

Section 1: Introduction

Introduction

Have you ever wondered how computers and robots know what to do?

They don't think for themselves – they follow instructions. Programming is how we give those instructions, in a special language, that computers and robots understand. Today, you are going to learn one of those languages: **Python**.

In this workshop, you'll use **Python** to control a **LEGO SPIKE Prime robot**. You'll make it move, light up, play sounds, and react to its surroundings.

What is Python?

Python is a **text-based programming language**.

It is easy to read and write.

It is used by **both beginners and professionals** to:

- build games, applications, and websites.
- study data and create smart tools using machine learning
- automate boring or repetitive tasks, making work quicker and easier
- control **robots** (like LEGO SPIKE Prime)

Setting up the SPIKE Prime Code Editor

The Code Editor is where we write our Python programs. When we press **Play**, the robot reads our code and follows those instructions.

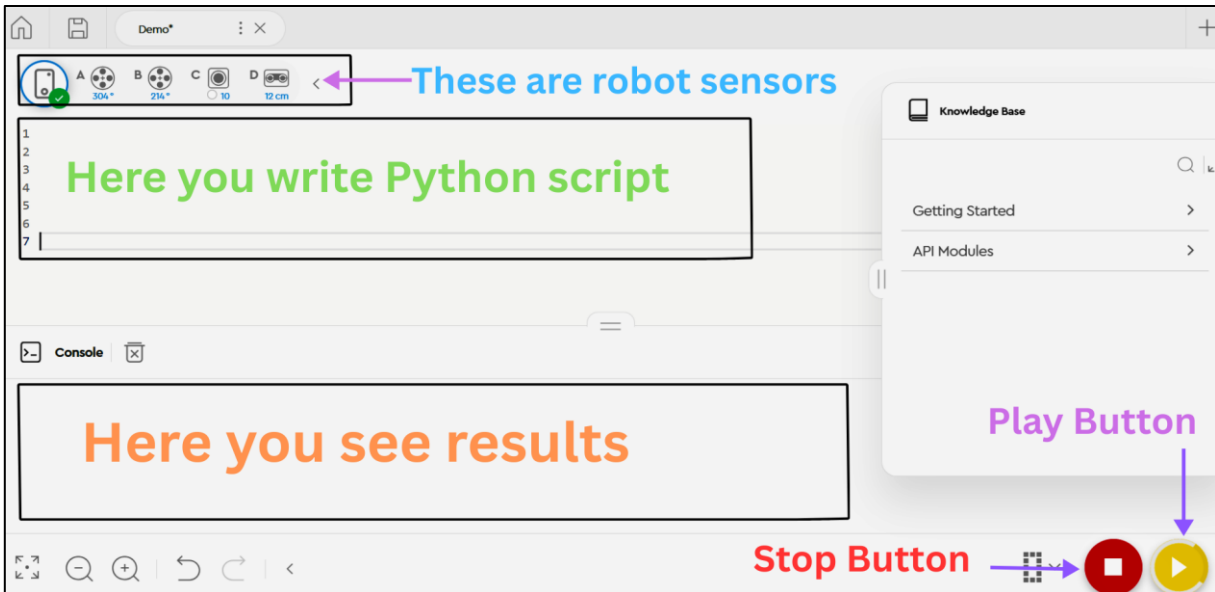
Open the SPIKE Prime webpage at <https://spike.legoeducation.com/prime/lobby/>

Click the following: New project → Python → Create.

Next to the save icon at the top of the screen is the file name. Click the three dots next to it and choose Rename Project. Name your file demo.py (all Python files end in .py).

The screen is split into two main sections:

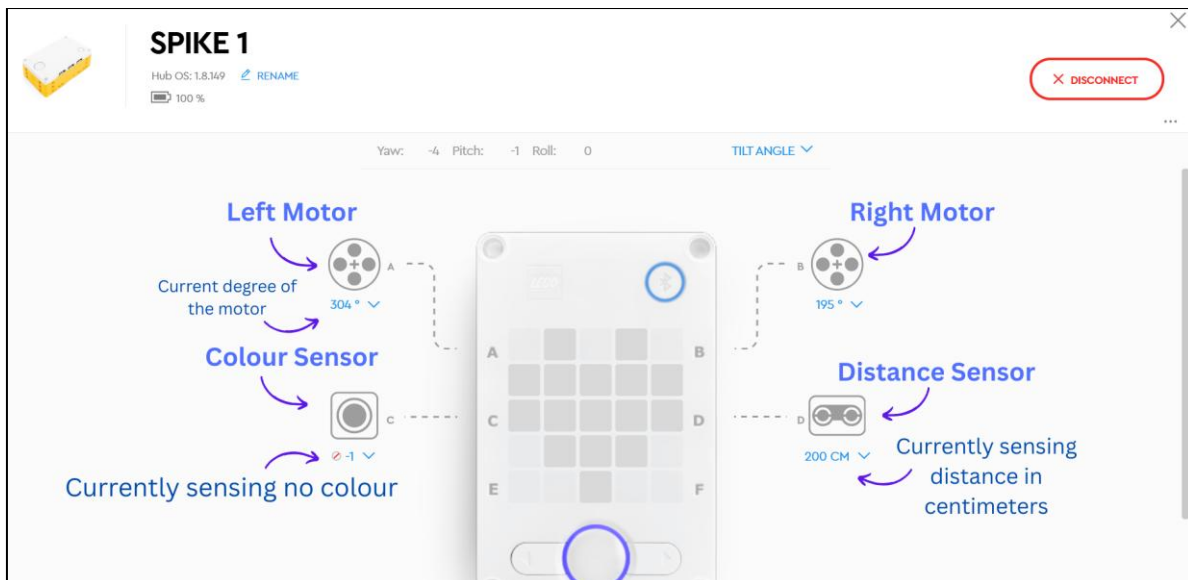
- The **top section** is where you write your code.
- The **bottom section (Console)** shows messages and errors.



