Quick Introduction to Swift

...with an eye to Flutter

Meta

1. Looking at native code in conjunction with Flutter, so Flutter-provided sample code reviewed

Meta

1. Looking at native code in conjunction with Flutter, so Flutter-provided sample code reviewed
2. Focus is on base language, not UI or platform

Meta

1. Looking at native code in conjunction with Flutter, so Flutter-provided sample code reviewed

2. Focus is on base language, not UI or platform
...but if you're interested in UI, check out SwiftUI
https://developer.apple.com/xcode/swiftui/

Meta

1. Looking at native code in conjunction with Flutter, so Flutter-provided sample code reviewed
2. Focus is on base language, not UI or platform
...but if you're interested in UI, check out SwiftUI
https://developer.apple.com/xcode/swiftui/
3. Far too much for one-hour talk, so these are just the highlights. The intent is to provide mostly just a feel for the language.

Docker

Swift is available via Docker: (~1.7GB)

```
docker pull swift
```

To run REPL:

```
docker run --security-opt seccomp=unconfined -it swift
# swift
...
^D to exit
```

More details here:

https://github.com/apple/swift-docker

Topics

1. Brief history (ObjC)
2. Decoding Flutter sample code in Objective-C
3. Brief history (Swift)
4. Swift basics
5. Decoding Flutter sample code in Swift
6. Beyond the basics

1. Brief History (ObjC)

Objective-C was used with Mac OS X 10.0 (2001) and onwards (.m extension)

Open source (clang.llvm.org), strict superset of C, used by NeXTSTEP

True "message-passing model" like Smalltalk

Lots of brackets [], like parentheses in LISP

Manual memory allocation/deallocation, "retain count"

```
MyObject * myObject = [[MyObject alloc] init];
[myObject release];
```

Brackets and manual memory management were addressed somewhat with dot notation and Automatic Reference Counting (ARC) in 2006 ("Objective-C 2.0").

```
[myObject setCount: [otherObject count]];
myObject.count = otherObject.count;
```

1. Brief History (cont'd)

Named parameters made clear what was being passed

```
NSString * fileContents =
    [NSString stringWithContentsOfFile: @"/x.txt"
                              encoding: NSUTF8StringEncoding
                                 error: NULL];
```

Easy to have memory leaks because of manual memory management (before ObjC 2.0)

Easy to miss nil pointers since no NPE (treated instead as no-op)

2019 Stack Overflow developer survey lists ObjC as second-most dreaded language

https://insights.stackoverflow.com/survey/2019/#most-loved-dreaded-and-wanted

2. Decoding Flutter (ObjC)

https://flutter.dev/docs/development/platform-integration/platform-channels

```
#import <Flutter/Flutter.h>
#import "GeneratedPluginRegistrant.h"

@implementation AppDelegate
- (BOOL)application:(UIApplication*)application didFinishLaunchingWithOptions:
(NSDictionary*)launchOptions {
  FlutterViewController* controller = (FlutterViewController*)self.window.rootViewController;

  FlutterMethodChannel* batteryChannel = [FlutterMethodChannel
                                          methodChannelWithName:@"samples.flutter.dev/battery"
                                          binaryMessenger:controller.binaryMessenger];

  [batteryChannel setMethodCallHandler:^(FlutterMethodCall* call, FlutterResult result) {
    // Note: this method is invoked on the UI thread.
    // TODO
  }];

  [GeneratedPluginRegistrant registerWithRegistry:self];
  return [super application:application didFinishLaunchingWithOptions:launchOptions];
}
```

2. Decoding Flutter (ObjC)

https://flutter.dev/docs/development/platform-integration/platform-channels

```
#import <Flutter/Flutter.h>
#import "GeneratedPluginRegistrant.h"
```

```
#import replaces #include + #ifndef + #define
```

```
@implementation AppDelegate
- (BOOL)application:(UIApplication*)application didFinishLaunchingWithOptions:
(NSDictionary*)launchOptions {
  FlutterViewController* controller = (FlutterViewController*)self.window.rootViewController;

  FlutterMethodChannel* batteryChannel = [FlutterMethodChannel
                                          methodChannelWithName:@"samples.flutter.dev/battery"
                                          binaryMessenger:controller.binaryMessenger];

  [batteryChannel setMethodCallHandler:^(FlutterMethodCall* call, FlutterResult result) {
    // Note: this method is invoked on the UI thread.
    // TODO
  }];

  [GeneratedPluginRegistrant registerWithRegistry:self];
  return [super application:application didFinishLaunchingWithOptions:launchOptions];
}
```

