Swift

Declaring

< Type Annotations>

- placing a colon after the variable name, followed by a space, followed by the name of the type to use.
  - colon means "…of type…"

```
var welcomeMessage: String
```

- define multiple variables of the same type on a single line

```
var red, green, blue: Double
```

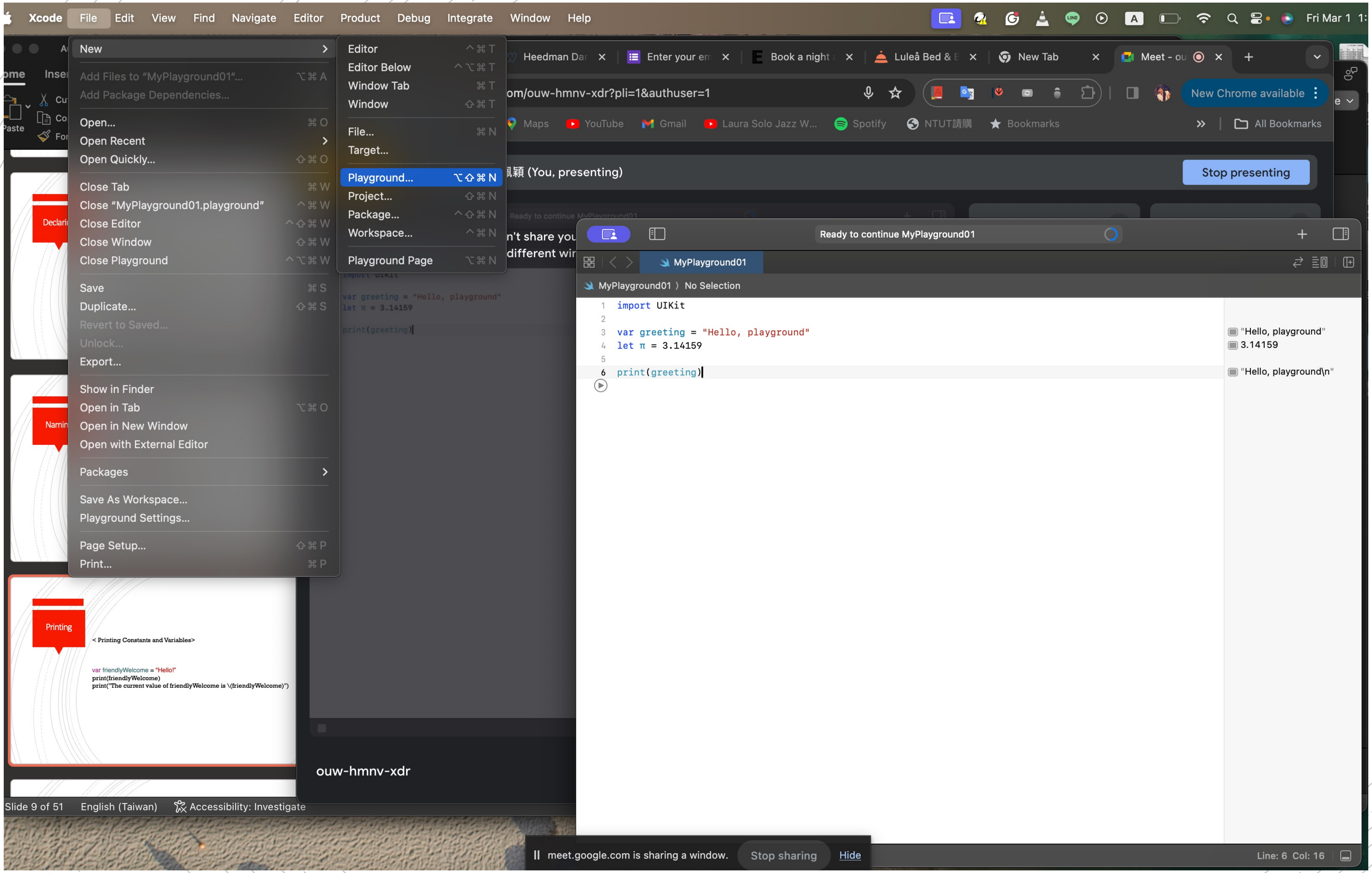
- It's rare that you need to write type annotations in practice.
- If you provide an initial value for a constant/variable at the point that it's defined, Swift can almost always infer the type to be used.
  - In the welcomeMessage example above, no initial value is provided, and so the type of the welcomeMessage variable is specified with a type annotation rather than being inferred from an initial value.

