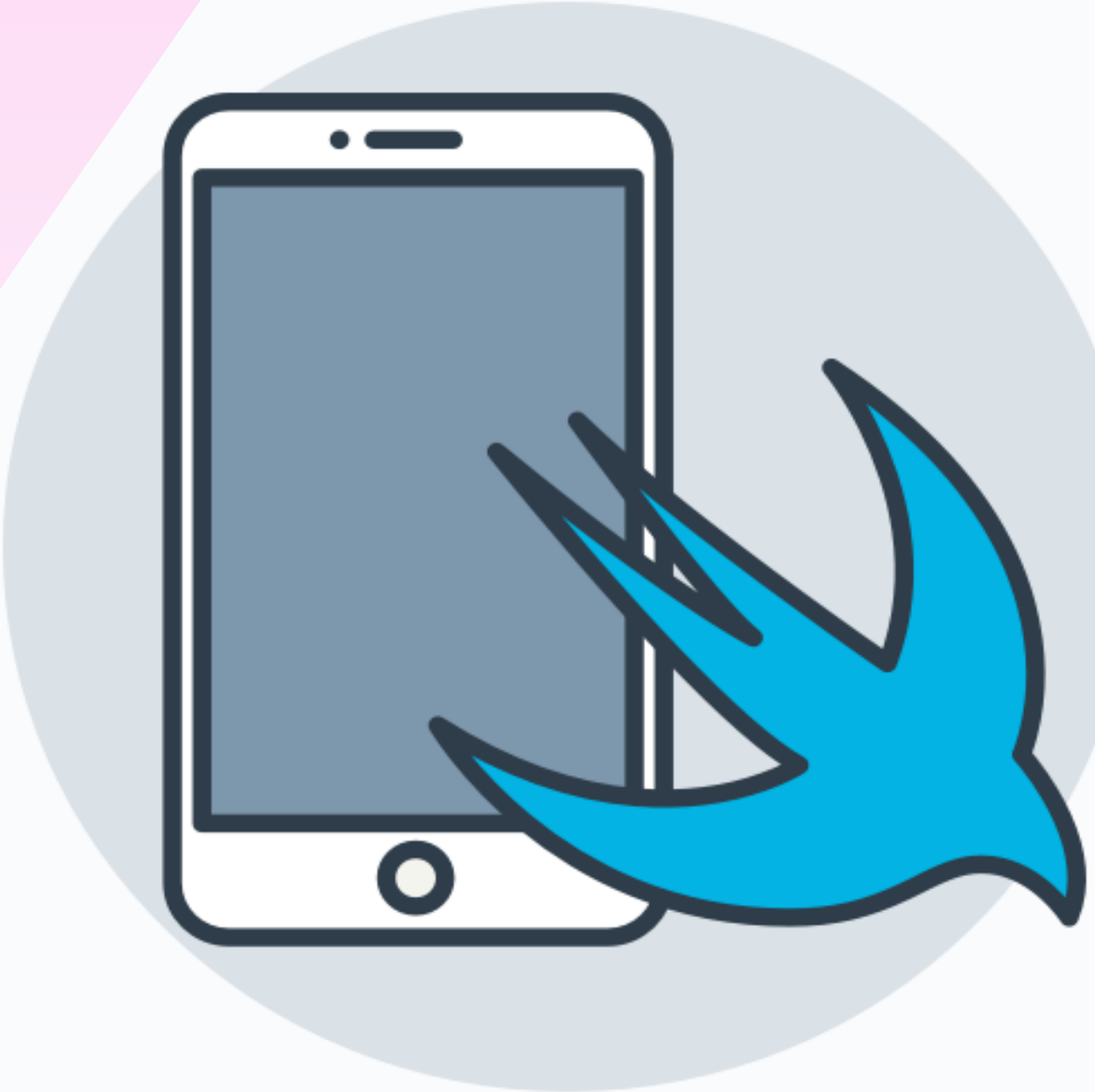
GAME 3004

Week 1

Mark Meritt
mark.meritt@georgebrown.ca



Course Objective

By the end of GAME 3004, I should be able to build a 2D iOS game using the required game framework and deploy it to the App Store