Ports and Sensors

Sensors allow the robot to detect and react to the world around it.
Each part of the robot plugs into a lettered **port**:

- **A** – Left Motor
- **B** – Right Motor
- **C** – Colour Sensor
- **D** – Distance Sensor



Section 2: Your First Program: “Hello LEGO”

When learning a new programming language, you often start with the “Hello World” program. This is a simple check that everything is working.

👉 Carefully type this into `demo.py`, then press the yellow *Play* button:

```
from hub import light_matrix  
  
light_matrix.write('HELLO LEGO')
```

What will happen when you run it?

The hub (screen) will display the text: HELLO LEGO

How does the code work?

```
from hub import light_matrix
```

`hub` is a **module** and `light_matrix` is an **object**.

The hub is like a toolbox. Inside there are different tools we can use to control robots. One of those tools is called **light_matrix**, which lets us control the hub screen on top of the robot

```
light_matrix.write('HELLO LEGO')
```

`.write()` is a function (**method**) of the `light_matrix` tool (**object**).

It is used to display text on the hub screen. Whatever you put inside the quotes ' ' or " " will appear on the hub screen.

💡 **Tip 1:** You can change how **bright** the letters appear on the Light Matrix or how **fast** each character shows up! Adding two numbers after your message separated by commas allows you to set these values, eg: `light_matrix.write('HELLO', 80, 400)`

80 = intensity (0 = dim, 100 = bright) | 400 = time per character in milliseconds (smaller = faster)

Comments


Sometimes, you want to **write notes in your code** to explain what it does. They **do not affect** how the program runs. A comment **start with the # (hashtag) symbol**.

 Add the comments to your code, then press the yellow *Play* button:

```
# This line in green is a comment. It explains what the code does
from hub import light_matrix # from hub toolbox get light_matrix tool

light_matrix.write("HELLO LEGO!") # this line prints a message on the hub
```

Did anything happen? No — nothing changed! That's because **comments do not run**.

 **Tip 2:** In this activity you don't need to type any of the comments you see in green. They are just to explain what the code does.

Exercise 1

1. Can you make the robot greet you by your name?

Challenge: Can you make the message brighter and faster? (*Hint:* use intensity and time_per_character — see Tip 1.)

Section 3: Functions

In the last example, you used `.write()` to make the robot show text. But what if you want the robot to **do something again and again** without typing the same code each time?

That's what a **function** is!

A **function** is a reusable block of code that performs a specific task. You **define** a function using the word `def`, followed by:

- the **function name**
- brackets `()`
- a colon `:`

The code that belongs to the function is written underneath it and **indented** (moved slightly to the right) by pressing TAB. This tells Python that those lines are part of the function.

You play the function by writing its name with brackets, for example: `function_name()`

Let's make a simple function that says "Hi!"

 Carefully type the following code, then press the yellow *Play* button:

```
# Function

from hub import light_matrix # from hub toolbox get light_matrix tool

def greet(): # define a function
    light_matrix.write('HI!') # print a message

greet() # run the function
```

What will happen when you run it?

The hub screen will display the message: HI!

How does the code work?

```
def greet():
```


- `def` creates a function called `greet`.

```
    light_matrix.write("Hi!")
```

- This instruction is stored inside the function. It tells the robot to display **Hi!** on the hub screen. Note the indentation!

```
greet()
```

- This is when the function runs. Python goes to the saved instructions inside the **greet** function and executes them, so the robot shows the message **Hi!**

 **Tip 3:** `greet()` can be used as many times as you want, and the robot will say "Hi!" each time. The next example demonstrates this.