Naming

- < Naming Constants and Variables>
  - Constant/variable names can contain almost any character, including Unicode characters:

```
let π = 3.14159
let 你好 = "你好世界”
let 🐶🐮 = "dogcow"
```

  - Constant and variable names can’t contain
    - whitespace characters
    - mathematical symbols
    - Arrows
    - begin with a number
    - private-use (or invalid) Unicode code points,
    - line- and box-drawing characters



```
import UIKit

var greeting = "Hello, playground"
let π = 3.14159

print(greeting)
```

```
"Hello, playground"
3.14159
"Hello, playground\n"
```

< Printing Constants and Variables>

```
var friendlyWelcome = "Hello!"
print(friendlyWelcome)
print("The current value of friendlyWelcome is \(friendlyWelcome)")
```

Printing

< Printing Constants and Variables>

```
var friendlyWelcome = "Hello!”
print(friendlyWelcome)
print("The current value of friendlyWelcome is \(friendlyWelcome)")
```

Semicolons

< Semicolons>

- Swift doesn’t require you to write a semicolon (;) after each statement
- although you can do so if you wish
- However, semicolons *are* required if you want to write multiple separate statements on a single line:

```
let cat = "🐱" ; print(cat)
```

The Basics

< Type Safety and Type Inference>

- Swift is a *type-safe* language.
  - If part of your code requires a String, you can't pass it an Int by mistake.
  - Swift performs *type checks* when compiling your code and flags any mismatched types as errors.
  - Type-checking helps you avoid errors when you're working with different types of values.
- *Type inference*
  - You do not have to specify the type of every constant and variable
  - Type inference enables a compiler to deduce the type of a particular expression automatically when it compiles your code, simply by examining the values you provide.

```
let meaningOfLife = 42
```

if you assign a literal value of `42` to a new constant without saying what type it is,
Swift infers that you want the constant to be an `Int`,
because you have initialized it with a number that looks like an integer

Optional

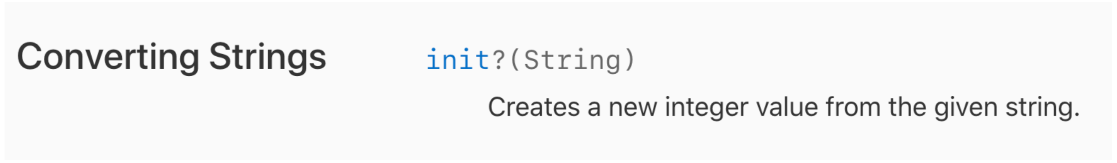
- Optionals are a special feature in Swift used to indicate that an instance may not have a value
  - “there *is* a value, and it equals *x*”
  - or “there *isn’t* a value at all”
- Similar to using nil with pointers in Objective-C,
  - but they work for any type, not just classes.
  - Not only are optionals safer and more expressive than nil pointers in Objective-C, they’re at the heart of many of Swift’s most powerful features

Optional

- A value may be absent.
- An optional represents two possibilities:
  1. there *is* a value, and you can unwrap the optional to access that value,
  2. there *isn't* a value at all.

- For example:
  - How optionals can be used to cope with the absence of a value?
    - Swift's Int type has an initializer which tries to convert a String value into an Int value.
    - However, not every string can be converted into an integer.
    - The string "123" can be converted into the numeric value 123, but the string "hello, world" doesn't have an obvious numeric value to convert to.

```
let three = Int("3")  // returns an optional Int
let a = Int("Hello, world") //????? (the result is nil)
```



Converting Strings `init?(String)`

Creates a new integer value from the given string.

Optional

- You set an optional variable to a valueless state by assigning it the special value nil:

```
var serverResponseCode: Int? = 404
//serverResponseCode contains an actual Int value of 404
serverResponseCode = nil
//serverResponseCode now contains no value
```

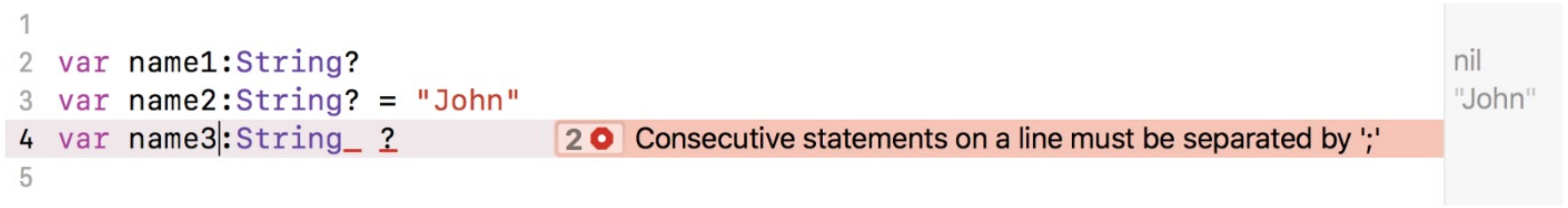
You can’t use **nil** with non-optional constants and variables.

