

The Official
ROBLOX
Guide



Coding with Roblox Lua

in **24**
Hours

Table of Contents

HOUR 1: Coding Your First Project	1
Installing Roblox Studio	1
Let's Take a Tour	2
Opening the Output Window	5
Writing Your First Script	6
Error Messages	11
Leaving Yourself Comments	12
HOUR 2: Properties and Variables	17
Object Hierarchy	18
Keywords	19
Properties	20
Finding Properties and Data Types	22
Creating Variables	22
Changing the Color Property	25
Instances	26
HOUR 3: Creating and Using Functions	31
Creating and Calling Functions	31
Understanding Scope	33
Using Events to Call Functions	33
Understanding Order and Placement	36
HOUR 4: Working with Parameters and Arguments	43
Giving Functions Information to Use	43
Working with Multiple Parameters and Arguments	45
Returning Values from Functions	49
Returning Multiple Values	50
Returning Nil	51
Dealing with Mismatched Arguments and Parameters	51
Working with Anonymous Functions	52

HOURL 5: Conditional Structures	57
if/then Statements	58
elseif	62
Logical Operators	62
else	63
HOURL 6: Debouncing and Debugging	73
Don't Destroy, Debounce	73
Figuring Out Where Things Go Wrong	82
HOURL 7: while Loops	91
Repeat Forever, while true do	91
Some Things to Keep in Mind	92
while Loops and Scope	98
HOURL 8: for Loops	101
How for Loops Work	102
Nested Loops	109
Breaking Out of Loops	110
HOURL 9: Working with Arrays	113
What Are Arrays?	113
Adding Items Later	114
Getting Information from a Specific Index	114
Printing an Entire List with ipairs()	115
Folders and ipairs()	116
Finding a Value on the List and Printing the Index	121
Removing Values from an Array	122
Numeric for Loops and Arrays	123
HOURL 10: Working with Dictionaries	127
Intro to Dictionaries	127
Adding and Removing from Dictionaries	130
Removing Key-Value Pairs	130
Working with Dictionaries and Pairs	132
Returning Values from Tables	133

HOURL 11: Client Versus Server	145
Understanding the Client and the Server	145
Working with GUIs	146
Understanding RemoteFunctions	149
Using RemoteFunctions	149
HOURL 12: Remote Events: One-Way Communication	161
Remote Events: A One-Way Street	161
Communicating from the Server to All Clients	162
Communicating from the Client to the Server	165
Communicating from the Server to One Client	170
Communicating from Client to Client	171
HOURL 13: Using ModuleScripts	173
Coding Things Just Once	173
Placing ModuleScripts	174
Understanding How ModuleScripts Work	174
Naming ModuleScripts	174
Adding Functions and Variables	175
Understanding Scope in ModuleScripts	176
Using Modules in Other Scripts	177
Don't Repeat Yourself	183
Dealing in Abstractions	183
HOURL 14: Coding in 3D World Space	187
Understanding X, Y, and Z Coordinates	187
Refining Placement with CFrame Coordinates	189
Offsetting CFrames	191
Adding Rotations to CFrames	191
Working with Models	192
Understanding World Coordinates and Local Object Coordinates	193
HOURL 15: Smoothly Animating Objects	199
Understanding Tweens	199
Setting TweenInfo Parameters	201
Chaining Tweens Together	205

HOUR 16: Solving Problems with Algorithms	209
Defining Algorithms	209
Sorting an Array	210
Sorting in Descending Order	212
Sorting a Dictionary	213
Sorting by Multiple Pieces of Information	216
HOUR 17: Saving Data	219
Enabling Data Stores	219
Creating a Data Store	220
Using Data in the Store	220
Limiting the Number of Calls	225
Protecting Your Data	225
Saving Player Data	226
Using <code>UpdateAsync</code> to Update a Data Store	226
HOUR 18: Creating a Game Loop	229
Setting Up Game Loops	229
Working with <code>BindableEvents</code>	230
HOUR 19: Monetization: One-Time Purchases	243
Adding Passes to Your Experience	243
Configuring the Pass	246
Prompting In-Game Purchases	247
HOUR 20: Object-Oriented Programming	259
What Is OOP?	259
Organizing Code and Projects	259
Making a New Class	260
Adding Class Properties	261
Using Class Functions	263
HOUR 21: Inheritance	271
Setting Up Inheritance	272
Inheriting Properties	274
Working with Multiple Child Classes	277

