gaps (for example 0.5 units) you will see how triangles are drawn overlapping: there is not enough separation to be able to see each triangle clearly.

Test by modifying the variables **x**, **a**, **c**, **y** at each repetition. With this you can move the triangle horizontally and change its height at each repetition.

There is no limit to the number of repetitions you can set. In addition, for those who dare, Cartesia allows you to do "repetitions of repetitions". For example, repeat ten times draw 25 triangles horizontally, modifying the vertical position at each time, so that we can fill the panel with triangles or any drawing that we can think of.

Another interesting possibility of repetitions is that we can change the colours in each repetition using the **New pen colour (?)** command.

Do all the tests you can think of. If at any given time you want to start over from scratch, press the "Clean All" button.

14. LOGICAL AND COMPARISON OPERATORS. CONDITION COMMAND.

Logical operators

Cartesia allows you to define a part of the code to run only if a condition is met. The condition command and two logical operators are available for this purpose. The logical operators in Cartesia are these:

Operator	Meaning	Example
and	And it is also true that	(x<3 and y<5)
or	Or is it true that	(x<3 or y<5)

For the examples indicated in the previous table, suppose that x is 5 while y is -2. The logical comparison (x < 3 and y < 5) is equivalent to asking: Is 5 less than 3 and -2 less than 5? The result you get is FALSE because 5 is not less than 3, regardless of whether -2 is less than 5, because both conditions are required to be met.

The logical comparison (x < 3 or y < 5) is equivalent to asking: Is 5 less than 3 or -2 less than 5? The result that is obtained is TRUE because although 5 is not less than 3, it is true that -2 is less than 5. In this case, to obtain TRUE it is enough if one of the two conditions is met.

Condition command

The condition command is written as follows:

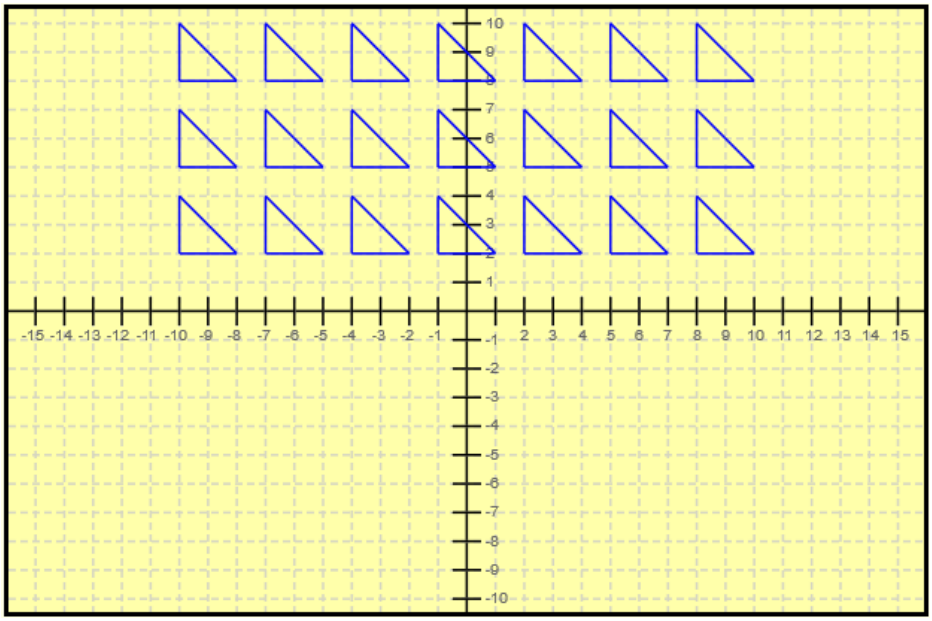
```
If truth (?) execute block
Block start
???
Block end
```

Where we should replace the ? by the condition that determines whether the statement block should be executed or not, and the ??? by the instructions we want to be executed when the condition is met. The code that will be executed when the condition is met is an **execution block** and can include as many commands as you want. In Cartesia there are two commands that involve the use of execution blocks: the repeat command and the condition command.

From the code we used earlier to draw 25 triangles, let's modify the program to make when the triangles are to come out of the drawing panel, instead of continuing to the right, keep drawing starting on the left side a little higher by creating rows of triangles. Write and execute this code:

```
Start program
--We define the initial coordinates for the Triangle
x new value is (-10)
y new value is (4)
a new value is (-8)
b new value is (2)
c new value is (-10)
d new value is (2)
--We will draw 25 triangles
Repeat (25) times execute block
Block start
--Statements to draw a triangle
Draw line from(x,y) to (a,b)
Draw line from(a,b) to (c,d)
Draw line from(c,d) to (x,y)
--We prepare next repetition
--We define coordinates for next triangle
x new value is (x+3)
a new value is (a+3)
c new value is (c+3)
-- If they are very to the right we go back to the left and
go up to create a new row
If truth (x>10) execute block
Block start
x new value is (-10) --Return to the left
a new value is (-8) --Return to the left
c new value is (-10) --Return to the left
y new value is (y+3) --We move up
b new value is (b+3) --We move up
d new value is (d+3) --We move up
Block end --The condition block ends
Block end --The loop block ends
End program
```

The result will be similar to the following:



Notice that after drawing the first triangle, we move 3 units to the right its vertices by changing the values of **x**, **a**, and **c**. The values of **y**, **b** and **d** do not change them because we are not changing their situation vertically.

The value of **x** tells us if a triangle is far to the left (values such as -10) or far to the right (values such as 10). What we do is in each repetition check if the **x** has become greater than 10, which would tell us that the triangle is already too much to the right. When this occurs, the **If truth (x>10) execute block condition** is met and the condition block is executed. When running this block, we reposition the triangle to the left with these instructions:

```
x new value is (-10) --Return to the left
a new value is (-8) --Return to the left
c new value is (-10) --Return to the left
```

Also, we move up with these instructions:

```
y new value is (y+3) --We move up
b new value is (b+3) --We move up
d new value is (d+3) --We move up
```

Once this is done, the repetitions continue, starting now from a position to the left and higher than the previous one. The result is that **a new row** of triangles is drawn. As in each repetition we check if they have gone too much to the right, when this happens, we will do the same as before. That is, set the coordinates to start on the left and a little higher. So until the repetitions are finished.

Will we see the 25 triangles in the drawing panel? We may not see them all because some of them are left out of the visible area. Or maybe some will show split because they are near the edge of the visible area. Pay attention to the message panel because you may have a warning.

You can make each row to be drawn with a different colour. To do this you can write as the first lines of the program:

```
e new value is (4)
New pen colour (e)
```

If you click Execute you will see that all triangles are drawn in green, because we have established that the colour of the drawing pen is 4 (green).

