Meanwhile, outside the Google universe

An introduction to Swift

About me

- Iskandar Abudiab
- Software developer @cluetec
- Java backends, native mobile clients
- @_iabudiab
- https://github.com/iabudiab

Agenda

- Data Types
- Control Flow
- Functions & Closures
- Classes & Structs
- Optionals
- Generics
- Extensions
- Protocols

print("Hello Swift!")

- Release June 2014
- Current version 2.1 (October 21, 2015)
- Open Source later in 2015
- iOS, OSX & Linux
- REPL, Playgrounds in Xcode
- Type-Safe, Flexible, Modern, Intuitive, Concise yet expressive

Variables

```
var answerToTheQuestionOfLife = 42 // Type is inferred
```

Variables

```
var answerToTheQuestionOfLife = 42 // Type is inferred
answerToTheQuestionOfLife = 9001
```

Variables

```
var answerToTheQuestionOfLife = 42 // Type is inferred
answerToTheQuestionOfLife = 9001
answerToTheQuestionOfLife = 3.14 // Compile error ~> Int
```

Constants

```swift
var numberOfTalks: Int = 1
let language = "Swift"
```

Constants

```swift
var numberOfTalks: Int = 1
let language = "Swift"
let pi: Double = 3.141592
```

Constants

```
var numberOfTalks: Int = 1
let language = "Swift"
let pi: Double = 3.141592
pi = 3.1415927 // Compile error ~> constant
```

Unicode

```swift
let π: Double = 3.1415927
let 🐶 = 2
let 🐰 = 1
let 👽 = 🐶 + 🐰 // ~> 3

let 🐮 = "Meat"
let 🍞 = "Bread"
let 🍔 = "1x \(🐮) and 2x \(🍞)"

let 🍌 = "For Scale!"
```

Strings

- Unicode compliant and correct
- String ≈ Collection of Characters
- String ≠ Collection
- Strings are not Objects, but rather *structs**
- *Swift will be open-sourced ~> details about the internals will be revealed!*

Strings

```
var empty = ""; var another = String()
var hello = "Hello "
let greeting = hello + "Swift"

let earth = "This is planet 🌍"
earth += "It also has a moon!" // Compile error
```

Characters

```
let wildcard: Character = "*"
let cthulhu: [Character] = ["C","t","h","u","l","h","u","🐙"]
```

Strings

```
// Å: ANGSTROM SIGN U+212B
let angstrom1 = "\u{212b}"
// A + ˚: A + COMBINING RING ABOVE U+030A
let angstrom3 = "A\u{30a}"

angstrom1 == angstrom2     // ?
angstrom1.characters.count // ?
angstrom2.characters.count // ?
```

Strings

```
// Å: ANGSTROM SIGN U+212B
let angstrom1 = "\u{212b}"
// A + ˚: A + COMBINING RING ABOVE U+030A
let angstrom3 = "A\u{30a}"

angstrom1 == angstrom2     // ~> true
angstrom1.characters.count // ~> 1
angstrom2.characters.count // ~> 1
```

Strings

```
// é: U+00E9 LATIN SMALL LETTER E WITH ACUTE
let first = "caf\u{e9}"
var second = "cafe"

// ´: U+030A COMBINING ACUTE ACCENT
second += "\u{301}"
second.characters.count // ?

first == second         // ?
```

Strings

```
// é: U+00E9 LATIN SMALL LETTER E WITH ACUTE
let first = "caf\u{e9}"
var second = "cafe"

// ´: U+030A COMBINING ACUTE ACCENT
second += "\u{301}"
second.characters.count // ~> 4

first == second         // ~> true
```

Strings

- String comparison uses the Unicode un-tailored collation algorithm, i.e. locale insensitive

```
"ampère" < "ångström" // ~> true
"ångström" < "bacon"  // ‘< bacon’ is always true!
```

