Módulo Swift [Swift avanzado]

Daniel Illescas Romero

KEEPCODING
Tech School

Índice

1. *Closures*.

Las funciones pueden aceptar otras funciones.

```
func add(a: Int, b: Int) -> Int {
    return a + b
}
func subtract(a: Int, b: Int) -> Int { a - b }
func doOperation(_ operation: (Int, Int) -> Int, a: Int, b: Int) -> Int {
    return operation(a, b)
}
doOperation(add, a: 1, b: 2)
doOperation(subtract, a: 1, b: 2)
```

Estas también pueden aceptar funciones anónimas (o *lambdas*), llamadas en Swift ***closures***:

```
doOperation({ a, b in a * b }, a: 1, b: 2)
```

2. Propiedades.

- Propiedades calculadas (*computed properties*): parecidas a las funciones pero no aceptan parámetros.

```
var diameter: Double {
    return radius * 2
}
```

- Observadores de propiedades: con `willSet` (antes de cambiar valor) y `didSet` (después).

```
var something: Int = 0 {
    didSet {
        print("did set", oldValue, something)
    }
}
```

- *lazy properties*: computan su valor solo cuando se invocan:

```
lazy var complexOperation: Int = factorial(20)
```


3. Protocolos/interfaces.

Conocido habitualmente como interfaz (*interface*), en Swift se denominan protocolos (*protocols*) y sirven para modelar qué propiedades o métodos debe tener una estructura.

```
protocol CanFly {
    var hasWings: Bool { get }
    func fly()
}

struct Bird: CanFly {

    var hasWings: Bool { true }

    func fly() {
        print("I am flying :)")
    }
}
```

4. Extensiones.

Con extensiones podemos añadir nuevas propiedades computadas y métodos a cualquier estructura (`enum` , `struct` , `class`). También podemos hacer que un tipo conforme a un protocolo que queramos.

```
enum Color { case blue, red, yellow }
// las extensiones pueden estar en un fichero diferente al que usamos para declarar el tipo
extension Color {
    var isBlueOrRed: Bool {
        return self == .blue || self == .red
    }
}
```

```
import Foundation
protocol IsBlank { var isBlank: Bool { get } }
extension String: IsBlank {
    var isBlank: Bool {
        return self.trimmingCharacters(in: .whitespacesAndNewlines).isEmpty
    }
}
```

5. Control de errores.

En general el control de errores en Swift es similar al de otros lenguajes en el que se usen excepciones y `try-catch` (`do-catch` en Swift), pero en este caso en vez de lanzar excepciones lanzamos errores, lo cuál hace que sea computacionalmente más liviano que una excepción.

```
enum RequestError: Error { // Así de fácil creamos nuestros propios errores
    case incorrectURL
    // ...
}
func request(url: String) throws -> Data { // importante el `throws`
    if url.isEmpty { throw RequestError.incorrectURL } // lanzamos error
    // ...
}
// `do` en vez de `try`
do { // `try` se utiliza para llamar a código que pueda lanzar errores
    try request(url: "https://example.com")
} catch { // o: } catch let error {
    // aquí tenemos disponible una variable `error`
}
```

6. Bibliografía y enlaces interesantes.

- *The Swift Programming Language*: https://docs.swift.org/swift-book/documentation/the-swift-programming-language

KEEPCODING

Tech School

Madrid | Barcelona | Bogotá

Datos de contacto

Daniel Illescas Romero
email: daniel.illescas.r@gmail.com