USING SWIFT OUTSIDE OF XCODE

ADAM MIKA

DEVELOPER @ ASTROHQ

GOAL OF THIS TALK

I AM NOT TRYING TO CONVINCE YOU XCODE SUCKS

WHAT WE'LL COVER

1. REPL
2. Scripting
3. Swift Package Manager

WHAT WE'RE NOT GOING TO COVER

1. Getting Swift environment setup
2. AppCode

EVERYTHING IS ON THE COMMAND LINE

REPL

READ EVAL PRINT LOOP

WORKS LIKE AN INTERPRETER

IMMEDIATELY COMPILE AND EXECUTE CODE

FROM TERMINAL

```
$ swift
Welcome to Apple Swift version 3.1 (swiftlang-802.0.48 clang-802.0.48).
Type :help for assistance.
  1> |
```

READ - EVAL - PRINT - LOOP

```
1> Int("100")
$R0: Int? = 100

2> let name = "SwiftMN"
name: String = "SwiftMN"

3> print("Hello, \(name)!")
Hello, SwiftMN!
```

AUTO INDENTATION

```
5> extension Int {
6.     |
```

FOLLOWS CONTINUED NESTING

```
1> extension Int {
2.      func timesTwo() -> Int {
3.          |
```

```
1> extension Int {
2.      func timesTwo() -> Int {
3.          return self * 2
4.      }
5. }
6> 4.timesTwo()
$R0: Int = 8
```

REPL IS DIFFERENT THAN COMPILED SWIFT FILES

REDEFINING IDENTIFIERS

THIS IS INVALID COMPILED SWIFT

```
var x = "Hello"
var x = 42
```

```
$ > swiftc -
var x = "Hello"
var x = 42
^D
<stdin>:2:5: error: invalid redeclaration of 'x'
var x = 42
    ^
<stdin>:1:5: note: 'x' previously declared here
var x = "Hello"
    ^
```

THIS IS COMPLETELY VALID IN THE REPL

```
1> var x = "Hello"
x: String = "Hello"

2> var x = 42
x: Int = 42

3> x
$R0: Int = 42
```

NEWER DEFINITION REPLACES THE EXISTING

REDEFINING FUNCTIONS

```
1> func sayHello {
2.   print("Hello")
3. }
4> sayHello()
Hello
```

```
 8> func sayHello() {
 9.  print("Bonjour")
10. }
11> sayHello()
Bonjour
```

REDEFINING IDENTIFIERS INSIDE FUNCTIONS

CAPTURING DEFINITIONS

```
22> var message = "Hello, World!"
message: String = "Hello, World!"

23> func printMessage() {
24.     print(message)
25. }
26> printMessage()
Hello, World!

27> message = "Goodbye"
28> printMessage()
Goodbye

29> var message = "New Message"
30> printMessage()
Goodbye

31> print(message)
New Message
```

REPL HAS FULL LLDB SUPPORT

EXAMPLE: ADDING BREAKPOINTS

```
1> func add(_ value1: Int, _ value2: Int) -> Int {
2.      return value1 + value2
3. }
4> :b 2

Breakpoint 1: where = $__lldb_expr2`__lldb_expr_1.add
(Swift.Int, Swift.Int) -> Swift.Int + 12 at repl.swift:2,
address = 0x000000001000c501c
```

```
5> 4> add(5, 10)
Execution stopped at breakpoint.  Enter LLDB commands to investigate (type help for assistance.)
Process 29818 stopped
* thread #1, queue = 'com.apple.main-thread', stop reason = breakpoint 1.1
    frame #0: 0x00000001000c501c $__lldb_expr2`add(value1=5, value2=10) -> Int at repl.swift:2
   1    func add(_ value1: Int, _ value2: Int) -> Int {
-> 2        return value1 + value2
   3    }
   4    add(5, 10)
(lldb)
```

