



Learn Swift Programming by Examples

ZHIMIN ZHAN

Learn Swift Programming by Examples

Zhimin Zhan

This book is for sale at <http://leanpub.com/learn-swift-programming-by-examples>

This version was published on 2019-09-21



Leanpub

This is a [Leanpub](#) book. Leanpub empowers authors and publishers with the Lean Publishing process. [Lean Publishing](#) is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

© 2016 Zhimin Zhan

Contents

Preface	i
What is unique about this book?	ii
Who should read this book	iii
How to read this book	iii
Send me feedback	iii
1. Introduction	1
1.1 Xcode	1
1.2 Swift Playgrounds in Xcode	2
1.3 Swift Projects in Xcode	4
1.4 Swift Tutorials	6
1.5 Rhythm for working on the exercises	6
1.6 Common Errors	7
1.7 Interactive Swift	10
1.8 Swift evolves	11
2. Printing Shapes	12
2.1 Print out Triangle	12
2.2 Print out a half diamond	15
2.3 Print out diamond shape	18
2.4 Print big diamond, name your size	20
2.5 Exercises	23
Resources	24
Online resources	24
Books	24
Software	25

Preface

On December 8, 2013, US President Barack Obama “asked every American to give it a shot to learn to code” ([watch it here](#)¹), kicking off the Hour of Code campaign for Computer Science Education Week 2013. “Learning these skills isn’t just important for your future, it’s important for our country’s future,” President Obama said.

In February 2013, Mark Zuckerberg and Bill Gates and several other big names in IT “want kids to learn code” ([video](#)²). I particularly like the quote at the beginning of the video:

“Everybody in this country should learn how to program a computer... because it teaches you how to think.” - Steve Jobs

You don’t have to be an American to get the message: coding (aka. programming) is an important skill for this information age. Besides the importance of programming, the other message those VIPs also tried to convey is that “you can do it”.

Learning programming is a way to master communication with computers, by giving them instructions to perform tasks for you. A programming language is a language that is used to write instructions for computers to understand and execute. There are several popular programming languages such as Java, C#, Ruby, and PHP. For beginners, don’t fixate on one. Computers internally work the same way, mastering thinking in programming is more important than an individual language. In my opinion, different programming languages are like dialects. I learned and taught myself over a dozen of programming languages. Once you have mastered one, it is easy to learn another.

In this book, I will use Swift, a new programming language from Mac OS X and iOS. Swift was first unveiled at the Apple Worldwide Developers Conference (WWDC) 2014, Apple describes Swift as “a successor to both the C and Objective-C languages”³ on its website. Before Swift, Objective-C was the only programming language you could use to build Mac and iOS apps. [Redmonk programming language rankings for 2015](#)⁴ shows a huge growth

¹<https://www.adafruit.com/blog/2013/12/09/president-obama-calls-on-every-american-to-learn-code/>

²<http://www.psfk.com/2013/02/mark-zuckerberg-bill-gates-coding-school.html>

³<https://developer.apple.com/swift>

⁴<http://redmonk.com/sograpy/2015/01/14/language-rankings-1-15/>

in popularity for Apple’s new Swift development language. According to the results, Swift went from the 68th most popular language last quarter (when it launched) to the 22nd, a jump of 46 spots which is “unprecedented in the history of these rankings.”

In June 8, 2015 at WWDC 2015, Apple announced that the company is [open-sourcing the Swift 2.0](#)⁵. “We think Swift is the next big programming language, the one that we’ll all be doing application and system programming on for 20 years to come,” Craig Federighi, Apple’s senior vice president of software engineering, said. “We think Swift should be everywhere and used by everyone.”

What do these mean? Swift will be the main programming (replacing Objective-C) for coding applications for Apple’s iOS, Mac OS and tvOS platforms, and developers like it. A good news for beginners, comparing to Objective-C, Swift is a lot easier to learn. So learn and master Swift, as President Obama said in the video, “Don’t just download the latest app, help design it; Don’t just play on your phone, program it”.

What is unique about this book?

A typical how-to-program book will go through the programming concepts, syntax and followed by demonstrations with simple examples. I have read dozens of them (for different programming languages or tools) before and have taught this way at universities. It was not an effective approach. It is more like a teacher dumping knowledge upon students. But I did not know a better way, until I discovered [The Michel Thomas Method](#)⁶.

The Michel Thomas Method was developed by Michel Thomas for teaching foreign languages. Thomas claimed that his students could “achieve in three days what is not achieved in two to three years at any college”. My understanding of this method is that the teacher starts with a simple conversation scenario, then gradually expands the scenario with a few new words each time. That way, students are familiar with the conversation topic and the majority of words or sentences, while learning some new, in real interesting conversations.

I believe this teaching method can be applied to programming. Not only a programming language may also be considered as ‘a language’, but also very practical. The ‘conversation’ in speaking languages are exercises in programming. People learn better when they get satisfaction or feedbacks and see their programs works..

As I said before, thinking in programming is much more important than being familiar with a programming language. There is no better way than writing real programs for real exercises.

