The Official ROBLOX Guide

Coding with Roblox Lua in 24 Hours

Pearson

The Official ROBLOX Guide

Coding with Roblox Lua in 24 Hours

Pearson

Coding with Roblox Lua in 24 Hours: The Official Roblox Guide

ISBN-13: 978-0-13-682942-3
ISBN-10: 0-13-682942-2

Library of Congress Control Number: 2021948694

ScoutAutomatedPrintCode

Trademarks

Warning and Disclaimer

Special Sales

Editor-in-Chief
Debra Williams Cauley

Acquisitions Editor
Kim Spenceley

Editorial Services
The Wordsmithery LLC

Managing Editor
Sandra Schroeder

Senior Project Editor
Tonya Simpson

Copy Editor
Charlotte Kughen

Indexer
Cheryl Lenser

Proofreader
Sarah Kearns

Editorial Assistant
Cindy Teeters

Cover Designer
Chuti Prasertsith

Compositor
Bronkella Publishing LLC

Graphics Processing
TJ Graham Art

Contents at a Glance

Hour 1 Coding Your First Project 1
2 Properties and Variables 17
3 Creating and Using Functions 31
4 Working with Parameters and Arguments 43
5 Conditional Structures 57
6 Debouncing and Debugging 73
7 `while` Loops 91
8 `for` Loops 101
9 Working with Arrays 113
10 Working with Dictionaries 127
11 Client Versus Server 145
12 Remote Events: One-Way Communication 161
13 Using ModuleScripts 173
14 Coding in 3D World Space 187
15 Smoothly Animating Objects 199
16 Solving Problems with Algorithms 209
17 Saving Data 219
18 Creating a Game Loop 229
19 Monetization: One-Time Purchases 243
20 Object-Oriented Programming 259
21 Inheritance 271
22 Raycasting 287
23 Plopping Objects in an Experience: Part 1 297
24 Plopping Objects in an Experience: Part 2 313
Appendix A Roblox Basics 321
Index 355

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

HOUR 5: Conditional Structures 57

`if/then` Statements 58

`elseif` 62

Logical Operators 62

`else` 63

HOUR 6: Debouncing and Debugging 73

Don't Destroy, Debounce 73

Figuring Out Where Things Go Wrong 82

HOUR 7: `while` Loops 91

Repeat Forever, `while true do` 91

Some Things to Keep in Mind 92

`while` Loops and Scope 98

HOUR 8: `for` Loops 101

How `for` Loops Work 102

Nested Loops 109

Breaking Out of Loops 110

HOUR 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

HOUR 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

HOUR 11: Client Versus Server 145

Understanding the Client and the Server 145
Working with GUIs 146
Understanding RemoteFunctions 149
Using RemoteFunctions 149

HOUR 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

HOUR 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

HOUR 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

HOUR 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 `UpdateAsync` to Update a Data Store 226

HOUR 18: Creating a Game Loop 229
Setting Up Game Loops 229
Working with BindableEvents 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

About the Author

Genevieve Johnson is the senior instructional designer for Roblox, the world's largest user-generated social platform for play. In her role, she oversees creation of educational content and advises educators worldwide on how to use Roblox in STEAM-based learning programs. Her work empowers students to pursue careers as entrepreneurs, engineers, and designers. Prior to Roblox, Johnson was educational content manager for iD Tech, a nationwide tech education program that reaches more than 50,000 students yearly, ages 6-18. While at iD Tech, she helped launch a successful all-girls STEAM program, and her team developed educational content for more than 60 technology-related courses, teaching a variety of subjects from coding to robotics and game design.

We Want to Hear from You!

As the reader of this book, *you* are our most important critic and commentator. We value your opinion and want to know what we're doing right, what we could do better, what areas you'd like to see us publish in, and any other words of wisdom you're willing to pass our way.

You can email or write to let us know what you did or didn't like about this book—as well as what we can do to make our books better.

Please note that we cannot help you with technical problems related to the topic of this book.

When you email, please be sure to include this book's title and author as well as your name, email address, and phone number. We will carefully review your comments and share them with the author and editors who worked on the book.

Email: community@informit.com

Reader Services

Register your copy of *Roblox Game Development in 24 Hours* at www.informit.com/register for convenient access to downloads, updates, and corrections as they become available. To start the registration process, go to informit.com/register and log in or create an account.* Enter the product ISBN (9780136829423) and click Submit.

*Be sure to check the box that you would like to hear from us to receive exclusive discounts on future editions of this product.

This page intentionally left blank

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).

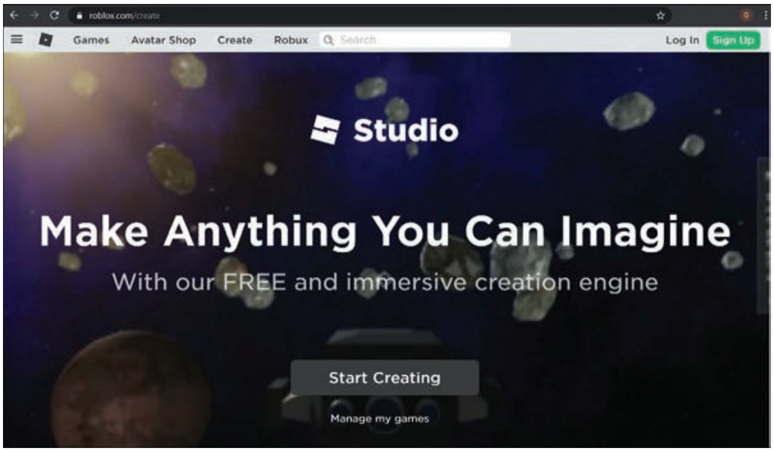


FIGURE 1.1
You need an account to use Roblox Studio. It's free and just a quick sign-up away.

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.

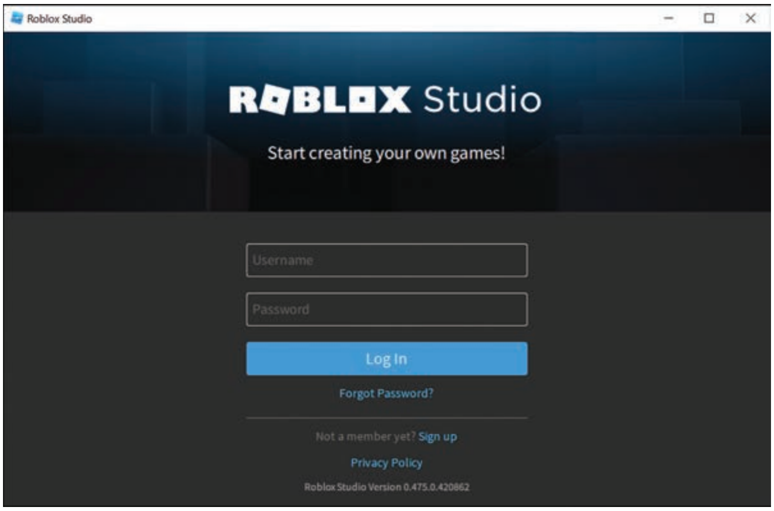

Roblox Studio
ROBLOX Studio
Start creating your own games!
Username
Password
Log In
Forgot Password?
Not a member yet? Sign up
Privacy Policy
Roblox Studio Version 0.475.0.420862

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.