EXECUTE LLDB COMMANDS

```
(lldb) po value1
5

(lldb) po value2
10

(lldb) po value1 * value2
50

(lldb) continue
Process 29818 resuming
```

LOADS OF OTHER THINGS YOU CAN DO IN REPL

```
1> :help


The REPL (Read-Eval-Print-Loop) acts like an interpreter.  Valid statements, expressions,
and declarations are immediately compiled and executed.

The complete set of LLDB debugging commands are also available as described below.
Commands must be prefixed with a colon at the REPL prompt (:quit for example.)
Typing just a colon followed by return will switch to the LLDB prompt.

Debugger commands:

.
.
.
```

MOST IMPORTANT

```
exit        -- Quit the LLDB debugger.
q           -- Quit the LLDB debugger.
```

```
$ > :exit
```

SOMETHING A LITTLE MORE PERMANENT

SCRIPTING

SCRIPTING HAS COME A LONG WAY

INITIAL IMPLEMENTATION

```
/Applications/Xcode6-Beta1.app/Contents/
Developer/usr/bin/xcrun swift -i
script.swift
```

```
xcrun swift -i script.swift
```

```
swift -i script.swift
```

```
swift script.swift
```

SIMPLE SCRIPT

HELLO.SWIFT

```
#!/usr/bin/env swift
print("Hello SwiftMN!")
```

HELLO.SWIFT

```
$> swift hello.swift
Hello SwiftMN
```

ABOUT THAT FIRST LINE

HASHBANG

```
#!/usr/bin/swift
#!/usr/bin/env swift
```

ESSENTIALLY WORKS LIKE PLAYGROUND

STRUCT.SWIFT

```
#!/usr/bin/env swift

struct Robot {
  func speak() {
    print("Beep-Boop")
  }
}

let robot = Robot()
robot.speak()
```

STRUCT.SWIFT

```
$ > swift struct.swift
Beep-Boop
```

ARGUMENTS

ARGUMENTS.SWIFT

```
#!/usr/bin/env swift
dump(CommandLine.arguments)
```

ARGUMENTS.SWIFT

```
$ > swift arguments.swift one two three
▿ 4 elements
  - "arguments.swift"
  - "one"
  - "two"
  - "three"
```

IMPORTANT NOTES

- All arguments are Strings
- First argument is script itself

PARSING ARGUMENTS

```
let args = CommandLine.arguments
guard args.count == 2 else {
  print("Requires one argument")
}

guard let times = Int(args[1]) else {
  print("Requires argument as Int")
}

for i in 0..<times {
  print("Hello SwiftMN")
}
```

'guard' body may not fall through, consider using 'return' or 'break' to exit the scope

USING GUARD IN SCRIPTS

- `return` doesn't work because we're not in a function
- `break` doesn't work because we're not in a loop
- We need something else

NEVER[2]

Tells compiler that the function *never* returns

[2] Swift Evolution 102

FAIL(_)

```swift
import Foundation

func fail(_ msg: String) -> Never {
    print("Error: \(msg)")
    exit(1)
}
```

MULTIPLE-HELLO.SWIFT

```
let args = CommandLine.arguments
guard args.count == 2 else {
  fail("Requires one argument")
}

guard let times = Int(args[1]) else {
  fail("Requires argument as Int")
}

for i in 0..<times {
  print("Hello SwiftMN")
}
```

MULTIPLE-HELLO.SWIFT

```
$ > swift multiple-hello.swift
Error: Requires one argument

$ > swift multiple-hello.swift foo
Error: Requires argument as Int

$ > swift multiple-hello.swift 5.5
Error: Requires argument as Int

$ > swift multiple-hello.swift 5
Hello SwiftMN
Hello SwiftMN
Hello SwiftMN
Hello SwiftMN
Hello SwiftMN
```

"SHELLING OUT"