Collections

```
var ints = Array<Int>()
ints = [Int]()
var doubles: [Double] = [3.14, 42]

var animals = ["🐼", "🐢"]
animals.append("🐰")
animals += ["🐥", "🐍"]
animals.indexOf("🐢")
animals.filter(...)
animals.reduce(...)
animals.map(...)
```

Collections

```
var languages = Dictionary<String, Int>()
languages = [String:Int]()
languages["Swift"] = 1
languages["Java"] = 2
```

Optionals

```
var possible = languages["Go"]
```

Optionals

```
var possible: Int? = languages["Go"]
```

Optionals

```swift
var possible: Int? = languages["Go"]

if possible != nil {
  print("Go is \(possible!)")
} else {
  print("It's a no Go")
}
```

Forced Unwrapping

Optionals

```swift
var possible: Int? = languages["Go"]

if possible != nil {
  print("Go is \(possible!)")
} else {
  print("It's a no Go")
}
```

Optionals

```
var possible: Int? = languages["Go"]

if let actualValue = possible {
  print("Go is \(actualValue)")
} else {
  print("It's a no Go")
}
```

Optional Binding

Optionals

```
var movies: [String]? = ["Rambo", "Matrix", "Snatch"]

if let count = movies?.count {
  print("I have \(count) movies")
} else {
  print("Movies collection is undefined")
}
```

Optionals

```
var x = Int("1000")
let y = Int("337")
let z = Int("11")
if let a = x {
  if let b = y {
    if let c = z {
      if c != 0 {
        print("(x + y) / z = \((a + b) / c)")
      }
    }
  }
}
```

Optionals

```
var x = Int("1000")
let y = Int("337")
let z = Int("11")
if let a = x, b = y, c = z where c != 0
{
  print("(x + y) / z = \((a + b) / c)")
}
```

Control Flow

```
if thisTrue {…} else if thatTrue {…} else {…}

while this != that {…}

repeat {…} while this != that
```

- No parenthesis

Control Flow

```
for index = 0; index < 10; ++index {…}
```

Control Flow

```
for index = 0; index < 10; ++index {…}
for index in 1...10 {…} // ~> range [1..10]
for index in 1..<10 {…} // ~> range [1..10)
for _ in 1...10 {…}     // ~> ignore index
```

Control Flow

```
let animals = ["🐼", "🐢", "🐍"]
for animal in animals {…}

let numberOfLegs = ["spider": 8, "ant": 6, "tiger": 4]
for (animal, legCount) in numberOfLegs {…}
```

Control Flow

```
let grade = 77
switch grade {
  case 0..<20: fallthrough
  case 20..<50: print("fail")
  case 50..<100: print("not bad")
  case 100: print("nice")
  default: print("no such grade")
}
```

- Ranges!
- No implicit fall-through!
- Compile error if not exhaustive!

Control Flow

```
let point = (3, 5)
switch point {
  case (0, 0): print("the origin")
  case (_, 0): print("\(point.0), 0")
  case (-1...2, -2...7): print("\(point.0, point.1)")
  case (let x, 0): print("x is \(x)")
  case let (x, y): print("\(x), \(y)")
}
```

- Tuples
- Ranges in tuples
- Value bindings!

Control Flow

```
let point = (3, 5)
switch point {
  case let (x, y) where x == y: print("x == y")
  case let (x, y) where x == y - 4: print("x == y - 4")
  case let (x, y): print("arbitrary")
}
```

- “where” clauses!

Control Flow

```
let coolStuff = ["Swift", "Objective-C", "C", "C++"]
let language = "Swift"
switch language {
  case "Java", "C++":
    print("...")
  case let lang where lang.hasPrefix("Ja"):
    print("Starts with Ja")
  case let lang where coolStuff.contains(lang):
    print("Something cool")
  default:
    print("...")
}
```

- Multiple values per case
- “where” clauses with value-binding!

