

StarCode Handbook

Version 1.2.9 · for StarCode v11.167.0 · 1 October 2026

Everything about StarCode, from the name of each panel on the screen to every block in the toolbox and the code it writes. Start at the beginning, or use the contents to jump to what you need.

Why the right word matters

StarCode writes real code, the kind grown-up programmers write. So we use the real words for everything too.

Every picture in here is a real screenshot of the app.

TABLE OF CONTENTS

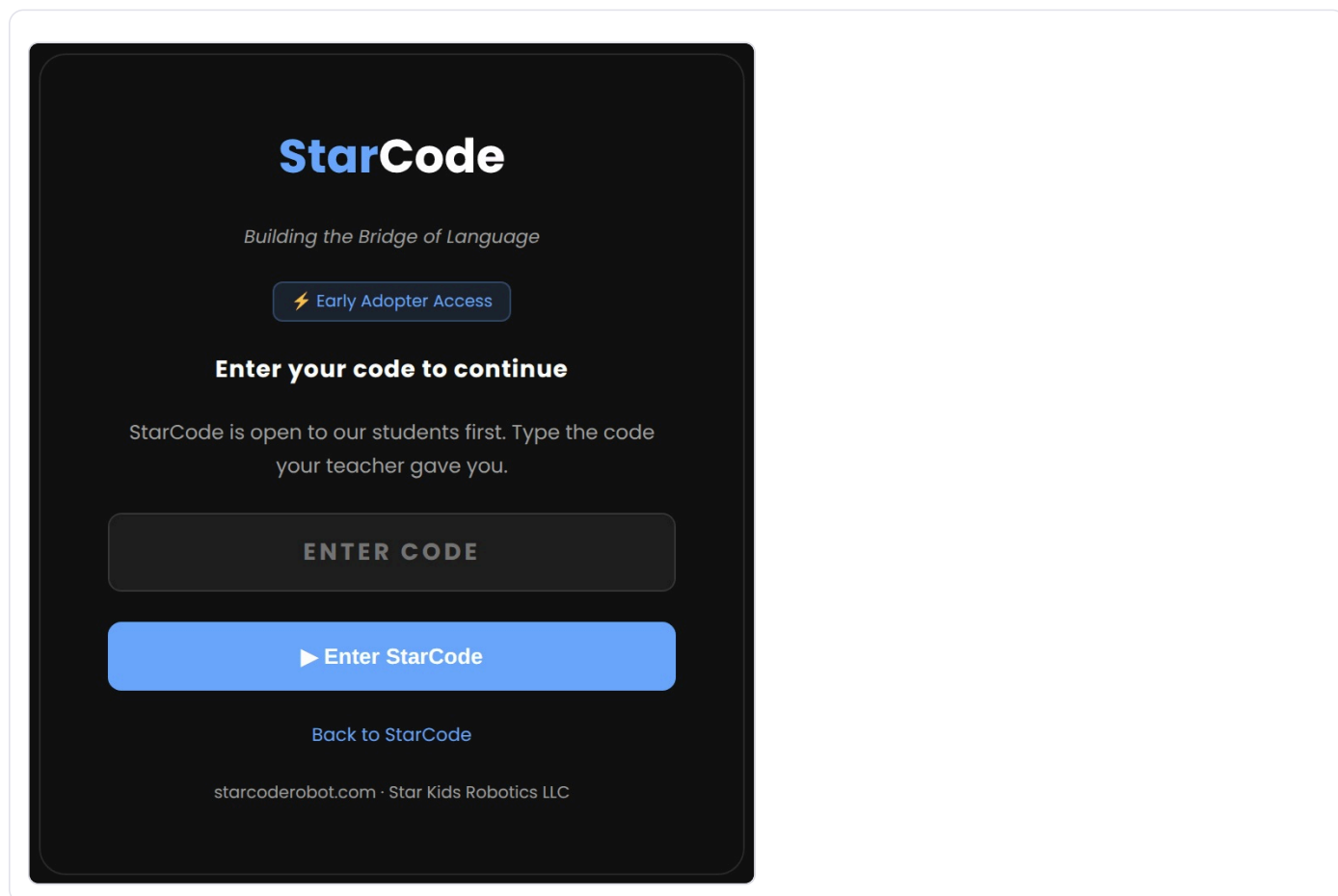
Part one	The two views, and the panels inside them	2
Part two	The categories	7
	1. Imports	8
	2. Events	12
	3. Motor	13
	4. MotorGroup	14
	5. Control	16
	6. Sensors	19
	7. Operators	21
	8. Variables	25
	9. Functions	27
	10. Sound	29
	11. Light	30
	12. Timing	31
	13. Remote	33
	14. Circuit	35
Part three	Ideas behind the code	39
Part four	Getting connected	47
Part five	What each robot can do	50
Part six	Words for the robot	52
Part seven	The Tutor	53
Part eight	Plans, codes and the portal	56
Part nine	Terms	59
Part ten	Every block, by robot and language	66

The two views, and the panels inside them

StarCode has two screens, and you swap between them with the button in the top left corner. Each screen is made of panels, and every panel has its own name. You can drag the bars between them to make one bigger. The word category is saved for the colored circles in the toolbox, and nothing else.

Getting in

Go to starcoderobot.com and press "Open StarCode". StarCode then asks for a code: "Enter your code to continue". The code for this launch is STARKID. A student with a class code from a teacher types that code instead. Either one opens the app. The code is remembered on that device for a year, so it is typed once.

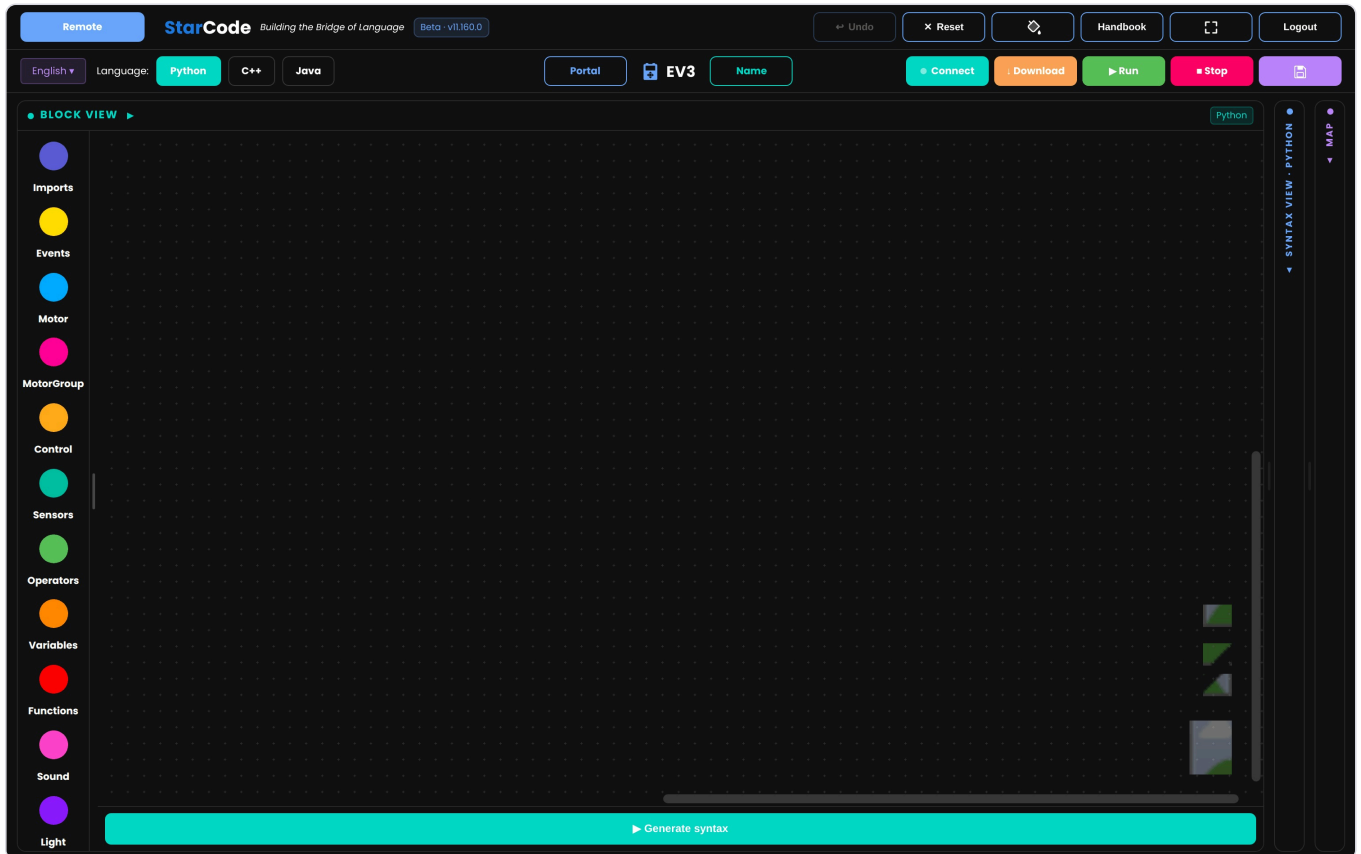


The language menu

The button that says "English ▼" opens the language menu, headed Input Language. It holds 22 languages: English, Spanish, French, German, Portuguese, Chinese, Japanese, Korean, Arabic, Hindi, Italian, Russian, Dutch, Turkish, Polish, Vietnamese, Bengali, Urdu, Indonesian, Marathi, Telugu and Thai. The whole app follows it: the categories, the titles, the buttons and every message. The blocks and the Syntax view do not change, because they are code, and code is written the same way in every language.

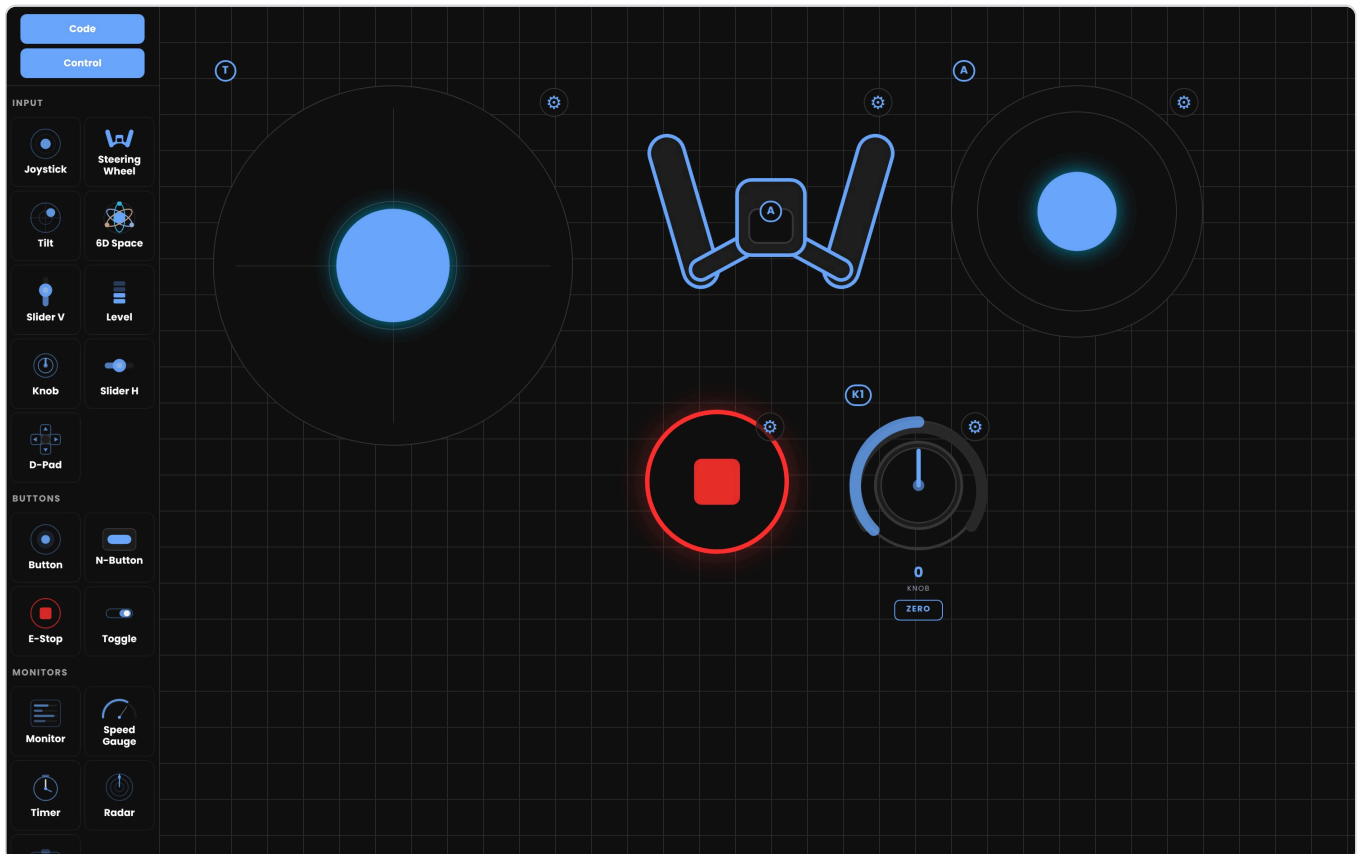
The coding view

Where you build a program. The toolbox and block canvas on the left, the real code in the middle with the monitor below it, and the StarCode Tutor on the right.



The control view

Where you build a controller for your robot. Drag sliders, buttons and joysticks onto the grid, and use them while your program runs.



The control view, widget by widget

A canvas of widgets a child lays out themselves, and drives the robot with while the program runs. The app sorts them into three groups: Input, Buttons and Monitors. Every widget comes in the same three sizes, and the Camera has two more. Every widget takes its color from the color picker, and the colors are the app's own: blue, teal, purple, orange, pink and white.

WIDGET	WHAT IT DOES
Input	
Joystick	Push it any way. It gives two numbers, one for across and one for up and down
Steering Wheel	Turn it left or right. It gives how far it is turned
Tilt	Reads the tablet's own tilt sensor and sends the angle to the robot, so tipping the tablet drives it. On an iPad it asks permission first, because Apple requires it
6D Space	One control that gives three numbers at once: X, Y and Z
Slider V	A slider that moves up and down
Slider H	A slider that moves across
Level	Fills from the bottom and stays where you put it
Knob	Turn it around to set a number
D-Pad	Four arrows. Each direction is fully on or fully off, never in between
Buttons	
Button	Pressed or not pressed
N-Button	A pill shaped button, the shape of the buttons on the front of a board. It carries a letter, so two of them are A and B
E-Stop	Emergency stop. One press stops the robot
Toggle	On or off, and it stays where you leave it
Monitors	
Monitor	Shows a value from the program
Speed Gauge	Shows a speed on a dial
Radar	A display and nothing more. It reads two numbers the program publishes: which way the sensor is pointing, and how far it can see. How the sensor gets turned is the program's business. It keeps one reading per degree until the sensor comes around again, so the picture is always the last full turn
Timer	Two faces driven by one clock: a ring outside and numbers inside. Count up runs from zero. Count down runs to zero from whatever the plus and minus buttons set, and the numbers turn pink at zero
Camera	Shows what the robot's camera sees, once a robot is connected
The letter	A, B, C or K1 sits above each widget, so a child can say out loud which one they mean

The top row



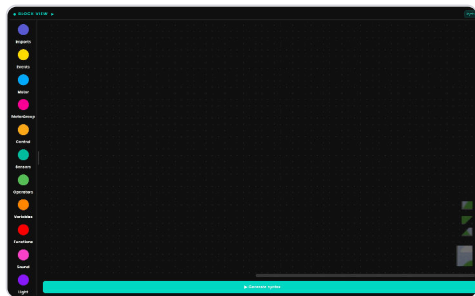
BUTTON	WHAT IT DOES
Handbook	Opens this handbook over the app
Full screen	Gives StarCode the whole screen. Made for a tablet
Install	Puts StarCode on the computer as its own app. It only appears where the browser offers it
Skins	The paint can. StarCode Dark or StarCode Light

The panels in the coding view

The control view has its own parts, and they come later in this handbook.

Block View

The whole left side of StarCode. The toolbox and the block canvas together.



Toolbox

The strip down the far left. It holds all the categories.

Not: library, palette, sidebar



Category

One colored circle in the toolbox. Click it and its blocks slide out.

Not: library



Block drawer

The blocks that slide out when you click a category. Drag one onto the canvas.

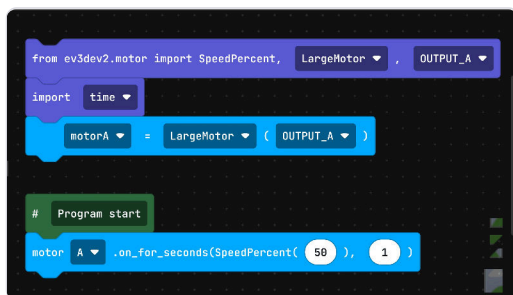
Not: menu, list



Block canvas

The big dotted space where you drag blocks and join them together.

Not: workspace, stage



Syntax view

The panel that shows your program as real written code, with line numbers.

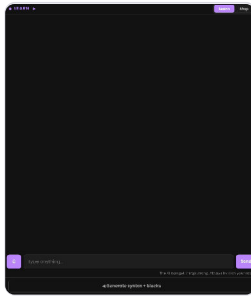
Not: code view, output



Learn view

Where you can ask the StarCode Tutor a question about your program.

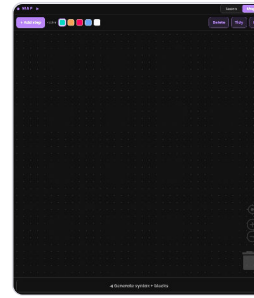
Not: AI panel, chat



Map

Where you lay out your steps before you build them, and join them with lines. Free for everybody.

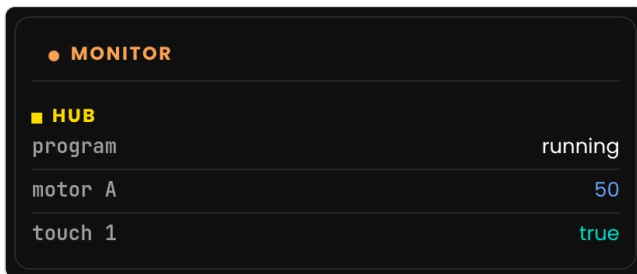
Not: flowchart, diagram



Monitor

Shows what is plugged into each port right now, and what the robot is measuring.

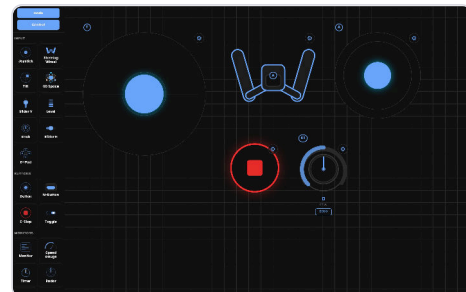
Not: console, dashboard



Remote page

The control view with its widgets. The button to swap to it is in the top left corner.

Not: control view



The categories

All fourteen, and their colors

Every category has its own color, and the blocks inside it are that color too. So a pink block on the canvas came from MotorGroup, wherever it ended up.

COLOR	CATEGORY	WHAT IS IN IT
 indigo	Imports	Importing your tools, and declaring your parts
 yellow	Events	Where a program starts
 blue	Motor	Turning one motor
 pink	MotorGroup	Making several motors act as one
 orange	Control	Loops, ifs, and waiting
 teal	Sensors	Reading color, distance and force
 green	Operators	Working out an answer, and comparing
 dark orange	Variables	Making and changing your own names
 red	Functions	Your own blocks, made from other blocks
 bright pink	Sound	Beeeps and notes
 purple	Light	The lights and the display
 light blue	Timing	Timers and waiting. Follows the skin, so it changes with the theme
 mint	Remote	Sliders and buttons you press yourself
 red and blue	Circuit	Pins on a board. Two colors because wiring has two sides

The categories, one by one

In the order they sit in the toolbox, top to bottom.

1. Imports · indigo

Everything that has to happen before your program can run. It is first because it comes first in the code. It holds two kinds of blocks, and they are two different jobs in the same category: the ones that import a tool, and the ones that declare a part.

KIND OF BLOCK	COLOR	WHAT IT DOES
import lines	purple	Tell the robot to load a tool from its own library into your program
declaring lines	multicolored	Give a part its own name, so every block after it can use that name

The import lines, purple

Every import line is purple. Every robot has its own library, so every one uses its own words. The shape never changes. Examples: `from ev3dev2.motor import LargeMotor`, `from pybricks.pupdevices import Motor`, `from machine import Pin`.

BLOCK	WHAT IT BRINGS IN
<code>from pybricks.pupdevices import ...</code>	Motors and sensors
<code>from pybricks.parameters import ...</code>	Port letters, colors and directions
<code>from pybricks.hubs import PrimeHub</code>	SPIKE Prime itself
<code>from pybricks.robotics import DriveBase</code>	Driving motors together
<code>from pybricks.tools import ...</code>	Waiting and timers

The declaring lines, multicolored

The declaring lines are multicolored: each one is the color of the category its part belongs to. Each one makes a name. After that line, every block below can use the name instead of saying the port again. Again the shape is the same on every robot and only the words change. Examples: `motorA = LargeMotor(OUTPUT_A)`, `motorA = Motor(Port.A)`, `tank = MoveTank(OUTPUT_A, OUTPUT_B)`.

BLOCK	WHAT IT NAMES
<code>hub = PrimeHub()</code>	The hub, so you can use its lights, screen and tilt
<code>motorA = Motor(Port.A)</code>	A motor in one port
<code>color_sensorA = ColorSensor(Port.A)</code>	A color sensor
<code>distance_sensorA = UltrasonicSensor(Port.A)</code>	A distance sensor
<code>force_sensorA = ForceSensor(Port.A)</code>	A force sensor
<code>drive_base = DriveBase(motorA, motorB, 56, 114)</code>	Two motors as one pair. The numbers are your wheel size and the gap between the wheels, in millimeters

The two groups in the drawer

Open the Imports drawer and you will see one clear gap. That gap is deliberate. Everything above it imports a tool, and it is all purple. Everything below it declares a part, and it is multicolored. Above them both sits a flat-topped block that starts the stack and writes nothing but a comment.

1 Importing

the import lines

One block per part. The block brings every word that part needs, including the word for its port, so a motor block already has OUTPUT_A in it. The robot does not know the word LargeMotor until you ask for it, and this line asks. Two parts wanting the same line only get it once.

```
# import

from ev3dev2.motor import SpeedPercent, LargeMotor, OUTPUT_A

from ev3dev2.motor import SpeedPercent, MoveTank, OUTPUT_A, OUTPUT_B

from ev3dev2.sensor.lego import TouchSensor
from ev3dev2.sensor import INPUT_1

from ev3dev2.sound import Sound

import time
```

2 Declaring

the declaring lines

Each of these makes a name, and you pick the name from the dropdown the same way you pick the port. Picking the name picks the kind of thing with it, so a gyro can never be declared as a touch sensor. After that line, every block below can use the name instead of saying the port again. Notice the colors: a declaring line is the color of the category the thing belongs to, so the line that declares a motor is Motor blue even though it lives here in Imports. Import lines are always purple, the color of Imports itself. The color tells you which of the two jobs a line is doing. The last two, the speaker and the lights, have no port in them, because they are built into the robot and there is only one of each.

```
motorA = LargeMotor ( OUTPUT_A )

tank = MoveTank ( OUTPUT_A, OUTPUT_B )

touch1 = TouchSensor ( INPUT_1 )

btn = Button()

spkr = Sound()

leds = Leds()
```

The rule for how many of each

One import block per part, one declaring line per actual thing. Two motors and a sensor is two import blocks and three declaring lines, because each part needs its own name.

One block is not always one line

Look at the touch sensor block: it writes two lines. A Python import line can only name one library, and the word for the sensor and the word for its port live in different libraries. So the block writes both, and you still only drag one thing.

The difference most people miss

The lines for things you plugged in have a port in them. The lines for the speaker and the lights do not. A motor could be in any of four ports, so you have to say which. There is only one speaker, so there is nothing to say. If you find yourself looking for a port on the speaker block, that is why it is not there.

Why the drive base needs two numbers

To drive exactly 20 centimeters, the robot has to know how big its wheels are and how far apart they sit. Measure yours and put those numbers in.

What a library is

A library is the collection of code an import brings a tool in from. It lives inside the robot's firmware, so you never see it on the screen. The colored circles in the toolbox are categories, not libraries.

Importing in all three languages

WHAT YOU MEAN	PYTHON	C++	JAVA
Bring in a tool	<code>from x import Motor</code>	<code>#include "ev3dev.h"</code>	<code>import lejos.hardware.motor.Motor;</code>

Why the category changes its name

Python and Java say `import`. C++ says `#include`. Same job, different word, so the category is called Imports in Python and Java and Include in C++.

