Developer Tools #WWDC15

What's New in LLDB

Debug your way to fame and glory

Session 402

Kate Stone Software Behavioralist
Sean Callanan Master of Expressions
Enrico Granata Data Wizard

The Year's Highlights

Changes big and small

Since WWDC 2014

Since WWDC 2014

Shipped Swift debugger

Since WWDC 2014

Shipped Swift debugger

Shipped Swift REPL

Since WWDC 2014

Shipped Swift debugger

Shipped Swift REPL

- LLDB in disguise

Since WWDC 2014

Shipped Swift debugger

Shipped Swift REPL

- LLDB in disguise

Numerous improvements

Since WWDC 2014

Shipped Swift debugger

Shipped Swift REPL

- LLDB in disguise

Numerous improvements

- Swift types show inherited Objective-C fields

Since WWDC 2014

Shipped Swift debugger

Shipped Swift REPL

- LLDB in disguise

Numerous improvements

- Swift types
- Help includes command aliases

Since WWDC 2014

Shipped Swift debugger

Shipped Swift REPL

- LLDB in disguise

Numerous improvements

- Swift types
- Help includes command aliases

```
help
```

Since WWDC 2014

Shipped Swift debugger

Shipped Swift REPL

- LLDB in disguise

Numerous improvements

- Swift types
- Help includes command aliases

help

Since WWDC 2014

Shipped Swift debugger

Shipped Swift REPL

- LLDB in disguise

Numerous improvements

- Swift types
- Help includes command aliases

help

expression **-O --**

Since WWDC 2014

Shipped Swift debugger

Shipped Swift REPL

- LLDB in disguise

Numerous improvements

- Swift types
- Help includes command aliases

help

expression **-O --** **po**

Since WWDC 2014

Shipped Swift debugger

Shipped Swift REPL

- LLDB in disguise

Numerous improvements

- Swift types
- Help includes command aliases
- Improved data formatting

help

expression **-O --** **po**

Since WWDC 2014

Shipped Swift debugger

Shipped Swift REPL

- LLDB in disguise

Numerous improvements

- Swift types
- Help includes command aliases
- Improved data formatting
  - Set<T>, NSIndexPath

help

expression **-O --** **po**

Since WWDC 2014

Shipped Swift debugger

Shipped Swift REPL

- LLDB in disguise

Numerous improvements

- Swift types
- Help includes command aliases
- Improved data formatting
  - Set<T>, NSIndexPath
- printf() prototype for expressions

help

expression **-O --** **po**

Since WWDC 2014

Shipped Swift debugger

Shipped Swift REPL

- LLDB in disguise

Numerous improvements

- Swift types
- Help includes command aliases
- Improved data formatting
  - Set<T>, NSIndexPath
- printf() prototype for expressions
- Improved disassembly

help

expression **-O --** **po**

Recent Breakpoint Enhancements

Named breakpoints

- Multiple non-unique names allowed
- Other breakpoint commands take names

Recent Breakpoint Enhancements

Named breakpoints

- Multiple non-unique names allowed
- Other breakpoint commands take names

breakpoint **s**et **-N** ***name***

Recent Breakpoint Enhancements

Named breakpoints

- Multiple non-unique names allowed
- Other breakpoint commands take names

breakpoint **s**et **-N** ***name***

breakpoint **e**nable ***name***

breakpoint **di**sable ***name***

breakpoint **de**lete ***name***

Recent Breakpoint Enhancements

Named breakpoints

- Multiple non-unique names allowed
- Other breakpoint commands take names

breakpoint **s**et **-N *name***

breakpoint **e**nable ***name***

breakpoint **di**sable ***name***

breakpoint **de**lete ***name***

Breakpoints in `~/.lldbinit`

- Create default breakpoints when LLDB starts
- Inherited by all debug targets

Recent Breakpoint Enhancements

Named breakpoints

- Multiple non-unique names allowed
- Other breakpoint commands take names

breakpoint **s**et **-N *name***

breakpoint **e**nable ***name***

breakpoint **di**sable ***name***

breakpoint **de**lete ***name***

Breakpoints in **`~/.lldbinit`**

- Create default breakpoints when LLDB starts
- Inherited by all debug targets

```
breakpoint set –n malloc –N memory
breakpoint set –n free –N memory
breakpoint disable memory
```

Xcode 7

Xcode 7

Improvements in expression evaluation

- Swift 2 support
- Objective-C module support

Xcode 7

Improvements in expression evaluation

- Swift 2 support
- Objective-C module support

Improved data formatters

- Vector types in Objective-C and Swift
- Custom data formatters written in Swift

Xcode 7

Improvements in expression evaluation

- Swift 2 support
- Objective-C module support

Improved data formatters

- Vector types in Objective-C and Swift
- Custom data formatters written in Swift

Address Sanitizer integration

Xcode 7

Improvements in expression evaluation

- Swift 2 support
- Objective-C module support

Improved data formatters

- Vector types in Objective-C and Swift
- Custom data formatters written in Swift

Address Sanitizer integration

memory **h**istory

memory **r**ead

memory **w**rite

memory **f**ind

Xcode 7

Improvements in expression evaluation

- Swift 2 support
- Objective-C module support

Improved data formatters

- Vector types in Objective-C and Swift
- Custom data formatters written in Swift

Address Sanitizer integration

Type lookup command

- Built-in type documentation

memory **h**istory

memory **r**ead

memory **w**rite

memory **f**ind

Type Lookup

Type Lookup

```
(lldb)
```

Type Lookup

```
(lldb) type lookup ErrorType
protocol ErrorType {
  var _domain: Swift.String { get }
  var _code: Swift.Int { get }
}

(lldb)
```

Type Lookup

```
(lldb) type lookup ErrorType
protocol ErrorType {
  var _domain: Swift.String { get }
  var _code: Swift.Int { get }
}
```

```
(lldb) type lookup Comparable
protocol Comparable : _Comparable, Equatable {
  func <=(lhs: Self, rhs: Self) -> Swift.Bool
  func >=(lhs: Self, rhs: Self) -> Swift.Bool
  func >(lhs: Self, rhs: Self) -> Swift.Bool
}
```

Type Lookup

```
(lldb) type lookup ErrorType
protocol ErrorType {
  var _domain: Swift.String { get }
  var _code: Swift.Int { get }
}
```

type **l**ookup

```
(lldb) type lookup Comparable
protocol Comparable : _Comparable, Equatable {
  func <=(lhs: Self, rhs: Self) -> Swift.Bool
  func >=(lhs: Self, rhs: Self) -> Swift.Bool
  func >(lhs: Self, rhs: Self) -> Swift.Bool
}
```

Compilers in LLDB

Making Swift and Objective-C better and easier to debug

Overview

Objective-C expression improvements

Swift error-handling support

Overview

Objective-C expression improvements

Swift error-handling support

Objective-C and Swift in LLDB

Objective-C and Swift in LLDB

Objective-C and Swift in LLDB

Objective-C and Swift in LLDB

How Swift Works in LLDB

```
(lldb)
```

How Swift Works in LLDB

```
(lldb) p for i in (0..<5) { print(i) }
```

How Swift Works in LLDB

```
(lldb) p for i in (0..<5) { print(i) }
```

expression **--** | **p**

How Swift Works in LLDB

```
(lldb) p for i in (0..<5) { print(i) }
0
1
2
3
4
```

expression **--** | **p**

How Swift Works in LLDB


```
(lldb) p for i in (0..<5) { print(i) }
0
1
2
3
4
```

