Swift Fundamentals III
Optionals / Functions

CS112 Unit 4
Max Luttrell, Fall 2016

optionals

- an **optional** is like a variable, but might not hold a value. either:
    - the optional has some value
    - the optional has no value (**nil**)
- we use question mark “**?**” to indicate an optional

optionals

- let's say our program wants to hold the name of a person
- we always will have a first name and a last name; however, we might or might not have a middle name

```
var firstName: String = "John"
var lastName: String = "Adams"
var middleName: String? = "Quincy"
```

nil

- the **nil** keyword represents that the optional contains no value

```
var firstName: String = "John"
var lastName: String = "Adams"
var middleName: String? = "Quincy"

middleName = nil

// now, middleName no longer contains any value
```

using optionals - forced unwrapping

- if an optional holds a value, we can access it by **force unwrapping** it, i.e. put an exclamation point "**!**" after the optional
- careful! if you force unwrap an optional, and there's no value inside, you get a run time error

```
var firstName: String = "John"
var lastName: String = "Adams"
var middleName: String? = "Quincy"

middleName = nil

// now, middleName no longer contains a value
// so the below will crash your program!!
print(middleName!)
```

using optionals - forced unwrapping

- if we don’t specify a value for an optional, it is initialized to nil by default

```
var firstName: String = "John"
var lastName: String = "Adams"
var middleName: String?

// middleName doesn’t contain a value
// so the below will crash your program:
print(middleName!)
```

if statement with forced unwrapping

- first we make sure that our optional isn't nil; then, we can safely use its value with forced unwrapping

```
var firstName = "John"
var lastName = "Adams"
var middleName: String?

middleName = "Quincy"

print(firstName, terminator: " ")

if middleName != nil {
    print(middleName!, terminator: " ")
}

print(lastName)
```

Sample Debug Output

John Quincy Adams

if statement with forced unwrapping

- here, middleName is nil

```
var firstName = "John"
var lastName = "Adams"
var middleName: String?

print(firstName, terminator: " ")

if middleName != nil {
    print(middleName!, terminator: " ")
}

print(lastName)
```

Sample Debug Output

```
John Adams
```

optionals:
another example

- it's possible that a String can contain an integer number, if it only holds digits

```
var riders: String = "123456"
```

- let's say we want to add one to riders. if we use the + operator on Strings, we don't get addition. we append instead!

```
var riders: String = "123456"
riders = riders + "1"
print("Riders: \(riders)")
```

Sample Debug Output

```
Riders: 1234561
```

another example

- we can use the Swift Int initializer to try to convert a String to an Int. this can only work if the String is in fact an Int!

- here, we store the result of the Int initializer in an Int optional. if the conversion is a success, ridersInt gets the converted value from ridersString. if it's not a success, it gets nil

```
var ridersString: String = "123456"
var ridersInt: Int?
ridersInt = Int(ridersString)
```

```
if ridersInt != nil {
    ridersInt! += 1
    print(ridersInt!)
}
```

Sample Debug Output

123457

optional binding

- another way to find out whether an optional contains a value and use it is **optional binding**

```
var ridersString: String = "123456"
if var ridersInt = Int(ridersString) {
    ridersInt += 1
    print(ridersInt)
}
```

- this can be read as: if the optional Int returned by Int(ridersString) contains a value, set (regular Int) ridersInt to this value and execute the statements inside the if statement; if it's nil, don't do anything

Sample Debug Output

123457

optional binding

- here, ridersString isn't an integer

```
var ridersString: String = "blah"
if var ridersInt = Int(ridersString) {
    ridersInt += 1
    print(ridersInt)
}
```

Sample Debug Output

Exercise 4A

- I have started a playground for you below to track some scores for a game using an array of Int **optionals**

- write a loop which determines how many days had games (i.e. a score which isn't nil)

- write another loop which adds up scores from the days with a game (i.e. computes the total score)

```
// Here, we create an array to track scores.  Each element
// contains a score for a given day; a nil means there was no
// game that day.  Thus, our array contains Int optionals.
let scores: [Int?] = [2, nil, 6, 3, nil, 5, 1, 9, nil, 0]
```

functions

- Swift, like many programming languages, provides **functions**
- a function is a self-contained chunk of code to do some task

```
//print a welcome
…

//compute fizzbuzz
…

//print result
…

//compute fizzbuzz
…

//print result
…
```

```
function printWelcome
```

```
function computeFizzBuzz
```

```
function printResult
```

functions

- to use functions, we first must define what it does:

```
func helloWorld() {
    print("Hello World!")
}
```

Sample Debug Output

functions

- now that we've defined our function, we can **call** it from other parts of our program

```
func helloWorld() {
    print("Hello World!")
}

helloWorld()
```

Sample Debug Output

```
Hello World!
```

functions

- we can call the function as many times as we want:

```
func helloWorld() {
    print("Hello World!")
}

helloWorld()
helloWorld()
```

Sample Debug Output

```
Hello World!
Hello World!
```

functions

- we can call a function from within a loop

```
func helloWorld() {
    print("Hello World!")
}

for i in 1...5 {
    helloWorld()
}
```

Sample Debug Output

```
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
```

parameters

- we can also pass data to our functions, these are known as **parameters**

```
func helloName(firstName: String) {
    print("Hello \(firstName)!")
}
```

parameter name

parameter type

Sample Debug Output

parameters

- since function helloName takes a parameter of type String, we must pass it a String when we call it

```
func helloName(firstName: String) {
    print("Hello \(firstName)!")
}

let myName = "Max"
helloName("Chan")
helloName(myName)
```

Sample Debug Output

```
Hello Chan!
Hello Max!
```

return value

- we can also return a value from a function

```
func createGreeting(firstName: String) -> String {
    var greeting: String
    greeting = "Hello " + firstName + "!"
    return greeting
}

print(createGreeting("Juan"))

var boGreeting: String
boGreeting = createGreeting("Bo")
print(boGreeting)
```

return type

Sample Debug Output

```
Hello Juan!
Hello Bo!
```

sumArray

- this function takes an array as parameter and returns its sum

```
// sumArray returns the sum of elements in
// an Int array
func sumArray(array: [Int]) -> Int {
    var total = 0
    for element in array {
        total += element
    }
    return total
}

// here is some code demonstrating calling sumArray()
let scores = [3, 6, 15, 1, 0, 8]
let total = sumArray(scores)
print("The array has a total of: \(total)")
```

Sample Debug Output

The array has a total of: 33

functions:
the big picture

- think of a function as a black box. the rest of the program doesn't care how a function works, it only needs to know how to use it

- you can create functions for common tasks

Exercise 4B

- use the same array from exercise 4A which tracks scores
- I have created a “stub” function below, gamesPlayed, which takes one parameter, an array of Int optionals. It currently just returns 0. Improve it so that it returns how many games were played, i.e. the number of days with a score that isn’t nil
- outside of your function, call the function with parameter scores, and print out the result
- if you have time, write another function which returns the sum of all the scores for days with games played, and call it to print out the result

```
// Here, we create an array to track scores.  Each element
// contains a score for a given day; a nil means there was no
// game that day.  Thus, our array contains Int optionals.
let scores: [Int?] = [2, nil, 6, 3, nil, 5, 1, 9, nil, 0]

// gamesPlayed returns the number of games played in
// gamesPlayed, i.e. the number of elements which are not
// nil
func gamesPlayed(scores: [Int?]) -> Int {
    return 0
}
```