Not every program needs an import line

Whether a program needs one depends on the language, not the category. In Python every Circuit block needs one. On the ESP32, `Pin(13, Pin.OUT)`, `Pin(13).value(1)`, `PWM`, `ADC` and `time_pulse_us` all need a line from the machine library, and on the micro:bit every pin block needs `from microbit import *`. In C++ the pin commands, `pinMode`, `digitalWrite`, `digitalRead`, `analogWrite`, `analogRead`, `tone` and `pulseIn`, are built into the language, so there is nothing to fetch and the Include stack stays empty. The one exception is the servo. It needs `#include <Servo.h>` on the Arduino or `#include <ESP32Servo.h>` on the ESP32, and a declaring line, `Servo servo13;`. So the same flashing light program is two import lines in Python and none in C++. That is the two languages being different, not StarCode choosing.

The Imports button

On the top row, to the right of the Java button, sits a solid purple button with white writing and a small down arrow. It says Imports in Python and Java and Include in C++. It is there while the program on the canvas is missing a line it needs, and it is gone when the program is complete. It does not matter how the lines got there: one press of the button, dragging the blocks out of the Imports category one at a time, or some of each. The button watches the program. It does not remember whether it was pressed.



One press puts every missing line in, as real blocks in the Imports stack, with their own dropdowns. A student can open them, change them, or throw them away. The Imports stack goes in above everything else on the canvas. It never writes a line the Imports category does not offer.

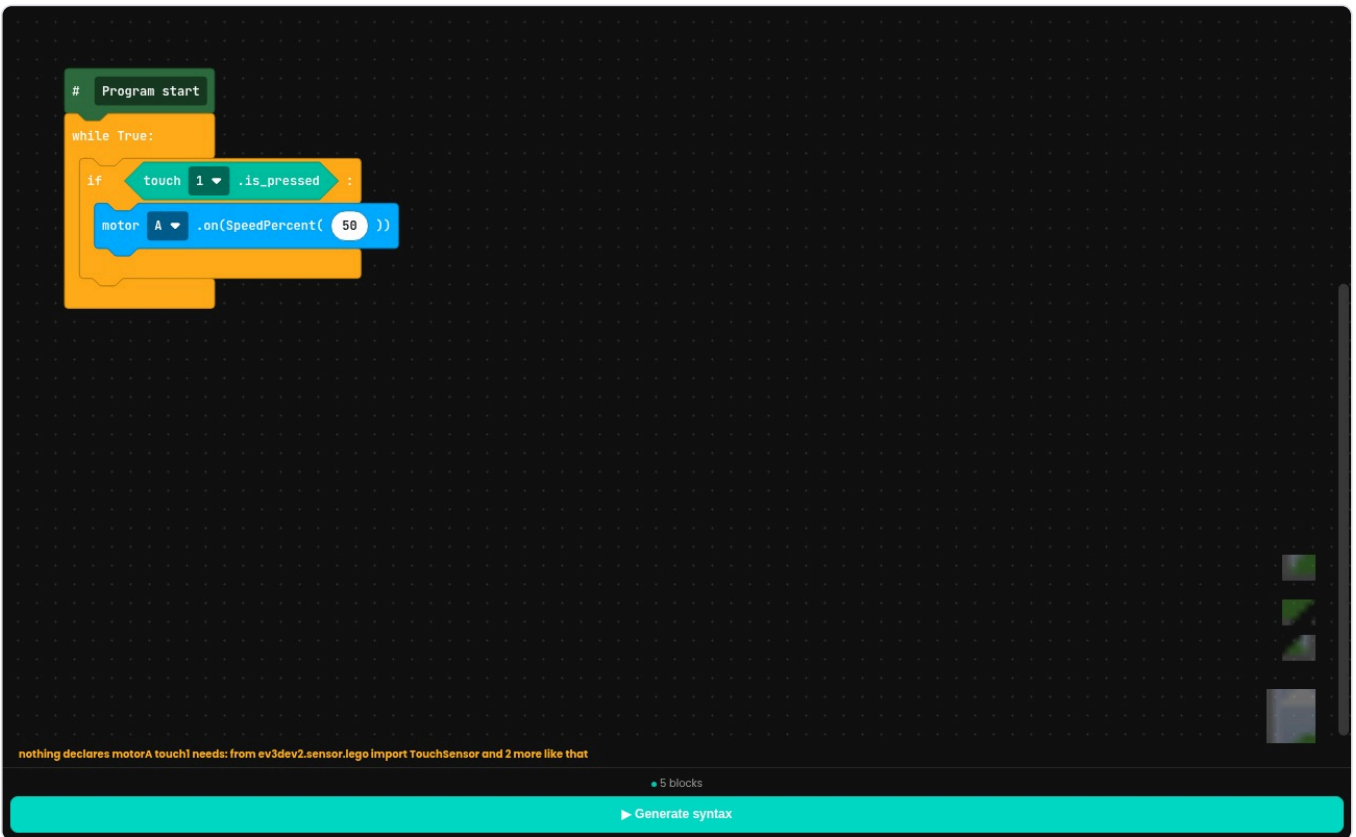
Who it is for

Grades 1 to 3, who cannot yet read the note under the canvas that names the missing line. Older students are meant to find the lines themselves, which is why a teacher can turn the button off in the portal.

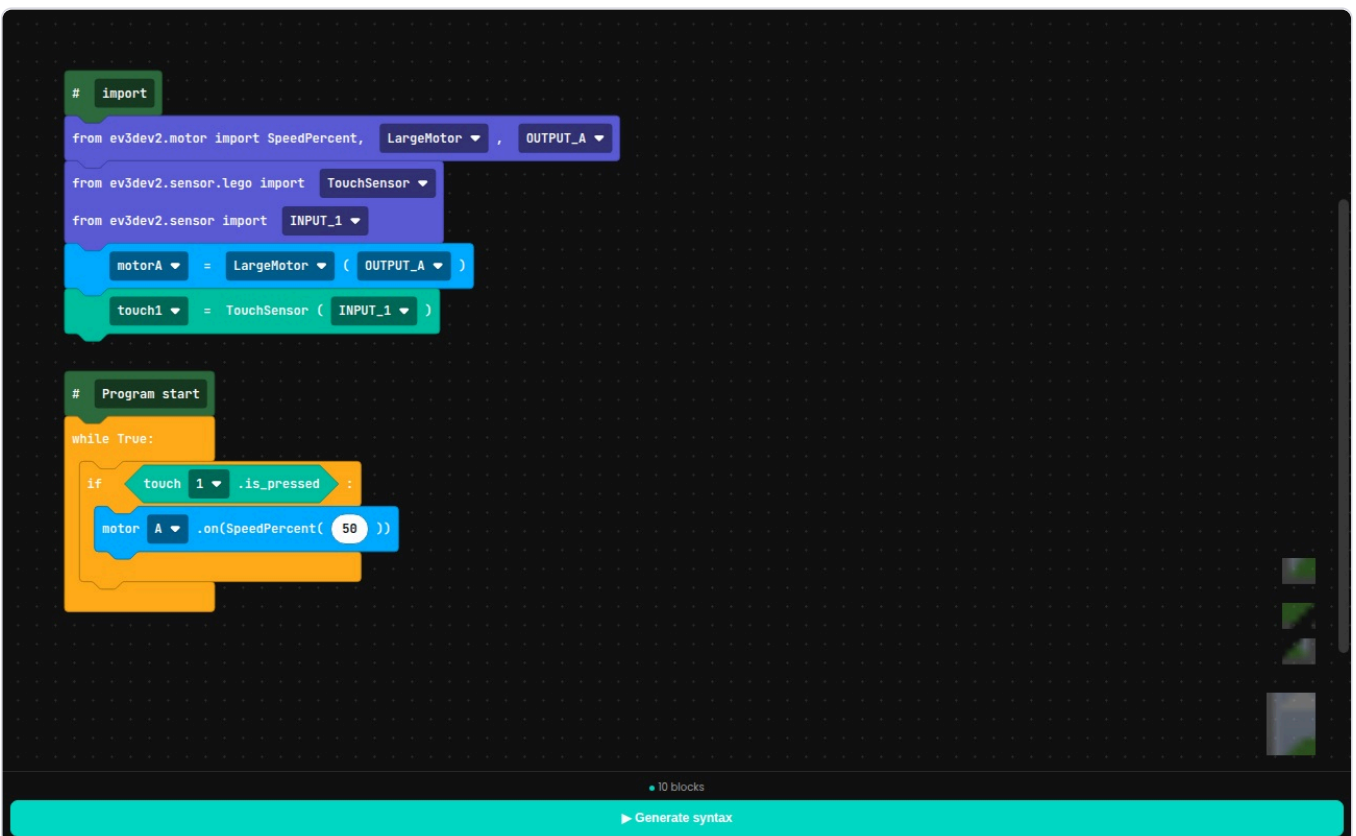
What has not changed

The note under the canvas still names every missing line, and a program missing a line still will not run and will not download. The button is an extra way to add the lines, not a replacement for learning them.

Before: a program with no Imports stack, and the note under the canvas naming what is missing



After one press: the Imports stack sits above everything else, and the note is gone



● 2. Events · yellow

Where a program begins.

BLOCK	WHAT IT DOES
# Program start	The top of your program. Everything hangs underneath it
async def task1():	A second stack that runs at the same time as the first
# comment here	A note for a human. The robot ignores it completely



Why Program start is a comment in Python

Python needs no start line. It simply runs from the top of the file to the bottom, so in Python the Program start block writes only a comment, `# Program start`, and it is green like every comment. C++ and Java do need a start: the program begins inside a function with a special name, and everything the program does goes inside its curly brackets.

LANGUAGE	THE START LINE ON THE EV3	THE START LINE ON A BARE BOARD
Python	# Program start	# Program start
C++	int main() {	void setup() {
Java	public static void main(String[] args) throws Exception {	not offered

Events in C++

The EV3 in C++. The program starts inside `int main()`.



Events in Java

The EV3 in Java. The program starts inside `main`.



3. Motor · blue

Turning one motor, and asking it questions.

The blocks shown here are the EV3's. Every robot has its own words for the same jobs, and Part ten lists them all.

BLOCK	WHAT IT DOES
<code>.on(...)</code>	Turn at a speed and keep going
<code>.on_for_degrees(...)</code>	Turn a set number of degrees, then stop
<code>.on_for_seconds(...)</code>	Turn for a set time, then stop
<code>.stop()</code>	Stop turning
<code>.position</code>	How far it has turned so far
<code>.position = 0</code>	Call where it is now zero

motor A ▾ .on(SpeedPercent(50))

motor A ▾ .on_for_degrees(SpeedPercent(50), 360)

motor A ▾ .on_for_seconds(SpeedPercent(50), 1)

motor A ▾ .stop()

motor A ▾ .position

motor A ▾ .position = 0

motor A ▾ .is_stalled

motor A ▾ .is_running

● 4. MotorGroup · pink

Making several motors act as one. Driving is the usual reason, but not the only one. Not every robot has all of these, because every library offers a different set: SPIKE Prime pairs two motors, the EV3 uses MoveTank and MoveSteering, and Pybricks builds a DriveBase. Where a robot has no equivalent, the block is not offered for that robot.

On the EV3 there are two ways to join motors, and both are here. Steering takes one speed and how hard to turn. Tank gives each side its own speed.

BLOCK	WHAT IT DOES
<code>steer.on(...)</code>	Drive while turning, and keep going
<code>steer.on_for_degrees(...)</code>	Drive a set number of degrees, then stop
<code>steer.on_for_seconds(...)</code>	Drive for a set time, then stop
<code>tank.on(...)</code>	Each side its own speed, and keep going
<code>tank.on_for_degrees(...)</code>	Each side its own speed, for a number of degrees
<code>tank.on_for_seconds(...)</code>	Each side its own speed, for a set time
<code>tank.off()</code>	Stop both motors

```
steer.on( 0 , SpeedPercent( 50 ))
```

```
steer.on_for_degrees( 0 , SpeedPercent( 50 ), 360 )
```

```
steer.on_for_seconds( 0 , SpeedPercent( 50 ), 1 )
```

```
tank.on(SpeedPercent( 50 ), SpeedPercent( 50 ))
```

```
tank.on_for_degrees(SpeedPercent( 50 ), SpeedPercent( 50 ), 360 )
```

```
tank.on_for_seconds(SpeedPercent( 50 ), SpeedPercent( 50 ), 1 )
```

```
tank ▼ .off()
```

Not every robot can do all of these

Tank steering, where each wheel gets its own speed, does not exist on every robot. A robot that cannot do something is not offered the block at all. A block that is greyed out and marked means something else: the language cannot do it, not the robot.

SPIKE Prime: why the category says group and the block says pair

A group can hold any number of motors, which is what you want for a robot with three or four wheels. But SPIKE Prime's own library only offers pairs, exactly two, so on SPIKE Prime the blocks say `motor_pair`. The category is the idea; the block is what that robot can really do.

```
motor_pair.pair(motor_pair. PAIR_1 ▾ , port. A ▾ , port. A ▾ )

motor_pair.unpair(motor_pair. PAIR_1 ▾ )

motor_pair.move(motor_pair. PAIR_1 ▾ , 0 , velocity= 360 )

await motor_pair.move_for_degrees(motor_pair. PAIR_1 ▾ , 360 , 0 , velocity= 360 )

await motor_pair.move_for_time(motor_pair. PAIR_1 ▾ , 1000 , 0 , velocity= 360 )

motor_pair.move_tank(motor_pair. PAIR_1 ▾ , 360 , 360 )

await motor_pair.move_tank_for_degrees(motor_pair. PAIR_1 ▾ , 360 , 360 , 360 )

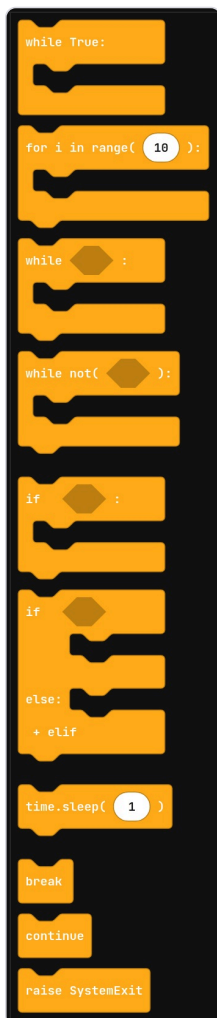
await motor_pair.move_tank_for_time(motor_pair. PAIR_1 ▾ , 360 , 360 , 1000 )

motor_pair.stop(motor_pair. PAIR_1 ▾ ) ▾
```

● 5. Control · orange

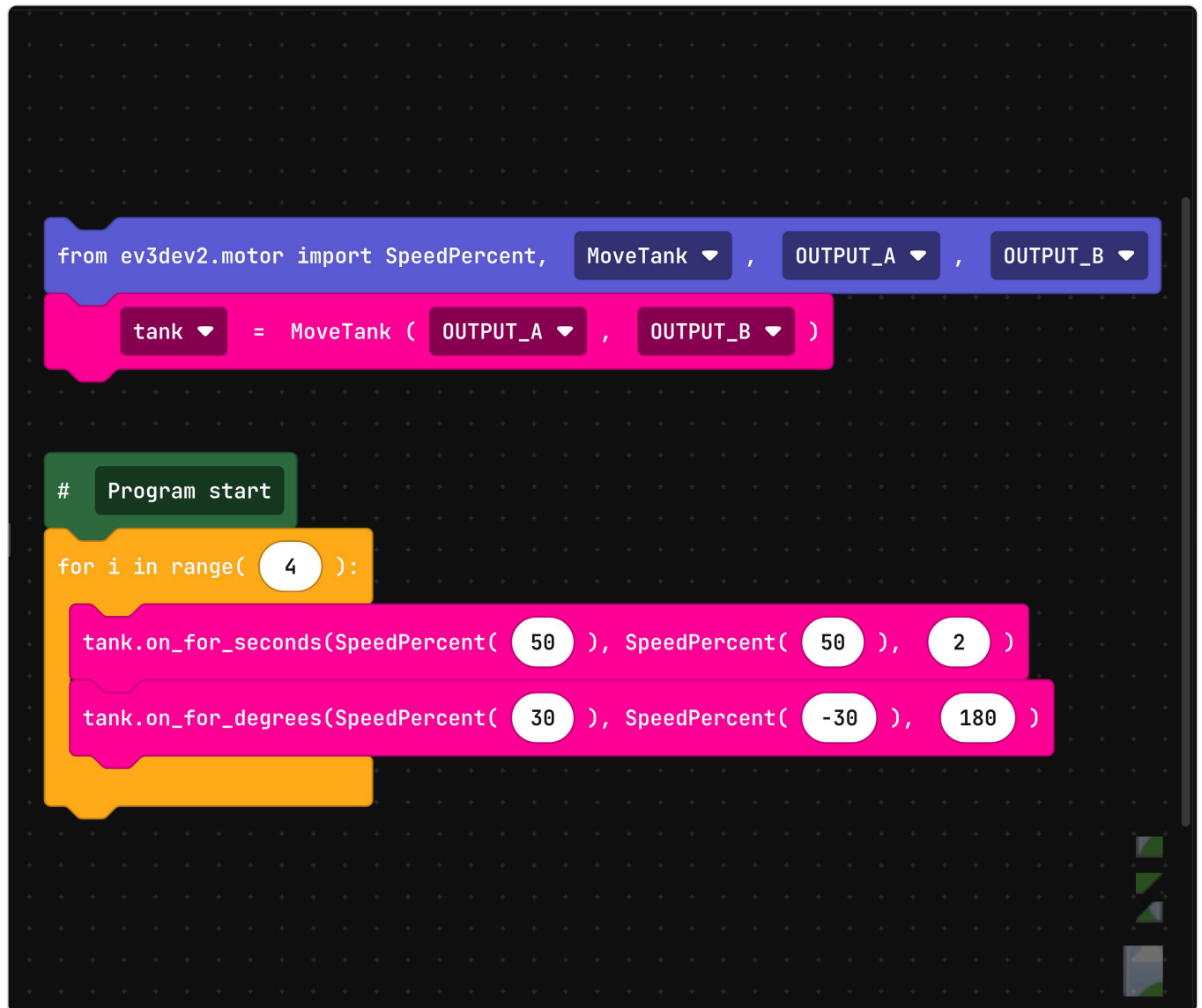
Deciding what happens, and how many times.

BLOCK	WHAT IT DOES
<code>while True:</code>	Keep going around and around, never stopping
<code>for i in range(10):</code>	Go around a set number of times
<code>while <>:</code>	Keep going while something is true
<code>while not(<>):</code>	Keep going until something becomes true
<code>if <>:</code>	Do this only when something is true
<code>if <>: else:</code>	Do this, or else do that
<code>time.sleep(1)</code>	Pause before the next line. On the EV3 it counts in seconds
<code>break</code>	Jump out of the loop right now
<code>continue</code>	Skip the rest of this time around
<code>raise SystemExit</code>	End the whole program



Example: drive a square

A for loop that goes around four times. Each time the robot drives forward for 2 seconds, then turns.



EV3, PYTHON

```

from ev3dev2.motor import SpeedPercent, MoveTank, OUTPUT_A, OUTPUT_B
tank = MoveTank(OUTPUT_A, OUTPUT_B)

# Program start
for i in range(4):
    tank.on_for_seconds(SpeedPercent(50), SpeedPercent(50), 2)
    tank.on_for_degrees(SpeedPercent(30), SpeedPercent(-30), 180)
  
```

The same idea in all three languages

All three columns are the EV3's. This is where the languages look most different. The logic is identical every time; only the writing changes.

WHAT YOU MEAN	PYTHON	C++	JAVA
Go around forever	<code>while True:</code>	<code>while(true) {</code>	<code>while(true) {</code>
Go around ten times	<code>for i in range(10):</code>	<code>for(int i=0;i<10;i++) {</code>	<code>for(int i=0;i<10;i++) {</code>
Only if something is true	<code>if x > 5:</code>	<code>if(x > 5) {</code>	<code>if(x > 5) {</code>
Otherwise	<code>else:</code>	<code>} else {</code>	<code>} else {</code>
Wait one second	<code>time.sleep(1)</code>	<code>std::this_thread::sleep_for(std::chrono::milliseconds(1000));</code>	<code>Thread.sleep(1000);</code>

The two things to notice

Python ends the line with a colon and uses indenting to show what is inside. C++ and Java use curly brackets instead, and end every statement with a semicolon. Same logic, different punctuation.

The blocks with a gap in them

Loops and ifs have a gap in the middle. Whatever you put in the gap is what they control. In the code, that gap is why the lines underneath are indented.

6. Sensors · teal

Reading what is going on around the robot.

BLOCK	WHAT IT DOES
<code>color1.color</code>	What color the color sensor can see
<code>dist1.distance_centimeters</code>	How far away something is
<code>touch1.is_pressed</code>	Is it being pushed, yes or no. The only pointed one here
<code>gyro1.angle</code>	Which way the robot has turned
<code>gyro1.rate</code>	How fast it is turning right now
<code>gyro1.reset()</code>	Call the way it is pointing now zero



Notice the shapes

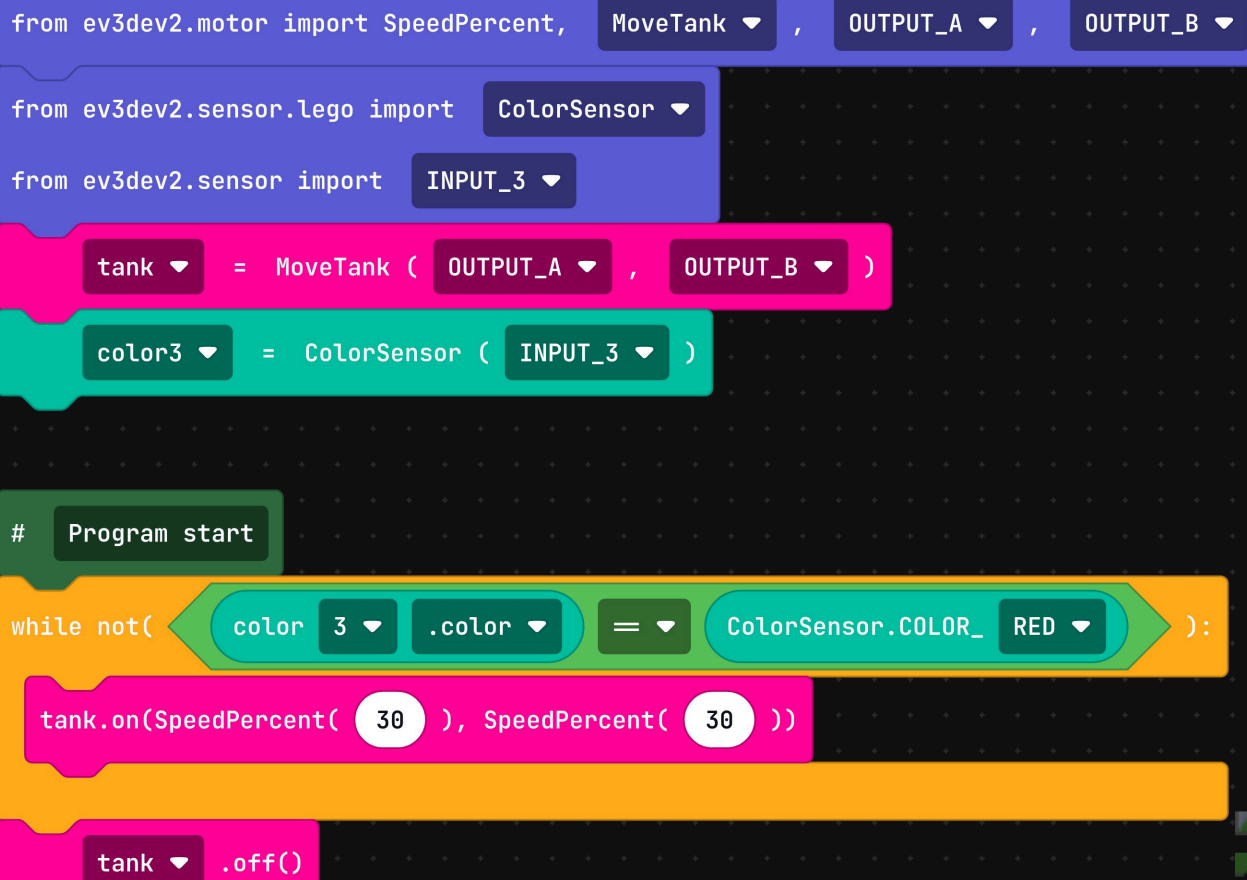
A rounded block gives back a **value**, such as a number or a color. A pointed block gives back a **boolean**: True or False, yes or no. Only the touch sensor block is pointed here, because it is the only one that answers yes or no. A pointed block fits in an if; a rounded one fits in a number slot.

A color is not a number

Most sensor blocks give back a number, so you can compare them with `>` or `<`. The color block is different: it gives back a color, so it can only be compared with another color.

