Introduction to programming with Lego Mindstorms

Coordinating teachers: María Luisa Porto Llamas and Yago Martínez de Fuenmayor
Authors of the work: Alejandro Rico and Alexander Mychlo from 1st Baccalaureate
IES Severo Ochoa
2021-2022

Abstract:

This research work is based on the study of the Fundamentals of *programming* and its applications to different areas of technological education, such as mathematics. The different types of programming and languages will be studied, to point out their differences. After having chosen *python*, its bases will be studied and applied in a final Lego Mindstorms project, in which the engineering and the program behind it will be explained.

Keywords: Programming, Lego Mindstorms, Python.

Abstract:

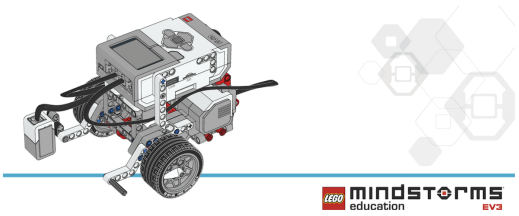
This research project is based on the study of the fundamentals of *programming* and its applications to different technological education, like in mathematics. Different programming types and languages will be studied, with the goal of pointing out their differences. After choosing *python*, its bases will be studied and will be applied in a final project of *Lego Mindstorms* in which the engineering and code behind it will be explained.

Keywords: Programming, Lego Mindstorms, Python.

Index:

Introduction

During the investigative value of our work we have discovered and documented the necessary materials to start this project:



Physical:

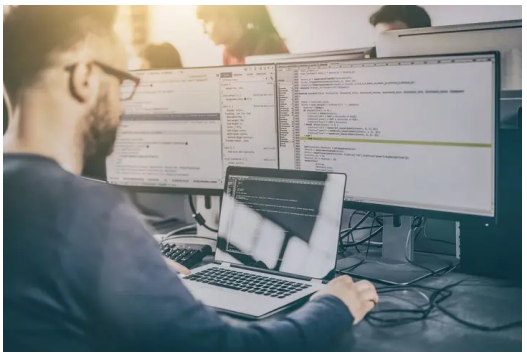
- Lego Mindstorms EV3 Basic Set.
- Charger.
- MicroSDHC (MicroSD card with a maximum capacity of 32GB).
- Computer or laptop with Internet access.

Digital:

- Microsoft Visual Code.
- Documentation.
- Official Lego Mindstorms EV3 software for programming with blocks.
- Latest version of Python.
- .iso MicroPython for the MicroSD card.
- Procedure

Purpose

The purpose of this project is to open a field of research on the possibilities of the lego mindstorms EV3 kit.
In this work we are seeking to set a precedent for future applications of this set in the institute. In addition to exposing the reasons why we should use one system before the other, with its pros and cons.
After assessing this step we will delve into the basic functions of the kit so that in the future the bases are completed to be able to do more complex work in this area.
In this project, in addition to what is mentioned above, we want to make engineering solutions to the problems that have arisen in the course of the project. This not only includes the software part but also the robot construction part, as we will see in the body of the project.
To conclude, our project also has the purpose of disseminating booming languages such as python, in addition to the world of software in general, and its wide possibilities, even professionals.



The project also has the purpose of teaching the possible use of Python and its relationship with mathematics, where the use of Python libraries such as *math* and *SymPy* to carry out mathematical concepts taught up to the level of 1st Baccalaureate.

Body of work

Using Git and Github to create knowledge for future projects.

Git is a tool used in the professional and scientific field of technological projects. It is used to organize, save and document the process that has been carried out among a group of people, since it allows each one to make their changes on their own and then “push” them to the server where the project is saved.

Those changes that a programmer makes are called "commits" and when sending it to the main server you can see who has made those changes and when they have done it. In addition, the server saves the status of the project in each commit, and therefore allows documenting the process that has been carried out.

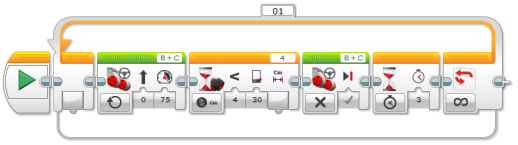
The Github platform is a web page that serves as a portal to the servers where the projects will be stored. It is similar to Google Drive cloud storage, but from a Git perspective.

Github turns out to be very useful for the workplace as various companies like Microsoft and Netflix use it to create their programs and applications. It is also useful for the creation, improvement and discovery of open source programs, which are programs where anyone can change the code, as long as the owner(s) or managers of that project have verified the changes, similar to what happens in Wikipedia. It turns out to be very important for our project since the program that is installed on the microSDHC is called ev3dev and it is just an open source program that has been verified by Lego.

https://github.com/ev3dev/ev3dev

Differences between block programming and python.

When we start with Lego Mindstorms EV3 and its documentation we find the official Lego Mindstorms program that will teach us how to program with blocks (image 1.1).



1.1 Example of programming with blocks.

That is why we began to look at the possibility of finding a way to program with an environment with which we are already accustomed, programming by code. The recommended environment to work with code and Lego Mindstorms is python. Here we can see how we solved the problem and in this code the robot will advance 1000 mm and turn 360 degrees to the right and will return to its initial position after making another turn with the same angle (image 1.2).

```
robot.straight(1000)
robot.turn(360)
robot.turn(-360)
```

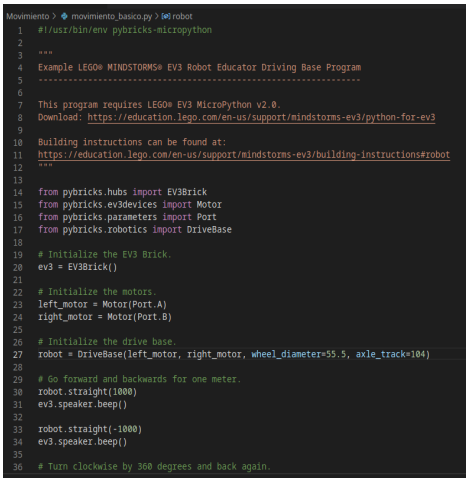
1.2 Example of nonlinear programming by code in Python.

Python is a very simple programming language that can help us delve into more complex languages such as C + +, Java... Python has the positive part that it is very simple, but it also has its negative part, which is that it takes time. Much slower response time than the languages mentioned above. Python has another face for which it is well known, since it can be used for the world of hacking, whether ethical or not (ethical hacking is the one whose purpose is not to harm users).
Likewise, programming by blocks is a very intuitive, easy and visual way to start having resources to be able to program with code later.
From our point of view, programming with code should be the way to do this project, since it can give you experiences that may even be valid for a future job or professional.
Programming with blocks can be a good idea, from our point of view, for people from the first year of ESO to the third year of this same cycle. Programming in code can be interesting from the third or fourth year of compulsory secondary education.

Getting Started with the Educator Robot

The first step is to complete the Lego Mindstorms EV3 basic set by following the steps within the manual to create the basic robot.

After we have built the basic robot and made sure that the program included within it works, we can continue to use the official Lego Mindstorms EV3 program to program with the blocks and complete the exercises within it.
Next, to venture further into what can be done with this project, install the MicroPython on the MicroSD and insert it into the robot's slot (it has to be powered off). Turning it on will see that the main block will boot into a Linux partition called “ev3dev” which allows you to stream Python files and use them. If you want to go back to the previous basic system, simply removing the MicroSD card when the robot is off will reboot into it.
Next, to venture further
Using a Microsoft Visual Code extension called “LEGO MINDSTORMS EV3 MicroPython” you can connect the main block to your computer and stream Python files to run within a single program.



2.1 First code of the educational robot.

Python principles applied to the robot.

To be able to continue with the project we need some very basic notions of the programming language that we are using, python. This language, as mentioned above, is one of the simplest you can find right now. This section will only deal with the theory necessary to understand the programs for the robot. There is a lot of documentation on this subject and very interesting on the subject so we encourage you to inform yourself.

Python libraries and theory applied to mindstorms programming.

Declaration of use of libraries and what each of them contains.

Python itself is a very limited language, but there are authors who add more features to this program. These groups of features are called libraries. We can find from simple libraries to create programs to download YouTube videos, through others that automatically remove the background from images, to others that help create facial recognition programs. These libraries occupy all imaginable fields, including, as mentioned above, the world of hacking.

As has already been mentioned in the previous section, when installing the Visual Studio Code extension "LEGO MINDSTORMS EV3 MicroPython" the libraries that contain what will help us create the programs for the robot are also installed.

Next we will see what each of these libraries do, to be able to observe what can be done with the kit. In total there are nine libraries but we will only talk about

Programmable Hubs

```
from pybricks.hubs
```

3.1 Declaration of the Programmable Hubs library.

This library does everything related to the EV3 Smart Brick. The most important function that this library fulfills is the initialization of the block, which we will talk about later. In addition to initializing the EV3 Smart Brick, this library also takes care of other very interesting tasks such as the use of the base buttons,

3.2 Buttons of the EV3 Smart Block

, for example, when pressing the central button the block does something. In image 3.2 we can see that the block has

a light (brick status light), it can also be controlled with this library by putting any color. Another very interesting function of this library is the speaker integrated in the block (speaker). We have used this function a lot to know what point of the program was failing, since if the "beep" sounded until the programmed point, we knew that until then everything was fine.

```
def
MoveAndRotate(Distance,Degrees)
:

    robot.straight(Distance)
    ev3.speaker.beep()

    robot.straight(-Distance)
    ev3.speaker.beep()
```

3.3 Code using the Programmable Hubs library.

Finally, it can also be used to put images on the screen, which we have used for the color sensor that we will mention in section 5.5.4. You can also check the status of the battery and its current charge.