Control Flow

```
let coolStuff = ["Swift", "Objective-C", "C", “C++"]

for lang in coolStuff {
  print(lang)
}
```

Control Flow

```
let coolStuff = ["Swift", "Objective-C", "C", “C++"]

for case let lang in coolStuff where lang.hasPrefix("C") {
  print(lang)
}
```

Control Flow

```
let coolStuff = ["Swift", "Objective-C", "C", “C++"]

for case let lang in coolStuff where lang.hasPrefix("C") {
  print(lang)
}

var tuple = (x: 3, y: 5)
while case let (x, y) = tuple where x < y {
  print("x < y")
  tuple.x++
}
```

- Unified use of “case”

Functions & Closures

```swift
func hello() {
  print("Hello Swift!")
}
```

Functions & Closures

```swift
func foo(x: Int) {
  print("int parameter")
}

func foo(x: Float) {
  print("float parameter")
}

func foo(x: String) -> Bool {
  return x.characters.count > 3
}
```

Parameters & return value can be overloaded

Functions & Closures

```
func multiplyNumber(number: Int, with: Int) -> Int {
  return number * with
}

multiplyNumber(4, with: 10)
```

Parameters can be named!

Functions & Closures

```
func multiplyNumber(number: Int, with: Int) -> Int
```

Functions & Closures

```
func multiplyNumber(_ number: Int, with with: Int) -> Int
```

ignored

named

Functions & Closures

```
func multiplyNumber(_ number: Int, with with: Int) -> Int
```

ignored

named

```
multiplyNumber(4, with: 10)
```

Functions & Closures

```
func multiplyNumber(_ number: Int, with with: Int) -> Int
                     ignored         named

multiplyNumber(4, with: 10)

func multiplyNumber(num x: Int, by y: Int) -> Int {
  return x * y
}
multiplyNumber(num: 4, by: 10)
```

Functions & Closures

```
func read(fileAtPath path: String,
                   offset: Int,
                   length: Int) {…}
read(fileAtPath: "path", offset: 10, length: 1024)


func createFileAtPath(path: String,
         overwriteIfExists: Bool) {…}
createFileAtPath("path", overwriteIfExists: false)
```

- Readability

Functions & Closures

```swift
func getImageInfoAtPath(path: String) -> (String,Int,Int) {
  return ("png", 1920, 1080)
}
```

Functions & Closures

```
func getImageInfoAtPath(path: String) -> (String,Int,Int) {
  return ("png", 1920, 1080)
}

let info = getImageInfoAtPath("path")
print("Image \(info.0, info.1, info.2)")
```

Functions & Closures

```
func getImageInfoAtPath(path: String) -> (String,Int,Int) {
  return ("png", 1920, 1080)
}

let info = getImageInfoAtPath("path")
print("Image \(info.0, info.1, info.2)")

let (_, w, h) = getImageInfoAtPath("path")
print("Image \(w, h)")
```

Functions & Closures

```
func getImageInfoAtPath(path: String) ->
        (type: String, width: Int, height: Int) {
  return ("png", 1920, 1080)
}
```

Functions & Closures

```
func getImageInfoAtPath(path: String) ->
        (type: String, width: Int, height: Int) {
  return ("png", 1920, 1080)
}

let info = getImageInfoAtPath("path")
print("Image \(info.type, info.width, info.height)")
```

Functions & Closures

```swift
let sayHello = {
  print("Hello")
}
```

Functions & Closures

```
let sayHello: () -> () = {
  print("Hello")
}
```

Functions & Closures

```
let sayHello: () -> () = {
  print("Hello")
}
sayHello()
```

Functions & Closures

```swift
let sayHello: () -> () = {
  print("Hello")
}
sayHello()

func sayHelloFunction() -> () {
 print("Hello")
}
sayHelloFunction()
```

Functions & Closures

```swift
func multiplyBy2(x: Int) -> Int { return x * 2 }
```

Functions & Closures

```swift
func multiplyBy2(x: Int) -> Int { return x * 2 }
let doubler = multiplyBy2
```