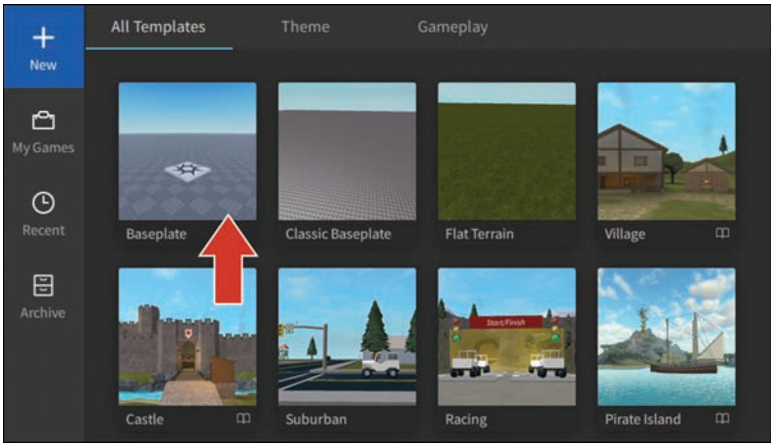


FIGURE 1.3
Studio offers template places you can use as starting points.

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.

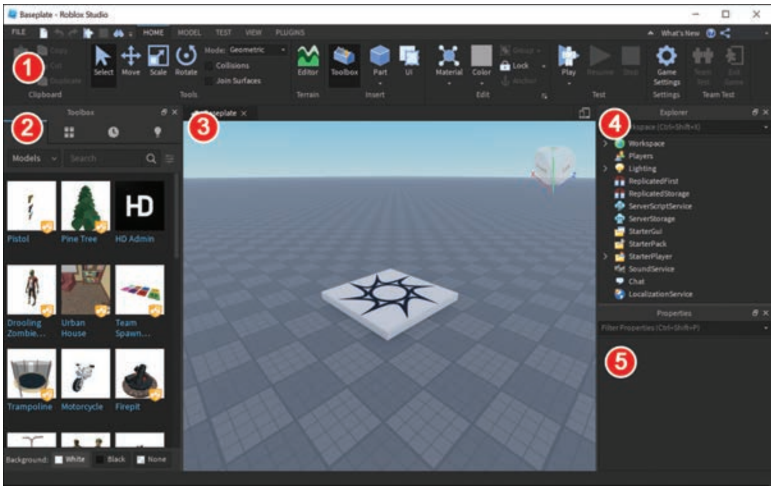


FIGURE 1.4
There are a number of panels, buttons, and lists in the Studio, and you'll quickly become familiar with them.

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.

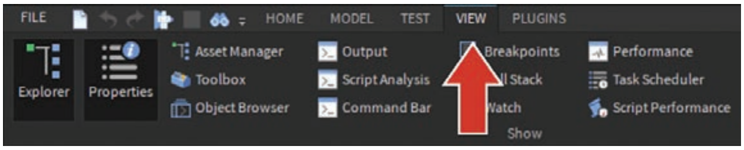


FIGURE 1.5
Use the View tab to control which windows are open.

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

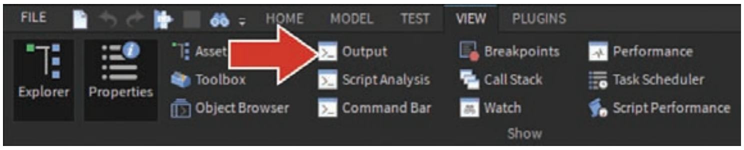
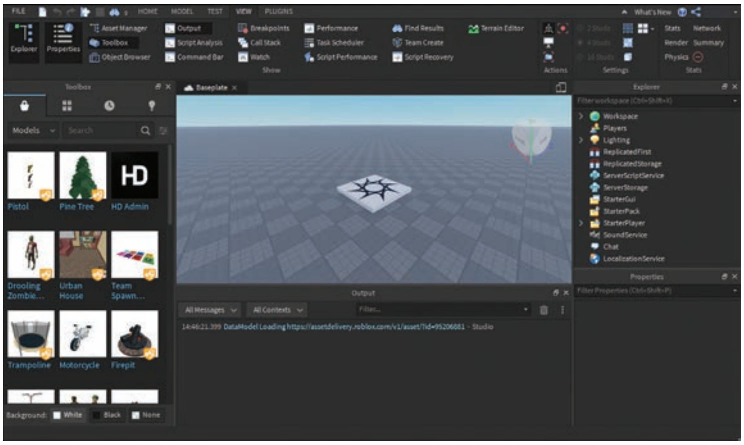


FIGURE 1.6
Click the Output option to open the Output window.



```
14:46:21.399 DataModel Loading https://assetdelivery.roblox.com/v1/asset/?id=95206881 - Studio
```

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.

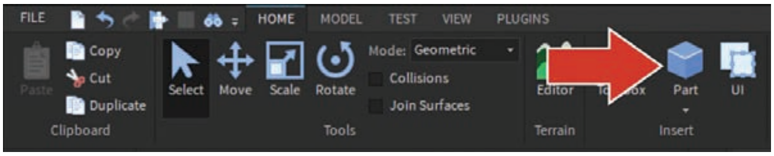


FIGURE 1.8
Click Part on the Home tab to insert a part.

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

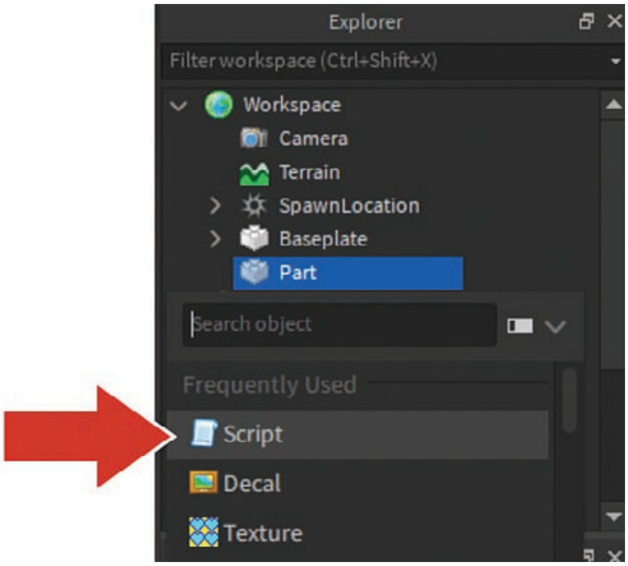


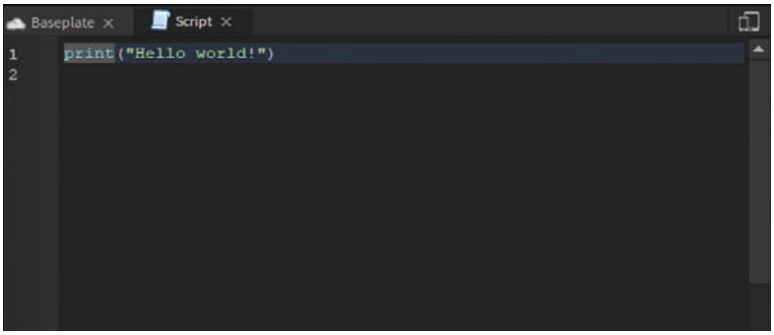
FIGURE 1.9
You use Explorer to insert a script into the part.

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).



```
print("Hello world!")
```

FIGURE 1.10
The window shows the default script and code.