- The default value of an optional variable is nil

```
var surveyAnswer: String?
// surveyAnswer is automatically set to nil
```

Optional

- 在型別後面加上？表示變數是個 Optional。切記問號需緊貼著型別，型別與？之間不可留空白



```
var name1:String?
var name2:String? = "John"
var name3:String_ ?
```

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

nil
"John"

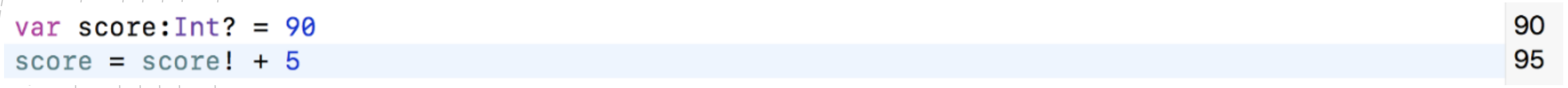
Optional

```
let intValue:Int = 0                    // OK!
let intValue2:Int = nil                 // No!
let optionalIntValue:Int? = nil         //Yes!

var i = 0, j= 10
var k = i + j                           //OK

var a:Int? = 10
var b:Int? = 12
var c = a + b                           //No，Int? 不是 Int
```

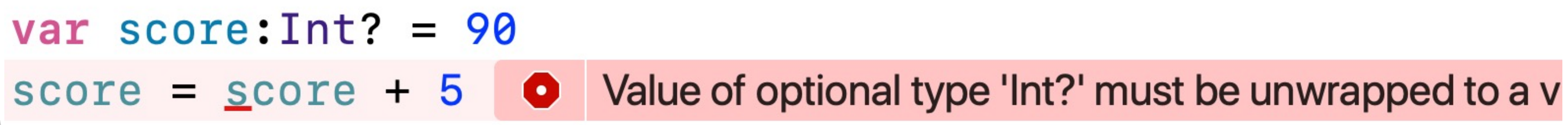
Forced Unwrapping

```
enum Optional<T> {
  case Some(T)
  case none
}
```

- Optional 是個包裝(wrapp) 型別的容器，所以當需要取出來使用時需要解開包裝，而！(驚嘆號)就是用來解開包裝的



```
var score:Int? = 90          90
score = score! + 5           95
```



```
var score:Int? = 90
score = score + 5    Value of optional type 'Int?' must be unwrapped to a v
```

Optional

- Implicitly Unwrapped Optional (自動取值)
- 適合在大部分的情況都是有值的時候

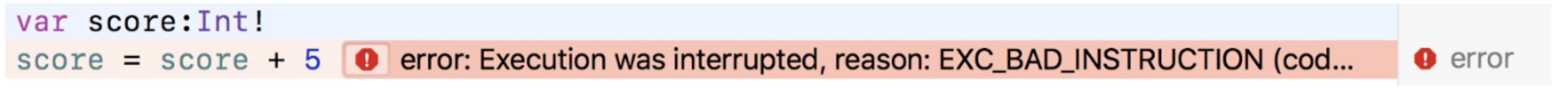
```
var score:Int! = 90
score = score + 5
```

```
90
95
```

Optional

- Implicitly Unwrapped Optional (自動取值)

- **不能沒有給初始值就使用！**
- 會造成程式Crash



```
var score:Int!
score = score + 5    error: Execution was interrupted, reason: EXC_BAD_INSTRUCTION (cod...    error
```

練習

- 用? 或 ! 宣告 Optional 變數
- 如何相加兩變數?