SHELL.SWIFT

```swift
import Foundation

func shell(_ args: String...) -> String {
    let task = Process()
    task.launchPath = "/usr/bin/env"
    task.arguments = args

    let pipe = Pipe()
    task.standardOutput = pipe
    task.launch()

    let data = pipe.fileHandleForReading.readDataToEndOfFile()
    let output = String(data: data, encoding: .utf8)!

    return output
}

let output = shell("ls", "-l")
print("output: \(output)")
```

SHELL.SWIFT

```
$ > swift shell.swift
output: total 0
drwx------@  4 mika  staff   136 Apr 11 21:53 Applications
drwxr-xr-x   7 mika  staff   238 Apr  6 22:03 Code
drwxrwxr-x@  4 mika  staff   136 Apr 10 15:03 Creative Cloud Files
drwx------+ 23 mika  staff   782 Apr 11 09:47 Desktop
drwx------+ 15 mika  staff   510 Apr 13 09:24 Documents
drwx------+  4 mika  staff   136 Apr 13 20:24 Downloads
drwx------@ 69 mika  staff  2346 Feb  6 07:46 Library
drwx------+  6 mika  staff   204 Feb 28 17:18 Movies
drwx------+  6 mika  staff   204 Jan 13 11:15 Music
drwx------+ 19 mika  staff   646 Mar 22 18:54 Pictures
drwxr-xr-x+  5 mika  staff   170 Dec 28 13:16 Public
```

WHY USE SWIFT FOR THIS?

MODULARIZING YOUR CODE

SPM
SWIFT PACKAGE MANAGER

TOOL FOR MANAGING THE DISTRIBUTION OF SWIFT CODE

BUILD COMPLEX SWIFT PACKAGES

SIMILAR TO COCOAPODS AND CARTHAGE

NO INTEGRATED IOS / MACOS / XCODE SUPPORT (IT'S COMING)

BUILDING A SWIFT MODULE

```
$ mkdir HelloKit
$ cd HelloKit
$ swift package init
```

```
Creating library package: HelloKit
Creating Package.swift
Creating .gitignore
Creating Sources/
Creating Sources/HelloKit.swift
Creating Tests/
Creating Tests/LinuxMain.swift
Creating Tests/HelloKitTests/
Creating Tests/HelloKitTests/HelloKitTests.swift
```

PACKAGE.SWIFT

```
// swift-tools-version:3.1

import PackageDescription

let package = Package(
    name: "HelloKit"
)
```

SOURCES/HELLO.SWIFT

```
struct Hello {
  var text = "Hello, World!"
}
```

```
$ swift build
Compile Swift Module 'HelloKit' (1 sources)
```

```
$ swift test
Compile Swift Module 'HelloKitTests' (1 sources)
Linking ./.build/debug/HelloKitPackageTests.xctest/Contents/MacOS/HelloPackageTests
Test Suite 'All tests' started at 2017-04-15 12:18:42.395
Test Suite 'HelloKitPackageTests.xctest' started at 2017-04-15 12:18:42.395
Test Suite 'HelloKitTests' started at 2017-04-15 12:18:42.395
Test Case '-[HelloKitTests.HelloKitTests testExample]' started.
Test Case '-[HelloKitTests.HelloKitTests testExample]' passed (0.071 seconds).
Test Suite 'HelloKitTests' passed at 2017-04-15 12:18:42.467.
     Executed 1 test, with 0 failures (0 unexpected) in 0.071 (0.072) seconds
Test Suite 'HelloKitPackageTests.xctest' passed at 2017-04-15 12:18:42.467.
     Executed 1 test, with 0 failures (0 unexpected) in 0.071 (0.072) seconds
Test Suite 'All tests' passed at 2017-04-15 12:18:42.467.
     Executed 1 test, with 0 failures (0 unexpected) in 0.071 (0.072) seconds
```

WHAT JUST HAPPENED?

WE JUST BUILT A SWIFT MODULE!