Now modify the condition block like this (by entering the two lines we point to):

```
If truth (x>10) execute block
Block start
x new value is (-10) --Return to the left
a new value is (-8) --Return to the left
c new value is (-10) --Return to the left
y new value is (y+3) --We move up
b new value is (b+3) --We move up
d new value is (d+3) --We move up
e new value is (e+1) -- e value increases in 1
New pen colour (e) -- Define new pen colour
Block end --The condition block ends
```

With this what we get is that every time a large value of x (x> 10) is reached, in addition to taking the triangle to the left and up, we increase the value of the variable "e" that represents the colour and change the colour (each time from 4 to 5 when adding 4 + 1, then from 5 to 6 when adding 5 + 1, then from 6 to 7, etc.)

If you press Execute, each row of triangles will be drawn in a different colour.

Comparison operators in conditions

The following operators are available to define conditions:

Symbol	Meaning	Example
==	It's the same as...	x == y
<	Is less than...	x < y
>	Is greater than...	x > y
<=	Is less than or equal to...	x <= y
>=	Is greater than or equal to...	x >= y
!=	It's different from...	x != y

Be careful because typing something like *If truth (x=10) execute block* will generate an error and the message panel will display something similar to the following: << ERROR: conditional statement contains = instead of ==. Please check code. Not valid line.>>

Remember that to compare if it is equal you must use the == symbol with two equals in a row. If you use a single equal you will see the error message that we have commented.

15. MATHEMATICAL OPERATORS AND CURVES WITH CARTESIA

Cartesia allows you to use a series of **mathematical operators** applicable anywhere where a numerical value is required. The available mathematical operators are eight:

Symbol	Meaning	Example
+	Addition	a + b
–	Substraction	a – b
*	Multiplication	a * b
/	Division	a / b

Symbol	Meaning	Example
SQR(num)	Square root of num	SQR (a+b)
^	Exponentiation (power)	a ^ 2 + b
%	Modulo operator (integer remainder of division between two integers)	a % b
( )	Parentheses. Operations priority	a ^ (2+b)

From these operators use those you know. It is not necessary to know all the operators to create programs in Didac-Prog Cartesia.

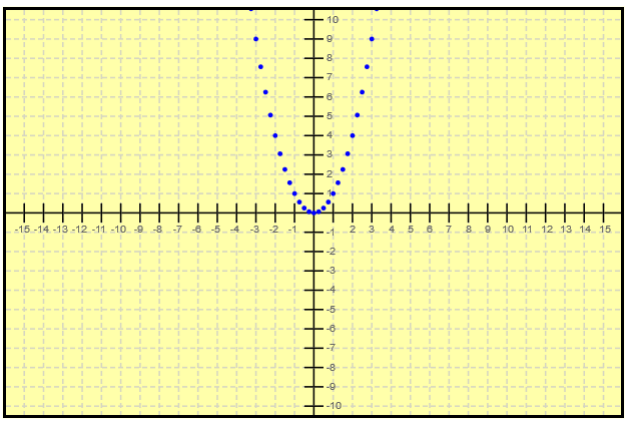
Representations using powers and square roots allow you to draw curves often used in mathematics. Therefore, although Cartesia does not directly allow curves to be drawn, **we can draw curves** by drawing points very close together, or by drawing small lines and joining them to make them look like a curve. Try doing this on paper, you will see that you can draw a curve by drawing points very close together or small straight segments and joining them.

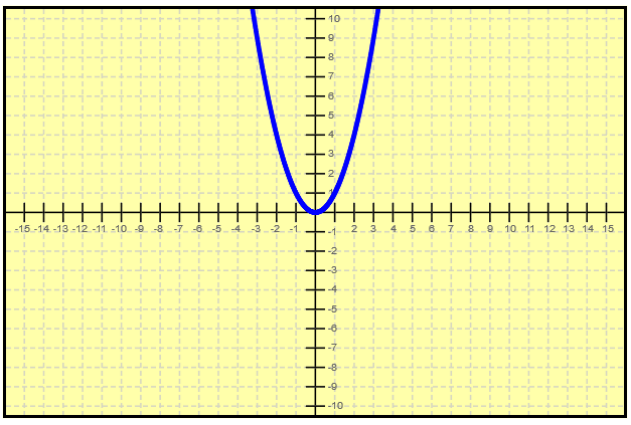
We are going to represent using very close points the curve that is usually called quadratic and is usually written as $y = x^2$, that is the same as y = x^2 or y = x*x. For each value of x that we choose, we will draw a point on the coordinate (x, x * x). For example for x = -3 we will draw the point (-3, -3 * -3) which is (-3, 9). Or for x = 2 we will draw the point (2, 2 * 2) which is (2, 4). Try executing these codes:

```
-- PROGRAM PLOTTING THE QUADRATIC FUNCTION WITH SEPARATED POINTS
Start program
x new value is (-5)
Repeat (700) times execute block
Block start
Draw point at (x, x^2)
x new value is (x+0.25)
Block end
End program
```

```
-- PROGRAM PLOTTING THE QUADRATIC FUNCTION WITH POINTS SO CLOSE TOGETHER THEY LOOK LIKE A CURVE
Start program
x new value is (-5)
Repeat (700) times execute block
Block start
Draw point at (x, x^2)
x new value is (x+0.015)
Block end
End program
```

If you look at both cases we have drawn 700 points starting from the value -5 and increasing the value x in each repetition in a case 0.25 units (-5, -4.75, -4.50, -4.25, -3.00, -2.75, -2.50, -2.25, etc.) and another 0.015 units (-5, -4.985, -4.970, -4.955, -4.940, -4.925, -4.910, etc.). The result may be similar to the following:





In one case, the points are quite separated so we sense the curved shape that would be generated by joining the points. In the other case, the points are so close that we get to see the curve. In both cases points have

probably been drawn outside the visible area but this does not worry us. You can check this by looking at the message panel.

Notice how moving from one result to another has depended on a very small change in a program code. However, the result has been very different. Try different tests:

a) Set 100, 200, 300 etc. repetitions instead of 700 and execute both codes. Try to interpret what's going on.

b) Instead of 0.25 and 0.015 use other values, for example 3, 2, 1, 0.5, 0.2, 0.1, 0.05 etc. Try to interpret what's going on.

Always check the message panel as it can give you valuable guidance.

There are several factors that can lead the results of our programs not being as expected, but you should pay special attention to:

- The **number of repetitions**: it may be too small or too large.
- How we increase (or reduce) the value of variables. They can have **increments** (or reductions) that are too large or too small.
- Whether statements in a conditional are executed or not executed. There are cases where the execution block of a conditional **never executes** because we set a condition that is never met.

16. OPTIONS MENU: OPEN AND SAVE PROGRAMS.

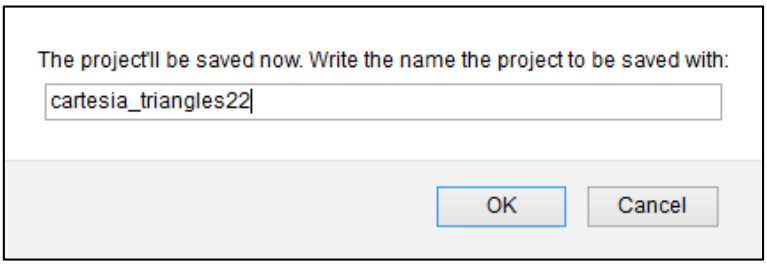
Introduction

When we listed the parts where the main screen of Didac-Prog Cartesia is divided we mentioned the existence of an options menu at the top left of the screen. Hovering over this menu displays different options: *Open Project, Save Project, Open Example, and Undo Clean*.