Example: drive until the color sensor sees red

The pointed question compares the color the sensor sees with the color red. The robot keeps driving while the answer is no, and stops when it is yes.



The image shows a Scratch-style block diagram of the Python code. The blocks are color-coded: purple for imports, pink for object creation, orange for loops, and green for comments. The code is as follows:

```
from ev3dev2.motor import SpeedPercent, MoveTank, OUTPUT_A, OUTPUT_B
from ev3dev2.sensor.lego import ColorSensor
from ev3dev2.sensor import INPUT_3

tank = MoveTank ( OUTPUT_A, OUTPUT_B )
color3 = ColorSensor ( INPUT_3 )

# Program start
while not( color 3 .color == ColorSensor.COLOR_ RED ):
    tank.on(SpeedPercent( 30 ), SpeedPercent( 30 ))
    tank.off()
```

EV3, PYTHON

```
from ev3dev2.motor import SpeedPercent, MoveTank, OUTPUT_A, OUTPUT_B
from ev3dev2.sensor.lego import ColorSensor
from ev3dev2.sensor import INPUT_3
tank = MoveTank(OUTPUT_A, OUTPUT_B)
color3 = ColorSensor(INPUT_3)

# Program start
while not (color3.color == ColorSensor.COLOR_RED):
    tank.on(SpeedPercent(30), SpeedPercent(30))
    tank.off()
```

7. Operators · green

Working out an answer: adding numbers, comparing two things, and asking yes or no questions.

Comparing two things

Every one of these gives back a yes or a no, never a number. That is why they fit in the pointed slot of an if block.

BLOCK	WHAT YOU ARE ASKING	EXAMPLE
<code>==</code>	Are these two the same?	<code>angle == 90</code>
<code>!=</code>	Are these two different?	<code>angle != 90</code>
<code>></code>	Is the first one bigger?	<code>angle > 90</code>
<code><</code>	Is the first one smaller?	<code>angle < 90</code>
<code>>=</code>	Bigger, or exactly the same?	<code>angle >= 90</code>
<code><=</code>	Smaller, or exactly the same?	<code>angle <= 90</code>

The one that catches everybody: = and ==

One equals sign **gives** a value. Two equals signs **ask** a question.

`speed = 500` means make speed be 500.

`speed == 500` means is speed the same as 500? The answer is yes or no.

If you use one when you meant two, the robot will not do what you expect, and it usually will not tell you why.

Yes and no questions

BLOCK	WHAT IT MEANS	YES WHEN
<code>and</code>	Both have to be true	both sides are yes
<code>or</code>	Either one will do	at least one side is yes
<code>not</code>	Flips the answer around	the thing inside was no

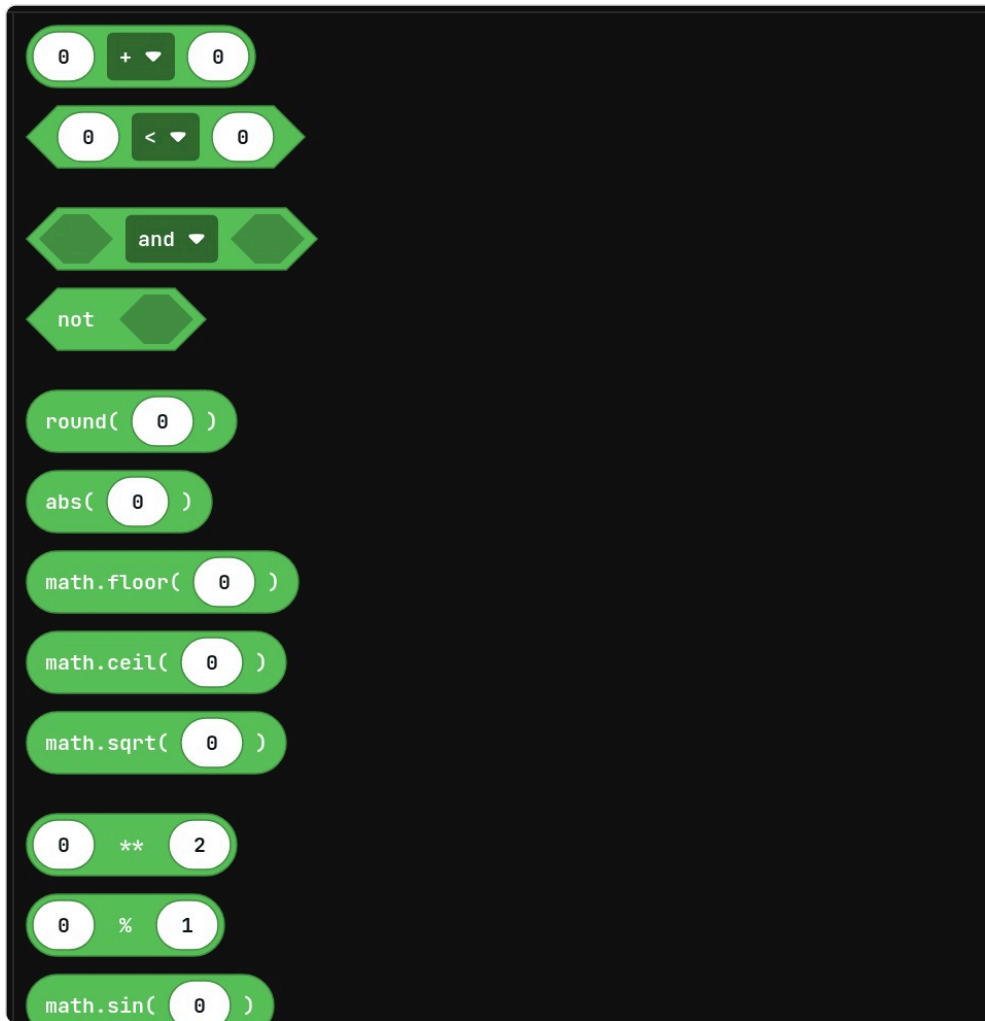
WHAT YOU MEAN	PYTHON	C++	JAVA
Both have to be true	<code>and</code>	<code>&&</code>	<code>&&</code>
Either one will do	<code>or</code>	<code> </code>	<code> </code>
Flip the answer	<code>not</code>	<code>!</code>	<code>!</code>
Yes	<code>True</code>	<code>true</code>	<code>true</code>
No	<code>False</code>	<code>false</code>	<code>false</code>

Watch the capital letter

Python writes True and False with a capital. C++ and Java write them lowercase. It is a small thing that stops a program running.

Working out a number

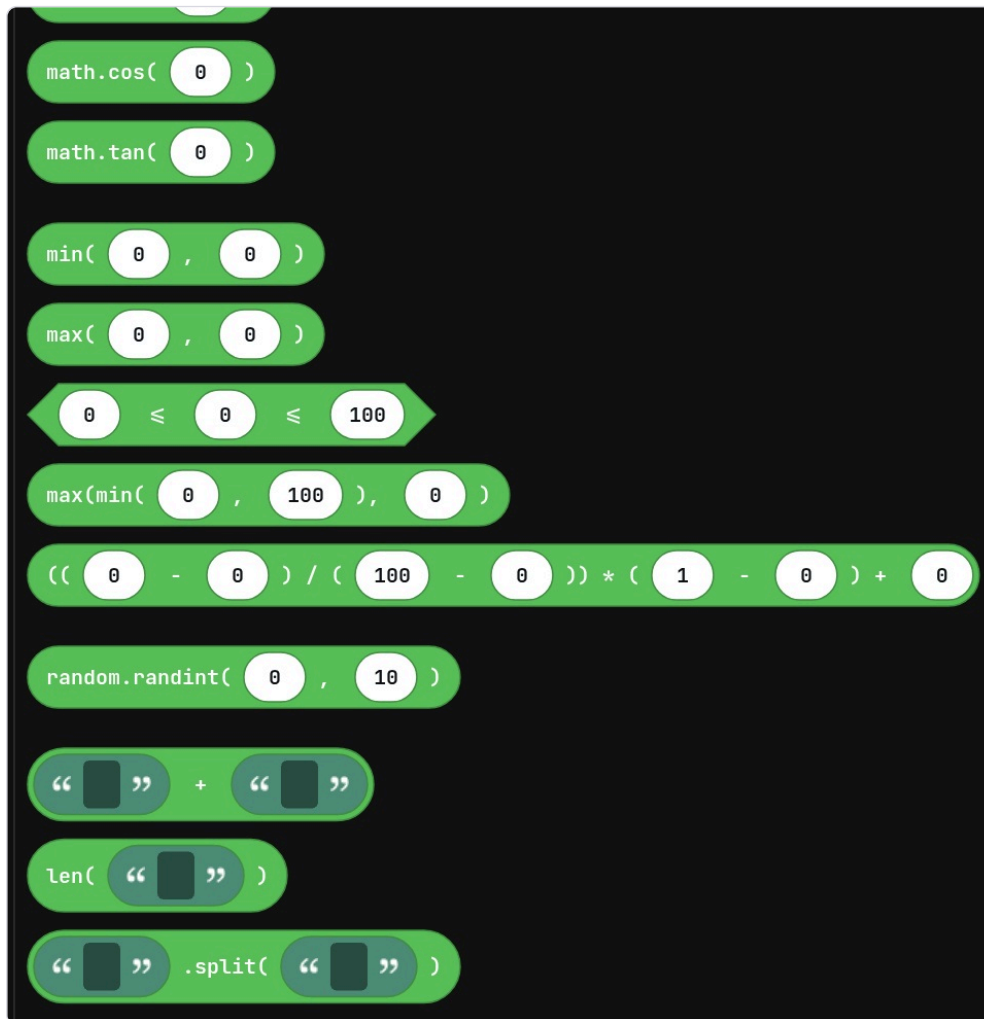
BLOCK	WHAT IT DOES	EXAMPLE
<code>+</code> <code>-</code> <code>*</code> <code>/</code>	Add, take away, multiply, divide	<code>speed + 100</code>
<code>%</code>	The remainder after dividing. 7 % 2 is 1	<code>count % 2</code>
<code>**</code>	To the power of. 2 ** 3 is 8	<code>2 ** 3</code>
<code>round</code>	To the nearest whole number	<code>round(3.7)</code>
<code>abs</code>	Throws away the minus sign	<code>abs(-40)</code>
<code>math.floor</code> , <code>math.ceil</code>	Always down, or always up, to a whole number	<code>math.floor(3.9)</code>
<code>sqrt</code>	Square root	<code>sqrt(16)</code>
<code>min</code> , <code>max</code>	The smaller, or the bigger, of two	<code>min(a, b)</code>
<code>random.randint</code>	A surprise number in a range	<code>random.randint(1, 6)</code>
<code>max(min())</code>	Keeps a number inside a low and a high. Called constrain in C++	<code>max(min(x, 100), 0)</code>



Working with words

Not every value is a number. These three work on words.

BLOCK	WHAT IT DOES
<code>" " + " "</code>	Joins two lots of words into one
<code>len(" ")</code>	How many letters are in it
<code>" ".split(" ")</code>	Chops it up wherever a letter you choose appears

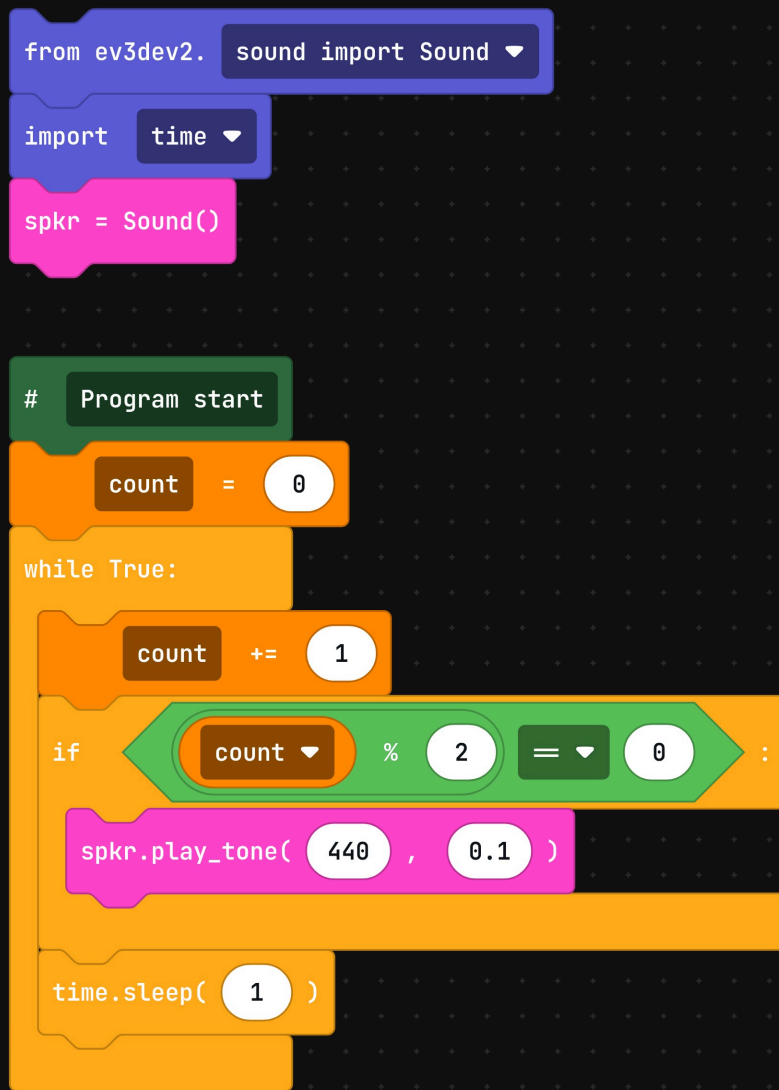


Why % is more useful than it looks

`count % 2` is 0 for every even number and 1 for every odd one. It is how you make something happen every other time.

Example: beep every other second

`count` goes up by one every second. When `count % 2` is 0, `count` is even, and the robot beeps.



EV3, PYTHON

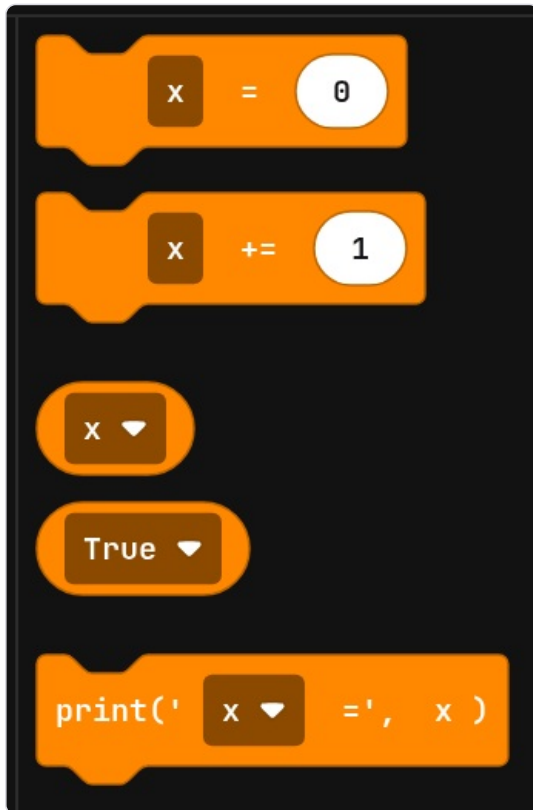
```
from ev3dev2.sound import Sound
import time
spkr = Sound()

# Program start
count = 0
while True:
    count += 1
    if count % 2 == 0:
        spkr.play_tone(440, 0.1)
    time.sleep(1)
```

● 8. Variables · dark orange

A variable is a name you make up to hold a value, such as a number, so your program can use it and change it later.

BLOCK	WHAT IT DOES
<code>x = 0</code>	Make a name, and give it a value
<code>x += 1</code>	Add to what is already in it
<code>x</code>	Use what is in it
<code>True</code>	A yes or a no you can put in a name
<code>print('x =', x)</code>	Show its value on the screen, so you can see what went wrong



Example: count, and watch it in the Monitor

x starts at 0 and goes up by one every second. print sends each new value to the Monitor, so you can watch the variable change while the program runs.

```
import time

# Program start
x = 0
while True:
    x += 1
    print('x = ', x)
    time.sleep(1)
```

EV3, PYTHON

```
import time

# Program start
x = 0
while True:
    x += 1
    print('x = ', x)
    time.sleep(1)
```

The same idea in all three languages

All three columns are the EV3's.

WHAT YOU MEAN	PYTHON	C++	JAVA
Make a name holding a number	<code>score = 0</code>	<code>int score = 0;</code>	<code>int score = 0;</code>
Add one to it	<code>score += 1</code>	<code>score++;</code>	<code>score++;</code>
Show it on screen	<code>print(score)</code>	<code>std::cout << score << std::endl;</code>	<code>System.out.println(score);</code>

Why C++ and Java say int

They want to know what kind of thing a name will hold before you use it, and a whole number is called an int. Python works it out for itself.

Declaring a motor makes a variable too

`motorA = Motor(Port.A)` and `score = 0` are the same shape. One holds a motor, the other holds a number.

● 9. Functions · red

Making your own block out of other blocks.

BLOCK

```
def my_function():
```

```
my_function()
```

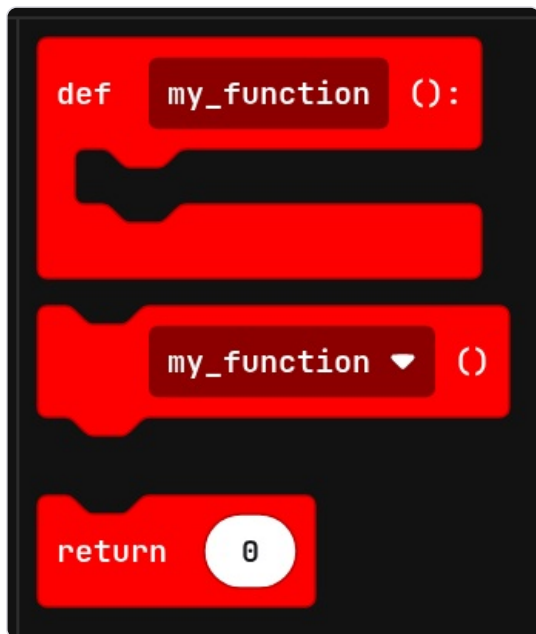
```
return 0
```

WHAT IT DOES

Make a new block, and say what it does

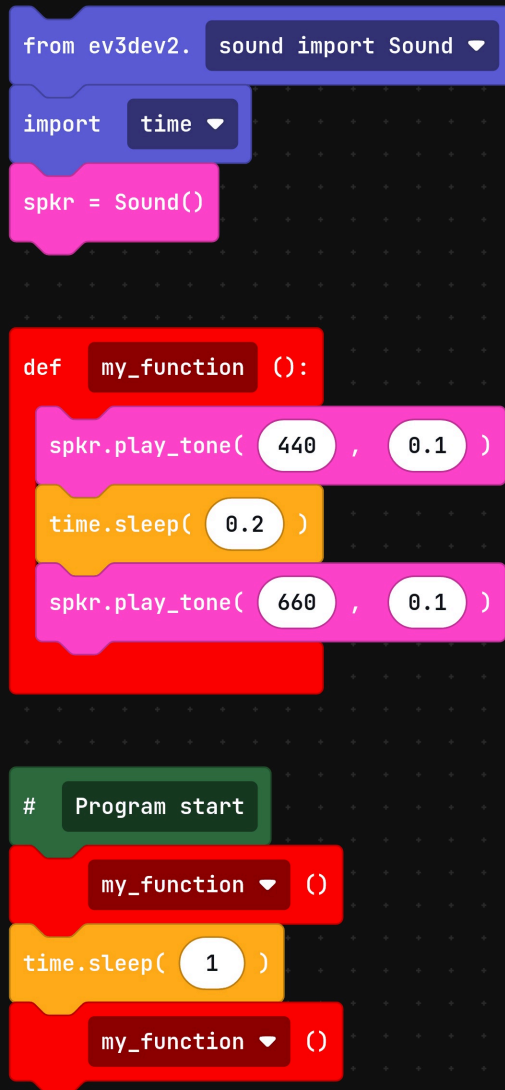
Use the block you made

Send an answer back to whoever called it



Example: one block that plays two notes

my_function plays a low note and then a high one. The program calls it twice, one second apart, instead of building the same three blocks twice.



EV3, PYTHON

```
from ev3dev2.sound import Sound
import time
spkr = Sound()

def my_function():
    spkr.play_tone(440, 0.1)
    time.sleep(0.2)
    spkr.play_tone(660, 0.1)

# Program start
my_function()
time.sleep(1)
my_function()
```

Why this is worth learning early

If you find yourself dragging the same four blocks over and over, make them into one block instead, and give it a name that says what it does.

● 10. Sound · bright pink

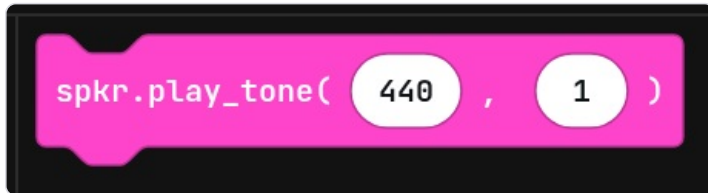
Making the robot play a sound.

BLOCK

WHAT IT DOES

```
spkr.play_tone(440, 1)
```

Play a note at a pitch, for a length of time



Playing a note waits

This is a blocking block. The program stops on that line until the note has finished, so a long note inside a loop holds everything else up.

● 11. Light · purple

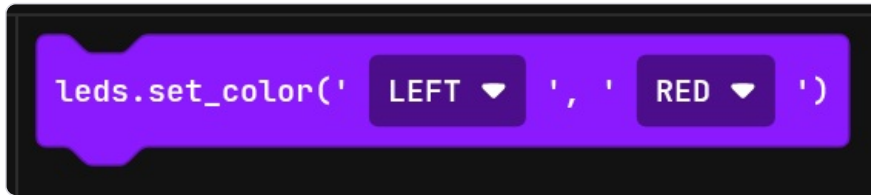
Turning on the robot's lights and its display.

BLOCK

WHAT IT DOES

```
leds.set_color('LEFT', 'RED')
```

Turn on one of the lights in a color



One block here, more on other robots

The EV3 has two lights and nothing else, so it has one block. A robot with a display has more.

● 12. Timing · light blue

Measuring how much time has passed.

BLOCK

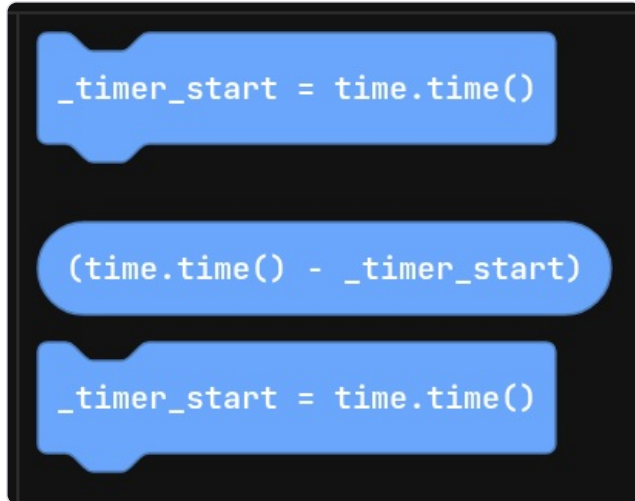
WHAT IT DOES

```
_timer_start = time.time()
```

Begin counting from zero

```
(time.time() - _timer_start)
```

How long since it started, in seconds



A timer is not the same as wait

Wait stops your program for a moment. A timer counts in the background while everything else carries on.