2. Decoding Flutter (ObjC)

https://flutter.dev/docs/development/platform-integration/platform-channels

@interface declares the public vars/methods (.h)
@implementation defines the vars/methods (.m)

```
#import <Flutter/Flutter.h>
#import "GeneratedPluginRegistrant.h"

@implementation AppDelegate
- (BOOL)application:(UIApplication*)application didFinishLaunchingWithOptions:
(NSDictionary*)launchOptions {
  FlutterViewController* controller = (FlutterViewController*)self.window.rootViewController;

  FlutterMethodChannel* batteryChannel = [FlutterMethodChannel
                                          methodChannelWithName:@"samples.flutter.dev/battery"
                                          binaryMessenger:controller.binaryMessenger];

  [batteryChannel setMethodCallHandler:^(FlutterMethodCall* call, FlutterResult result) {
    // Note: this method is invoked on the UI thread.
    // TODO
  }];

  [GeneratedPluginRegistrant registerWithRegistry:self];
  return [super application:application didFinishLaunchingWithOptions:launchOptions];
}
```

2. Decoding Flutter (ObjC)

https://flutter.dev/docs/development/platform-integration/platform-channels

```
#import <Flutter/Flutter.h>
#import "GeneratedPluginRegistrant.h"                  - = instance method; + = class method

@implementation AppDelegate
- (BOOL)application:(UIApplication*)application didFinishLaunchingWithOptions:
(NSDictionary*)launchOptions {
  FlutterViewController* controller = (FlutterViewController*)self.window.rootViewController;

  FlutterMethodChannel* batteryChannel = [FlutterMethodChannel
                                          methodChannelWithName:@"samples.flutter.dev/battery"
                                          binaryMessenger:controller.binaryMessenger];

  [batteryChannel setMethodCallHandler:^(FlutterMethodCall* call, FlutterResult result) {
    // Note: this method is invoked on the UI thread.
    // TODO
  }];

  [GeneratedPluginRegistrant registerWithRegistry:self];
  return [super application:application didFinishLaunchingWithOptions:launchOptions];
}
```

2. Decoding Flutter (ObjC)

https://flutter.dev/docs/development/platform-integration/platform-channels

defines method **application:didFinishLaunchingWithOptions:**
returning type **BOOL** (values **Yes** or **No**)

```
#import <Flutter/Flutter.h>
#import "GeneratedPluginRegistrant.h"

@implementation AppDelegate
- (BOOL)application:(UIApplication*)application didFinishLaunchingWithOptions:
(NSDictionary*)launchOptions {
  FlutterViewController* controller = (FlutterViewController*)self.window.rootViewController;

  FlutterMethodChannel* batteryChannel = [FlutterMethodChannel
                                          methodChannelWithName:@"samples.flutter.dev/battery"
                                          binaryMessenger:controller.binaryMessenger];

  [batteryChannel setMethodCallHandler:^(FlutterMethodCall* call, FlutterResult result) {
    // Note: this method is invoked on the UI thread.
    // TODO
  }];

  [GeneratedPluginRegistrant registerWithRegistry:self];
  return [super application:application didFinishLaunchingWithOptions:launchOptions];
}
```

2. Decoding Flutter (ObjC)

https://flutter.dev/docs/development/platform-integration/platform-channels

```
uses dot notation
same as [[self window] rootViewController]
    and [controller binaryMessenger]
```

```
#import <Flutter/Flutter.h>
#import "GeneratedPluginRegistrant.h"

@implementation AppDelegate
- (BOOL)application:(UIApplication*)application didFinishLaunchingWithOptions:
(NSDictionary*)launchOptions {
  FlutterViewController* controller = (FlutterViewController*)self.window.rootViewController;

  FlutterMethodChannel* batteryChannel = [FlutterMethodChannel
                                          methodChannelWithName:@"samples.flutter.dev/battery"
                                          binaryMessenger:controller.binaryMessenger];

  [batteryChannel setMethodCallHandler:^(FlutterMethodCall* call, FlutterResult result) {
    // Note: this method is invoked on the UI thread.
    // TODO
  }];

  [GeneratedPluginRegistrant registerWithRegistry:self];
  return [super application:application didFinishLaunchingWithOptions:launchOptions];
}
```

2. Decoding Flutter (ObjC)

https://flutter.dev/docs/development/platform-integration/platform-channels

```
#import <Flutter/Flutter.h>
#import "GeneratedPluginRegistrant.h"
```

```
@"..." creates NSString object
"..." would create nil-terminated C string
```

```
@implementation AppDelegate
- (BOOL)application:(UIApplication*)application didFinishLaunchingWithOptions:
(NSDictionary*)launchOptions {
  FlutterViewController* controller = (FlutterViewController*)self.window.rootViewController;

  FlutterMethodChannel* batteryChannel = [FlutterMethodChannel
                                          methodChannelWithName:@"samples.flutter.dev/battery"
                                          binaryMessenger:controller.binaryMessenger];

  [batteryChannel setMethodCallHandler:^(FlutterMethodCall* call, FlutterResult result) {
    // Note: this method is invoked on the UI thread.
    // TODO
  }];

  [GeneratedPluginRegistrant registerWithRegistry:self];
  return [super application:application didFinishLaunchingWithOptions:launchOptions];
}
```