🔧 Carefully update your code with the following lines, then press the yellow *Play* button:

```
from hub import light_matrix # from hub toolbox get light_matrix tool
import time # to pause

def greet(): # define a function
    light_matrix.write('HI!') # print a message

greet() # run the function
time.sleep(1) # pause for one second
greet()
```

What happens when you run it?

Hub displays HI! → one second pause → HI! again

How does the code work?

```
import time
```

- Time is a module, it has tools to control timing in our program.

```
time.sleep(1)
```

- .sleep() is a function (**method**) of the time module. It pauses the program for 1 seconds before the next task plays.

💡 **Tip 4:** You can increase or decrease the value of time.sleep() to make pause longer or shorter.

Add a Parameter

Sometimes we want the robot to display **different names** each time.

- We can do this by adding a **parameter** to the function.

- A **parameter** is like a “blank space” or **placeholder** that the function can use when it runs. You fill in this blank space with different values each time you call the function.

Example:

- If the function has a parameter called name, you can give it 'WORLD' one time, 'LEGO' another time, and the robot will greet whoever you choose.

 Carefully type the following code, then press the yellow Play button:

```
# Function with Parameter
from hub import light_matrix # from hub toolbox get light_matrix tool

def greet(name): # define a function with a parameter name
    light_matrix.write('Hi ' + name + '!') # print the message along with the parameter

greet('WORLD') # run the function
```

What will happen when you run it?

The hub screen will display: Hi WORLD! The robot is now greeting a specific name.

How does the code work?

```
def greet(name):
```

- def creates a function greet that takes a **parameter** called **name**.

```
light_matrix.write('Hi ' + name + '!')
```

- This line displays text on the hub screen. Here, plus sign + is used to join separate items, for example: Hi WORLD!

```
greet('WORLD')
```

- This runs the greet function and fills the name parameter with 'WORLD'. The hub screen will display: **Hi WORLD!**

Exercise 2

1. Can you make the robot greet you and your partner? (*Hint: run greet() function twice and use tip 4 to see both names one after the other*)

Section 4: Lights

In this section, you will learn how to display images and control center light on your SPIKE Prime robot.

The robot has:

- A **light matrix screen** on top that can show images
- A **centre power light** that can change colours

Displaying a Face on the Hub

The light matrix can show images, such as animals, faces (moods), and clock faces.

 Carefully type the following code, then press the yellow Play button:

```
from hub import light_matrix # from hub toolbox get light_matrix tool

light_matrix.show_image(light_matrix.IMAGE_HAPPY) # display a happy face on the hub screen
```


What will happen when you run it?

The hub screen will light up with a happy face.

How does the code work?

```
light_matrix.show_image(light_matrix.IMAGE_HAPPY)
```

- a. **.show_image()** is a function (method) of the light_matrix tool (object) - **it tells the screen what to show**
- b. **.IMAGE_HAPPY** is an image of happy face (constant)

 **Tip 5:** You can replace IMAGE_HAPPY with other faces, like IMAGE_SILLY, IMAGE_SAD, or IMAGE_ANGRY.

Exercise 3

1. Can you:

- a. Add sad and angry faces? (Hint: Use tip 5)
- b. Then make the faces last longer? (Hint: Use tip 4)

Introducing “for” Loops (Making Repeating Tasks Easy)

In the last example, if we wanted many faces, we had to use same commands again and again...

But remember — **we are programmers!**

Programming is about making repetitive tasks **easy**. So instead of writing the same lines many times, we use something called a “**for**” loop.

 Carefully type the following code, then press the yellow Play button:

```
# Introduce to for loop and list

from hub import light_matrix # from hub toolbox get light_matrix tool
import time # we need this to make the hub wait for a moment

# moods is a list
moods = [
    light_matrix.IMAGE_HAPPY,
    light_matrix.IMAGE_SILLY,
    light_matrix.IMAGE_SAD,
    light_matrix.IMAGE_ANGRY
]

# we use for loop to loop through each mood in the list
for mood in moods:
    light_matrix.show_image(mood) # show the current face
    time.sleep(1) # wait for 1 second each time so we can see it clearly

light_matrix.clear() # clear the screen only at the end
```

What will happen when you run it?

You will see each faces for one second i.e. **happy face** → **silly face** → **sad face** → **angry face**. At the end, the screen will **clear**.

How does the code work ?

```
moods = [ ... ]
```

- This creates a **list** called moods. A list is like a container that stores multiple items. In this case, the items are different faces the robot can show.

```
for mood in moods:
```

- This is a **for loop**. It goes through each face in the list one by one so the robot can show all the faces without writing the same line of code again and again.

```
light_matrix.show_image(mood)
```

- It tells the hub screen to display the current face from the list.

Controlling the Centre Light

Did you know you can change the colour of the centre light?


