Unleashing *Swift*


@iwantmyrealname

Choose

your own

Adventure

swift

fast

modern

safe

interactive

Strong typing

Mutability

Value Types

ARC

Some code

```
var arr = [3, 4, 1, 6, 2, 5];

arr.sort();

arr; //= ??
```

What's the langauge?

and the result?

Javascript

```
var arr = [3, 4, 1, 6, 2, 5];

arr.sort();

arr; //= [1, 2, 3, 4, 5, 6]
```

Swift

```
var arr = [3, 4, 1, 6, 2, 5];

arr.sort();

arr; //= [3, 4, 1, 6, 2, 5]
```

getting started

Getting Started

— Mutability
— Value types
— Classes / Structs
— Functions
— Extensions
— Closures

Mutability

```
let twelve = 12
//twelve = 13

var thirteen = 13
thirteen = 12
//thirteen = 13.0
```

Value Types

```
var title = "This is a string"
var secondTitle = title
secondTitle += ", extended"
print(title)       // "This is a string"
print(secondTitle) // "This is a string, extended"
```

Classes

— Reference types

— Inheritance / Dynamic Dispatch

```
class Vehicle {
  var maxSpeed: Double
}

class Car: Vehicle {
  var numberOfWheels: Int
}
```

Structs

— Value types

— No inheritance

— Immutable by default

```
struct Point3D {
  let x: Double
  let y: Double
  let x: Double
}
```

Class Initialisation

1. All stored properties
2. super.init() / self.init()
3. Change inherited properties
4. Call instance methods / access `self`

Functions

```
func square(value: Int) -> Int {
  return value * value
}

func exp(value: Int, power: Int) -> Int {
  return power > 0 ? value * exp(value, power: power-1) : 1
}

func curryPower(power: Int)(_ value: Int) -> Int {
  return exponentiate(value, power: power)
}
let cube = curryPower(3)
cube(3) //= 27
```

Extensions

— Add functionality to types

```
extension Int {
  func toThePower(power: Int) -> Int {
    return exponentiate(self, power: power)
  }
}

2.toThePower(5)
```

Closures

```
typealias Callback = (reply: String) -> ()

func asyncHello(name: String, callback: Callback) {
  callback(reply: "Hello \(name)")
}
```

Functional Tinge

— Functions are 1st class

— `map`, `reduce`, `filter` on `CollectionType`

— TLO implemented

```
let scores = [1, 2, 4, 5, 3, 5]
let cubedScores = scores.map { cube($0) }
                //= [1, 8, 64, 125, 27, 125]
```

"modern"
concepts

Modern Concepts

— Enums
— Pattern Matching
— Optionals
— Protocols

Enums

— Raw values

— Associated values

— Equivalent to mapping | |

— Recursive enums

```
enum Activity: String {
   case Running
   case Swimming
   case Cycling
}
```

Pattern Matching

— Concept from functional languages

— Part of `if` and `switch`

```
func evaluate(expresion: ArithmeticExpression) -> Int {
  switch expresion {
  case .Number(let value):
    return value
  case .Addition(let left, let right):
    return evaluate(left) + evaluate(right)
  case .Multiplication(let left, let right):
    return evaluate(left) * evaluate(right)
  }
}
```

Optionals

— Handling of nil

— Just an enum with some syntactic sugar

```
enum Optional<Wrapped> {
   case None
   case Some(Wrapped)
}
```

?!??

Protocols

The problems with classes:

— Inheritance
— Implicit sharing
— Lose type relationships

Protocols

(a.k.a. Interfaces)

- Supports value types
- Can add functionality to existing types
- No instance data available
- Can include default implementations

Other Stuff

Other Stuff

— Interoperability
— Tuples
— Generics
— Error Handling
— Playgrounds

the future

Open source

how can I play?

buy a

proper computer

or wait

you become a **better programmer** by **playing** with other lanugages

— Me, just now

go forth and **play**

@iwantmyrealname

github.com/sammyd