EV3 Devices

With this library we are going to use the motors and sensors from the Lego Mindstorm EV3 kit. This library will be essential for the movement and operation of the sensors (which we will detail in section 5.5.4. To talk about this library we will need to know what units the robot handles:

-Time: milliseconds (ms)
-Angle: degrees (deg)
-Angular velocity: degrees per second (deg/s)
-Distance and dimensions (wheel radius): millimeters (mm)
-Relative distance (sensors): percentage (%)
-Frequency (speaker): Hertz (Hz)
- Current (battery): milliamps (mA)

Knowing this, this library has two main parts, sensors and motors. The motors will also be divided into three parts: Measurement (Measuring): which can measure the speed of the motor and its angular speed. Stopping (Stopping) It makes the movement slow down in two ways, slowing down little by little or suddenly And finally Action (Action) that makes the motor do actions.These actions such as moving forward, turning etc…

```
def
MoveAndRotate(Distance,Degrees)
:

    robot.straight(Distance)
    ev3.speaker.beep()

    robot.straight(-Distance)
    ev3.speaker.beep()

    robot.turn(Degrees)
    ev3.speaker.beep()

    robot.turn(-Degrees)
    ev3.speaker.beep()
```

3.4 Example of using the EV3 devices.

We will continue the section on sensors in section e, since it is more complex.

Robotics

This library is also very important as with this we will give you the information of the robot that we have built around the Smart Block Ev3. The

most used function is the DiveBase(left motor, right motor, wheel diameter, axle track). This function will be further developed in section d.2.

```
robot = DriveBase(left_motor,
right_motor,
wheel_diameter=55.5,
axle_track=104)
```

3.5 Example of using the Robotics library.

Parameters and Constants

In parameters and constants we will find the parameters that we need to introduce to our program so that it understands it.

In this section we will tell the Smart Block where the different "devices" are located so that it executes the orders given through the EV3 devices libraries. the block of d.2)

```
left_motor = Motor(Port.A)
right_motor = Motor(Port.B)
```

3.6 Example of using the constants and parameters to name motors.

```
front_line_sensor = ColorSensor(Port.S1)
```

3.7 Example of using the constants and parameters to name sensors.

tools

In the tools library we will find the tools to be able to work with time. This is very useful for using time instead of using the mm measurements in the motor actions. You can also measure how long the program takes to perform Actions.

media

This library acts as a complement for *Programmable Hubs,* and its screen section. It gives you the ability to project ready-made icons and to change the text font. It also makes use of sound files which are very useful when working with sensors and you can't constantly stare at the screen.

the libraries

The first step in any python project is to declare the libraries that we are going to use. There are two ways to do this step, declaring the entire active library, or using the part of the library that interests you.

```
desde pybricks.hubs matter EV3Brick
from pybricks.ev3devices import
Engine
from pybricks.parameters import
port
from pybricks.robotics import
DriveBase
```

4.1 Library declaration example

In this case we are declaring the *Programmable Hubs* that we import only the Smart Block.library *EV3 Devices* we have imported the motor to make the robot move, if we had also required a sensor we would have also put it here. For parameters we have imported only the motors and finally we give the robot information with the *Robotics library.*

Here we see how we can import more than one part of the library.

```
desde pybricks.ev3devices
import Motor, ColorSensor
from pybricks.parameters import
Port, Color
```

4.2 Library declaration example

We have imported the color sensor and color parameters so that the robot knows what color it is seeing when it receives data from the sensor.

In this other way we see how we load the entire library. The * can be replaced by an "everything".

```
desde  pybricks.ev3devices  import
*
from pybricks.parameters import *
from pybricks.robotics import *
```

4.3 Example declaration of complete libraries.

When we require a library in the code, we write the library function that we want to use after the object of that action.

```
    robot.straight(Distance)
    robot.turn(Degrees)
    robot.turn(-Degrees)
```

4.4 Use of libraries in the code.

Here we make the robot use the *EV3 Devices* to send the instruction to turn and go straight. We remember that the *EV3 Devices* is in charge of the motors and sensors.

Python Functions

Functions in the Python programming language are defined using the word "def" followed by the name of the function, and in parentheses the variable(s) needed, if any, and a colon at the end.

```
def Move(x,y):
    [...]

def Initialize():
    [...]
```

5.1 Example of function use in Python.

This would be an example of how to declare a function in Python, but even so we must consider that to indicate the code that is subject to the function we need to leave an indentation margin so that Python differentiates the contents of the function

```
def Move(x,y):
        [...]    #  Inside  the
function

[...] # Outside the function
```

5.2 Example of using a function in Python.

Normally the editing program does this automatically when pressing "Enter" from the colon, but it is necessary to take it into account since it can cause errors when programming.

The usefulness of functions in programming is to relate code to each other and declare a way that the computer or program can call that code in a concise and fast way, without having to repeat the same lines of code.

```
def Example(x,y)
    [...]
Example(1000,360)
Example(100,90)
Example(2000,45)
```

5.3 Example of using a function in Python with data.

Use of control or conditional structures and loops.

Control structures allow you to modify the execution flow of a program's instructions. This means that depending on what they are, a function will do if a part of the code is executed.

Python gives you several resources to modify the flow. We are going to see the three that we have used the most in this project.

```
if front_line_sensor.color() == Color.BLACK:
 ev3.speaker.beep()
 robot.straight(-10)
 robot.turn
```

6.1 Example of using an "if" control structure.

In this example we can observe the use of the "if" control structure, which can be translated into Spanish by "si". A variable (explained in point 5.4.1.2) which in this case is the previous sensor of our robot, detects that if the color you are looking at is black to go back 10mm, and then rotate.

The next method is the "else" it is used if something is not included in the previous "if" it will execute the code below this conditional. In the previous example it could have been put that if the color was not black it would have continued straight.

The last method and the one that has been most useful in this project is the "While"..

```
while True:
   color_recognition = front_line_sensor.rgb()
   ev3.screen.print(color_recognition)
```

6.2 Example of using a "While" control structure.

"While" can be translated as "while". In the previous project we used it so that when a condition is true, that code is constantly executed. As we can see, the color that the sensor has detected will be printed on the screen, using the *Programmable Hubs library.*

EV3 Smart Brick port selection.

After understanding the theoretical principles of python we will put it to the test with examples using the codes we have made. After having initialized the libraries, we will start by declaring, with the help of the constants and parameters library, which objects external to the Smart Block we have and on which ports they are located.

7.1 Motor and Sensor Connections to the EV3 Smart Brick.

The EV3 Smart Brick has eight different ports, four input ports identified by numbers, which will be used to connect sensors.

7.2 Smart Block Input Ports.

and four output, identified by letters that will serve to connect motors.

7.3 Smart Block Output Ports.

Next we will comment on the code in which we tell the block where each thing is and what it is.

```
# Initialize the motors.
left_motor = Motor(Port.A)
right_motor = Motor(Port.B)

# Initialize the color sensor.
front_line_sensor                    =
ColorSensor(Port.S1)
```

7.3 Initialization of motors and sensors.

As we already know, we have created a variable called left_motor and inside it we have stored the information that what is connected to port A is a motor and that there is another to port B. We have also mentioned that there is a color sensor in the S1 port.

Initialization of the block and basic movements.

After having declared the position of the motors, we must declare the shape that our robot will have, so that our Ev3 Smart Block can know how to execute the commands that we send it.

```
robot = DriveBase(left_motor, right_motor,
wheel_diameter=56, axle_track=152)
```

7.4 Block initialization.

Here we can see how using programmable hubs we have named that the motor has two motors, which we have previously inserted in a variable declaring its position. Then we declare the diameter of the wheel, which in this case is 56mm and the distance between the two wheels (the axle) is 152mm.

The movement in EV3 is very simple, as we have seen before, inserting the desired command (straight, turn... etc) after the block initialization variable, in this case as we have seen before robot.

Sensor initialization

Finally, one of the benefits of using python ev3 is its great variety of sensors, which we will explain in section 5.5. The use of the sensors is very similar to that of the motors, in which, as already explained in the point of Port Selection of the Intelligent Block Ev3.

The big difference between motors and sensors, aside from the ports being different, is that they use a much larger number of libraries.

```
front_line_sensor = ColorSensor(Port.S1)
```

8.1 Initialization of a color sensor.

First, as we have done before, we put the color sensor in the S1 port. We are facing a color sensor.

```
color_recognition = front_line_sensor.rgb()
    ev3.screen.print(color_recognition)
```

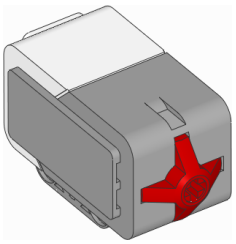
8.2 Using a color sensor.

Here we can see how we create a variable called color recognition and use the sensor previously called front line sensor. We have put the .rgb apposition on this sensor so that it tells us on the rgb scale (color composition in terms of the intensity of the primary colors of light) what color it is observing. Finally this observation will be printed on the screen.

Tests with different sensors.

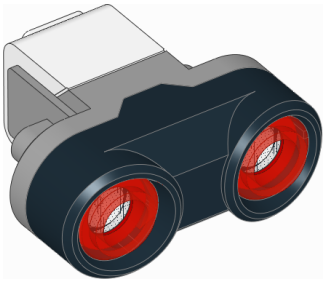
Touch Sensor The touch

sensor is the simplest sensor of all, as it answers “true” or “false” when asked if it is being pressed or not. It is very useful to use as a button.



Ultrasonic

sensor This sensor uses ultrasonic sound waves to measure distances between an object and the sensor. It can also measure distances between two ultrasonic sensors.



```
obstacle_sensor                    =
UltrasonicSensor(Port.S4)
while   obstacle_sensor.distance()   >
300:
        wait(10)
```

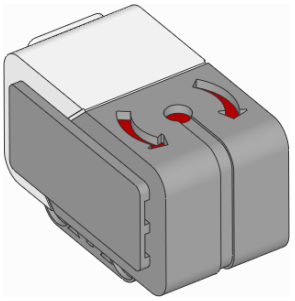
Here we can see how if the object that the sensor is observing is less than 300 mm away the robot will stop for 10 seconds, and as long as the object is still there it will not move.

gyro

sensor has two fundamental functions: Reading the angular velocity, the angle at which it is placed.

```
gyro_sensor_value                  =
gyro_sensor.speed()
```

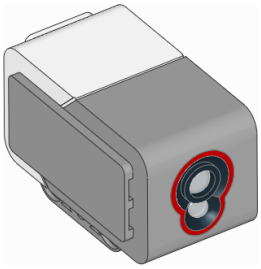
Here we can see a case where the gyro sensor is reading the angular speed.