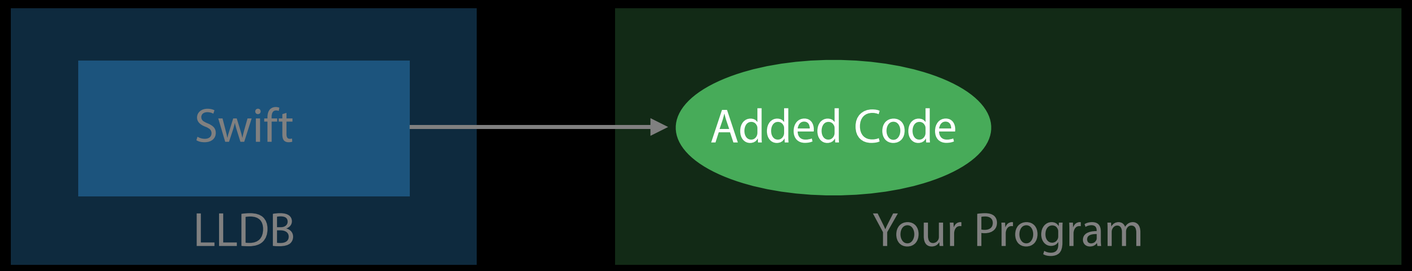
expression **--** **p**

How Swift Works in LLDB


```
(lldb) p for i in (0..<5) { print(i) }
0
1
2
3
4
```

expression **--** **p**

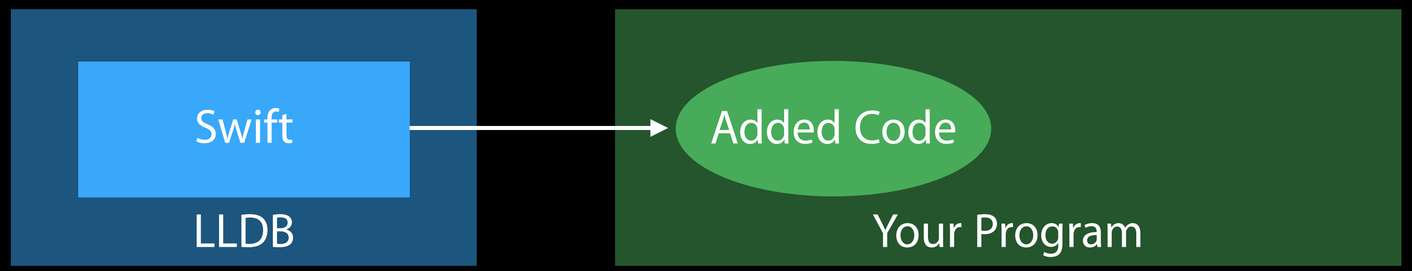
How Swift Works in LLDB




```
(lldb) p for i in (0..<5) { print(i) }
0
1
2
3
4
```

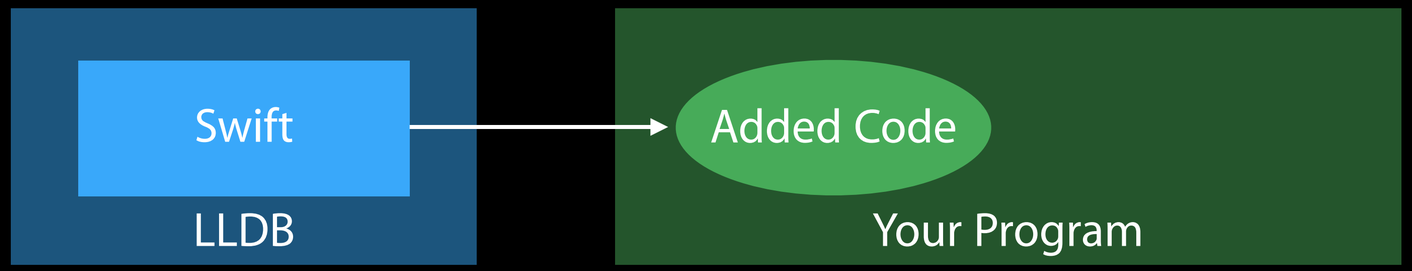
expression **--** | **p**

How Swift Works with Your Variables



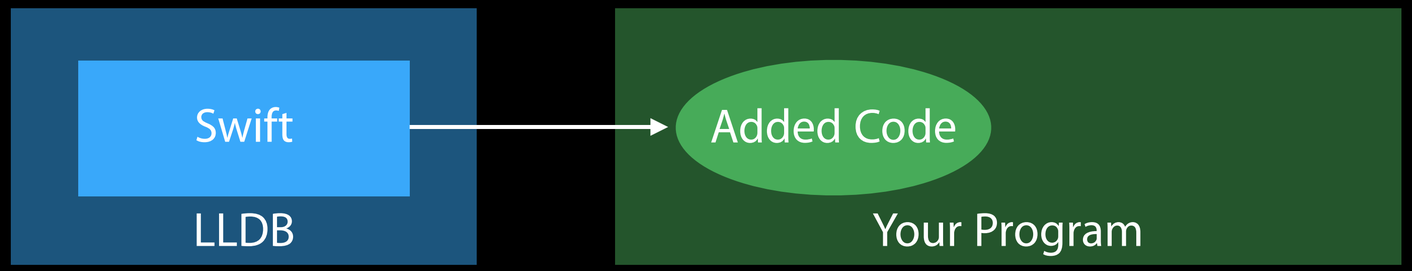

```
(lldb)
```

How Swift Works with Your Variables



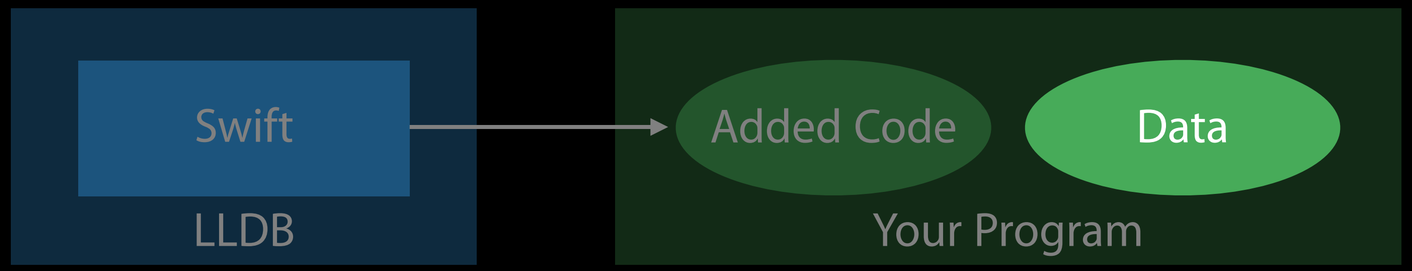

```
(lldb) p for i in (0..<3) { print(names[i]) }
```

How Swift Works with Your Variables



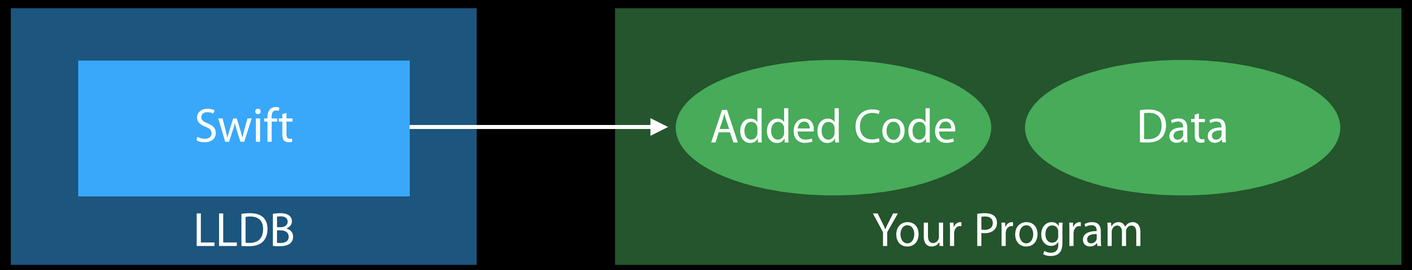

```
(lldb) p for i in (0..<3) { print(names[i]) }
Kate
Sean
Enrico
```

