A Swift Kickstart

Daniel H Steinberg
@dimsumthinking

Copyright Dim Sum Thinking, Inc 2014

Structs

Create struct

```swift
struct Vertex {
}

let point = Vertex()
```

Properties

```swift
struct Vertex {
    var x = 3.0
    var y = 4.0
}

let point = Vertex()
point.x
point.y
```

Can't

```
struct Vertex {
    var x = 3.0
    var y = 4.0
}

let point = Vertex()
point.x
point.y
point.x = 20.0
```

let => var

```swift
struct Vertex {
    var x = 3.0
    var y = 4.0
}

var point = Vertex()
point.x
point.y
point.x = 20.0
```

Automatically generated init

```swift
struct Vertex {
    var x, y: Double
}

var point = Vertex(x: 3.0, y: 4.0)
point.x
point.y
point.x = 20.0
```

By Value

```swift
struct Vertex {
    var x, y: Double
}

var point = Vertex(x: 3.0, y: 4.0)
var anotherPoint = point
point.x
point.y
point.x = 20
anotherPoint.y = 50

point
anotherPoint
```

Custom Operator

```
import Foundation

prefix operator √ {}  // option - v

prefix func √ (argument: Double) -> Double {
    return sqrt(argument)
}
```

Custom Operator

```
postfix operator ** {}

postfix func ** (number:Double) -> Double {
    return number * number
}
```

Custom Operator

```swift
struct Vertex {
    var x, y: Double
    var magnitude: Double {
        return √(x** + y**)
    }
}

var point = Vertex(x: 3.0, y: 4.0)

point.magnitude
```

Get and Set

```
struct Vertex {
    var x, y: Double
    var magnitude: Double {
        get{
            return √(x** + y**)
        }
        set(newValue) {
            let multiplier = newValue/magnitude
            x *= multiplier
            y *= multiplier
        }
    }
}

var point = Vertex(x: 3.0, y: 4.0)
point.magnitude
point.magnitude = 10
```

Will Set Did Set

```
struct Vertex {
    var x: Double {
        willSet(newValue) {
            println("About to change x from \(x) to \(newValue)")
        }
        didSet(oldValue) {
            println("Did change x from \(oldValue) to \(x)")
        }
    }
    var y: Double
    var magnitude: Double {
        get{
            return √(x** + y**)
        }
        set(newValue) {
            let multiplier = newValue/magnitude
            x *= multiplier
            y *= multiplier
        }
    }
}
```

Mutating

```
struct Vertex {
    var x, y: Double  // slides omit willSet and didSet for space
    var magnitude: Double {
        get{
            return √(x** + y**)
        }
        set(newValue) {
            let multiplier = newValue/magnitude
            x *= multiplier
            y *= multiplier
        }
    }
    mutating func moveByX(x: Double) {
        self.x += x
    }
}

var point = Vertex(x: 3.0, y: 4.0)
point.magnitude
point.magnitude = 10
point.moveByX(4)
point
```

Non - Mutating

```
struct Vertex {
    var x, y: Double
    var magnitude: Double {
        get{
            return √(x** + y**)
        }
        set(newValue) {
            let multiplier = newValue/magnitude
            x *= multiplier
            y *= multiplier
        }
    }
    mutating func moveByX(x: Double) -> Vertex {
        return Vertex(x: self.x + x, y: y)
    }
}

var point = Vertex(x: 3.0, y: 4.0)
point.magnitude
point.magnitude = 10
let shiftedPoint = point.moveByX(4)
point
shiftedPoint
```

Composition

```swift
struct Size {
    var width, height: Double
}

struct Rectangle {
    var size: Size
    var topLeftCorner: Vertex
}

var point = Vertex(x: 3.0, y: 4.0)

let rectangle = Rectangle(size: Size(width: 200, height: 100),
                             topLeftCorner: point)
```

Try this

- Continue with the example from the last section. Convert the two classes to structs. A HotBeverage should have a Beverage.

- Create the Beverage init() uses a SizeOfCup. Add a mutating function to it named sizeOfDrink(). Implement didSet for amount to print the amount left in the cup or that it is empty.

- Create an init method for HotBeverage that takes a SizeOfCup and creates its beverage. Sip should reduce its beverage's amount by one.

Try this

```
struct Beverage {
    var amount: Int {
        didSet {
            if isEmpty {
            println("The drink is now empty")
            } else {
                println("The drink has \(amount) left")
            }
        }
    }
    var isEmpty: Bool {
        return amount <= 0
    }
    init (sizeOfCup: SizeOfCup ) {
        self.amount = sizeOfCup.rawValue
    }
    mutating func drink(deltaAmount: Int){
        amount -= deltaAmount
    }
}

struct HotBeverage {
    var beverage: Beverage
    init(sizeOfCup: SizeOfCup) {
        beverage = Beverage(sizeOfCup: sizeOfCup)
    }
    mutating func sip() {
        beverage.drink(1)
    }
}
```

A Swift Kickstart

DANIEL H STEINBERG

Introducing
the Swift Programming Language

Editors Cut

https://itunes.apple.com/us/book/a-swift-kickstart/id891801923?mt=11&uo=4&at=11l56E