Functions & Closures

```
func multiplyBy2(x: Int) -> Int { return x * 2 }
let doubler = multiplyBy2

func transform(x: Int, function: (Int) -> Int) -> Int {

}
```

Functions & Closures

```swift
func multiplyBy2(x: Int) -> Int { return x * 2 }
let doubler = multiplyBy2

func transform(x: Int, function: (Int) -> Int) -> Int {
  return function(x)
}
```

Functions & Closures

```swift
func multiplyBy2(x: Int) -> Int { return x * 2 }
let doubler = multiplyBy2

func transform(x: Int, function: (Int) -> Int) -> Int {
  return function(x)
}
transform(3, function: doubler) // ~> 6
```

Functions & Closures

```
func multiplyBy2(x: Int) -> Int { return x * 2 }
let doubler = multiplyBy2

func transform(x: Int, function: (Int) -> Int) -> Int {
  return function(x)
}
transform(3, function: doubler) // ~> 6
transform(3, function: {

})
```

Functions & Closures

```
func multiplyBy2(x: Int) -> Int { return x * 2 }
let doubler = multiplyBy2

func transform(x: Int, function: (Int) -> Int) -> Int {
  return function(x)
}
transform(3, function: doubler) // ~> 6
transform(3, function: { (x: Int) -> Int

})
```

Functions & Closures

```
func multiplyBy2(x: Int) -> Int { return x * 2 }
let doubler = multiplyBy2

func transform(x: Int, function: (Int) -> Int) -> Int {
  return function(x)
}
transform(3, function: doubler) // ~> 6
transform(3, function: { (x: Int) -> Int in

})
```

Functions & Closures

```swift
func multiplyBy2(x: Int) -> Int { return x * 2 }
let doubler = multiplyBy2

func transform(x: Int, function: (Int) -> Int) -> Int {
  return function(x)
}
transform(3, function: doubler) // ~> 6
transform(3, function: { (x: Int) -> Int in
  return x * 5
})
```

Functions & Closures

```
transform(3, function: { (x: Int) -> Int in
  return x * 5
})
```

Functions & Closures

```
transform(3, function: { (x) -> Int in
  return x * 5
})
```

Functions & Closures

```
transform(3, function: { x -> Int in
  return x * 5
})
```

Functions & Closures

```
transform(3, function: { x in
  return x * 5
})
```

Functions & Closures

```
transform(3, function: { x in
  x * 5
})
```

Functions & Closures

```
transform(3, function: { x in x * 5 })
```

Functions & Closures

```
transform(3, function: { $0 * 5 })
```

Functions & Closures

```
transform(3) { $0 * 5 }
```

Functions & Closures

```
languages = ["Java": 4, "JavaScript": 3, "Swift": 1,
"C++": 4, "Objective-C": 2]

languages.map({ (key, value) -> String in
    return key
 }).filter({ (language) -> Bool in
    return language.characters.count > 4
 }).forEach({ (language) -> () in
    print(language)
 })
```

Functions & Closures

```
languages = ["Java": 4, "JavaScript": 3, "Swift": 1,
"C++": 4, "Objective-C": 2]

languages.map({ (key, value) in
    return key
 }).filter({ (language) in
    return language.characters.count > 4
 }).forEach({ (language) in
    print(language)
 })
```

Functions & Closures

```
languages = ["Java": 4, "JavaScript": 3, "Swift": 1,
"C++": 4, "Objective-C": 2]

languages.map({ key, value in
    return key
 }).filter({ language in
    return language.characters.count > 4
 }).forEach({ language in
    print(language)
 })
```

Functions & Closures

```
languages = ["Java": 4, "JavaScript": 3, "Swift": 1,
"C++": 4, "Objective-C": 2]

languages.map({ key, value in
    key
 }).filter({ language in
    language.characters.count > 4
 }).forEach({ language in
    print(language)
 })
```