Color

sensor The color sensor is in turn the most complex sensor and the most useful. The color sensor, as explained above, will read the color of a surface using the RGB method (color composition in terms of the intensity of the colors). primary colors of light).

This can have many applications, although it is mostly used for the following: measuring the color of surfaces, measuring the intensity of light in a room, the reflection of a material given the intensity of light, and finally reading the RGB code of a color.



Mathematical principles of Python and its Python expression

, in addition to being able to be used to declare actions to a robot, it can be used as a tool for mathematical calculations.

In the following sections we will see the possible applications towards the mathematical field, trying to explain the basic concepts and relating them to concepts already given up to the level of 1st year of high school.

Python mathematical libraries

As mentioned in the previous section, the libraries serve to link and standardize the language code since they allow the use of variables, functions and structures already created previously. In this case, we are going to use several specific libraries that allow us to do more complex operations, as would be the case with polynomials, functions, and complex numbers.

library will be used *math* since it contains the most basic concepts of mathematics. We are also going to use *SymPy which* is used to solve simple problems such as equations, vectors, polynomials, etc.

To install the math libraries we will need to have

Python already installed and have access to the terminal. After doing that we are going to put the following commands in the terminal part of VSCode:

```
pip install sympy
```

After installing them we can include them in the projects as follows.

```
import math
import sympy
```

Mathematical signs and operations

To see the result of the operations we are going to use the print() function

since it can express values automatically.

The basic signs follow the general structure used in mathematics like +, - , * (multiplication) and / (division) but more complicated expressions such as roots, logarithms and powers since they are difficult to express digitally due to their lack of a sign on the keyboard.

In these cases we are going to express them as follows:

Powers and Roots

```
n = 2 # Basis
x = 5 # Exponent

# ** Indicates the power and means that the next number is going to be its exponent.

print(n**x) # 32
```

In the same way we can express the roots since we know: $\sqrt[n]{x} = x^{1/n}$ and therefore we can calculate the roots as follows:

```
n = 2 # Basis
x = 1/2 # Exponent

print(n**x)                    # 1.4142135623730951
```

library *math* to do this using functions like sqrt() and pow() (acronyms for square root and power)

```
import math
```

```
n = 2 # Basis
x = 5 # Exponent

print(math.pow(n,x)) # 32.0
print(math.sqrt(2))            # 1.4142135623730951
```

It is important to note that these functions return a value with decimals. This is due to the fact that in programming languages numbers with (float) and without decimals (int) are different units since numbers with decimals require more bits to be represented by the computer. Fortunately, this loophole is not required in Python since the computer automatically finds out the unit of a variable, whereas in other programming languages that is not the case.

Random

numbers If we wanted to include random numbers so that we can use them in a program to change the result, we can use the *random* which gives us a pseudo-random result:

```
from random import  *

print(random()) #Returns a value in [ 0.0, 1.0)

print(uniform(2.0,10.0)) #Returns a value in [2.0, 10.0]

print(randrange(10)) #Returns integer values 0 and 9 (not including 10)

print(randrange(0, 101, 2)) #Returns integer values from 0 to 100 that are even
```

```
print(choice(['rock', 'paper',
'scissors'])) #Returns one of
three values
```

Equations

Equations are evaluated with the *solve(equation)*:

```
from sympy import *

x = symbols('x') we want to
find out

equation = Eq((2*x - 4),0) #
2*x - 4 = 0

print(solve(equation)) # [2]
```

Logarithms Logarithms

could be represented in a similar way as above using the *math*:

```
import math

n = 2 # Basis
p = 8 # Argument

# math.log(Argument, Base)

print(math.log(8,2)) # 3
```

But we could also create our own function using the *sympy* that allows us to calculate the logarithm using equations:

$$\log_n(p) = x \implies n^x = p$$

```
import sympy

x = sympy. symbols('x')

#The function we are going to use
def Logarithm(Base, Argument):
                        eq       =
(sympy.Eq(Base**x,Argument))
   print(sympy.solve(eq))
#Result
Logarithm(n,p) #3
```

Use of variables and relation to

algebra Variables are not only used in python, but in all programming. Variables are words, letters, or numbers that contain information within them.

```
left_motor = Motor(Port.A)
right_motor = Motor(Port.B)
```

In this image we see that we have called “left_motor” any motor that is in port A of the smart block. By using this we no longer need to be all the time naming the motor that isin port A.
This is the same as using a system of equations, solving with the substitution method, but the other way around since we start from x= 5+3y and then use x, while in the system of equations like the following, the x is omitted to put what it is equivalent to. This is done to avoid errors when using a lot of typed text, and makes the job of reading and understanding the code easier.

$$2(5+3y)+y=3$$

$$10+6y+y=3$$

$$10+7y=3$$

$$y=\frac{3-10}{7}$$

$$y=\frac{-7}{7}=-1$$

```
robot = DriveBase(left_motor,
right_motor,
wheel_diameter=55.5,
axle_track=104)
```

Here, continuing with the previous example, left_motor by Motor(Port.A) and right_motor is by Motor(Port.B)

Algebra

Polynomials with integers

Factorization

Using the Previous concepts of expression of exponents, roots and logarithms in the Python language can be followed by more advanced concepts as would be the case with polynomials.

Suppose we have this polynomial and we wanted to factor it:

$$2x^2-5x-3$$

Then we could use the following code:

```
from sympy import *

x = symbols('x')

polynomial = 2*x**2 - 5*x - 3
#Declaration of the polynomial

print(factor(polynomial)) #(x -
3)*(2*x + 1)
```

Operations $2x^2-5x-3=$

Knowing how to factor polynomials, we can now carry out various operations such as addition, subtraction, product and quotient:

```
from sympy import *

x = symbols('x')

a = 2*x**2 - 5*x - 3
b = 4*x**2 + 6*x + 2

print(a+b) # 6*x**2 + x - 1
print(ab) # -2*x**2 - 11*x - 5
```

Now, as we can see, addition and subtraction are easy to understand, but the difficulty in the multiplication and division process is more complicated to explain:

```
print(Poly.mul(poly(a),poly
(b)).as_expr()) # 8*x**4 -
8*x**3 - 38*x**2 - 28* x -
6
print(div(a,b)) # (1/2,
-8*x - 4)
```

The multiplication needs to transform the expression we have into a *Poly* in order to do the multiplication, so simply converting the two polynomials to that object and then de-convert the result with *as_expr()* we get the multiplication as an expression.

Division is a function that first returns the quotient and then the remainder of the division, therefore the result (½, -8*x - 4) would have the following mathematical formula:

$$2x^2 - 5x - 3 = \left(\frac{1}{2}\right)\left(4x^2 + 6x + 2\right) + \left(-8x - 4\right)$$

System

Systems are evaluated with the function *solve((equations),(variables))* same as in one-variable equations but join the equations with a parenthesis and the variables used too:

```
from sympy import *

x = symbols('x')
y = symbols('y')

a = 2*x - y
b = 4*x + 6*y

eq1 = Eq(a,3) # a = 3
eq2 = Eq(b,-2) # b = -2

print(solve((eq1,eq2),(x,y)) )
```

Would $\begin{cases} 2x - y & = 3 \\ 4x + 6y & = -2 \end{cases}$ be:

Inequalities

Inequalities are found using a method similar to equations, they can be found in two different ways, using *solve()* or *solveset()*

```
from sympy import *

x = symbols('x')

of inequalities
a = x**2 + 6*x < 0
b = x**2 + 6*x > 0
c = x**2 + 6*x <= 0
d = x**2 + 6*x >= 0
```

Solve( )

```
print(solve(a,x,S.Reals)) #(-6 < x) & (x < 0)
print(solve(b,x,S.Reals)) #((-oo < x) & ( x < -6)) | ((0 < x) & (x < oo))
print(solve(c,x,S.Reals))   #(-6 <= x) & (x <= 0)
print(solve(d,x,S  .Reals)) #((0 <= x) & (x < oo)) | ((-oo < x) & (x <= -6))
```

Solveset()

```
print(solveset(a,x,S.Reals))
#Interval.open(-6, 0)
print(solveset(b,x,  S.Reals))
#Union(Interval.open(-oo, -6), Interval.open(0, oo))
print(solveset(c,x,S.Reals))
#Interval(-6, 0)
print  (solveset(d,x,S.Reals))
#Union(Interval(-oo,      -6), Interval(0, oo))
```

We can see that .*open* means that the set is open and if it lacks it means that it is closed.

Systems can only be represented using *solve()* since *solveset*() cannot solve systems:

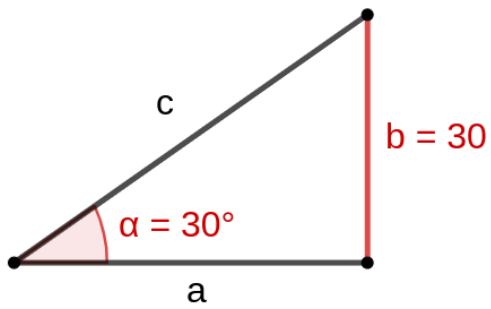
```
e = x - 2 > 0
f = x - 1 < 3

print(solve((e,f),x,S.Reals)   )
# (2 < x) & (x < 4)
```

Trigonometry

Solving Triangles

Python does not have a visual trigonometry representation built into the language, but it can compute the values of *sin, cos, tg, arcsin, arccos, arctan,* and using the method of solving equations and systems we can solve triangles knowing these data:

$$\sin(\alpha) = \frac{b}{c} \qquad \cos(\alpha) = \frac{a}{c} \qquad \tan(\alpha) = \frac{sen(\alpha)}{\cos(\alpha)} = \frac{b}{a}$$




```
from sympy import *
from math import *

a   =   symbols('a')   #Contiguous
leg

b = 30 #Opposite leg

c = symbols('c') #Hypotenuse


#Suppose   we   have   a   30   degree
triangle where we only know its
opposite leg.

eqC = Eq(sin(radians(30)), b/c)
#sin(30) = b/c
eqA   =   Eq(tan(radians(30)),b/a)
#tg(30) = b/a
print(solve(eqC))
#[60.0000000000000]
print(solve(eqA))
#[51.9615242270663]
```

To find a trigonometric result you need to use *sympy* and *math* because the equations in *sympy* are in radians, but since we need the result In degrees we are going to use the *radians()* from *math* which passes the equivalent of 30 degrees in radians (30° × π/180 = 0.5236rad) to the function.

