A WHIRLWIND TOUR OF

SWIFT

MIA ALEXIOU

INTRODUCTION

INTRODUCTION

Mia Alexiou
Subsymbolic Software Ltd

www.subsymbolicsoftware.com
mia.alexiou@subsymbolicsoftware.com

Tweet to @subsymbolicnet

SWIFT

THE LANGUAGE

HELLO WORLD

- Comments
- Import Statment
- Optional semi-colons
- Constants versus variables

```
/*
 *  A basic Hello World swift application
 */
import Foundation

// Constants and variables
let hello = "Hello"
var name = "Tech Meetup"

// Printing and strings
println("\(hello) \(name)")
println("Today's date is: " + NSDate().description);
```

IMMUTABILITY

```
// fine
var title = "Tech Meetup"
title = "Tech Meetup Edinburgh"

// not fine
let greeting = "Hello"
greeting = "Hi"
```

STRONG TYPING

- Strongly typed
- Types may be inferred

```
// Inferred Type
let hello = "Hello"
var name = "Tech Meetup"

// Specified Type
let colour: String = "green"
var age: Int
age = 35
```

FUNCTIONS: SIMPLE CASE

```
// Create the function
func printHello() {
    print("Hello")
}

// Run it
printHello()
```

FUNCTIONS:
PARAMS AND RETURN TYPES

```
func dayAfterDate(date: NSDate) -> NSDate {
    let calendar = NSCalendar.currentCalendar()
    return calendar.dateByAddingUnit(.CalendarUnitDay,
        value: 1,
        toDate: date,
        options: nil)!
}

let tomorrow = dayAfterDate(NSDate())
```

FUNCTIONS: VARIABLE PARAMS

```
func sumOf(numbers: Int...) -> Int {
    var sum = 0
    for number in numbers {
        sum += number
    }
    return sum
}
```

```
sumOf()
sumOf(42, 597, 12)
```

https://developer.apple.com/library/ios/documentation/Swift/Conceptual/Swift_Programming_Language/Functions.html#//apple_ref/doc/uid/TP40014097-CH10-ID158

FUNCTIONS: PARAM NAMES

```
// Param names not required
func calculateArea(width: Double, height: Double) -> Double {
    return width * height
}
let area = calculateArea(10, 15)
```

```
// Param names required
func calculateArea(#width: Double, #height: Double) -> Double {
    return width * height
}
let area = calculateArea(width: 10, height: 15)
```

TUPLES

- Quickly group multiple variables

```
let (x,y) = (10, 15)
println(x)                          // prints 10
println(y)                          // prints 15

let position = (10, 15)
println(position)                   // prints (10, 15)
println(position.0)                 // prints 10
```

- Could instead define a struct, class, use existing CGPoint

TUPLES

- Particularly useful when you want multiple return types

```
// Blocking operation. Do not use on main thread
func downloadImageFromUrl(imageUrl: NSURL) -> (image: UIImage?, error: NSError?){

    let request = NSURLRequest(URL: imageUrl)
    var error: NSError?
    var image: UIImage?

    let data = NSURLConnection.sendSynchronousRequest(request,
        returningResponse: nil,
        error: &error)

    if let responseData = data {
        image = UIImage(data: responseData)
    }

    return (image, error)
}
```

- Beware of overuse

OPTIONALS

- Swift clearly distinguishes between variables which are required and those which are optional
- An optional MAY or MAY NOT contain another variable
- Clean contract for variables and return types
- Removes uncertainty around whether a variable could be nil / null

OPTIONALS

- Let's revisit our previous function…

```
// Blocking operation. Do not use on main thread
func downloadImageFromUrl(imageUrl: NSURL) -> (image: UIImage?, error: NSError?)
{

    let request = NSURLRequest(URL:imageUrl)
    var error: NSError?
    var image: UIImage?

    let data = NSURLConnection.sendSynchronousRequest(request,
        returningResponse: nil,
        error: &error)

    if let responseData = data {
        image = UIImage(data: responseData)
    }

    return (image, error)
}
```