⁵<https://developer.apple.com/swift/blog/?id=29>

⁶<http://www.michelthomas.com/>

In this book, I have chosen the exercises that are very simple to understand, besides teaching values, they are useful and fun to do.

There are also some programming quiz books. I often find some of those exercises are long and hard to understand. Quite commonly, the authors seem to be fond of showing off their programming skills or smart solutions. It won't be the case in this book. This book is a guide to programming and its purpose is to teach. After you finish all the exercises, you will be able to write working (might not be perfect) programs, and with confidence to continue to learn and grow.

Who should read this book

Every one who wants to write apps (and games) for Mac OS X, iOS, watchOS and tvOS. In particular, I would strongly encourage young people to give it a go.

How to read this book

It is highly recommended to read this book from page to page. The exercises are organized into chapters, exercises within each chapter generally follows an 'easy-to-hard' pattern.

The solutions for all exercises are listed in Appendix 2, and also can be downloaded on [the book website](#)⁷, for access code see the [Resources](#) section of this book.

Send me feedback

We'd appreciate your comments, suggestions, reports on errors in the book and code. You may submit your feedback on the book site.

Zhimin Zhan

Brisbane, Australia

⁷<http://zhimin.com/books/learn-swift-programming-by-examples>

1. Introduction

I still remember my first programming lesson. The first sentence the coach said was “computers are not mysterious”. Nobody uses the term ‘mysterious’ to describe computers nowadays. It was the case in 1980’s, computers were rare back then.

We are in the “Information Age” now, where computers are a large part of our lives. It seems to me that programming remains mysterious and difficult to the majority of people despite the fact that they spend most of their work hours in front of computers.

Once you have mastered programming, there are many things you can do, such as:

- Rename hundreds of files with a script instead of doing it one by one
- Generate a nice Excel report from raw data instead of typing it in
- Write a document once and use scripts to generate several different formats: HTML (online), PDF, ePub and Kindle
- Turn on or off certain electronic devices when a certain condition is met
- Write a cool iOS App or game

The bottom line is that when you know how software works you will definitely use computers better.

Before we start, just like my coach, I am telling you that “programming is not mysterious”, and you can master it. I believe this book can guide you to the wonderful programming world.

Like many skills, you cannot master programming by reading the book only, you need to **do** it. Let’s begin.

1.1 Xcode

Coding Swift requires Xcode 6 or later installed on a Mac computer. Xcode is the integrated development environment (IDE) for developing software for OS X and iOS. [Xcode](https://developer.apple.com/xcode/downloads/)¹ (the latest version to date is 8.0) is available free of charge on the Mac App Store.

¹<https://developer.apple.com/xcode/downloads/>



Dedicated folder for coding exercises

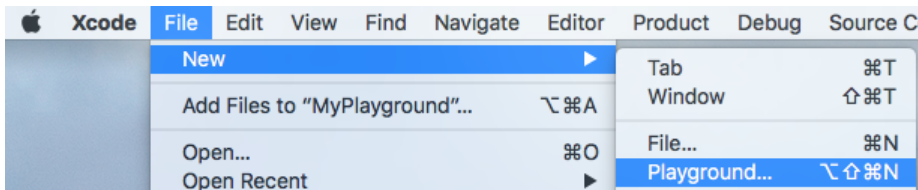
I suggest creating a dedicated folder to put all your code (for the exercises in this book) in, for example, `/Users/YOURUSERNAME/swiftcode`.

1.2 Swift Playgrounds in Xcode

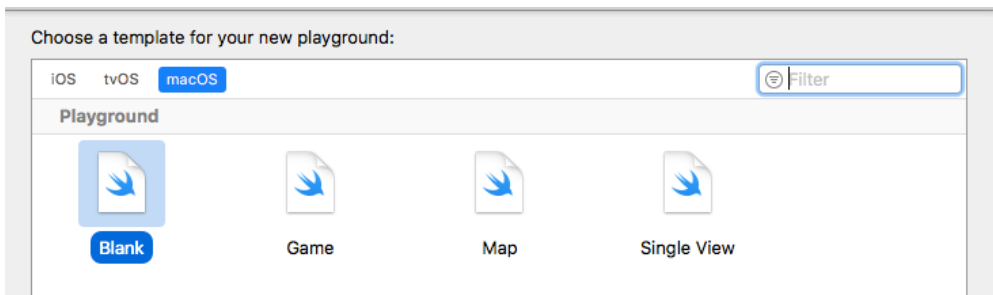
Swift Playgrounds, as its name suggests, is an interactive playground for trying out Swift. In playground, Xcode evaluates code as you write it.

