

1

introduction to Python programming

One exciting thing about LEGO robotics is that you can program your robots by using Word Blocks, a visual programming language based on Scratch, or Python, a popular text-based programming language. Word Blocks provides a great introduction to coding, but Python takes your coding skills to the next level, while still being very intuitive. Because Python code is written in English, you can generally get a sense of what a program is doing just by reading it, even if you don't understand all of the specific details.

Knowing Python has benefits far beyond the world of LEGO: the language is widely used in companies and universities. When you code in Python, you're working with a programming language used by millions of professional coders around the world.

In this chapter, you'll learn some of the basics of Python programming in the MINDSTORMS App. You'll try adding lights

and sounds to your robots and using the Hub buttons to control what happens in your programs.

first steps

When you open a new Python project in the MINDSTORMS App, you'll see the coding area with some starting code already written for you. You'll write your own code below that starting code. The screen also has other useful features. To get you ready to start programming in Python, you'll examine two of those features, the console and the Help Center. Then you'll take a closer look at the starting code.

the console

At the bottom of the screen is an area called the *console* (see Figure 1-1). This is where you'll see error messages if your

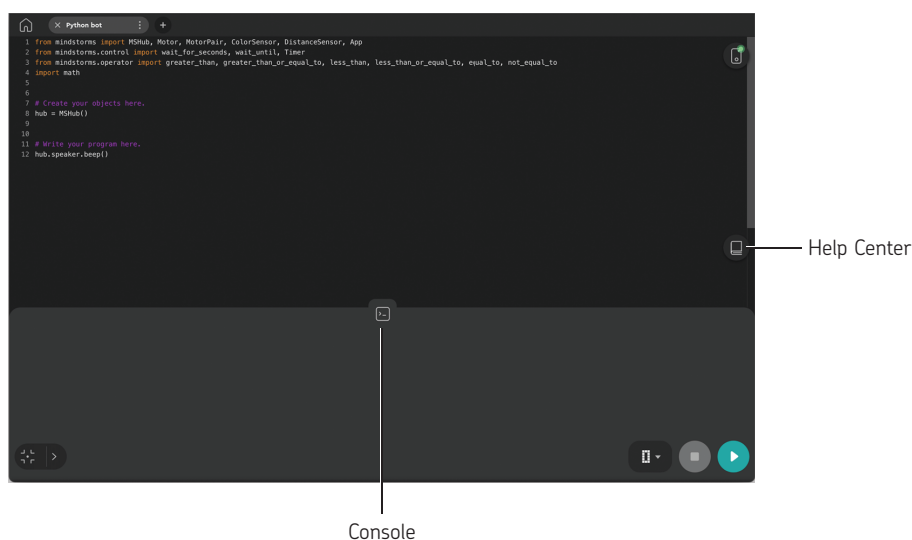




Figure 1-1: A Python project in the MINDSTORMS App

code has any problems as it runs. You can also print messages to the console from within your code, which can be a useful way to keep track of how your program is working.

It's a good idea to have the console open when you're programming with Python. To expand the console, click the Console icon  at the bottom of the window.

the Help Center

The built-in *Help Center* is another valuable tool to help you program in Python. To open the Help Center, click the book icon  on the right side of the screen and choose the Python option. The Help Center contains lots of useful information, including detailed descriptions of the features available in MINDSTORMS Python.

the starting code

As we already noted, when you create a new Python program in the MINDSTORMS App, some code will already be in the coding area. Don't delete any of this starting code—it does some important setup for your program. You don't need to understand all the starting code to begin programming with Python, but for the curious, here's a rundown of what the code means. (This information will be useful as you get further into writing Python code.)

The first few lines of the starting code are called *imports*:

```
from mindstorms import MSHub, Motor, MotorPair, ↵
ColorSensor, DistanceSensor, App
from mindstorms.control import wait_for_seconds, wait ↵
_until, Timer
from mindstorms.operator import greater_than, greater ↵
_than_or_equal_to, less_than, less_than_or_equal_to, ↵
equal_to, not_equal_to
import math
```

This code imports, or loads, the main libraries you'll need for programming your robot. *Libraries* are collections of code. These collections allow you to communicate with the various pieces of MINDSTORMS hardware and to perform operations in programs. For example, to use the Color Sensor in your program, you must import the `ColorSensor` library. Once these libraries have been imported, you can use them in the programs you write.