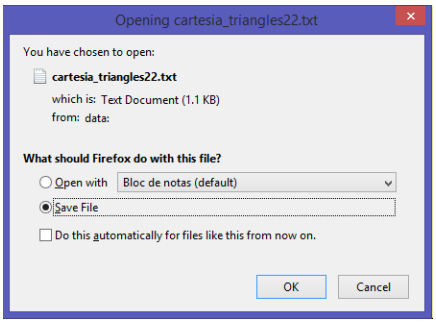
Save project

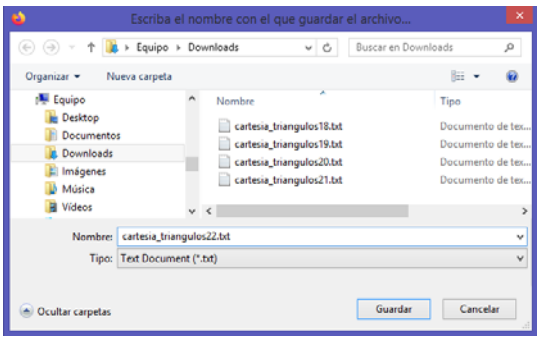
The *Save Project* option allows us to save the code that we have written in the code panel (what we call "project") to a file so that we can recover it whenever we want. Suppose we have written our code and after executing it we see that the result is correct and we want to save it to send it to the teacher or to use it another day. To do this we place ourselves on top of the options menu and when it is displayed we choose "Save Project". At this point we will see a dialog telling us "*The project will be saved now. Write the name the project to be saved with:*". By default it appears as the start of the name "cartesia_" but this can be removed if we want.

To test this functionality, write a program to draw a triangle where the first line is a comment with your name, date, and time. Press "Save Project" and name it cartesia_triangles22. Once this is written, press OK.



Depending on the browser you use and the options you have configured, the file will be saved directly (for example in the Downloads folder), or a new dialog will open asking if you want to open or save the file. In this case, we choose *Save* and click OK.





The file will have been saved either in a folder (such as Downloads, *Descargas* or other) or in the path you have chosen to do so. We recommend saving your projects in the Projects folder in the Cartesia directory, in a path that will look like C:/Users/Gustavo/cartesia2020/projects/, but this is optional and you can save your projects in your desired path, even on a flash drive if preferred.

Use the File Explorer on your computer and check that the file has been saved. Open it (double click on it) and check that it contains the code you wrote.

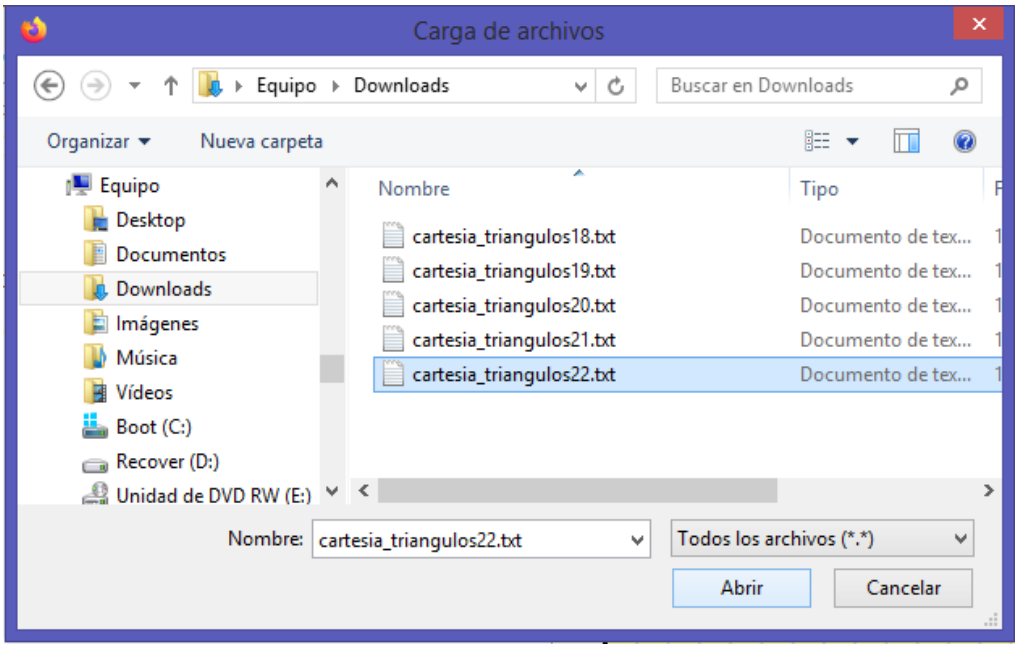
Didac-Prog Cartesia files are simple text files (txt).

Open project

The *Open Project* option allows us to open a project from a file. This project may have been saved before, downloaded from aprenderaprogramar.com, received by email, received it on a flash memory, etc.

Be careful: before you choose this option, **save the code** you have in the code panel. If you don't, you'll lose the code you were working on, as it will be replaced by the project code you're opening.

Once you click on the options menu, "Open Project", you will be presented with a window from which you will have to search and select the file to open.



After that, the project will appear in the code panel and we can start working on it. If we make changes that we want to keep, we must save it under the same name or under a new name, as we think is more convenient for us.

17. OPTIONS MENU: OPEN EXAMPLE AND UNDO CLEAN.

Open example

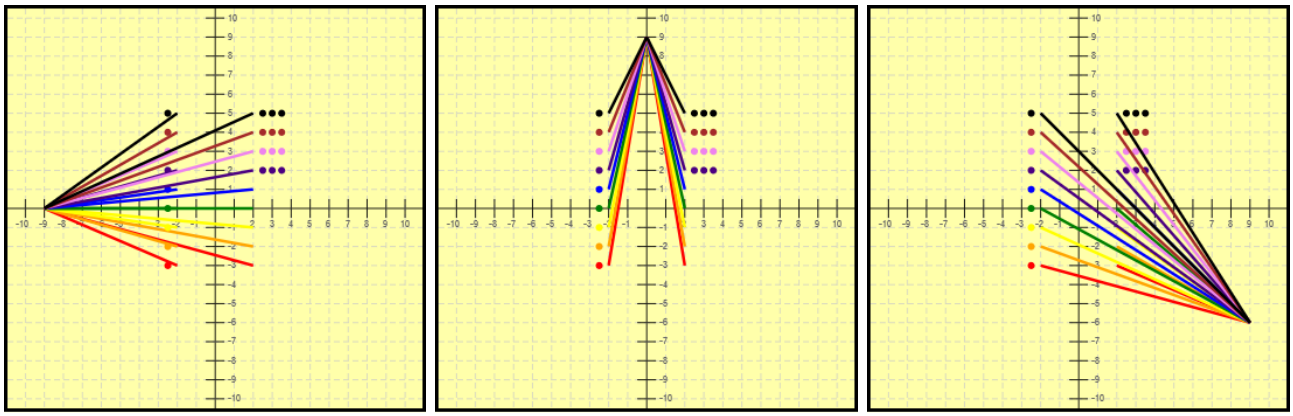
Within the options menu, "Open example" allows us to open an example program that makes use of all the commands of Didac-Prog Cartesia.