Start a new Playground

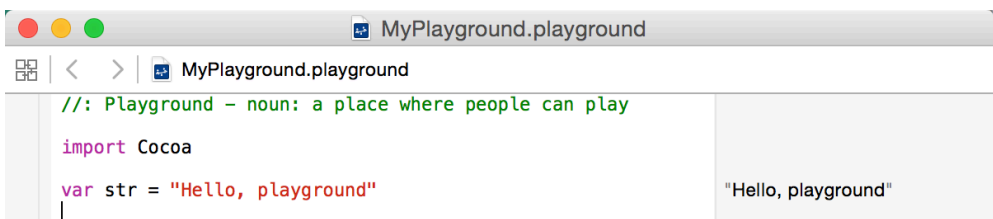
1. In Xcode, select menu “File” → “New” → “Playground...”



2. Select target platform and template



3. Choose the location to save the playground
4. Playground is shown



The code in on the left and evaluated output on the right (in a slight darker background).

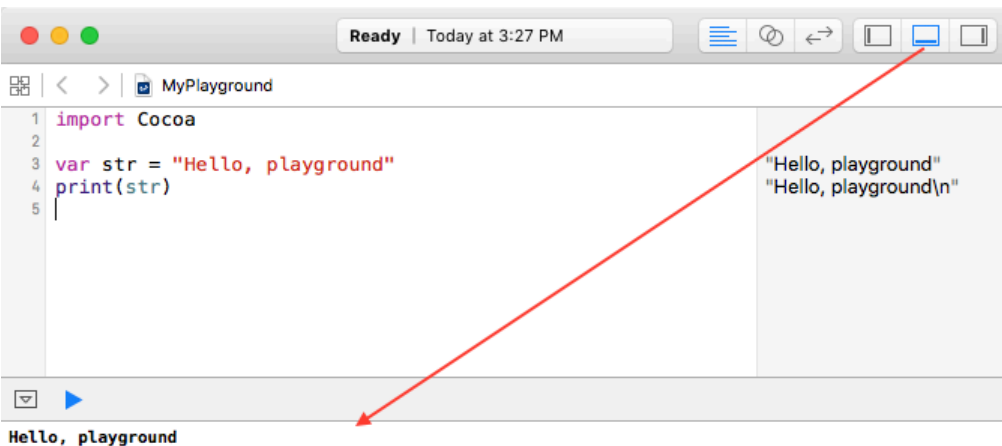
You are ready to go, type in some Swift code. Xcode supports IntelliSense, i.e. providing code competition and code suggestions.

```
5 var str = "Hello, playground"
6 print(items: Any...)

Void print(items: Any...)
Void print(items: Any..., separator: String, terminator: String)
Void print(items: Any..., separator: String, terminator: String, toStream: &Target)
Void print(items: Any..., toStream: &Target)
```

Show output console

In Xcode Playground, the output of each statement will be shown on the right pane (gray one). To display the whole program's output, which is always a good idea, show the Debug area as below.

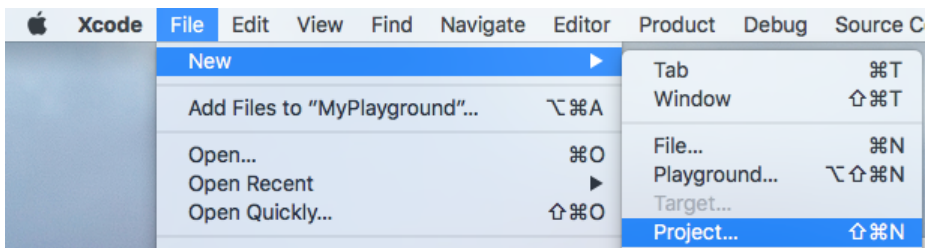


1.3 Swift Projects in Xcode

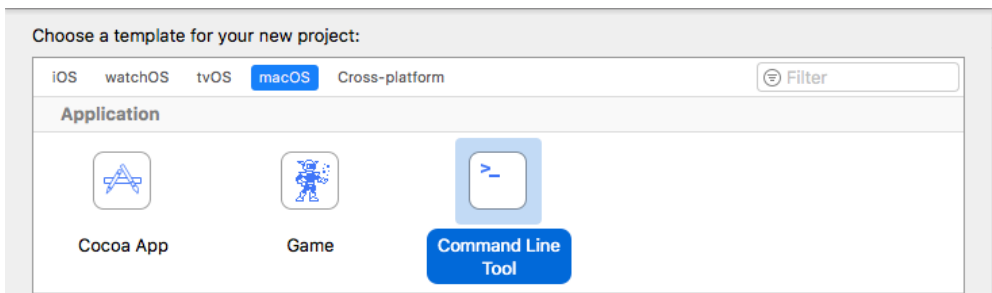
More commonly, we organize code and related files in a folder structure, known as “Project”. When we a Xcode project file (with extension `.xcodeproj`) in Xcode, all the files in the project are included.

Create a new project

1. In Xcode, select menu “File” → “New” → “Project...”



2. Select a project template.



There are quite a number of combinations.

- **macOS - Command Line Tool**

The application for running from command line. This will be main project type we will be using up to Chapter 10.

