# Table of Contents (the summary)

<table>
<thead>
<tr>
<th>Intro</th>
<th>xxxi</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Start building with C#: Building something great... fast!</td>
</tr>
<tr>
<td>2</td>
<td>Dive into C#: Statements, classes, and code</td>
</tr>
<tr>
<td>3</td>
<td>Unity Lab 1: Explore C# with Unity</td>
</tr>
<tr>
<td>4</td>
<td>Objects... get oriented: Making code make sense</td>
</tr>
<tr>
<td>5</td>
<td>Types and references: Getting the reference</td>
</tr>
<tr>
<td>6</td>
<td>Unity Lab 2: Write C# Code for Unity</td>
</tr>
<tr>
<td>7</td>
<td>Encapsulation: Keep your privates... private</td>
</tr>
<tr>
<td>8</td>
<td>Inheritance: Your object’s family tree</td>
</tr>
<tr>
<td>9</td>
<td>Unity Lab 3: GameObject Instances</td>
</tr>
<tr>
<td>10</td>
<td>Interfaces, casting, and “is”: Making classes keep their promises</td>
</tr>
<tr>
<td>11</td>
<td>Enums and collections: Organizing your data</td>
</tr>
<tr>
<td>12</td>
<td>Unity Lab 4: User Interfaces</td>
</tr>
<tr>
<td>13</td>
<td>LINQ and lambdas: Get control of your data</td>
</tr>
<tr>
<td>14</td>
<td>Reading and writing files: Save the last byte for me!</td>
</tr>
<tr>
<td>15</td>
<td>Unity Lab 5: Raycasting</td>
</tr>
<tr>
<td>16</td>
<td>Captain Amazing: The Death of the Object</td>
</tr>
<tr>
<td>17</td>
<td>Exception handling: Putting out fires gets old</td>
</tr>
<tr>
<td>18</td>
<td>Unity Lab 6: Scene Navigation</td>
</tr>
<tr>
<td>19</td>
<td>Downloadable exercise: Animal match boss battle</td>
</tr>
<tr>
<td>ii</td>
<td>Visual Studio for Mac Learner’s Guide</td>
</tr>
<tr>
<td>i</td>
<td>Code Kata: A learning guide for advanced and impatient readers</td>
</tr>
</tbody>
</table>

# Table of Contents (the real thing)

## Intro

**Your brain on C#.** You’re sitting around trying to learn something, but your brain keeps telling you all that learning isn’t important. Your brain’s saying, “Better leave room for more important things, like which wild animals to avoid and whether nude archery is a bad idea.” So how do you trick your brain into thinking that your life really depends on learning C#?

- Who is this book for? xxx
- We know what you’re thinking. xxxi
- And we know what your brain is thinking. xxxi
- Metacognition: thinking about thinking xxxiii
- Here’s what WE did xxxiv
- Here’s what YOU can do to bend your brain into submission xxxv
- README xxxvi
- The technical review team xli
- Acknowledgments xlii
- O’Reilly online learning xlii
start building with C#

Build something great...fast!

Want to build great apps...right now?

With C#, you’ve got a modern programming language and a valuable tool at your fingertips. And with Visual Studio, you’ve got an amazing development environment with highly intuitive features that make coding as easy as possible. Not only is Visual Studio a great tool for writing code, it’s also a really valuable learning tool for exploring C#. Sound appealing? Turn the page, and let’s get coding.

Why you should learn C#  2
Visual Studio is a tool for writing code and exploring C#  3
Create your first project in Visual Studio  4
Let’s build a game!  6
Here’s how you’ll build your game  7
Create a WPF project in Visual Studio  8
Use XAML to design your window  12
Design the window for your game  13
Set the window size and title with XAML properties  14
Add rows and columns to the XAML grid  16
Make the rows and columns equal size  17
Add a TextBlock control to your grid  18
Now you’re ready to start writing code for your game  21
Generate a method to set up the game  22
Finish your SetUpGame method  24
Run your program  26
Add your new project to source control  30
The next step to build the game is handling mouse clicks  33
Make your TextBlocks respond to mouse clicks  34
Add the TextBlock_MouseDown code  37
Make the rest of the TextBlocks call the same MouseDown event handler  38
Finish the game by adding a timer  39
Add a timer to your game’s code  40
Use the debugger to troubleshoot the exception  42
Add the rest of the code and finish the game  46
Update your code in source control  47
dive into C#

Statements, classes, and code

You’re not just an IDE user. You’re a developer.