Writing Some Code

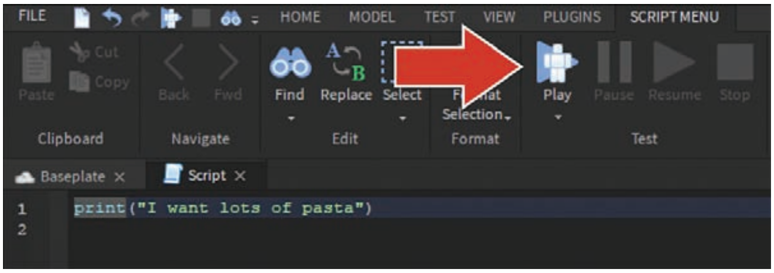
Since the 1970s, `"Hello World!"` has been one of the first pieces of code people have learned. Here it's being used in the print function. Functions are chunks of code that serve a specific purpose. As you learn to code, you'll use prebuilt functions like `print()`, which displays messages in the Output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

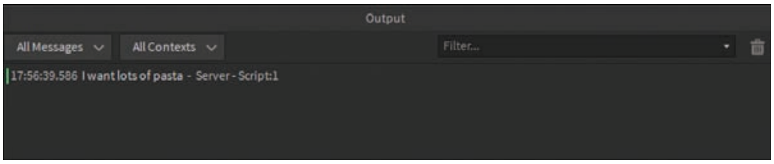
2. To test the code, in the Home tab, click Play (see Figure 1.11).



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

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).



```
17:56:39.586 I want lots of pasta - Server - Script:1
```

FIGURE 1.12
The string is displayed in Output.

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

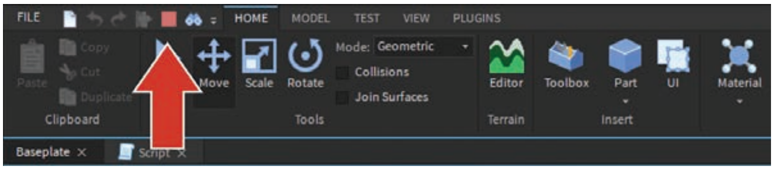
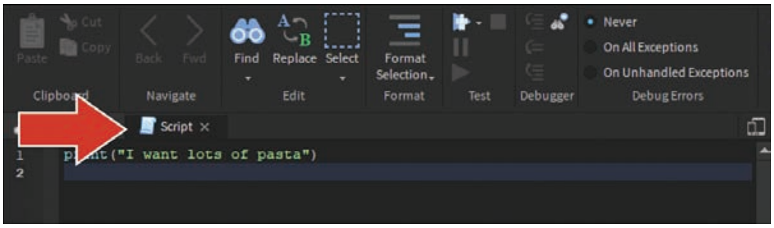


FIGURE 1.13
Click Stop to quit the playtest.

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



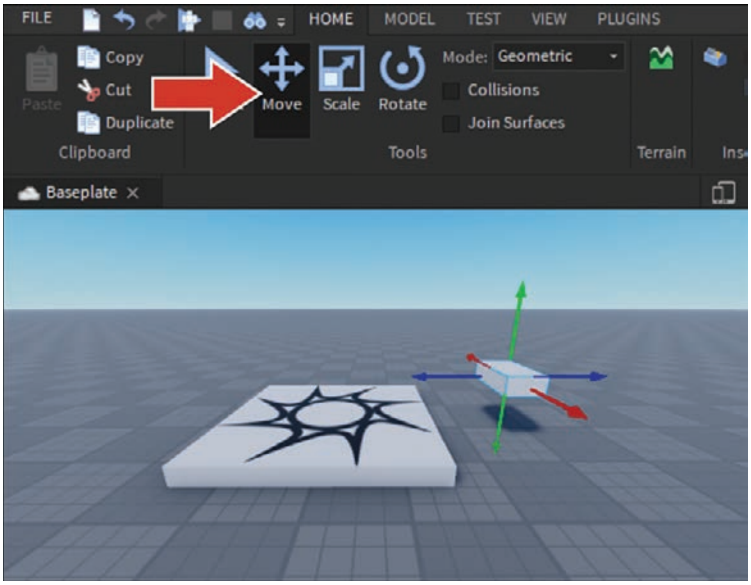
```
print("I want lots of pasta")
```

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:

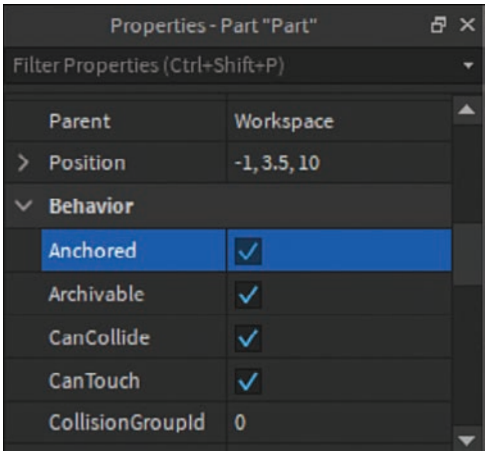
1. Use the Move tool (see Figure 1.15) to move the block off the ground and away from the spawn point. The code you're going to write will destroy anything it touches, and you don't want it to go off prematurely.



Mode: Geometric

FIGURE 1.15
Move the part up and away from the spawn.

2. 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.



Properties - Part "Part"

Filter Properties (Ctrl+Shift+P)

Parent	Workspace
Position	-1, 3.5, 10
Behavior	
Anchored	✓
Archivable	✓
CanCollide	✓
CanTouch	✓
CollisionGroupId	0

FIGURE 1.16
Check Anchored to keep the blocks from falling.

3. 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.

4. Click Play and run up and touch the part.

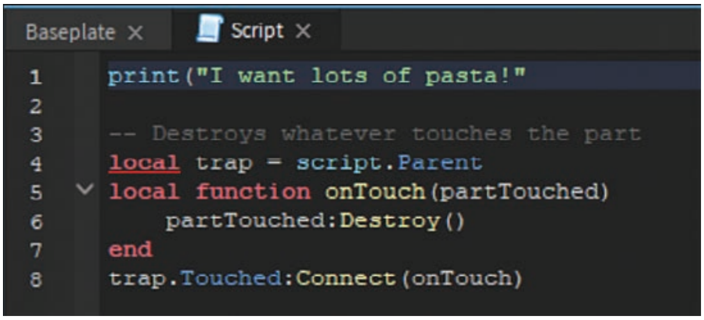
The result should be that your character breaks or parts of your avatar are destroyed. You may notice that this code only destroys what touches it directly, such as your feet. Try jumping on top of the block or brushing against it with just a hand. You'll see only that part of your avatar is destroyed.

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.