How Swift Works with Your Variables



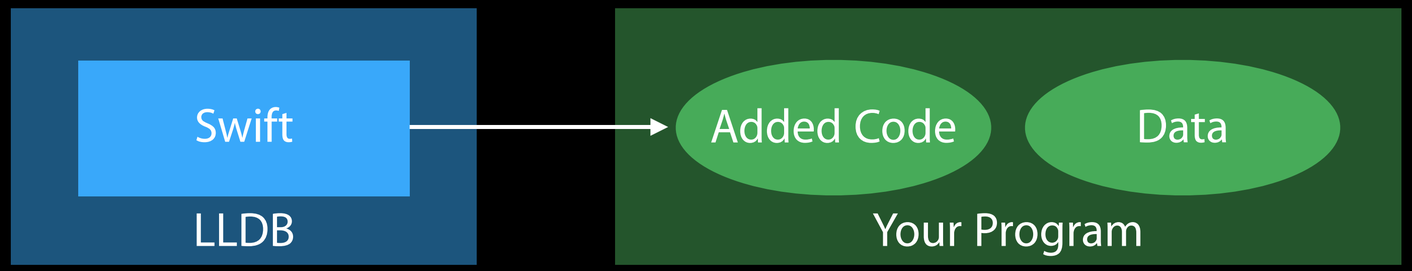

```
(lldb) p for i in (0..<3) { print(names[i]) }
Kate
Sean
Enrico
```

How Swift Works with the SDK



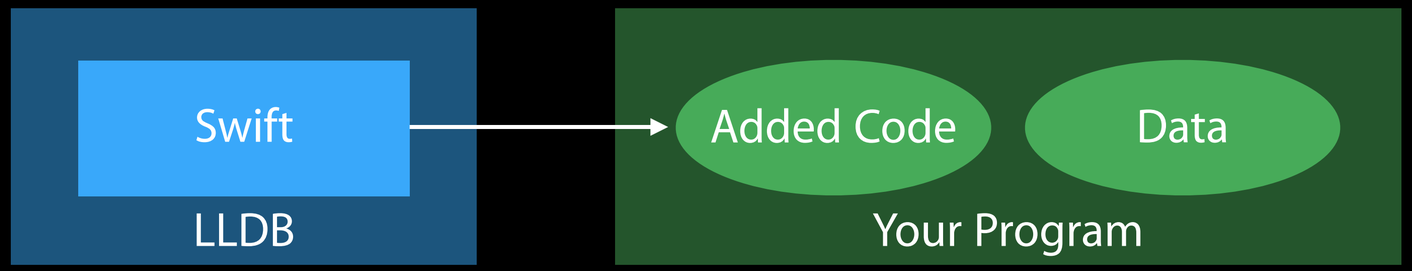

```
(lldb)
```

How Swift Works with the SDK



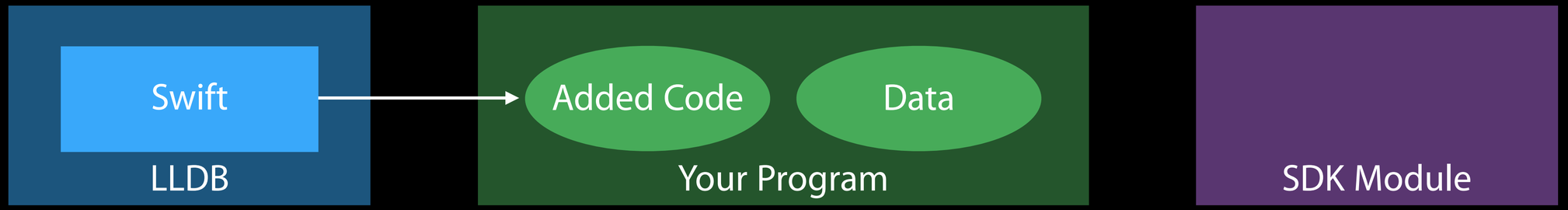

```
(lldb) p NSApplication.sharedApplication()
```

How Swift Works with the SDK



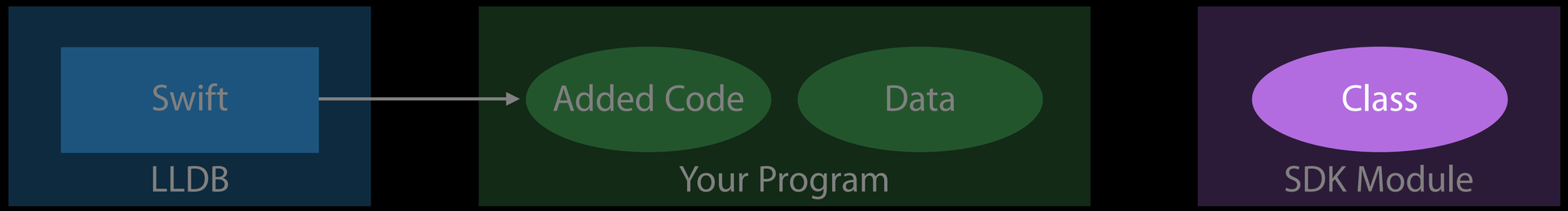

```
(lldb) p NSApplication.sharedApplication()
(NSApplication) $R1 = 0x0006000000100240 {
  AppKit.NSResponder = {
    NSObject = {
      isa = NSApplication
    }
    _nextResponder = nil
  }
}
```

How Swift Works with the SDK




```
(lldb) p NSApplication.sharedApplication()
(NSApplication) $R1 = 0x0006000000100240 {
  AppKit.NSResponder = {
    NSObject = {
      isa = NSApplication
    }
    _nextResponder = nil
  }
}
```

How Swift Works with the SDK




```
(lldb) p NSApplication.sharedApplication()
(NSApplication) $R1 = 0x0006000000100240 {
  AppKit.NSResponder = {
    NSObject = {
      isa = NSApplication
    }
    _nextResponder = nil
  }
}
```

It’s All Automatic!

It’s All Automatic!

…in Swift

Let's Try the Same Thing in Objective-C!

```
(lldb)
```

Let’s Try the Same Thing in Objective-C!

```
(lldb) p NSLog(@“%d”, i)
```

Let’s Try the Same Thing in Objective-C!

```
(lldb) p NSLog(@“%d”, i)
error: 'NSLog' has unknown return type;
  cast the call to its declared return type
error: 1 errors parsing expression
```

Let’s Try the Same Thing in Objective-C!

```
(lldb) p NSLog(@“%d”, i)
error: 'NSLog' has unknown return type;
  cast the call to its declared return type
error: 1 errors parsing expression
```

How it’s defined	`void NSLog(NSString *format, ...)`
What LLDB sees	`??? NSLog(...)`

Let’s Try the Same Thing in Objective-C!

```
(lldb) p NSLog(@“%d”, i)
2015-05-14 21:50:41.852 DemoApp[58380:1079308] 0
```

Let's Try the Same Thing in Objective-C!

```
(lldb)
```

Let's Try the Same Thing in Objective-C!

```
(lldb) p NSMakeRect(0, 0, 10, 10)
```

Let’s Try the Same Thing in Objective-C!

```
(lldb) p NSMakeRect(0, 0, 10, 10)
error: use of undeclared identifier ‘NSMakeRect’
error: 1 errors parsing expression
```

Let’s Try the Same Thing in Objective-C!

```
(lldb) p NSMakeRect(0, 0, 10, 10)
error: use of undeclared identifier ‘NSMakeRect’
error: 1 errors parsing expression
```

How it’s defined

```
NS_INLINE NSRect NSMakeRect(
    CGFloat, CGFloat, CGFloat, CGFloat)
```

What LLDB sees

Let's Try the Same Thing in Objective-C!

```
(lldb) p NSMakeRect(0, 0, 10, 10)
(NSRect) $0 = (origin = (x = 0, y = 0), size = (width = 10, height = 10))
```

Let's Try the Same Thing in Objective-C!

```
(lldb)
```

Let's Try the Same Thing in Objective-C!