Here's the next part of the starting code:

```
# Create your objects here.
hub = MSHub()
```

The first line, `# Create your objects here`, is a *comment*. In other words, it's a note to yourself or anyone else looking at the program. Any line that starts with `#` is treated as a comment and isn't executed when the program runs. It's there just for your information. Including comments in your programs is a good habit, since they make your code easier to understand.

The second line, `hub = MSHub()`, creates an *object*, a named piece of code that comes with a set of behaviors. Python objects often represent objects in the real world. In this case, `hub =`

`MSHub()` creates an object representing your actual Hub. Other objects might represent your motors or sensors.

You create an object by giving it a name and telling your program where to look for all the behaviors that go with it. Those behaviors are called *methods*, and they're defined in libraries like the ones imported at the start of the program. Writing `hub = MSHub()` is like saying, "Use the `MSHub` library to create an object and name it `hub`." The `MSHub` library includes methods for controlling many parts of your robot—the buttons, the speaker, the lights, the internal sensors, and the ports—everything that's part of the Hub.

After you create an object, you can use its name inside your program anytime you want that object to do something—that is, anytime you want to call one of the object's methods. You call a method by adding a period after the object's name, followed by the name of the method, followed by a set of parentheses. That's exactly what happens in the last part of the starting code:

```
# Write your program here.
hub.speaker.beep()
```

After another comment, the second line, `hub.speaker.beep()`, calls the `beep()` method of the `hub` object's speaker. This code is a bit like saying, "Hey Hub, use your speaker to play a beep." If you write your program below this line, your program will start with a beep. If you don't want the beep, delete this line of code. (However, don't delete any of the lines above it.)

SPIKE PRIME STARTING CODE

New Python projects in the SPIKE Prime App also start with some code already in the coding area. As you can see, it's similar to the Robot Inventor starting code:

```
from spike import PrimeHub, LightMatrix, Button, ↵
StatusLight, ForceSensor, MotionSensor, Speaker, ↵
ColorSensor, App, DistanceSensor, Motor, MotorPair
from spike.control import wait_for_seconds, wait ↵
_until, Timer
from math import *
hub = PrimeHub()
hub.light_matrix.show_image('HAPPY')
```

The code begins by importing libraries. Then the line `hub = PrimeHub()` creates an object representing the Hub, much like in the Robot Inventor starting code. The last line of code displays a smiley face on the Hub's light matrix. If you write your program below this line, your program will start with the Hub displaying a smiley face, or you can delete this last line if you choose.

controlling Hub outputs and inputs

The `hub` object comes with lots of methods for controlling the Hub's outputs and inputs. As you've already seen, the default starting code calls a method that plays a beep. In fact, you can write code to play many sounds, and Python also has methods for controlling the Hub's lights. These sounds and lights are examples of *outputs*—signals your program sends out into the world. You can also write programs that respond to the outside world by taking *inputs* from the Hub's buttons.

sounds

To make the Hub play a beep, as in the Robot Inventor starting program, use this:

```
hub.speaker.beep()
```

Like many methods, `beep()` comes with a set of *parameters*, options that let you customize the way the method will be executed. You set them inside the parentheses after the method name. In the case of `beep()`, its parameters set the beep's pitch and duration. Pitches are represented by numbers corresponding to MIDI notes, and durations are given in seconds. For example, `beep(67, 1.0)` will play the note G (MIDI note number 67) for one second.

NOTE The system for numbering notes, such as 60 for middle C, comes from MIDI. Short for *Musical Instrument Digital Interface*, MIDI is a standard way for electronic devices to represent music.

You must enter values for the parameters in the correct order (pitch followed by duration), separated by a comma. If you don't specify any values, the beep will play with the default values: a note of 60 (equivalent to middle C) for a duration of 0.2 seconds. If you enter only one number, it will be treated as the first parameter (pitch), and the default duration will be used. For example, `hub.speaker.beep(67)` will play the note G for 0.2 seconds.

The `beep()` method halts the program until the beep has finished playing. To start playing a beep and move immediately to the next line of code, use `hub.speaker.start_beep()` instead. This method is useful if you want to do two things simultaneously, such as playing a sound while displaying an image. To stop a beep, use `hub.speaker.stop()`.

TRY THIS

Write code to play the first line of “Mary Had a Little Lamb.” The notes are B, A, G, A, B, B, B.