You can get a lot of work done using the IDE, but there’s only so far it can take you. Visual Studio is one of the most advanced software development tools ever made, but a powerful IDE is only the beginning. It’s time to dig into C# code: how it’s structured, how it works, and how you can take control of it...because there’s no limit to what you can get your apps to do.

- Let’s take a closer look at the files for a console app .......................... 50
- Two classes can be in the same namespace (and file!) .......................... 52
- Statements are the building blocks for your apps ................................. 55
- Your programs use variables to work with data .................................. 56
- Generate a new method to work with variables .................................. 58
- Add code that uses operators to your method ..................................... 59
- Use the debugger to watch your variables change ............................. 60
- Use operators to work with variables .................................................. 62
- “if” statements make decisions ......................................................... 63
- Loops perform an action over and over .............................................. 64
- Use code snippets to help write loops .............................................. 67
- Controls drive the mechanics of your user interfaces ......................... 71
- Create a WPF app to experiment with controls ................................ 72
- Add a TextBox control to your app ................................................... 75
- Add C# code to update the TextBlock .............................................. 78
- Add an event handler that only allows number input .......................... 79
- Add sliders to the bottom row of the grid ........................................ 83
- Add C# code to make the rest of the controls work ............................ 84
Welcome to your first Head First C# Unity Lab. Writing code is a skill, and like any other skill, getting better at it takes practice and experimentation. Unity will be a really valuable tool for that. In this lab, you can begin practicing what you’ve learned about C# in Chapters 1 and 2.

- Unity is a powerful tool for game design 88
- Download Unity Hub 89
- Use Unity Hub to create a new project 90
- Take control of the Unity layout 91
- Your scene is a 3D environment 92
- Unity games are made with GameObjects 93
- Use the Move Gizmo to move your GameObjects 94
- The Inspector shows your GameObject’s components 95
- Add a material to your Sphere GameObject 96
- Rotate your sphere 99
- Get creative! 102
objects...get oriented!

Making code make sense

Every program you write solves a problem.

When you’re building a program, it’s always a good idea to start by thinking about what problem your program’s supposed to solve. That’s why objects are really useful. They let you structure your code based on the problem it’s solving so that you can spend your time thinking about the problem you need to work on rather than getting bogged down in the mechanics of writing code. When you use objects right—and really put some thought into how you design them—you end up with code that’s intuitive to write, and easy to read and change.

If code is useful, it gets reused 104
Some methods take parameters and return a value 105
Let’s build a program that picks some cards 106
Create your PickRandomCards console app 107
Finish your PickSomeCards method 108
Your finished CardPicker class 110
Ana’s working on her next game 113
Build a paper prototype for a classic game 116
Up next: build a WPF version of your card picking app 118
A StackPanel is a container that stacks other controls 119
Reuse your CardPicker class in a new WPF app 120
Use a Grid and StackPanel to lay out the main window 121
Lay out your Card Picker desktop app’s window 122
Ana can use objects to solve her problem 126
You use a class to build an object 127
When you create a new object from a class, it’s called an instance of that class 128
A better solution for Ana… brought to you by objects 129
An instance uses fields to keep track of things 133
Thanks for the memory 136
What’s on your program’s mind 137
Sometimes code can be difficult to read 138
Use intuitive class and method names 140
Build a class to work with some guys 146
There’s an easier way to initialize objects with C# 148
Use the C# Interactive window to run C# code 154
4

Getting the reference

What would your apps be without data? Think about it for a minute. Without data, your programs are... well, it’s actually hard to imagine writing code without data. You need information from your users, and you use that to look up or produce new information to give back to them. In fact, almost everything you do in programming involves working with data in one way or another. In this chapter, you’ll learn the ins and outs of C#’s data types and references, see how to work with data in your program, and even learn a few more things about objects (guess what... objects are data, too!).

Owen could use our help!