Course Outline

Week 1 Swift intro

Week 2 Swift advanced topics

Week 3 Spritekit intro

Week 4 Spritekit scenes

Week 5 Spritekit rendering

Week 6 Physics & collision

Week 7 Scene scrolling

Intermission Week - Break

Week 9 Actions & animations

Week 10 Particle effects

Week 11 Points & sound/transitions

Week 13 Final Project milestones

Week 15 Publish to app store

Week 15 Demo final project

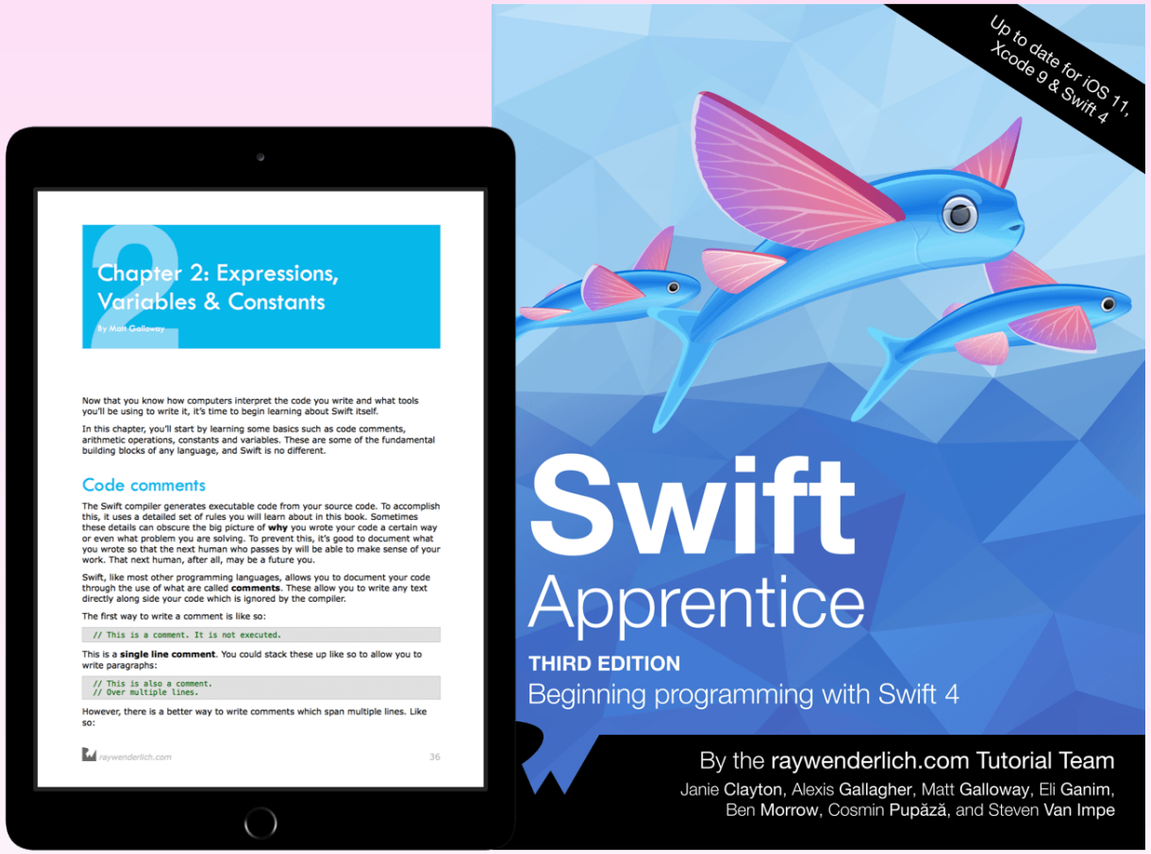
Evaluation System

Assignment 1	Swift game	Week 3	12.5%
Assignment 2	Spritekit game	Week 6	12.5%
Quizzes	In-lab tests	Weeks 4, 7, 10	15%
Participation	In-lab exercises	On going	10%

Final Project	**iOS game**		**50%**
Milestone 1	Gdd	Week 6	10%
Milestone 2	Ui prototype	Week 8	10%
Milestone 3	Game implement	Week 12	15%
Milestone 4	Published app	Week 15	10%
Milestone 5	Demo game	Week 15	5%