Functions & Closures

```
languages = ["Java": 4, "JavaScript": 3, "Swift": 1,
"C++": 4, "Objective-C": 2]

languages.map { key, value in
    key
 }.filter { language in
    language.characters.count > 4
 }.forEach { language in
    print(language)
 }
```

Functions & Closures

```
languages = ["Java": 4, "JavaScript": 3, "Swift": 1,
"C++": 4, "Objective-C": 2]

languages.map { $0.0 }
         .filter { $0.characters.count > 4 }
         .forEach { print($0) }
```

Functions & Closures

```
languages = ["Java": 4, "JavaScript": 3, "Swift": 1,
"C++": 4, "Objective-C": 2]

languages.map { $0.0 }
          .filter { $0.characters.count > 4 }
          .forEach { print($0) }
```

Enumerations

```swift
enum Direction {
  case North, East, South, West








}
```

Enumerations

```
enum Direction {
  case North, East, South, West

  func whereAreYouGoing() -> String {
    switch self {
      case .North: return "Going up"
      default: return "Somewhere"
    }
  }
}
```

Enumerations

```swift
enum Direction {
  case North, East, South, West

  func whereAreYouGoing() -> String {
    switch self {
      case .North: return "Going up"
      default: return "Somewhere"
    }
  }
}
var direction = Direction.South
```

Enumerations

```
enum Direction {
  case North, East, South, West

  func whereAreYouGoing() -> String {
    switch self {
      case .North: return "Going up"
      default: return "Somewhere"
    }
  }
}
var direction = Direction.South
direction = .North // Type is known ~> can be omitted
```

Enumerations

```
enum Currency : String {
  case Euro = "€"
  case Dollar = "$"
  case Pound = "£"
}
let currency = Currency.Pound
print(currency.rawValue) // ~> £
```

Enumerations

```
enum Currency : String {
  case Euro = "€"
  case Dollar = "$"
  case Pound = "£"
}
let currency = Currency.Pound
print(currency.rawValue) // ~> £

let another = Currency(rawValue: "€") // ~> Currency.Euro
let andAnother = Currency(rawValue: "¢") // ~> nil
```

Enumerations

```
enum Planet : Int {
  case Mercury = 1
  case Venus, Earth, Mars // 2, 3, 4
  case Jupiter = 9000
  case Saturn, Uranus, Neptune // 9001, 9002, 9003
  case Pluto = -100
}
```

Enumerations

```
enum Barcode {
  case UPCA(sys: Int, id: Int, check: Int)
  case QRCode(String)
}

var code = Barcode.UPCA(sys: 1, id: 23456_78999, check: 9)
code = .QRCode("Swift")
```

- Associated values
- With named parameters!

Enumerations

```
var code = Barcode.UPCA(sys: 1, id: 23456_78999, check: 9)

switch code {
  case let .UPCA(numberSystem, identifier, check):
    print("UPC-A: \(numberSystem), \(identifier), \(check)")

  case let .QRCode(productCode):
    print("QR code: \(productCode)")
}
```

Enumerations

```
enum TrainStatus {
  case OnTime
  case Delayed(Int)
}
```

Enumerations

```
switch status {
  case .OnTime:
    print("on time")
  case .Delayed(1):
    print("1 minute")
  case .Delayed(3...10):
    print("3 to 10 min")
  case .Delayed(let x) where x % 2 == 0:
    print("now you're just showing off")
  default:
    print(“changed my mind!”)
}
```

Enumerations

```
enum CurrentActivity {
  case Traveling(TrainStatus)
  ...
}
```

Enumerations

```
enum CurrentActivity {
  case Traveling(TrainStatus)
  ...
}

switch activity {
  case .Traveling(.OnTime):
    print("on time")
  case .Traveling(.Delayed(10...30)):
    print("delayed.")
```

Enumerations

```
enum MathExpression {
  case Num(Int)
  case Add(MathExpression, MathExpression)
  case Mult(MathExpression, MathExpression)
}
```