BUILDING AN EXECUTABLE PACKAGE

```
$ mkdir Hello
$ cd Hello
$ swift package init --type executable
```

```
Creating executable package: Hello
Creating Package.swift
Creating .gitignore
Creating Sources/
Creating Sources/main.swift
Creating Tests/
```

SOURCES/MAIN.SWIFT

```
print("Hello, world!")
```

```
$ swift build
Compile Swift Module 'Hello' (1 sources)
Linking ./.build/debug/Hello
```

```
./.build/debug/Hello
Hello, world!
```

LINKING THE TWO TOGETHER

HELLOKIT/SOURCES/GREETER.SWIFT

```swift
public struct Greeter {
  let name: String

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

  public func greeting() -> String {
    return "Hello, \(name)!"
  }
}
```

COMMIT THE CHANGE, TAG A RELEASE

Follow semantic versioning pattern[4]

```
$ > git add .
$ > git commit -m "Add Greeter class"
$ > git tag 1.0.0
```

[4] http://semver.org

HELLO/PACKAGE.SWIFT

```
// swift-tools-version:3.1

import PackageDescription

let package = Package(
    name: "Hello",
    targets: [],
    dependencies: [
      .Package(url: "../HelloKit", majorVersion: 1)
    ]
)
```

```
$ > swift build

Fetching /Users/mika/Code/SwiftMN/CommandLineSwift/HelloKit
Cloning /Users/mika/Code/SwiftMN/CommandLineSwift/HelloKit
Resolving /Users/mika/Code/SwiftMN/CommandLineSwift/HelloKit at 1.0.0
Compile Swift Module 'HelloKit' (1 sources)
Compile Swift Module 'Hello' (1 sources)
Linking ./.build/debug/Hello
```

HELLO/SOURCES/MAIN.SWIFT

```
import HelloKit

let g = Greeter(name: "SwiftMN")
print(g.greeting())
```

```
$ > swift build

Fetching /Users/mika/Code/SwiftMN/CommandLineSwift/HelloKit
Cloning /Users/mika/Code/SwiftMN/CommandLineSwift/HelloKit
Resolving /Users/mika/Code/SwiftMN/CommandLineSwift/HelloKit at 1.0.0
Compile Swift Module 'HelloKit' (1 sources)
Compile Swift Module 'Hello' (1 sources)
Linking ./.build/debug/Hello
```

```
$ > ./.build/debug/Hello
Hello, SwiftMN!
```

DEBUGGING

```
$> lldb .build/debug/Hello

(lldb) target create ".build/debug/Hello"
Current executable set to '.build/debug/Hello' (x86_64).
(lldb) |
```

```
(lldb) breakpoint set -f Greeter.swift -l 9
Breakpoint 1: where = Hello`HelloKit.Greeter.greeting
  () -> Swift.String + 20 at Greeter.swift:9,
  address = 0x0000000100001e04
```

```
(lldb) run
Process 11513 launched: '/Users/mika/Code/SwiftMN/CommandLineSwift/Hello/.build/debug/Hello' (x86_64)
Process 11513 stopped
* thread #1, queue = 'com.apple.main-thread', stop reason = breakpoint 1.1
    frame #0: 0x0000000100001e04 Hello`Greeter.greeting(self=(name = "SwiftMN")) -> String at Greeter.swift:9
   6       }
   7
   8       public func greeting() -> String {
-> 9         return "Hello, \(name)!"
   10      }
   11  }
```

```
6           }
7
8           public func greeting() -> String {
-> 9           return "Hello, \(name)!"
10          }
11      }

(lldb) po name
"SwiftMN"
```

```
(lldb) continue
Process 11513 resuming
Hello, SwiftMN!
Process 11513 exited with status = 0 (0x00000000)
```

SPM TESTING

- Only supports XCTest
- Very specific naming conventions
- Linux has it's own testing conventions

SPM DIRECTORIES

https://packagecatalog.com
https://swiftmodules.com

WRAPPING UP

Q/A

SWIFTMN SLACK CHANNEL