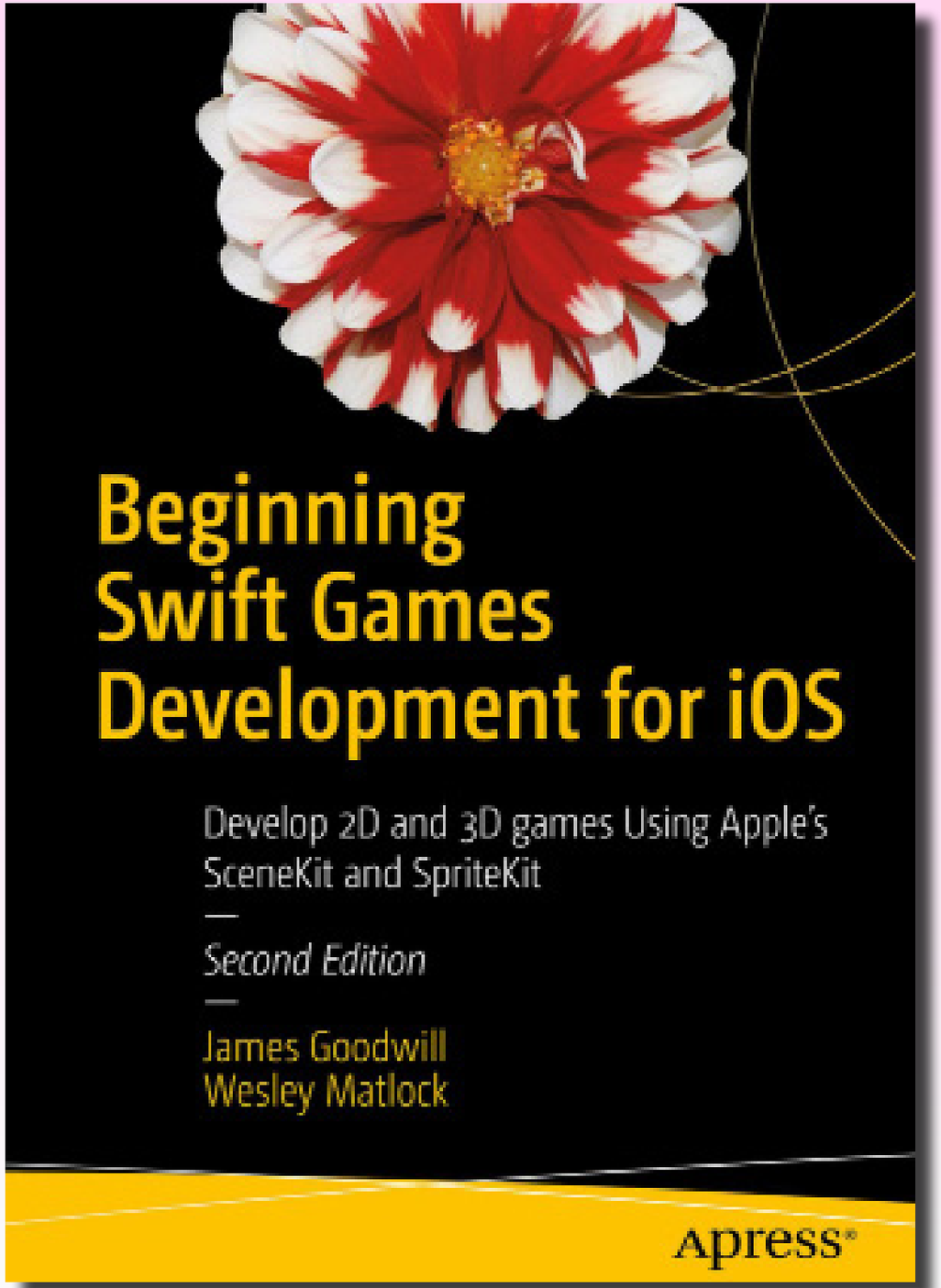
Swift Textbook

Swift Apprentice 3rd Edition
Raywenderlich Team
(ch 1-17)



Spritekit Textbook

Beginning Swift Games Development for iOS
(ch 1-8)



Lesson 1

Expectation

Prior understanding of Object Oriented Programming

Outcome

Understanding the basic principles of Swift

Key Concepts

Intro to Swift
Playgrounds
Variables & Constants
Type Inferencing
Strings
Control Flow
Functions
Tuples
Collections

What does Swift have to offer?

Swift is fast, safe, current, and attractive

Swift is a successor to the C and Objective-C languages

You do not need to understand Objective-C to understand Swift

Swift is its own language and coexists with Objective-C and interprets Objective-C code

Swift

Fairly new language

- Released in summer 2014

Beautiful syntax

- Modern and approachable

Interoperability

- Can plug directly into your existing projects and run side-by-side with your Objective-C code

Swift

Strong typing

- Compiler will catch more bugs at compile time

Smart type inference

- Automatically detect the types of your variables and constants based upon their initial value assigned

Automatic memory management

- “Memory management just works in Swift.”

Playgrounds

Playgrounds

1	Source Editor	Write code here
2	Activity Viewer	Status of the playground
3	Results Sidebar	Results of your code
4	Execution Control	Set automatic running

Variables & Constants

Variables values **can** vary (mutable)

Keyword var is used to create a **variable**

Constant values **cannot** be changed (immutable)

Keyword let is used to create a **constant**

Value Structure

var myValue : **Int** = 5

1. keyword can be **var** or **let**
2. **name** of value
3. **type** of value declared
4. value **assigned** and **initialized**

Variables & Constants

var myVariable: **Int** = 42

myVariable = 50 ✅

let myConstant: **Int** = 14

myConstant = 15 ❌ Error

Type Inferencing

When declaring a variable or constant you **do not** have to specify the type.
Swift knows what you are trying to make before you tell it.

```
let typeInferredInt = 42
```

Swift knows this is an Int, without you writing it. **Optionally**, you can tell Swift what you are creating as well.

```
let typeInferredInt : Int = 42
```

Strings

Strings store multiple characters values together

The **values** within the quotes are called **string literal**

var label: **String** = **“Hello there“**

String Concatenation

String concatenation is the **manipulation** of strings.