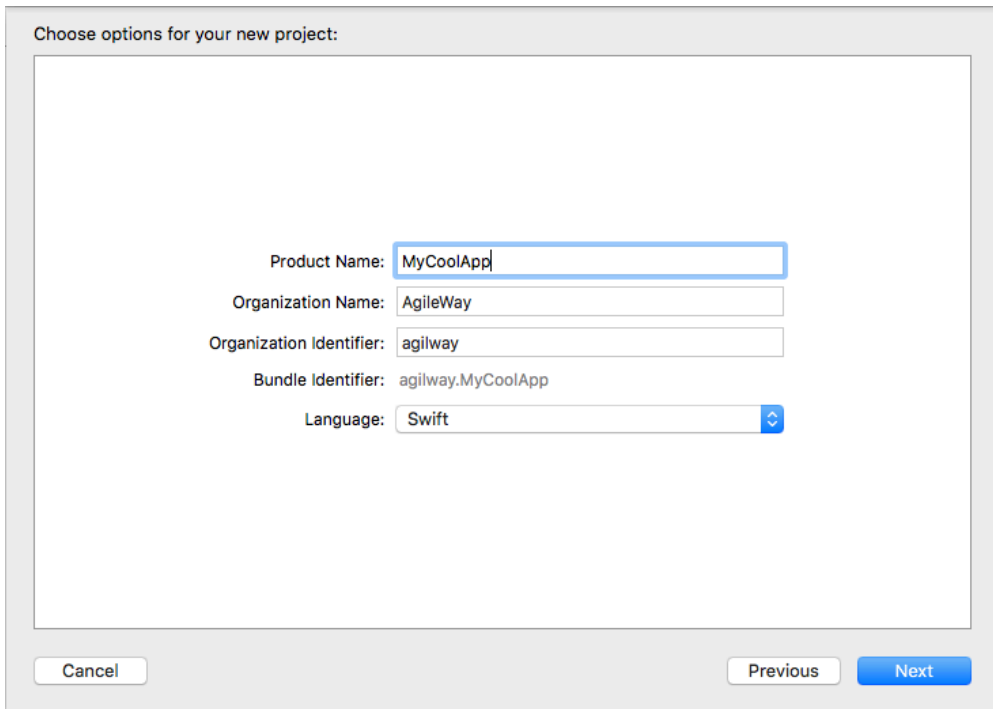
- **macOS - Cocoa App**

GUI application on Mac computers.

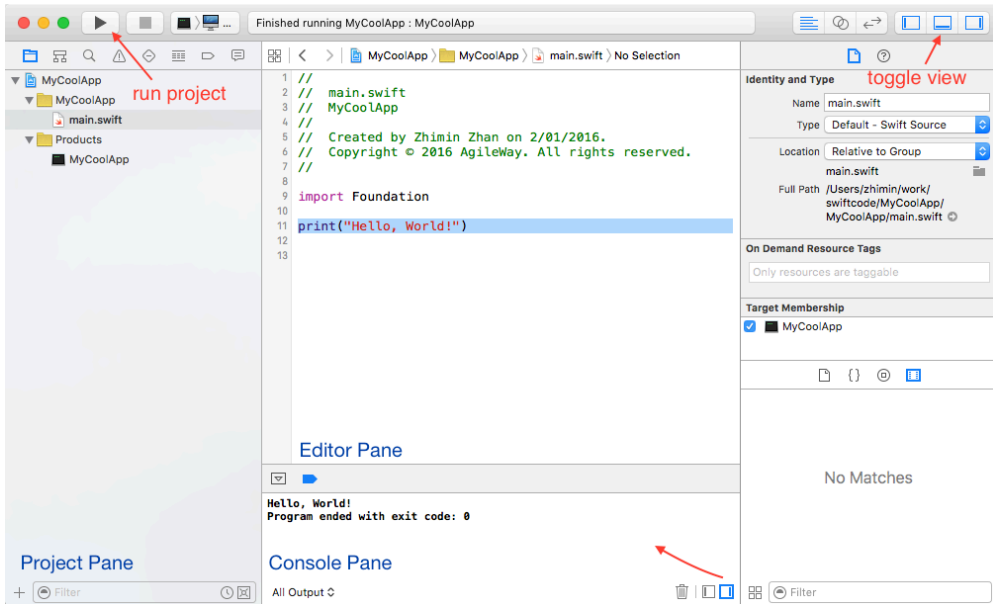
- **iOS**

Apps running on iPhone and iPad.

3. Enter project name and select destination folder



4. Project created



1.4 Swift Tutorials

While I believe you can learn basic Swift programming with this book, there are online tutorials that you may use as supplement. For example, read them on your iPad while waiting at bus stops. Here are two good (and free) ones.



Why bother this book if I can get 'The Swift Programming Language' book free?

Online tutorials teach you the basic Swift syntax and some programming concepts. While they are important, this knowledge is only be useful if can be put into practice. For example, to be a good chess player, knowing chess rules is not enough.

Programming, in my opinion, is a problem solving skill to solve problems in a computer friendly way. This knowledge can only gained by actual coding, which is what this book for. Online tutorials, especially video tutorials, put learners in a passive mode. You need a book such as this one to turn passive knowledge to your own.