Inheriting Functions	278
Understanding Polymorphism	278
Calling Parent Functions	282
HOUR 22: Raycasting	287
Setting Up the Function to Raycast	287
3D Math Trick: Getting the Direction	289
Setting Raycast Parameters	290
3D Math Trick: Limit Direction	293
HOUR 23: Plopping Objects in an Experience: Part 1	297
Setting Up the Object	298
Creating a Plop Button	302
Tracking Mouse Movements	303
Previewing the Object	307
HOUR 24: Plopping Objects in an Experience: Part 2	313
Detecting Mouse Input	314
Sending a Message to the Server	316
Getting the Message	317
APPENDIX A: Roblox Basics	321
Keywords	322
DataType Index	322
Operators	324
Naming Conventions	325
Animation Easing	325
Possible Solutions to Exercises	326
Index	355

HOUR 1

Coding Your First Project

What You'll Learn in This Hour:

- ▶ Why Roblox and Lua are a perfect combination
- ▶ What Roblox Studio's main windows are
- ▶ How to say "Hello" to the world with your first code
- ▶ How to make a part explode
- ▶ How to check for errors
- ▶ How to leave a comment

Roblox is the world's most popular game development platform. All types of people come together to create amazing virtual experiences: artists, musicians, and—you guessed it—coders. Coding is what allows players to interact with the world that they see.

In Roblox, the coding language used is Lua. Lua is one of the easiest coding languages to learn, and when used with Roblox Studio, you can see the results of your code fast. For example, want to create an enormous explosion with a massive blast radius? You can do that with just a couple of lines of Lua.

Roblox Studio is the tool in which all Roblox games are created, and when paired with Lua, it offers seamless access to multiplayer servers, physics and lighting systems, world-building tools, monetization systems, and more. And even though Roblox provides the environment in which your program runs, you control the vision. You are the creator and artist. Roblox gives you the canvas and paints, and Lua the brushes and actions. But *you*, with some well-placed dabs of code, get to create your masterpiece. This first hour covers how to set up Roblox Studio, make your first script, and test your code.

Installing Roblox Studio

Before you get started, make sure you have Roblox Studio installed. It runs on Windows and MacOS, and you can grab a copy at <https://roblox.com/create>. Click Start Creating to begin. You'll need to create a Roblox account if you don't yet have one (see Figure 1.1).

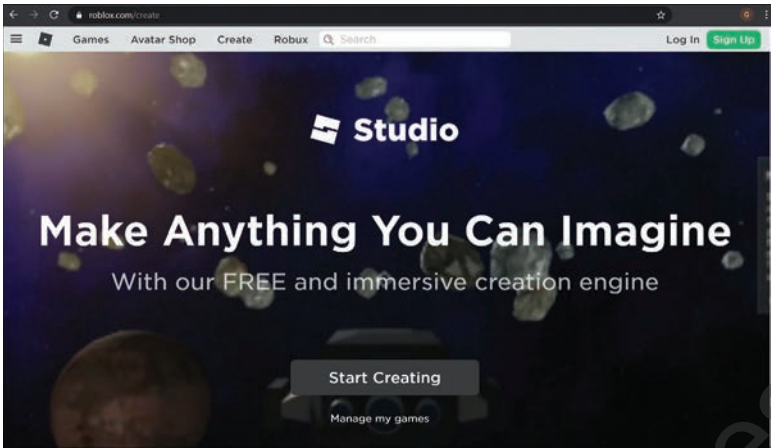


FIGURE 1.1
You need an account to use Roblox Studio. It's free and just a quick sign-up away.

Let's Take a Tour

Roblox Studio provides everything you need to create games. It includes assets such as character models, items to put in the world, graphics for the sky, soundtracks, and more.

Go ahead and launch Roblox Studio to see the window shown in Figure 1.2. Enter the login information for the account you created when you signed up on the Roblox website and click Log In.