Character sheets store different types of data on paper

A variable’s type determines what kind of data it can store

C# has several types for storing integers

Let’s talk about strings

A literal is a value written directly into your code

A variable is like a data-to-go cup

Other types come in different sizes, too

10 pounds of data in a 5-pound bag

Casting lets you copy values that C# can’t automatically convert to another type

C# does some conversion automatically

When you call a method, the arguments need to be compatible with the types of the parameters

Let’s help Owen experiment with ability scores

Use the C# compiler to find the problematic line of code

Use reference variables to access your objects

Multiple references and their side effects

Objects use references to talk to each other

Arrays hold multiple values

Arrays can contain reference variables

null means a reference points to nothing

Welcome to Sloppy Joe’s Budget House o’ Discount Sandwiches!
Unity isn’t *just* a powerful, cross-platform engine and editor for building 2D and 3D games and simulations. It’s also a **great way to get practice writing C# code**. In this lab, you’ll get *more practice writing C# code for a project in Unity.*

C# scripts add behavior to your GameObjects 214  
Add a C# script to your GameObject 215  
Write C# code to rotate your sphere 216  
Add a breakpoint and debug your game 218  
Use the debugger to understand `Time.deltaTime` 219  
Add a cylinder to show where the Y axis is 220  
Add fields to your class for the rotation angle and speed 221  
Use `Debug.DrawRay` to explore how 3D vectors work 222  
Run the game to see the ray in the Scene view 223  
Rotate your ball around a point in the scene 224  
Use Unity to take a closer look at rotation and vectors 225  
Get creative! 226
Keep your privates...private

Ever wished for a little more privacy?

Sometimes your objects feel the same way. Just like you don't want anybody you don't trust reading your journal or paging through your bank statements, good objects don't let other objects go poking around their fields. In this chapter, you're going to learn about the power of encapsulation, a way of programming that helps you make code that's flexible, easy to use, and difficult to misuse. You'll make your objects' data private, and add properties to protect how that data is accessed.

Let's help Owen roll for damage
Create a console app to calculate damage
Design the XAML for a WPF version of the damage calculator
The code-behind for the WPF damage calculator
Tabletop talk (or maybe...dice discussion?)
Let's try to fix that bug
Use Debug.WriteLine to print diagnostic information
It's easy to accidentally misuse your objects
Encapsulation means keeping some of the data in a class private
Use encapsulation to control access to your class's methods and fields
But is the RealName field REALLY protected?
Private fields and methods can only be accessed from instances of the same class
Why encapsulation? Think of an object as a black box...
Let's use encapsulation to improve the SwordDamage class
Encapsulation keeps your data safe
Write a console app to test the PaintballGun class
Properties make encapsulation easier
Modify your Main method to use the Bullets property
Auto-implemented properties simplify your code
Use a private setter to create a read-only property
What if we want to change the magazine size?
Use a constructor with parameters to initialize properties
Specify arguments when you use the new keyword

RealName: "Herb Jones"
Alias: "Dash Martin"
Password: "the crow flies at midnight"
Your object’s family tree

Sometimes you DO want to be just like your parents.

Ever run across a class that almost does exactly what you want your class to do? Found yourself thinking that if you could just change a few things, that class would be perfect? With inheritance, you can extend an existing class so your new class gets all of its behavior—with the flexibility to make changes to that behavior so you can tailor it however you want. Inheritance is one of the most powerful concepts and techniques in the C# language: with it you’ll avoid duplicate code, model the real world more closely, and end up with apps that are easier to maintain and less prone to bugs.

- Calculate damage for MORE weapons 274
- Use a switch statement to match several candidates 275
- One more thing…can we calculate damage for a dagger? and a mace? 277
- When your classes use inheritance, you only need to write your code once 278
- Build up your class model by starting general and getting more specific 279
- How would you design a zoo simulator? 280
- Every subclass extends its base class 285
- Use a colon to extend a base class 290
- A subclass can override methods to change or replace members it inherited 292
- Some members are only implemented in a subclass 297
- Use the debugger to understand how overriding works 298
- Build an app to explore virtual and override 300
- A subclass can hide methods in the base class 302
- Use the override and virtual keywords to inherit behavior 304
- When a base class has a constructor, your subclass needs to call it 307
- It’s time to finish the job for Owen 309
- Build a beehive management system 316
- The Queen class: how she manages the worker bees 318
- The UI: add the XAML for the main window 319
- Feedback drives your Beehive Management game 328
- Some classes should never be instantiated 332
- An abstract class is an intentionally incomplete class 334
- Like we said, some classes should never be instantiated 336
- An abstract method doesn’t have a body 337
- Abstract properties work just like abstract methods 338
C# is an object-oriented language, and since these Head First C# Unity Labs are all about getting practice writing C# code, it makes sense that these labs will focus on creating objects.