Be careful: before choosing this option, **save the code** you have in the code panel. If you don't, you will lose the code you were working on, as it will be replaced by the code in the example you open.

Press *Open example*, and then press the Execute button. On the screen you will see the execution result.

Try modifying this program. For example change the initial value of **x**. To do this, look for the line where **x** is given value and where it says "x new value is (0)" write "x new value is (-9)". Try other values like -3, 3, 6, 9, etc. and check the results. Also change the initial value of variable **c**, check the results and try to interpret the why of those results.

If at any time you want to start over from the original example, all you have to do is select "Open example" from the options menu. Of course, remember that you will lose the code that exists in the code panel if you have not previously saved it.



Different executions of the example program, changing the value of the variables x and c.

Every time you press *Open example*, the same program is loaded. If you want to see other examples, you can do it using the *Open Project* option, and in the path where Cartesia is located, choose the *Examples* folder and open any of the more than 40 projects that you will find there saved.

Undo clean

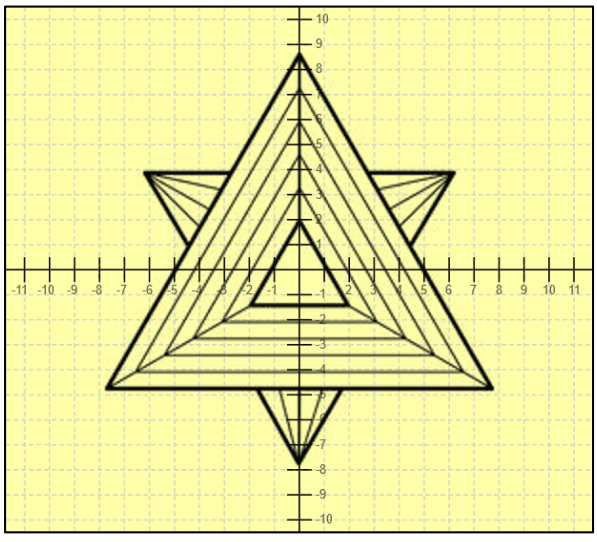
The "Clean All" button makes the code and drawings that might exist disappear and Cartesia returns to its initial situation with everything blank. Sometimes we have been working in a program for a while and, without realizing it, we press "Clean all" without saving our code. If this happens to you, you can recover your code if immediately after cleaning everything you do one of these two things:

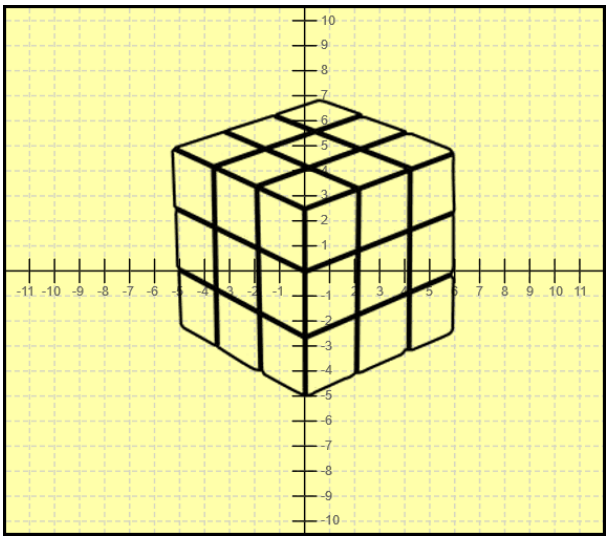
a) Press the CTRL+Z key combination to **undo** the clean all and recover your code. Then save your code so you don't lose it and keep working.

b) Choose ***Undo clean*** from the options menu. This, like CTRL+Z, allows you to recover your code so you can save it and keep working.

18. POSING CHALLENGES WITH DIDAC-PROG CARTESIA.

An interesting activity with Didac-Prog Cartesia is to pose a challenge. For example, choose a figure and ask ourselves how could we make this figure or drawing without having to draw all the lines that form it one by one? Challenge yourself or a friend and improve yourself as a programmer. In case you don't think of any challenges, here are two, and in aprenderaprogramar.com you can find more.





DIDAC-PROG

CARTESIA

ANNEX FOR TEACHERS

A-1. INTRODUCTION

Cartesia is an application conceived for didactic and teaching purposes. It was deliberately designed with one goal: **to be simple and powerful.** A common problem in the classroom is the lack of time to learn how to use applications with huge amounts of functions and possibilities. Cartesia does not aspire to be one of them. Hence it has limited functionalities compared to professional programming languages or multipurpose applications for teaching mathematics or programming in the broader sense.

The intention in creating this application has not been the teaching of a programming language, nor of general aspects of programming languages, but the teaching of algorithmic and the foundations of the logic of programming or ***computational thinking*** that, in somewhat bombastic words, "constitutes the basic substrate of our computerized societies". With another perspective, at the same time it is a way of interactive teaching of mathematics by allowing to work concepts such as Cartesian system of coordinates, points and lines, graphic solution of equations, functions, etc. in an environment where the student can feel that he masters the application and knows he masters everything he does. Cartesia is therefore conceived for another clear purpose: to prevent the student from feeling overwhelmed by an excess of information and possibilities.

Other virtues of Cartesia's simplicity are to take up very little space (<5 Mb) and allow you to work fully operationally without having to be connected to the internet.

Those teachers looking for a math-oriented tool with more power and complexity can use Geogebra (https://www.geogebra.org/?lang=es-ES), an application that allows both advanced teaching in multiple fields of mathematics and the introduction of programming.

Those teachers looking for other environments to teach programming to children have *Scratch*, *Alice*, *App-Inventor*, to name a few possibilities that are very popular.

Cartesia can be used if desired as a previous, subsequent, or complementary step, with respect to any of the tools we have cited.

A-2. FLEXIBLE DESIGN OF CARTESIA'S LANGUAGE. ADVANTAGES AND DISADVANTAGES.

Didac-Prog Cartesia is based on a programming language that does not have a specific name: we refer to it simply as "the application language" or "Didac-Prog Cartesia language". This language has been designed **specifically** for the application. It is designed for educational purposes and has some similarities and some differences from conventional programming languages.

As main similarity we can point out the use of the three control structures of classical structured programming: sequential, conditional and repetition (looping). Among the differences we can mention having a very limited and simplified set of commands compared to standard programming languages. Another significant difference is that Cartesia has been designed with a lax syntax based on regular expressions, as opposed to the more rigid syntax typical of programming languages.