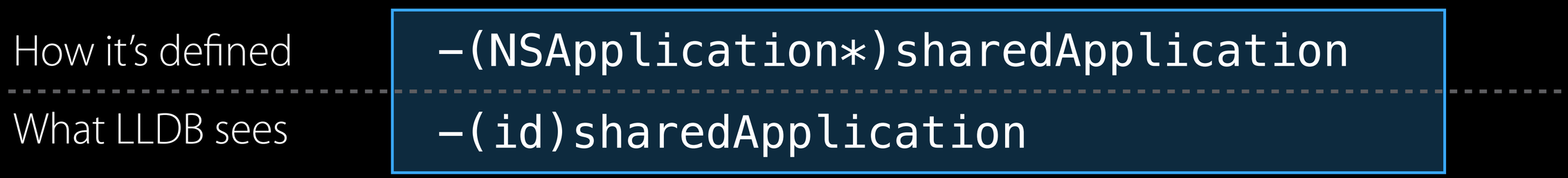
```
(lldb) p [NSApplication sharedApplication].undoManager
```

Let’s Try the Same Thing in Objective-C!

```
(lldb) p [NSApplication sharedApplication].undoManager
error: property ‘undoManager’ not found on object of type ‘id’
error: 1 errors parsing expression
```

Let’s Try the Same Thing in Objective-C!

```
(lldb) p [NSApplication sharedApplication].undoManager
error: property ‘undoManager’ not found on object of type ‘id’
error: 1 errors parsing expression
```



Let's Try the Same Thing in Objective-C!

```
(lldb) p [NSApplication sharedApplication].undoManager
(NSUndoManager * __nullable) $1 = nil
```

“The Information Is Right There!”

“The Information Is Right There!”

Your Code

```
names, MyFunc(), MyView
```

“The Information Is Right There!”

Your Code	`names, MyFunc(), MyView`	
SDK Functions	`NSLog, NSMakeRect`	

“The Information Is Right There!”

Your Code	`names, MyFunc(), MyView`	
SDK Functions	`NSLog, NSMakeRect`	
SDK Classes	`NSView, NSApplication`	 *

“The Information Is Right There!”

Your Code	`names, MyFunc(), MyView`	✓
SDK Functions	`NSLog, NSMakeRect`	✗
SDK Classes	`NSView, NSApplication`	✓ *
SDK Constants	`NSASCIIStringEncoding`	✗

“The Information Is Right There!”

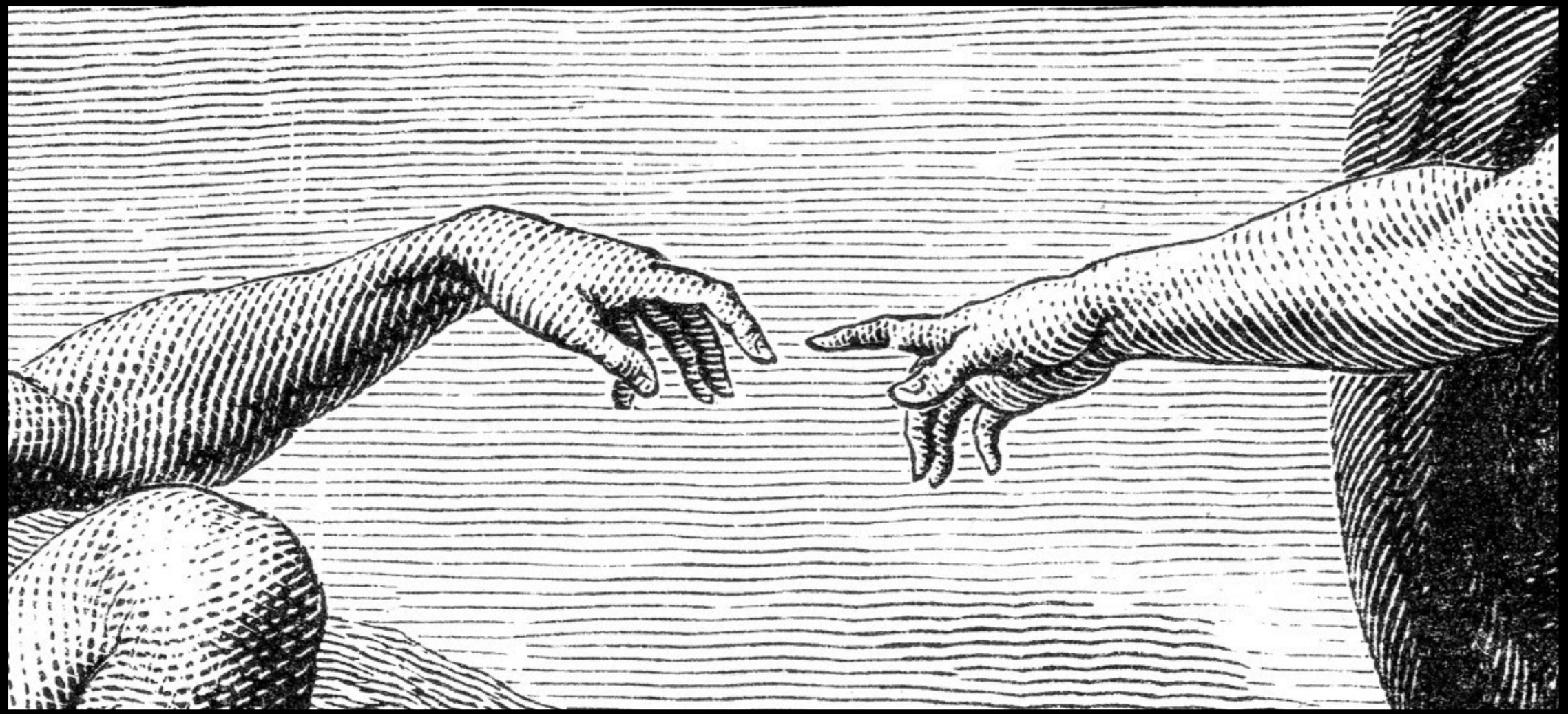
Your Code	`names, MyFunc(), MyView`	✓
SDK Functions	`NSLog, NSMakeRect`	✗
SDK Classes	`NSView, NSApplication`	✓ *
SDK Constants	`NSASCIIStringEncoding`	✗
Macros	`INT_MAX, MAX()`	✗

“The Information Is Right There!”

