Top Surprises in Swift

Tammo Freese (@tammofreese)

January 14th, 2015

Why Focus on Surprises?

Not enough time for a complete introduction into Swift

Why Focus on Surprises?

Not enough time for a complete introduction into Swift

Surprises: Unexpected, astonishing, highlights

Why Focus on Surprises?

Not enough time for a complete introduction into Swift

Surprises: Unexpected, astonishing, highlights

Don't want to talk about expected, non-astonishing :)

Top Surprises in Swift

Top Surprises in Swift

Seven top surprises from the Ruby perspective

Top Surprises in Swift

Seven top surprises from the Ruby perspective

Only surprises in the language

- Won't cover REPL, Playground, command line scripts
- Won't cover current problems with tools

1) No Garbage Collection

1) No Garbage Collection

Automatic Referenc Counting (ARC) instead

1) No Garbage Collection

Automatic Referenc Counting (ARC) instead

References up the reference graph:

- To single owning object: **`unowned`**
- Otherwise: **`weak`**

1) No Garbage Collection

Automatic Referenc Counting (ARC) instead

References up the reference graph:

- To single owning object: **`unowned`**
- Otherwise: **`weak`**

Less of a problem than you may expect

2) Constants

2) Constants

Declare variables with **`var`**

Declare constants with **`let`**

```
var i = 3
let j = 4
```

2) Constants

Declare variables with **`var`**

Declare constants with **`let`**

```
var i = 3
let j = 4
```

Default for function parameters: **`let`**

```
func printTripleOf(i: Int) {
    i *= 3 // COMPILER ERROR
    println(i)
}
```

3) Types

3) Types

Swift is a strongly typed language

```
var numbers: [Int] = [42, 4, 8, 15, 16, 23]
numbers.sort({ (a: Int, b: Int) -> Bool in
    a < b
})
```

3) Types

Swift is a strongly typed language

```
var numbers: [Int] = [42, 4, 8, 15, 16, 23]
numbers.sort({ (a: Int, b: Int) -> Bool in
    a < b
})
```

Good thing: Swift has type inference

```
var numbers = [42, 4, 8, 15, 16, 23]
numbers.sort({ a, b in a < b })
```

3) Types

Swift is a strongly typed language

```
var numbers: [Int] = [42, 4, 8, 15, 16, 23]
numbers.sort({ (a: Int, b: Int) -> Bool in
    a < b
})
```

Good thing: Swift has type inference (and more goodies)

```
var numbers = [42, 4, 8, 15, 16, 23]
numbers.sort { $0 < $1 }
```

4) Optionals

4) Optionals

Typical in typed languages (example Java):

```
int i;                          // Value: Cannot be null
Object object;                  // Reference: Can be null
```

4) Optionals

Typical in typed languages (example Java):

```
int i;                          // Value: Cannot be null
Object object;                  // Reference: Can be null
```

In Swift, non-optional is default, optional is an own type:

```
var i: Int                      // Value: Cannot be nil
var object: AnyObject           // Reference: Cannot be nil
var i2: Int?                   // Optional value: can be nil
var object2: AnyObject? // Optional reference: can be nil
```

4) Optionals: Advantages

We can make values optional

We can make references non-optional

```
func indexOfItem(item: Item) -> Int? {
    // ...
}
```

5) Focus on Values

5) Focus on Values

Ruby: Objects are referenced

Java: Objects are referenced, but primitive types are values

Swift: Objects are referenced, but primitive types are values, …

5) Focus on Values

Ruby: Objects are referenced

Java: Objects are referenced, but primitive types are values

Swift: Objects are referenced, but primitive types are values,

- and **`struct`**s are values,
- and **`enum`**s are values,
- and **`Array`**, **`Dictionary`**, **`String`** are values!

5) Focus on Values

Advantage: Imagine storing an array of 10,000 3D points

5) Focus on Values

Advantage: Imagine storing an array of 10,000 3D points

- Ruby/Java: Around 10,002 memory allocations
- Swift: 1 memory allocation

6) Tuples

6) Tuples

Tuples allow easy grouping of values/references

```
let result = (200, "OK")
// Type of result is (Int, String)
```

Allow optional naming of the elements

```
let result = (code: 200, status: "OK")
// Type of result is (code: Int, status: String)
```

6) Tuples: Special Cases

One tuple type is equivalent to the type inside

```
let x: (Int) = (42) // x has type Int
```

-> Allows parentheses for grouping expressions

Empty tuple type is Void

```
let x: Void = ()
// x has type Void or ()
```

-> Allows function type to be (tuple type) -> (tuple type)

7) Custom Operators

7) Custom Operators

Can be formed from a couple of ASCII characters,
and lots of unicode symbols

```
let numbers = [23, 42]

infix operator ∈ {}
func ∈(candidate: Int, array: [Int]) -> Bool {
    return contains(array, candidate)
}

assert(42 ∈ numbers)
```

7) Custom Operators

Can be formed from a couple of ASCII characters,
and lots of unicode symbols

```
let numbers = [23, 42]

infix operator ∈ {}
func ∈<T: Equatable>(candidate: T, array: [T]) -> Bool {
    return contains(array, candidate)
}

assert(42 ∈ numbers)
```

Phew! That's quite a lot of new stuff, isn't it?

The One Thing to Remember: Swift Will Be Huge

Swift is a complete replacement for both the C and Objective-C languages.

— *Xcode 6 Release Notes*

Thanks for listening!