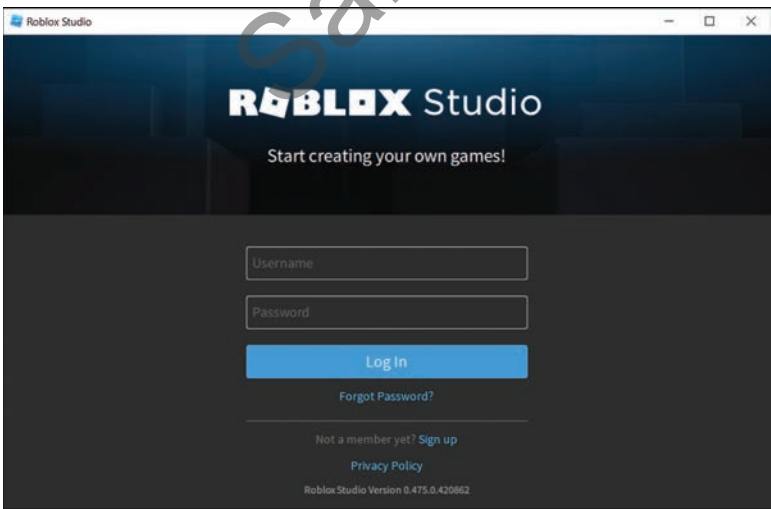


FIGURE 1.2
Enter your normal Roblox account information.

When you first open up Studio, you see templates. These are starting places you can use for your experiences. The simplest starting point for any project is the *Baseplate* template. Click on the Baseplate template, as shown in Figure 1.3.

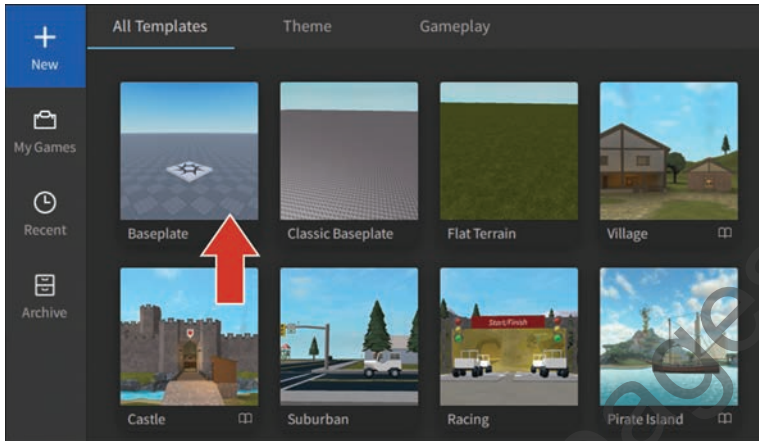


FIGURE 1.3

Studio offers template places you can use as starting points.

Let's start with a quick overview of the main parts of the screen in Figure 1.4, and then move straight into your first line of code:

1. The offerings in the Toolbar ribbon change according to the menu tab you've selected.
2. The Toolbox contains existing assets to add to your game. You can also create your own assets through a 3D modeling program such as Blender3D, and Studio includes a set of mesh-editing tools to customize the 3D models already available.
3. The 3D Editor provides a view of the world. Hold your right mouse button to turn the view, and use the WASD keys to reposition the camera. Table 1.1 describes the different controls to move the camera.
4. The Explorer window provides convenient access to every key asset or system in the game. You use this to insert objects into your experience.
5. Use the Properties window to make changes to objects in the game, such as color, scale, value, and attributes. Select an object in the Explorer to see available properties.

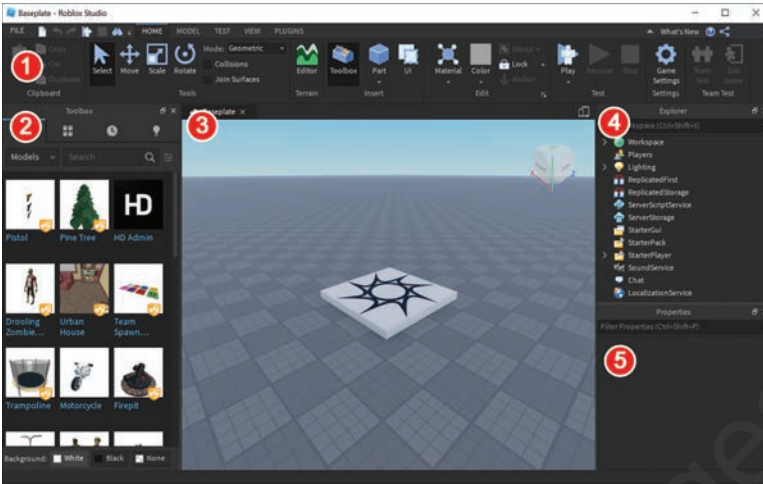


FIGURE 1.4 There are a number of panels, buttons, and lists in the Studio, and you'll quickly become familiar with them.

