Swift flashcards
A guide to success at cocktail parties

Sash Zats - zats.io - @zats - September 8, Tel Aviv

```
[self helloAgain:AAPLAudienceTypeSwiftLovers];
```

```
[self helloAgain:AudienceTypeSwiftLovers];
```

```
self.helloAgain(AudienceTypeSwiftLovers)
```

```
self.helloAgain(.SwiftLovers)
```

Design

- Small runtime (~5Mb? Pff), deployable to the previous OSes
- Statically compiled, no JIT, garbage collection… duh…
- No legacy, no abstraction penalties: `Int`s & `Float`s are `struct`s
- ARC
- Multithreading built into language (aka `atomic`). `NO`!

KVO

KVO-ish

KVO-ish

Take action while property setter is in flight

```
var pet: Pet {
    willSet (newPet) {
        // cleanup
    }
    didSet {
        // awesome
    }
}
```

Functions overloading

Same function name, different signatures

```
func add(a: Int, b: Int) -> Int

func add(a: String, b: String) -> String
```

Operator overloading


Operator?

Typed collections

```
var bag: Cat[] = [ snowball, oliver, jasper ]
bag.append(scoobyDoo) // compiler error

var mapping: [Bool: String] = Dictionary()
mapping[true] = "true"
mapping[true]!.utf8 // don't forget to unwrap
mapping[false] = 1 // compiler error
```

Optional chaining

I don't always unwrap, but when I do…

```
let y: SomeClass? = nil
let z = y?.someMethod() // will produce nil
```

Mutable, immutable collections

```
var strings: [String] = [ "a", "b", "c" ]
strings.append("d")
let iStrings = string
iStrings.append("e") // compiler error

let iInts: [Int] = [ 1, 2, 3 ]
var ints = iInts
ints.append("d") // hmmm…
```

siwtch statments

```
switch size {
case "a", "b":
    println("Small")
case "c"..."e":
    println("Medium")
case "f"..<"e":
    println("Large")
default:
    println("Run!!")
}
```

Syntactic sweetness

- Trailing closures

```
dispatch_async(queue) {
    println("dispatch!")
}
```

- Freaking emoji in variables!

```
let 🐶🐮 = "Moof"
```

No emoji in operators 😿

Optional Bools

```
var b: Bool?
if let b = b {
    if (b) {
        println("YES")
    } else {
        println("NO")
    }
} else {
    println("Don't know")
}
```

Runtime

- Compatible Mach-O binaries, ≈ Objective C
- No dynamic lookup: virtual tables (`isa` → `isa` → … `SwiftObject`), "Protocol witness table"
- Devirtualization: no subclasses, `@final`
- Meta programming only through `@objc` (no `Mantle`, no swizzling)
- `struct`'s `functions`

Uniqueness

- Modules
- Name mangling
- `MyClass` & `Model` gone wild
- LLDB can use modules instead of `DWARF` to understand types, including "not included" generics!
- `xcrun swift-demangle _TF5MyApp6myFuncFTSiSi_TSS_ → MyApp.myFunc(Int, Int) -> (String)`

Thanks

```
func memoize<T: Hashable, U>( body: ((T)->U, T)->U ) -> (T)->U {
  var memo = Dictionary<T, U>()
  var result: ((T)->U)!
  result = { x in
    if let q = memo[x] { return q }
    let r = body(result, x)
    memo[x] = r
    return r
}
  return result
}

let factorial = memoize { factorial, x in x == 0 ? 1 : x * factorial(x - 1) }
```

Stop reading NSHipster,
all the cool kids are at dev forums!