For example, a statement like "New pen colour (2)" in general in another language could not be written otherwise. In Cartesia in addition to as indicated it can also be written as "The new colour of the pencil becomes (2) from now on" or as "the colour of the pencil changes: now it will be (2)" which are more extensive forms. But you can also write it as "New colour (2)" or as "Colour (2)" which are smaller forms.

Cartesia accepts **both** upper and lower case letters, as well as some other characters. It is permissible both "Draw line" and "Draw Line" as well as "DRAW LINE "or "draw line".

The way of writing indicated in the user manual (and inserted if clicked in the command panel) is the "recommended" way intended by the creators of Cartesia (e.g. "New pen colour (?)"). However, if you as a teacher consider it preferable for your students to use another form within the range of allowed expressions, you can opt for any of the valid alternative ways.

The flexible design of the language has been made by seeking:

a) That the student can focus on the **logic and algorithmic** of his programs (the "computational thinking"), without having to memorize a rigid syntax. By supporting variants, even two different ways for the same command within the same program, the language is closer to natural language and distances itself from formal languages.

b) That the teacher may propose the use of **alternative ways** that he deems most suitable according to his teaching criteria.

Flexible design we think it has relevant pedagogical advantages over other programming languages. Also some inconvenience. The "New pen colour (2)" statement is flexible for the ways we have quoted and others consistent, but it also supports undesirable forms (by incoherent, for having spelling faults, etc.) if they meet the required regular expression. However, it has been considered that the pedagogical advantages outweighed the disadvantages and were sufficient reason to bet on this design.

The following table lists the syntax requirements for each Cartesia command and some valid syntax examples. Brackets indicate optionality. Quotes and parentheses indicate obligatoriness. You can decide, within the specified validity range, if make a greater or lesser use of the flexibility offered by the language.

Command	Syntax requirements	Valid examples
Start program	It must contain "start" as a line beginning, whether or not followed by additional strings.	Start Start program START EXECUTION Start program execution
Draw point	It must contain [string] "draw point" [string] (expr1,expr2) [string]	Draw point (5,3) draw point at (a/5,b) Now draw point (5,3) Now draw point (a,b) and go on
Draw line	It must contain [string] "draw line" [string] (expr1,expr2) [string] (expr3,expr4) [string]	Draw line (-3,-2) (5,1) Draw line from (-3,-2) to (5,1) Now draw line from (-3,-2) to (5,1) Draw line (-3,-2) (5,1) and continue
New pen colour	It must contain [string] "colour" [string] (expr1) [string] and the expression must be equivalent to a numeric value between 1 and 9.	Colour (4) New pen colour (4) now new colour is (4) green the colour changes to (4)
New pen width	It must contain [string] "width" [string] (expr1) [string] and the expression must be equivalent to a numeric value between 1 and 3.	Width (3) New pen width (3) Now new width is (3) thick width changes to (3)
Condition (on a block) *	It must contain the following lines: [string] "truth" (expr1) [string] "block start" [string] [commands] "block end" [string]	If truth (x>10) then Block start x new value is (-8) y new value is (y+1) Block end
Loop (for a block) **	It must contain the following lines: [string] "repeat" (expr1) [string] "block start" [string] [commands] "block end" [string]	Repeat (21) times block block start draw point at (x, sqr (4*(x^2-1))) draw point at (x, -sqr (4*(x^2-1))) x new value is (x+0.2) block end

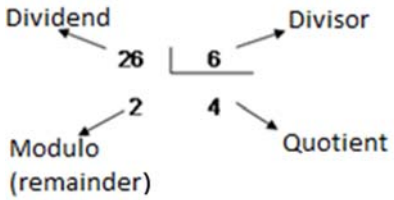
Command	Syntax requirements	Valid examples
Assign variable value	It must contain: identifier "value" [string] (expr1) [string] and the expression must allow to obtain a numeric value.	x value (-5) x new value is (-5) x new value will be (x-5) next x now takes value (x-5)
End program	It must contain "end" as line beginning, whether or not followed by additional strings.	End end program End execution End the program execution

* The expression must be a comparison that evaluates to true or false.
** The expression must generate a positive integer numeric value.

A-3. USING THE MODULE OPERATOR TO CREATE PATTERNS

The modulo operator is often little or nothing studied in primary or secondary education, but is **of great interest in programming**. It can have different uses being one of them creating numeric patterns, as we will see below.

The modulo, rest or remainder of a division between two integers is the numerical value that after multiplying divisor by quotient would allow to reach the dividend, i.e., Dividend = Quotient x Divisor + modulo or rest.



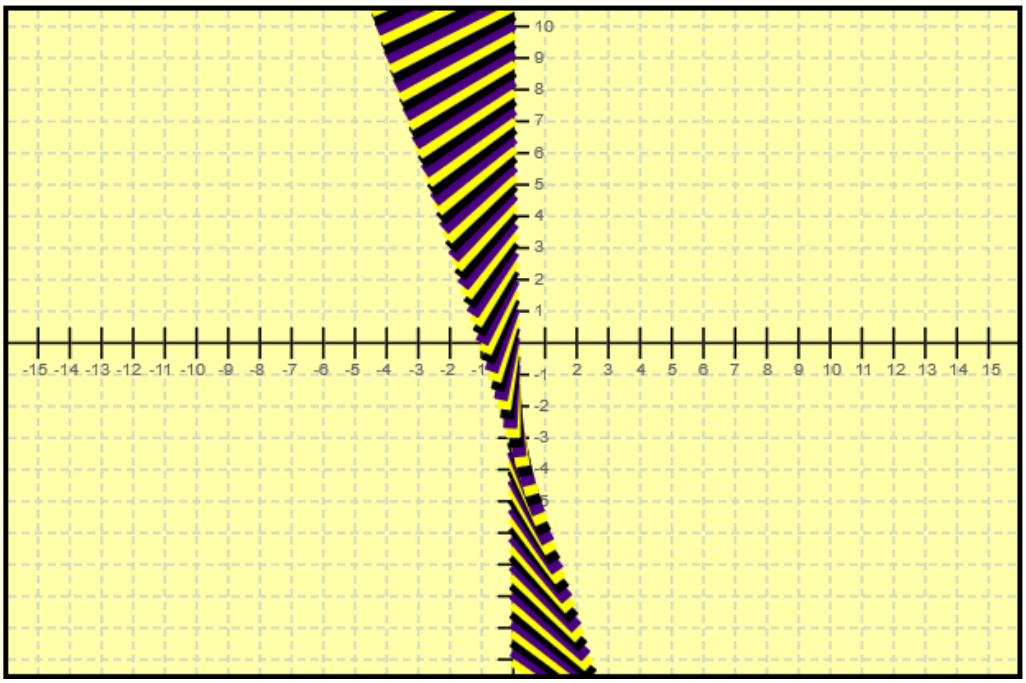
In Cartesia the remainder of a division between integers can be obtained using the modulo operator (%). For example (26% 6) returns the rest of the division, that is, 2. Although this operator uses the same percent symbol as a percentage, it is unrelated to anything about percentages. The reason this symbol is used is to follow the convention that many programming languages are adhered to.

It is an operator that may not be convenient to use with young children as it may involve some difficulties for them. The teacher must decide whether to explain this operator and at what age.