OPTIONALS

- Invoking the previous function

```
var url = NSURL(string: "http://techmeetup.co.uk/static/img/
techmeetup_logo.png")!

let download = downloadImageFromUrl(url)
if let image = download.image {
    //do something with image
}

if let error = download.error {
    NSLog("Could not download image: \(error.description)")
}
```

OPTIONAL CHAINING & DOWNCASTING

```
if let postcode = clientForOrder("ABC834239")?.address?.postcode {
    marketingRegions.registerPostcode(postcode)
}
```

```
if let address = jsonRecord["address"] as? NSDictionary,
       postcode = address["zipcode"] as? String {
    marketingRegions.registerPostcode(postcode)
}
```

CLOSURES

- Self contained blocks of functionality that can be passed around
- Capture and store references to any constants and variables from the context in which they are defined.

CLOSURES

```
let names = ["Chris", "Alex", "Ewa", "Barry", "Daniella"]
func backwards(s1: String, s2: String) -> Bool {
    return s1 > s2
}
var reversed = sorted(names, backwards)
```

```
let names = ["Chris", "Alex", "Ewa", "Barry", "Daniella"]
var reversed = sorted(names, {(s1: String, s2: String) -> Bool in
    return s1 > s2
}
```

https://developer.apple.com/library/ios/documentation/Swift/Conceptual/Swift_Programming_Language/Closures.html#//apple_ref/doc/uid/TP40014097-CH11-ID94

CLOSURES

```
let names = ["Chris", "Alex", "Ewa", "Barry", "Daniella"]
var reversed = sorted(names, { s1, s2 in return s1 > s2 } )
```

```
let names = ["Chris", "Alex", "Ewa", "Barry", "Daniella"]
var reversed = sorted(names, { s1, s2 in s1 > s2 } )
```

```
let names = ["Chris", "Alex", "Ewa", "Barry", "Daniella"]
var reversed = sorted(names, { $0 > $1 } )
```

https://developer.apple.com/library/ios/documentation/Swift/Conceptual/Swift_Programming_Language/Closures.html#//apple_ref/doc/uid/TP40014097-CH11-ID94

STRUCTS

- Define properties to store values
- Define methods to provide functionality
- Define subscripts to provide access to their values using subscript syntax
- Define initializers to set up their initial state
- Be extended to expand their functionality beyond a default implementation
- Conform to protocols to provide standard functionality of a certain kind

From https://developer.apple.com/library/ios/documentation/Swift/Conceptual/Swift_Programming_Language/ClassesAndStructures.html#//apple_ref/doc/uid/TP40014097-CH13-ID82

STRUCTS

```
struct Point {
    var x: Double
    var y: Double
}

// Memberwise initialiser
let point = Point(x: 5.0, y: 10.0)
```

STRUCTS

```
struct Point {
    var x = 0.0
    var y = 0.0
}

// Default initialiser now available too
let centre = Point()

// Memberwise initialiser
let anotherPoint  = Point(x: 5.0, y: 10.0)
```

METHODS

```
struct Point {
    var x = 0.0
    var y = 0.0

    func distanceFromOrigin() -> Double {
        return sqrt( pow(x,2.0) + pow(y,2.0) )
    }
}

let point  = Point(x: 5.0, y: 10.0)
println(point.distanceFromOrigin())
```

COMPUTED PROPERTIES

```
struct Point {
    var x = 0.0
    var y = 0.0

    var distanceFromOrigin: Double {
        return sqrt( pow(x,2.0) + pow(y,2.0) )
    }
}

let point  = Point(x: 5.0, y: 10.0)
println(point.distanceFromOrigin)
```

CLASSES

- No header / implementation separation
- Convention to put a class in its own file but not necessary

CLASSES: DIFFERENT TO STRUCTS

- Inheritance enables one class to inherit the characteristics of another.
- Type casting enables you to check and interpret the type of a class instance at runtime.
- Deinitializers enable an instance of a class to free up any resources it has assigned.
- Reference counting allows more than one reference to a class instance.