2. Decoding Flutter (ObjC)

https://flutter.dev/docs/development/platform-integration/platform-channels

^(args) {code} defines a "block" that can be passed around like any other object

```
#import <Flutter/Flutter.h>
#import "GeneratedPluginRegistrant.h"

@implementation AppDelegate
- (BOOL)application:(UIApplication*)application didFinishLaunchingWithOptions:
(NSDictionary*)launchOptions {
  FlutterViewController* controller = (FlutterViewController*)self.window.rootViewController;

  FlutterMethodChannel* batteryChannel = [FlutterMethodChannel
                                          methodChannelWithName:@"samples.flutter.dev/battery"
                                          binaryMessenger:controller.binaryMessenger];

  [batteryChannel setMethodCallHandler:^(FlutterMethodCall* call, FlutterResult result) {
    // Note: this method is invoked on the UI thread.
    // TODO
  }];

  [GeneratedPluginRegistrant registerWithRegistry:self];
  return [super application:application didFinishLaunchingWithOptions:launchOptions];
}
```

2. Decoding Flutter (ObjC)

https://flutter.dev/docs/development/platform-integration/platform-channels

self refers to the instance, like **this** in Java

```
#import <Flutter/Flutter.h>
#import "GeneratedPluginRegistrant.h"

@implementation AppDelegate
- (BOOL)application:(UIApplication*)application didFinishLaunchingWithOptions:
(NSDictionary*)launchOptions {
  FlutterViewController* controller = (FlutterViewController*)self.window.rootViewController;

  FlutterMethodChannel* batteryChannel = [FlutterMethodChannel
                                          methodChannelWithName:@"samples.flutter.dev/battery"
                                          binaryMessenger:controller.binaryMessenger];

  [batteryChannel setMethodCallHandler:^(FlutterMethodCall* call, FlutterResult result) {
    // Note: this method is invoked on the UI thread.
    // TODO
  }];

  [GeneratedPluginRegistrant registerWithRegistry:self];
  return [super application:application didFinishLaunchingWithOptions:launchOptions];
}
```

2. Decoding Flutter (ObjC)

https://flutter.dev/docs/development/platform-integration/platform-channels

```
#import <Flutter/Flutter.h>
#import "GeneratedPluginRegistrant.h"

@implementation AppDelegate
- (BOOL)application:(UIApplication*)application didFinishLaunchingWithOptions:
(NSDictionary*)launchOptions {
  FlutterViewController* controller = (FlutterViewController*)self.window.rootViewController;

  FlutterMethodChannel* batteryChannel = [FlutterMethodChannel
                                          methodChannelWithName:@"samples.flutter.dev/battery"
                                          binaryMessenger:controller.binaryMessenger];

  [batteryChannel setMethodCallHandler:^(FlutterMethodCall* call, FlutterResult result) {
    // Note: this method is invoked on the UI thread.
    // TODO
  }];

  [GeneratedPluginRegistrant registerWithRegistry:self];
  return [super application:application didFinishLaunchingWithOptions:launchOptions];
}
```

3. Brief History (Swift)

Swift introduced in 2014; ABI stable in 2019

Chris Lattner was initial author; left Apple in 2017

"Objective-C without the C"

Open source (swift.org)

Three stated goals: Safe, Fast, Expressive

Apple uses for all platforms: iOS, iPadOS, macOS, watchOS, tvOS

Also available for server-side apps (Kitura, Vapor, others)

Early-stage "Swift for TensorFlow" (tensorflow.org/swift)

https://developer.apple.com/swift

https://hackingWithSwift.com

https://whatsNewInSwift.com

4. Swift basics (types)

Basic types: Int, Float, Double, Bool (true, false), String (also Void, Any, AnyObject)

Swift supports "class" for reference type, "struct" for value type; prefers "struct"

Assignment: "var" for mutable, "let" for immutable

```
    let success: Bool = true
```

Swift reduces visual clutter through extensive code inference.

Since we're assigning "true", a boolean, the constant is inferred to be boolean as well.

```
    let success = true
```

C increment (++) and decrement (--) operators are *not* supported

```
    ++count      // no...but nice try
    count += 1   // nice comeback
```

4. Swift basics (conversion)

Swift does *not* automatically convert types:

```
let name = "Bob"
let age = 39

NO:  let desc = name + " is " + age
OK:  let desc = name + " is " + String(age)
YES: let desc = "\(name) is \(age)"
```

Uses "String interpolation"

4. Swift basics (Optionals)

Swift supports Optionals:

```
var maybeAgeVerbose: Optional<Int>
var maybeAge: Int?
if let maybeAge = maybeAge {
    print("age is \(maybeAge)")
}
```

```
// or...
if maybeAge != nil {
    print("age is \(maybeAge!)")
}
```

Nil coalescing:

```
var i1: Int?
var i2: Int? = 2
let i = i1 ?? i2 ?? -1
print(i)
```

Optional chaining:

```
car.engine?.cylinders = 8
let model = car.make!.model
```

4. Swift basics (collections)

Arrays:

```
let carMakes1: Array<String> = ["Ford", "Fiat", "Dodge"]
let carMakes2: [String] = ["Ford", "Fiat", "Dodge"]
let carMakes3 = ["Ford", "Fiat", "Dodge"]
var carMakesEmpty = [String]() //or : [String] = []
```