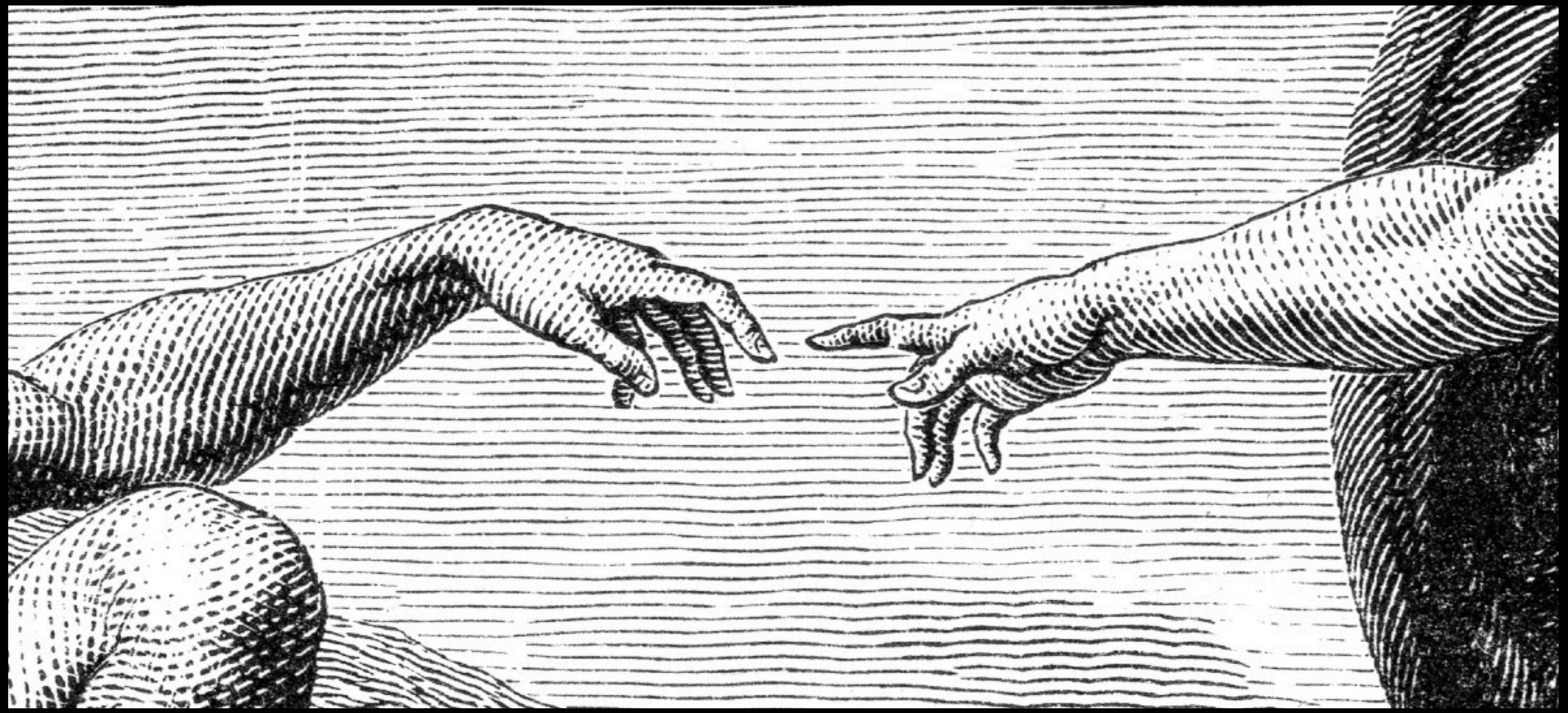
Your Code	`names, MyFunc(), MyView`	✓
SDK Functions	`NSLog, NSMakeRect`	✓
SDK Classes	`NSView, NSApplication`	✓ *
SDK Constants	`NSASCIIStringEncoding`	✓
Macros	`INT_MAX, MAX()`	✓

“The Information Is Right There!”

Your Code	`names, MyFunc(), MyView`	
SDK Functions	`NSLog, NSMakeRect`	
SDK Classes	`NSView, NSApplication`	
SDK Constants	`NSASCIIStringEncoding`	
Macros	`INT_MAX, MAX()`	



expr @import AppKit



```
expr @import UIKit
```

Overview

Objective-C expression improvements

Swift error-handling support

Handling Errors in Swift Expressions

Handling Errors in Swift Expressions

Expressions can handle Swift errors

Handling Errors in Swift Expressions

Expressions can handle Swift errors

- No need to “try” in LLDB

Handling Errors in Swift Expressions

Expressions can handle Swift errors

- No need to "try" in LLDB

```
(lldb)
```

Handling Errors in Swift Expressions

Expressions can handle Swift errors

- No need to "try" in LLDB

```
(lldb) expr ThisFunctionThrows()
```

Handling Errors in Swift Expressions

Expressions can handle Swift errors

- No need to "try" in LLDB

```
(lldb) expr ThisFunctionThrows()
(a.EnumError) $E0 = SeriousError
```

Handling Errors in Swift Expressions

Expressions can handle Swift errors

- No need to "try" in LLDB

```
(lldb) expr ThisFunctionThrows()
(a.EnumError) $E0 = SeriousError
```

- Works in the REPL, too!

Handling Errors in Swift Expressions

Expressions can handle Swift errors

- No need to "try" in LLDB

```
(lldb) expr ThisFunctionThrows()
(a.EnumError) $E0 = SeriousError
```

- Works in the REPL, too!

```
1>
```

Handling Errors in Swift Expressions

Expressions can handle Swift errors

- No need to "try" in LLDB

```
(lldb) expr ThisFunctionThrows()
(a.EnumError) $E0 = SeriousError
```

- Works in the REPL, too!

```
1> ThisFunctionThrows()
```

Handling Errors in Swift Expressions

Expressions can handle Swift errors

- No need to "try" in LLDB

```
(lldb) expr ThisFunctionThrows()
(a.EnumError) $E0 = SeriousError
```

- Works in the REPL, too!

```
1> ThisFunctionThrows()
$E0: EnumError = MyError
```

Stopping When Swift Errors Occur

Stopping When Swift Errors Occur

Use breakpoints

Stopping When Swift Errors Occur

Use breakpoints

- On Objective-C exceptions

Stopping When Swift Errors Occur

Use breakpoints

- On Objective-C exceptions

```
(lldb)
```

Stopping When Swift Errors Occur

Use breakpoints

- On Objective-C exceptions

```
(lldb) br s -E objc
```

Stopping When Swift Errors Occur

Use breakpoints

- On Objective-C exceptions

```
(lldb) br s –E objc
```

breakpoint **s**et **-E *language***

Stopping When Swift Errors Occur

Use breakpoints

- On Objective-C exceptions

```
(lldb) br s -E objc
Breakpoint 1: where = libobjc.A.dylib`objc_exception_throw,
              address = 0x00007fff9046bd18
```

breakpoint **s**et **-E *language***

Stopping When Swift Errors Occur

Use breakpoints

- On Objective-C exceptions

```
(lldb) br s -E objc
Breakpoint 1: where = libobjc.A.dylib`objc_exception_throw,
            address = 0x00007fff9046bd18
```

breakpoint **s**et **-E *language***

- On Swift errors

Stopping When Swift Errors Occur

Use breakpoints

- On Objective-C exceptions

```
(lldb) br s -E objc
Breakpoint 1: where = libobjc.A.dylib`objc_exception_throw,
            address = 0x00007fff9046bd18
```

breakpoint **s**et **-E *language***

- On Swift errors

```
(lldb)
```

Stopping When Swift Errors Occur

Use breakpoints

- On Objective-C exceptions

```
(lldb) br s -E objc
Breakpoint 1: where = libobjc.A.dylib`objc_exception_throw,
            address = 0x00007fff9046bd18
```

breakpoint **s**et **-E *language***

- On Swift errors

```
(lldb) br s -E swift
```

Stopping When Swift Errors Occur

Use breakpoints

- On Objective-C exceptions

```
(lldb) br s -E objc
Breakpoint 1: where = libobjc.A.dylib`objc_exception_throw,
            address = 0x00007fff9046bd18
```

breakpoint **s**et **-E *language***

- On Swift errors

```
(lldb) br s -E swift
Breakpoint 1: where = libswiftCore.dylib`swift_willThrow,
            address = 0x00000001001f71e0
```