 Carefully type the following code, then press the yellow Play button:

```
from hub import light # from hub toolbox get light tool
import color # get different set of colours
import time # we need this to make the hub wait for a moment

light.color(light.POWER, color.RED) # to show red light
time.sleep(1) # pause for a second

light.color(light.POWER, color.BLUE) # to show blue light
```

What will happen when you run it?

The button glows  → 1 second pause → then glows 

How does the code work?

```
import light
```


- light is toolbox (**module**). It has tools to control the hub's center light

```
import color
```

- color is a toolbox (**module**). It has different colours.

```
light.color(light.POWER, color.RED)
```

- **.color()** is a function (**method**) of the light tool (object).
- **.POWER** tells hub to control power (center light) button **.RED** to glow power button in colour red.

 **Tip 6:** You can use more colours such as `color.ORANGE`, `color.YELLOW`, `color.GREEN`, `color.AZURE`, `color.BLUE`, `color.PURPLE`

Exercise 4

1. **Make a Rainbow** 🌈: Can you display all the colours of the rainbow using a loop?
Hint: Use “for” loop, list and tip 6. The color module does not have all the colours of the rainbow, so use the purple and azure in place of violet and indigo.

Section 5: Motors

In this section, we will learn to program the motors to move our robot in straight lines and circles.

 **Carefully type the following code, then press the yellow Play button:**

```
import motor_pair # to control pairs of motors as a single unit
from hub import port # to control motors

motor_pair.pair(motor_pair.PAIR_1, port.A, port.B) # pairs two motors together as pair 1
motor_pair.move_for_degrees(motor_pair.PAIR_1, 1000, 0, velocity=500) # move straight and forward
```

What will happen when you run it?

Robot moved forward, in straight line to some distance.

How does the code work?

```
import motor_pair
```

- **motor_pair** is a (module) toolbox like hub toolbox (module).
- It has many tools that help control two motors together as a pair. This becomes very useful when you want to start and stop both motors at the same time.

```
from hub import port
```

- hub and port are also toolboxes (modules).
- Inside the toolbox, there is **port**, which tells the robot where the motors are plugged in (for example port A or port B).

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

- **.pair()** is a function (method) of the **motor_pair** (module).
- This connects **two motors together as one pair**. In this case, the motor on **port A** and the motor on **port B** are grouped into **PAIR_1** so they can move together.

```
motor_pair.move_for_degrees(motor_pair.PAIR_1, 1000, 0, velocity=500)
```

- **.move_for_degrees()** is a function (method) of the **motor_pair** (module).
- This tells the motor pair to move for a specific number of **degrees of wheel rotation**.
- 1000 means the motors will rotate **1000 degrees**.
- 0 means **no steering**, so the robot moves **straight**.
- velocity controls the **speed and direction** of the robot. Example: A **positive velocity (500)** makes the robot move **forward**, while a **negative** value would make it move in the opposite direction.

Exercise 5

Here we'll explore how the robot moves.

1. Can you change the distance using degrees? Hint: The degrees value controls how far the robot moves.
2. Can you change direction and speed using velocity? Hint: The velocity value controls how fast the robot moves.
3. Can you change direction using steering? Hint: The steering value controls the direction of the robot.
4. *Challenge*: Can you make your robot run in a circle?

Run Loop, Async, and Await

Sometimes we want the robot to perform **several tasks in one program one after the other**, such as: move → turn → move again

The code below shows the general structure of a program using a run loop.

```
import runloop

async def main():
    await task_one
    await task_two

runloop.run(main())
```

 Carefully type the following code, then press the yellow Play button:

```
# Runloop, Async, Await

import motor_pair, runloop # motor_pair to control pairs of motors as a single unit
# runloop to run multiple tasks in one program
from hub import port # to control motors

async def main(): # define a function that will run one task after another

    motor_pair.pair(motor_pair.PAIR_1, port.A, port.B) # pairs two motors together as pair 1

    await motor_pair.move_for_degrees(motor_pair.PAIR_1, 1000, 0, velocity=500) # move straight and forward

    await motor_pair.move_for_degrees(motor_pair.PAIR_1, 1000, 0, velocity=-500) # move straight and backward

runloop.run(main()) # to play the function
```

What will happen when you run it?

The robot moves forward then backward in a straight line.

How does the code work?

```
import runloop
```

- the **runloop** module to let the robot do multiple tasks one after another.

```
async def main():
```

- This creates a special program. Using **async** lets the robot finish one task before starting the next.

```
await motor_pair.move_for_degrees(motor_pair.PAIR_1, 1000, 0,
velocity=500)
```

- Because we used **await**, the robot finishes the first movement **before starting the next one**.

```
runloop.run(main())
```

- This tells the robot to run all the commands in order.

Exercise 6

1. Make the robot move forward, turn, and then come back to where it started!

Hint: Change the steering value to make the robot turn:

- steering = 0 → straight
- steering < 0 → turn left
- steering > 0 → turn right