TABLE 1.1 Camera Controls

Key	Movement
W A S D	Move the camera up, left, down, or right
E	Move the camera
Q	Lower the camera down
Shift	Move the camera slower
Right mouse button (hold and drag mouse)	Turn the camera
Middle mouse button	Drag the camera
Mouse scroll wheel	Zoom the camera in or out
F	Focus on selected object

There are numerous ways to configure this main screen, including hiding different sections, rearranging their positioning to be more convenient, and changing their size.

Roblox Studio is a very complete game development environment that goes well beyond Lua. It's a big topic on its own, so you may want to check out our other book, *Roblox Game Development in 24 Hours*, for help.

Opening the Output Window

The Output window in Studio isn't open by default, but you need this before you continue so that you can see errors and messages that are related to your code.

Use the following steps to display the Output window:

1. Click the View tab (see Figure 1.5). If you ever close a window and need to reopen it, you can find it here.

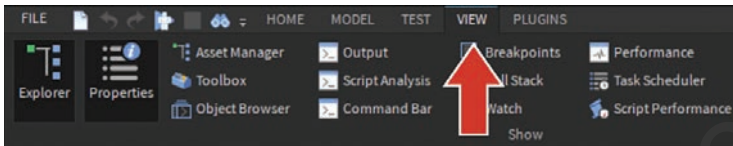


FIGURE 1.5

Use the View tab to control which windows are open.

2. Click Output (see Figure 1.6) to display the Output window at the bottom of your screen, as shown in Figure 1.7.

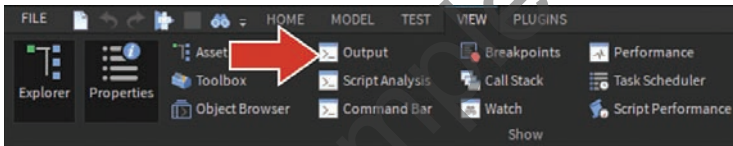


FIGURE 1.6

Click the Output option to open the Output window.

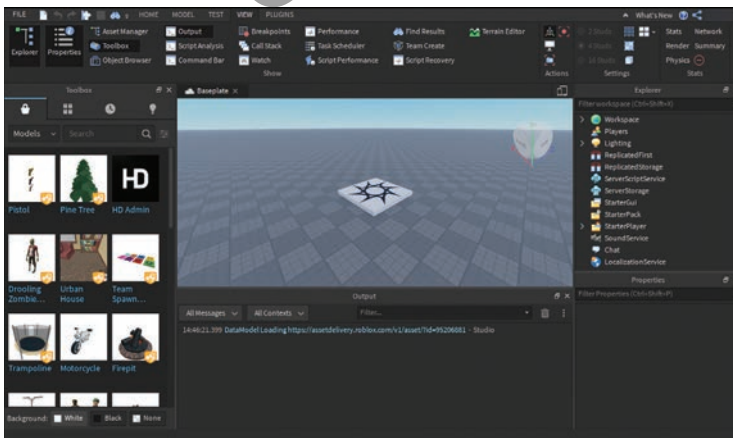


FIGURE 1.7

The Output window opens beneath the 3D Editor.

Writing Your First Script

On to coding! You need something to hold your code, and that's a script. You can insert scripts directly into objects within the world. In this case, you're inserting a script into a part.

Insert a Script into a Part

A part is the basic building block of Roblox. Parts can range in size from very tiny to extremely large. They can be different shapes such as a sphere or wedge, or they can be combined into more complex shapes.

1. Return to the Home tab and click Part (see Figure 1.8). The part appears in the 3D Editor at the center of your camera view.

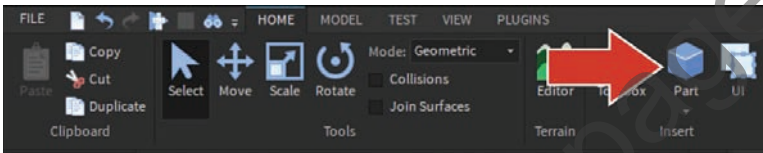


FIGURE 1.8

Click Part on the Home tab to insert a part.

2. To add a script, in Explorer, hover the part and click the + symbol, and then select Script from the drop-down menu (see Figure 1.9).