Stopping on Specific Errors

Stopping on Specific Errors

Supported for Swift

Stopping on Specific Errors

Supported for Swift

```
(lldb)
```

Stopping on Specific Errors

Supported for Swift

```
(lldb) br s -E swift -O EnumError
Breakpoint 1: where = libswiftCore.dylib`swift_willThrow,
              address = 0x00000001001f71e0
(lldb)
```

Stopping on Specific Errors

Supported for Swift

```
(lldb) br s -E swift -O EnumError
Breakpoint 1: where = libswiftCore.dylib`swift_willThrow,
               address = 0x00000001001f71e0
(lldb)
```

breakpoint **s**et
-E *language*
-O *type-name*

Stopping on Specific Errors

Supported for Swift

```
(lldb) br s -E swift -O EnumError
Breakpoint 1: where = libswiftCore.dylib`swift_willThrow,
               address = 0x00000001001f71e0
(lldb) continue
   25          if input > 100
   26          {
-> 27              throw EnumError.ImportantError
   28          }
   29          else if input > 10
   30          {
```

breakpoint **s**et
-E *language*
-O *type-name*

Of Course, You Can Still Catch Them

```
1>
```

Of Course, You Can Still Catch Them

```
1> import Foundation
2>
```

Of Course, You Can Still Catch Them

```
1> import Foundation
2> do {
       throw NSError(domain: “My domain”, code: 0,
                     userInfo:nil)
   } catch let x {
       print(x)
   }
```

Of Course, You Can Still Catch Them

```
1> import Foundation
2> do {
       throw NSError(domain: “My domain”, code: 0,
                     userInfo:nil)
   } catch let x {
       print(x)
   }
Error Domain=My domain Code=0
“The operation couldn’t be completed. (My domain error 0.)”
```

Presentation Is Everything

Formatted data is comprehended data

Examining Data

Examining Data

frame **v**ariable

Examining Data

frame **v**ariable

expression **--**

Examining Data

frame **v**ariable

expression **--** **p**

Examining Data

frame **v**ariable

expression -- **p**

expression **-O** --

Examining Data

frame **v**ariable

expression -- **p**

expression **-O** -- **po**

The “Frame Variable” Command

The “Frame Variable” Command

frame **v**ariable

The “Frame Variable” Command

```
(lldb) frame variable
(Int, String) myTuple = (0 = 12, 1 = “Hello World”)
(Int) theYear = 1984
```

frame **v**ariable

The “Frame Variable” Command

```
(lldb) frame variable
(Int, String) myTuple = (0 = 12, 1 = “Hello World”)
(Int) theYear = 1984
```

```
(lldb) frame variable theYear
(Int) theYear = 1984
```

frame **v**ariable

The “Frame Variable” Command

```
(lldb) frame variable
(Int, String) myTuple = (0 = 12, 1 = “Hello World”)
(Int) theYear = 1984
```

```
(lldb) frame variable theYear
(Int) theYear = 1984
```

```
(lldb) frame variable ––format hex theYear
(Int) theYear = 0x00000000000007c0
```

frame **v**ariable

The “Frame Variable” Command

```
(lldb) frame variable
(Int, String) myTuple = (0 = 12, 1 = “Hello World”)
(Int) theYear = 1984
```

children

frame **v**ariable

```
(lldb) frame variable theYear
(Int) theYear = 1984

(lldb) frame variable —-format hex theYear
(Int) theYear = 0x00000000000007c0
```

The “Expression” Command

The "Expression" Command

expression **--** **p**

The "Expression" Command

```
(lldb) p theYear + 1
(Int) $R0 = 1985
```

expression **--** | **p**

The "Expression" Command

```
(lldb) p theYear + 1
(Int) $R0 = 1985

(lldb) p String($R0 + 1)
(String) $R1 = "1986"
```

expression **--** | **p**

The "Expression" Command

```
(lldb) p theYear + 1
(Int) $R0 = 1985

(lldb) p String($R0 + 1)
(String) $R1 = "1986"

(lldb) expr --format hex -- [1,2,3]
([Int]) $R0 = 3 values {
  [0] = 0x0000000000000001
  [1] = 0x0000000000000002
  [2] = 0x0000000000000003
}
```

expression **--** | **p**

The "Expression" Command

```
(lldb) p theYear + 1
(Int) $R0 = 1985

(lldb) p String($R0 + 1)
(String) $R1 = "1986"

(lldb) expr --format hex -- [1,2,3]
([Int]) $R0 = 3 values {
  [0] = 0x0000000000000001
  [1] = 0x0000000000000002
  [2] = 0x0000000000000003
}
```

expression **--** | **p**

children

The “po” Command

The "po" Command

expression **-O --** **po**

The “po” Command

```
(lldb) po [[MyView alloc] initWithFrame: myFrame]
<MyView: 0x101405110>
```

expression **-O --** | **po**

The "po" Command

```
(lldb) po [[MyView alloc] initWithFrame: myFrame]
<MyView: 0x101405110>

(lldb) po @[@1, @2, @3]
<__NSArrayI 0x10050b2a0>(
1,
2,
3
)
```

expression **-O --** | **po**

The “po” Command

```
(lldb) po [[MyView alloc] initWithFrame: myFrame]
<MyView: 0x101405110>

(lldb) po @[@1, @2, @3]
<__NSArrayI 0x10050b2a0>(
1,
2,
3
)

(lldb) po @“Hello World”
Hello World
```

expression **-O --** | **po**

Formatting Models

Formatting Models

frame **v**ariable	**ex**pression **--**	**p**	**ex**pression **-O --**	**po**
does not run code	runs your code		runs your code	
uses LLDB formatters	uses LLDB formatters		+ code to format object	

Formatting Models

frame **v**ariable	**ex**pression **--** **p**	**ex**pression **-O --** **po**
does not run code	runs your code	runs your code
uses LLDB formatters	uses LLDB formatters	+ code to format object

Formatting Models

frame **v**ariable	**ex**pression **--**	**p**	**ex**pression **-O --**	**po**
does not run code	runs your code		runs your code	
uses LLDB formatters	uses LLDB formatters		+ code to format object	

Formatting Models

Formatting Models

Out-of-process formatting

- Data and formatter are separate
- Easy access to debugger's object model
- Easier to keep program state intact

Formatting Models

Out-of-process formatting

- Data and formatter are separate
- Easy access to debugger's object model
- Easier to keep program state intact

In-process formatting

- Data and formatter live together
- Easy access to your object model
- Care required to keep program state intact

In-Process Swift Formatting

In-Process Swift Formatting

Powers playgrounds since day one

In-Process Swift Formatting

Powers playgrounds since day one

Now

- Public API
- Also powers LLDB “po” command

In-Process Swift Formatting

In-Process Swift Formatting

Four Swift protocols

In-Process Swift Formatting

Four Swift protocols

- `CustomStringConvertible`
- `CustomDebugStringConvertible`
- `CustomPlaygroundQuickLookable`
- `CustomReflectable`