1.5 Rhythm for working on the exercises

In the following chapters, each consists of around 5 guided exercises, arranged in order of difficulty. Each guided exercise has five sections:

- **The problem to solve**

It usually comes with sample output for easier understanding. Make sure you understand it well.

- **Purpose**

What you can learn from this exercise.

- **Analyze**

Analyze a problem like a programmer. This is an important step. Quite often we know how to do it but cannot describe it well. Take number sorting as an example; you can sort 5 numbers instantly in head. But how about 100 numbers? This requires you to translate your understanding of sorting step by step into procedures that a computer can execute.

Now **write the code for the exercise**. No one can learn programming by reading, you have to actually do it. You may refer to the hints section to help you get through.

- **Hints**

Techniques and short executable code that may help you to solve the problem when you get stuck.

- **Solution(s)**

Solutions (can be found at Appendix II) to the most of exercises are between 20 to 50 lines of code. The runnable solution scripts can be downloaded at the book site.

If you are struggling to solve an exercise, feel free to check out our solutions (at Appendix II). The exercises are selected to introduce new programming skills/practices as well as previous knowledge. Don't worry if you cannot get it right the first time, you will have chances to apply in later exercises. As long as you are trying, you are learning.

- **Review**

Think about what you have learned, look at the code you just wrote, try apply new-learned skills to solve similar problems.

1.6 Common Errors

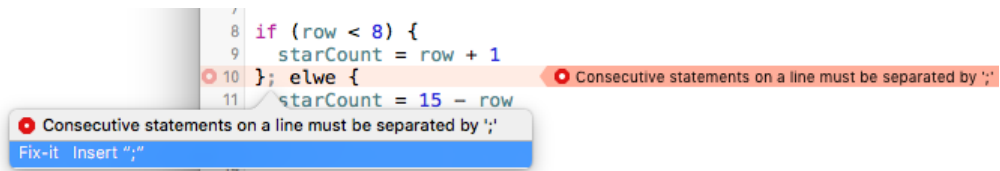
Programmers (new or experienced) encounter code errors every day. I don't expect you to get the exercises right at the first go. We learn from mistakes.

Typos

It is normal that we make typing mistakes when writing code. Xcode checks the syntax of code before running it. If there are syntax errors in code, the error messages are usually quite helpful for identifying the error. For example, in the code below, line 10 has a typo `elwe` instead of `else`.

```
8 | if (row < 8) {
9 |     starCount = row + 1
10 | } elwe {
11 |     starCount = 15 - row
12 | }
```

Xcode highlights compile time errors.



```
8 if (row < 8) {
9   starCount = row + 1
10 }; else {
11   starCount = 15 - row
```

Consecutive statements on a line must be separated by ';'

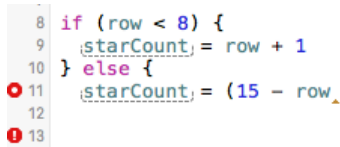
Consecutive statements on a line must be separated by ';'

Fix-it Insert ";"

The error message means `else` is undefined (don't worry if it is not meaningful to you yet, you will understand soon). It helps identify where the error is.

No matching parenthesis or brackets.

Just like Math, if there is a left bracket (“ in code, there shall be a matching right bracket “)”. There are also matching keywords for certain code structures, such as `if { ... }`. For example, there are two errors in the code below.



```
8 if (row < 8) {
9   starCount = row + 1
10 } else {
11   starCount = (15 - row)
12
13
```

1. at line 11: missing ')', shall be `(15 - row)`.
2. at line 13: missing `}` for `if` at line 8.

Code logic errors

The above two kinds of errors (called syntax errors) are relatively easy to spot, as highlighted in Xcode. The difficult errors for programmers are code logic errors. That is, the code can run without syntax errors, but does not behave as it is supposed to. The ability to debug code errors (find and fix them) separates good programmers from the average. With practice, you will get better on this.

For beginners, I have some practical tips.

1. One step at a time

Write one line of code, run the code immediately. This may sound uninteresting, but in practice, many find it is the most useful tip to learn programming. If newly added or changed code fragment caused the error, a click of 'Undo' button (in your editor) will get your code back to previous working state.

2. Work out on paper if necessary

Our brains work differently from computers. We can tell “2, 3, 4, and 6” are actors of “12” (excluding 1 and self) instantly, but we don’t know the “11 and 11443” are factors of “125873”. For computers, there are no differences on calculating the factors for any numbers, because computers follow an algorithm and carry out calculations step by step.

The dilemma is, for many problems, we do know the algorithm, but fuzzy on the details, i.e, the exact steps. When your program is not going anywhere for hours, move away from the computer, get a pencil and start to work out with simple examples. Once you are more clear, turn your understandings into code and try it on the computer.