An example will show the utility of this operator:

```
-- Example of using the modulo operator to create patterns
Start program
New pen width (3) -- Drawing with thick pencil
x new value is (0) -- x coordinate origin point
y new value is (-10) -- y coordinate origin point
b new value is (3) -- Represents the colour jump, we will count intervals of 3: 3+3, 6+3, 9+3, 12+3, 15+3...
c new value is (-1) -- Variable to store the colour, we start from -1 for convenience
d new value is (-0.60) -- Variable that will control the horizontal movement of the destination point
Repeat (100) times execute block -- The number of repetitions is set to convenience to achieve the desired effect
Block start
c new value is (c+b) -- We create the numeric pattern; c takes values 3, 6, 9, 12, 15, 18, 21...
New pen colour (c%9+1) -- At each repetition changes the colour with pattern 3, 6, 9, 3, 6, 9, 3, 6, 9, 3...
Draw line from(x,y) to (2-d, y-2) -- As d and y are variables, the destination coordinates move; also the y coordinate of origin moves
d new value is (d+0.1) --Change for next loop
y new value is (y+0.3) -- Change for next loop
Block end
End program
```

This program makes use of variables and a repeat loop to draw lines that can create (with a little imagination) the appearance of a ladder or twisted figure:



Result of execution of the proposed example. Note that the appearance may vary depending on the device (screen size and resolution) on which it runs.

The program can be interpreted from the comments (which we have indicated in blue).

Here **c** is a variable with values -1, 2, 5, 8, 11, 14, 17, 20, 23 ... thanks to reassigning to **c** on each repetition (c + b), that is, **c** takes values (-1 + 3), (2 + 3), (5 + 3), (8 + 3), (11 + 3), (14 + 3), (17 + 3), (20 + 3) ... The value -1 is not used and it's unique purpose is to start the series using the first useful value (-1 + 3).

By affecting **c** with the modulus of division by 9 (operation c%9), we obtain 2, 5, 8, 2, 5, 8, 2, 5, 8, 2, 5, 8, 2 ... with which we already have a pattern of integer values in the range of the drawing colours, which will allow us to create a repeating colour pattern.

Finally adding 1 (we make c%9 + 1) we transform the series into 3, 6, 9, 3, 6, 9, 3, etc. that applied to the command *New pen colour* generates the pattern of colours yellow, purple, black, yellow, purple, black, yellow ... since in Cartesia 3 it is yellow, 6 purple and 9 black.

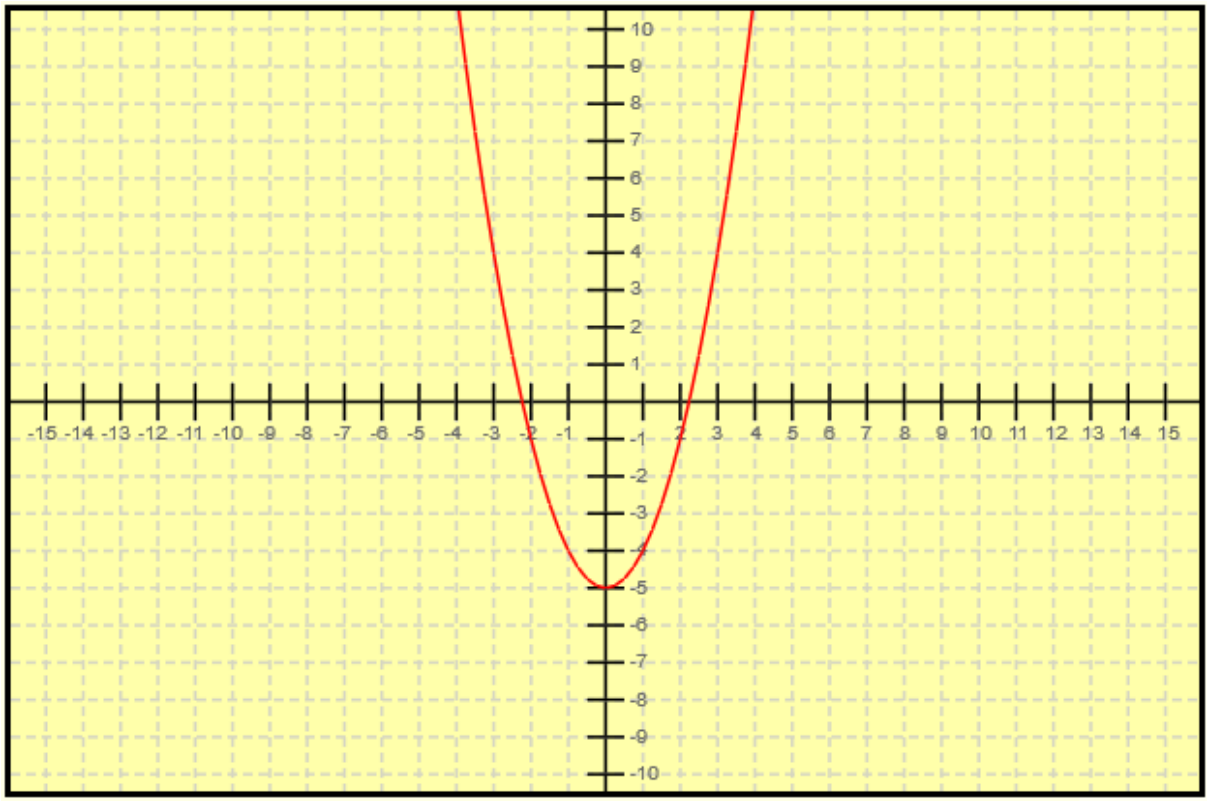
As we stroke the drawing with a thick line we achieve the desired visual effect.

A-4. APPROXIMATION OF CURVES USING SEGMENTS

The general part of the manual explains and gives an example, specifically a parabola, of how to plot curves with Cartesia from points that may be more or less separated. An alternative that can be used if desired is to plot **curves defined from line segments** of greater or lesser length at convenience. The idea for this is to define in a repeat loop the drawing of segments between two points of the curve in the visible area zone of Cartesia, taking in each repetition as the starting point of the trace the one that in the previous repetition was the end point. When compared to the use of points, it can be a more efficient method by requiring considerably fewer repetitions than plotting with quite joined points, although since Cartesia is commonly used to generate drawings that do not require processor intensive use, execution times in general will be indistinguishable to a human. Therefore, this efficiency improvement cannot be observed (in general).

The following program exemplifies the approximation of a curve path using segments:

```
-- PROGRAM THAT APPROXIMATES THE DRAWING OF A PARABOLA BASED ON SEGMENTS
-- Parabola y = x^2-5
Start program
New pen colour (1)
x new value is (-7) -- We start with x=-7 for convenience
Repeat (56) times execute block
Block start
Draw line from(x,x^2-5) to (x+0.25, (x+0.25)^2-5) -- Point to point stroke
x new value is (x+0.25) --Advance for next loop, increase 0.25 for convenience
Block end
End program
```



Result of plotting the parabola $y = x^2-5$ approximated by segments

A-5. OTHER MATHEMATICAL FUNCTIONS: ABSOLUTE VALUE, TRIGONOMETRIC FUNCTIONS, ETC.