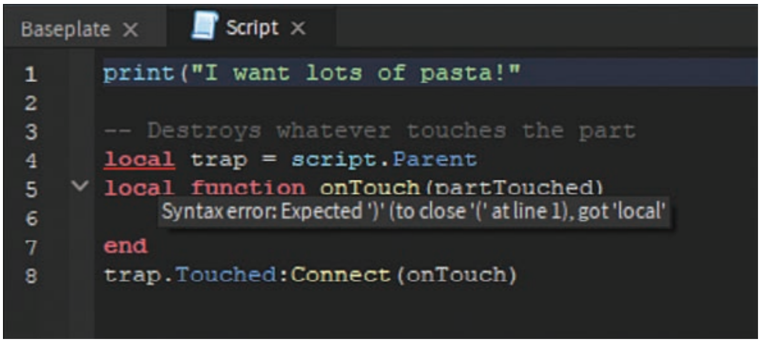


```
print("I want lots of pasta!"

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

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.

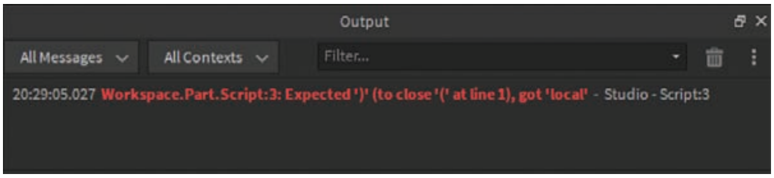


```
print("I want lots of pasta!"