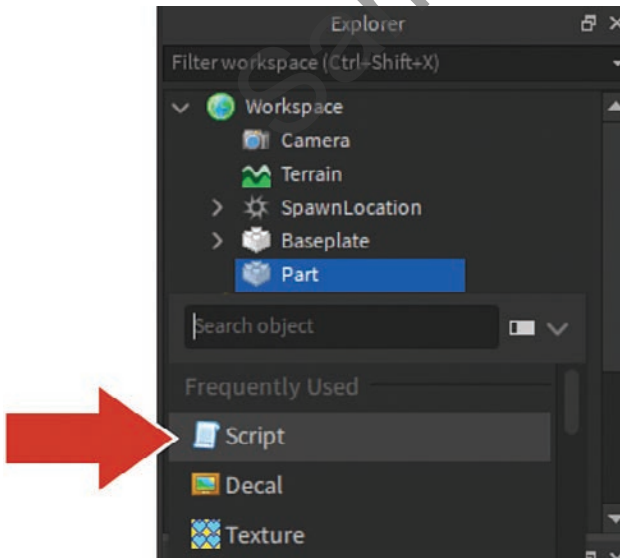


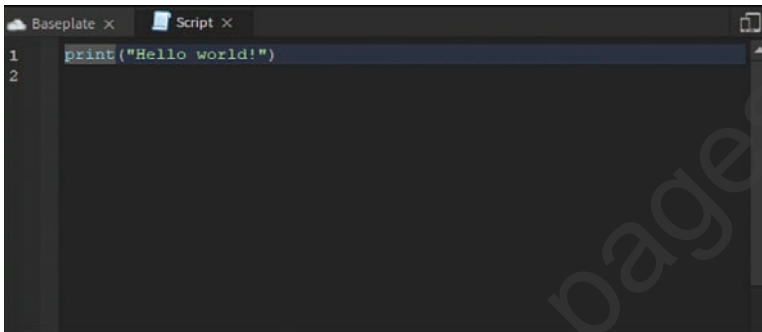
FIGURE 1.9

You use Explorer to insert a script into the part.

TIP**Finding Items Quickly**

Typing the first letter (S, in this case) or two of the items you are adding filters the list so you can locate that item quickly.

The script automatically opens. At the top, you see words familiar to any coder: "Hello world!" (see Figure 1.10).

**FIGURE 1.10**

The window shows the default script and code.

Writing Some Code

Since the 1970s, "Hello World!" has been one of the first pieces of code people have learned. Here it's being used in the `print` function. Functions are chunks of code that serve a specific purpose. As you learn to code, you'll use prebuilt functions like `print()`, which displays messages in the Output window. You will, of course, also learn how to create functions of your own.

`print()` displays a string, which is a type of data usually used with letters and numbers that need to stay together. In this case, you're printing "Hello world!":

1. Make this code your own by changing the message inside of the quotation marks to what you want for dinner tonight. Here's an example:

```
print("I want lots of pasta")
```

2. To test the code, in the Home tab, click Play (see Figure 1.11).

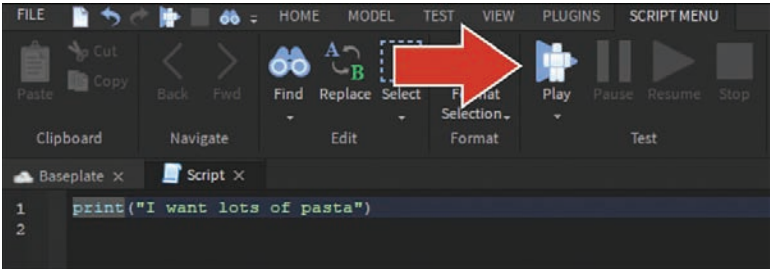


FIGURE 1.11
Click Play to test your script.

Your avatar will fall into the world, and you can see your dinner dreams displayed in the Output window, along with a note about which script that message came from (see Figure 1.12).

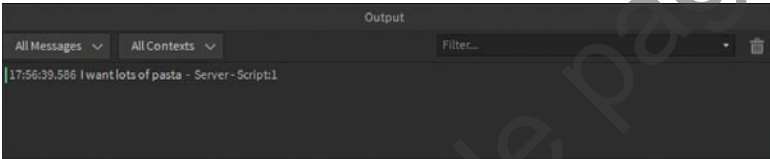


FIGURE 1.12
The string is displayed in Output.

