intro to Swift

Instructors: John Clem, Brad Johnson
TA's: Cole Bratcher, Dan Fairbanks, Reed Sweeney

Swift tips

- Swift, Xcode6, iOS8, OSX 10.10 are still in beta
- Apple wants your feedback
  - (and reserves the right to make sudden changes to everything)
- Please ask relevant questions during the corresponding lecture,
- We have two Q&A sessions as well as hands-on lab time

meet Swift

- concise
- expressive
- safe
- fast
- modern

Objective-C without the C

- Same runtime
- Same LLVM compiler & LLDB debugger
- Same Cocoa & Cocoa Touch Frameworks
- ‘Spiritual Successor’ to Objective-C

variables and constants

- var for variable variables (value can be set to a different value)
- let for constant variables (value cannot be set to different value)
- similar to mutable and immutable BUT
- let isn't just for constants

primitive types

primitive types : Int

- Int8,Int6,Int32,Int64 bit
- unsigned can’t be negative
- You usually don’t choose, just use Int!

primitive types : Float, Double

- use Float for 32-bit
- use Double for 64-bit
- Double is the default inferred type

primitive types : Float, Double

```
var myFloat : Float = 1.0

var myDouble = 1.0

var myInt = 1
```

number conversions

- Fundamentally different from the Objective-C conversion system
- Must cast with Type() for any combination of varying number types
- Safer
- Faster

```
var : String
```

var : String

- var declares mutable String, let declares immutable
- No longer a reference type, copied when passed
- String interpolation

```
var name = “johnny”
name += “clem”
var githubName = “@“ + name
var gmailName = name + “@gmail.com”
```

var : String

- Strings are just unicode character arrays
- let cats = [😼, 😾, 😿]
- Iterating over a String

```
var name = “johnny”
for character in name {
  // do stuff
}
```

```
var : Array<T>
```

var : Array<T>

- strongly typed array, not just a random-object-container
- use isEmpty property to check if count is 0
- append() or += to add items to end of array

```
var : Dictionary<T1, T2>
```

var : Dictionary<T1, T2>

- Always clear about type of values AND keys
- Dictionary<KeyType, ValueType>
- .keys and .values properties (for loop)

functions

functions

- self-contained chunks of code
- functions have names that are used to call the function
- parameters are comma separated

```
func doSomethingWith( object : AnyObject?) {
    object.doSomething()
}
```

functions

- parameters and return values are not required
- Tuples allow functions to return multiple values
- return type denoted with ->

```
func fullName( firstName : String, lastName : String) -> String {
    return firstName + “ “ + lastName
}
```

methods

- methods are functions associated with a type (class,type,enums)
- methods still use the `func` keyword!
- methods are called just like functions with one difference:
  - parameter names in methods are also used when calling the method (except for the first one!!!)

tuples

- values in a tuple can be of any type and different types
- no limit on how many values inside
- use _ to ignore parts of the tuple when decomposing

typeInference

- if you don't specify the type, Swift will work out the appropriate type.
- far fewer type declarations than Objective-C.
- give variables a default value, or declare their type
- AnyObject is like the Objective-C id or instancetype
- you can easily type-cast objects, as long as it's a downcast

```
var tires = 4.0023
let tiresInt = tires as Int
```

optionals!?

- variables whose value may or may not be present
- Swift does not allow you to leave properties in an undetermined state.
- variables must either:
  - be given a default value
  - have their value set in the initializer
  - be marked as optional using either the ? or ! symbol