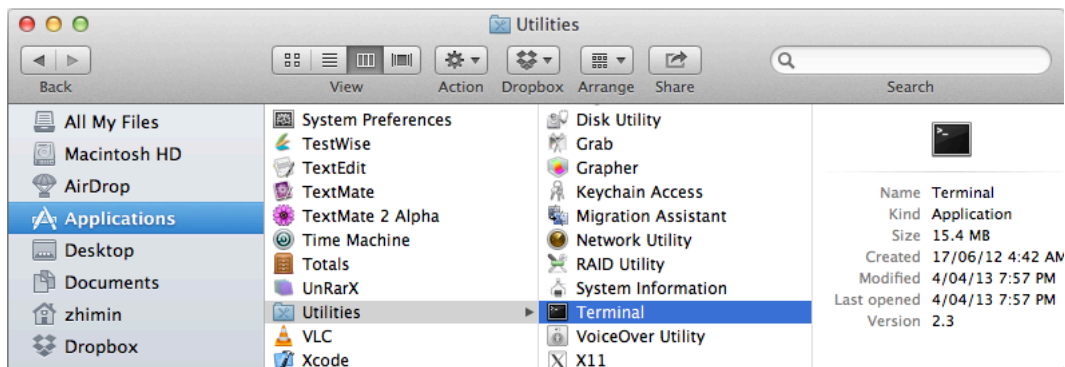
3. If feeling confused, restart

If you are stuck with existing code, chances are the complexity of the code is beyond your control. Try guessing around to get computers to work as instructed (by your code) is highly unlikely. In this case, it is better to restart from scratch. For most of the exercises in this book, solutions are less than 30 lines of code.

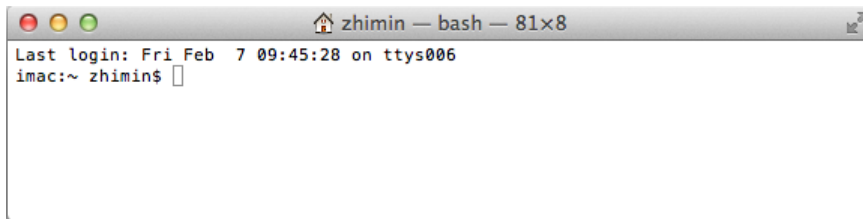
Open terminal

The best way to interact with your programs is from the command line, you might have seen these scenes in some Hollywood movies: hackers typed some commands in some text windows, and something (usually big) happened. That windows that accept user text commands are called terminals.

The application to access the command line in Mac OS X is called ‘Terminal’. Open in Finder: ‘Applications’ → ‘Utilities’ → ‘Terminal’.



It looks like this:

A terminal window titled 'zhimin — bash — 81x8' with a home icon and window control buttons. The terminal shows the output of the 'swift --version' command: 'Last login: Fri Feb 7 09:45:28 on ttys006' and 'imac:~ zhimin\$'.

```
zhimin — bash — 81x8
Last login: Fri Feb 7 09:45:28 on ttys006
imac:~ zhimin$
```

Type `swift --version`, then press Enter key

```
Apple Swift version 5.1 (swiftlang-1100.0.270.13 clang-1100.0.33.7)
Target: x86_64-apple-darwin18.7.0
```

The Swift version might be different on your machine, this won't matter.

```
$ cd swiftcode
$ swift hello_world.swift
```

(*cd means 'change directory'; swift filename means running this swift file.*)

You will see the output:

```
Hello World!
```

1.7 Interactive Swift

Interactive Swift is a tool that allows the execution of Swift code with immediate response, which can be very helpful for learning Swift syntaxes and debugging simple code errors. Interactive Swift is launched from a command line, run `swift` from a command line window then try Swift code there.

```
$ swift
Welcome to Apple Swift version 5.1 (swiftlang-1100.0.270.13 clang-1100.0.33.7).
 1> print("Hello " + "Swift")
Hello Swift
 2>
```

To exit the application, type `:exit`, `:quit`, or `Ctrl+D`.

In Appendix 1 ('Swift in Nutshell') I summarized the core Swift syntax and usage in examples which you can conveniently run in Swift Console.

1.8 Swift evolves

Swift is a new language, its features and syntax are still evolving, probably too rapidly in my opinion. Thankfully, Xcode provides good hints for upgrading your old version of Swift code.

- 5.1, 2019-09-10 (current stable release)
- 5.0, 2019-03-25
- 4.2, 2018-09-17
- 4.0, 2017-09-19
- 3.0, 2016-09-13
- 2.0, 2015-06-08
- 1.2, 2015-04-18
- 1.0, 2014-09-09

For example, the common `print` method, which prints out text to console, has changed three times from 1.0 to 2.0.

```
print("Hello") // Swift 1
print("Hello", appendNewline: false) // Swift 1.2
print("Hello", terminator: "") // Swift 2
```

2. Printing Shapes

Printing out asterisk characters (*) in shapes is often used as beginner's programming exercises, as they are simple, easy to understand and visually interesting.