-- Destroys whatever touches the part
local trap = script.Parent
local function onTouch(partTouched)
    Syntax error: Expected ')' (to close '(' at line 1), got 'local'
end
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.



```
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.

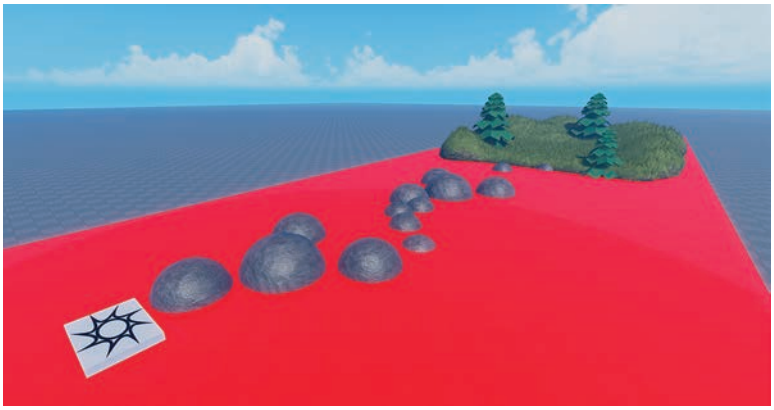


FIGURE 1.20
Use what you've learned so far to create a lava obstacle course.

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.

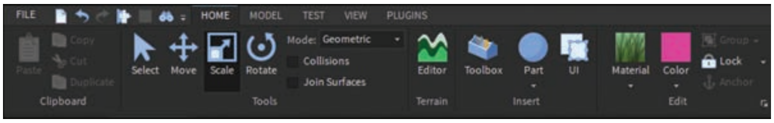


FIGURE 1.21
The Home tab has the tools you need to create and manipulate parts.

- ▶ 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.

Index

Symbols

[] (brackets), in key-value pairs, 128-129

: (colon)
- accessing functions, 68
- for function notation, 281

{} (curly brackets)
- for arrays, 113
- for dictionaries, 128

. (dot operator)
- for dictionary values, 129-130
- for embedded objects, 47
- object hierarchy and, 18-19
- properties and, 20

== (double equal sign) operator, 58

= (equal sign), variable values, 22

>= (greater than or equal to) operator, 59

__index, naming classes, 260

"" (quotation marks), in key-value pairs, 128

3D Editor, 3

3D space
- CFrames, 189
  - offsetting, 191
  - Position property, 190
  - rotating with, 191
  - teleporting exercise, 196-197, 341-342
- models, positioning, 192
- relative jumps example, 194-195
- world versus local coordinates, 193-194
- X, Y, Z coordinates, 187-189

A

abstractions, 183-184

accessing
- Data Stores, 220
- functions, 68
- ModuleScripts, 177-178, 182-183

adding
- class functions, 263-268
- class properties, 261-263
- items to arrays, 114
- key-value pairs to dictionaries, 130-132

algorithms
- for sorting
  - alphabetically, 210-211
  - arrays, 210
  - ascending, 210-212
  - descending, 212-213
  - dictionaries, 213-215, 218, 343
  - mixed data types, 212
  - multiple pieces of information, 216-218, 343
  - numerically, 211-212
- purpose of, 209-210

alphabetical sorts, 210-211

anchoring blocks, 10

and operator, 62

animal sounds example (polymorphism), 279-282

animation
- CFrames, LoadCharacter() function versus, 241
- easing, 325
- tweens
  - chaining, 205-206
  - changing colors, 199-200, 208, 342
  - elevator doors example, 202-205
  - setting parameters for, 201-202
  - TweenService, 199

anonymous functions, 52-55, 328

arguments
- definition of, 43
- mismatched, 51-52
- multiple, 45-49
- value types, 86

arithmetic operators, 324

arrays
- adding items, 114
- converting dictionaries to, 213-215
- creating, 113-114
- finding and removing all specific values, 123
- indexes, 113
  - finding from values, 121
  - retrieving specific values, 114-115
- printing with ipairs() function, 115
- purpose of, 113
- removing items, 122
- searching part of, 123-124
- sorting, 210
  - alphabetically, 210-211
  - ascending, 210-212
  - descending, 212-213
  - mixed data types, 212
  - by multiple pieces of information, 216-218, 343
  - numerically, 211-212
- voting simulator, 133-142

ascending sorts, 210-212

assets, organizing, 231-234

assigning variable values, 41

attributes, 64-67
- checking values, 85
- code reusability and, 79

autocomplete feature, 20

B

Baseplate template, 3

BindableEvents, 230

BindAction() function, 314

BindToRenderStep() function, 303-305

blacklists versus whitelists in raycasting, 310

blocks, anchoring, 10

boolean data type, 22, 36

brackets ([]), in key-value pairs, 128-129

break keyword, 110

bridges
- reactivating, 38-40
- solidifying, 42, 328
- vanishing, 34-36

burning fire, 93-97

buttons
- for placing objects, creating, 302-303
- testing, 170
- viewing/hiding, 320

buying items. *See* monetization; Robux

C

calling functions, 32
with events, 33-36
parent functions, 282
camera, moving, 4, 321
camouflage raycasting example, 288-289
car class example
adding properties, 262-263
property inheritance, 275-277
case-sensitivity of keywords, 19
cashing out Robux, 243
CFrame.Angles() function, 191
CFrames, 189
LoadCharacter() function versus, 241
offsetting, 191
Position property, 190
rotating with, 191
teleporting exercise, 196-197, 341-342
chaining tweens, 205-206
changing
gravity, 233
properties, 25
changing seasons exercise, 125-126, 334
child classes, 271-272
calling parent functions, 282
function inheritance, 278
inheritance setup, 272-274
multiple, 277
polymorphism, 278-282
property inheritance, 274-277
child objects, 18
searching, 223
classes. *See also* child classes; parent classes
calling parent functions, 282
creating, 260-261, 270, 346
functions of, 263-268
inheritance, 271-272
of functions, 278
job roles exercise, 285, 347
multiple child classes, 277
of properties, 274-277
setup, 272-274
naming, 260
polymorphism, 278-282
properties of, 261-263
purpose of, 259
clients, 145
GUIs. *See* GUIs
RemoteEvent object, 161-162
client-to-client communication, 171
client-to-server communication, 165-170
server-to-all-clients communication, 162-165
server-to-single-client communication, 170-171
RemoteFunction object, 149-151
server/client divide, 149
store purchases, 151-158
cloning particle emitters, 100, 330-332
code organization with OOP, 259
collecting firewood, 100, 330-332
colon (:)
accessing functions, 68
for function notation, 281
color picker, 25
colors, changing, 25, 199-200, 208, 342
comments, 12
concatenation, 23
concatenation operator, 325
conditional structures, 57
elseif keyword, 62
else keyword, 63
if/then statements, 58-59
portals, creating, 63-70
configuring passes, 246-249
connect() function, 33
constants, 84
constructors, 260, 265
ContextActionService, 314-316, 320
control variables in for loops, 103, 111
converting dictionaries to arrays, 213-215
coordinates in 3D space
CFrames, 189
offsetting, 191
Position property, 190
rotating with, 191
teleporting exercise, 196-197, 341-342
relative jumps example, 194-195
world versus local, 193-194
X, Y, Z coordinates, 187-189

copying meshes, 78
countdowns, creating with RemoteEvent object, 163-165
crown sales example, 248-255
curly brackets ({})
- for arrays, 113
- for dictionaries, 128

custom leaderboards, 87

D

Damage Over Time (DoT), 111-112, 333
dance floor, creating, 92-93
Data Stores
- accessing, 220
- creating, 220
- enabling, 219
- limiting network calls, 225
- unique key names, 224
- updating, 220-228, 344

data types, 22, 27
- in Lua, 322
- in Roblox Studio, 323

debouncing
- Humanoid objects, 73-75, 88-89, 330
- with ProximityPrompts, 78-79

debugging
- argument value types, 86
- attribute values, 85
- exercise, 88, 329
- string debugging, 82-84
- variable order and placement, 84

decals, inserting, 28-29, 327
descending sorts, 212-213
descriptions, 255
destroy() function, 18-19
detecting mouse input, 314-316
detector exercise, 295, 348
Developer Exchange Program, 243
Developer Products, 256
dictionaries
- converting to arrays, 213-215
- creating, 128
- key-value pairs, 128
  - adding, 130-132
  - formatting keys, 128-129
  - removing, 130-131
  - unique keys, 130
  - value usage, 129-130
- pairs() function, 132-133
- purpose of, 127-128
- sorting, 213-215, 218, 343
- voting simulator, 133-142

direction parameter for raycasting, 289-290
distance, limiting for raycasting, 293
doors, creating for elevator, 202-205
DoT (Damage Over Time), 111-112, 333
dot operator (.)
- for dictionary values, 129-130
- for embedded objects, 47
- object hierarchy and, 18-19
- properties and, 20

double equal sign (==) operator, 58
doubling and halving variables, 85
DRY coding. *See also* OOP
- abstractions, 183-184
- purpose of, 183

E

easing in animation, 325
elevator doors, creating, 202-205
else keyword, 63
elseif keyword, 62
embedded objects, finding in hierarchy, 47
enabling Data Stores, 219
end value in for loops, 103
engagement payouts, 256
equal sign (=), variable values, 22
error messages, 11-12
errors
- list of, 228
- string debugging, 82-84

event connections, order and placement, 138
events
- BindableEvents, 230
- calling functions, 33-36
- RemoteEvent object, 161-162
  - client-to-client communication, 171
  - client-to-server communication, 165-170
  - server-to-all-clients communication, 162-165
  - server-to-single-client communication, 170-171
- Touched, 34-35