4. Swift basics (collections)

Arrays:

```
let carMakes1: Array<String> = ["Ford", "Fiat", "Dodge"]
let carMakes2: [String] = ["Ford", "Fiat", "Dodge"]
let carMakes3 = ["Ford", "Fiat", "Dodge"]
var carMakesEmpty = [String]() //or : [String] = []
```

Dictionaries:

```
let lunchOrders: Dictionary<String, String> =
    ["Betty":"BLT", "Peter":"Pizza"]
let lunchOrders: [String: String] =
    ["Betty":"BLT", "Peter":"Pizza"]
let lunchOrders = ["Betty":"BLT", "Peter":"Pizza"]
var lunchOrdersEmpty = [String:String]()
```

4. Swift basics (collections)

Arrays:

```
let carMakes1: Array<String> = ["Ford", "Fiat", "Dodge"]
let carMakes2: [String] = ["Ford", "Fiat", "Dodge"]
let carMakes3 = ["Ford", "Fiat", "Dodge"]
var carMakesEmpty = [String]() //or : [String] = []
```

Dictionaries:

```
let lunchOrders: Dictionary<String, String> =
    ["Betty":"BLT", "Peter":"Pizza"]
let lunchOrders: [String: String] =
    ["Betty":"BLT", "Peter":"Pizza"]
let lunchOrders = ["Betty":"BLT", "Peter":"Pizza"]
var lunchOrdersEmpty = [String:String]()
```

Sets:

```
let carMakesSet: Set<String> = ["Ford", "Fiat", "Dodge"]
var carMakesEmptySet = Set<String>()
```

4. Swift basics (collections)

Arrays:

```
let carMakes1: Array<String> = ["Ford", "Fiat", "Dodge"]
let carMakes2: [String] = ["Ford", "Fiat", "Dodge"]
let carMakes3 = ["Ford", "Fiat", "Dodge"]
var carMakesEmpty = [String]() //or : [String] = []
```

Dictionaries:

```
let lunchOrders: Dictionary<String, String> =
     ["Betty":"BLT", "Peter":"Pizza"]
let lunchOrders: [String: String] =
     ["Betty":"BLT", "Peter":"Pizza"]
let lunchOrders = ["Betty":"BLT", "Peter":"Pizza"]
var lunchOrdersEmpty = [String:String]()
```

Sets:

```
let carMakesSet: Set<String> = ["Ford", "Fiat", "Dodge"]
var carMakesEmptySet = Set<String>()
```

4. Swift basics (loops)

`while {}` and **`repeat {} while`** are as usual, including **`break`** and **`continue`**.

`for`, however...

```
for i in 1...10 { print(i) }
for i in 1...10 where i % 3 == 0 { print(i) }

for i in stride(from:0, to:10, by:2) { print(i) } //exclude 10
for i in stride(from:0, through:10, by:2) {print(i)} //incl 10

let names = ["John", "Paul", "George", "Ringo"]
for name in names { print(name) }
for (ndx,name) in names.enumerated() { print("\(ndx): \(name)")}

let band = ["Paul":"bass", "George":"guitar", "Ringo":"drums"]
for (name,inst) in band { print("\(name) played \(inst)" }
```

4. Swift basics (functions)

Three styles of method signatures:

```
1. plain: p(x:)
      func p(x:Int)         {print(x)} //called as p(x:3)

2. "special" label: p(_:)
      func p(_ x:Int)       {print(x)} //called as p(3)

3. with label: p(count:)
      func p(count x:Int) {print(x)} //called as p(count:3)
```

Return types indicated by "->"

```
      func squared(_ x: Int) -> Int { return x * x }
      print(squared(3))
```

4. Swift basics (closures)

Functions/closures are first-class citizens that may be passed around

```
func applyInt(label: String, value: Int, operation: (Int) -> Int) {
    print("\(label) \(value) yields \(operation(value))")
}
func squared(_ x: Int) -> Int { return x * x } //print(squared(2))
applyInt(label: "squaring", value: 2, operation: squared(_:))
```

4. Swift basics (closures)

Functions/closures are first-class citizens that may be passed around

```
func applyInt(label: String, value: Int, operation: (Int) -> Int) {
    print("\(label) \(value) yields \(operation(value))")
}
func squared(_ x: Int) -> Int { return x * x } //print(squared(2))
applyInt(label: "squaring", value: 2, operation: squared(_:))

applyInt(label: "squaring", value: 2, operation: { (x: Int) in
    return x * x
})
```

4. Swift basics (closures)

Functions/closures are first-class citizens that may be passed around

```
func applyInt(label: String, value: Int, operation: (Int) -> Int) {
    print("\(label) \(value) yields \(operation(value))")
}
func squared(_ x: Int) -> Int { return x * x } //print(squared(2))
applyInt(label: "squaring", value: 2, operation: squared(_:))

applyInt(label: "squaring", value: 2, operation: { (x: Int) in
    return x * x
})
applyInt(label: "squaring", value: 2) { x in   //trailing closure
    x * x
}
```