Various mathematical functions were not included thinking of keeping the application simple. However, they may be incorporated in future versions depending on the pedagogical evaluation and the perceived demand from users.

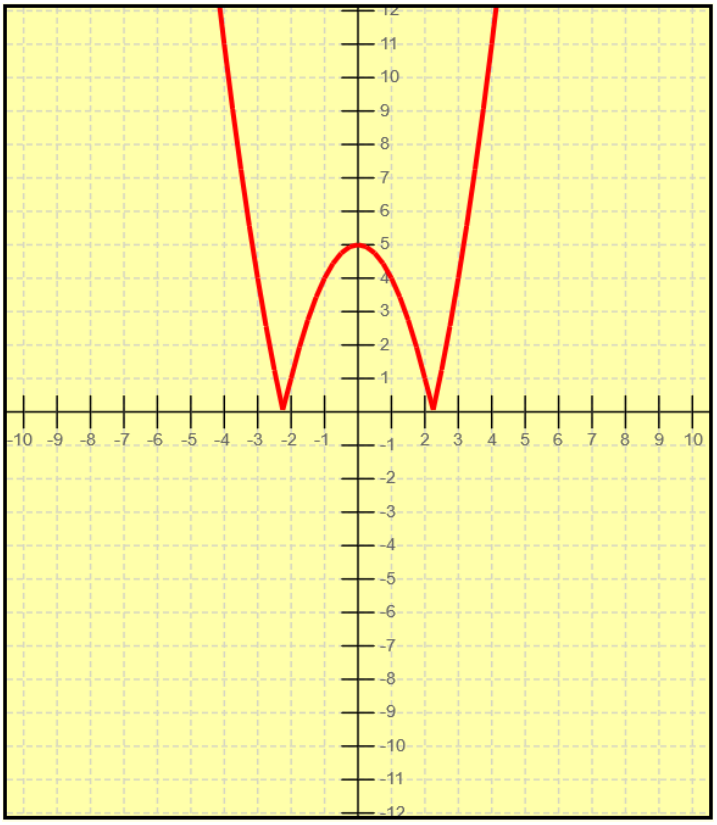
Among the functions not explicitly existing in Cartesia we have the absolute value, logarithms or trigonometric functions among others. However, some of the functions not included can be simulated from the computational power of language, which can be pedagogically interesting to stimulate computational thinking.

The absolute value is easily imitable using the conditional so it can be a great example of applying this command. Trigonometric and other functions can be simulated with a little more effort using Taylor

polynomials. Each teacher must decide whether to incorporate any of these “imitable functions” through language tools or to work without with them.

The following is a program that serves as an example of how to represent the absolute value function of the same parabola used earlier.

```
-- ABSOLUTE VALUE PROGRAM: APPROXIMATES THE ABSOLUTE VALUE DRAWING OF A SEGMENT-BASED PARABOLA
-- Parabola y = x^2-5
Start program
New pen width (2)
New pen colour (1)
x new value is (-6) -- current point x value
y new value is (x^2-5) -- current point y value
a new value is ((x+0.25)^2-5) -- next point y value
Repeat (64) times execute block
Block start
If truth (y<0) execute block
Block start
y new value is (-y) -- absolute value
Block end
If truth (a<0) execute block
Block start
a new value is (-a) -- absolute value
Block end
Draw line from(x,y) to (x+0.25, a)
x new value is (x+0.25) -- new current point x value
y new value is (x^2-5) -- new current point y value
a new value is ((x+0.25)^2-5) -- next point y value
Block end
End program
```



Result of plotting the function absolute value of the parabola $y = x^2-5$ approximated using segments.

A-6. EXPANDING THE USE OF LOGICAL OPERATORS AND THE CONDITION COMMAND

More ways of using logical operators

The operator “and” is equivalent to the operator **AND** of most programming languages. Many languages use && to represent this operator.

The “OR” operator is equivalent to the **OR** operator of most programming languages. Many languages use || to represent this operator.

The & and | operators also exist in many programming languages, but have a different meaning than && and ||, which often causes errors.

In Cartesia it is possible to use && instead of “and”, just as it is allowed to use || instead of “or”. However, its use is not recommended except for advanced users for two reasons:

a) We lose expressiveness and is less didactic.

b) It is easy to write & instead of &&, or | instead of ||. If this happens, the application may generate strange results without any warning message.

Complex logical expressions

In addition to simple conditions Cartesia also allows you to create conditions of arbitrary complexity with as many variables and logical operators as desired. In this case, it is suggested to recommend to students **the use of parentheses** to group conditions and thus make explicit the order in which they will be evaluated. Let's take an example. We want to establish the condition: a must be less than 10 or b must be less than 5, as well as b must be less than or equal to 3.

We will write this as follows (first case):

((a<10 or b<5) and (b<=3))

Keep in mind that that is different from this other (second case):

((a<10) or (b<5 and b<=3))

If a is worth 9 and b is worth 4 in the first case we would obtain FALSE, since:

(a < 10 or b < 5) returns TRUE

(b <= 3) returns FALSE

Both conditions are not met and therefore the final result of the first case is FALSE.

On the other hand, in the second case we would obtain TRUE, since:

a < 10 returns TRUE

b < 5 as well as b <= 3 returns FALSE

The first condition is met, therefore the final result of the second case is TRUE.

If the grouping manner is not specified using parentheses, how can we know how operations will be done? The way you operate is governed by **Boolean algebra**, which determines that AND runs before OR. If you are going to use complex logical expressions with students and want to explain the hierarchy of logical operators, you can do it didactically as follows:

If a value is 9 and b is 4, what result will we get for the condition (a<10 or b<5 and b<= 3)? To find out the order of operation, we substitute the ands as multiplication symbols and the ors as addition symbols. (a < 3 + b < 5 * a >= 2). Now we can say that the multiplications will be executed first and then the additions. We put parentheses grouping the multiplications and we are left with: (a<3 + (b<5 * a>=2)) Now we calculate the logical values of the multiplications taking into account that all the conditions must be met. In this case, b<5 and a>=2 would have to be fulfilled. As both are true, we substitute TRUE and we are left with (a<3 + (TRUE)). Now we only have sums that are equal to or. If any of the summation terms is true, the end result is true. Therefore, the result of this condition is TRUE.

A-7. DECIMAL ACCURACY AND PROBLEMS WITH DECIMAL OPERATION

Cartesia operates with the decimal precision proper to JavaScript and the IEEE 754 standard. The range of accepted integer values is approximately plus minus 9 trillion, and the range of floating numbers approximately plus minus 1.79 E+307. The number of decimals with which it operates is approximately 15 digits.

All these magnitudes far exceed the precision required for such an application. In Cartesia in general it is not necessary to operate with more than 3 decimals because the graphic precision of work makes it irrelevant to do it with greater precision. Many times programs will only use integers.

However, in one-off circumstances, the **loss of decimals** can lead to small distortions.