Example: drive for 3 seconds, timed

The timer starts at zero. The loop keeps the motors on while less than 3 seconds have passed, then the motors stop.

```
from ev3dev2.motor import SpeedPercent, MoveTank, OUTPUT_A, OUTPUT_B
```

```
import time
```

```
tank = MoveTank ( OUTPUT_A, OUTPUT_B )
```

```
# Program start
```

```
_timer_start = time.time()
```

```
while (time.time() - _timer_start) < 3:
```

```
    tank.on(SpeedPercent(50), SpeedPercent(50))
```

```
tank.off()
```

EV3, PYTHON

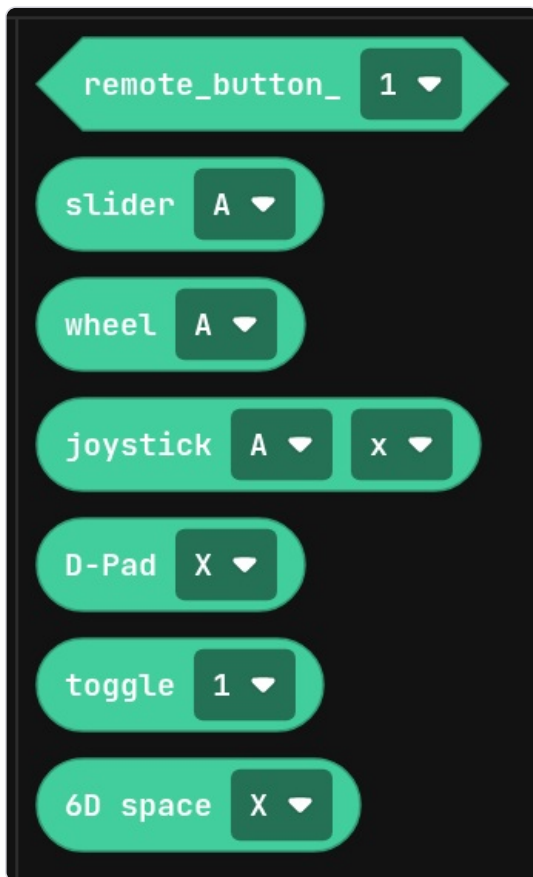
```
from ev3dev2.motor import SpeedPercent, MoveTank, OUTPUT_A, OUTPUT_B
import time
tank = MoveTank(OUTPUT_A, OUTPUT_B)

# Program start
_timer_start = time.time()
while (time.time() - _timer_start) < 3:
    tank.on(SpeedPercent(50), SpeedPercent(50))
tank.off()
```

13. Remote · mint

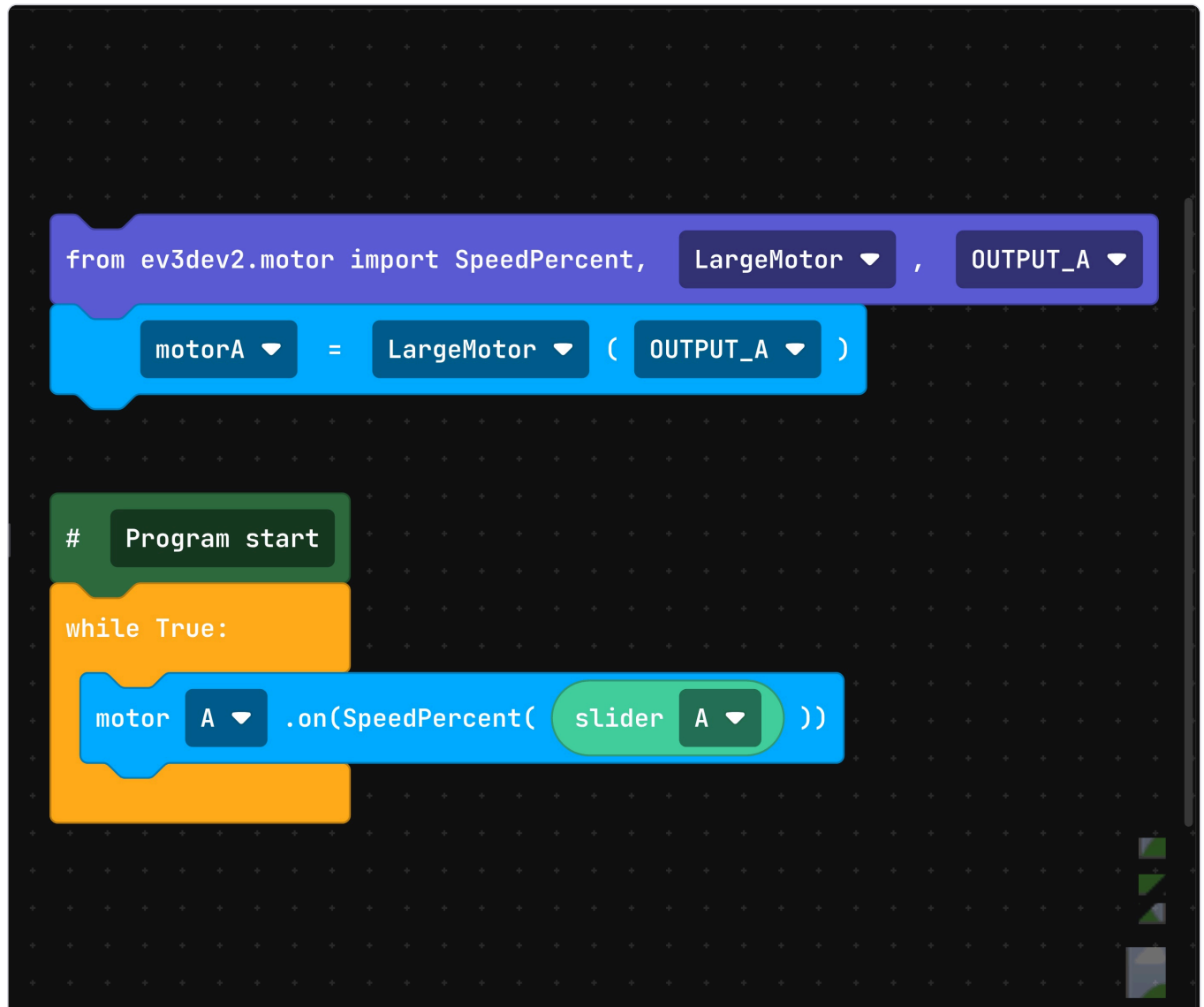
The widgets you build on the remote control page, read by your program while it runs. Each block is the value of one widget, and it drops into any block that wants a number. They are the same on every robot and in every language.

BLOCK	WHAT IT READS
remote_button_1	A button, pressed or not
remote_slider_A	Where a slider is
remote_wheel_A	How far the steering wheel is turned
remote_joy_A_x	One direction of a joystick
remote_dpad_x	One direction of the D-Pad
remote_toggle_1	A toggle, on or off
remote_space_x	One direction of the 6D Space



Example: a slider sets the motor speed

Slider A on the remote control page gives a number. The motor block reads it every time the loop comes around, so moving the slider changes the speed while the program runs.



EV3, PYTHON

```
from ev3dev2.motor import SpeedPercent, LargeMotor, OUTPUT_A
motorA = LargeMotor(OUTPUT_A)

# Program start
while True:
    motorA.on(SpeedPercent(remote_slider_A))
```

Why it feels different from every other category

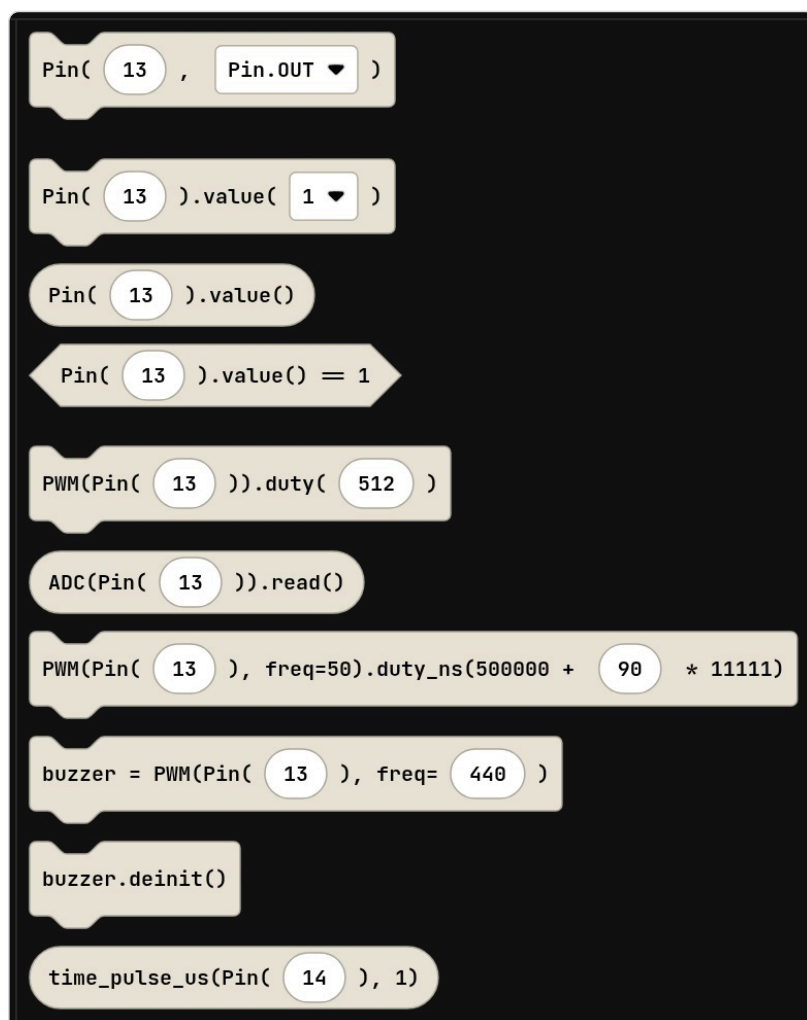
Every other block tells the robot what to do. These let you tell it what to do while it is already running.

14. Circuit · red and blue

Wiring your own parts to a board, one pin at a time.

Circuit appears on robots with numbered pins, such as the ESP32, the Arduino and the micro:bit. The table and the picture here are the ESP32's, in Python.

BLOCK	WHAT IT DOES
<code>Pin(13, Pin.OUT)</code>	Say whether a pin is listening or talking
<code>Pin(13).value(1)</code>	Turn a pin fully on or fully off
<code>Pin(13).value()</code>	Is this pin on or off
<code>Pin(13).value() == 1</code>	Is this pin on, yes or no
<code>PWM(Pin(13)).duty(512)</code>	Turn a pin partly on, for dimming or speed
<code>ADC(Pin(13)).read()</code>	Read a value that is not just on or off
<code>PWM(Pin(13), freq=50).duty_ns(500000 + 90 * 11111)</code>	Move a servo to an angle, here 90
<code>buzzer = PWM(Pin(13), freq=440)</code>	Make a buzzer sound
<code>buzzer.deinit()</code>	Stop the buzzer
<code>time_pulse_us(Pin(14), 1)</code>	Time a pulse, which is how a distance sensor on two pins is read



```
Pin( 13 , Pin.OUT )

Pin( 13 ).value( 1 )

Pin( 13 ).value()

Pin( 13 ).value() == 1

PWM(Pin( 13 )).duty( 512 )

ADC(Pin( 13 )).read()

PWM(Pin( 13 ), freq=50).duty_ns(500000 + 90 * 11111)

buzzer = PWM(Pin( 13 ), freq= 440 )

buzzer.deinit()

time_pulse_us(Pin( 14 ), 1)
```

This category only appears on a robot with pins

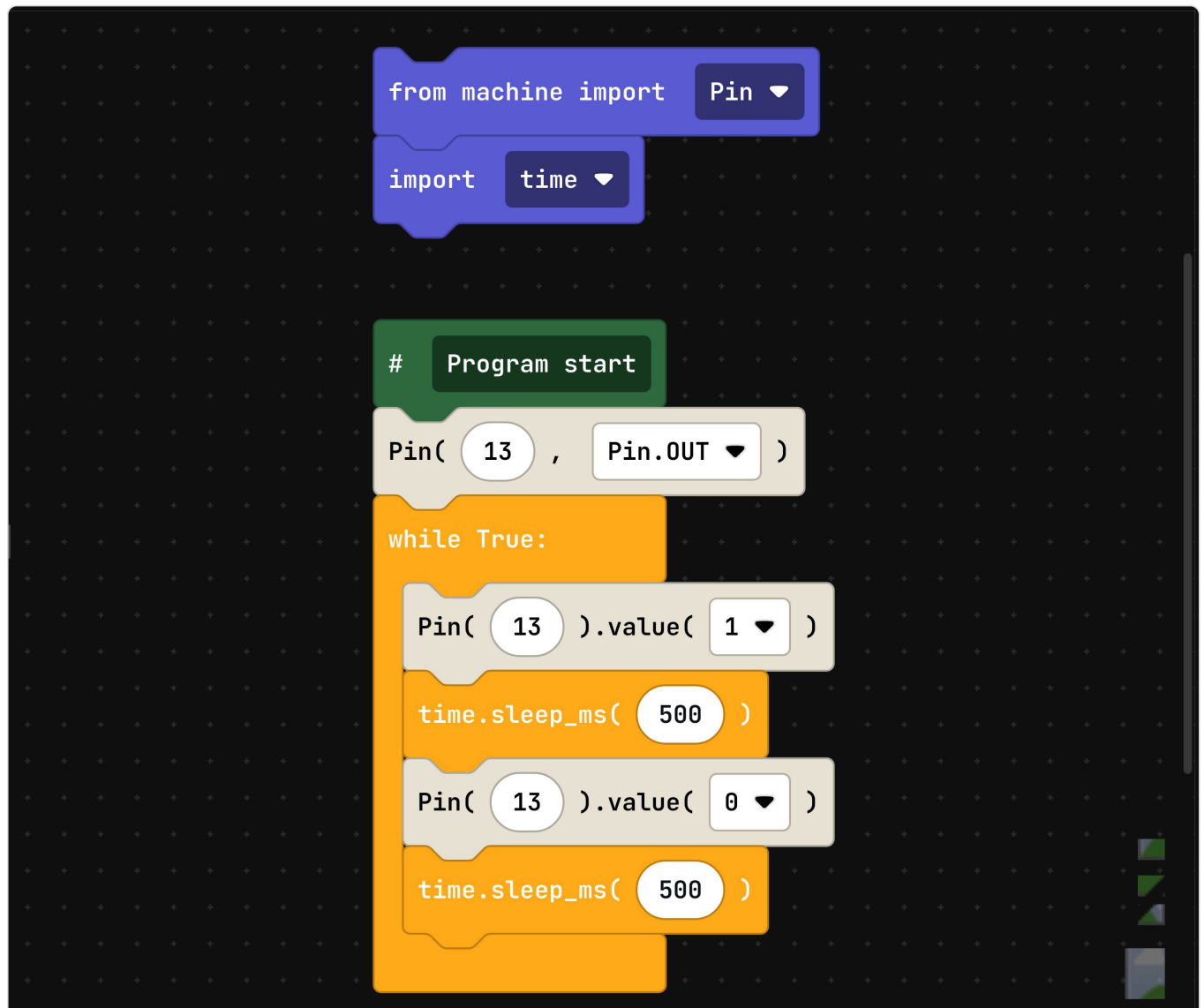
A hub has ports and no bare pins, so Circuit is not offered for a hub. A robot with numbered pins has it.

The same pin, two different numbers

Read a pin as a number and the Arduino answers 0 to 1023, while the ESP32 answers 0 to 4095. That is the two boards being built differently, not StarCode choosing.

Example: blink an LED

Pin 13 is set as an output, then turned on and off every half second. Wire an LED and a resistor from pin 13 to ground to see it.



ESP32, PYTHON

```
from machine import Pin
import time

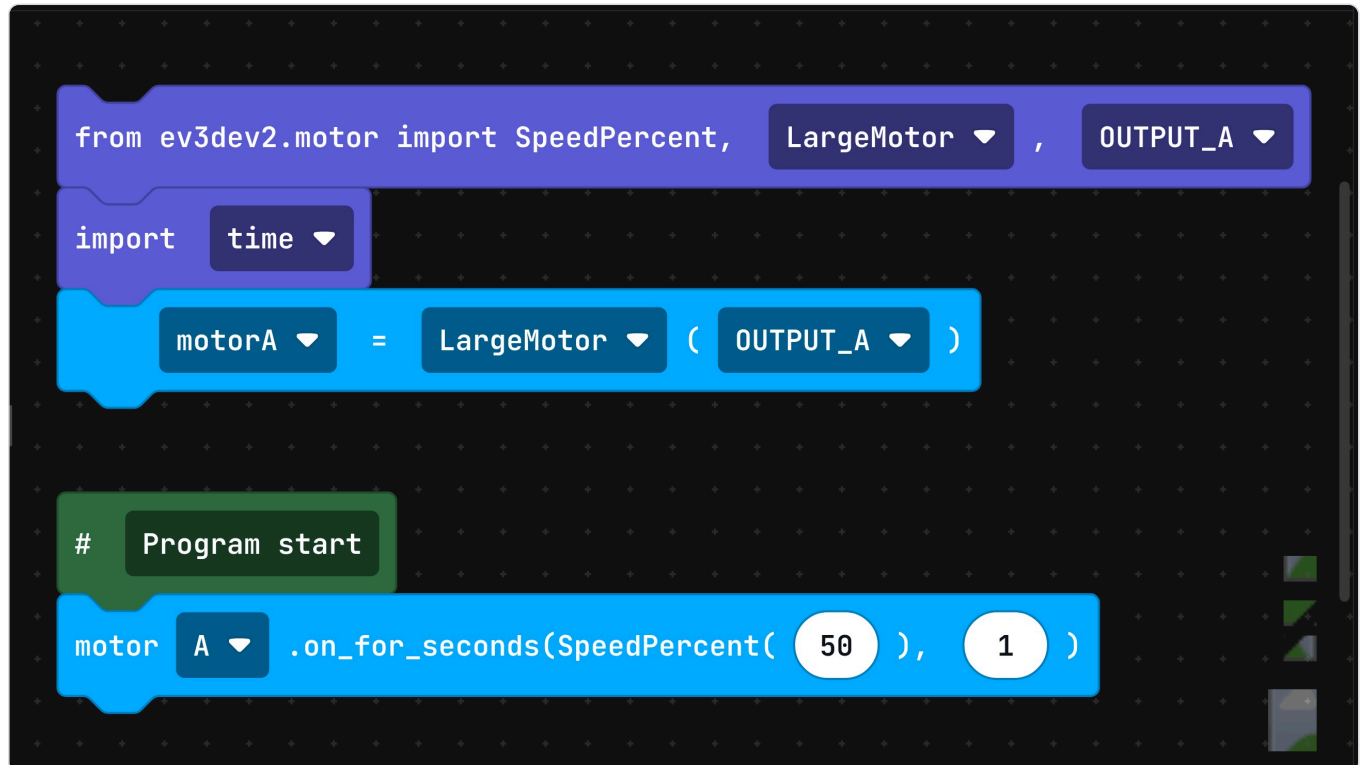
# Program start
Pin(13, Pin.OUT)
while True:
    Pin(13).value(1)
    time.sleep_ms(500)
    Pin(13).value(0)
    time.sleep_ms(500)
```

Three whole programs

The tables above say what each block does. These are three finished programs, each one harder than the last. All three are on the EV3, so they match the pictures above. Every one has two stacks: the imports and declaring lines on one, and the program on the other.

Example one · the smallest program that works

Two import lines, one declaring line, and a motor that runs for one second. Five blocks. Two to bring in the tools, one to give the motor a name, two to use it.

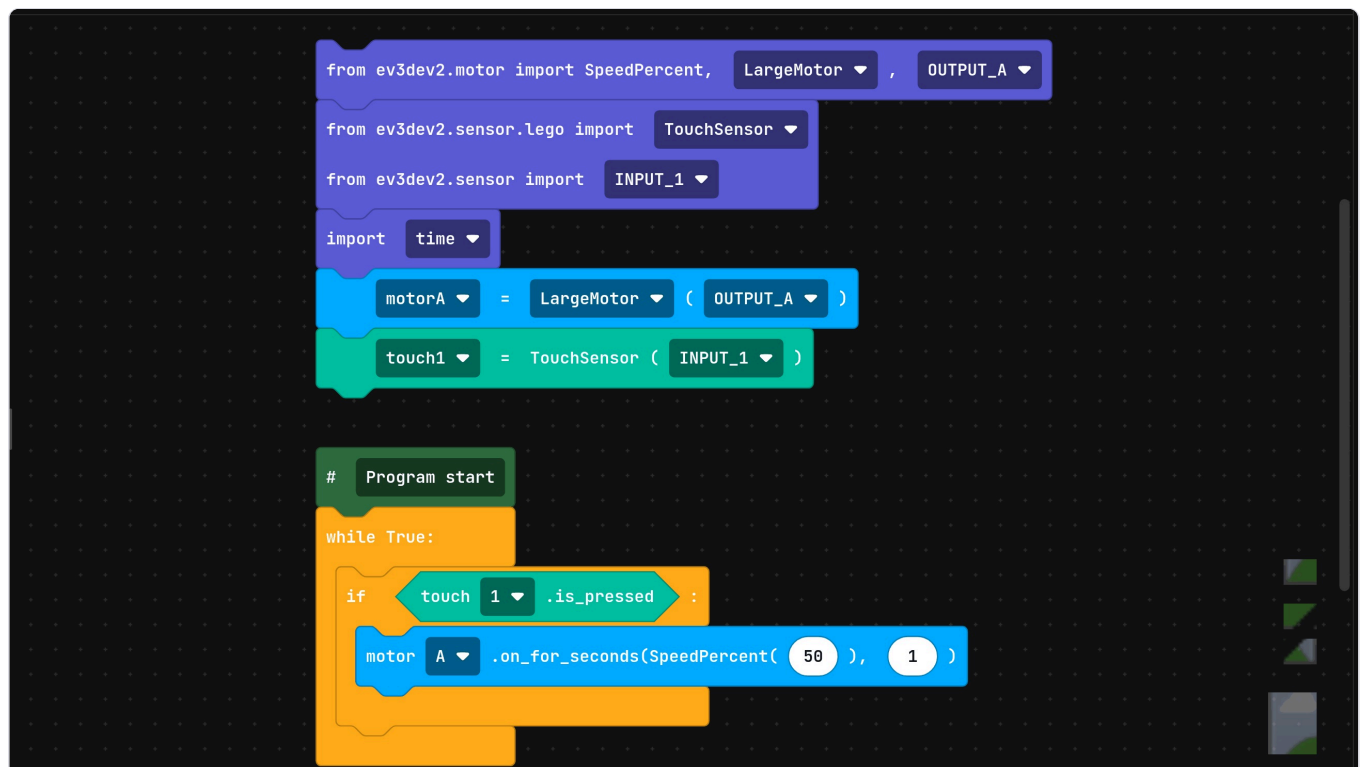


```
from ev3dev2.motor import SpeedPercent, LargeMotor, OUTPUT_A
import time
motorA = LargeMotor ( OUTPUT_A )

# Program start
motor A .on_for_seconds(SpeedPercent( 50 ), 1 )
```

Example two · adding a sensor and a question

One more import block, the one that writes two lines, and one more declaring line, because there is a second thing plugged in. In the program, a loop that keeps asking whether the touch sensor is pressed. Notice the pointed teal block sitting in the if: only a pointed block fits there, because only a question can be answered yes or no.

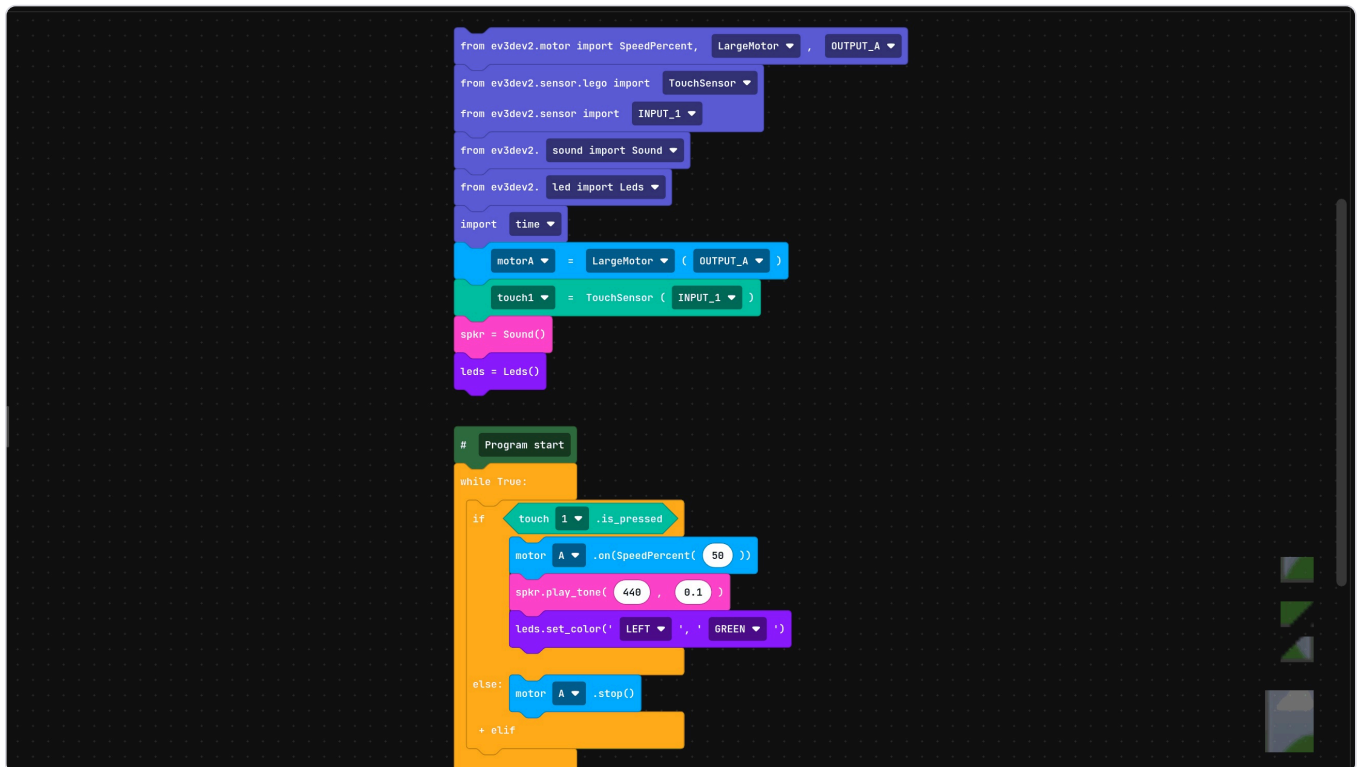


```
from ev3dev2.motor import SpeedPercent, LargeMotor, OUTPUT_A
from ev3dev2.sensor.lego import TouchSensor
from ev3dev2.sensor import INPUT_1
import time
motorA = LargeMotor ( OUTPUT_A )
touch1 = TouchSensor ( INPUT_1 )

# Program start
while True:
    if touch 1 .is_pressed :
        motor A .on_for_seconds(SpeedPercent( 50 ), 1 )
```

Example three · the parts with no port

Two more import lines and two more declaring lines, for the speaker and the lights. Those two declaring lines have no port in them, because there is only one speaker and one set of lights and they are built into the robot. In the program, the same question now sets off three things at once, and an else underneath stops the motor when the sensor is let go.