var var1:Int! = 9

var var2:Int? = 3

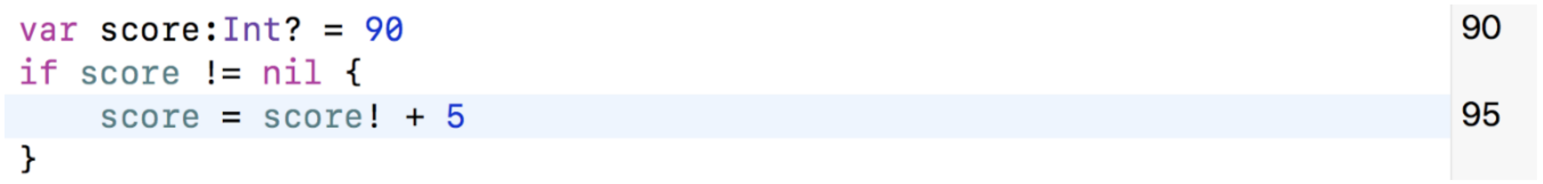
```
var var1:Int! = 9

var var2:Int? = 3

var var3 = var1+var2!
```

Optional Unwrap

```
//如果是沒有東西的包裹
if let 沒有東西 = 沒有東西的包裹 {
   // 不會執行這裡
} else {
  // 這裡會執行
}
```

Optional Unwrap

- 利用if檢查是否有值，如果不是nil，再使用！讀取



```swift
var score:Int? = 90
if score != nil {
    score = score! + 5
}
```

```
90

95
```

Optional Unwrap

使用 if … let 先判斷再解開

```
var x:String?="Hello World"
if let y=x {
    print(y)
}
```

```
var x2: String? = nil
if let y = x2 {
    print("has a Value = \(y)")
}else{
    print("Error! no value")
}
```

// Output
Hello World
Error!!!

Optional Unwrap

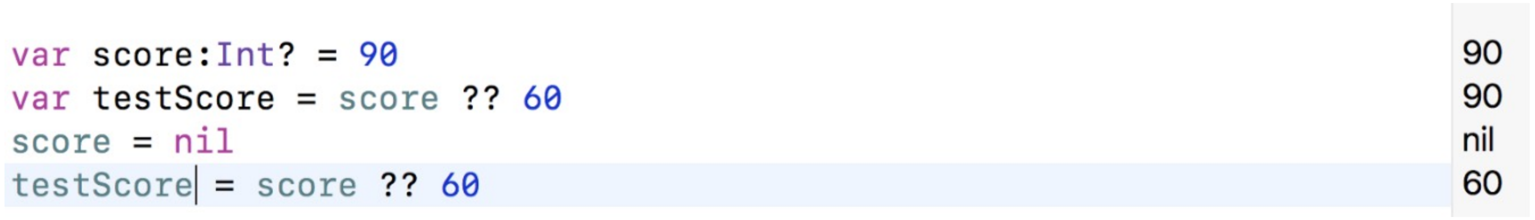
Exercise

Declare two optional-Int variables
Print the sum

```
var x1:Int?=3
var x2:Int?=4
if let y1=x1 {
    print(y1)
    if let y2=x2 {
        print(y2)
        print(y1+y2)
    }


}
```

Optional Unwrap

- 利用判斷式



```
var score:Int? = 90                    90
var testScore = score ?? 60            90
score = nil                            nil
testScore = score ?? 60                60
```

optional

- Swift is a *type-safe* language
  - helps you to be clear about the types of values your code can work with.
    - E.g. If part of your code requires a String, type safety prevents you from passing it an Int by mistake.
  - Also prevents you from accidentally passing an optional String to a piece of code that requires a non-optional String.

Declaring

<Declaring Constants and Variables>

If a stored value in your code won't change, always declare it as a constant with the let keyword. Use variables only for storing values that need to be able to change.

- declare constants with the let keyword

```
let maximumNumberOfLoginAttempts = 10
```

- declare variables with the var keyword

```
var currentLoginAttempt = 0
```

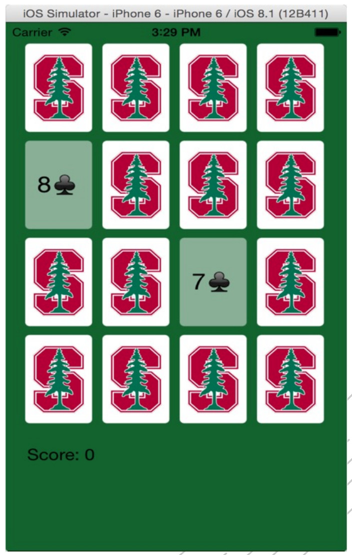
- declare multiple constants or multiple variables on a single line, separated by commas:

```
var x = 0.0, y = 0.0, z = 0.0
```