Therefore Cartesia operates with as many decimals as possible, but when a value is assigned to a variable it is always stored with a maximum of 6 decimals. If you have a higher number of decimals, the rounding is done to 6 decimals (maximum number of decimals stored in a variable).

The decimal operation on computers can be problematic as the internal numerical representation form can lead to seemingly erratic results. This may be useful to know in order to interpret somewhat strange situations that may arise.

For example, in an operation executed by a computer, 0.1 + 0.2 may result in 0.30000000000000004 instead of 0.3. The reason for this is the form of internal representation of the numbers used, which is not the decimal and may differ depending on the hardware and software used.

In general, Cartesia corrects or makes these decimal precision problems irrelevant or not significant, but there may be cases where a Cartesia program displays error messages or does not run due to problems in decimal settings.

Let's think for example that we did x = 1/3 + 1/3 - 2/3 which should return zero, but that due to decimal precision the result was -0.000001. If we try next something like Draw line from (sqr (x), 3) to (6,6) we will get an error of the type "*ERROR: check your code. Reference: Draw line command. Numerical values error. Possible division by zero or illegal operation. Please check the code.*", because it is wrongly interpreted to mean the square root of a negative number. With the added problem that our code is formally correct and it can be difficult to determine that the error is due to a **decimal precision** problem.

This type of errors are rare, in fact we may not be able to generate such an error even deliberately trying. However, they may appear on time and for teachers who regularly work with this software it will be good to be aware of it.

In general, these decimal accuracy errors are correctable. In the example above we would include a conditional to detect if the number is less than zero, and in that case set it to zero. Or detect if the number is very close to zero and set it to zero (this avoids admitting manifestly erroneous values):

```
If truth (x<0) execute block
Block start
x new value is (0)
Block end
```

```
If truth (x<0 and x>-0.001) execute block
Block start
x new value is (0)
Block end
```

A-8. CHARSETS. PROBLEMS WITH STRANGE CHARACTERS WHEN OPENING OR SAVING.

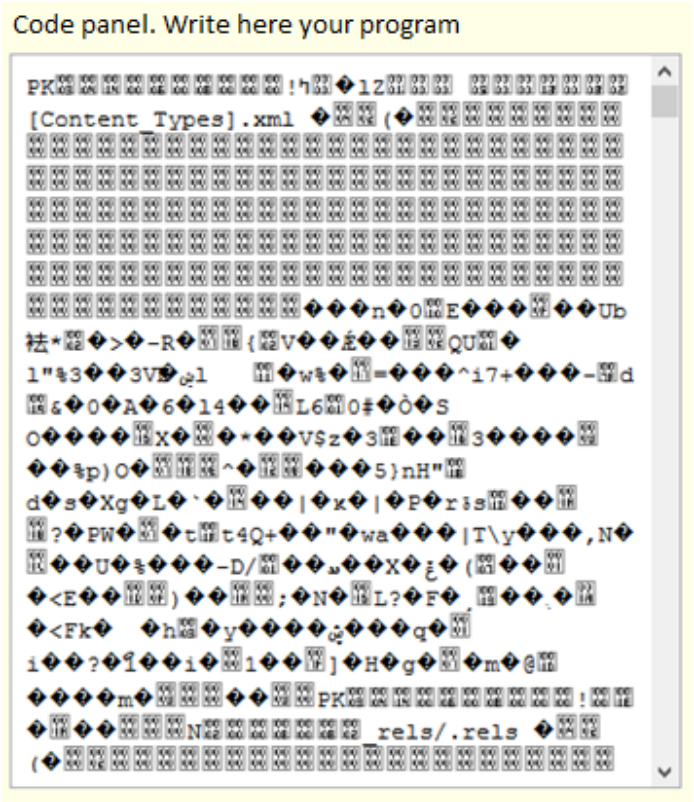
Incorrect Cartesia display. Solutions.

If strange characters are shown when opening Cartesia in the browser, we recommend trying:

- Check in the **browser settings** that the English language is chosen.

- Check in the browser and / or file settings that the **UTF-8** character set is chosen (or failing that ISO-8859-1).

Cartesia supports opening any plain text file regardless of its extension. By default projects are saved with extension txt and it is this extension that is recommended to use, but if we try to open a file with another extension, for example .data, .HTML, .css, .js, etc. will also be possible. Certain formats such as docx are not supported because they are not plain text, and trying to open a file of this type will show strange characters as in the following image.



Problems with strange characters in files. Solutions.

Even using txt files the character set used for file encoding can be problematic in certain cases, especially for characters not common in the Anglo-Saxon world such as special characters used in the Spanish language. When you save or open a project, apostrophes or other characters might be replaced with foreign characters, for example --penÃ¡s width instead of --pen's width.

Cartesia saves the files using the UTF-8 character set but there may be problems with the operating system, browser, etc. character sets.

In some browsers or operating systems there is an option to choose the character set with which to save a file: in this case it is recommended to choose UTF-8 (in default ISO-8859-1).

Cartesia is ready to open files encoded in UTF-8 but also supports ISO-8859-1 and other character sets.

Editors such as Notepad in Windows let you choose the character set to use ("Encoding") when saving a file. If you edit code with Notepad, we recommend that you save with **UTF-8 encoding**.

Editors like Notepad++ let you choose the character set to apply to a file, and transform it from one encoding to another.

A-9. COLLABORATION OF TEACHERS, PARENTS, ETC. WITH THE PROJECT.

Cartesia has been conceived, designed and written by Mario Rodríguez Rancel under the direction of Professor Anselmo Peñas Padilla from the Department of Languages and Computer Systems, UNED Open University (Madrid, Spain).

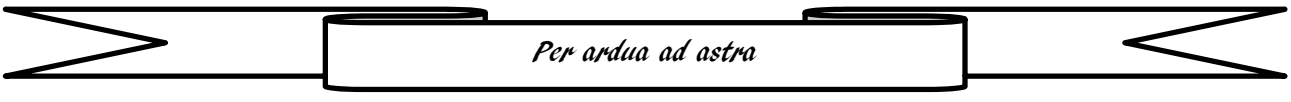
The collaboration of teachers, trainers, parents and educators in general who want to collaborate with the creators of Didac-Prog Cartesia with both a focus on the application in particular and on **teaching programming to children** in general will be appreciated. Collaboration can be embodied in various ways.

The simplest ways of collaboration would be the sending of suggestions of improvement, communication of possible bugs or failures of the application and the experiences with it.

More advanced formats could be to participate as collaborators in the evaluation of Cartesia as an educational tool, dissemination of educational experiences and materials, evaluation of results of teaching programming to children, design of methodologies and training materials, etc.

On the website aprenderaprogramar.com, within the programming section for children, updates and all the available material related to Cartesia will be published.

To contact you can use the email contacto@aprenderaprogramar.com or the contact section of the website where you can find the telephone and postal address.



Explanatory note: all the terms that may give rise to gender doubts in this manual should be understood as gender neutral. At least that has been the spirit of the editors. If you use this text in an educational center with specific gender requirements, you may need to adapt it.