In-Process Swift Formatting

Four Swift protocols

- `CustomStringConvertible`
- `CustomDebugStringConvertible`
- `CustomPlaygroundQuickLookable`
- `CustomReflectable`

Partial opt-in possible

CustomStringConvertible

String representation of the object

Used by the print() function, and string interpolation

CustomStringConvertible

String representation of the object

Used by the print() function, and string interpolation

```
struct BottlesOfBeer : CustomStringConvertible {
  var Count: Int
  var description: String {
    return “\(Count) bottles of beer on the wall”
  }
}
```

CustomDebugStringConvertible

Debugger-specific String representation of the object

Used by debugPrint() (and print() if necessary)

CustomDebugStringConvertible

Debugger-specific String representation of the object

Used by debugPrint() (and print() if necessary)

```
extension BottlesOfBeer : CustomDebugStringConvertible {
  var debugDescription: String {
     return “\(Count) bottles of stout on the wall.”
   }
}
```

CustomPlaygroundQuickLookable

Graphical representation for display in playgrounds

CustomPlaygroundQuickLookable

Ready | Today at 10:29 AM

WWDCSession_Playground

```
1  import Cocoa
2
3  struct Person : CustomPlaygroundQuickLookable {
4      func customPlaygroundQuickLook() -> PlaygroundQuickLook {
5          let path = NSBezierPath(rect: NSRect(x:0, y:0, width: 30, height: 75))                (2 times)
6          path.appendBezierPathWithRoundedRect(NSRect(x: 7.5, y: 75, width: 15, height: 30),    (2 times)
               xRadius: 10, yRadius: 10)
7          path.moveToPoint(NSPoint(x: 30, y:50))                                                (2 times)
8          path.lineToPoint(NSPoint(x: 45, y:40))                                                (2 times)
9          path.moveToPoint(NSPoint(x: 0, y:50))                                                 (2 times)
10         path.lineToPoint(NSPoint(x: 15, y:40))                                                (2 times)
11         path.moveToPoint(NSPoint(x: 5, y: 0))                                                 (2 times)
12         path.lineToPoint(NSPoint(x: 0, y: -10))                                               (2 times)
13         path.moveToPoint(NSPoint(x: 25, y:0))                                                 (2 times)
14         path.lineToPoint(NSPoint(x: 30, y:-10))                                               (2 times)
15         return PlaygroundQuickLook(reflecting: path)                                          (2 times)
16     }
17 }
18
19 Person()                                                                                      22 path elements
```

– 30 sec +

CustomReflectable

CustomReflectable

Allows vending an entirely custom children hierarchy

CustomReflectable

Allows vending an entirely custom children hierarchy

Returns a Mirror for an object

- Representation of an object's structure

Temperature Data

Temperature Data

```
struct Timestamp {
  var Hour: Int
  var Minute: Int
}
struct TemperatureData {
  var Time: Timestamp
  var Temperature: Float
}
```

Temperature Data

```
struct Timestamp {
  var Hour: Int
  var Minute: Int
}
struct TemperatureData {
  var Time: Timestamp
  var Temperature: Float
}


let Temps = [
  TemperatureData(Time: Timestamp(Hour: 6, Minute: 30), Temperature: 10),
  TemperatureData(Time: Timestamp(Hour: 18, Minute: 30), Temperature: 34)
]
```

Temperature Data

Temperature Data

```
(lldb) po Temps
```

Temperature Data

```
(lldb) po Temps
▿ 2 elements
  ▿ [0] : TemperatureData
    ▿ Time : Timestamp
      - Hour : 6
      - Minute : 30
    - Temperature : 10
  ▿ [1] : TemperatureData
    ▿ Time : Timestamp
      - Hour : 18
      - Minute : 30
    - Temperature : 34
```

Temperature Data

```
(lldb) po Temps
▿ 2 elements
  ▿ [0] : TemperatureData
    ▿ Time : Timestamp
      – Hour : 6
      – Minute : 30
    – Temperature : 10
  ▿ [1] : TemperatureData
    ▿ Time : Timestamp
      – Hour : 18
      – Minute : 30
    – Temperature : 34
```

Time on one line, AM/PM format

Temperature Data

```
(lldb) po Temps
▿ 2 elements
  ▿ [0] : TemperatureData
    ▿ Time : Timestamp
      – Hour : 6
      – Minute : 30
    – Temperature : 10
  ▿ [1] : TemperatureData
    ▿ Time : Timestamp
      – Hour : 18
      – Minute : 30
    – Temperature : 34
```

Temperature in F

Time on one line, AM/PM format

Step 1:Time on One Line

Step 1: Time on One Line

```
extension Timestamp : CustomStringConvertible {
  var description: String {
    let formatter = NSDateFormatter()
    formatter.dateStyle = NSDateFormatterStyle.NoStyle
    formatter.timeStyle = NSDateFormatterStyle.ShortStyle
    formatter.timeZone = NSTimeZone(forSecondsFromGMT: 0)
    let date = NSDate(timeIntervalSince1970:
        NSTimeInterval(60 * (Minute + (60*Hour))))
    return formatter.stringFromDate(date)
  }
}
```

Step 2: Fahrenheit Degrees

Step 2: Fahrenheit Degrees

```
extension TemperatureData : CustomReflectable {
  func customMirror() -> Mirror {
    return Mirror(self, children: [

      “Time” : “\(Time)”,

      “Temp C” : Temperature,

      “Temp F” : 1.8*Temperature+32

    ])
  }
}
```

Step 2: Fahrenheit Degrees

```
extension TemperatureData : CustomReflectable {
  func customMirror() -> Mirror {
    return Mirror(self, children: [

      “Time” : “\(Time)”,
      “Temp C” : Temperature,

      “Temp F” : 1.8*Temperature+32

    ])
  }
}
```

CustomStringConvertible from previous slide

Temperature Data

Temperature Data

```
(lldb) po Temps
```

Temperature Data

```
(lldb) po Temps
▿ 2 elements
  ▿ [0] : TemperatureData
    - Time : “6:30 AM”
    - Temp C : 10
    - Temp F : 50
  ▿ [1] : TemperatureData
    - Time : “6:30 PM”
    - Temp C : 34
    - Temp F : 93.2
```

Temperature Data

```
(lldb) po Temps
▿ 2 elements
  ▿ [0] : TemperatureData
    – Time : “6:30 AM”
    – Temp C : 10
    – Temp F : 50
  ▿ [1] : TemperatureData
    – Time : “6:30 PM”
    – Temp C : 34
    – Temp F : 93.2
```

That’s hot!

At Runtime Too!

Conformances can also be added while debugging

At Runtime Too!