Why the tone in example three is so short

It is 0.1 of a second, not one second. Playing a note is a blocking block, so the loop cannot come around again until the note has finished. With a one second note the motor kept running for up to a second after the sensor was let go. This is the blocking idea in Part three, met in a real program.

What grows and what does not

Look at the left stack across the three. It gets longer every time, because every new part needs importing and declaring. The right stack stays almost the same shape. That is the pattern: the setup grows with the robot, the program grows with the idea.

Ideas behind the code

Not words, and not blocks. These are the ideas a student has to understand before a program behaves the way they expect. Added to as they come up.

Blocking and non-blocking

This is the one that explains most of the times a robot does something surprising.

KIND	WHAT IT DOES	LOOKS LIKE
Blocking	The program stops on that line and waits until the motor has finished. Nothing below it happens until then.	<pre>motorA.run_angle(500, 360) motorA.on_for_seconds(SpeedPercent(50), 1)</pre>
Non-blocking	The program turns the motor on and moves straight to the next line. The motor keeps going while everything below runs.	<pre>motorA.run(500) motorA.on(SpeedPercent(50))</pre>

How to tell which one you have

If the block says how far or how long, it waits. A number of degrees, or a number of seconds, means the program stops there until it is done. If the block only says how fast, it does not wait.

Why this catches people out

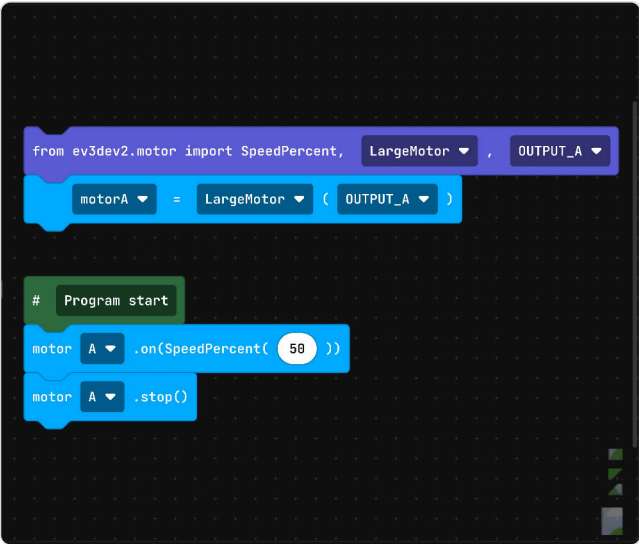
A student puts a motor block and then a stop block underneath, and the robot seems to ignore the motor. With a non-blocking block that is exactly right: the motor turned on, the very next line turned it off, and it happened too fast to see. With a blocking block the same two lines work, because the second one does not run until the first has finished.

Why you need both

Blocking is how you say do this, then that: drive forward two turns, then turn. Non-blocking is how you do two things at once, or how you keep checking a sensor while the motor is still running. A line follower cannot work with blocking blocks, because the program would never get back to reading the sensor.

The motor seems to do nothing

on is non-blocking, so stop runs straight away.



```
from ev3dev2.motor import SpeedPercent, LargeMotor, OUTPUT_A
motorA = LargeMotor(OUTPUT_A)

# Program start
motorA.on(SpeedPercent(50))
motorA.stop()
```

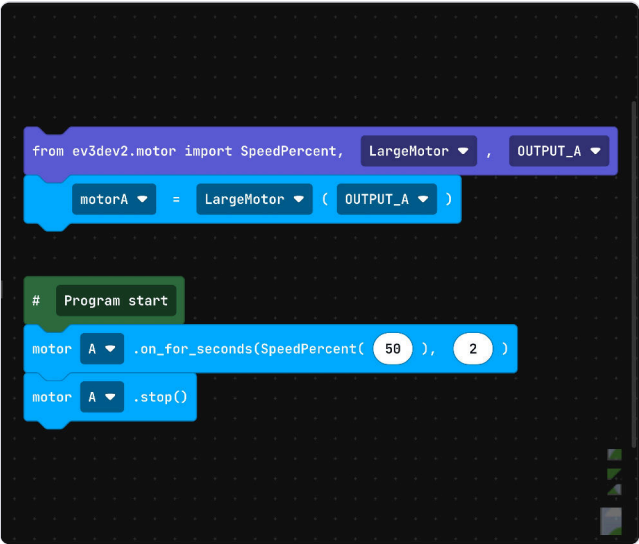
EV3, PYTHON

```
from ev3dev2.motor import SpeedPercent, LargeMotor, OUTPUT_A
motorA = LargeMotor(OUTPUT_A)

# Program start
motorA.on(SpeedPercent(50))
motorA.stop()
```

The motor runs for 2 seconds

on_for_seconds is blocking, so stop waits its turn.



```
from ev3dev2.motor import SpeedPercent, LargeMotor, OUTPUT_A
motorA = LargeMotor(OUTPUT_A)

# Program start
motorA.on_for_seconds(SpeedPercent(50), 2)
motorA.stop()
```

EV3, PYTHON

```
from ev3dev2.motor import SpeedPercent, LargeMotor, OUTPUT_A
motorA = LargeMotor(OUTPUT_A)

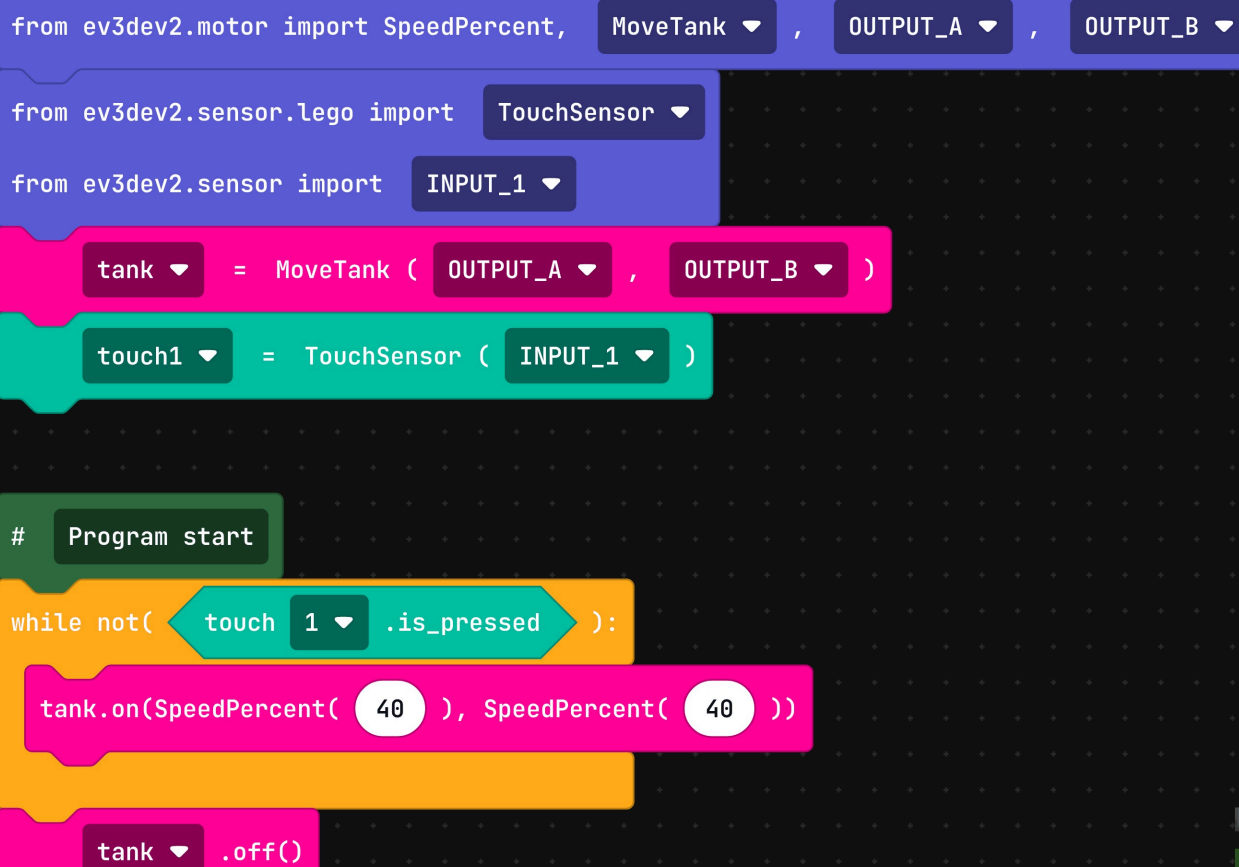
# Program start
motorA.on_for_seconds(SpeedPercent(50), 2)
motorA.stop()
```

The ports have to match three times

A motor or a sensor is named in three places, and all three must agree. The import line brings in the port, such as `INPUT_1`. The declaring line gives the part a name on that port, such as `touch1 = TouchSensor(INPUT_1)`. And the cable on the robot has to be plugged into port 1. If any one of the three says a different port, the program reads the wrong port and the robot does nothing, or does the wrong thing, with no error to tell you why.

Example: drive until the touch sensor is pressed

`INPUT_1` in the import line, `INPUT_1` in the declaring line, and the touch sensor cable in port 1. The motors are on `OUTPUT_A` and `OUTPUT_B`, so their cables go in ports A and B.



```
from ev3dev2.motor import SpeedPercent, MoveTank, OUTPUT_A, OUTPUT_B
from ev3dev2.sensor.lego import TouchSensor
from ev3dev2.sensor import INPUT_1

tank = MoveTank(OUTPUT_A, OUTPUT_B)
touch1 = TouchSensor(INPUT_1)

# Program start
while not(touch1.is_pressed):
    tank.on(SpeedPercent(40), SpeedPercent(40))
    tank.off()
```

EV3, PYTHON

```
from ev3dev2.motor import SpeedPercent, MoveTank, OUTPUT_A, OUTPUT_B
from ev3dev2.sensor.lego import TouchSensor
from ev3dev2.sensor import INPUT_1
tank = MoveTank(OUTPUT_A, OUTPUT_B)
touch1 = TouchSensor(INPUT_1)

# Program start
while not (touch1.is_pressed):
    tank.on(SpeedPercent(40), SpeedPercent(40))
    tank.off()
```

Check this first

When a program looks right and the robot does nothing, check the three ports before changing any code. It is the most common fault there is.

Sequencing: the order is the program

A program runs from the top down, one line at a time. The same blocks in a different order are a different program. Driving and then turning is not the same as turning and then driving.

Open loop and closed loop

Driving for 2 seconds is **open loop**: the robot does not check anything, so a low battery or a slippery floor changes how far it goes. Driving until a sensor says stop is **closed loop**: the robot checks and corrects, so it ends up in the same place every time. The drive a square example above is open loop; the touch sensor example is closed loop.

Thresholds: pick the number by measuring

A **threshold** is the number a reading is compared with, such as dark when reflected light is below 40. Do not guess it. Print the sensor's value to the Monitor, hold the sensor over the dark part and the light part, and pick a number halfway between the two readings. A room with different light needs a different threshold.

Debugging: print what the program sees

When a robot does something unexpected, the program is doing exactly what it was told with the values it has. Print those values to the Monitor and you see what the robot sees. The counting example in Variables shows how.

Programs to build

Standard robot programs, written with StarCode's own blocks on the EV3 in Python. They use a drive base on ports A and B, a color sensor pointing down on port 3, and a distance sensor pointing forward on port 4. If your robot is wired differently, change the ports in the import and declaring lines so all three places match.

Line follower

Follows the edge of a dark line. When the sensor sees dark, the robot curves one way; when it sees light, it curves the other way, so it zigzags along the edge. This is bang bang control. Measure your own threshold instead of 40.

```

from ev3dev2.motor import SpeedPercent, MoveTank , OUTPUT_A , OUTPUT_B

from ev3dev2.sensor.lego import ColorSensor

from ev3dev2.sensor import INPUT_3

tank = MoveTank ( OUTPUT_A , OUTPUT_B )

color3 = ColorSensor ( INPUT_3 )

```

```

# Program start

```

```

while True:

```

```

    if color 3 .reflected_light_intensity < 40:
        tank.on(SpeedPercent( 10 ), SpeedPercent( 40 ))
    else:
        tank.on(SpeedPercent( 40 ), SpeedPercent( 10 ))
    + elif

```

EV3, PYTHON

```

from ev3dev2.motor import SpeedPercent, MoveTank, OUTPUT_A, OUTPUT_B
from ev3dev2.sensor.lego import ColorSensor
from ev3dev2.sensor import INPUT_3
tank = MoveTank(OUTPUT_A, OUTPUT_B)
color3 = ColorSensor(INPUT_3)

```

```

# Program start

```

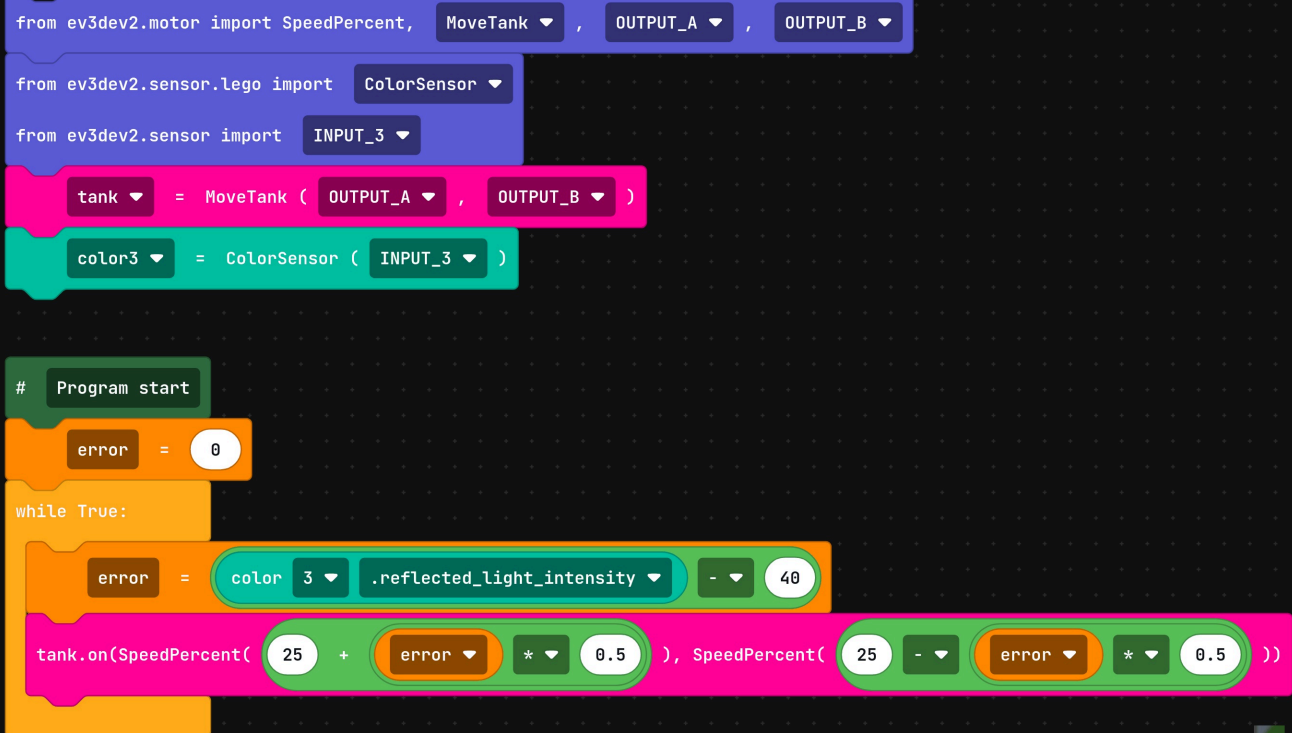
```

while True:
    if color3.reflected_light_intensity < 40:
        tank.on(SpeedPercent(10), SpeedPercent(40))
    else:
        tank.on(SpeedPercent(40), SpeedPercent(10))

```

Line follower, smoother

error is how far the reading is from the threshold. The robot turns a little when error is small and a lot when it is big. This is proportional control, the first part of PID, and it follows the line far more smoothly.



The image shows a Scratch-style block diagram for a line follower program. The code is as follows:

```
from ev3dev2.motor import SpeedPercent, MoveTank, OUTPUT_A, OUTPUT_B
from ev3dev2.sensor.lego import ColorSensor
from ev3dev2.sensor import INPUT_3

tank = MoveTank(OUTPUT_A, OUTPUT_B)
color3 = ColorSensor(INPUT_3)

# Program start
error = 0
while True:
    error = color3.reflected_light_intensity - 40
    tank.on(SpeedPercent(25 + error * 0.5), SpeedPercent(25 - error * 0.5))
```

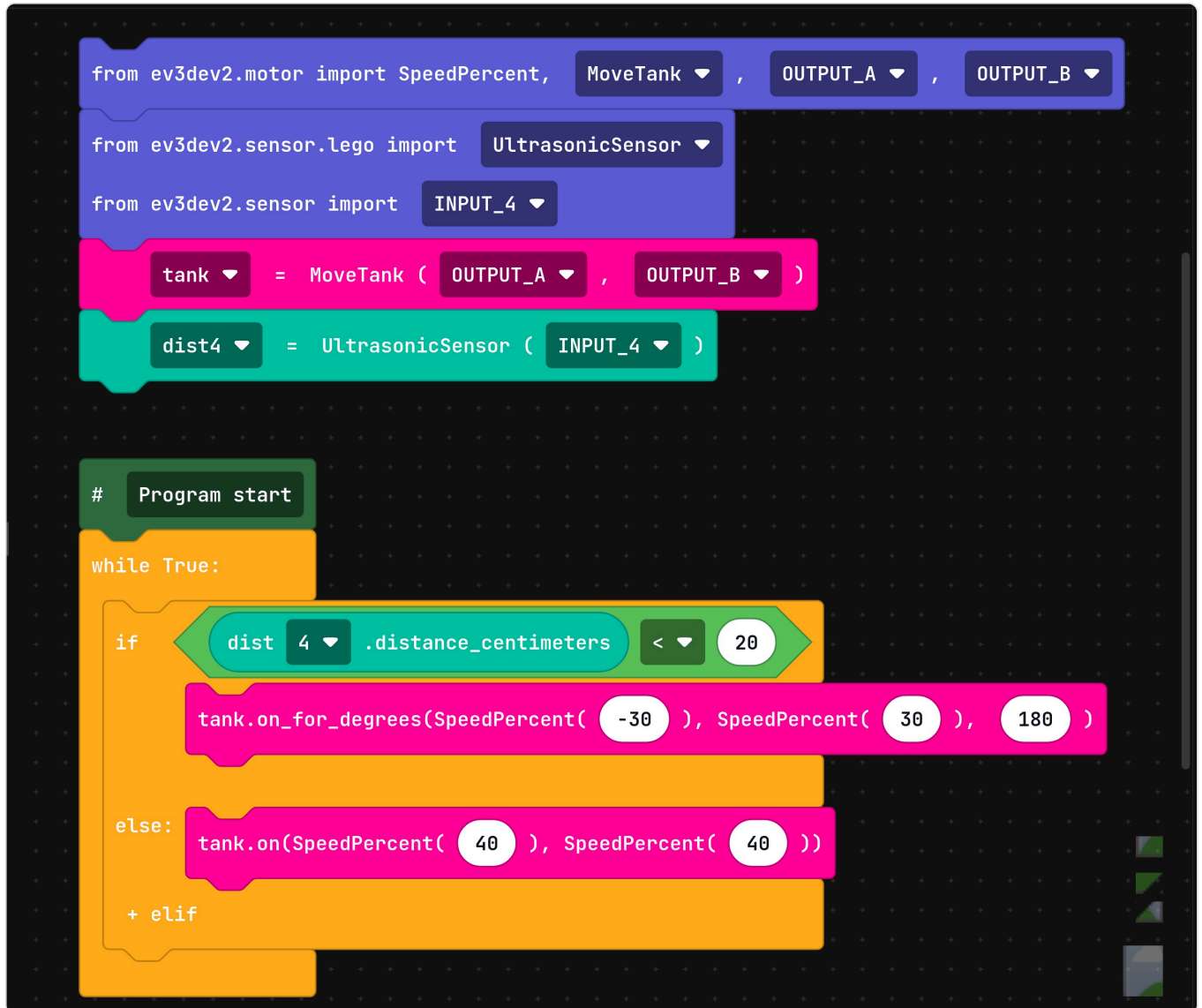
EV3, PYTHON

```
from ev3dev2.motor import SpeedPercent, MoveTank, OUTPUT_A, OUTPUT_B
from ev3dev2.sensor.lego import ColorSensor
from ev3dev2.sensor import INPUT_3
tank = MoveTank(OUTPUT_A, OUTPUT_B)
color3 = ColorSensor(INPUT_3)

# Program start
error = 0
while True:
    error = color3.reflected_light_intensity - 40
    tank.on(SpeedPercent(25 + error * 0.5), SpeedPercent(25 - (error * 0.5)))
```

Obstacle avoidance

Drives forward. When something is closer than 20 cm, it turns on the spot, then carries on.



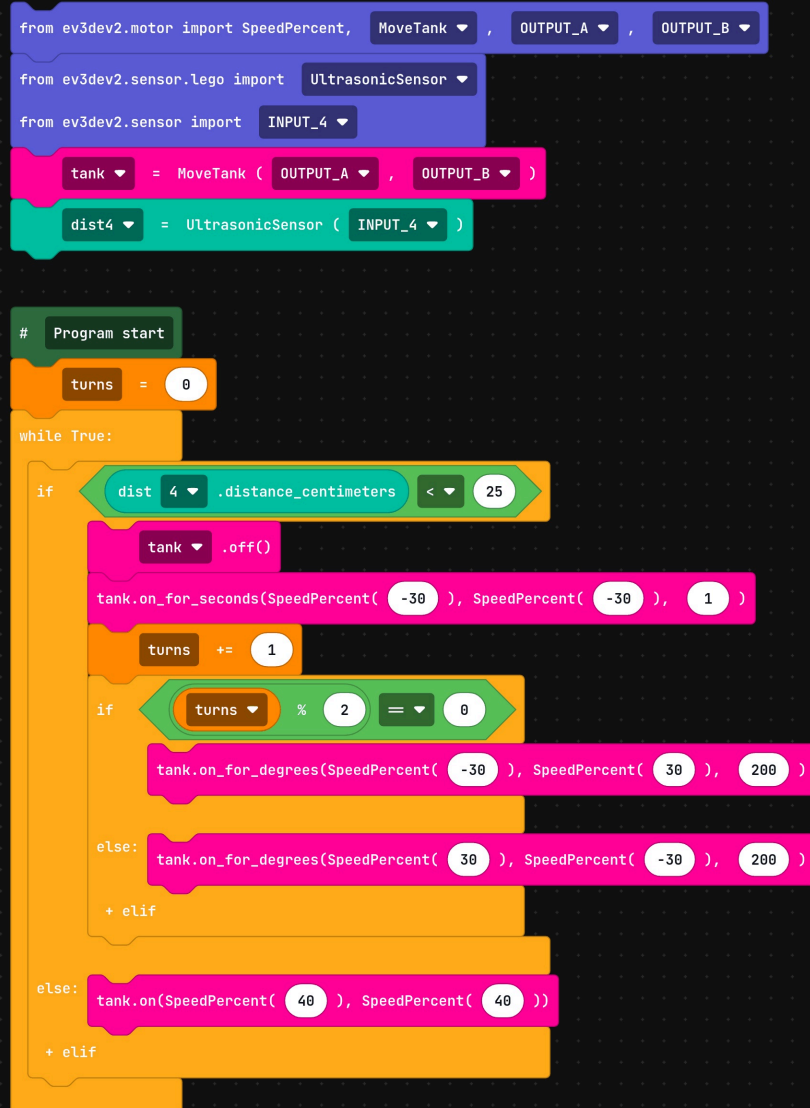
EV3, PYTHON

```
from ev3dev2.motor import SpeedPercent, MoveTank, OUTPUT_A, OUTPUT_B
from ev3dev2.sensor.lego import UltrasonicSensor
from ev3dev2.sensor import INPUT_4
tank = MoveTank(OUTPUT_A, OUTPUT_B)
dist4 = UltrasonicSensor(INPUT_4)

# Program start
while True:
    if dist4.distance_centimeters < 20:
        tank.on_for_degrees(SpeedPercent(-30), SpeedPercent(30), 180)
    else:
        tank.on(SpeedPercent(40), SpeedPercent(40))
```

Obstacle avoidance, advanced

When something is close, the robot stops, backs away, and turns. turns counts how many times it has turned, and turns % 2 makes it turn left one time and right the next, so it does not get stuck turning into the same corner.