3. To stop the playtest, click the Stop button (see Figure 1.13).

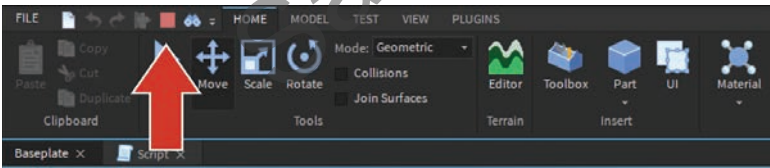
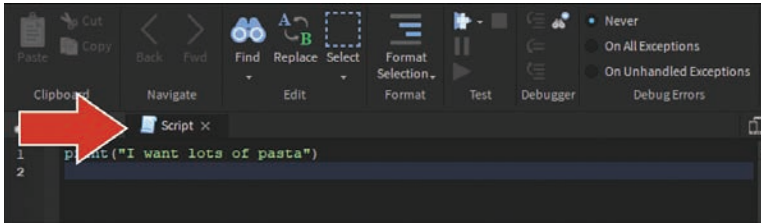


FIGURE 1.13
Click Stop to quit the playtest.

4. Return to your script by clicking on the tab above the 3D Editor, as shown in Figure 1.14.

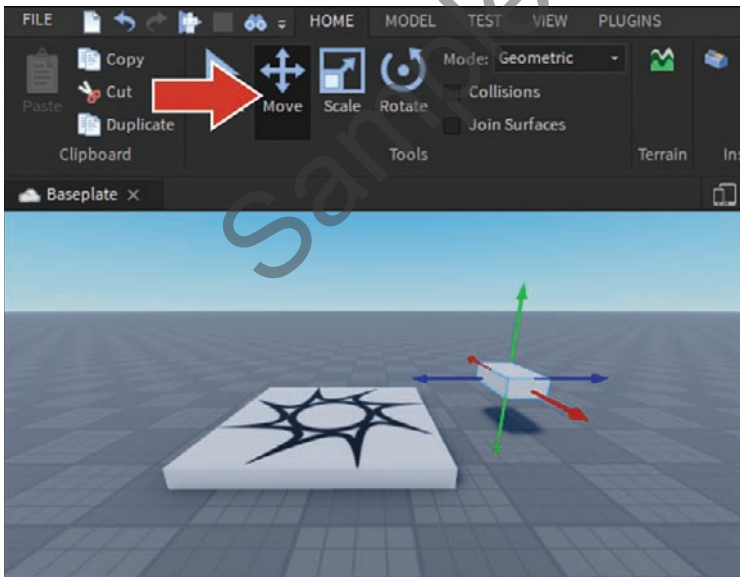
**FIGURE 1.14**

Click Script to return to the window where your script is visible.

Code an Explosion

Code of course can do more than just display messages to the output window. It can completely change how players interact with the world and make it come alive. Let's take a slightly longer piece of code and make the block in the Baseplate template destroy anything it touches:

1. Use the Move tool (see Figure 1.15) to move the block off the ground and away from the spawn point. The code you're going to write will destroy anything it touches, and you don't want it to go off prematurely.

**FIGURE 1.15**

Move the part up and away from the spawn.

- In the Properties window, scroll to Behavior and make sure Anchored (see Figure 1.16) is selected so the block doesn't fall when you click Play.

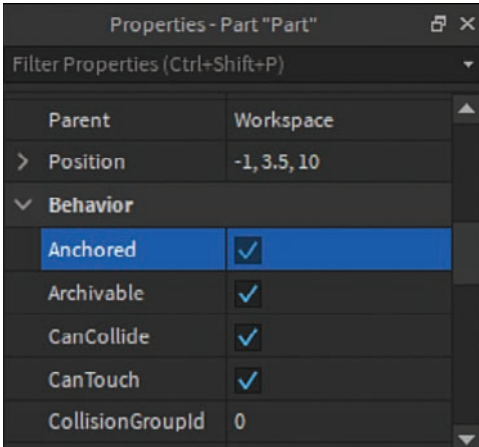


FIGURE 1.16

Check Anchored to keep the blocks from falling.

- In the script, below the print function, add the following code:

```
print("I want lots of pasta!")

-- Destroys whatever touches the part
local trap = script.Parent
local function onTouch(partTouched)
    partTouched.Destroy()
end
trap.Touched:Connect(onTouch)
```

NOTE

Code Boxes

Code boxes for this book will be presented in light mode, unless specifically calling attention to Studio UX.