Enumerations

```
enum MathExpression {
  case Num(Int)
  indirect case Add(MathExpression, MathExpression)
  indirect case Mult(MathExpression, MathExpression)
}
```

Enumerations

```
indirect enum MathExpression {
  case Num(Int)
  case Add(MathExpression, MathExpression)
  case Mult(MathExpression, MathExpression)
}
```

Enumerations

```
func eval(expression: MathExpression) -> Int {
  switch expression {
  case .Num(let value):
    return value
  case .Add(let lhs, let rhs):
    return eval(lhs) + eval(rhs)
  case .Mult(let lhs, let rhs):
    return eval(lhs) * eval(rhs)
  }
}
```

Enumerations

```
let expression = MathExpression.Add(
  .Mult(.Num(3), .Num(4)),
  .Num(7)
)

eval(expression) // ~> 19
```

Classes

```
class Vehicle {
  let model: String
  var color: String = "Black"
}
```

Classes

```
class Vehicle {
  let model: String
  var color: String = "Black"

  init(model: String, color: String) {
    self.model = model
    self.color = color
  }
}
```

Classes

```
class Vehicle {
  let model: String
  var color: String = "Black"

  init(model: String, color: String) {
    self.model = model
    self.color = color
  }
  deinit {…}
}
let someVehicle = Vehicle(model: "VW", color: "Red")
```

Classes

```
class Vehicle {
  let model: String
  var color: String = "Black”

  var description: String {


  }
}
```

Classes

```
class Vehicle {
  let model: String
  var color: String = "Black”

  var description: String {
    get {…}
    set {…}
  }
}
```

Classes

```
class Vehicle {
  let model: String
  var color: String = "Black”

  var description: String {
    get {
      return "This is a \(color) \(model)"
    }
    set {
    }
  }
}
```

Classes

```
class Vehicle {
  let model: String
  var color: String = "Black”

  var description: String {
    return "This is a \(color) \(model)"
  }
}
```

Classes

```
class Car: Vehicle {
  var speed: Double = 0.0
  init(model: String) {
    super.init(model: model, color: "Blue")
  }
}
let someCar = Car(model: "BMW")
```

Classes

```
class Car: Vehicle {
  var speed: Double = 0.0
  init(model: String) {
    super.init(model: model, color: "Blue")
  }

  override var description: String {
    return "A blue car"
  }
}
```

Classes

```
class Car: Vehicle {
  var speed: Double {
    willSet {
       // newValue
    }
    didSet {
       // oldValue
    }
  }
}
```

Classes

```
class Car: Vehicle {
  var speed: Double {
    willSet {
      if newValue > 300 {
        print("Are you nuts!?")
      }
    }
    didSet {
      print("Was \(oldValue), now \(speed)")
    }
  }
}
```

- Property observing

Classes

```
class GameBoard {
  var array: [Int] = Array(count: 4, repeatedValue: 0)

  subscript(x:Int, y:Int) -> Int {



  }
}
```

Classes

```
class GameBoard {
  var array: [Int] = Array(count: 4, repeatedValue: 0)

  subscript(x:Int, y:Int) -> Int {
    get { return array[(x * y) + y] }
    set { array[(x * y) + y] = newValue }
  }
}
```

Classes

```
class GameBoard {
  var array: [Int] = Array(count: 4, repeatedValue: 0)

  subscript(x:Int, y:Int) -> Int {
    get { return array[(x * y) + y] }
    set { array[(x * y) + y] = newValue }
  }
}
var board = GameBoard()
board[1, 2] = 3
```

- Custom subscripts

Classes

```
class Speed {
  var metersPerSecond: Double = 0
  init(metersPerSecond: Double) {
    self.metersPerSecond = metersPerSecond
  }
}
```

Operators

```
func + (left: Speed, right: Speed) -> Speed {
  return Speed(metersPerSecond:
         left.metersPerSecond + right.metersPerSecond)
}
```