EV3, PYTHON

```

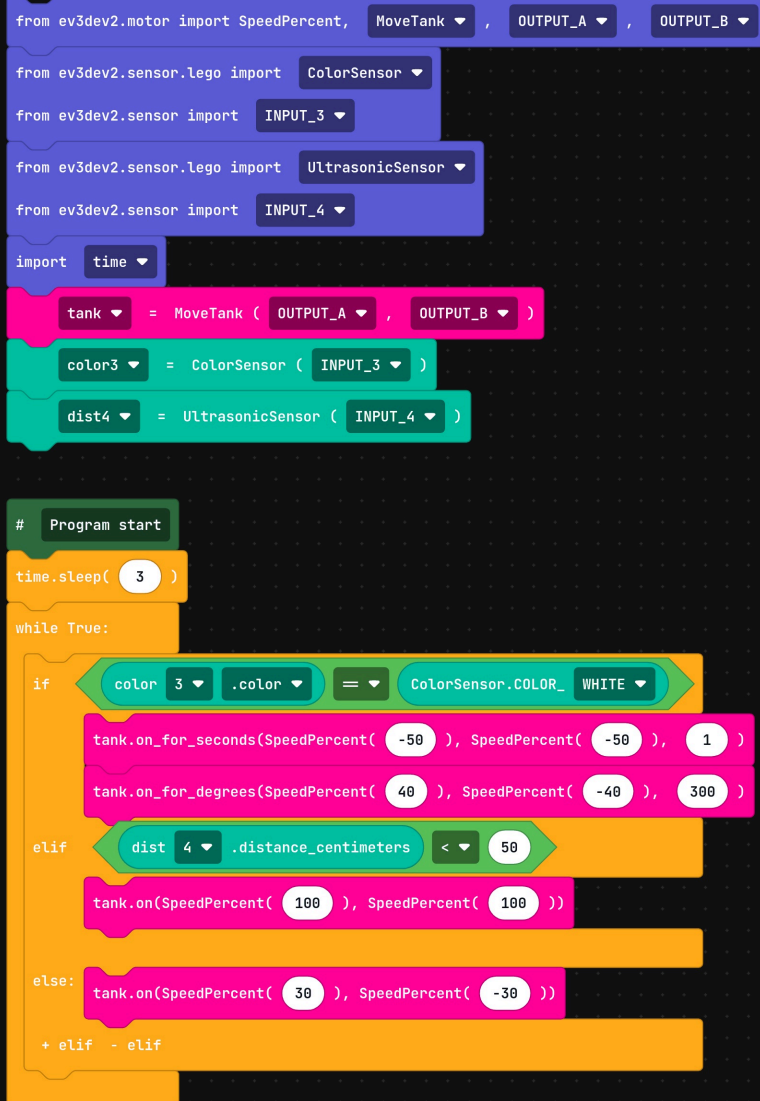
from ev3dev2.motor import SpeedPercent, MoveTank, OUTPUT_A, OUTPUT_B
from ev3dev2.sensor.lego import UltrasonicSensor
from ev3dev2.sensor import INPUT_4
tank = MoveTank(OUTPUT_A, OUTPUT_B)
dist4 = UltrasonicSensor(INPUT_4)

# Program start
turns = 0
while True:
    if dist4.distance_centimeters < 25:
        tank.off()
        tank.on_for_seconds(SpeedPercent(-30), SpeedPercent(-30), 1)
        turns += 1
        if turns % 2 == 0:
            tank.on_for_degrees(SpeedPercent(-30), SpeedPercent(30), 200)
        else:
            tank.on_for_degrees(SpeedPercent(30), SpeedPercent(-30), 200)
    else:
        tank.on(SpeedPercent(40), SpeedPercent(40))

```

Sumo

Waits 3 seconds, as the rules require. Then: if the color sensor sees the white edge of the ring, back away and turn. Otherwise, if the other robot is closer than 50 cm, push at full speed. Otherwise, spin to search for it. The order of the questions matters: staying in the ring comes first.



EV3, PYTHON

```

from ev3dev2.motor import SpeedPercent, MoveTank, OUTPUT_A, OUTPUT_B
from ev3dev2.sensor.lego import ColorSensor
from ev3dev2.sensor import INPUT_3
from ev3dev2.sensor.lego import UltrasonicSensor
from ev3dev2.sensor import INPUT_4
import time
tank = MoveTank(OUTPUT_A, OUTPUT_B)
color3 = ColorSensor(INPUT_3)
dist4 = UltrasonicSensor(INPUT_4)

# Program start
time.sleep(3)
while True:
    if color3.color == ColorSensor.COLOR_WHITE:
        tank.on_for_seconds(SpeedPercent(-50), SpeedPercent(-50), 1)
        tank.on_for_degrees(SpeedPercent(40), SpeedPercent(-40), 300)
    elif dist4.distance_centimeters < 50:
        tank.on(SpeedPercent(100), SpeedPercent(100))
    else:
        tank.on(SpeedPercent(30), SpeedPercent(-30))

```

Getting connected

How each robot connects, what it shows on itself, and what to do when something does not work.

The browser comes first

This is the most common problem there is, and it is not a StarCode limit. The app opens and works in any browser. It is the connection to the robot that does not.

ON THIS	USE THIS
Mac	Chrome, and plug the robot in with a cable
Windows	Chrome or Edge, cable or Bluetooth
Chromebook	Chrome, cable or Bluetooth
Android	Chrome, Bluetooth
iPad	Nothing connects. Every iPad browser is Safari underneath, and Safari has no cable or Bluetooth connection. A child can still build, see the code and use Learn

On a tablet

Everything is smaller on a tablet: the two top rows, the category column and the panel buttons. All fourteen categories fit down the side on an iPad held sideways, and both top rows use one button size, the way a computer does. The Full screen button gives StarCode the whole screen.

Open it from the website, not a saved copy

The app has to be opened from starcoderobot.com. A copy saved to a computer and opened as a file cannot use Bluetooth at all.

Connecting each robot

ROBOT	HOW IT CONNECTS
EV3	The cable works, including on a Mac. Bluetooth works on Android. Bluetooth on a Mac does not: Chrome lists the EV3, then refuses to open it. Download to the EV3 works over the cable
micro:bit	V2 only. Cable and Bluetooth both work
SPIKE Prime	Bluetooth, on either of two firmwares, and StarCode asks which when SPIKE Prime is picked. The LEGO SPIKE firmware is what the hub comes with: programs only, no live control from the remote control canvas. The Pybricks firmware does programs and live control. The words on the blocks change between the two, because the two firmwares are two different sets of lines
ESP32	Cable and Bluetooth both work, for live control and for running a program

Two EV3 things that waste an evening

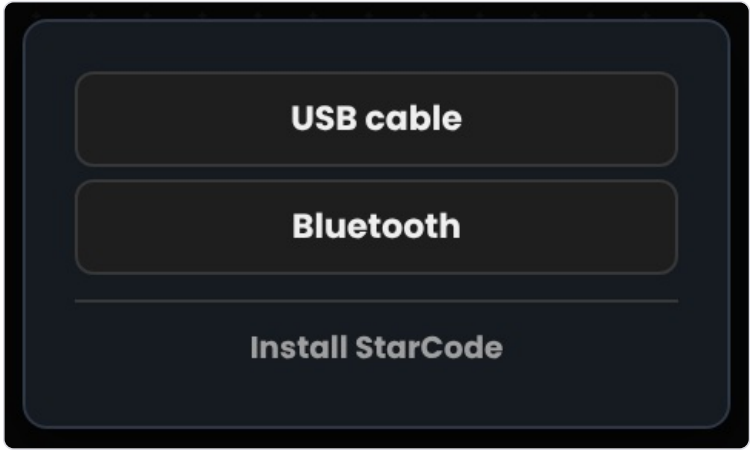
iPad mode on the EV3 must be unchecked, or it allows only one connection and offers the wrong kind. And the browser lists the same EV3 twice: the one that works is the one without "cu." in front of it. The mini USB port marked PC is the one that goes to the computer, not the square one.

What the micro:bit shows before it is set up

A board that has never been set up shows the demo it left the factory with, HELLO and a heart. One StarCode has let go of scrolls "connect to StarCode", and says it again every time the battery goes in, until Run or Download replaces it.

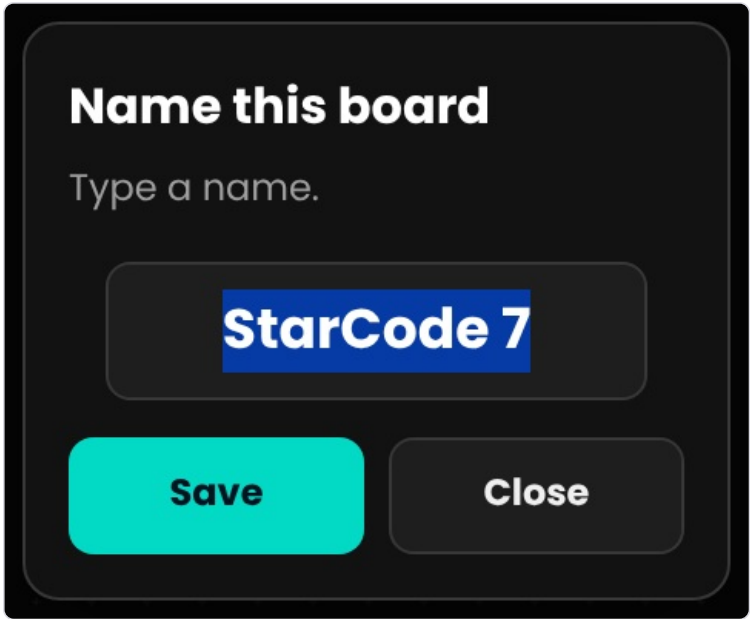
Installing from inside the app

Press Connect. Under USB cable and Bluetooth is the install line, and it goes over the USB cable. On the ESP32 it reads Install StarCode, and puts StarCode's own firmware on the board. The micro:bit has two install choices of its own: Install for USB cable puts MicroPython on it, and Install for Bluetooth puts the Bluetooth build on it. The EV3 and SPIKE Prime have no install line. The files ship with the app, so nobody opens the Arduino IDE. A bar shows how far it has got, and a board that is not an ESP32 is refused rather than written to. A board with nothing on it is noticed on connecting and offered the firmware, instead of sitting there looking connected and doing nothing. This is a grown-up job that happens once.



Naming a board, for a room with twelve of them

The Name button sits beside the robot's name in the middle row, and opens Name this board. On an ESP32, type the number on the board's sticker, 1 to 99, and it calls itself StarCode 07 in the Bluetooth list from then on. The number survives being switched off, so it is set once when the kit is unpacked. The board restarts afterwards. An EV3 can be renamed the same way, because its own firmware takes a name and broadcasts it. A micro:bit cannot be renamed by anybody, so the button shows the five letters it was given instead, the same five letters on its own screen. A SPIKE Prime shows the name it already has.



What each robot shows on itself

A robot with a light or a screen says what it is doing, and each state looks different from across a room, so a teacher with twelve robots can see which one is waiting and which one has gone wrong without walking over.

ROBOT	WHAT IT SHOWS
micro:bit	A plug when it connects by cable, the Bluetooth mark when it connects that way, then a tick. The middle light breathes while it waits. An arrow while a program goes in, a square when stopped, a cross when something went wrong
EV3	A word on the screen and a color on the buttons. READY green, SENDING orange, RUNNING orange, STOPPED green, ERROR red
SPIKE Prime	Pybricks firmware only. Green and a dash waiting, orange and S while a program goes in, green and X stopped, red and an exclamation mark when something went wrong
ESP32	Nothing, unless a light is wired to a pin

Running is the exception everywhere

Once the child's program starts, the screen and the lights belong to it. StarCode puts nothing on them until the program stops.

Stop

Stop is deliberately not something a student builds out of blocks, because a stop that runs through the program only works while the program is working. It ends the program and then switches the motors off, since ending a loop on its own leaves a motor turning at its last speed. Stop is pink, the same pink as an error line and a block that cannot work, so a child only has to learn one color for look here. On an EV3, the Escape button on the EV3 ends a downloaded program and stops the motors too.

When a robot is disconnected

When a robot is disconnected, StarCode releases it at once. The Monitor shows that the board is unplugged, Run no longer shows the program as running, and the Connect button returns to Connect. A motor or sensor disconnected from a port is removed from the Monitor when the robot's firmware reports it.

Run and Download

Run starts the program while the robot is connected to StarCode. **Download** stores the program on the robot itself, so it runs with no computer attached. If a robot cannot download yet, the Download button says so when it is pressed.

What each robot can do

The handbook says in several places that not every robot can do everything. This is which. Each number is how many blocks that robot offers in that category, and a dash means none at all. Read out of the app itself, so it cannot claim something the app does not do.

ROBOT	LANGUAGE	IMPORTS	EVENTS	MOTOR	GROUP	CONTROL	SENSORS	OPERATORS	VARIABLES	FUNCTIONS	SOUND	LIGHT	TIMING	CIRCUIT
SPIKE Prime on Pybricks firmware	Python	11	3	11	12	10	12	23	5	3	2	6	3	–
SPIKE Prime on LEGO firmware	Python	6	3	9	9	10	11	23	5	3	3	6	3	–
micro:bit	Python	3	2	–	–	10	6	23	5	3	4	5	3	5
ESP32	Python	3	3	–	–	10	–	23	5	3	–	–	3	10
ESP32	C++	5	3	–	–	11	–	22	6	3	–	–	3	10
EV3	Python	12	3	8	7	10	8	23	5	3	1	1	3	–
EV3	C++	5	3	8	–	11	6	22	6	3	1	1	3	–
EV3	Java	2	3	7	4	11	8	23	7	3	1	1	3	–
Arduino	C++	5	2	–	–	11	–	22	6	3	–	–	3	10

Which robots can be picked

Every robot StarCode knows is on the robot picker. The ones that can be picked are lit. The greyed out ones cannot be picked yet.

Choose your robot



ESP32



micro:bit



EV3



SPIKE



Arduino



VEX V5



mBot2



Raspberry Pi



Sphero



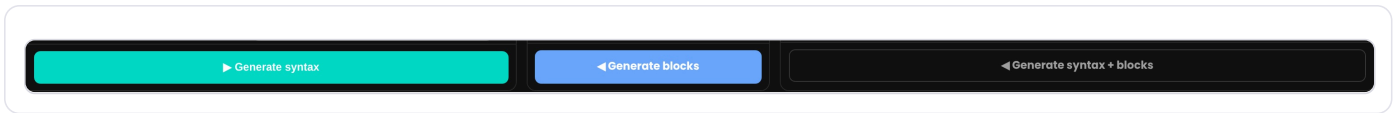
Cube

A dash is not a missing feature

A block only exists when the line it writes runs natively on that robot in that language. So a dash means that robot has no such line. The ESP32 has no Motor blocks because it has no lettered motor ports, and the EV3 has no Circuit blocks because it has no bare pins. Both are the app being honest.

The three buttons

The same on every robot. Each one says what it makes, and its arrow points the way the program travels across the screen.



BUTTON	WHERE	WHAT IT DOES
Generate syntax	under Block View	Writes your blocks out as code in the Syntax view. Free. The app does this by itself and never asks the Tutor
Generate blocks	under the Syntax view	Reads the code in the panel and builds the blocks. No Tutor at all: every block already knows the line it writes, so this is a lookup rather than a guess. It is exact, or it lists the lines it could not match
Generate syntax + blocks	under the Learn view	Turns what you typed as a question into code and blocks. It is part of Learn: the Tutor writes the program

You can type code in yourself

The Syntax view takes typing and pasting once a language is chosen. Edit the code, press Generate blocks, and the blocks follow the change. Python for now; C++ and Java use curly brackets rather than indenting, and it says so instead of guessing.

Numbers are held inside what the robot allows

Type 600 into a place that only goes to 100, press enter, and it comes back as 100. Every limit is read from the robot's own reference, so a light level on the micro:bit stops at 9 and the same idea on SPIKE Prime stops at 100.

Pasting a program in

The Syntax view takes typing and pasting from the moment a language is chosen, and an empty one says so. A paste is stripped to plain text, so nothing carries over from wherever it was copied. Edit it, press Generate blocks again, and the blocks follow the change. The import lines become their own stack on the canvas, the way a child builds them, and every stack is laid out in the order it was pasted.

IT CAN READ	EXAMPLE
A block that writes two lines	The touch sensor import
A named block of lines	<code>def my_function():</code> and everything under it
A word that comes with the name	<code>distance_sensorB</code> makes the block say UltrasonicSensor
A dropdown, by what it shows	rather than by what it holds
Brackets	Kept when writing the line back, because <code>a - b * c</code> is a different sum from <code>(a - b) * c</code>

Running a program without a computer

A program that has been downloaded is stored on the robot and runs with no computer attached. A program started with Run needs the robot to stay connected to StarCode. Part four explains Run and Download.

Why some robots have only one language

A language is offered only once it has been checked on that robot. A micro:bit can be written in C++, but StarCode cannot put C++ on it yet, so its C++ button has a dashed edge: the blocks and the code can be read, not run. A language a robot cannot run at all, like Java on a micro:bit, is not on the screen.

Words for the robot

Port

A connection on a hub, marked with a letter or a number, such as port A or port 1. You plug a motor or a sensor into it.

Not: pin

Pin

A connection on the board with a number on it. You wire your own parts to it.

Not: port

Hub

A robot with a shell, ports and buttons, such as the EV3 or SPIKE Prime. Always call it by its own name.

Not: brick, controller

Board

The ESP32 or the Arduino. A bare board with numbered pins and nothing built in.

Not: the StarCode board

Firmware

The software living inside the robot itself, underneath your program.

Not: operating system

Dialect

A version of a language made for one kind of machine. The EV3 runs full Python; the ESP32, the micro:bit and SPIKE Prime run MicroPython, a smaller Python.

The easy way to keep them straight

Ports are on a hub, marked with letters or numbers. Pins are on a board, and they are numbered.

One SPIKE Prime, three firmwares

The same LEGO hub can run three different sets of firmware, and they do not do the same things: Pybricks, the LEGO SPIKE firmware that the SPIKE app puts on, and an older LEGO firmware. StarCode works with the first two and asks which one is on the hub. The older one uses a kind of Bluetooth a browser cannot reach. And there is no such thing as a StarCode board. It is the ESP32, because that is what is printed on it.

The Tutor

Written for parents and teachers. StarCode has a helper called the StarCode Tutor. A student opens the Learn panel, types a question in ordinary words in their own language, and gets an answer. It is not a general chatbot that happens to live in a coding app: it is fenced into one subject and it stays there. It is powered by Claude, an AI service from Anthropic.

What it draws on

StarCode's Tutor is not a general chatbot pointed at robots. It draws on four separate bodies of work.

Star Kids teaching	Twelve years of Star Kids class logs, lesson plans and camp curriculum, taught to real children across after school classes and summer camps
Recorded lessons	550 recorded robotics lessons from 381 other instructors across eight platforms, each broken into four things: what the student asked for in their own words, the exact blocks that answered it, why those blocks and not others, and the mistakes students commonly make on that task. 507 of them also record which sensors a given robot does not have, so the Tutor never suggests hardware that is not on the desk
Concept inventory	About eighty ideas across programming, controls, engineering and math, each carrying its difficulty, its grade level, and what has to be taught before it
Code reference	A per robot code reference covering five boards in three languages, which is what lets the same lesson run on a different robot without being rewritten

What that adds up to

Well over 100 lessons of material, from first day sequencing and loops through sensor calibration and competition formats up to PID control, state machines, arrays, parallel programming and machine learning.

What it can teach, first day to last

The concepts run from the first day forward: sequencing, loops, if and else, variables and data types, functions, arrays, events, and debugging as a skill of its own. Then sensors and what each one can actually tell you, motors, encoders and gear ratios, parallel programs running two things at once, and combining more than one sensor to decide what to do. At the far end, closed loop control and PID, state machines, autonomous and driver controlled programming, and machine learning: object, face and audio recognition, supervised learning, and the sense, think, decide, act, learn loop that turns an AI decision into a robot actually moving.

What it can see

Every time a student writes to it, StarCode sends it two things: the code on that student's canvas, and the exact list of blocks that student's robot can run.

Why that matters in class

The Tutor is not guessing about a student's program, it is looking at it. It will never suggest a block their robot does not have, and it will never ask a student to describe what is on their screen, because it can already see it.

It helps with the wires too

Most of the time a robot that does nothing is not a coding problem. It is a wire in the wrong hole, a motor in a port the program is not using, or a part that was never working. The Tutor knows about both sides. It can walk a student through wiring an LED the right way around, the three wires on a servo and why only one of them carries instructions, which port a motor belongs in, and how to prove a part works before blaming the program.

It cannot see anything physical

It asks the student what they can see and works from the answer. It gives advice for the robot in front of that student and never another one, because the rules genuinely differ: on one robot any device goes in any port, and on another the motors and sensors have separate ports. When it has not been taught something physical, it says so and sends the student to you rather than guessing while a child is holding the hardware.

How it teaches

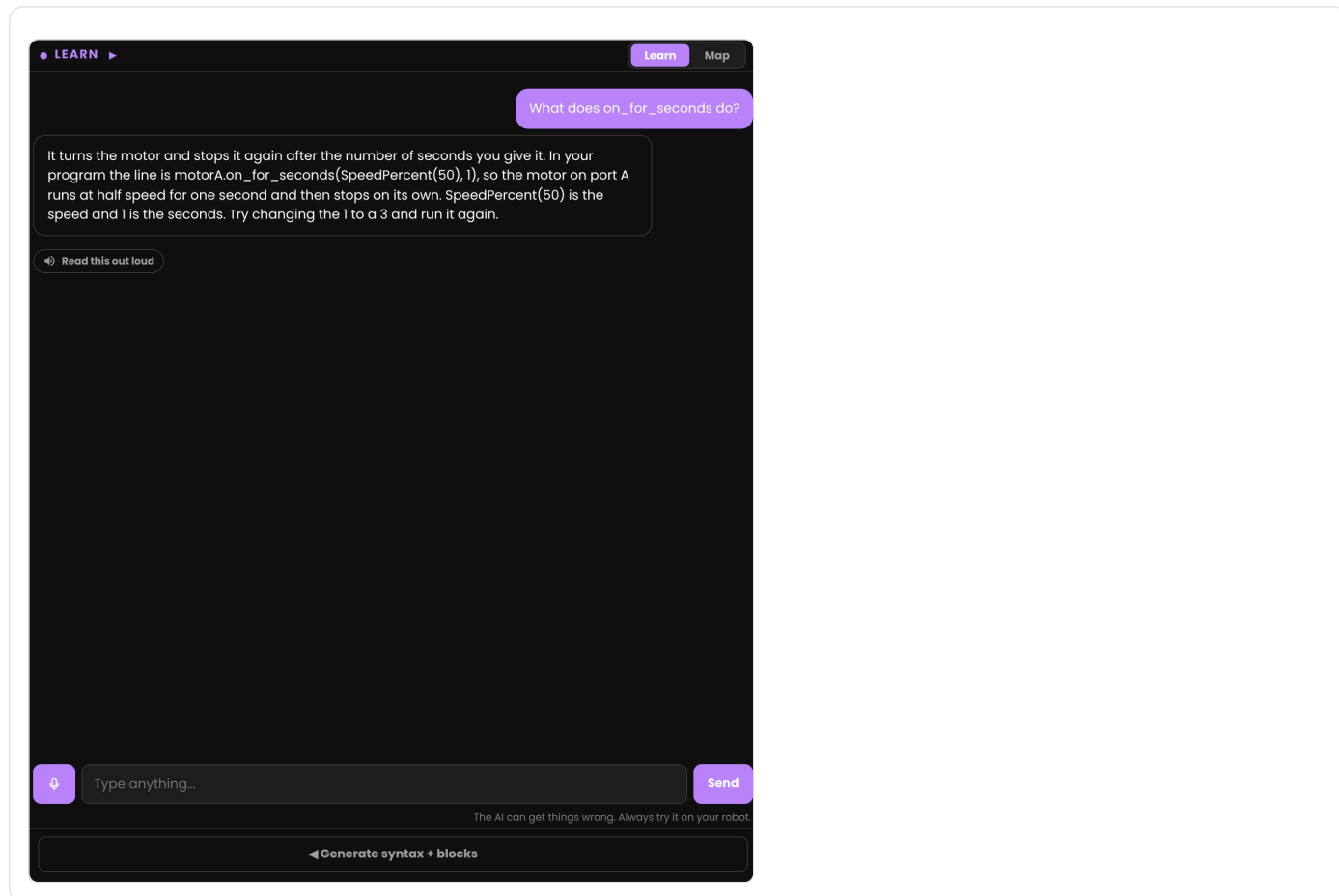
It gives a hint before it gives an answer, and it will not write a student's program for them. Its first message is always the same question, "What would you like to learn?" It does not offer a menu. If a student says they do not know what to do, it looks at what they have already

built and suggests one thing, not a list.