We could also have solved both parts of the triangle using a system, as mentioned above:

```
print(solve((eqC,eqA),(a,c)))
#   {c:   60.0000000000000,   a:
51.9615242270663}
```

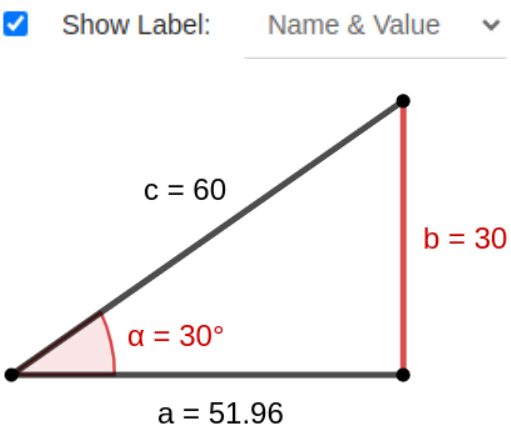
Note that although the result does not have decimal numbers, the result is still going to include decimal numbers since when it comes to trigonometric

operations, *solve()* returns the value as a list of *floats* and to make the decimal numbers go away we need to call the number explicitly:

```
valueC = solve (eqC)
valueA = solve(eqA)

print(valueC[0]) # 60.0
print(valueA[0])                #
51.9615242270663
```

```
print(round(valueC[0])) # 60
print(round(valueA[0])) # 52
```

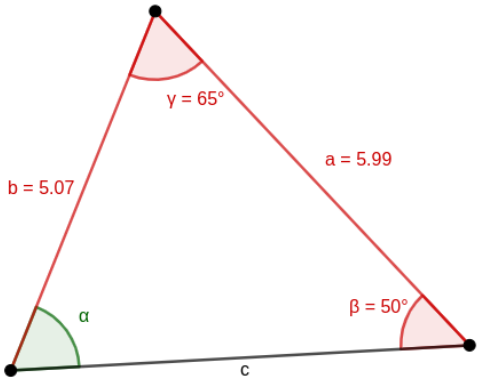
We can check if the result is correct, when putting the data inside the GeoGebra application, if it were, the values would be equal to those that we have calculated when selecting the segments and in settings put that its value is shown.



Sine Theorem

Using the previous methods we can find out and solve triangles only knowing at least two of their angles and one opposite side, or on the contrary two sides and one opposite angle.

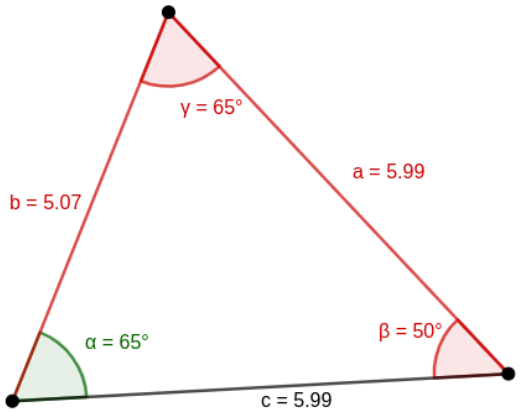
$$\frac{a}{\alpha} = \frac{b}{\beta} = \frac{c}{y}$$




```
from sympy import *
from math import radians, degrees


#Side a and its opposite angle
a = 5.99
alpha = symbols('alpha')

#Side b and its opposite angle
b = 5.07
beta = 50

#Side c and its opposite angle
c = symbols(' c')
y = 65

eqA                                =
Eq(a/sin(alpha),b/sin(radians(beta
))) # a/sin(alpha) = b/sin(beta)
eqB        =        Eq(b/sin(radians
(beta)),c/sin(radians(y)))        #
b/sin(beta) = c/sin(y)

print("alpha   =",   solve(eqA))   #
alpha      =      [1.13149852804901,
2.01009412554079]
print  ("c  =",  solve(eqB))  #  c  =
[5.99832101329159]
```

```
valueA = solve(eqA)
valueB = solve(eqB)

print(degrees(valueA[0]))          #
64.83009018247306
print(degrees(valueA[1])     )     #
115.16990981752696
print(valueB[0])                   #
5.99832101329159
```

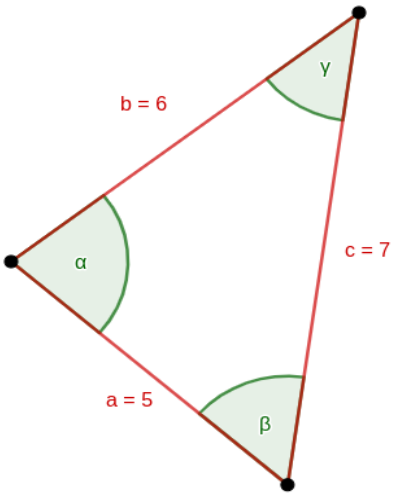
As we can see, we have used the principles of the previous section. In addition we had to convert the angles from *alpha* to degrees and there were two values since *sin(65) = sin(115)*



Theorem of cosines

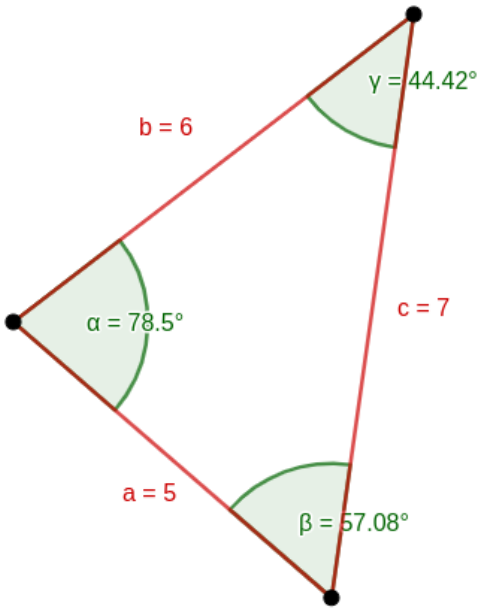
We can do the same using the theorem of cosines, which will allow us to find out the data of a triangle just knowing its three sides or two sides and an angle:

$$a^2 = b^2 + c^2 - 2bc \cdot \cos(y)$$
$$b^2 = a^2 + c^2 - 2ac \cdot \cos(\beta)$$
$$c^2 = a^2 + b^2 - 2ab \cdot \cos(\alpha)$$




```
from sympy import *
from math import radians,
degrees

#The sides
a = 5
b = 6
c = 7

#The angles
alpha = symbols('alpha')
beta = symbols ('beta')
y = symbols ('y')

eqA = Eq(c**2, a**2 + b**2
-2*a*b*cos(alpha))
eqB = Eq(b**2, a**2 + c**2 -
2*a*c*cos(beta))
eqY = Eq(a**2, b**2 + c**2 -
2*b*c*cos(y))

valueA = solve(eqA)
valueB = solve(eqB)
valueY = solve(eqY)

# We know there are going to be
2**3 results, so we're going to
use
# control loops to organize
them all and pass the result to
degrees.
```

```
for x in valueA:
   print("A:", degrees(x))
#A: 281.5369590328155
#A: 78.46304096718453
print("\n")
for x in valueB:
   print("B:", degrees(x))
#B: 302.8783495643775
#B: 57.12165043562251
print("\n")
for x in valueY:
   print("Y:", degrees(x))
#Y: 315.58469140280704
#Y: 44.415308597192976
```



Formulas

We can use the same methodology to check the trigonometric ratios that exist

```
from sympy import *
from math import radians, degrees

x = symbols('x')
y = symbols('y')

# trigsimp() Simplify the ratio trigonometric
#  expand_trig()  Expands  the trigonometric ratio

#Example:
print(trigsimp(sin(x)*cos(y)  + sin(y)*cos(x))) # sin(x + y)
print(expand_trig(sin( x + y)))
#sin(x)*cos(y) + sin(y)*cos(x)

print("\n")

#Angle ratios sum
print(expand_trig(sin(x  +  y)))
# sin(x)*cos(y) + sin(y)*cos(x)
print(expand_trig(cos(x  +  y)))
#  -sin(x)*sin(y)  +  cos(x)*cos(y)
print(expand_trig(tan(x  +  y)))
#           (tan(x)           + tan(y))/(-tan(x)*tan(y) + 1)

print("\n")

# Difference angle
print(expand_trig(sin(x  -  y)))
# sin(x)*cos(y) - sin(y)*cos(x)
print(expand_trig(cos(x  -  y)))
#      if      n(x)*sin(y)      + cos(x)*cos(y)
print(expand_trig(tan(x  -  y)))
#  (tan(x)  -  tan(y))/(tan(x)*tan(y) + 1)

print("\n")

#Double Angle Ratios
print(expand_trig(sin(2*x)))   # 2*sin(x)*cos(x)
print(expand_trig(cos(2*x)))   # 2*sin(x)*cos(x)
print(expand_trig(tan(2*x)))   # 2*tan(x)/(1 - tan(x)**2)
print("\n")
```

Geometry

Numbers complex

We can calculate complex numbers and separate the real parts from the imaginary ones using the *cmath* which allows us to do calculations with complex numbers:

```
import cmath

complex = cmath.sqrt(-24 - 70j)
# Complex number

#Binomial form
print(complex) # (5-7j)

#Polar form
print(cmath.polar(complex)) #
(8.602325267042627,
-0.9505468408120752)
```

This library allows us to do calculations with complex numbers easily.