Lab – Card Game

3/23/2018

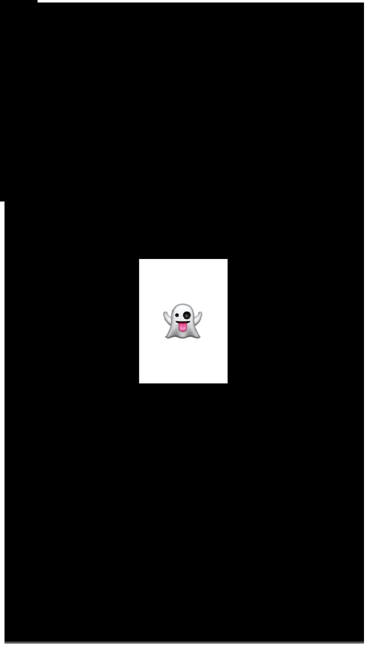
- Do not count flip on matched (disabled) card
- Keep matched cards facing up
- Flip the selected card
- Shuffle the card
- Start over a game



```
class ViewController: UIViewController {



    @IBAction func touchCard(_ sender: UIButton) {

        flipCard(withEmoji: " 👻 ", on: sender)

    }

    func flipCard(withEmoji emoji: String, on button:UIButton){

    }
}
```

External name

Internal name



```
func flipCard(withEmoji emoji: String, on button:UIButton)->Void

func flipCard(withEmoji emoji: String, on button:UIButton)->()
```

Same external name and internal name

```swift
func hello(name: String, age: Int, location: String) {
        print("Hello \(name). I live in \(location). When is your \(age + 1)th 
birthday?")
    }
```

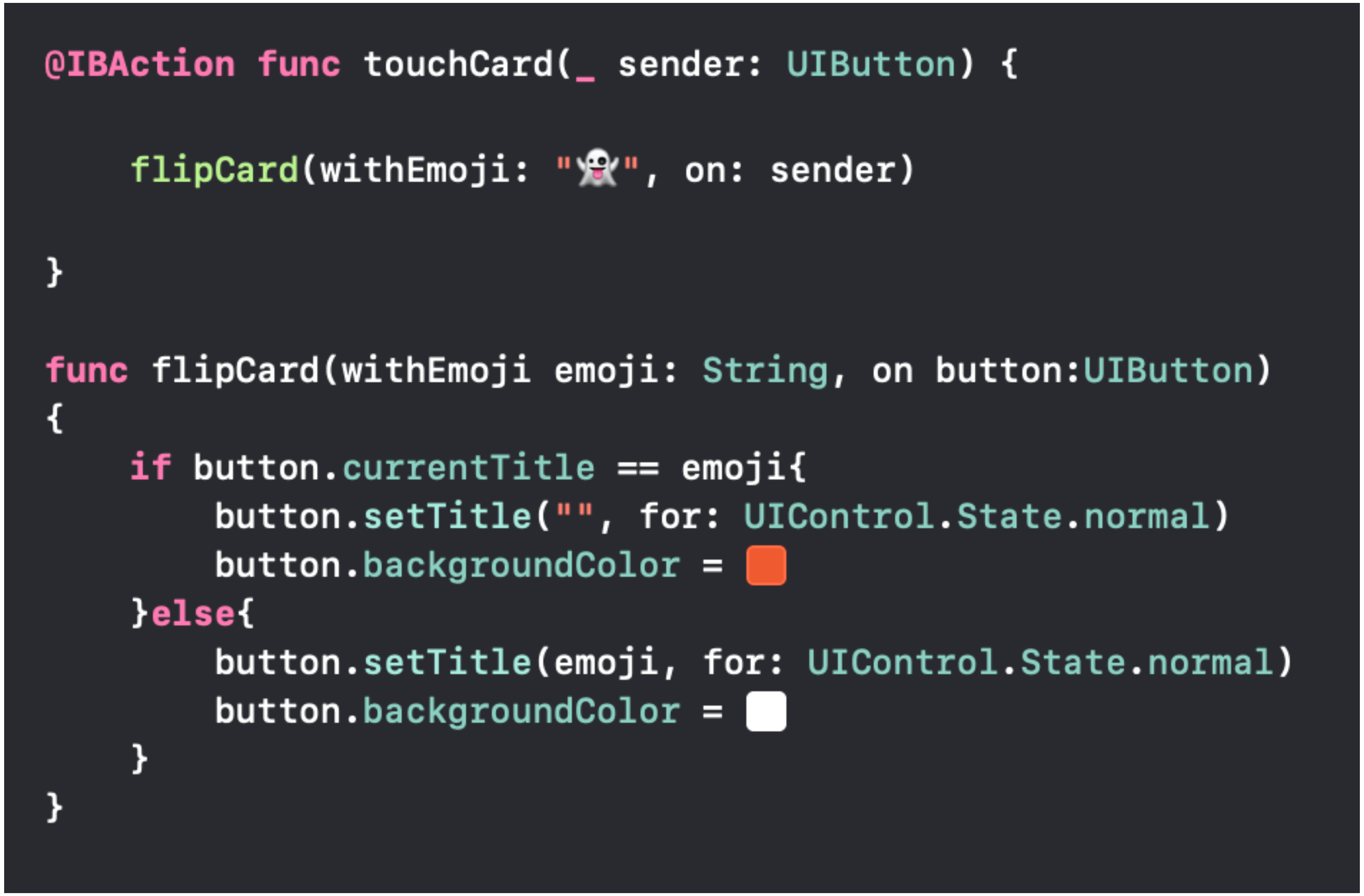
```swift
hello(name:"Mr. Roboto", age:5, location:"San Francisco")
```