It teaches the line StarCode writes, not a block word. A student who says "repeat until" is told once that it is `while not` in written code, and from then on the Tutor says `while not`.

Every answer can be read out loud

Every answer has a speaker under it that reads it out loud, in the student's own language from the language menu. The Tutor is told never to write emojis, and anything that slips through is taken out before the answer is read, because the voice would read a picture out by its long name.



What it refuses

Anything that is not StarCode, the program, or the robot. Homework in other subjects, stories, games, films, music, sport, famous people, the news, jokes, personal questions about the student or about itself, shopping, and other students.

The reply is always the same shape: one short friendly line saying it only helps with coding here, then a question about what they are building. It will not answer partly, briefly, or as a joke, and it will not hint toward an answer.

Expect students to test this

A second attempt gets the same reply. The rule does not bend because a student says a teacher or parent gave permission, calls it a project, or tells the Tutor to ignore its instructions. Text hidden inside a student's own program cannot change it either. It holds.

If a student writes something worrying

The Tutor does not counsel them and does not continue the conversation. It says kindly that this is something to tell their teacher, their parent, or another trusted adult, and stops.

That hand-off is to you

Treat it the way you would treat a student saying the same thing out loud in your classroom, and follow your school's own procedure.

What you can and cannot see

You can see that a student has been using the Tutor, and roughly how much, in the progress record in the portal. You cannot see what

was said. The conversation is not stored anywhere. It exists on the student's screen while they are working and is gone when the window closes. StarCode records only that a conversation took place and when.

WHAT FOLLOWS FROM THAT	WHAT TO DO
You cannot read transcripts to find who is struggling	Use the progress record instead. It shows repeated steps, long pauses with nothing built, and programs that failed several times, which are the signs the app can honestly observe
Students may treat it as private	Tell them plainly it is not a place to say private things, even though nobody reads it. It is a coding helper, and it will send them to you if they try

The two switches

In the portal, next to each student's name, there are two switches: AI Learn and Imports button. Each is set for one student at a time. AI Learn is on when a class starts. Imports button is on for every student until a teacher turns it off.



AI Learn: turn it off and the Learn button disappears from that student's screen, so they are not tapping something that does not respond.

Imports button: purple means on, grey means off. Turn it off and that student does not see the Imports button at all, so they find the missing lines themselves. The button in the app never changes color.



Generate syntax + blocks has no switch of its own. It lives inside the conversation, so it is on when AI Learn is on and off when AI Learn is off. Even with Learn on, it stays asleep until the Tutor decides there is a good reason to offer it, so it cannot be used as a shortcut on every task.

Say this out loud to your class

Once, at the start of a course, say these three things.

The Tutor is a computer, not a person.

It only helps with coding. It will not answer anything else.

It can be wrong. If it says something, try it on your robot and see.

Why to say the third one out loud

StarCode already tells students that in their own language, in the Learn panel. Saying it yourself makes it real.

The teaching opportunity in it

There is a computer science standard, MS-PRO-RD-03, about students examining code an AI wrote and identifying its strengths and limits. The Tutor is a way to teach that directly.

How to use it that way

When a student brings you something the Tutor suggested, ask them why it would work before they try it. When the Tutor is wrong and the robot proves it wrong, that is the lesson, not a failure. A student who learns to check an AI against the real world has learned something more durable than the loop they were debugging.

Plans, codes and the portal

For parents and teachers. The app is free with the launch code and no payment at all. Two things need a plan: Learn, which is the Tutor inside the app, and the portal that comes with it, where a parent or teacher looks in and sees what a student has been doing.

What is free and what is paid

FREE	PAID, AND OPTIONAL
The blocks, the Syntax view, Map, the Monitor, every robot, every language, the remote control page, this handbook and the help page	Learn, and the portal that comes with it

Pressing Learn or Portal opens the plans panel, "Unlock Learn and the portal". The card is taken by Stripe, on Stripe's own page.

The plans

PLAN	MONTHLY	YEARLY
Teacher	\$99	\$990
Parent	\$29	\$199
Just me	\$19	\$190

Unlock Learn and the portal

Blocks, syntax, Map, every robot and the remote stay free. Learn and the portal come with a subscription plan. Add-ons are available with a subscription plan.

Monthly

Yearly

Teacher

\$99 a month
or \$990 a year

Up to 12 students
Learn for every student
The teacher portal

Plan

+ Add ons coming soon

Animatronic Robot Tutor. The tutor's voice moves a motor and a light on the robot.

Parent

\$29 a month
or \$199 a year

Up to 2 children
Learn for both
The parent portal

Plan

+ Add ons coming soon

Animatronic Robot Tutor. The tutor's voice moves a motor and a light on the robot.

Just me

\$19 a month
or \$190 a year

One person, over 18
Learn for yourself
The progress portal

Plan

+ Add ons coming soon

Animatronic Robot Tutor. The tutor's voice moves a motor and a light on the robot.

Already have a code?

Enter code

Help Terms

Teacher is for up to 12 students. Parent is for up to 2 children. Just me is for one person, over 18. AniBot is listed in the plans panel but is not for sale. Its card says "+ Add ons coming soon" and cannot be pressed.

Access codes

Paying makes a code. It is shown on screen straight after paying, and emailed from starcode@starcoderobot.com to the address used at the checkout. Type it into the app to switch the paid parts on, and into the portal to see progress.

Nobody can look a code up

A lost code is found in the emails from StarCode, or in the Stripe receipt.

StarCode Handbook v1.2.9

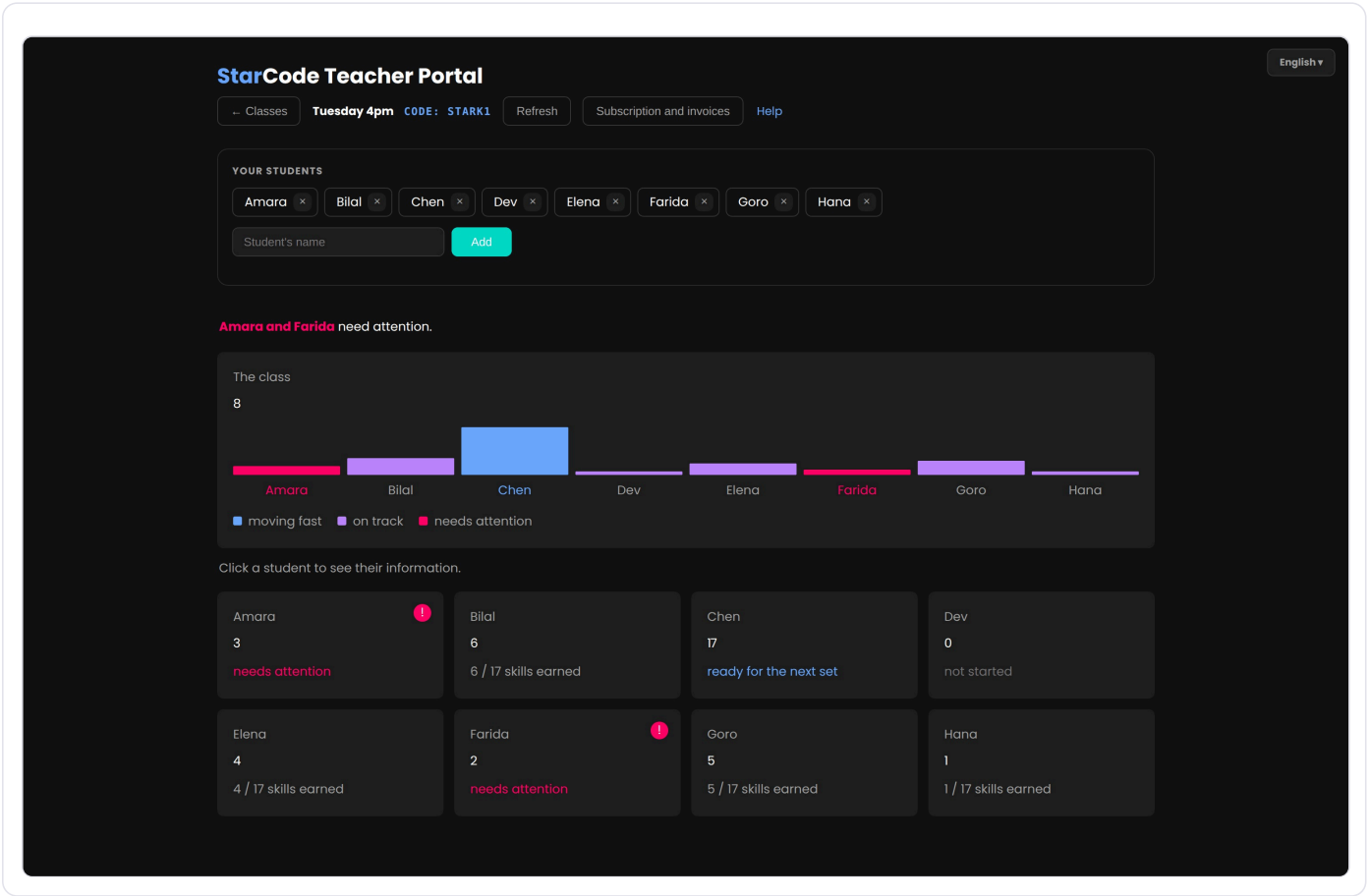
56

The portal

At starcoderobot.com/portal, type the code. Before signing in it just says Portal. After signing in it names itself for whoever bought it: Teacher Portal, Parent Portal, or My Progress.

WHAT IT SAYS	TEACHER	PARENT	JUST ME
The list	Your students	Your children	You
The big card	The class	Your children	You
The hour line	Their hour	Their hour	Your hour

Children never type their own name
The grown-up writes the list once, and the app offers those names for a child to tap. A name appears in the portal the moment it is added, before the child has coded anything.

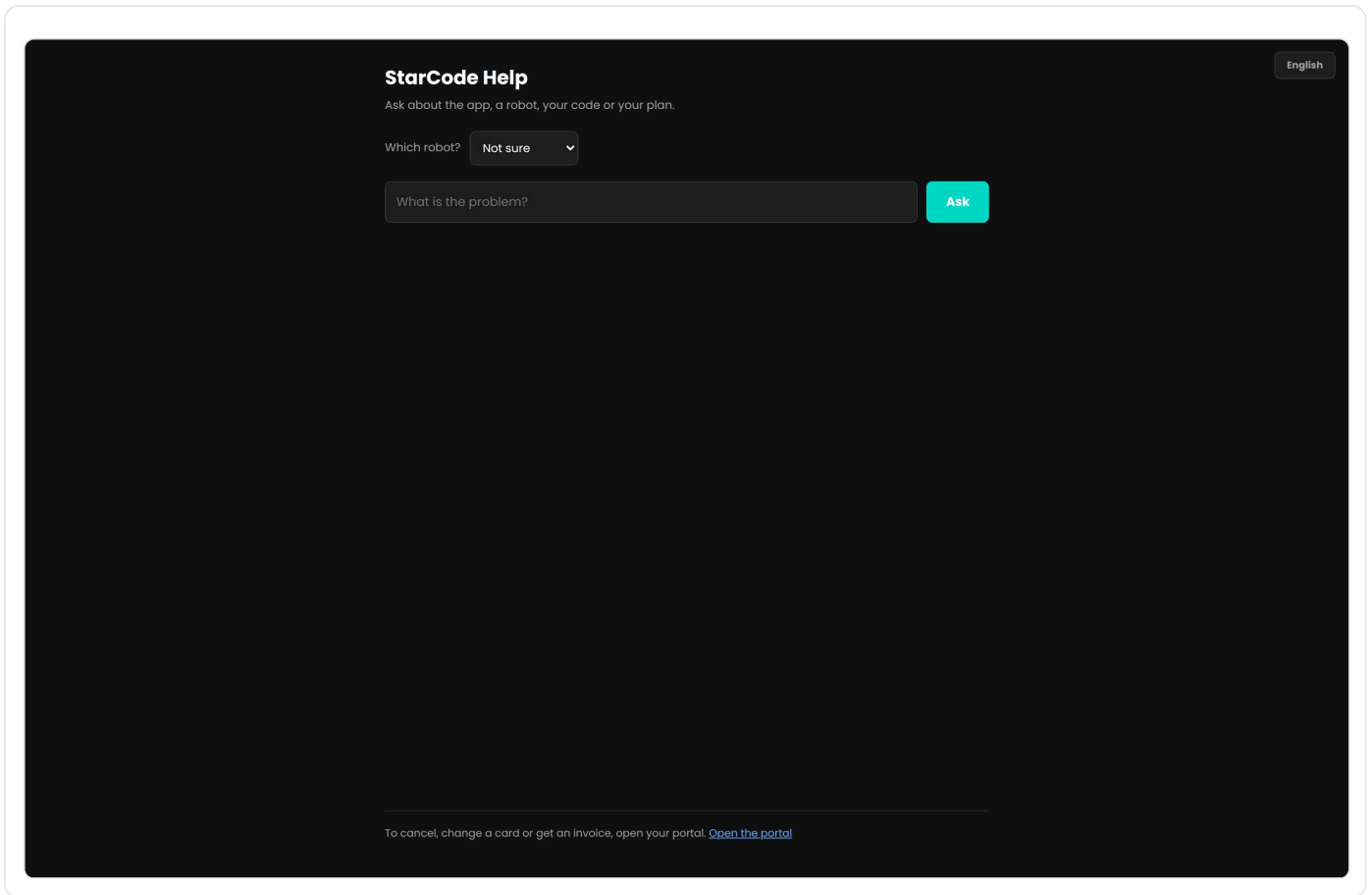


Cancelling, invoices and changing a card

In the portal, click "Subscription and invoices", at the bottom of the classes screen. That is Stripe's own page: cancel, change the card and download every invoice there. Cancelling ends at the end of the month already paid for, not on the spot.

The help page

At starcoderobot.com/help there is an AI that answers questions about StarCode, in whatever language the question is asked in. It is free, eight questions a day per person. When it cannot solve something it offers a button, "Send this to StarCode", which files the conversation as a fault report. Nothing is sent unless the button is pressed.



Where it is linked, and where it is not

From the plans panel, the portal, and the code email. There is deliberately no question mark on the coding screen, because that screen is for children. The mailbox that sends the code is not read, and the email says so.

Terms

Every word this app uses, whether or not it appears earlier in this handbook. The parts above are where you learn a word; this is where you look one up. If a word here disagrees with the app, the app is wrong and gets changed.

The screen

SAY THIS	WHAT IT MEANS	NOT
Toolbox	The whole left panel holding the categories	library, palette, sidebar
Category	One colored circle in the toolbox	library
Block canvas	Where blocks are dragged and joined	workspace, stage
Syntax view	The middle panel showing the real code	code view, output
Learn view	Where the StarCode Tutor answers	AI panel, chat
Map	The other half of the right panel, for planning before writing	flowchart, diagram
Monitor	The live readings panel	console, dashboard
Remote page	The controller view with widgets	control view

The buttons, by their exact names

Remote, Undo, Reset, Skins, Handbook, Full screen, Install and Logout across the top. Python, C++ and Java, Portal, Name, Connect, Download, Run, Stop and Save across the second row. Under the block canvas, **Generate syntax**. Under the Syntax view, **Generate blocks**. Under the Learn view, **Generate syntax + blocks**.

The skins

StarCode Dark and **StarCode Light**. Never dark mode or light mode.

The Map

A **step** is one colored rectangle. A **wire** is a line joining two steps. The step being worked on wears a thin magenta ring, deliberately a color no step can be.

How a block is named

A block is always named by the exact code on its face, such as `while True: .` Never by a description like "the loop block".

What greyed out means, and what it never means

A greyed block means the language cannot do that thing. It never means the robot cannot. A robot that cannot do something is not offered the block at all.

The three Pythons, which are not the same Python

NAME	WHERE IT RUNS
Pybricks Python	SPIKE Prime on Pybricks firmware. A version of MicroPython
MicroPython	ESP32, micro:bit, and SPIKE Prime on the LEGO SPIKE firmware. A cut down Python for small chips
Python	The full desktop Python. The EV3 on an ev3dev card runs this one

Say which one. Python on its own is only correct for the full one.

The robots, by their exact names

Never say "the hub". It means nothing on its own, because every one of these has a hub.

SAY THIS	NOT THIS
SPIKE Prime	the hub, SPIKE, the brick
SPIKE Prime on Pybricks firmware	Pybricks, the hub
SPIKE Prime on LEGO firmware	the newer firmware
EV3	the brick, Mindstorms
ESP32	the board, the StarCode board
micro:bit	microbit, the micro
Arduino	the board

Port never means a connection

A **port** is a port on the robot. On the ESP32 the same idea is a **pin**. Those are the only two meanings we use.

Chrome uses the word port for something else entirely

Chrome calls the Bluetooth or cable link a port. We never repeat that. On our screens it is a **connection**. Chrome's own list cannot be reworded, so wherever a student meets it, say plainly that Chrome means the link, not the ports on the robot.

Ways of talking to a robot

TERM	WHAT IT MEANS
Live control	The robot obeys the moment you move a slider, with no program sent. What the Remote page does
Download	Sending a whole program to the robot to be kept and run
Run	Starting the program while the robot is connected to StarCode
Direct commands	LEGO's name for the EV3's live control. One command in, the motor moves, with nothing installed on the EV3

Live control is required on every robot. That is a decision, not a preference.

The monitor sections

By their headings: **PORTS** on a robot with ports, **PINS** on the ESP32, then **REMOTE, VARIABLES**, and the robot's own line at the bottom. The headings translate with the language; the meanings do not change.

The tutor

WHERE	WHAT IT IS CALLED
On screen	StarCode Tutor
In documents	Large Language Model StarCode Tutor

It is not trained here

It is a large language model, which is a kind of AI. It is not machine learning built or trained by Star Kids. Never say "our model", and never imply it learned from Star Kids data.

The circuit parts

Each part is called what is printed on it or what it is sold as: LED, resistor, breadboard, jumper wire, servo, potentiometer. No invented names, the same rule as everywhere else.

Programming

TERM	WHAT IT MEANS
Program	A list of instructions a computer or robot carries out, in order
Code	The written instructions of a program, in a language such as Python, C++ or Java
Syntax	The rules for how a line of code must be written: the words, symbols and order a language accepts
Programming language	A set of words and rules for writing instructions a computer can follow. Python, C++ and Java are three
Algorithm	A step by step plan for solving a problem, written so a person can follow it before it becomes code
Pseudocode	An algorithm written in plain words that look like code but are not a real language
Sequencing	The order in which a program's lines run: top to bottom, one after another
Statement	One line of code that does something
Comment	A note in a program for people to read. The computer skips it. In Python it starts with #, in C++ and Java with //
Value	A single piece of information a program works with, such as the number 5, the word "hello", or True
Variable	A name that holds a value. The program can read the value and change it
Data type	The kind of value a variable holds: a whole number, a decimal number, a letter, some words, or True or False
Integer	A whole number with no decimal point, such as 7 or -3. Written int in C++ and Java
Float	A number with a decimal point, such as 0.5 or 3.14. Short for floating point
Character	One letter, digit or symbol, such as 'A'. Written char in C++ and Java
String	A piece of text: a row of characters, such as "Hello"
Boolean	A value that can only be True or False. Every yes or no question in a program gives back a boolean
Constant	A value that is set once and never changes while the program runs. Written const in C++
Operator	A symbol that does something to values: + adds, == compares, and joins two questions
Expression	Values and operators put together to give one value, such as speed + 10
Condition	A question whose answer is True or False, used to decide what happens next
Conditional statement	Code that runs only when a condition is True. if and else are conditional statements
Loop	Code that repeats. A for loop repeats a set number of times; a while loop repeats as long as a condition is True
Infinite loop	A loop that never ends, such as while True:. On a robot this is usually on purpose
Loop interrupt	Leaving a loop early. break leaves it completely; continue skips to the next time around
Function	A named group of lines that runs when it is called. Also called a method in Java
Parameter	A value a function is given to work with when it is called
Return value	The value a function sends back to whoever called it
Call	To run a function by writing its name followed by brackets
Library	A collection of code someone else wrote that a program brings in with an import
Import	The line that brings a library, or part of one, into a program. Called include in C++
Declaring	Creating a name for a variable or a part before it is used, such as motorA = LargeMotor(OUTPUT_A)
Scope	Where in a program a name can be used. A name made inside a function can only be used inside it
Namespace	A group of names kept under one heading so two things with the same name do not clash, such as std:: in C++
Enum	A short list of named values that go together, such as the colors a color sensor can report. Short for enumeration
Array	A list of values stored under one name and reached by their position, starting at 0

List	Python's name for an array: several values kept in order under one name
Index	The position of a value in an array or a list, counting from 0
Bug	A mistake in a program that makes it do something other than what was meant
Debugging	Finding and fixing bugs, often by printing values to see what the program is really doing
Print	Showing a value as text. In StarCode printed values appear in the Monitor
Input	Information going into a program, such as a sensor reading or a button press
Output	What a program sends out, such as turning a motor or lighting an LED
Event	Something that happens, such as a program starting or a button being pressed, that code can respond to
Multi-threading	Running more than one part of a program at the same time. Also called parallel programming
Object	A thing in code that holds its own values and has its own functions, such as a motor that knows its port and can be told to turn
Class	The plan an object is made from. LargeMotor is a class; motorA = LargeMotor(OUTPUT_A) makes one object from it
Object-oriented programming	Writing programs as objects that each hold their own values and functions. Often shortened to OOP
Inheritance	When one class takes all the abilities of another class and adds its own
Polymorphism	When different classes answer the same function call in their own way
Public and private	Whether a name inside a class can be used from outside it (public) or only inside it (private)
Pointer	In C++, a variable that holds the memory address of another value instead of the value itself
Reference	In C++, another name for a variable that already exists. & and * move between an address and a value
Static	A variable that keeps its value for the whole time a program runs, even between function calls
Compile	Turning code into instructions the machine runs directly. C++ and Java are compiled before they run
Compile time and run time	Compile time is when code is being turned into machine instructions; run time is when the program is actually running
Interpreter	A program that reads code and runs it line by line without compiling it first, the way Python usually runs
Firmware	The software that lives inside a robot or board, underneath your program
Dialect	A version of a language made for a particular kind of machine. MicroPython is a smaller Python made for small chips
MicroPython	A smaller version of Python made to run on small chips such as the ESP32 and the micro:bit

Robotics and control

TERM	WHAT IT MEANS
Robot	A machine that senses what is around it, decides what to do, and acts
Hub	A robot's main unit with ports, buttons and a battery, such as the EV3 or SPIKE Prime
Board	A small circuit board with a chip and numbered pins, such as the ESP32 or the micro:bit
Microcontroller	A small computer on one chip that runs one program and controls hardware
Port	A connection on a hub, marked with a letter or a number, where a motor or sensor plugs in
Pin	A numbered electrical connection on a board, used to wire your own parts
Motor	A part that turns when it is given power
Servo	A motor that turns to an exact angle and holds it there
Encoder	A sensor inside a motor that counts how far it has turned
Sensor	A part that measures something, such as light, distance, touch or turning, and sends it to the robot as input
Touch sensor	A button sensor that reports whether it is pressed
Color sensor	A sensor that reports the color it sees or how much light bounces back
Reflected light	How much of the color sensor's own light bounces back. Dark surfaces reflect little, light ones a lot
Ultrasonic sensor	A distance sensor that sends out a sound too high to hear and times the echo
Gyro sensor	A sensor that measures turning: which way the robot has turned and how fast
Threshold	The value a sensor reading is compared with to decide something, such as 40 for dark and light
Calibration	Measuring what a sensor really reads on your own table and your own lines before choosing a threshold
Drive base	Two motors that drive a robot together, one on each side. Called a drive base, a tank or a motor pair depending on the library
Tank drive	Steering by giving each side of the robot its own speed
Steering	Driving by giving one speed and how hard to turn
Degrees	How far a motor turns. 360 degrees is one full turn
Speed	How fast a motor turns, often given as a percent of its top speed
Open loop control	Moving without checking a sensor, such as driving for 2 seconds and hoping the robot went far enough
Closed loop control	Moving while checking a sensor and correcting, such as driving until the distance sensor reads 10 cm. Also called a feedback system
Feedback	Information from a sensor that tells a program how its last action worked out
Bang bang control	The simplest closed loop control: full one way when below the target, full the other way when above it
Proportional control	Correcting a little when a reading is a little off and a lot when it is far off
PID	Proportional, Integral, Derivative: a control method that tunes three numbers so a robot reaches its goal smoothly without overshooting
Motion profile	A planned path of speeds a motor follows, such as speeding up, cruising, then slowing down
State machine	A program that is always in one of a few named states, such as searching or pushing, and moves between them on rules
Lag	A delay between an input and the robot's response. Also called hysteresis when the delay depends on direction
Line follower	A robot program that keeps a robot on a line using a color or light sensor
Obstacle avoidance	A robot program that detects something in the way and drives around it
Sumo	A robot competition where two robots try to push each other out of a ring

Electronics and math

TERM	WHAT IT MEANS
Circuit	A complete loop that electricity flows around
Voltage	How hard electricity is pushed through a circuit, measured in volts
Current	How much electricity is flowing, measured in amps. Also called amperage
Resistance	How much a part slows the flow of electricity, measured in ohms
Ohm's law	Voltage equals current times resistance: $V = I \times R$
LED	A light emitting diode: a small light that only works one way around, with a long leg and a short leg
Resistor	A part that limits current, often used to protect an LED
Breadboard	A board with rows of holes for building circuits without soldering
Jumper wire	A short wire with ends that push into a breadboard or a board's pins
Potentiometer	A knob that changes resistance as it turns, often read as a number
Digital	A signal that is only on or off, 1 or 0
Analog	A signal that can be any value in a range, not just on or off
PWM	Pulse width modulation: switching a pin on and off very fast so a light looks dimmer or a motor runs slower
Wire gauge	The thickness of a wire. A bigger gauge number means a thinner wire
Torque	Turning force. More torque lets a motor push harder
Velocity	Speed in a particular direction
Acceleration	How quickly speed changes
Sine and cosine	Math functions that turn an angle into a number between -1 and 1, used to work out directions and positions
Remainder	What is left over after dividing. $7 \% 2$ is 1. The % operator is also called modulo

The rules, by name

NAME	WHAT IT SAYS
Syntax Mirror Rule	A block exists only if the code it writes runs natively in that language on that robot
Language Parity Rule	Switching language keeps the logic and makes any loss visible. A block with no equivalent stays on the canvas, marked, and out of the generated code

The Language Parity Rule is about blocks, not about the language buttons. The buttons show what a robot can run, and the reason a language is missing differs from robot to robot.