- Click Play and run up and touch the part.

The result should be that your character breaks or parts of your avatar are destroyed. You may notice that this code only destroys what touches it directly, such as your feet. Try jumping on top

of the block or brushing against it with just a hand. You'll see only that part of your avatar is destroyed.

The reason is that code only does what you tell it, and you told the part to destroy only what it touches and nothing more. You have to tell it how to destroy the rest of the player. Throughout this book, you'll learn how to write additional instructions so that the code can handle more scenarios like this one. In Hour 4, "Parameters and Arguments," you'll learn how to make sure it destroys the entire player character.

Error Messages

What if the code didn't work? The truth is, all engineers make mistakes in their code. It's no big deal, and the editor and the output window can help you spot mistakes and fix them. Try making a couple of mistakes to learn how to better spot them later:

1. Delete the second parenthesis from the `print` function. A red line appears under `local`. (See Figure 1.17.) In the editor, red lines indicate a problem.

A screenshot of a code editor window with two tabs: "Baseplate x" and "Script x". The script content is as follows:

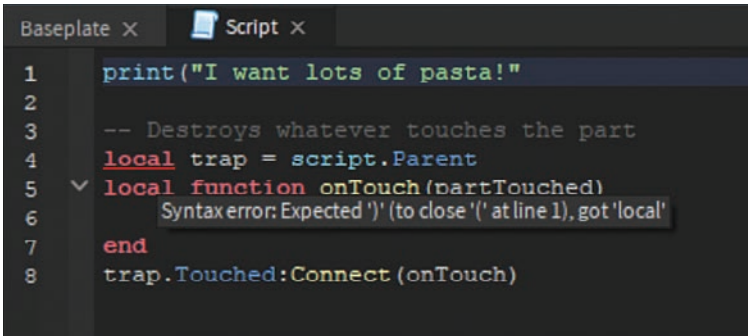
```
1 print("I want lots of pasta!"
2
3 -- Destroys whatever touches the part
4 local trap = script.Parent
5 local function onTouch(partTouched)
6     partTouched:Destroy()
7 end
8 trap.Touched:Connect(onTouch)
```

A red squiggly line is under the word "local" on line 4, indicating a syntax error. The code is color-coded: strings are green, comments are grey, and keywords like "local", "function", "end", and "Connect" are red.

FIGURE 1.17

A red line indicates Studio has spotted an error.

2. Hover over the red line, and the editor gives you a clue about what's gone wrong, as shown in Figure 1.18. But don't fix the mistake quite yet.



```

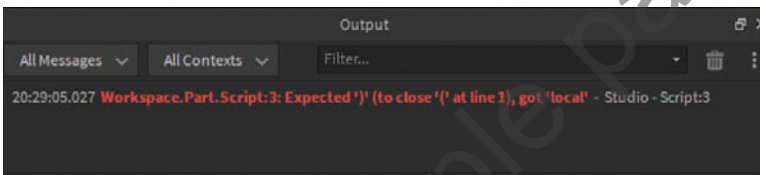
1  print("I want lots of pasta!")
2
3  -- Destroys whatever touches the part
4  local trap = script.Parent
5  local function onTouch(partTouched)
6      Syntax error: Expected ')' (to close '(' at line 1), got 'local'
7  end
8  trap.Touched:Connect(onTouch)

```

FIGURE 1.18

An error message displays when you hover over the red line.

3. Click Play, which causes an error message to display in the Output window, as shown in Figure 1.19. Click the red error, and Studio takes you to where it thinks the problem is.



```

Output
All Messages ▾ All Contexts ▾ Filter...
20:29:05.027 Workspace.Part.Script:3: Expected ')' (to close '(' at line 1), got 'local' - Studio - Script:3

```

FIGURE 1.19

The error shows up as a clickable red message in the Output window.

Stop the playtest and fix the issue.

TIP

Changes Made While Playtesting Aren't Permanent

Be careful about making changes while in a playtest because the work you've done is not automatically saved. If you do make changes, be sure to click Preserve Changes when you stop the playtest.

Leaving Yourself Comments

In the previous code, you may notice the sentence `-- Destroys whatever touches the part`. This is a comment. Comments begin with two dashes. Anything on the same line as the dashes doesn't affect the script.