4. Swift basics (closures)

Functions/closures are first-class citizens that may be passed around

```
func applyInt(label: String, value: Int, operation: (Int) -> Int) {
    print("\(label) \(value) yields \(operation(value))")
}
func squared(_ x: Int) -> Int { return x * x } //print(squared(2))
applyInt(label: "squaring", value: 2, operation: squared(_:))

applyInt(label: "squaring", value: 2, operation: { (x: Int) in
    return x * x
})
applyInt(label: "squaring", value: 2) { x in   //trailing closure
    x * x
}
applyInt(label: "squaring", value: 2) { $0 * $0 }
```

4. Swift basics (closures)

Functions/closures are first-class citizens that may be passed around

```
func applyInt(label: String, value: Int, operation: (Int) -> Int) {
    print("\(label) \(value) yields \(operation(value))")
}
func squared(_ x: Int) -> Int { return x * x } //print(squared(2))
applyInt(label: "squaring", value: 2, operation: squared(_:))

applyInt(label: "squaring", value: 2, operation: { (x: Int) in
    return x * x
})
applyInt(label: "squaring", value: 2) { x in   //trailing closure
    x * x
})
applyInt(label: "squaring", value: 2) { $0 * $0 }
```

5. Decoding Flutter (Swift)

https://flutter.dev/docs/development/platform-integration/platform-channels

```
@UIApplicationMain
@objc class AppDelegate: FlutterAppDelegate {
  override func application(
    _ application: UIApplication,
    didFinishLaunchingWithOptions launchOptions: [UIApplication.LaunchOptionsKey: Any]?) -> Bool {

    let controller : FlutterViewController = window?.rootViewController as! FlutterViewController
    let batteryChannel = FlutterMethodChannel(name: "samples.flutter.dev/battery",
                                              binaryMessenger: controller.binaryMessenger)
    batteryChannel.setMethodCallHandler({
      (call: FlutterMethodCall, result: @escaping FlutterResult) -> Void in
      // Note: this method is invoked on the UI thread.
      // Handle battery messages.
    })

    GeneratedPluginRegistrant.register(with: self)
    return super.application(application, didFinishLaunchingWithOptions: launchOptions)
  }
}
```

5. Decoding Flutter (Swift)

https://flutter.dev/docs/development/platform-integration/platform-channels

```
@UIApplicationMain
@objc class AppDelegate: FlutterAppDelegate {     inherit from parent class FlutterAppDelegate
  override func application(
    _ application: UIApplication,
    didFinishLaunchingWithOptions launchOptions: [UIApplication.LaunchOptionsKey: Any]?) -> Bool {

    let controller : FlutterViewController = window?.rootViewController as! FlutterViewController
    let batteryChannel = FlutterMethodChannel(name: "samples.flutter.dev/battery",
                                              binaryMessenger: controller.binaryMessenger)
    batteryChannel.setMethodCallHandler({
      (call: FlutterMethodCall, result: @escaping FlutterResult) -> Void in
      // Note: this method is invoked on the UI thread.
      // Handle battery messages.
    })

    GeneratedPluginRegistrant.register(with: self)
    return super.application(application, didFinishLaunchingWithOptions: launchOptions)
  }
}
```

5. Decoding Flutter (Swift)

https://flutter.dev/docs/development/platform-integration/platform-channels

Must declare when overriding parent method

```
@UIApplicationMain
@objc class AppDelegate: FlutterAppDelegate {
  override func application(
    _ application: UIApplication,
    didFinishLaunchingWithOptions launchOptions: [UIApplication.LaunchOptionsKey: Any]?) -> Bool {

    let controller : FlutterViewController = window?.rootViewController as! FlutterViewController
    let batteryChannel = FlutterMethodChannel(name: "samples.flutter.dev/battery",
                                              binaryMessenger: controller.binaryMessenger)
    batteryChannel.setMethodCallHandler({
      (call: FlutterMethodCall, result: @escaping FlutterResult) -> Void in
      // Note: this method is invoked on the UI thread.
      // Handle battery messages.
    })

    GeneratedPluginRegistrant.register(with: self)
    return super.application(application, didFinishLaunchingWithOptions: launchOptions)
  }
}
```

5. Decoding Flutter (Swift)

https://flutter.dev/docs/development/platform-integration/platform-channels

defines method
  application(_:didFinishLaunchingWithOptions:)
returning type **Bool** (values **true** or **false**)

```
@UIApplicationMain
@objc class AppDelegate: FlutterAppDelegate {
  override func application(
    _ application: UIApplication,
    didFinishLaunchingWithOptions launchOptions: [UIApplication.LaunchOptionsKey: Any]?) -> Bool {

    let controller : FlutterViewController = window?.rootViewController as! FlutterViewController
    let batteryChannel = FlutterMethodChannel(name: "samples.flutter.dev/battery",
                                              binaryMessenger: controller.binaryMessenger)
    batteryChannel.setMethodCallHandler({
      (call: FlutterMethodCall, result: @escaping FlutterResult) -> Void in
      // Note: this method is invoked on the UI thread.
      // Handle battery messages.
    })

    GeneratedPluginRegistrant.register(with: self)
    return super.application(application, didFinishLaunchingWithOptions: launchOptions)
  }
}
```