The panels, the categories and the code words

Taught in parts one to four, listed here so every word is in one place.

SAY THIS	WHAT IT MEANS	NOT
Block View	The whole left side: the toolbox and the block canvas together	
Block drawer	The blocks that slide out when a category is clicked	menu, list
Import	The line that brings in a tool before you use it. Called include in C++, and the category is called Include there too	
Library	The collection of code an import brings a tool in from. Lives in the robot's firmware, never shown on screen	the colored circles
Declaring	Creating a name for a part, such as <code>motorA = Motor(Port.A)</code>	
Statement	One line that does something	
Block, in code	A group of lines under a line ending in a colon, like a loop or an if	
Dialect	A version of a language made for one kind of machine	
Firmware	The software on the robot itself, underneath your program	operating system

Every block, by robot and language

Every block StarCode offers, robot by robot, with the exact line each one writes into the Syntax view. Read out of StarCode v11.167.0 itself, so it answers "does StarCode do this?" without opening it. Where a robot runs more than one language, or SPIKE Prime runs on either firmware, they sit side by side. [] is a space on the block for a number or another block. A number shown is the one the block starts with. A block that writes two lines shows both. Remote is not listed here, because its blocks are the same on every robot and in every language, and they are in Part two.

SPIKE Prime

Imports

PYBRICKS FIRMWARE, PYTHON

```
# import
from pybricks.pupdevices import Motor
from pybricks.parameters import Port
from pybricks.pupdevices import ColorSensor
from pybricks.parameters import Port
from pybricks.hubs import PrimeHub
from pybricks.robotics import DriveBase
from pybricks.tools import wait
from pybricks.parameters import Color
hub = PrimeHub()
motorA = Motor(Port.A)
drive_base = DriveBase(motorA, motorB, 56, 114)
color_sensorA = ColorSensor(Port.A)
```

LEGO FIRMWARE, PYTHON

```
# import
import motor
from hub import port
import motor_pair
from hub import port
import color_sensor
from hub import port
import runloop
from hub import button
```

Events

PYBRICKS FIRMWARE, PYTHON

```
# Program start
async def task1():
# comment here
```

LEGO FIRMWARE, PYTHON

```
# Program start
async def task1():
# comment here
```

Motor

PYBRICKS FIRMWARE, PYTHON

```
motorA.run(500)
motorA.run_angle(500, 360)
motorA.run_time(500, 1000)
motorA.run_target(500, 90)
motorA.stop()
motorA.angle()
motorA.reset_angle(0)
motorA.speed()
motorA.stalled()
motorA.done()
motorA.dc(100)
```

LEGO FIRMWARE, PYTHON

```
motor.run(port.A, 500)
await motor.run_for_degrees(port.A, 360, 500)
await motor.run_for_time(port.A, 1000, 500)
await motor.run_to_relative_position(port.A, 90, 500)
motor.stop(port.A)
motor.relative_position(port.A)
motor.reset_relative_position(port.A, 0)
motor.velocity(port.A)
motor.set_duty_cycle(port.A, 5000)
```

MotorGroup

PYBRICKS FIRMWARE, PYTHON

```
drive_base.drive(360, 0)
drive_base.turn(90)
drive_base.stop()
drive_base.straight(200)
drive_base.arc(100, 90)
drive_base.use_gyro(True)
drive_base.settings(200, 400, 90, 180)
drive_base.angle()
drive_base.distance()
drive_base.reset(distance=0, angle=0)
drive_base.stalled()
drive_base.done()
```

LEGO FIRMWARE, PYTHON

```
motor_pair.pair(motor_pair.PAIR_1, port.A, port.B)
motor_pair.unpair(motor_pair.PAIR_1)
motor_pair.move(motor_pair.PAIR_1, 0, velocity=360)
await motor_pair.move_for_degrees(motor_pair.PAIR_1, 360, 0,
velocity=360)
await motor_pair.move_for_time(motor_pair.PAIR_1, 1000, 0,
velocity=360)
motor_pair.move_tank(motor_pair.PAIR_1, 360, 360)
await motor_pair.move_tank_for_degrees(motor_pair.PAIR_1, 360, 360,
360)
await motor_pair.move_tank_for_time(motor_pair.PAIR_1, 360, 360,
1000)
motor_pair.stop(motor_pair.PAIR_1)
```

Control

PYBRICKS FIRMWARE, PYTHON

```
while True:
for i in range(10):
while [ ]:
while not ([ ]):
if [ ]:
if [ ]: ... else:
wait(1000)
break
continue
raise SystemExit
```

LEGO FIRMWARE, PYTHON

```
while True:
for i in range(10):
while [ ]:
while not ([ ]):
if [ ]:
if [ ]: ... else:
time.sleep_ms(1000)
break
continue
raise SystemExit
```

Sensors

PYBRICKS FIRMWARE, PYTHON

```
color_sensorA.color()
Color.RED
distance_sensorA.distance()
force_sensorA.pressed()
force_sensorA.touched()
force_sensorA.force()
hub.imu.heading()
hub.imu.acceleration(Axis.X)
hub.imu.angular_velocity(Axis.X)
hub.imu.reset_heading(0)
hub.imu.stationary()
Button.LEFT in hub.buttons.pressed()
```

LEGO FIRMWARE, PYTHON

```
color_sensor.color(port.A)
color.BLACK
distance_sensor.distance(port.A)
force_sensor.pressed(port.A)
force_sensor.force(port.A)
motion_sensor.tilt_angles()[0]
motion_sensor.acceleration(False)
motion_sensor.angular_velocity(False)
motion_sensor.reset_yaw(0)
motion_sensor.stable()
button.pressed(button.LEFT) > 0
```

Operators

PYBRICKS FIRMWARE, PYTHON

```
0 + 0
0 < 0
[ ] and [ ]
not [ ]
round(0)
abs(0)
math.floor(0)
math.ceil(0)
math.sqrt(0)
0 ** 2
0 % 1
math.sin(0)
math.cos(0)
math.tan(0)
min(0, 0)
max(0, 0)
0 <= 0 <= 100
max(min(0, 100), 0)
((0 - 0) / (100 - 0)) * (1 - 0) + 0
random.randint(0, 10)
'' + ''
len('')
''.split('')
```

LEGO FIRMWARE, PYTHON

```
0 + 0
0 < 0
[ ] and [ ]
not [ ]
round(0)
abs(0)
math.floor(0)
math.ceil(0)
math.sqrt(0)
0 ** 2
0 % 1
math.sin(0)
math.cos(0)
math.tan(0)
min(0, 0)
max(0, 0)
0 <= 0 <= 100
max(min(0, 100), 0)
((0 - 0) / (100 - 0)) * (1 - 0) + 0
random.randint(0, 10)
'' + ''
len('')
''.split('')
```

Variables

PYBRICKS FIRMWARE, PYTHON

```
x = 0
x += 1
x
True
print('x =', x)
```

LEGO FIRMWARE, PYTHON

```
x = 0
x += 1
x
True
print('x =', x)
```

Functions

PYBRICKS FIRMWARE, PYTHON

```
def my_function():
my_function()
return 0
```

LEGO FIRMWARE, PYTHON

```
def my_function():
my_function()
return 0
```

Sound

PYBRICKS FIRMWARE, PYTHON

```
hub.speaker.beep(440, 500)
hub.speaker.volume(100)
```

LEGO FIRMWARE, PYTHON

```
await sound.beep(440, 500, 100)
sound.stop()
sound.volume(100)
```

Light

PYBRICKS FIRMWARE, PYTHON

```
hub.light.on(Color.RED)
hub.display.pixel(0, 0, 50)
hub.display.icon(Matrix([[50]*5]*5))
hub.display.icon(Icon.HAPPY)
hub.display.text('')
hub.display.off()
```

LEGO FIRMWARE, PYTHON

```
light.color(light.POWER, color.RED)
light_matrix.set_pixel(0, 0, 50)
light_matrix.show([50] * 25)
light_matrix.show_image(light_matrix.IMAGE_HEART)
light_matrix.write('')
light_matrix.clear()
```

Timing

PYBRICKS FIRMWARE, PYTHON

```
_timer = Stopwatch()  
_timer.time()  
_timer.reset()
```

LEGO FIRMWARE, PYTHON

```
_timer_start = time.ticks_ms()  
time.ticks_diff(time.ticks_ms(), _timer_start)  
_timer_start = time.ticks_ms()
```

micro:bit

Imports

PYTHON

```
# import
from microbit import *
import music
```

Events

PYTHON

```
# Program start
# comment here
```

Control

PYTHON

```
while True:
    for i in range(10):
        while [ ]:
            while not ([ ]):
                if [ ]:
                    if [ ]: ... else:
                        sleep(1000)
                        break
                continue
            raise SystemExit
```

Sensors

PYTHON

```
compass.heading()
accelerometer.get_x()
accelerometer.is_gesture('shake')
microphone.was_event(SoundEvent.LOUD)
button_a.is_pressed()
button_a.get_presses()
```

Operators

PYTHON

```
0 + 0
0 < 0
[ ] and [ ]
not [ ]
round(0)
abs(0)
math.floor(0)
math.ceil(0)
math.sqrt(0)
0 ** 2
0 % 1
math.sin(0)
math.cos(0)
math.tan(0)
min(0, 0)
max(0, 0)
0 <= 0 <= 100
max(min(0, 100), 0)
((0 - 0) / (100 - 0)) * (1 - 0) + 0
random.randint(0, 10)
'' + ''
len('')
''.split('')
```

Variables

PYTHON

```
x = 0
x += 1
x
True
print('x =', x)
```

Functions

PYTHON

```
def my_function():
my_function()
return 0
```

Sound

PYTHON

```
music.pitch(440, 500)
music.play(music.DADADADUM, wait=False, loop=True)
music.stop()
set_volume(100)
```

Light

PYTHON

```
display.set_pixel(0, 0, 9)
display.show(Image.HEART)
display.show(Image('00000:00000:00000:00000:00000'))
display.scroll('', delay=150)
display.clear()
```

Timing

PYTHON

```
_timer_start = running_time()
(running_time() - _timer_start)
_timer_start = running_time()
```

Circuit

PYTHON

```
pin0.write_digital(1)
pin0.read_digital()
pin0.is_touched()
pin0.write_analog(511)
pin0.read_analog()
```

Imports, called Include in C++

PYTHON	C++
<pre># import from machine import Pin import time</pre>	<pre>// include #include <ESP32Servo.h> const int motorA = 13; const int dirA = 25; Servo servo13;</pre>

Events

PYTHON	C++
<pre># Program start async def task1(): # comment here</pre>	<pre>void setup() { void task1(void *p) { // comment here</pre>

Control

PYTHON	C++
<pre>while True: for i in range(10): while []: while not ([]): if []: if []: ... else: time.sleep_ms(1000) break continue raise SystemExit</pre>	<pre>while(true) { for(int i=0;i<10;i++) { while([]) { while(!([])) { if([]) { if([]) { ... } else { delay(1000); break; continue; switch(x) { return;</pre>

Operators

PYTHON	C++
<pre>0 + 0 0 < 0 [] and [] not [] round(0) abs(0) math.floor(0) math.ceil(0) math.sqrt(0) 0 ** 2 0 % 1 math.sin(0) math.cos(0) math.tan(0) min(0, 0) max(0, 0) 0 <= 0 <= 100 max(min(0, 100), 0) ((0 - 0) / (100 - 0)) * (1 - 0) + 0 random.randint(0, 10) '' + '' len('') ''.split('')</pre>	<pre>0 + 0 0 < 0 [] && [] ![] round(0) abs(0) floor(0) ceil(0) sqrt(0) pow(0, 2) 0 % 1 sin(0) cos(0) tan(0) min(0, 0) max(0, 0) 0 >= 0 && 0 <= 100 constrain(0, 0, 100) map(0, 0, 0, 0, 0) random(0, 10) "" + "" "".length()</pre>

Variables

PYTHON

```
x = 0
x += 1
x
True
print('x =', x)
```

C++

```
int x = 0;
x += 1;
x++;
x--;
x
Serial.print("x = "); Serial.println(x);
```

Functions

PYTHON

```
def my_function():
my_function()
return 0
```

C++

```
void my_function() {
my_function();
return 0;
```

Timing

PYTHON

```
_timer_start = time.time()
time.time() - _timer_start
_timer_start = time.time()
```

C++

```
unsigned long _timer_start = millis();
(millis() - _timer_start)
_timer_start = millis();
```

Circuit

PYTHON

```
Pin(13, Pin.OUT)
Pin(13).value(1)
Pin(13).value()
Pin(13).value() == 1
PWM(Pin(13)).duty(512)
ADC(Pin(13)).read()
PWM(Pin(13), freq=50).duty_ns(500000 + 90 * 11111)
buzzer = PWM(Pin(13), freq=440)
buzzer.deinit()
time_pulse_us(Pin(14), 1)
```

C++

```
pinMode(13, OUTPUT);
digitalWrite(13, HIGH);
digitalRead(13)
digitalRead(13) == HIGH
analogWrite(13, 128);
analogRead(A0)
servo13.write(90);
tone(13, 440, 500);
noTone(13);
pulseIn(14, HIGH)
```

EV3

Imports, called Include in C++

PYTHON	C++	JAVA
<pre># import from ev3dev2.motor import SpeedPercent, LargeMotor, OUTPUT_A from ev3dev2.motor import SpeedPercent, MoveTank, OUTPUT_A, OUTPUT_B from ev3dev2.sensor.lego import TouchSensor from ev3dev2.sensor import INPUT_1 from ev3dev2.sound import Sound import time motorA = LargeMotor(OUTPUT_A) tank = MoveTank(OUTPUT_A, OUTPUT_B) touch1 = TouchSensor(INPUT_1) btn = Button() spkr = Sound() leds = Leds()</pre>	<pre>// include #include "ev3dev.h" using namespace ev3dev; large_motor motorA(OUTPUT_A); touch_sensor touch1(INPUT_1);</pre>	<pre>// import import lejos.hardware.motor.Motor;</pre>

Events

PYTHON	C++	JAVA
<pre># Program start async def task1(): # comment here</pre>	<pre>int main() { void task1() { // comment here</pre>	<pre>public static void main(String[] args) throws Exception { Thread task1 = new Thread(() -> { // comment here</pre>

Motor

PYTHON	C++	JAVA
<pre>motorA.on(SpeedPercent(50)) motorA.on_for_degrees(SpeedPercent(50), 360) motorA.on_for_seconds(SpeedPercent(50), 1) motorA.stop() motorA.position motorA.position = 0 motorA.is_stalled motorA.is_running</pre>	<pre>motorA.set_speed_sp(500); motorA.run_forever(); motorA.set_speed_sp(500); motorA.set_time_sp(1000); motorA.run_timed(); motorA.stop(); motorA.set_speed_sp(500); motorA.set_position_sp(360); motorA.run_to_rel_pos(); motorA.position() motorA.set_position(0); motorA.state().count("stalled") motorA.state().count("running")</pre>	<pre>Motor.A.setSpeed(450); Motor.A.forward(); Motor.A.setSpeed(450); Motor.A.rotate(360); Motor.A.setSpeed(450); Motor.A.forward(); Thread.sleep(1000); Motor.A.stop(); Motor.A.getTachoCount() Motor.A.resetTachoCount(); Motor.A.isStalled()</pre>

MotorGroup

PYTHON	C++	JAVA
<pre>steer.on(0, SpeedPercent(50)) steer.on_for_degrees(0, SpeedPercent(50), 360) steer.on_for_seconds(0, SpeedPercent(50), 1) tank.on(SpeedPercent(50), SpeedPercent(50)) tank.on_for_degrees(SpeedPercent(50), SpeedPercent(50), 360) tank.on_for_seconds(SpeedPercent(50), SpeedPercent(50), 1) tank.off()</pre>	<pre>-</pre>	<pre>pilot.setLinearSpeed(360); pilot.forward(); pilot.setLinearSpeed(360); pilot.travel(50); pilot.setLinearSpeed(360); pilot.forward(); Thread.sleep(1000); pilot.stop();</pre>

Control

PYTHON	C++	JAVA
<pre>while True: for i in range(10): while []: while not ([]): if []: if []: ... else: time.sleep(1) break continue raise SystemExit</pre>	<pre>while(true) { for(int i=0;i<10;i++) { while([]) { while(!([])) { if([]) { if([]) { ... } else { std::this_thread::sleep_for(std::chrono::mil liseconds(1000)); } break; } continue; } switch(x) { return 0; } } } }</pre>	<pre>while(true) { for(int i=0;i<10;i++) { while([]) { while(!([])) { if([]) { if([]) { ... } else { Thread.sleep(1000); } break; } continue; } switch(x) { System.exit(0); } } } }</pre>

Sensors

PYTHON	C++	JAVA
<pre>color1.color ColorSensor.COLOR_NOCOLOR dist1.distance_centimeters touch1.is_pressed gyro1.angle gyro1.rate gyro1.reset() btn.up</pre>	<pre>color1.color() dist1.distance_centimeters() touch1.is_pressed() gyro1.angle() gyro1.rate() button::up.pressed()</pre>	<pre>new EV3ColorSensor(SensorPort.S1).getColorID() Color.RED new EV3UltrasonicSensor(SensorPort.S1).getDistan ceMode().fetchSample(sample, 0) new EV3TouchSensor(SensorPort.S1).isPressed() new EV3GyroSensor(SensorPort.S1).getAngleMode(). fetchSample(sample, 0) new EV3GyroSensor(SensorPort.S1).getRateMode().f etchSample(sample, 0) new EV3GyroSensor(SensorPort.S1).reset(); Button.LEFT.isDown()</pre>

Operators

PYTHON	C++	JAVA
<pre>0 + 0 0 < 0 [] and [] not [] round(0) abs(0) math.floor(0) math.ceil(0) math.sqrt(0) 0 ** 2 0 % 1 math.sin(0) math.cos(0) math.tan(0) min(0, 0) max(0, 0) 0 <= 0 <= 100 max(min(0, 100), 0) ((0 - 0) / (100 - 0)) * (1 - 0) + 0 random.randint(0, 10) '' + '' len('') ''.split('')</pre>	<pre>0 + 0 0 < 0 [] && [] ! [] round(0) abs(0) floor(0) ceil(0) sqrt(0) pow(0, 2) 0 % 1 sin(0) cos(0) tan(0) min(0, 0) max(0, 0) 0 >= 0 && 0 <= 100 std::max(0, std::min(100, 0)) ((0 - 0) / (100 - 0)) * (0 - 0) + 0 rand() % (10 - 0 + 1) + 0 "" + "" "".length() "".split("")</pre>	<pre>0 + 0 0 < 0 [] && [] ! [] Math.round(0) Math.abs(0) Math.floor(0) Math.ceil(0) Math.sqrt(0) Math.pow(0, 2) 0 % 1 Math.sin(0) Math.cos(0) Math.tan(0) Math.min(0, 0) Math.max(0, 0) 0 >= 0 && 0 <= 100 Math.max(Math.min(0, 100), 0) ((0 - 0) / (100 - 0)) * (1 - 0) + 0 (int)(Math.random() * (10 - 0)) + 0 "" + "" "".length() "".split("")</pre>

Variables

PYTHON	C++	JAVA
<pre>x = 0 x += 1 x True print('x =', x)</pre>	<pre>int x = 0; x += 1; x++; x--; x std::cout << "x = " << x << std::endl;</pre>	<pre>int x = 0; x += 1; x++; x--; x true System.out.println("x = " + x);</pre>

Functions

PYTHON	C++	JAVA
<pre>def my_function(): my_function() return 0</pre>	<pre>void my_function() { my_function(); return 0;</pre>	<pre>void my_function() { my_function(); return 0;</pre>

Sound

PYTHON	C++	JAVA
<pre>spkr.play_tone(440, 1)</pre>	<pre>sound::tone(440, 500);</pre>	<pre>Sound.playTone(440, 500);</pre>

Light

PYTHON	C++	JAVA
<pre>leds.set_color('LEFT', 'RED')</pre>	<pre>led::set_color(led::left, led::red);</pre>	<pre>Button.LEDPattern(0);</pre>

Timing

PYTHON	C++	JAVA
<pre>_timer_start = time.time() (time.time() - _timer_start) _timer_start = time.time()</pre>	<pre>auto _timer_start = std::chrono::steady_clock::now(); std::chrono::duration<double> (std::chrono::steady_clock::now() - _timer_start).count() _timer_start = std::chrono::steady_clock::now();</pre>	<pre>long _timer_start = System.currentTimeMillis(); (System.currentTimeMillis() - _timer_start) _timer_start = System.currentTimeMillis();</pre>

Arduino

Imports, called Include in C++

C++

```
// include
#include <Servo.h>
const int motorA = 13;
const int dirA = 25;
Servo servo13;
```

Events

C++

```
void setup() {
// comment here
```

Control

C++

```
while(true) {
for(int i=0;i<10;i++) {
while([ ]) {
while(!([ ])) {
if([ ]) {
if([ ]) { ... } else {
delay(1000);
break;
continue;
switch(x) {
return;
```

Operators

C++

```
0 + 0
0 < 0
[ ] && [ ]
![ ]
round(0)
abs(0)
floor(0)
ceil(0)
sqrt(0)
pow(0, 2)
0 % 1
sin(0)
cos(0)
tan(0)
min(0, 0)
max(0, 0)
0 >= 0 && 0 <= 100
constrain(0, 0, 100)
map(0, 0, 0, 0, 0)
random(0, 10)
"" + ""
"".length()
```

Variables

C++

```
int x = 0;
x += 1;
x++;
x--;
x
Serial.print("x = "); Serial.println(x);
```

Functions

C++

```
void my_function() {
my_function();
return 0;
```

Timing

C++

```
unsigned long _timer_start = millis();
(millis() - _timer_start)
_timer_start = millis();
```

Circuit

C++

```
pinMode(13, OUTPUT);
digitalWrite(13, HIGH);
digitalRead(13)
digitalRead(13) == HIGH
analogWrite(13, 128);
analogRead(A0)
servo13.write(90);
tone(13, 440, 500);
noTone(13);
pulseIn(14, HIGH)
```