Let's build a game in Unity!

- Create a new material inside the Materials folder
- Spawn a billiard ball at a random point in the scene
- Use the debugger to understand Random.value
- Turn your GameObject into a prefab
- Create a script to control the game
- Attach the script to the Main Camera
- Press Play to run your code
- Use the Inspector to work with GameObject instances
- Use physics to keep balls from overlapping
- Get creative!
interfaces, casting, and “is”

Making classes keep their promises

Actions speak louder than words.
Sometimes you need to group your objects together based on the things they can do rather than the classes they inherit from. That’s where interfaces come in—they let you work with any class that can do the job. But with great power comes great responsibility, and any class that implements an interface must promise to fulfill all of its obligations…or the compiler will break its kneecaps, see?

The beehive is under attack!
Yes, ma’am!

The RoboBee 4000 can do a worker bee’s job without using valuable honey

The IWorker’s Job property is a hack

Use “is” to check the type of an object
Use “is” to access methods in a subclass

What if we want different animals to swim or hunt in packs?

Use interfaces to work with classes that do the same job
Safely navigate your class hierarchy with “is”

C# has another tool for safe type conversion: the “as” keyword

Use upcasting and downcasting to move up and down a class hierarchy

Upcasting turns your CoffeeMaker into an Appliance
Upcasting and downcasting work with interfaces, too

Downcasting turns your Appliance back into a CoffeeMaker

Interfaces can inherit from other interfaces
Interfaces can have static members
Default implementations give bodies to interface methods
Add a ScareAdults method with a default implementation

Data binding updates WPF controls automatically
Modify the Beehive Management System to use data binding

Polymorphism means that one object can take many different forms
Organizing your data

When it rains, it pours.

In the real world, you don’t receive your data in tiny little bits and pieces. No, your data’s going to come at you in loads, piles, and bunches. You’ll need some pretty powerful tools to organize all of it, and that’s where enums and collections come in. Enums are types that let you define valid values to categorize your data. Collections are special objects that store many values, letting you store, sort, and manage all the data that your programs need to pore through. That way, you can spend your time thinking about writing programs to work with your data, and let the collections worry about keeping track of it for you.

- Strings don’t always work for storing categories of data
- Enums let you work with a set of valid values
- Enums let you represent numbers with names
- We could use an array to create a deck of cards...
- Lists make it easy to store collections of...anything
- Lists are more flexible than arrays
- Let’s build an app to store shoes
- Generic collections can store any type
- Collection initializers are similar to object initializers
- Let’s create a List of Ducks
- Lists are easy, but SORTING can be tricky
- IComparable<Duck> helps your list sort its ducks
- Use IComparer to tell your List how to sort
- Create an instance of your comparer object
- Overriding a ToString method lets an object describe itself
- Update your foreach loops to let your Ducks and Cards write themselves to the console
- You can upcast an entire list using IEnumerable<T>
- Use a Dictionary to store keys and values
- The Dictionary functionality rundown
- Build a program that uses a dictionary
- And yet MORE collection types…
- A queue is FIFO—first in, first out
- A stack is LIFO—last in, first out
- Downloadable exercise: Two Decks
Unity Lab 4

User Interfaces

In the last Unity Lab you started to build a game, using a prefab to create GameObject instances that appear at random points in 3D space and fly in circles. This Unity Lab picks up where the last one left off, allowing you to apply what you’ve learned about interfaces in C# and more.