Operators

```
func + (left: Speed, right: Speed) -> Speed {
  return Speed(metersPerSecond:
         left.metersPerSecond + right.metersPerSecond)
}

func += (inout left: Speed, right: Speed) {
  left = left + right
}
```

Operators

```
func + (left: Speed, right: Speed) -> Speed {
  return Speed(metersPerSecond:
         left.metersPerSecond + right.metersPerSecond)
}

func += (inout left: Speed, right: Speed) {
  left = left + right
}

var speed = Speed(metersPerSecond: 2)
speed += Speed(metersPerSecond: 5)
let rocket = speed + Speed(metersPerSecond: 9000)
```

Operators

```
prefix operator <<+ {}

prefix func <<+ (inout speed: Speed) {
  speed.metersPerSecond = 299_792_458
}
```

Operators

```
prefix operator <<+ {}

prefix func <<+ (inout speed: Speed) {
  speed.metersPerSecond = 299_792_458
}

<<+speed
```

- Custom operators

Classes

```
class Vehicle {
  let model: String
  var color: String = "Black"
  var speed: Speed = Speed(metersPerSecond: 0.0)
}
```

Structs

```
var speed = Speed(metersPerSecond: 10)
let myCar = Car(model: "BMW", color: "Black")
myCar.speed = speed
```

Structs

```
var speed = Speed(metersPerSecond: 10)
let myCar = Car(model: "BMW", color: "Black")
myCar.speed = speed

let myJet = Jet(model: "G550", color: "Green")
speed.metersPerSecond = 250
myJet.speed = speed
```

Structs

```
var speed = Speed(metersPerSecond: 10)
let myCar = Car(model: "BMW", color: "Black")
myCar.speed = speed

let myJet = Jet(model: "G550", color: "Green")
speed.metersPerSecond = 250
myJet.speed = speed

print(myCar.description) // Black BMW going at 250.0 m/s
print(myJet.description) // Green G550 going at 250.0 m/s
```

Structs

```
class Speed {
  var metersPerSecond: Double = 0

  init(metersPerSecond: Double) {
    self.metersPerSecond = metersPerSecond
  }
}
```

Structs

```
struct Speed {
  var metersPerSecond: Double = 0

  init(metersPerSecond: Double) {
    self.metersPerSecond = metersPerSecond
  }
}
```

- structs are value-types

Protocols

```
protocol SomeProtocol {







}
```

Protocols

```
protocol SomeProtocol {
  static func typeMethod() -> Bool







}
```

Protocols

```
protocol SomeProtocol {
  static func typeMethod() -> Bool
  static var typeProperty: Float { get set }





}
```

Protocols

```
protocol SomeProtocol {
  static func typeMethod() -> Bool
  static var typeProperty: Float { get set }

  func instanceMethod(x: Int) -> String



}
```

Protocols

```
protocol SomeProtocol {
  static func typeMethod() -> Bool
  static var typeProperty: Float { get set }

  func instanceMethod(x: Int) -> String
  var instanceProperty: Double { get }


}
```

Protocols

```
protocol SomeProtocol {
  static func typeMethod() -> Bool
  static var typeProperty: Float { get set }

  func instanceMethod(x: Int) -> String
  var instanceProperty: Double { get }

  init(name: String)
}
```

Protocols

```
protocol Comparable {
  func compare(lhs: Self, rhs: Self) -> Int
}
```

- Self requirement

Protocols

```
protocol Comparable {
  func compare(lhs: Self, rhs: Self) -> Int
}
struct Speed : Comparable {








}
```

Protocols

```
protocol Comparable {
  func compare(lhs: Self, rhs: Self) -> Int
}
struct Speed : Comparable {
  func compare(lhs: Speed, rhs: Speed) -> Int {





  }
}
```

Protocols

```
protocol Comparable {
  func compare(lhs: Self, rhs: Self) -> Int
}
struct Speed : Comparable {
  func compare(lhs: Speed, rhs: Speed) -> Int {
    if lhs.metersPerSecond < rhs.metersPerSecond {
      return -1
    }


  }
}
```

Protocols

```
protocol Comparable {
  func compare(lhs: Self, rhs: Self) -> Int
}
struct Speed : Comparable {
  func compare(lhs: Speed, rhs: Speed) -> Int {
    if lhs.metersPerSecond < rhs.metersPerSecond {
      return -1
    }
    else if … {…}
    else {…}
  }
}
```

- Type information is conserved!

