Welcome back

Or welcome for the first time

We talked about

- Swift
- Variables & Constants
- Standard data types
- Strings

Prefix and Suffix Comparisons

- Simple way to check the beginning and end of strings

- Most of the time when we're scanning for substrings we know if it should be at the beginning or end

```
let theCurrentCurrency = "U. S. Dollars"
if theCurrentCurrency.hasPrefix("U") {
    print ("U know it")
}
```

Functions in Swift

- A function is a well-defined unit of work (informal definition)

- Swift functions can run from simple C-syle functions to complex Objective-C-style methods with named parameters

- Functions have a type

Defining a function

- precede with "func" keyword
- functions should retain camelCase style naming
- Parameters, with optional names, in parenthesis
- Return type provided with an ->

Simple function

```
func doSomethingUseful(numberToUse: Int) -> String {
    return "I did something useful with \(numberToUse)"
}
print(doSomethingUseful(numberToUse:3))
```

Defining parameters

- Parameters have a local name, which must be unique to the function
- Parameters have a type

Default Parameters

- Parameters may have a default value

```
func functionWithDefault(someParameter:  Int = 42) {
    print("I was handed \(someParameter)")
 }
functionWithDefault()
functionWithDefault(someParameter: 13)
functionWithDefault(someParameter: 42)
```

Named parameters

- Parameters have local names
- Parameters may have external names
- func myFunc(externalName internalName: Int)
- You must double up on parameter names to benefit from named parameters
- Only first parameter can be unnamed externally, second and subsequent parameters will default to local name if an external name is not provided

Variadic Parameters

- Not a common use case, but you will see these
- Variadic parameter must be final parameter in list

```
func workWithIntegers(values: Int...) {
    var sum = 0
    for value in values {
        sum += value
    }
    print("it all adds up to \(sum)")
}
workWithIntegers(5,4,3,2,1)
```

In/Out Parameters

- Edits a variable in place
- This should seem familiar : C-style pointer notation

```
func reverseString(inout toReverse: String) {
    toReverse = String(toReverse.characters.reverse())
}
var stringToReverse = "panama"
reverseString(&stringToReverse)
```

Return types

- All functions return a value
- If not specified, return type is void
- "void" means an empty tuple
- Most return types will be plain type
- Optionally return tuples : multiple values
- Do not overuse tuples! This will be viewed as a code smell!

Function Types

- Same as a variable type
- Defined by return type and number / types of parameters

Nested Functions

- A function can contain other functions for local use
- Due to function type naming, it is easy to return a local function to a caller for future use, syntax does get wordy

```
func getTransformFunction(kind: Int) -> ((Int) -> Int) {
    func transformer(inVal: Int) -> Int {
        return inVal * 2
    }
    return transformer
}
```

Flow Control

- All standard C flow control methods
- for-in loop from Objective-C
- Very powerful Switches

Traditional for loop

- Follows lexical scope from C
- index no longer valid when loop ends
- This loop will no longer be valid in Swift 3.0 and beyond.

```
for var index = 0; index < 3; ++index {
    print(index)
}
```

for…in loops

- As used in Objective-C

```
for index in 1...5 {
    print(index)
}
```

for...in loops

- Can also use tuples from dictionary

```
let dictionaryEntries = ["a":1,"b":2,"c":3]
for (key, value) in dictionaryEntries {
    print("the value for \(key) is \(value)")
}
```

while / repeat...while

- As from C

```
var x = 0
while x < 100 {
    print(x)
    x += 1
}

var x = 0
repeat {
    print(x)
    x += 1
} while x < 100
```

while vs repeat… while

- As with C, the difference is when the exit condition is checked.
- while checks the condition before executing the loop
- repeat...while checks the condition after executing the loop
- repeat...while is the same as do...while in C

Have you noticed?

- Parenthesis are entirely optional in Swift
- Curly braces are non-optional in swift

if statements

```
var todaysDate = 8
enum Days { case Monday, Tuesday, Wednesday, Thursday,
Friday }
let todaysDay = Days.Monday

if todaysDate == 13 {
    print("be careful out there")
}

if todaysDate == 13 && todaysDay == .Friday {
    print("more caution advised")
}
```

Adding some else-es

```
let temperatureInCelsius = 32
if temperatureInCelsius < 0 {
    print("baby it’s cold outside")
} else if temperatureInCelsius > 100 {
    print("you’re dead")
} else {
    print("who can tell? it’s celsius!")
}
```

Enumerations in Swift

- Groups of related values
- Provides type safety
- Not restricted to integral values

Enumeration sample

```
var directionForTravel = CompassPoint.South
directionForTravel = .East
switch directionForTravel {
    case .North:
        print("Visiting santa, eh?")
    case .South:
        print("Antarctic research, how exciting!")
    case .East:
        print("East of what, exactly?")
    case .West:
        print("Heading to the mothership?")
}
```

Associated Values in Enumerations

```
enum Instrument {
    case Guitar(Int)
    case Drum(Double, Double)
}
//This gives us an enumeration of a guitar where we are told the number of strings, or a dr
with its diameter and depth
var eightStringGuitar = Instrument.Guitar(8)
```

Raw Values

```
enum Planet: Int {
case Mercury = 1, Venus, Earth, Mars, Jupiter, Saturn,
Uranus, Neptune
}
let marsOrder = Planet.Mars.rawValue
```

An enum is so much more!

- Remember to think of an enum as a new type of data
- It's not a placeholder for a single value, it's closely related to a class or struct
- Move switching method into an enum, rather than outside.