5. Decoding Flutter (Swift)

https://flutter.dev/docs/development/platform-integration/platform-channels

Optional dictionary
"Any" like Object in Java

```
@UIApplicationMain
@objc class AppDelegate: FlutterAppDelegate {
  override func application(
    _ application: UIApplication,
    didFinishLaunchingWithOptions launchOptions: [UIApplication.LaunchOptionsKey: Any]?) -> Bool {

    let controller : FlutterViewController = window?.rootViewController as! FlutterViewController
    let batteryChannel = FlutterMethodChannel(name: "samples.flutter.dev/battery",
                                              binaryMessenger: controller.binaryMessenger)
    batteryChannel.setMethodCallHandler({
      (call: FlutterMethodCall, result: @escaping FlutterResult) -> Void in
      // Note: this method is invoked on the UI thread.
      // Handle battery messages.
    })

    GeneratedPluginRegistrant.register(with: self)
    return super.application(application, didFinishLaunchingWithOptions: launchOptions)
  }
}
```

5. Decoding Flutter (Swift)

https://flutter.dev/docs/development/platform-integration/platform-channels

```
as? = "I think this can be cast"
as! = cast...and crash if it fails
```

```
@UIApplicationMain
@objc class AppDelegate: FlutterAppDelegate {
  override func application(
    _ application: UIApplication,
    didFinishLaunchingWithOptions launchOptions: [UIApplication.LaunchOptionsKey: Any]?) -> Bool {

    let controller : FlutterViewController = window?.rootViewController as! FlutterViewController
    let batteryChannel = FlutterMethodChannel(name: "samples.flutter.dev/battery",
                                              binaryMessenger: controller.binaryMessenger)
    batteryChannel.setMethodCallHandler({
      (call: FlutterMethodCall, result: @escaping FlutterResult) -> Void in
      // Note: this method is invoked on the UI thread.
      // Handle battery messages.
    })

    GeneratedPluginRegistrant.register(with: self)
    return super.application(application, didFinishLaunchingWithOptions: launchOptions)
  }
}
```

5. Decoding Flutter (Swift)

https://flutter.dev/docs/development/platform-integration/platform-channels

Pass closure taking two parameters, returning Void

```
@UIApplicationMain
@objc class AppDelegate: FlutterAppDelegate {
  override func application(
    _ application: UIApplication,
    didFinishLaunchingWithOptions launchOptions: [UIApplication.LaunchOptionsKey: Any]?) -> Bool {

    let controller : FlutterViewController = window?.rootViewController as! FlutterViewController
    let batteryChannel = FlutterMethodChannel(name: "samples.flutter.dev/battery",
                                              binaryMessenger: controller.binaryMessenger)
    batteryChannel.setMethodCallHandler({
      (call: FlutterMethodCall, result: @escaping FlutterResult) -> Void in
      // Note: this method is invoked on the UI thread.
      // Handle battery messages.
    })

    GeneratedPluginRegistrant.register(with: self)
    return super.application(application, didFinishLaunchingWithOptions: launchOptions)
  }
}
```

5. Decoding Flutter (Swift)

https://flutter.dev/docs/development/platform-integration/platform-channels

Closure named by FlutterResult
may be called OUTSIDE of this method/closure

```
@UIApplicationMain
@objc class AppDelegate: FlutterAppDelegate {
  override func application(
    _ application: UIApplication,
    didFinishLaunchingWithOptions launchOptions: [UIApplication.LaunchOptionsKey: Any]?) -> Bool {

    let controller : FlutterViewController = window?.rootViewController as! FlutterViewController
    let batteryChannel = FlutterMethodChannel(name: "samples.flutter.dev/battery",
                                              binaryMessenger: controller.binaryMessenger)
    batteryChannel.setMethodCallHandler({
      (call: FlutterMethodCall, result: @escaping FlutterResult) -> Void in
      // Note: this method is invoked on the UI thread.
      // Handle battery messages.
    })

    GeneratedPluginRegistrant.register(with: self)
    return super.application(application, didFinishLaunchingWithOptions: launchOptions)
  }
}
```

5. Decoding Flutter (Swift)

https://flutter.dev/docs/development/platform-integration/platform-channels

self refers to the instance, like **this** in Java

```
@UIApplicationMain
@objc class AppDelegate: FlutterAppDelegate {
  override func application(
    _ application: UIApplication,
    didFinishLaunchingWithOptions launchOptions: [UIApplication.LaunchOptionsKey: Any]?) -> Bool {

    let controller : FlutterViewController = window?.rootViewController as! FlutterViewController
    let batteryChannel = FlutterMethodChannel(name: "samples.flutter.dev/battery",
                                              binaryMessenger: controller.binaryMessenger)
    batteryChannel.setMethodCallHandler({
      (call: FlutterMethodCall, result: @escaping FlutterResult) -> Void in
      // Note: this method is invoked on the UI thread.
      // Handle battery messages.
    })

    GeneratedPluginRegistrant.register(with: self)
    return super.application(application, didFinishLaunchingWithOptions: launchOptions)
  }
}
```