Vectors

Due to the complexity of sympy, its vectors module is intended for more complex vectors that use arrays since they are 3-dimensional, but since we still don't know how to use and handle such vectors, we can recreate the behavior of vectors using a collection of two variables (x,y) with the *numpy* that allows us to do more advanced calculations:

```
from numpy import *

vectorA = array([10,9])
vectorB = array([2,3])

#Add and subtract of vectors
print(vectorA+vectorB) #[12 12]
print(vectorA-vectorB) #[8 6]

#Dot product (10*2 + 9*3)
print(vectorA.dot(vectorB)) #47

Comb1 = 3 #Base 1
Comb2 = 5 # Base 2

Combination
print (vectorA*Comb1 +
vectorB*Comb2)

modA = linalg.norm(vectorA)
modB = linalg.norm(vectorB)
#Modules

of vectors
print('A:',modA,'B:', modB) #A:
13.45362404707371 B:
3.605551275463989

solution =
arccos((vectorA.dot(vectorB))/(
modA*modB)) #Angle

between two vectors
print(degrees(solution))
```

Relation to mathematical

functions Python functions are are related to mathematical functions since both use variables to find out the possible values that function gives, but obviously they are not intended to be the same thing, since mathematical functions are used purely to find out the value of an expression with an unknown

variable, while that the function within the world of programming can be that, or it can be used to indicate what actions the computer should perform, or a mixture of both.

For example:

```
def f(x):
   result = 2 * x - 4
   print(result)

f(2) # 2*2-4 = 0
f(3) # 2*3-4 = 2
f(4) # 2*4-4 = 4
```

```
def g(x):
   result = 2 / x

f(0) # Would give zero division
error
```

Where ***dom(g)*** = $\mathbb{R}$ ***- {0}***
And ***Img(g) = (-inf, +inf)***

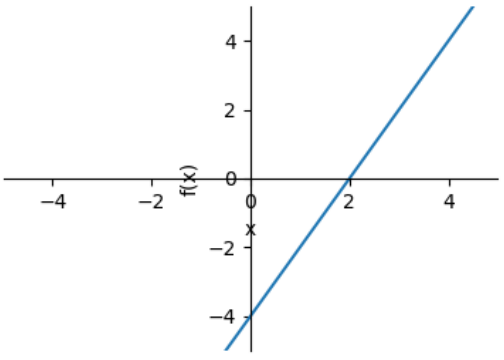
To present these results we go to use *sympy*:

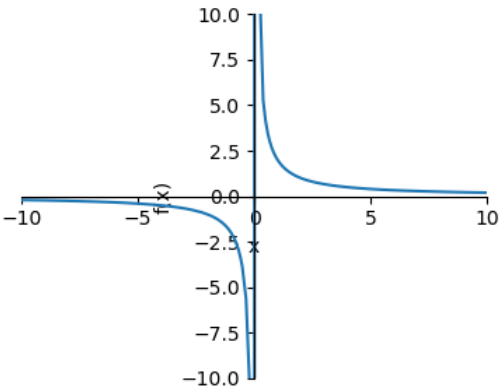
```
from sympy import *
from sympy.calculus.util import
continuous_domain

x = Symbol("x")

f = 2*x - 4 # Function
declaration

print(f.subs(x,2)) # 0
print(f.subs(x,3)) # 2
print(f.subs(x,4)) # 4

print(continuous_domain(f,
x,S.Reals)) #Reals (Domain is
all real numbers)
```

```
g = 2/x

print(continuous_domain(g,x,S.R
eals))
#Union(Interval.open(-oo, 0),
Interval. open(0, oo))

print(g.subs(x,0)) #Returns zoo
which involves division by
zero. Not for the show.
```

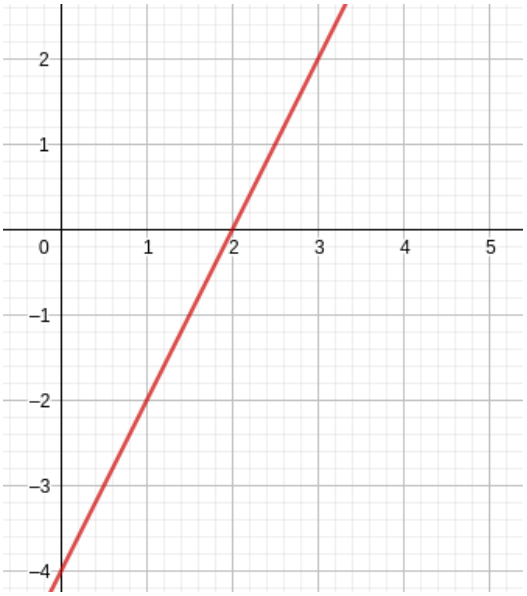
We can represent the functions using the *plot()* inside *sympy*, in that case we would have to give it limits on the x and y axes so that the function can be seen clearly:

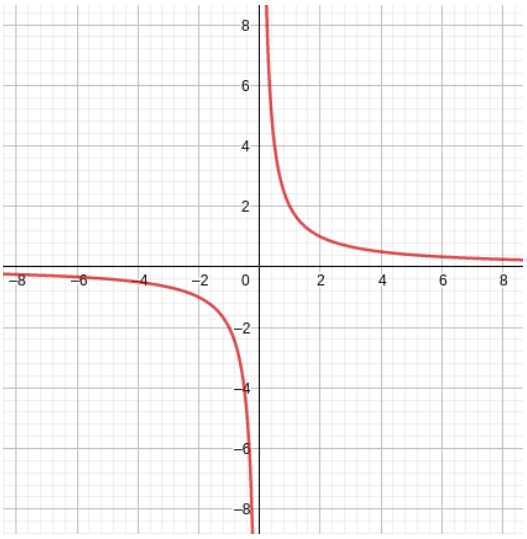
```
#We give them limits on the x
and y axes so that the graph
looks better .
p1 =
plot(f,xlim=[-5,5],ylim=[-5,5])
p2 =
plot(g,xlim=[-10,10],ylim=[-10,
10])
```





while we could also represent them using GeoGebra, as we have done before.





LeoCAD and model engineering

To create the 3D models of the various robots that we create, we are going to use a CAD (Computer-Aided Design) program called LeoCAD, which allows us to create 3D models of Lego using the models of the pieces

(LeoCAD Logo)

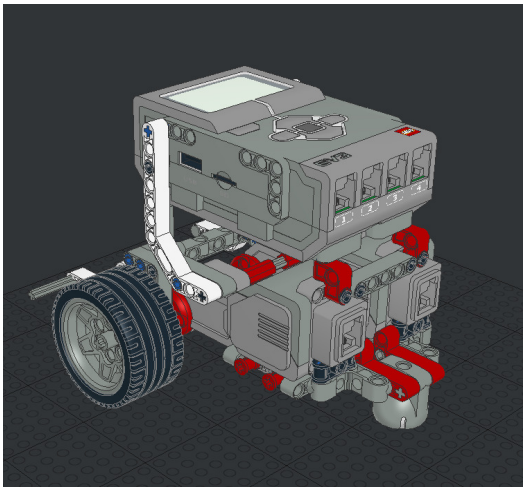
3D design of the educator

In order to create the basic model of the educator robot, the procedure that we are going to follow is to follow the instructions that we have been given in the kit or that we can also find on the internet on the official Lego website.

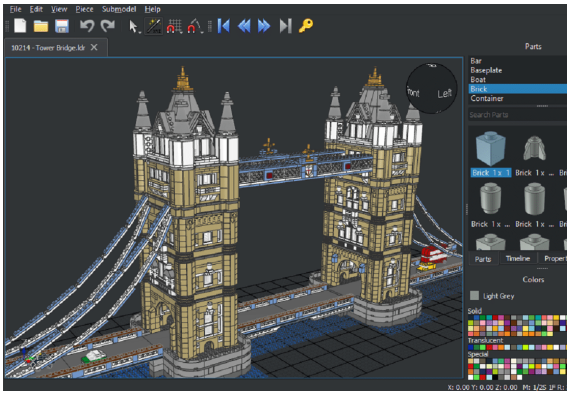
Most of the pieces can be found in the "Technic" section and the "Vehicles" section, but even so there are going to be pieces that we are not going to find easily, for those cases we can use the official Lego page to

search for it by name or ID https://education.lego.com/en-us/product-resources/mindstorms-ev3/downloads/whats-in-the-box

This is what the output of the educator model in LeoCAD would look like:



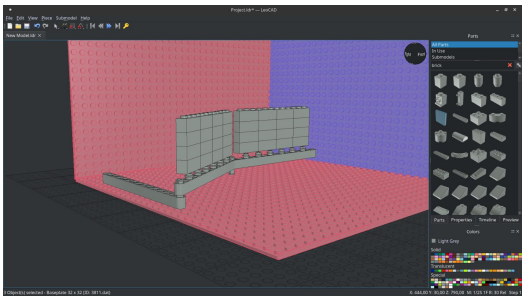
The downside of using LeoCAD it is having to rebuild the robot exactly as in the instructions, since the program does not present symmetry tools, since as the author says, it can result in parts that do not exist.

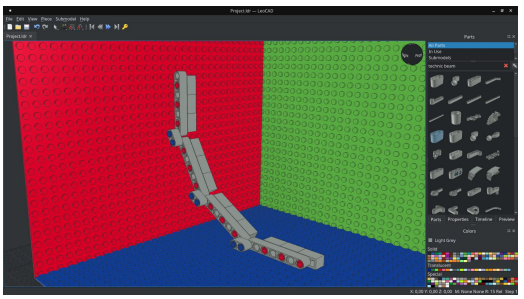


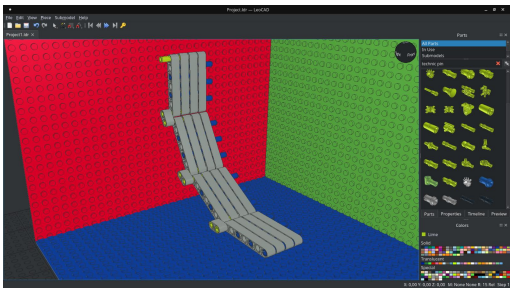
3D design of solutions for the SUMO robot.