Here's one possible solution:

```
hub.speaker.beep(71, 0.5)
hub.speaker.beep(69, 0.5)
hub.speaker.beep(67, 0.5)
hub.speaker.beep(69, 0.5)
hub.speaker.beep(71, 0.5)
hub.speaker.beep(71, 0.5)
hub.speaker.beep(71, 1)
```

In addition to playing beeps, you can play sound files, but to do this, you first have to create another object. This is because most sound files play through your programming device's speakers, not from the Hub. To access these sound files, first create an object by using the `App` library and give it the name `app`:

```
app = App()
```

Then use the `app` object's `play_sound()` method, with the name of the desired sound in the parentheses:

```
app.play_sound('Emotional Piano')
```

The name of the sound must be given inside quotation marks. Single or double quotes will work, as long as they match—either both single or both double. You'll find a list of the many available sounds in the `App` section of the Help Center.

With `play_sound()`, the entire sound will play before the program moves on to the next line of code. You can also start to play a sound and move immediately to the next line of code by using `app.start_sound()`.

light matrix

The Hub's light matrix can be programmed to display a smiley face:

```
hub.light_matrix.show_image('HAPPY')
```

Many other images can be displayed—simply replace `HAPPY` with the name of your desired image. Go to **Hub ▶ Light Matrix** in the Help Center for a list of options.

THINK ABOUT IT

If you run the following two pieces of code, will the results be the same? If not, what will be different?

```
app.play_sound('Emotional Piano')
hub.light_matrix.show_image('HEART')
```

```
app.start_sound('Emotional Piano')
hub.light_matrix.show_image('HEART')
```

Answer: The first version uses the `play_sound()` method, so the entire Emotional Piano sound will play and then a heart will appear on the light matrix. The second version uses `start_sound()`, so the program will play the music and display the heart simultaneously.

To light up individual squares, or *pixels*, on the light matrix, use this:

```
hub.light_matrix.set_pixel(x, y)
```

The x-coordinate defines the pixel's column, and the y-coordinate defines the pixel's row. Rows and columns are numbered from 0 to 4, as shown in Figure 1-2. For example, `set_pixel(0, 0)` lights up the pixel in the upper-left corner of the light matrix, and `set_pixel(4, 4)` lights up the pixel in the lower-right corner.

NOTE This numbering system differs from the one used in Word Blocks programming, which numbers the light matrix's rows and columns from 1 to 5. As you learn more about Python, you'll find that it's common in text-based programming to start counting from 0 rather than 1.

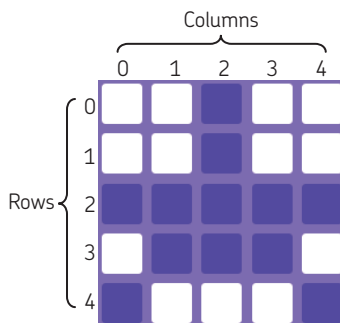


Figure 1-2: In Python programming, the rows and columns of the light matrix are numbered from 0 to 4.

By stringing together multiple `set_pixel()` commands, you can create your own patterns on the light matrix. For example, these lines of code will light up the entire middle column:

```
hub.light_matrix.set_pixel(2, 0)
hub.light_matrix.set_pixel(2, 1)
hub.light_matrix.set_pixel(2, 2)
hub.light_matrix.set_pixel(2, 3)
hub.light_matrix.set_pixel(2, 4)
```

All the commands use 2 as the x-coordinate, corresponding to the middle column of the light matrix, but each command has a different y-coordinate, so the middle pixel in each row will be lit up.

To turn off all the pixels in the light matrix, use this:

```
hub.light_matrix.off()
```

TRY THIS

Write a program to light the bottom row of the light matrix.

Here's one possible solution:

```
hub.light_matrix.set_pixel(0, 4)
hub.light_matrix.set_pixel(1, 4)
hub.light_matrix.set_pixel(2, 4)
hub.light_matrix.set_pixel(3, 4)
hub.light_matrix.set_pixel(4, 4)
```

The light matrix can scroll words and numbers as well as display static images. The method `write()` scrolls whatever is in its parentheses on the light matrix. If the text is in quotation marks, it will be displayed exactly as you type it (minus the quotation marks). For example, this line of code scrolls the words *Hello there*:

```
hub.light_matrix.write('Hello there')
```

The sequence of characters in quotes is called a *string*. We've already used strings in this chapter to select sound and image files. You'll learn more about strings in later chapters, where you'll also learn how to display sensor readings and other information on the light matrix without having to type it out exactly.