exercises
- animating color changes, 208, 342
- anonymous functions, 55, 328
- changing player speed, 72, 328
- changing seasons, 125-126, 334
- cloned particle emitters, 100, 330-332
- collecting firewood, 100, 330-332
- creating NPCs, 29, 327
- debouncing, 88-89, 330
- debugging, 88, 329
- detector with raycasting, 295, 348
- dictionary sorting, 218, 343
- DoT (Damage Over Time), 111-112, 333
- inserting decals, 28-29, 327
- job roles, 285, 347
- loops, 112, 333-334
- map choice announcement, 172, 338-340
- NPC person class, 270, 346
- obstacle course, 14-15, 326
- pass creation, 257, 346
- placing objects, 311, 350
- player announcements, 242, 345
- price lists, 160, 336-338
- rotating objects, 320, 351
- solidifying bridges, 42, 328
- solutions to, 326-351
- team assignments, 143, 334
- teleporting with CFrames, 196-197, 341-342
- traps with ModuleScripts, 185, 340-341
- updating player information, 228, 344

Explorer window, 3

explosion script, 9-11

F

false conditions, loops for, 98

files, saving, 13

filtering
- lists, 7
- objects for raycasting, 294

finding
- all specific array values, 123
- array indexes from values, 121
- embedded objects in hierarchy, 47
- list items, 7

fire
- burning, 93-97
- collecting firewood, 100, 330-332

folders, modifying items
- with for loops, 116-121
- with ipairs() function, 116

for loops, 98, 101-102
- default increment, 105
- examples and exercises, 105-106, 112, 333-334
- finding and removing all specific array values, 123
- generic, 115
- *i* in, 111
- numeric, 123-124
- printing arrays, 115
- searching part of arrays, 123-124
- turning lights on/off, 116-121
- values in, 102-105

formatting dictionary keys, 128-129

functions
- accessing, 68
- anonymous, 52-55, 328
- arguments
  - definition of, 43
  - mismatched, 51-52
  - multiple, 45-49
  - value types, 86
- BindAction(), 314
- BindToRenderStep(), 303-305
- calling, 32
  - with events, 33-36
  - from parent classes, 282
- CFrame.Angles(), 191
- of classes, 263-268
- connect(), 33
- constructors, 260, 265
- creating, 31-32
- definition of, 31
- destroy(), 18-19
- GetAsync(), 220, 225
- IncrementAsync(), 227
- inheriting, 278
- insert(), 114

ipairs()
  finding array indexes, 121
  with folders, 116
  pairs versus, 142
  printing arrays, 115
LoadCharacter(), 241
as methods, 33
in ModuleScripts
  accessing, 177-178
  adding, 175-176
  scope, 176
MoveTo(), 264
multiple in scripts, 41
named, 52-55, 328
naming conventions, 32, 35, 69
new(), 26
order and placement, 36-40
paint(), 44-48
pairs()
  with dictionaries, 132-133
  ipairs() versus, 142
parameters
  creating, 43-45
  definition of, 43
  maximum, 54
  mismatched, 51-52
  multiple, 45-49
pcall(), 225
polymorphism, 278-282
print(), 7-9, 23, 43
  for debugging, 82-84
RemoteFunction object, 149-151, 159
remove(), 122
require(), 177
return values
  definition of, 49
  multiple, 50, 80
  nil, 51
scope, 33, 37-38
SetAsync(), 220, 225
table.sort(), 210-213
tostring(), 212
UnbindAction(), 314
UpdateAsync(), 226-227
wait(), 42-43, 201
  default value, 86
  with while loops, 92-93
workspace:Raycast()
  camouflage example, 288-289
  direction parameter, 289-290
  limiting distance, 293
  setup, 287-288

G

game loops
  BindableEvents in, 230
  creating, 231-240
  for player announcements, 242, 345
  purpose of, 229-230
gameplay, moving camera in, 321
generic for loops, 115
GetAsync() function, 220, 225
global coordinates, local versus, 193-194
global variables, 22, 41
glowing lights, 120
goal value in for loops, 103
gold ore script (mining simulator), 79-82
gold ore setup (mining simulator), 78-79
graphical user interfaces. *See* GUIs
gravity, changing, 233
greater than or equal to (>=) operator, 59
grouping parts, 166, 192
GUIs (graphical user interfaces)
  creating, 106-109, 146-148, 335
  customizing, 147
  moving, 154
  purpose of, 146
  script placement, 148

H

Hello World! script, 7-9
hiding buttons, 320
hierarchy (of objects), 18
  finding embedded objects, 47
  instances, 26
  IntValue objects, 77
  naming conventions, 24
  properties, 20-22
    changing, 25
    data types for, 22, 27
  variables and, 28

Humanoid objects, 59-61
- changing player speed, 72, 328
- debouncing, 73-75, 88-89, 330
- VectorForce objects, adding, 179-182

HumanoidRootPart, MoveTo() function and, 264

I

***i* as control variable, 111**

if/then statements, 58-59

ignoring objects in raycasting, 290-293

IncrementAsync() function, 227

increment value in for loops, 103-105

indenting code, 32

indexes, 113
- finding from values, 121
- key-value pairs versus, 129
- retrieving specific values, 114-115

in-game purchases. *See* monetization; Robux

inheritance, 271-272
- of functions, 278
- job roles exercise, 285, 347
- multiple child classes, 277
- overriding, 278-282
- of properties, 274-277
- setup, 272-274

insert() function, 114

inserting
- decals, 28-29, 327
- scripts into parts, 6-7

installing Roblox Studio, 1-2

instances, 26

IntValue objects, 77

ipairs() function
- finding array indexes, 121
- with folders, 116
- pairs() versus, 142
- printing arrays, 115

iterations, 105

J–K

job roles exercise, 285, 347

jump pads
- creating, 178-183
- relative jumps with, 194-195

keys
- for moving camera, 321
- uniqueness in Data Stores, 224

key-value pairs, 128
- adding, 130-132
- in Data Stores, accessing, 220
- formatting keys, 128-129
- indexes versus, 129
- removing, 130-131
- unique keys, 130
- value usage, 129-130

keywords, 19-20
- break, 110
- case-sensitivity, 19
- else, 63
- elseif, 62
- nil, 51
- reserved names, 322
- return, 49-50
- script, 20
- type, 217
- workspace, 19

L

leaderboards
- creating, 75-77, 87
- maximum number of stats, 87
- value types, 86-87

leaderstats folder, 77

length operator, 325

lights
- colors, changing via tweens, 208, 342
- glowing, 120
- SpotLight objects, 117
- turning on/off, 116-121

limiting
- distance for raycasting, 293
- network calls, 225

lists, filtering, 7

LoadCharacter() function, 241

load times for scripts, 109

local object coordinates, world versus, 193-194

LocalScript object, 148, 154-155

local variables, 22, 184

logging in to Roblox Studio, 2

logical operators, 62-63, 324

loops

- break keyword, 110
- exercises, 112, 333-334
- for false conditions, 98
- for, 98, 101-102
  - default increment, 105
  - examples, 105-106
  - finding and removing all specific array values, 123
  - generic, 115
  - *i* in, 111
  - numeric, 123-124
  - printing arrays, 115
  - searching part of arrays, 123-124
  - turning lights on/off, 116-121
  - values in, 102-105
- game loops
  - BindableEvents in, 230
  - creating, 231-240
  - for player announcements, 242, 345
  - purpose of, 229-230
- nested, 109-110
- repeat until, 237
- while, 91-92
  - with ProximityPrompts, 93-97
  - scope, 98
  - with wait() function, 92-93

Lua, 1

- arrays
  - adding items, 114
  - converting dictionaries to, 213-215
  - creating, 113-114
  - finding and removing all specific values, 123
  - indexes, 113-115, 121
  - printing with ipairs() function, 115
  - purpose of, 113
  - removing items, 122
  - searching part of, 123-124
  - sorting, 210-213, 216-218, 343
  - voting simulator, 133-142
- classes. *See also* child classes; parent classes
  - calling parent functions, 282
  - creating, 260-261, 270, 346
  - function inheritance, 278
  - functions of, 263-268
  - inheritance, 271-274, 285, 347
  - multiple child classes, 277
  - naming, 260
  - polymorphism, 278-282
  - properties of, 261-263
  - property inheritance, 274-277
- conditional structures, 57
  - elseif keyword, 62
  - else keyword, 63
  - if/then statements, 58-59
  - portals, creating, 63-70
- data types, 22, 27, 322
- debugging
  - argument value types, 86
  - attribute values, 85
  - exercise, 88, 329
  - string debugging, 82-84
  - variable order and placement, 84
- dot operator
  - for dictionary values, 129-130
  - for embedded objects, 47
  - object hierarchy and, 18-19
  - properties and, 20
- functions. *See* functions
- keywords, 19-20
- loops
  - break keyword, 110
  - exercises, 112, 333-334
  - for false conditions, 98
  - for, 98, 101-106, 111, 115-124
  - game loops, 229-242, 345
  - nested, 109-110