5. Decoding Flutter (Swift)

https://flutter.dev/docs/development/platform-integration/platform-channels

```swift
@UIApplicationMain
@objc class AppDelegate: FlutterAppDelegate {
  override func application(
    _ application: UIApplication,
    didFinishLaunchingWithOptions launchOptions: [UIApplication.LaunchOptionsKey: Any]?) -> Bool {

    let controller : FlutterViewController = window?.rootViewController as! FlutterViewController
    let batteryChannel = FlutterMethodChannel(name: "samples.flutter.dev/battery",
                                              binaryMessenger: controller.binaryMessenger)
    batteryChannel.setMethodCallHandler({
      (call: FlutterMethodCall, result: @escaping FlutterResult) -> Void in
      // Note: this method is invoked on the UI thread.
      // Handle battery messages.
    })

    GeneratedPluginRegistrant.register(with: self)
    return super.application(application, didFinishLaunchingWithOptions: launchOptions)
  }
}
```

6. Beyond the basics (fun!)

Basics are fine and necessary, but there's so much more:

a. guard!

b. switch! (no, really -- it *is* exciting!)

c. Tuples!

d. Enums!

e. Structs!

f. Computed Properties!

g. "Closure Properties"!

h. Extensions!

i. Protocols! (okay, maybe not so exciting...yet)

j. Protocols + Extensions!

6a. guard!

```
// guard can appear anywhere "if" can, but its intent is more clear
// guard must have an "else" which leaves the guard's code block (return, break, etc.)
// vars/constants assigned in guard are available afterwards in the same parent code block

struct Engine {  // more on structs in (6e)
    let numCylinders: Int?
    let isElectric: Bool
}
let bigEngine = Engine( numCylinders: 12, isElectric: false )

func rev(engine: Engine?) -> Void {
    guard let engine = engine,
          let numCylinders = engine.numCylinders,
          numCylinders > 4
    else {
        print("...crickets...")
        return
    }

    print("Hear the mighty roar of all \(numCylinders) cylinders! VROOM!")
}
rev(engine: bigEngine)
```

6b. switch!

```
// does NOT fall through by default (use "fallthrough" for that)
// can use ranges
// can check more than one value simultaneously
// must be exhaustive

let count = 3
switch count {
    case 0..<5: print("less than 5")
    case 5...10: print("5 to 10")
    default: print("too much!")
}

if case 5...10 = count { print("5 to 10") }

let largeCountOkay = true
switch (count, largeCountOkay) {
    case (0..<5, _): print("less than 5")
    case (5...10, _): print("5 to 10")
    case (11...100, true): print("that's a lot!")
    default: print("too much!")
}

// More to come!
```

6c. Tuples!

```
var tuple1 = ("Bob", 39)
print("\(tuple1.0) is \(tuple1.1)")

var tuple2 = (name: "Bob", age: 39)
print("\(tuple2.name) is \(tuple2.age)")

typealias NameAge = (name: String, age: Int)
var tuple3: NameAge = (name: "Bob", age: 39)

func randomPerson() -> NameAge {
    let names = ["Gilligan", "Skipper", "Professor", "Ginger", "Mary Ann"]
    let ages = 5 ... 85
    return (name: names.randomElement()!, age: ages.randomElement()!)
}
for _ in 0 ..< 10 {
    print(randomPerson())
}

func changeAge(for person: inout NameAge) { person.age += 10 }
var person = randomPerson()
changeAge(for: &person)

10.quotientAndRemainder(dividingBy: 3)  //yields TUPLE (quotient: 3, remainder: 1)
```

6c. Tuples! (cont'd)

```
var tuple1 = ("Bob", 39)
print("\(tuple1.0) is \(tuple1.1)")

switch tuple1 {
    case (let name, 0...2): print("\(name) is a baby!")
    case (let name, 3...12): print("\(name) is sure growing fast!")
    default: print("I can't believe \(tuple1.0) is \(tuple1.1)")
}

let point = (0.25, 1.3)
switch point {
    case (0.0, let r), (let r, 0.0): print("point on an axis, \(r) away from the origin")
    case (-1...1, -1...1): print("\(point) is within one unit of the origin")
    default: print("\(point) is far out")
}

// still more switch to come!
```

6d. Enums!

```swift
enum Shape {
    case square, circle
}
let square1 = Shape.square, square2: Shape = .square

// enum can have a type
enum Shape: Int {
    case square = 0
    case circle
}
let circle1 = Shape(rawValue: 1)  //circle1 is of type "Shape?"

enum LightBulb {
    case on, off

    mutating func toggle() {
        switch self {
            case .on: self = .off
            case .off: self = .on
        }
    }
}
var bulb = LightBulb.on; bulb.toggle()
```

6d. Enums! (cont'd)

```
// "associated values"
enum Shape {
    case square(Double)  //length of side
    case rectangle(height:Double, width:Double)

    func area() -> Double {
        switch self {
            case .square(let side): return side * side
            case let .rectangle(height: h, width: w): return h * w
        }
    }
}

let rect = Shape.rectangle(height: 2.0, width: 4.0)
print( "\(rect) has area \(rect.area())" )

// to pattern-match one condition, use if case let
if case let Shape.rectangle(h,w) = rect, w == 2.0 * h {
    print("\(rect) has a width twice its height!")
}
```