status light

The *status light* is the round center button on the front of the Hub. In Word Blocks programming, it's referred to as the *center button light*. To turn on the Hub's status light, use this command:

```
hub.status_light.on()
```

If you don't specify a color in the parentheses, the light will be white. To choose a different color, enter its name as a string (that is, in quotes) in the parentheses, like this:

```
hub.status_light.on('blue')
```

You have 11 colors to choose from: azure, blue, cyan, green, orange, pink, red, violet, yellow, white, and black (which turns off the light). These two lines of code, for example, turn on a red light and then turn the light off:

```
hub.status_light.on('red')
hub.status_light.on('black')
```

If you run the code as it's written, however, the light will turn red and then immediately turn off so fast that you won't actually see anything happen. In the next chapter, you'll learn how to fix this problem.

Rather than set the light to black, you can turn off the status light:

```
hub.status_light.off()
```

TRY THIS

Try reversing the red and black status light commands, like so:

```
hub.status_light.on('black')
hub.status_light.on('red')
```

What do you think will happen when you execute the program? Run it and see.

Answer: The status light glows red. The first command, which turns off the light, is followed immediately by the command to turn the light red, so you see only the red light.

Hub buttons

The Hub's left and right buttons function as inputs in your programs. You can use these inputs in several ways. For example, you can wait until a button is pushed. This code waits until the left button is pushed and then turns the status light pink:

```
hub.left_button.wait_until_pressed()
hub.status_light.on('pink')
```

You can wait until a button is released too. The following code turns on the pink light when the button is pressed and turns it off when the button is released:

```
hub.left_button.wait_until_pressed()
hub.status_light.on('pink')
hub.left_button.wait_until_released()
hub.status_light.off()
```

This way, the light will stay on for as long as you hold down the button.

TRY THIS

Let's say you want to display a new color of status light every time you press the left Hub button. Will the following program display red, then green, then blue? Try it and see:

```
hub.status_light.on('red')
hub.left_button.wait_until_pressed()
hub.status_light.on('green')
hub.left_button.wait_until_pressed()
hub.status_light.on('blue')
```

Answer: This program does not display all three colors. Instead, the status light glows red when you start the program and then goes directly to blue when you press the Hub button. You never see the green light. Here's why: both `hub.left_button.wait_until_pressed()` commands in the program are triggered by your single button press.

What if you want to use multiple button presses in a program? To make sure that each press of the button triggers only one `wait_until_pressed()` command, pair each `wait_until_pressed()` command with a `wait_until_released()` command. That way, the code will wait for the button to be *bumped*—both pressed and released.

For example, the code in the red/green/blue light program you just tried could be rewritten like this:

```
hub.status_light.on('red')
hub.left_button.wait_until_pressed()
hub.left_button.wait_until_released()
hub.status_light.on('green')
hub.left_button.wait_until_pressed()
hub.left_button.wait_until_released()
hub.status_light.on('blue')
```

This program displays all three light colors, switching colors each time you press and release the left Hub button.

You can also check whether the left button is pressed currently by using `hub.left_button.is_pressed()`, or you can check whether the button was pressed at any time since you last checked its status by using `hub.left_button.was_pressed()`. You'll learn how to incorporate these commands into `if` statements (conditionals) in the next chapter.

To program with the Hub's right button instead of the left button, use `right` instead of `left` in your code. For example, `hub.right_button.wait_until_pressed()` pauses the program until the right button is pressed.

NOTE The Hub's built-in Accelerometer and Gyro Sensor, which together are called the *Motion Sensor* in Python programming, provide other ways of getting input from the Hub. We cover programming with the Motion Sensor in Chapter 4.

TRY THIS

Write a program that displays a heart on the light matrix and then makes it smaller when the right Hub button is pressed.

Here's one possible solution:

```
hub.light_matrix.show_image('HEART')
hub.right_button.wait_until_pressed()
hub.light_matrix.show_image('HEART_SMALL')
```

some programming basics

You've gotten your first taste of Python as you've learned to write simple programs that control the Hub's inputs and outputs. Now let's take a step back and look at some basic rules and best practices for programming with Python.

indenting and spacing

Unlike many programming languages, Python uses *indentation*, or spaces at the beginning of lines, to indicate to the computer how to run a program. For that reason, getting your indentations correct really matters! You'll see this in the next chapter when you start using control structures like loops and conditionals.