Extensions

```
extension Double {


}
```

Extensions

```
extension Double {
  var meters: Double { return self }
  var kilometers: Double { return self * 1_000.0 }
}
```

Extensions

```
extension Double {
  var meters: Double { return self }
  var kilometers: Double { return self * 1_000.0 }
}

1.7.kilometers // ~> 1700
```

- Extend existing types

Extensions

```
extension Int {
  func times(closure: ()->Void) {
    for _ in 1...self {
      closure()
    }
  }
}
```

Extensions

```
extension Int {
  func times(closure: ()->Void) {
    for _ in 1...self {
      closure()
    }
  }
}

2.times {
  print("Hi")
}
```

- Define new methods

Generics

```
func sort(items: [Int]) -> [Int] {
  // implementation
}
```

Generics

```
func sort(items: [String]) -> [String] {
  // implementation
}
```

Generics

```swift
func sort<T>(items: [T]) -> [T] {
  // implementation
}
```

Generics

```
func sort<T: Comparable>(items: [T]) -> [T] {
  // implementation
}
```

Generics

```
func sort<T where T:Comparable, T:Hashable>(items:[T])->[T]
{
  // implementation
}
```

- Multiple criteria via “where” clause

Generics

```
protocol Container {
  typealias ItemType



}
```

Generics

```
protocol Container {
  typealias ItemType

  func append(item: ItemType)
  subscript(i: Int) -> ItemType { get }
}
```

Generics

```
struct StringContainer : Container {








}
```

Generics

```
struct StringContainer : Container {
  typealias ItemType = String







}
```

Generics

```
struct StringContainer : Container {
  typealias ItemType = String

  func append(item: String) {
    // implementation
  }

  subscript(i: Int) -> String {
    // implementation
  }
}
```

Generics

```swift
struct StringContainer : Container {

  func append(item: String) {
    // implementation
  }

  subscript(i: Int) -> String {
    // implementation
  }
}
```

Generics

```
func allItemsMatch
  (firstContainer: C1, _ secondContainer: C2) -> Bool
{
  // implementation
}
```

Generics

```
func allItemsMatch
  <



  >
  (firstContainer: C1, _ secondContainer: C2) -> Bool
{
  // implementation
}
```

Generics

```
func allItemsMatch
  <
   C1: Container, C2: Container


  >
  (firstContainer: C1, _ secondContainer: C2) -> Bool
{
  // implementation
}
```

Generics

```
func allItemsMatch
  <
   C1: Container, C2: Container
   where

  >
  (firstContainer: C1, _ secondContainer: C2) -> Bool
{
  // implementation
}
```

Generics

```
func allItemsMatch
  <
   C1: Container, C2: Container
   where C1.ItemType == C2.ItemType

  >
  (firstContainer: C1, _ secondContainer: C2) -> Bool
{
  // implementation
}
```

Generics

```
func allItemsMatch
  <
   C1: Container, C2: Container
   where C1.ItemType == C2.ItemType,

  >
  (firstContainer: C1, _ secondContainer: C2) -> Bool
{
  // implementation
}
```

Generics

```
func allItemsMatch
  <
   C1: Container, C2: Container
   where C1.ItemType == C2.ItemType,
         C1.ItemType: Equatable
  >
  (firstContainer: C1, _ secondContainer: C2) -> Bool
{
  // implementation
}
```

Swift

- Really fun!
- Open Source (later in 2015)
- Growing community

Thanks for listening!

Feedback is welcome and appreciated!

Q&A