2.1 Print out Triangle

Write a program to print out asterisks like the triangle shape below:

```
*
**
***
****
*****
*****
*****
*****
*****
*****
*****
```

Purpose

- Develop ability to analyze patterns
- Variables
- Use of looping

Analyze

Row	The number of stars
1	1
2	2
3	3
...	...
n	n



Code in Xcode Playground

The first 3 exercises are simple output text pattern ones. I recommend you coding them in Xcode Playground, simple and getting quick feedback.

Hints

Print out text.

```
print("*")
print("**") // will be in a separate line
```

Generate multiple occurrences of the same character.

```
// get multiple '$'s
String(repeating: "$", count: 3) // => "$$$"
```

Using a variable to store an integer.



Variables

You can think of a variable is a 'labeled box' in computers to store data, its data can be changed. The [syntax convention](https://github.com/raywenderlich/swift-style-guide)¹ for Swift variables is [lower Camel Case](http://c2.com/cgi/wiki?LowerCamelCase)², for example, myBirthDate.

¹<https://github.com/raywenderlich/swift-style-guide>

²<http://c2.com/cgi/wiki?LowerCamelCase>

```
var starCount = 2
print(String(repeating: "*", count: starCount)) // => '**'

starCount = starCount + 1 // now starCount => 3
print(String(repeating: "*", count: starCount)) // => '***'
```

Print out text multiple times in a loop (fixed number of times).

```
for index in 1...3 {
    print(index)
}
```

Output:

```
1
2
3
```

The { ... } mark the beginning and end of loop respectively. The variable `index` is the looping index, often used in the code within the loop. `print(index)` is the code fragment that executes for specified 3 times.



Working out the solution on your computer

Make sure you understand the *Analyse* and *Hints* parts before you start.

2.2 Print out a half diamond

Write a program that prints out half of the diamond shape using asterisks.

```
*
**
***
****
*****
*****
*****
*****
*****
*****
*****
****
***
**
*
```

Purpose

- Decrement count in loops

Analyze

The key to this problem is to determine the number of stars for the corresponding rows.

```
row 1 to 8: the same as row number
row 9 to 16: 16 - row
```

Hints

Control flows using if ... else

Code, in its simplest form, is executed from top to bottom. But if there are `if` conditions and loops (and later methods and classes), it will change the flow of execution. The conditional expressions (if-then-else statements) run different code statements depending on a boolean condition (true or false).

```
var score = 75
if score < 60 {
    print("Failed!")
} else {
    print("Pass!")
}
```

Output:

Pass!

If you change the `var score = 59` and run again, you will get `Failed!`.

Boolean condition

The statement `score < 60` after `if` is called a boolean condition. Its value can only be either `true` or `false` (which are called boolean values).

Common comparison operators in Swift

<code>==</code>	equal to
<code>!=</code>	not equal to
<code><</code>	less than
<code><=</code>	less than or equal to
<code>></code>	greater than
<code>>=</code>	greater than or equal to

Examples:

```
2 > 1 // => true
2 == 1 // => false (equal to)
2 != 1 // => true (not equal to)
2 <= 2 // => true
```



Equal sign = and Double equal sign ==

The equal sign (=) is the “assignment operator”, it assigns a value to a variable on the left.

```
var a = 1 + 2 // assign 3 to a
```

Please note the “assignment operator” is different from the “equality symbol” in Math. For example, the statement below increases the value of `a` by 1 (*assign a new value to a*) in programming code. The same equation in Math is invalid.

```
a = a + 1 // increment a by 1
```

The double equal signs (==) is a comparison operator, it compares two values for equality (returns `true` if `a` is equal to `b`, `false` otherwise).

```
if a == b {  
  print("Found a match!")  
}
```

Incorrect use of `=` for `==` is [one of the most common mistakes in programming](http://www.cprogramming.com/tutorial/common.html)³.

³<http://www.cprogramming.com/tutorial/common.html>

2.3 Print out diamond shape

Print 7 rows of ' * ' in a diamond shape as below:

```
 *
***
*****
*****
*****
***
 *
```

Purpose

- Analyse more complex patterns
- Math operators

Analyze

Below are formulas to calculate the number of star; where `rowNumber` represents the row number and `totalRows` represents the total number of rows,

1. The number of stars for the rows before the middle one is $(\text{rowNumber} - 1) * 2 + 1$.
2. the number of stars for the rows after the middle one is $(\text{totalRows} - \text{rowNumber}) * 2 + 1$

Think about the spaces in front of each row, except for the 4th row (the longest middle one).

Hints

Write down the number of spaces and stars for each row.


```
row 1: print 3 spaces + 1 star
row 2: print 2 spaces + 3 stars
row 3: print 1 space + 5 stars
row 4: print 0 space + 7 stars
row 5: print 1 space + 5 stars
row 6: print 2 spaces + 3 stars
row 7: print 3 spaces + 1 star
```

Math operators: multiply and divide

The Math multiply and divide operator are * and / respectively.

```
8 / 2 // => 4
9 / 2 // => 4, ignore the remainder
(1 + 2) * 3 + 3 / 2 // => 10
```



If you have difficulty, do it step by step. You may try to print out the top triangle first.