- Add a score that goes up when the player clicks a ball 454
- Add two different modes to your game 455
- Add game mode to your game 456
- Add a UI to your game 458
- Set up the Text that will display the score in the UI 459
- Add a button that calls a method to start the game 460
- Make the Play Again button and Score Text work 461
- Finish the code for the game 462
- Get creative! 466

This screenshot shows the game in its running mode. Balls are added and the player can click on them to score.

When the last ball is added, the game switches to its Game Over mode. The Play Again button pops up and no more balls get added.
# LINQ and lambdas

## Get control of your data

You’re ready for a whole new world of app development.

Using WinForms to build Windows Desktop apps is a great way to learn important C# concepts, but there’s so much more you can do with your programs. In this chapter, you’ll use XAML to design your Windows Store apps, you’ll learn how to build pages to fit any device, integrate your data into your pages with data binding, and use Visual Studio to cut through the mystery of XAML pages by exploring the objects created by your XAML code.

<table>
<thead>
<tr>
<th>Clause #2: Include only certain values</th>
<th>Clause #3: Order the elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jimmy's a Captain Amazing super-fan...</td>
<td>0 12 36 13 8</td>
</tr>
<tr>
<td>Use LINQ to query your collections</td>
<td>36 13 12 8 0</td>
</tr>
<tr>
<td>LINQ works with any IEnumerable&lt;T&gt;</td>
<td></td>
</tr>
<tr>
<td>LINQ's query syntax</td>
<td></td>
</tr>
<tr>
<td>LINQ works with objects</td>
<td></td>
</tr>
<tr>
<td>Use a LINQ query to finish the app for Jimmy</td>
<td>468</td>
</tr>
<tr>
<td>The var keyword lets C# figure out variable types for you</td>
<td>470</td>
</tr>
<tr>
<td>LINQ queries aren't run until you access their results</td>
<td>472</td>
</tr>
<tr>
<td>Use a group query to separate your sequence into groups</td>
<td>475</td>
</tr>
<tr>
<td>Use join queries to merge data from two sequences</td>
<td>477</td>
</tr>
<tr>
<td>Use the new keyword to create anonymous types</td>
<td>480</td>
</tr>
<tr>
<td>Add a unit test project to Jimmy’s comic collection app</td>
<td>481</td>
</tr>
<tr>
<td>Write your first unit test</td>
<td>482</td>
</tr>
<tr>
<td>Write a unit test for the GetReviews method</td>
<td>483</td>
</tr>
<tr>
<td>Write unit tests to handle edge cases and weird data</td>
<td>484</td>
</tr>
<tr>
<td>Use the <code>=&gt;</code> operator to create lambda expressions</td>
<td>485</td>
</tr>
<tr>
<td>A lambda test drive</td>
<td>486</td>
</tr>
<tr>
<td>Refactor a clown with lambdas</td>
<td>487</td>
</tr>
<tr>
<td>Use the <code>:</code> operator to make your lambdas make choices</td>
<td>488</td>
</tr>
<tr>
<td>Lambda expressions and LINQ</td>
<td>489</td>
</tr>
<tr>
<td>LINQ queries can be written as chained LINQ methods</td>
<td>490</td>
</tr>
<tr>
<td>Use the <code>=&gt;</code> operator to create switch expressions</td>
<td>491</td>
</tr>
<tr>
<td>Explore the Enumerable class</td>
<td>492</td>
</tr>
<tr>
<td>Create an enumerable sequence by hand</td>
<td>493</td>
</tr>
<tr>
<td>Use yield return to create your own sequences</td>
<td>494</td>
</tr>
<tr>
<td>Use yield return to refactor ManualSportSequence</td>
<td>495</td>
</tr>
<tr>
<td>Downloadable exercise: Go Fish!</td>
<td>496</td>
</tr>
</tbody>
</table>
Sometimes it pays to be a little persistent.

So far, all of your programs have been pretty short-lived. They fire up, run for a while, and shut down. But that’s not always enough, especially when you’re dealing with important information. You need to be able to save your work. In this chapter, we’ll look at how to write data to a file, and then how to read that information back in from a file. You’ll learn about the .NET stream classes, and also take a look at the mysteries of hexadecimal and binary.
When you set up a scene in Unity, you’re creating a virtual 3D world for the characters in your game to move around in. But in most games, things aren’t directly controlled by the player. So how do these objects find their way around a scene? In this lab, we’ll look at how C# can help.