6e. Structs!

```
// structs are VALUE types; classes are REFERENCE types
// structs include "automatically generated memberwise initializer" (unless explicit initializer)
// classes support inheritance (but not multiple inheritance); structs do not
// functions which change properties must be labelled "mutating" in struct (not in class)
// Swift "encourages" structs over classes for safety (String is a struct!)

struct Box {
    let height, width, depth: Double

    func volume() -> Double { height * width * depth }
    mutating func grow() { height *= 2; width *= 2; depth *= 2 }
}
let box = Box(height: 1, width: 1, depth: 1)

class Flexibox {
    var height, width, depth: Double
    init( height h: Double, width w: Double, depth d: Double ) {
        height = h; width = w; depth = d
    }
    func volume() -> Double { height * width * depth }
    func grow() { height *= 2; width *= 2; depth *= 2 }
}
var fbox = Flexibox(height: 1, width: 1, depth: 1)
```

6f. Computed Properties!

```
struct Box {
    let height, width, depth: Double

    ~~func volume() -> Double { height * width * depth }~~
    var volume: Double { height * width * depth }
}
let box = Box(height: 1, width: 1, depth: 1)
print( "box volume is \(box.volume)" )

struct Cube {
    var side: Double
    var volume: Double {
        get { side * side * side }
        // set(newVolume) { side = pow(newVolume, 1.0/3.0) }
        set { side = pow(newValue, 1.0/3.0) }
    }
}
var cube = Cube(side: 1.0)
print("cube with side \(cube.side) has volume \(cube.volume)")

cube.volume = 8.0
print("cube with volume \(cube.volume) has side \(cube.side)")
```

6g. "Closure Properties"!

```
// Properties can be defined using closures as well!

// Instead of:
let dateFormatter: DateFormatter
...
init() {
    dateFormatter = DateFormatter()
    dateFormatter.dateStyle = .medium
    dateFormatter.timeStyle = .none
}

// Use a closure!
let dateFormatter: DateFormatter = {
    let formatter = DateFormatter()
    formatter.dateStyle = .medium
    formatter.timeStyle = .none
    return formatter
}()

// Note the signature: () -> DateFormatter
```

6h. Extensions!

```
// extensions cannot have stored properties
// extensions also work on types for which YOU DO NOT HAVE THE SOURCE!!
// extensions allow structs to keep their auto-generated memberwise initializer

struct Box {
    var height, width, depth: Double
}

extension Box {
    var volume: Double { height * width * depth }

    init(withAllSides side: Double) { self.init(height: side, width: side, depth: side) }

    mutating func expand() { height *= 2; width *= 2; depth *= 2 }
}

let box = Box( height: 1, width: 1, depth: 1 )
print("box has volume \(box.volume)")

let cube = Box( withAllSides: 1 )
```

6i. Protocols!

```swift
// a protocol is just an interface declaration
// structs, classes, and enums can all implement protocols

protocol Shape {}
protocol TwoD: Shape { var area: Double { get } }
protocol ThreeD: Shape { var volume: Double { get } }
protocol Expandable { mutating func expand() }

struct Square: TwoD, Expandable {
    var side: Double
    var area: Double { side * side }
    mutating func expand() { side *= 2 }
}

class Cube: ThreeD, Expandable {
    var edge: Double = 1
    var volume: Double { edge * edge * edge }
    func expand() { edge *= 2 }
}

// to declare a protocol only for classes, inherit from "AnyObject"
protocol ClassOnlyProtocol: AnyObject {}
```

6j. Protocols + Extensions!

```
protocol Shape: CustomStringConvertible {}
extension Shape { var description: String { "some " + String(describing: type(of: self)) }


protocol TwoD: Shape { var area: Double { get } }
protocol ThreeD: Shape { var volume: Double { get } }
protocol Expandable { mutating func expand() }

struct Square: TwoD, Expandable {
    var side: Double
    var area: Double { side * side }
    mutating func expand() { side *= 2 }
}
extension Square: TwoD { var area: Double { side * side } }
extension Square: Expandable { mutating func expand() { side *= 2 } }
extension Square: CustomStringConvertible { var description: String { "Square with sides \(side)" }}

let shapes: [Shape] = [ Square(1.0), Cube() ]
for shape in shapes { print(shape) }
// -- prints: --
// Square with sides 1.0
// some Cube
```

SO much more...

more options to `switch` (`case is, case...as, @unknown default:`)

higher-order functions (`chequeArray.reduce(0) { $0 + $1.amount }`)

`guard case let`

`for case let, for case let...where`

property observers (`willSet, didSet`)

generics

threading

exception handling (`try?/try!/try/catch`)

optional/default parameters with `func`

built-in support for JSON (`Codable` protocol)

custom operators (`static func + (lhs: Vector, rhs: Vector) -> Vector {...}`)

...and more