While spacing is important at the starts of lines, the spacing between elements on the same line doesn't matter. For example, Python will treat `hub.speaker.beep(71,1)` and `hub.speaker.beep(71, 1)` as equivalent. Whether or not you have a space between 71 and 1, the program will run the same way. However, the comma between the parameter values is required.

uppercase and lowercase letters

Python is case sensitive: it views uppercase and lowercase versions of a letter as different. So, if you write `hub.light_matrix.show_image('Heart')` instead of `hub.light_matrix.show_image('HEART')`, you'll get an error message. HEART in all caps is a valid image to display on the light matrix, but Heart is not.

Be especially careful about capitalization when you're naming things; Python will treat `myHub` and `myhub` as two different objects. Python doesn't allow spaces in the middle of a name, so capitalization can help make the names you choose more readable, as long as the capitalization you use is consistent. For example, capitalizing the *H* in `myHub` helps make it clear where the first word of the name ends and the second word begins.

use comments!

As you already saw, any line that starts with `#` is treated as a comment and isn't executed. This feature is extremely useful for documenting your programs. For example, it helps to leave a comment before some lines of code, explaining what that code is supposed to do. You might think the code's purpose is clear as you're writing it, but if you come back to the code a week later, you might not remember what you were thinking.

Comments can also be handy for *debugging*—testing code and fixing mistakes. For example, if you want to temporarily stop a line of code from executing, but you don't want to have to delete and retype it later, simply turn that line into a comment by placing a `#` in front of it. If you want to stop multiple lines of code from executing (or if you want to write a long comment), enclose the lines of code in triple quotes (`"""`), like this:

```
"""
these lines won't execute
because they're in triple quotes:

hub.speaker.beep(67, 1.0)
hub.light_matrix.show_image('HEART')
"""
```

printing to the console

You can display words and numbers in the console at the bottom of the coding area by using `print()`. Enter the text you want to display inside the parentheses as a string (using quotes). For example, this code will display the message *Hello there* in the console:

```
print('Hello there')
```

You can also print sensor values and other information to the console. For example, here's how to print the Hub's current volume level:

```
print(hub.speaker.get_volume())
```

Notice that `hub.speaker.get_volume()` isn't enclosed in quotation marks. This is because it isn't a string—it's a method of the `hub` object. When this line of code runs, your program will know to look up the current volume level of the Hub, and it will print whatever that number is to the console.

Printing to the console can be helpful in debugging your programs and is often much easier than displaying text on the Hub's light matrix. For example, you could print output values from the Distance Sensor to the console to check whether the sensor is detecting an object. You can even combine strings and methods inside `print()` statements to make your output easier to understand. For example, this line of code prints the word *volume* followed by the current volume level:

```
print('volume', hub.speaker.get_volume())
```

Since `volume` is enclosed in quotes, it's treated as a string and is printed exactly the way it's typed, whereas `hub.speaker.get_volume()` isn't in quotes, so it's treated as a method and will result in printing the current volume level. When you mix strings and non-strings inside a `print()` statement, separate each element with a comma.

summary

In this chapter, you learned how to write Python programs to control the Hub inputs and outputs. You also learned a few Python programming basics, such as how indents, spaces, and capitalization are treated, and the value of using comments and `print()` statements. In the next chapter, you'll expand your knowledge of Python by learning how to use control structures like loops and conditionals, as well as operators.

PROJECTS

SLEEPING

Write a program that displays a sleeping face on the light matrix while playing snoring sounds.

Here's one possible solution:

```
app = App()
hub.light_matrix.show_image('ASLEEP')
app.play_sound('Snoring')
```

TRAFFIC LIGHT

Write a program that switches the status light from green to yellow to red, changing to the next color when you press a Hub button.

Here's one possible solution:

```
hub.status_light.on('green')
hub.left_button.wait_until_pressed()
hub.left_button.wait_until_released()
hub.status_light.on('yellow')
hub.left_button.wait_until_pressed()
hub.left_button.wait_until_released()
hub.status_light.on('red')
```

WALK SIGNAL

Write a program that displays a green status light until the left Hub button is pressed. Then, the status light turns red and the light matrix scrolls *WALK*. After *WALK* finishes scrolling, the status light turns green again.

Here's one possible solution:

```
hub.status_light.on('green')
hub.left_button.wait_until_pressed()
hub.status_light.on('red')
hub.light_matrix.write('WALK')
hub.status_light.on('green')
```