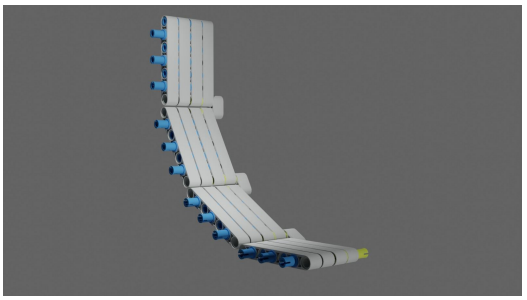
At the beginning of the project we began to think of solutions for a supposed SUMO robot to participate in robo-champions, but since there was no news of its realization, we changed to a slightly more theoretical project, but much more enriching. Here we can see one of the solutions designed for this robot, which would be a shovel with an ultrasonic sensor, which would detect and pick up opponents upon noticing their presence.

Here we have different concepts of the shovel, to finish arriving at the final concept. Screenshots are also included to see how LeoCad's interface is.





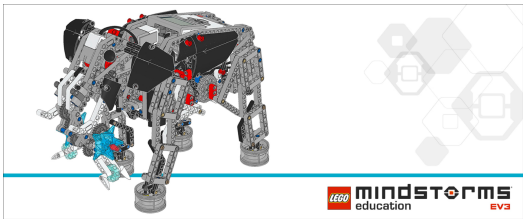




3D render of the shovel.

Realization of a global project with what has been learned.

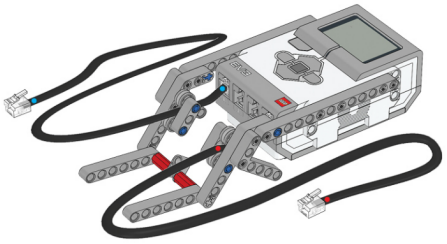
ev3 model expansion set elephant:
To complete the project and not be as theoretical as it already is, we decided to assemble a robot included in the kit: The ev3 model expansion set elephant.



In this section of the project we will deal with the construction with the different sections of the robot and its operation, as well as commenting on the code used in it.

Construction and external operation:

The robot construction process has a total of 251 steps. In this section we will not talk about the steps or the construction section, but we will focus on the reason for the pieces and their movement as well as understanding how it works.

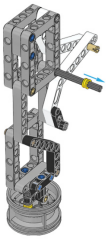


As we can see in the previous image, the robot uses 2 sensors, one color and one contact. Later in the code we will explain its function.

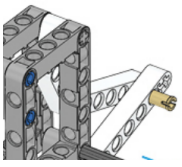
The robot also uses two motors, although the same motor is used for different functions.

After the presentation of the robot's electronic accessories, we will analyze its construction.

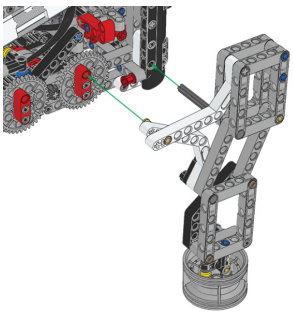
We have divided the construction into three different parts: The body, the trunk and the legs. The first thing we finish building are the legs. The robot has two equal legs two by two, this means that the front legs will not be the same as the rear ones.



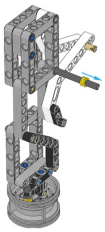
Here we have a 3D design of one of the front legs. This leg will function as a car suspension, and even if it is placed horizontally and the wheel that supports it removed it could function as such. Its real function is that it will be connected to the engine through the beige projection.



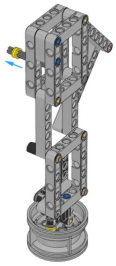
Here we can also see how it connects to the robot.



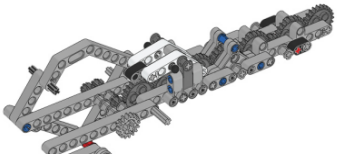
and will move the third joint forward.



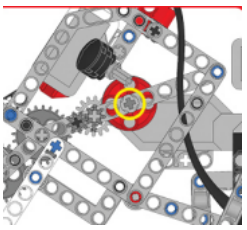
This will make the robot move forward. The rear legs work in the same way but they are connected to the second gear of the motor that we will now see.



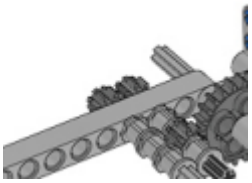
The second part we finished of the robot was the trunk. From our point of view, the trunk is the easiest part to make mistakes since it is a series of more than 14 gears of different sizes to perform different movements that will be explained below.



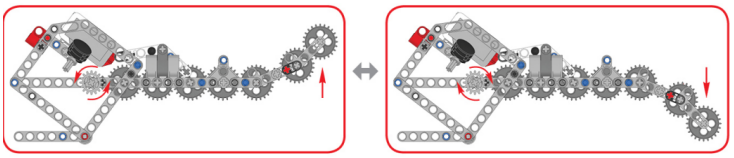
Here we have the complete trunk and we will gradually pay attention to the details.



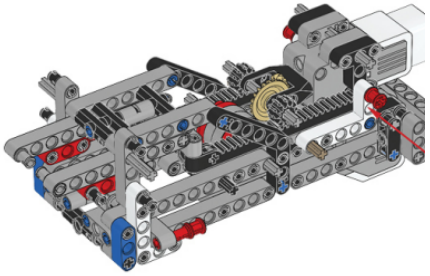
Here we can see the connection of the last component of the series with the motor. Really if we look closely the only thing that moves the motor is the gray gear. The Gray gear will make these two small pieces move as seen in the diagram above. These two pieces will in turn move the black bar to which it is connected to the last gear.

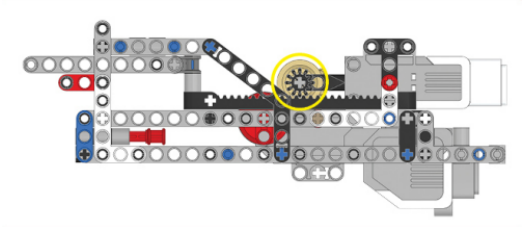


This will cause the chain to continue through the remaining 10 links which will cause the horn depending on the direction of motor rotation to move the horn up or down as the first nut rotates clockwise or counterclockwise. the same.



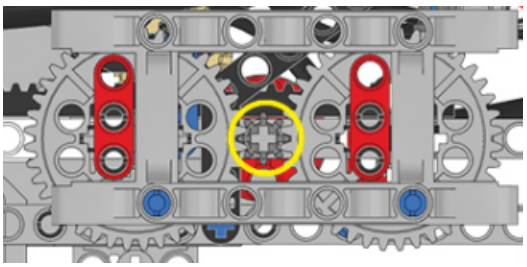
After having finished the trunk we connect it to what we call "the neck" although it is really part of the body.





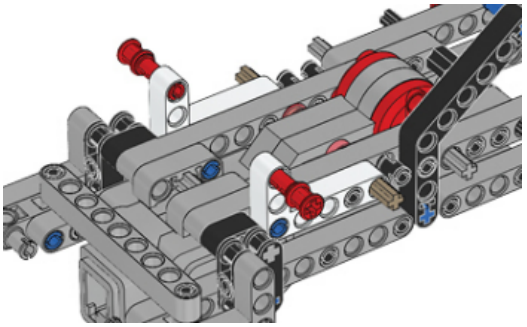
To explain this part of the robot we will use the isometric projection and a side view that we can see above this text. In these images we already have the motor connected. If we look closely the motor is connected to a gear that has a rod inserted that is connected to two other gears. When the Beige gear moves it moves the rod which in turn moves the two black gears. These move two horizontal pieces that will move the head when connected.

To finish the last part that we complete is the body, although really what it has incorporated is the EV3 Smart Block and a motor in which it makes the legs work, the part in which we will focus now.

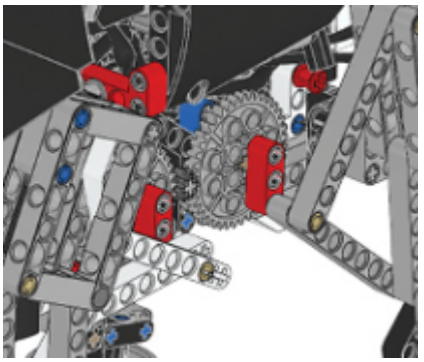


To build this part of the project we used a "beam" to give consistency to the body while we were finishing assembling it. We must remember to remove this part when we finish this part.

Moving on to the parts used for the mechanism of the legs, we have a motor, as seen in the photo below, and three gears, 2 with an upper diameter and another with a lower diameter. The motor is connected to a gray rod, which in turn is connected to one of the top spoke gears. This gray gear transmits the movement to the gear transmits the movement to the one with the lower radius, which will in turn move the last gear.



Finally, when the "beams" are removed, the body remains as we can see in the image below, leaving two red structures that connect with the "arms" of the legs.



Here we have a closer video of how this mechanism works:
Movement of the legs mechanism

Sensors:

To complete this section of the construction and engineering of the model, we will talk about the two sensors that the robot has, although we will see it more in depth in the code comment.

The robot has a contact sensor, used to check that the trunk does not exceed its limit and break, since when the sensor notices that it is being pressed, it leaves the trunk as it is and goes on to the next action. Here we have a video of how it works:
Contact sensor.

The last sensor is a color sensor, which is used to know which is the top of the neck, in addition to this function the sensor when it detects the red color makes an elephant sound. Here we have a video of how it works.
color sensing.

Code comment used for the elephant.

The code that will be commented below can be found on the institute's github, in the section dedicated to the project.
https://github.com/IES-Severo-Ochoa-Tecnologia/Mindstormsrobocampeones/blob/main/Elefante/elephant.py.