From https://developer.apple.com/library/ios/documentation/Swift/Conceptual/Swift_Programming_Language/ClassesAndStructures.html#//apple_ref/doc/uid/TP40014097-CH13-ID82

CLASSES: SIMPLE CASE

```
class Person {
    var name: String?
}

var person = Person()
person.name = "Jon"
```

CLASS CONTRACT

```
class Customer {

    let givenName: String
    let surname: String
    var dob: NSDate?
    private(set) var loyaltyScore = 0
    private var creditStatus: CreditStatus?

  // what kind of init do we need?

}
```

CLASS CONTRACT

```
class Customer {
    let givenName: String
    let surname: String
    var dob: NSDate?
    private(set) var loyaltyScore = 0

    init(givenName: String, surname: String) {
        self.givenName = givenName
        self.surname = surname
    }

    // and so on...
}
```

CLASS CONTRACT

```
var customer = Customer(givenName: "Jon",
                        surname: "Snow")

println(customer.loyaltyScore)

customer.loyaltyScore = 100 // not allowed

customer.dob = NSDate()
```

PROPERTY OBSERVERS

```
class Customer {
    let givenName: String
    let surname: String

    var dob: NSDate? {
        didSet {
            updateAge()
        }
    }
    // do more stuff
}
```

LAZY STORED PROPERTIES

```
class DataManager {
    lazy var importer = DataImporter()
    var data = [String]()
    // and so on...
}
```

https://developer.apple.com/library/prerelease/ios/documentation/Swift/Conceptual/Swift_Programming_Language/Properties.html#//apple_ref/doc/uid/TP40014097-CH14-ID254

SUBCLASS

- No automatic base class
- Single inheritance
- Final keyword on class to block subclassing or method overriding

```
class Trike: Vehicle {
    override func numberOfWheels() -> Int {
        return 3
    }
}
```

PROTOCOLS

```
protocol Creature {
    var name: String { get }
    var dailyEnergyConsumption: Int { get set}
}

class Dog: Creature {
    let name: String
    var dailyEnergyConsumption: Int = 1000

    init(name: String) {
        self.name = name
    }
}
```

```
let bango = Dog(name: "banjo")
bango.dailyEnergyConsumption = 975
```

EXTENSIONS

- Add methods or computed properties to a type
- Allows programmer to add behaviour to an object that they did not instantiate
- Can mutate self
- Conform to a new protocol
- Can NOT override existing methods or add stored properties

EXTENSIONS

```
extension NSTimeInterval {

    var hh: Int {
        let secInHour = 60.0 * 60.0
        return (Int)(self / secInHour)
    }
}
```

```
let hoursWorked = getTimeWorked().hh
```

THE ATTIC

THE ATTIC

- Topics for next time:
  - Collections, enums, control flow, generics
  - Memory management
  - Testing, Testing, Testing

FINISHING THOUGHTS

PAIN POINTS

- Pain points of the past are fixed
- A few outstanding
  - no code coverage reporting
  - compiler seg faults
  - core data / test namespacing

FINISHING THOUGHTS

- **Existing devs:** Switch now
- **New devs:** Try it now

WHERE TO NEXT?

- Apple: WWDC Videos
  https://developer.apple.com/videos/wwdc

- Apple: Swift Resources
  https://developer.apple.com/swift/resources/

- Apple: Swift Programming Guide
  https://developer.apple.com/library/ios/documentation/Swift/Conceptual/Swift_Programming_Language/

- Stanford: Developing iOS 8 Apps with Swift
  https://itunes.apple.com/gb/course/developing-ios-8-apps-swift/id961180099

- Ray Wenderlich Tutorials
  http://www.raywenderlich.com/

QUESTIONS COMMENTS?

THANK YOU!

STAY IN TOUCH

Mia Alexiou
Subsymbolic Software Ltd

www.subsymbolicsoftware.com
mia.alexiou@subsymbolicsoftware.com

Tweet to @subsymbolicnet