  - repeat until, 237
  - while, 91-98
- ModuleScripts
  - accessing in scripts, 177-178, 182-183
  - code structure, 174
  - creating, 234-237
  - DRY coding and, 183
  - functions and variables in, 175-176
  - jump pad example, 179-182

naming, 174-175
placing, 174
purpose of, 173
scope in, 176
trap exercise, 185, 340-341
naming conventions, list of, 325
object hierarchy, 18
finding embedded objects, 47
instances, 26
IntValue objects, 77
naming conventions, 24, 260
properties, 20-22, 25, 27
variables and, 28
operators
arithmetic, 324
concatenation, 325
double equal sign (==), 58
greater than or equal to (>=), 59
length, 325
logical, 62-63, 324
most common, 58
purpose of, 324
relational, 324
reserved names, 322
scripts, 6
autocomplete feature, 20
comments, 12
DRY coding, 183-184
error messages, 11-12
explosion example, 9-11
GUI script placement, 148
Hello World!, 7-9
indenting code, 32
inserting into parts, 6-7
load times, 109
for mining simulator, 79-82
multiple functions in, 41
opening, 13
order and placement in, 36-40
renaming, 18-19
saving, 13
strings, 7
variables
combining with strings, 23
creating, 22-25
naming conventions, 24
properties and, 28
updating, 23

M

map choice announcement exercise, 172, 338-340
map pickers, creating, 166-170
meshes, copying, 78
messages
receiving on server, 317-319
sending to server, 316
methods. *See* functions
mining simulator, 75
gold ore script, 79-82
gold ore setup, 78-79
leaderboard, creating, 75-77
mismatched arguments/ parameters, 51-52
mixed data types, sorting, 212
models
creating, 192
grouping parts into, 166
positioning, 192
modifying folder items
with for loops, 116-121
with ipairs(), 116
ModuleScripts
accessing in scripts, 177-178, 182-183
code structure, 174
creating, 234-237
DRY coding and, 183
functions and variables in, 175-176
jump pad example, 179-182
naming, 174-175
placing, 174
purpose of, 173
scope in, 176
trap exercise, 185, 340-341
monetization. *See also* Robux
Developer Products, 256
engagement payouts, 256
ideas for, 256
passes
checking for ownership, 252-255
configuring, 246-249
creating, 244-245, 257, 346
crown sales example, 248-255
prompting purchases, 247-250
purpose of, 243
testing, 251-252
updating, 245

mouse input, detecting, 314-316
mouse movements, tracking, 303-306
BindToRenderStep() function, 303-305
raycasting from mouse, 305-306
MoveTo() function, 264
moving
camera, 4, 321
GUIs, 154
multiple arguments, 45-49
multiple child classes, 277
multiple functions in scripts, 41
multiple parameters, 45-49
multiple pieces of information, sorting by, 216-218, 343
multiple player interactions, variables for, 70
multiple players, testing for, 138-139
multiple return values, 80

N

named functions, 52-55, 328
naming
classes, 260
ModuleScripts, 174-175
objects, 260
naming conventions, 24, 32, 35, 69
constants, 84
constructors, 265
list of, 325
nested loops, 109-110
network calls, 225
Network Simulator, testing for multiple people, 138-139
new() function, 26
nil keyword, 51, 157
not operator, 62
NPCs (Non Playable Characters), 17
adding face to, 28-29, 327
creating, 23-25, 29, 327
exercise, 270, 346
number data type, 22
numbers, sorting with strings, 212
numeric for loops, 123-124
numerical sorts, 211-212

O

object hierarchy, 18
finding embedded objects, 47
instances, 26
IntValue objects, 77
naming conventions, 24
properties, 20-22
changing, 25
data types for, 22, 27
variables and, 28
object-oriented programming. *See* OOP (object-oriented programming)
objects
filtering for raycasting, 294
ignoring in raycasting, 290-293
naming, 260
placing, 297-298, 313. *See also* 3D space
creating button for, 302-303
detecting mouse input, 314-316
with other object coordinates, 190
previewing placement, 307-309
receiving messages on server, 317-319
second object exercise, 311, 350
sending messages to server, 316
setup, 298-301
tracking mouse movements, 303-306
purpose of, 259
rotating
with CFrames, 191
while placing, 320, 351
obstacle course exercise, 14-15, 326
offsetting CFrames, 191
one-time purchases. *See* passes
OOP (object-oriented programming)
classes. *See also* child classes; parent classes
calling parent functions, 282
creating, 260-261, 270, 346
function inheritance, 278
functions of, 263-268
inheritance, 271-274, 285, 347

multiple child classes, 277
naming, 260
polymorphism, 278-282
properties of, 261-263
property inheritance, 274-277
purpose of, 259
code organization with, 259
objects
filtering for raycasting, 294
ignoring in raycasting, 290-293
naming, 260
placing. *See* placing, objects
purpose of, 259
rotating, 191, 320, 351
opening
Output window, 5
scripts, 13
operating system requirements, 13
operators
arithmetic, 324
concatenation, 325
double equal sign (==), 58
greater than or equal to (>=), 59
length, 325
logical, 62-63, 324
most common, 58
purpose of, 324
relational, 324
organizing
assets, 231-234
with OOP, 259
variables, 305
or operator, 62
Output window, opening, 5
overriding inheritance, 278-282

P

paint() function, 44-48
pairs() function
with dictionaries, 132-133
ipairs() versus, 142
parameters
creating, 43-45
definition of, 43
maximum, 54
mismatched, 51-52
multiple, 45-49
setting for raycasting, 290-293
for tweens, 201-202
parent classes, 271-272
calling parent functions from child classes, 282
inheritance setup, 272-274
parent objects, 18
ParticleEmitter objects, 72
particle emitters, cloning, 100, 330-332
parts, 6
colors, changing via tweens, 199-200
creating instances, 26
grouping, 166, 192
initial location, 26
inserting scripts into, 6-7
ProximityPrompts for, 78-79
textures, showing activation, 67
Touched event, 34-35
passes
checking for ownership, 252-255
configuring, 246-249
creating, 244-245, 257, 346
crown sales example, 248-255
prompting purchases, 247-250
purpose of, 243
testing, 251-252
updating, 245
pcall() function, 225
pet class example, adding functions, 264-268
placing
models, 192
ModuleScripts, 174
objects, 297-298, 313. *See also* 3D space
creating button for, 302-303
detecting mouse input, 314-316
with other object coordinates, 190
previewing placement, 307-309
receiving messages on server, 317-319
rotating while, 320, 351
second object exercise, 311, 350
sending messages to server, 316

setup, 298-301
tracking mouse movements, 303-306
player announcements exercise, 242, 345
playerID, saving data with, 226
player management, services for, 237-240
playtesting
changes during, 12
for multiple players, 138-139
references, checking, 165
scripts, 7-8
polymorphism, 278-282
portals, creating, 63-70
positioning. *See* placing
Position property (CFrames), 190
previewing object placement, 307-309
price list exercise, 160, 336-338
PrimaryParts (models), 192
print() function, 7-9, 23, 43
for debugging, 82-84
printing arrays with ipairs() function, 115
prompting in-game purchases, 247-250
properties, 20-22
changing, 25
of classes, 261-263
data types for, 22, 27
inheriting, 274-277
variables and, 28
Properties window, 3
protected calls, 225
ProximityPrompt objects, 64, 67-70
debouncing with, 78-79
with ServerScriptService, 79-82
viewing, 268
with while loops, 93-97
ProximityPromptService, 68-70
purchases. *See* monetization; Robux

Q–R

quotation marks ("") in key-value pairs, 128

raycasting
camouflage example, 288-289
detector exercise, 295, 348
direction parameter, 289-290
filtering objects, 294
from mouse, 305-306
function setup, 287-288
limiting distance, 293
purpose of, 287
setting parameters, 290-293
through windows, 292-293
whitelists versus blacklists, 310
reactivating bridges, 38-40
receiving messages on server, 317-319
red lines in editor, 11-12
references, checking, 165
relational operators, 324
relative jumps, creating, 194-195
RemoteEvent object, 161-162
client-to-client communication, 171
client-to-server communication, 165-170
server-to-all-clients communication, 162-165
server-to-single-client communication, 170-171
RemoteFunction object, 149-151, 159
remove() function, 122
removing
all specific array values, 123
items from arrays, 122
key-value pairs from dictionaries, 130-131
renaming scripts, 18-19
renderstep, 303-305
repeat until loops, 237
require() function, 177
reserved names, 322
resources for information, 13, 319