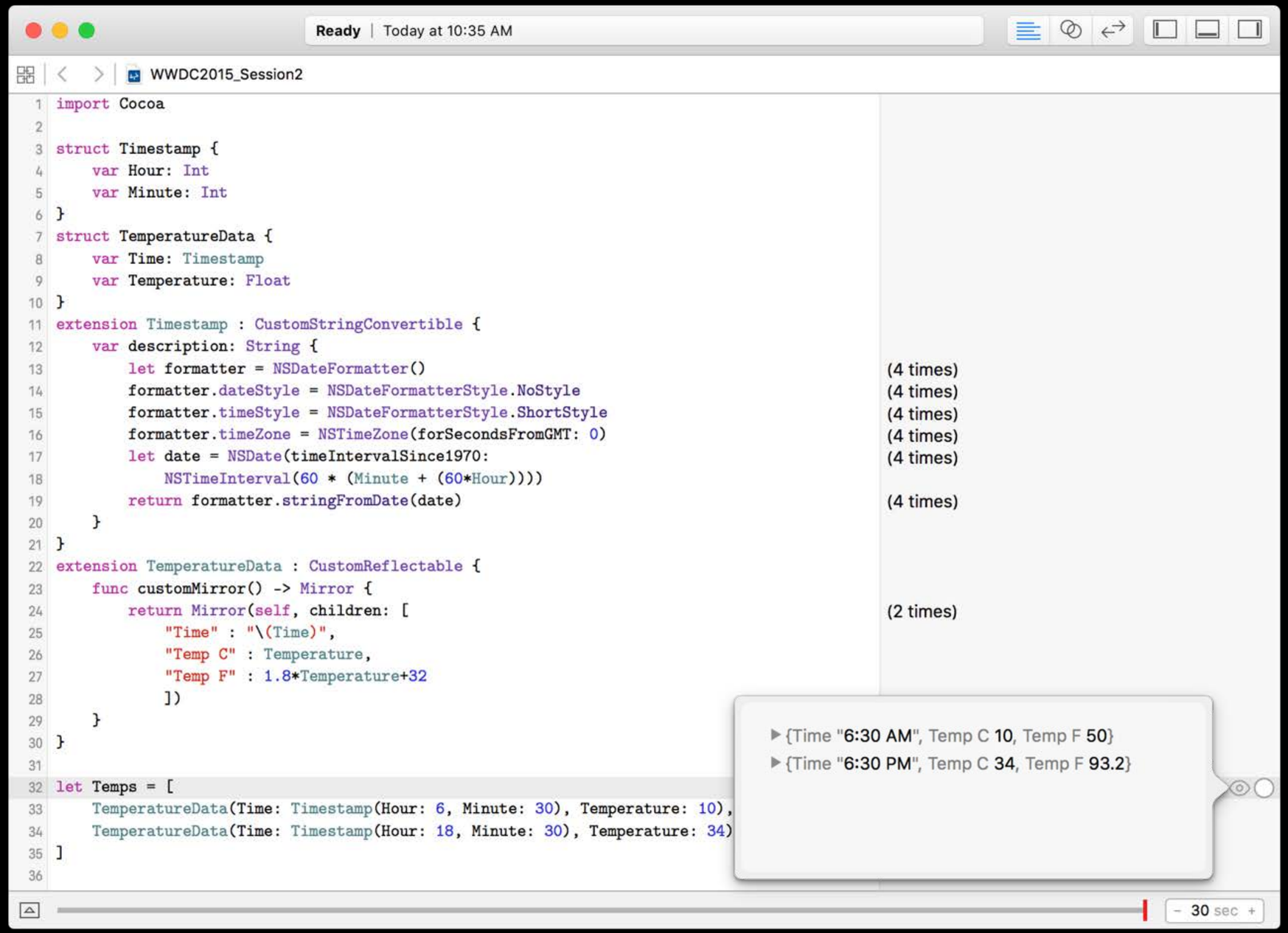
Conformances can also be added while debugging but not changed

At Runtime Too!

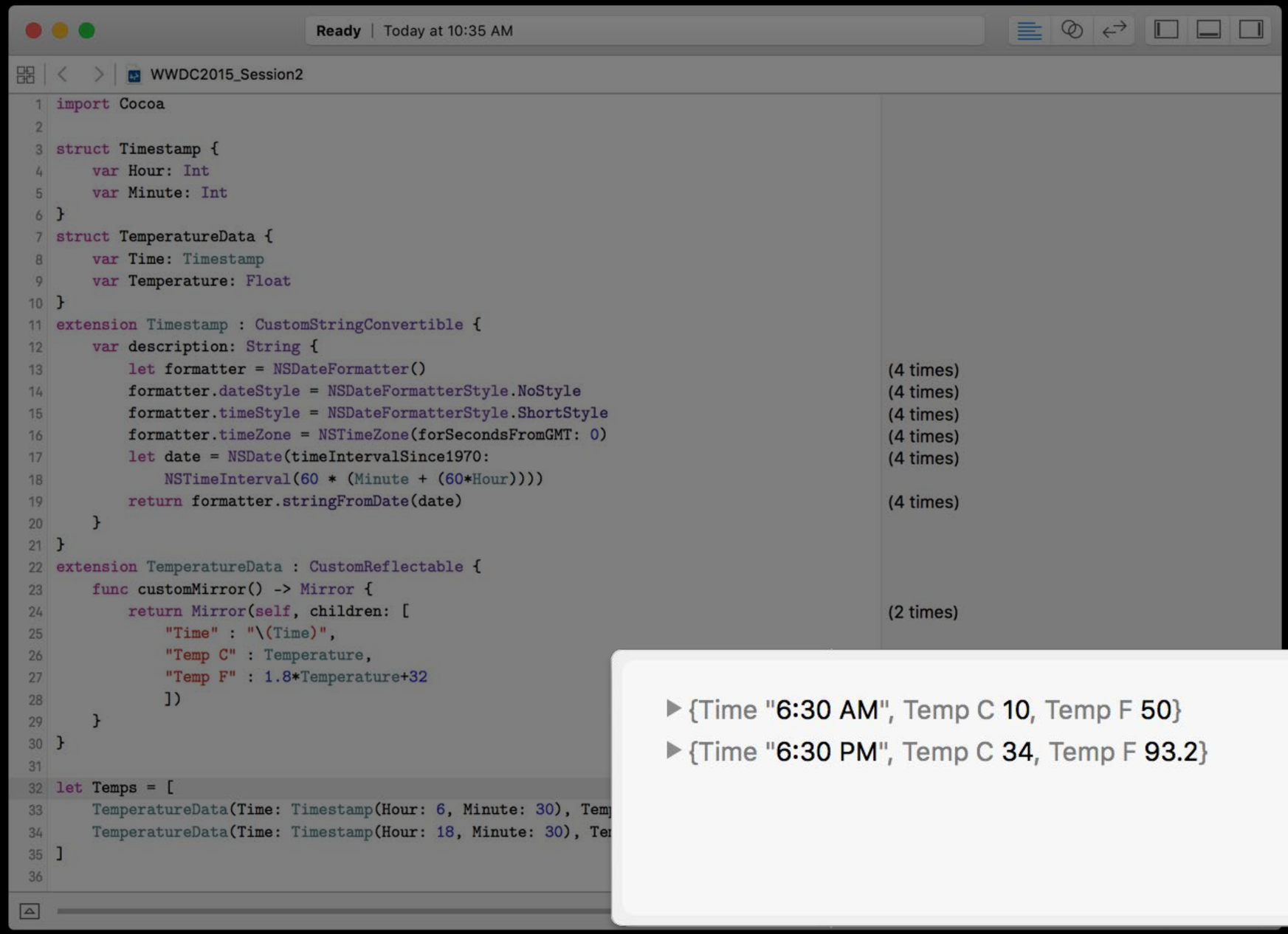
Conformances can also be added while debugging but not changed

They can also be added in the REPL

And in Playgrounds!



And in Playgrounds!



Summary

Summary

More information available more often

Summary

More information available more often

- Objective-C runtime

Summary

More information available more often

- Objective-C runtime
- SDK modules

Summary

More information available more often

- Objective-C runtime
- SDK modules
- In-process formatting

More Information

LLDB Documentation
http://lldb.llvm.org

Swift Language Documentation
http://developer.apple.com/swift

Apple Developer Forums
http://developer.apple.com/forums

Stefan Lesser
Developer Tools Evangelist
slesser@apple.com

Related Sessions

Authoring Rich Playgrounds	Presidio	Wednesday 10:00AM
Advanced Debugging and the Address Sanitizer	Mission	Friday 9:00AM


WWDC15