If you want to omit an external name you override it with an underscore:

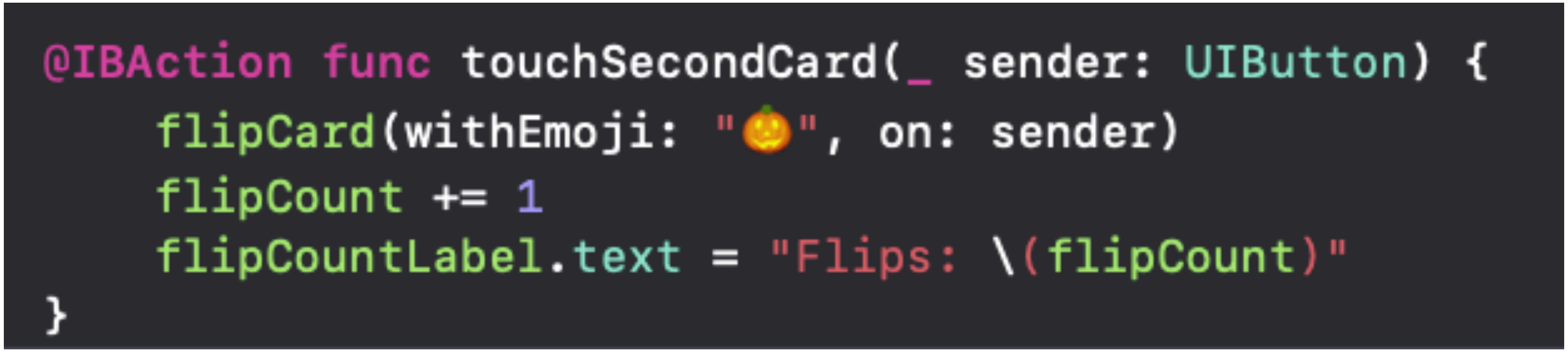
```
init(_ x: Int, _ y: Int) {
    self.x = x
    self.y = y
}
```

This gives us the more concise initializer:

```
let origin = init(0, 0)
```



```swift
@IBAction func touchCard(_ sender: UIButton) {

    flipCard(withEmoji: "👻", on: sender)

}

func flipCard(withEmoji emoji: String, on button:UIButton)
{
    if button.currentTitle == emoji{
        button.setTitle("", for: UIControl.State.normal)
        button.backgroundColor = 🟧
    }else{
        button.setTitle(emoji, for: UIControl.State.normal)
        button.backgroundColor = ⬜
    }
}
```



```swift
@IBAction func touchSecondCard(_ sender: UIButton) {
    flipCard(withEmoji: "🎃", on: sender)
    flipCount += 1
    flipCountLabel.text = "Flips: \(flipCount)"
}
```

Card Matching Game

包含的類別 (class)：

1. **Card**
2. **Deck**
3. **PlayingCard**
4. **PlayingCardDeck**
5. **CardMatchingGame**
6. **CardGameViewController**

Card Matching Game

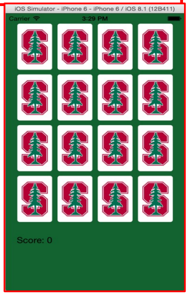
包含的類別 (class)：

1. **Card**
2. **Deck**
3. **PlayingCard**
4. **PlayingCardDeck**
5. **CardMatchingGame**

Model

6. **CardGameViewController**

Controller



View