Coders use comments to leave notes to themselves and others about what the code does. Trust us: When you haven't looked at a piece of code in months, it's very easy to forget what it does.

The following code shows what it might look like to add a comment at the top of the script you wrote earlier in this hour:

```
-- What do I want for dinner?  
print("I want lots of pasta!")
```

Summary

In just one hour, you've come a long way, particularly if this happened to be your first time coding or using Roblox Studio. This hour covered creating an account and opening Roblox for the first time. By using the + button, you were able to insert a script into a part, and then you added code that turned the part into a trap for anyone who happened to touch it.

In addition, you learned how to test code using the Play button and use the built-in error detection within the script editor and Output window to help you troubleshoot when something goes wrong.

Finally, you learned about comments, which are only readable in the script editor and can be used to leave notes about the purpose of the code.

Q&A

Q. Can you use Studio on a Chromebook?

A. To create, Studio must be run on a MacOS or Windows machine. Once a game has been published, it's available to be played on Android, Apple, Mac, PC, Chrome, and potentially even Xbox Live.

Q. How do I reopen a script if I close it?

A. If you close out of the script editor, you can reopen it by double-clicking the script object in Explorer.

Q. How do I save my work?

A. Go to File, Publish to Roblox to save to the cloud, which makes your game accessible from any computer.

Q. Where do I go if I want additional information about how Roblox Studio works?

A. You can visit developer.roblox.com to find documentation on all of Studio's features and API.

Workshop

Now that you have finished, let's review what you've learned. Take a moment to answer the following questions.

Quiz

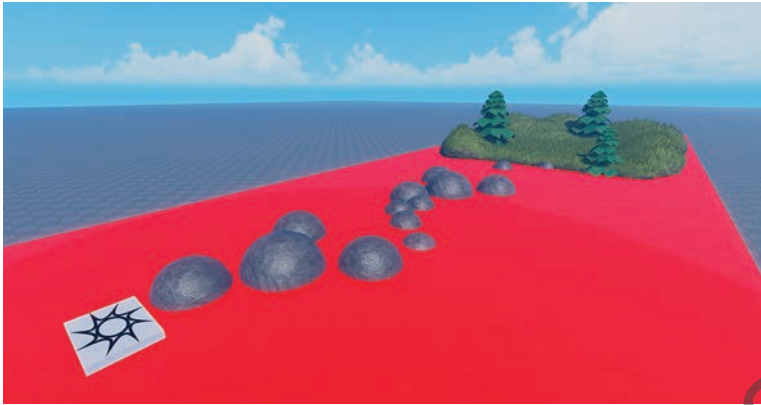
1. Roblox uses the _____ coding language.
2. Aspects of an object such as color, rotation, and anchored can be found in the _____ window.
3. Game objects are found in the _____ window.
4. To enable the Output window, which displays code messages and errors, enable it in the _____ tab.
5. True or false: Comments change the code to enable new functionality.
6. To force parts to stay in place, they need to be _____.

Answers

1. Lua
2. Properties
3. Explorer
4. View
5. False. Comments do not affect the code and are used to leave notes to yourself and other coders as to the purpose of the script.
5. Anchored

Exercise

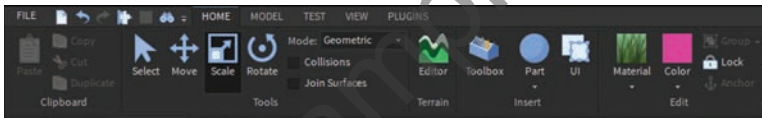
Before moving on, take a moment to experiment with the creation tools by creating a mini obstacle course. It could be individual parts the player has to avoid, or it could be a lava floor like the one shown in Figure 1.20.

**FIGURE 1.20**

Use what you've learned so far to create a lava obstacle course.

Tips

- ▶ Create more parts and manipulate them with the Move, Translate, and Scale tools found on the Home tab (see Figure 1.21). You can also change the parts' appearance with Material and Color.

**FIGURE 1.21**

The Home tab has the tools you need to create and manipulate parts.

- ▶ Use a single large part and insert a script as you did earlier to turn it into lava.
- ▶ Additional models can be found in the Toolbox; just be aware that some models may already have scripts in them.
- ▶ Don't forget to anchor all parts and models.
- ▶ If you know how to use the terrain tools, you can work that into your obstacle course as well.