- Create a new Unity project and start to set up the scene 578
- Set up the camera 579
- Create a GameObject for the player 580
- Introducing Unity’s navigation system 581
- Set up the NavMesh 582
- Make your player automatically navigate the play area 583
The life and death of an object
Use the GC class (with caution) to force garbage collection
Your last chance to DO something…your object’s finalizer
When EXACTLY does a finalizer run?
Finalizers can’t depend on other objects
A struct looks like an object…
Values get copied; references get assigned
Structs are value types; objects are reference types
The stack vs. the heap: more on memory
Use out parameters to make a method return more than one value
Pass by reference using the ref modifier
Use optional parameters to set default values
A null reference doesn’t refer to any object
Non-nullable reference types help you avoid NREs
The null-coalescing operator ?? helps with nulls
Nullable value types can be null…and handled safely
“Captain” Amazing…not so much
Extension methods add new behavior to EXISTING classes
Extending a fundamental type: string
Putting out fires gets old
Programmers aren’t meant to be firefighters.
You’ve worked your tail off, waded through technical manuals and a few engaging Head First books, and you’ve reached the pinnacle of your profession. But you’re still getting panicked phone calls in the middle of the night from work because your program crashes, or doesn’t behave like it’s supposed to. Nothing pulls you out of the programming groove like having to fix a strange bug…but with exception handling, you can write code to deal with problems that come up. Better yet, you can even plan for those problems, and keep things running when they happen.

Your class, now with exception handling

```
int[] anArray = {3, 4, 1, 11};
int aValue = anArray[15];
```

Your hex dumper reads a filename from the command line

When your program throws an exception, the CLR generates an Exception object

All Exception objects inherit from System.Exception

There are some files you just can’t dump

What happens when a method you want to call is risky?

Handle exceptions with try and catch

Use the debugger to follow the try/catch flow

If you have code that ALWAYS needs to run, use a finally block

Catch-all exceptions handle System.Exception

Use the right exception for the situation

Exception filters help you create precise handlers

The worst catch block EVER: catch-all plus comments

Temporary solutions are OK (temporarily)
In the last Unity Lab, you created a scene with a floor (a plane) and a player (a sphere nested under a cylinder), and you used a NavMesh, a NavMesh Agent, and raycasting to get your player to follow your mouse clicks around the scene. In this lab, you’ll add to the scene with the help of C#.

Let’s pick up where the last Unity Lab left off 652
Add a platform to your scene 653
Use bake options to make the platform walkable 654
Include the stairs and ramp in your NavMesh 655
Fix height problems in the NavMesh 657
Add a NavMesh Obstacle 658
Add a script to move the obstacle up and down 659
Get creative! 660

This NavMesh Obstacle carves a moving hole in the NavMesh that prevents the Player going up the ramp. You’ll add a script that lets the user drag it up and down to block and unblock the ramp.
appendix i: ASP.NET Core Blazor projects

Visual Studio for Mac Learner’s Guide

Why you should learn C# 664
Create your first project in Visual Studio for Mac 666
Let’s build a game! 670
Create a Blazor WebAssembly App in Visual Studio 672
Run your Blazor web app in a browser 674
Start writing code for your game 676
Finish creating your emoji list and display it in the app 680
Shuffle the animals so they’re in a random order 682
You’re running your game in the debugger 684
Add your new project to source control 688
Add C# code to handle mouse clicks 689
Add click event handlers to your buttons 690
Test your event handler 692
Use the debugger to troubleshoot the problem 693
Track down the bug that’s causing the problem 696
Add code to reset the game when the player wins 698
Add a timer to your game’s code 702
Clean up the navigation menu 704
Controls drive the mechanics of your user interfaces 706
Create a new Blazor WebAssembly App project 707
Add a page with a slider control 708
Add text input to your app 710
Add color and date pickers to your app 713
Build a Blazor version of your card picking game 714
The page is laid out with rows and columns 716
The slider uses data binding to update a variable 717
Welcome to Sloppy Joe’s Budget House o’Discount Sandwiches! 720

appendix ii: Code Kata

A learning guide for advanced and impatient readers