2.4 Print big diamond, name your size

Ask the user for the size of diamond (based on the total number of rows) and then print out a diamond shape using asterisks '*'.

```
Enter the maximum number of rows (odd number): 9
```

```
  *
 ***
*****
*****
*****
*****
  ***
   *
```

Purpose

- Read user's input into a variable
- Convert string to integer
- Use variable control loop times
- Swift optionals



Code in Xcode Project

From this exercise onwards, please code them in a Xcode project. The reason: this exercise requires user's input and this does not work in Playground.

Moreover, using Xcode project is the standard way to developer Swift App. The project type is Mac OS X : Command Line Tool For instructions of creating a Xcode project, please refer to Chapter 1.

For exercises before Chapter 11, select project template: OS X → Application → Command Line Tool.

Analyze

The size of the diamond is not fixed, it depends on the number the user entered. The number the program asks the user to enter is the total number of rows, which can be stored in an variable.

If you divide the diamond into two parts (top and bottom), work out the number of rows for each part.

Hints

Read user's input

We can use `readLine()` function to read user's input from a command line application.

```
let userInput = readLine()
```

String and Integer

The `String` and `Integer` are two most common data types.

```
var a = "12"  
var b = "3"  
a + b      // => "123"
```

```
var c = 12  
var d = 3  
c + d      // => 15
```

Math operations, such as Add, between `String` and `Integer` will get compile error.

```
var b = "3"  
var c = 12  
b + c  // Binary operator '+' cannot be applied to 'String' and 'Int'
```

Convert a number string to integer

```
var a = "12"  
Int(a) // => to integer 12
```

```
**
```

Swift Optionals

Optionals in Swift is quite difficult concept for me, as it is not in other languages I have used. The purpose of Optionals is to avoid assign `nil` (means no value) to a variable. It is better explained with example.

```
var str1:String
str1 = nil // error: Nil cannot be assigned to type "string"
```

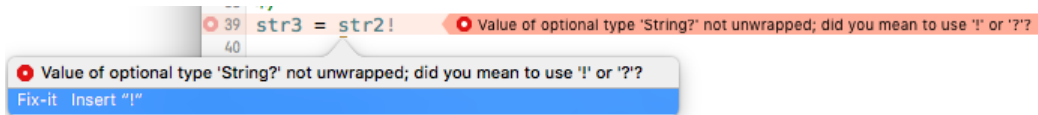
Assign `nil` to an Optional is OK, with an extra `?`.

```
var str2:String?
str2 = nil // OK
```

However, you can force assigning an optional variable to non-optional one by using `!`.

```
var str3:String
str3 = str2 // error: Value of optional type "String?" not unwrapped
str3 = str2! // OK
```

You might still find it confusing. Good news is that Xcode will give you hints and even offer 'Fix it'.



You must know it first, then instruct computers

Trying to enter a big number for the last program, say 99. It will print a big diamond, Wow!

This an important aspect of programming. Once you figure out the pattern and logic of a problem and translate it into computer understandable language (program), it can solve the similar problems at any scale. For example, the effort taken for computers to calculate 2×2 is not much different from 12343×35345 . In other words, we (human) must understand **how** to solve the problem first. Programming translates the **how** into instructions that computers can follow.

Resources

Solutions to exercises

<http://zhimin.com/books/learn-swift-programming-by-examples>⁴

Username: agileway

Password: CURRENCYWISE16

Log in with the above, or scan QR Code to access directly.



Online resources

- [The Swift Programming Language Guide and Reference](#)⁵
The official Swift language guide.
- [Raywenderlich Tutorials](#)⁶
Good quality tutorials on Swift and App development.

Books

- [The Swift Programming Language](#)⁷
The official Swift ebook from Apple.
- [Learn Ruby Programming by Examples](#)⁸ by Zhimin Zhan and Courtney Zhan
Learn Ruby programming to empower you to write scripts and cool web applications (Ruby on Rails). Master Ruby quickly by leveraging this book.
- [Cocoa Programming for OS X](#)⁹ by Aaron Hillegass, Adam Preble and Nate Chandler, Big Nerd Ranch
A good book on Cocoa programming.

⁴<http://zhimin.com/books/learn-swift-programming-by-examples>

⁵https://developer.apple.com/library/ios/documentation/Swift/Conceptual/Swift_Programming_Language/

⁶<http://www.raywenderlich.com/tutorials>

⁷<https://itunes.apple.com/au/book/swift-programming-language/id881256329?mt=11>

⁸<https://leanpub.com/learn-ruby-programming-by-examples-en>

⁹<https://www.bignerdranch.com/we-write/cocoa-programming/>

Software

- **CocoaPods**¹⁰

CocoaPods is a long-standing dependency manager for Cocoa.

- **Carthage**¹¹

Alternative to CocoaPods, a simpler dependency manager.

¹⁰<https://cocoapods.org/>

¹¹<https://github.com/Carthage/Carthage>