Here we have the complete code, which we break down little by little to explain it.

```
from pybricks.hubs import EV3Brick
from pybricks.ev3devices import Motor, ColorSenso
from pybricks.parameters import Port, Direction,
from pybricks.tools import wait, StopWatch
from pybricks.media.ev3dev import SoundFile
```

As in any project, the first step that must be done is to import the libraries, as we have previously explained what each one was about, we are not going to go into depth again, only that we have imported from them.

From programmable hubs we have imported the EV3 brick to make the buttons work, which will be essential in this code. From the Ev3 devices library we are going to import the three things that we are going to need, the motor, and the color and contact sensors. The parameters library will give us the knowledge that it is connected to each port, which is each address to which you can move the elephant and finally what buttons does the EV3 Smart Brick have. Of all of them we will use two constants that will be essential in our project, the "wait" and the "chronometer" that will serve to leave a time between each action. The last library is the middle one that will make the robot make an elephant sound.

```
ev3 = EV3Brick()
legs_motor = Motor(Port.A, Direction.COUNTERCLOCKWISE)
trunk_motor = Motor(Port.B, Direction.COUNTERCLOCKWISE)
neck_motor = Motor(Port.D)
touch_sensor = TouchSensor(Port.S1)
color_sensor = ColorSensor(Port.S4)
timer = StopWatch()
```

The code continues with the initialization of the variables. This step is preparatory for everything that remains in the program. The first variable we come across is the smart block initialization which we will save as “ev3”. From this moment we will begin to name variables with the devices of the robot. The first variable we find in legs_motor which will hold a motor that is located in port A that will rotate clockwise. The second variable is the one that contains the horn, which is a motor located in port B and will also rotate clockwise. Then we

have the neck motor, which is located in port D.
Finally, the two variables of the sensors, contact and color, are called, which are located in port 1 and 4 respectively. Finally we use the previously mentioned "parameters" library in which we insert a stopwatch in a variable called timer.

```
def reset():
    neck_motor.run(750)
    while color_sensor.color() != Color.RED:
        wait(10)
    neck_motor.brake()

    trunk_motor.run(600)
    while not touch_sensor.pressed():
        wait(10)
    trunk_motor.brake()


    ev3.speaker.play_file(SoundFile.ELEPHANT_CALL)

    neck_motor.run_angle(-600, 700, wait=False)
    trunk_motor.run_angle(-900, 750)
    wait(0.2)

    neck_motor.reset_angle(0)
    trunk_motor.reset_angle(0)
```

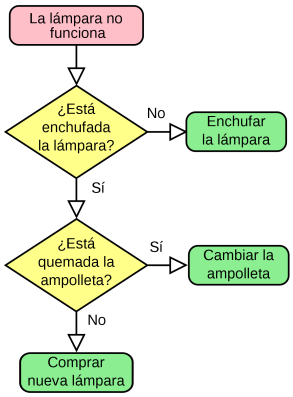
To conclude the variable and function definition part, we will find our first function that will be called "reset()". We will break it down little by little:
The first thing the robot will do is move its neck until the color sensor detects red, as we have already seen in the previous section. When this happens the robot waits 10 seconds and the motor will stop with the brake() function.
Finished with the neck motor, the trunk motor will do the same, but using the contact sensor, but on the contrary since we see that "while not pressed".
Finally when this has happened the middle library will play a rather peculiar elephant sound. Then the two motors will return to their original angles.
We will use this function throughout the code.

```
def grab():

    reset()

    trunk_motor.run_angle(1000, 300, wait=False)
    neck_motor.run_angle(1500, 350)
    neck_motor.run_angle(-750, 350)
    neck_motor.run_time(-150, 1000, wait=False)
    trunk_motor.run_angle(-700, 500)
    trunk_motor.run_angle(-300, 300, wait=False)
    neck_motor.run_angle(450, 400)
```

The second function of our code is this, which is used to catch objects with the trunk. This function is not implemented in the robot demo.
As we can see, it first uses the function explained above and then uses the angles to be able to pick up objects.

```
while True:
    timer.reset()
    steps = 0
    while not any(ev3.buttons.pressed()):
        wait(10)

    while timer.time() < 1000:
        if Button.UP in ev3.buttons.pressed():
            steps += 1
            timer.reset()
            ev3.speaker.beep(600)

            while Button.UP in ev3.buttons.pressed():
                wait(10)
        if Button.DOWN in ev3.buttons.pressed():
            steps -= 1
            timer.reset()
            ev3.speaker.beep(1200)
            while Button.DOWN in ev3.buttons.pressed():
                wait(10)
        if Button.LEFT in ev3.buttons.pressed():
            trunk_motor.run(300)
            while not touch_sensor.pressed():
                wait(10)
            trunk_motor.run_angle(-100, 30)
            reset()
        if Button.RIGHT in ev3.buttons.pressed():
            grab()
        if Button.CENTER in ev3.buttons.pressed():
            ev3.speaker.play_file(SoundFile.ELEPHANT_CALL)
    if steps != 0:
        angle = 900 * steps
        legs_motor.run_angle(1000, angle)
```

Finally we move on to the last step of the code, which may seem very messy or complex, but if we see it as a flowchart it is much simpler. This is an example of a flowchart:



Once we are focused on the code, we see that the first thing we have done is put a “while true” which means that while it is true, this is very useful to start the programs to put all the code together, It should be emphasized that it will always be true.

Later we will start the aforementioned stopwatch and call a new variable called “steps” which will be very useful later.

The first thing we do is follow the flowchart, if nothing happens we wait 10 milliseconds. When the 10 milliseconds have passed, it will ask again if a button has been pressed.

If less than 1000 milliseconds have passed and the robot has not received any response, the program is wrong. But if less than 1000 milliseconds have passed the program will start working.

If (“if” in the code) the upper button is pressed, the robot will take a step and make a beep, which is like a light beep. The while used is so that while the while is pressed, no other button can be pressed and that the program does two actions at the same time and the robot "is lying". If the lower button is pressed, it will do the same process as the previous one but the step will be backwards instead of forwards. The next “if” is a little more complex since we say that if the left button is pressed, the horn motor will start and while the contact sensor is not pressed, it will continue to work. When we press, we will use the previously defined function “reset”. Finally we have the right click control loop which makes grab work. Finally we have the middle button loop which when pressed makes an elephant call. Finally we have a piece of code that makes the elephant reposition itself.

Using the “Lego Mindstorms Controller” app

Finally, for people who are not very fond of programming, there is an application called Lego Mindstorms Controller that makes you control motors without having to program codes or resolve bugs.

Conclusion:

To conclude, python has seemed to us to be the most suitable language for this work, although it is still the simplest we can find, although less so than programming with blocks, which we reject due to its linearity.

Programming with blocks can be a good idea, from our point of view, for people from the first year of ESO to the third year of this same cycle. Programming in code can be interesting from the third or fourth year of compulsory secondary education.

Turning to the functionality of python in the area of mathematics, we have been pleasantly surprised, given its wide

variety of libraries and their quality, which from our point of view is one of the strong points of this language, in addition to those mentioned above.
The Lego mindstorms Education Kit is very complete, with many necessary sensors and motors. From our point of view we should not focus so much on the kit which is quite simple, but more on the software and the infinite possibilities with it.

Personal assessment:

From our point of view, this project is very enriching and useful, but also Too big for just two people. For the future, having the kit already in the institute, it would be a very good idea to elaborate it again with a vision of robo-champions, since there is already a very complete theoretical base.
On the subject of the kit, as we have already mentioned, it has many possibilities if it is used correctly, but even if it has all these facilities, the price of this kit seems very high, not only the kit itself, but also the individual parts. too high a cost.

Acknowledgments:

We ask a big thank you to our mentors in this research project María Teresa Porto and Yago Martínez de Fuenmayor and we would also like to mention our Mathematics teachers Paloma Marcos Prado and María Torres Botia.

We appreciate the loan that the institute has made to us of the necessary material to complete this project, as would be the case with the Lego Mindstorms kit and a laptop to be able to program the robot.

We also ask thanks to our teacher Fernando for helping us and guiding us in this project.

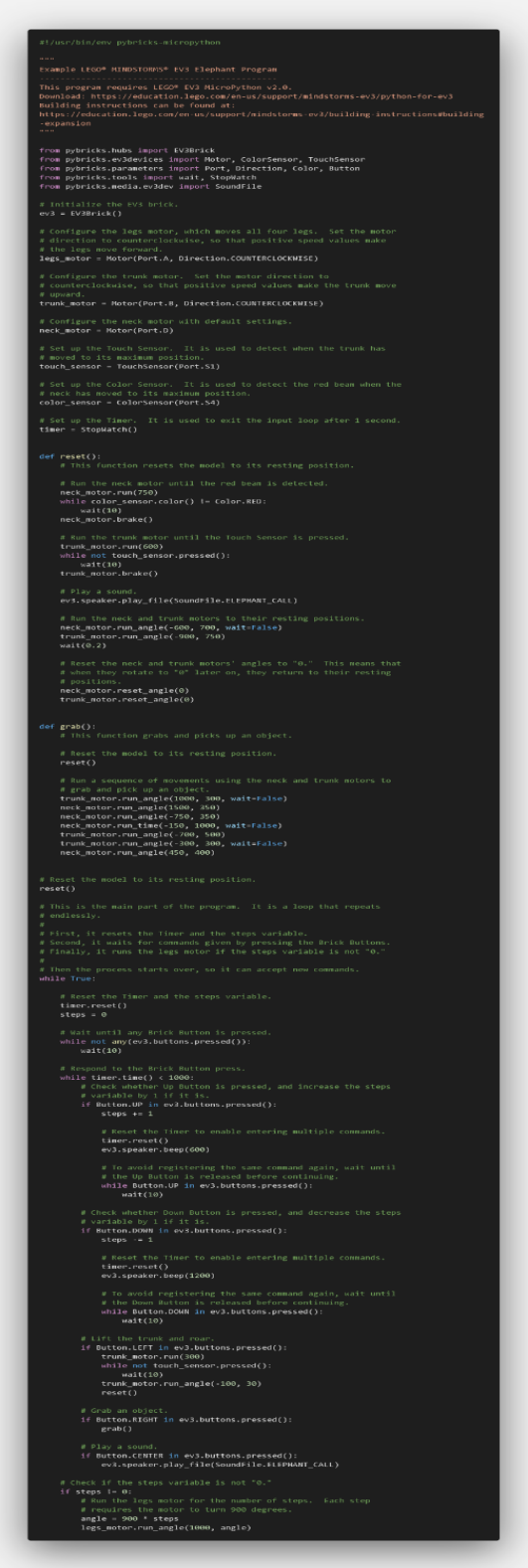
Bibliography:

Python libraries and third parties with their documentation:

https://www.sympy.org/en/index.html
https://matplotlib.org/
https://numpy.org/doc/stable/
https:// docs.python.org/3/
https://pybricks.com/
https://pybricks.com/ev3-micropython/

Annex:

Complete code of the elephant robot.



```
#!/usr/bin/env pybricks-micropython

"""
Example LEGO® MINDSTORMS® EV3 Elephant Program
----------------------------------------------

This program requires LEGO® EV3 MicroPython v2.0.
Download: https://education.lego.com/en-us/support/mindstorms-ev3/python-for-ev3
Building instructions can be found at:
https://education.lego.com/en-us/support/mindstorms-ev3/building-instructions#building
-expansion
"""

from pybricks.hubs import EV3Brick
from pybricks.ev3devices import Motor, ColorSensor, TouchSensor
from pybricks.parameters import Port, Direction, Color, Button
from pybricks.tools import wait, StopWatch
from pybricks.media.ev3dev import SoundFile

# Initialize the EV3 brick.
ev3 = EV3Brick()

# Configure the legs motor, which moves all four legs.  Set the motor
# direction to counterclockwise, so that positive speed values make
# the legs move forward.
legs_motor = Motor(Port.A, Direction.COUNTERCLOCKWISE)

# Configure the trunk motor.  Set the motor direction to
# counterclockwise, so that positive speed values make the trunk move
# upward.
trunk_motor = Motor(Port.B, Direction.COUNTERCLOCKWISE)

# Configure the neck motor with default settings.
neck_motor = Motor(Port.D)

# Set up the Touch Sensor.  It is used to detect when the trunk has
# moved to its maximum position.
touch_sensor = TouchSensor(Port.S1)

# Set up the Color Sensor.  It is used to detect the red beam when the
# neck has moved to its maximum position.
color_sensor = ColorSensor(Port.S4)

# Set up the Timer.  It is used to exit the input loop after 1 second.
timer = StopWatch()


def reset():
    # This function resets the model to its resting position.

    # Run the neck motor until the red beam is detected.
    neck_motor.run(750)
    while color_sensor.color() != Color.RED:
        wait(10)
    neck_motor.brake()

    # Run the trunk motor until the touch sensor is pressed.
    trunk_motor.run(600)
    while not touch_sensor.pressed():
        wait(10)
    trunk_motor.brake()

    # Play a sound.
    ev3.speaker.play_file(SoundFile.ELEPHANT_CALL)

    # Run the neck and trunk motors to their resting positions.
    neck_motor.run_angle(-600, 700, wait=False)
    trunk_motor.run_angle(-900, 750)
    wait(0.2)

    # Reset the neck and trunk motors' angles to "0."  This means that
    # when they rotate to "0" later on, they return to their resting
    # positions.
    neck_motor.reset_angle(0)
    trunk_motor.reset_angle(0)


def grab():
    # This function grabs and picks up an object.

    # Reset the model to its resting position.
    reset()

    # Run a sequence of movements using the neck and trunk motors to
    # grab and pick up an object.
    trunk_motor.run_angle(1000, 300, wait=False)
    neck_motor.run_angle(1500, 450)
    neck_motor.run_angle(-750, 350)
    neck_motor.run_time(-150, 1000, wait=False)
    trunk_motor.run_angle(-700, 500)
    trunk_motor.run_angle(-300, 300, wait=False)
    neck_motor.run_angle(450, 400)


# Reset the model to its resting position.
reset()

# This is the main part of the program.  It is a loop that repeats
# endlessly.
#
# First, it resets the timer and the steps variable.
# Second, it waits for commands given by pressing the Brick Buttons.
# Finally, it runs the legs motor if the steps variable is not "0."
#
# Then the process starts over, so it can accept new commands.
while True:

    # Reset the timer and the steps variable.
    timer.reset()
    steps = 0

    # Wait until any Brick Button is pressed.
    while not any(ev3.buttons.pressed()):
        wait(10)

    # Respond to the Brick Button press.
    while timer.time() < 1000:
        # Check whether Up Button is pressed, and increase the steps
        # variable by 1 if it is.
        if Button.UP in ev3.buttons.pressed():
            steps += 1

            # Reset the timer to enable entering multiple commands.
            timer.reset()
            ev3.speaker.beep(600)

            # To avoid registering the same command again, wait until
            # the Up Button is released before continuing.
            while Button.UP in ev3.buttons.pressed():
                wait(10)

        # Check whether Down Button is pressed, and decrease the steps
        # variable by 1 if it is.
        if Button.DOWN in ev3.buttons.pressed():
            steps -= 1

            # Reset the timer to enable entering multiple commands.
            timer.reset()
            ev3.speaker.beep(1200)

            # To avoid registering the same command again, wait until
            # the Down Button is released before continuing.
            while Button.DOWN in ev3.buttons.pressed():
                wait(10)

        # Lift the trunk and reset.
        if Button.LEFT in ev3.buttons.pressed():
            trunk_motor.run(300)
            while not touch_sensor.pressed():
                wait(10)
            trunk_motor.run_angle(-100, 30)
            reset()

        # Grab an object.
        if Button.RIGHT in ev3.buttons.pressed():
            grab()

        # Play a sound.
        if Button.CENTER in ev3.buttons.pressed():
            ev3.speaker.play_file(SoundFile.ELEPHANT_CALL)

    # Check if the steps variable is not "0."
    if steps != 0:
        # Run the legs motor for the number of steps.  Each step
        # requires the motor to turn 900 degrees.
        angle = 900 * steps
        legs_motor.run_angle(1000, angle)
```

Math codes:

Equations:

```
from sympy import *
```

```
x = symbols('x') we want to
find

equation = Eq((2*x - 4),0)
# 2*x - 4 = 0

print(solve(equation)) #
[2]
```

Functions:

```
    from sympy import *
from sympy.calculus.util import
continuous_domain

x = Symbol("x")

= 2*x - 4 # Declarationf from
the function(

print(f.subsx,2)) # 0
print(f.subs(x,3)) # 2
print(f.subs(x,4)) # 4

print(continuous_domain(f,x,
S.Reals)) #Reals (Domain is all
realnumbers)

g = 2/x

print(continuous_domain(g,x,S.R
eals))
#Union(Interval.open(-oo, 0),
Interval.open(0, oo))

print(g.subs(x,0)) #Returns zoo
which implies the division_
from zero. Not for the show.

#We give them limits on axes
and so that the graphiclook
better .
p1 =
plot(f,xlim=[-5.5],ylim=[-5.5])
p2 =
plot(g,xlim=[-10.10],ylim=[-10.
10])
```

Inequalities:

```
from sympy import *

x = symbols('x')

of inequalities
a = x**2 + 6*x < 0
b = x**2 + 6*x > 0
c = x**2 + 6*x <= 0
d = x**2 + 6*x >= 0


print(solveset(a,x,S.Reals))
#Interval.open(-6, 0)
print(solveset(b,x,S.Reals))
#Union(Interval.open(-oo,
-6), Interval.open(0, oo))
print(solveset(c,x,S.Reals))
#Interval(-6, 0)
print(solveset(d,x,S.Reals))
#Union(Interval(-oo , -6),
Interval(0, oo))

print ('\n')

print(solve(a,x,S.Reals))
#(-6 < x) & (x < 0)
print(solve(b ,x,S.Reals))
#((-oo < x) & (x < -6)) |
((0 < x) & (x < oo))
print(solve(c,x,S.Reals))
#(-6 <= x) & (x <= 0)
print(solve(d,x,S .Reals))
#((0 <= x) & (x < oo)) |
((-oo < x) & (x <= -6))

print ('\n')

e = x - 2 > 0
f = x - 1 < 3

print(solve((e,f),x,S
.Reals)) # (2 < x) & (x < 4)
```

Logarithms:

```
import math
import sympy
n = 2 # Basis
p = 8 # Argument
# math.log(Argument, Base)
print(math.log(8,2)) # 3
x = sympy.symbols('x') we
want to find out
#The function we are going
to use
def Logarithm(Base ,
Argument):
    equation =
(sympy.Eq(Base**x,Argument))
#Declaration

print(sympy.solve(equation))
#Result
Logarithm(n,p) #3
```

Complex Numbers:

```
import

complex cmath =
cmath.sqrt(-24 - 70j) #
Complex number

print(complex) # (5-7j)
print(cmath.polar(complex))
# (8.602325267042627,
-0.9505468408120752)
```

Polynomial Factorization:

```
from sympy import *
x = symbols('x ')
polynomial = 2*x**2 - 5*x -
3 #Polynomial declaration
print(factor(polynomial))
#(x - 3)*(2*x + 1)
```

Polynomial Operations

```
from sympy import *

x = symbols ('x')

a = 2*x**2 - 5*x - 3
b = 4*x**2 + 6*x + 2

print(a+b) # 6*x**2 + x - 1
print(ab) # -2*x**2 - 11*x -
```

```
5

print
(Poly.mul(poly(a),poly(b)).a
s_expr()) # 8*x**4 - 8*x**3
- 38*x**2 - 28*x - 6
print(div (a,a)) # (1/2,
-8*x - 4)
```

```
from sympy import *
```