retrieving specific array values, 114-115
return keyword, 49-50
return values
definition of, 49
multiple, 50, 80
nil, 51
returning table values, 133
Roblox Premium
engagement payouts, 256
monetization and, 247
Roblox Studio, 1
blocks, anchoring, 10
camera controls, 4

data types, 323
files, saving, 13
GUIs
    creating, 106-109, 146-148, 335
    customizing, 147
    moving, 154
    purpose of, 146
    script placement, 148
Humanoid objects, 59-61
    changing player speed, 72, 328
    debouncing, 73-75, 88-89, 330
    VectorForce objects, adding, 179-182
installing, 1-2
leaderboards
    creating, 75-77, 87
    maximum number of stats, 87
    value types, 86-87
leaderstats folder, 77
logging in, 2
moving camera in, 321
object hierarchy, 18
    finding embedded objects, 47
    instances, 26
    IntValue objects, 77
    naming conventions, 24
    properties, 20-22, 25-27
    variables and, 28
operating system requirements, 13
Output window, opening, 5
parts, 6
    colors, changing via tweens, 199-200
    creating instances, 26
    grouping, 166, 192
    initial location, 26
    inserting scripts into, 6-7
    ProximityPrompts for, 78-79
    textures, showing activation, 67
    Touched event, 34-35
red lines in editor, 11-12
resources for information, 13, 319
user interface, 2-4
Robux. *See also* monetization
    cashing out, 243
    engagement payouts, 256
    uses for, 243
rotating objects, 320, 351
    with CFrames, 191
Run command, testing code, 48

S

saving data
    in Data Stores
        accessing, 220
        creating, 220
        enabling, 219
        limiting network calls, 225
        unique key names, 224
        updating, 220-228, 344
    methods of, 227
    with playerID, 226
saving scripts, 13
scope
    of functions, 33, 37-38
    in ModuleScripts, 176
    of variables, 41
    of while loops, 98
ScreenGui object, 146-147
script keyword, 20
scripts, 6
    arrays
        adding items, 114
        converting dictionaries to, 213-215
        creating, 113-114
        finding and removing all specific values, 123
        indexes, 113-115, 121
        printing with ipairs() function, 115
        purpose of, 113
        removing items, 122
        searching part of, 123-124
        sorting, 210-213, 216-218, 343
        voting simulator, 133-142
    autocomplete feature, 20
    comments, 12
    conditional structures, 57
        elseif keyword, 62
        else keyword, 63
        if/then statements, 58-59
        portals, creating, 63-70
    DRY coding
        abstractions, 183-184
        purpose of, 183
    error messages, 11-12

explosion example, 9-11
functions. *See* functions
GUI script placement, 148
Hello World!, 7-9
indenting code, 32
inserting into parts, 6-7
load times, 109
loops
  break keyword, 110
  exercises, 112, 333-334
  for false conditions, 98
  for, 98, 101-106, 111, 115-124
  game loops, 229-242, 345
  nested, 109-110
  repeat until, 237
  while, 91-98
for mining simulator, 79-82
ModuleScripts
  accessing, 177-178, 182-183
  code structure, 174
  creating, 234-237
  functions and variables in, 175-176
  jump pad example, 179-182
  naming, 174-175
  placing, 174
  purpose of, 173
  scope in, 176
  trap exercise, 185, 340-341
multiple functions in, 41
opening, 13
order and placement in, 36-40
renaming, 18-19
saving, 13
ServerScriptService, 156-158

searching
  child objects, 223
  part of arrays, 123-124
seasons, changing, 125-126, 334
self, as naming convention, 260
sending messages to server, 316
servers, 145
  receiving messages, 317-319
  RemoteEvent object, 161-162
    client-to-server communication, 165-170
    server-to-all-clients communication, 162-165
    serve-to-single-client communication, 170-171
  RemoteFunction object, 149-151
  sending messages to, 316
  server/client divide, 149
  store purchases, 151-158
ServerScriptService, 76, 79-82, 156-158
services
  ContextActionService, 314-316, 320
  definition of, 68
  player management, 237-240
  ProximityPromptService, 68-70
  ServerScriptService, 76, 79-82, 156-158
SetAsync() function, 220, 225
solidifying bridges, 42, 328
solutions to exercises, 326-351
  animating color changes, 342
  anonymous functions, 328
  changing player speed, 328
  changing seasons, 334
  cloned particle emitters, 330-332
  collecting firewood, 330-332
  creating NPCs, 327
  debouncing, 330
  debugging, 329
  detector with raycasting, 348
  dictionary sorting, 343
  DoT (Damage Over Time), 333
  inserting decals, 327
  job roles, 347
  loops, 333-334
  map choice announcement, 338-340
  NPC person class, 346
  obstacle course, 326
  pass creation, 346
  placing objects, 350
  player announcements, 345
  price lists, 336-338
  rotating objects, 351
  solidifying bridges, 328
  team assignments, 334
  teleporting with CFrames, 341-342
  traps with ModuleScripts, 340-341
  updating player information, 344
sorting
  arrays, 210
    alphabetically, 210-211
    ascending, 210-212
    descending, 212-213

mixed data types, 212
by multiple pieces of information, 216-218, 343
numerically, 211-212
dictionaries, 213-215, 218, 343

SpeedBoost tweaks, 85

speed of players, changing, 72, 328

SpotLight objects, 117

StarterGUI object, 146

storage
for BindableEvents, 230
for ModuleScripts, 174

store purchases, 151-158

string debugging, 82-84

strings, 7, 22
combining with variables, 23
sorting with numbers, 212

Studio. *See* Roblox Studio

SurfaceGui objects, 106-108

T

tables, 22
arrays
adding items, 114
converting dictionaries to, 213-215
creating, 113-114
finding and removing all specific values, 123
indexes, 113-115, 121
printing with ipairs() function, 115
purpose of, 113
removing items, 122
searching part of, 123-124
sorting, 210-213, 216-218, 343
voting simulator, 133-142
dictionaries
converting to arrays, 213-215
creating, 128
key-value pairs, 128-132
pairs() function, 132-133
purpose of, 127-128
sorting, 213-215, 218, 343
voting simulator, 133-142
purpose of, 113
returning values, 133

table.sort() function, 210-213

team assignments exercise, 143, 334

teleporting exercise, 196-197, 341-342

templates, Baseplate, 3

testing
buttons, 170
changes during, 12
for multiple players, 138-139
passes, 251-252
references, checking, 165
with Run command, 48
scripts, 7-8

TextLabel objects, 106-109

textured parts, showing activation, 67

Toolbar ribbon, 3

Toolbox, 3

tostring() function, 212

Touched event, 34-35

tracking mouse movements, 303-306
BindToRenderStep() function, 303-305
raycasting from mouse, 305-306

traps exercise, 185, 340-341

turning lights on/off, 116-121

tweens
chaining, 205-206
changing colors, 199-200, 208, 342
elevator doors example, 202-205
setting parameters for, 201-202
TweenService, 199

TweenService, 199

type keyword, 217

U

UnbindAction() function, 314

unique keys
in Data Stores, 224
in dictionaries, 130

UpdateAsync() function, 226-227

updating
Data Stores, 220-228, 344
passes, 245
variables, 23

user interface for Roblox Studio, 2-4

V

vanishing bridges, 34-36

variables

- combining with strings, 23
- creating, 22-25
  - without assigning value, 41
- doubling and halving, 85
- in for loops, 102-105
- local, 184
- in ModuleScripts
  - accessing, 177-178
  - adding, 175-176
  - scope, 176
- for multiple player interactions, 70
- naming conventions, 24
- order and placement, 36-40, 45, 84
- organizing, 305
- properties and, 28
- scope, 41
- updating, 23

VectorForce objects, adding to humanoids, 179-182

vehicle class example (property inheritance), 275-277

viewing

- buttons, 320
- ProximityPrompts, 268

voting simulator, 133-142

W

wait() function, 42-43, 201

- default value, 86
- with while loops, 92-93

WET coding, 183

while loops, 91-92

- exercises, 112, 333-334
- with ProximityPrompts, 93-97
- scope, 98
- with wait() function, 92-93

whitelists versus blacklists in raycasting, 310

windows

- raycasting through, 292-293
- in Roblox Studio, opening, 5

workspace keyword, 19

workspace:Raycast() function

- camouflage example, 288-289
- direction parameter, 289-290
- limiting distance, 293
- setup, 287-288

world coordinates, local versus, 193-194

writing scripts, 7-9

X–Z

X coordinates, 187-189

Y coordinates, 187-189

Z coordinates, 187-189