You can **concatenate** strings using '+' **addition operator** or '+=' **addition-equals operator.**

var name = "Mark"
var output = "Hello there my name is " + name

String Interpolation

String interpolation is a special Swift syntax that lets you build a string in a way that's easy to read.

Use **\()** to replace certain parts of the string with other values.

var name = "**Mark**"
var output = "**Hello there my name is \(name)**"

Control Flow

The flow of a program is controlled by various methods

In Swift we can use **if statements**, **switch statements**, **for-in**, **while and repeat-while** loops for control flow

Use loops to execute code multiple times based on a set condition

Control Flow - If Statements

Use **if statements** to do something based on **if a condition is true.**

Parentheses around the condition are **optional**.

```
if 2 > 1 {
    print("Yes 2 is greater than 1")
}
```

Control Flow - If Statements

You can check if a variable is **of a specific type** using **is operator**

```
if shape is Rectangle {
    print("Shape is of type rectangle")
}
```

Control Flow - Switch Statements

Switch statements are primarily useful for comparing a value of any data type against a number of potential cases

Every switch statement has a **default** case

```
let age = 25
switch age {
case 18...35:
    print("Cool demographic")
default:
    break
}
```

Control Flow - For-in

Iterate through an array using a temporary value within a for-in loop

Note that the **score value** is only **temporary** and is created by the for-in to iterate through **individualScores array**

```
let individualScores = [75, 43, 103, 87, 12]
for score in individualScores {
    print(score)
}
```

Control Flow - While Loops

While loops will repeat a block of code while a condition is true

```
while shields > 0 {
    print("Fire blasters!")
}
```

Control Flow - Repeat While

Repeat-while has replaced the traditional do-while and only runs a loop while a condition is true

The condition is checked at the **end** of the loop

```
repeat {
    print("Fire blasters!")
} while shields > 0
```

Functions

A function is a block of code that performs a task

A function can be created using the keyword **func**

Call a function by following its name with a list of arguments in parentheses.

```
func printGreeting() {
print("Hello, playground.")
}
```

Functions w/ Parameters

Declare the name of the parameter following a ‘:’ and the type of parameter

Multiple parameters have a comma after them

```
func printPersonalGreeting(name: String) {
print(“Hello \(name), welcome to your playground.”
}
```

Functions w/ Return

```
func greet(name: String, day: String) -> String {
    return “Hello \(name), today is \(day).”
}
```

A return value is denoted by the -> syntax, which indicates that the function will return an instance of the specified type.

My function would return “**Hello Mark today is Monday.**”

Tuples

Use a tuple to make a compound value of different types. The elements of a tuple can be either be a name or be a number

```
let tuple = (5, “tuple”)
```

For example, to return a tuple from a function:

```
func calculateScore(scores: [Int]) -> (min: Int, max: Int, sum: Float) {}
```

Collections

Collections are **flexible containers** that let you store any number of **values together**

Create **mutable** and **immutable** collections by using **var** or **let**

Collections can be **arrays**, **dictionaries or sets**

Collections - Arrays

An array is an ordered collection of values of the same type.

Arrays are not the same as tuples, elements in arrays can **only** be of the same type.

An array can be defined in a few different ways within Swift syntax.

Collections - Arrays

var players = [“Alice”, “Bob”, “Cindy”, “Dan”]

var players = Array(repeating: 0, count: 5)

var players: [String] = [“Alice”, “Bob”, “Cindy”, “Dan”]

var players = [String]()

Values	Index
Craig	0
Don	1
Anna	2

Collections - Arrays

You can iterate through an array's indices within a for loop

```
var players = 1...10

for player in players {
    print(players)
}
```

Collections - Sets

A set stores distinct values of the same type in a collection with no defined ordering.

You can use a set instead of an array when the order of items is **not important, or when you need to ensure that an item only appears once.**

Craig Anna Brian Donna

Collections - Sets

```
var favoriteGenres: Set<String> = [“Rock”, “Classical”, “Hip hop”]

var favoriteGenres: Set = [“Rock”, “Classical”, “Hip hop”]
```

Collections - Dictionaries

A dictionary is an unordered collection of pairs, where each pair is comprised of a **key** and a **value.** You use a dictionary **when you need to look up values based on their identifier.**

Keys and values can be of **different types**

```
var namesAndScores = [“Anna”: 2, “Brian”: 2, “Craig”: 8, “Donna”: 6]
print(namesAndScores)
```

Collections - Dictionaries

Access a dictionary using a **key value**, instead of **indices**.

print(**namesAndScores["Anna"]**)

Keys	Values
Craig	8
Anna	2
Brian	2
Donna	6