Funciones

Sintáxis para declaración

Las funciones sirven para escribir código modular y reusable. La declaración de una función en swift se hace con la palabra clave `func`.

```
func ejemplo() {
   let mensaje = "Hola mundo"
   print(mensaje)
}
```

Esta función llamada `ejemplo` tiene algunas instrucciones en su interior, no recibe argumentos ni devuelve algún resultado. Para ser invocada la función, se debe escribir su nombre y los paréntesis (aunque se encuentren vacios). Si no se incluyen los paréntesis el compilador señalará un error de sintáxis.

Funciones que reciben argumentos

Para que una función reciaba un argumento es necesario indicarlo dentro de los paréntesis al declarar la función. Se debe indicar el nombre del argumento y su tipo de dato.

```
func numero(n:Int) {
   print("El número recibido es \(n)")
}

numero(n:2)
```

Observe que al invocar a la función y enviarle un argumento, se debe hacer referencia a la constante que recibirá el valor. Estos se llaman parámetros etiquetados. Si omitiera el nombre de la constante que recibirá el valor (aún y cuando sea sólo un parámetro) el compilador indicará un error.

Si se desean más parámetros de entrada hay que listarlos en la declaración de la función.

```
func numeros(n1:Int, n2: Double) {
    print("Los números recibidos son \(n1) y \(n2)")
}

numeros(n1:5, n2:6)
```

Observe que los tipos de datos de los argumentos pueden ser diferentes entre sí.

Considere además que cuando se envían varios parámetros a la función, se deben listar en el orden

en el que fueron declarados.

Además considere que en swift los argumentos de entrada son siempre constantes, por lo que si desea cambiar el valor del argumento se debe declarar una variable del mismo tipo y asignarle el valor dentro de la misma función. Esa variable puede tener incluso el mismo nombre si así lo desea.

Funciones que devuelven resultado

Para que las funciones devuelvan un resultado es necesario indicarlo en la declaración. Se debe indicar el tipo de dato que será devuelto y dentro de la función se debe utilizar la palabra clave `return`.

```
func suma(n1:Int, n2:Int) -> Int {
    return n1+n2
}

let s = suma(n1:5, n2:6)
print(s)
```

Argumentos no etiquetados

Swift permite que los argumentos no tengan una etiqueta si así lo desea, aunque no se recomienda. Para ello, se debe agregar el caracter especial `_` previo al nombre de la etiqueta en la declaración de la función.

```
func suma(_ n1:Int, _ n2:Int) -> Int {
    return n1+n2
}

let s = suma(5, 6)
print(s)
```

Argumentos personalizados

Es posible cambiar las etiquetas de los argumentos de las funciones. Se puede personalizar la etiqueta e indicar la constante en la cual se recibirá el valor.

```
func saludo(_ persona: String, en dia: String) -> String {
    return "Hola \(persona), hoy es \(dia)."
}
let x = saludo("Juan", en: "Miercoles")
print(x)
```

Funciones anidadas

Se pueden anidar funciones, esto es, definir una función dentro de otra función.

```
func returnFifteen() -> Int {
    var y = 10
    func add() {
        y += 5
    }
    add()
    return y
}
let y = returnFifteen()
print(y)
```

Funciones devueltas por otra función

Las funciones son tipo primera clase, esto significa que una función puede devolver otra función.

```
func makeIncrementer() -> ((Int) -> Int) {
    func addOne(number: Int) -> Int {
        return 1 + number
    }
    return addOne
}
var increment = makeIncrementer()
increment(7)
```

Funciones como argumento

Las funciones pueden tener una función como su argumento de entrada.

```
func hasAnyMatches(list: [Int], condition: (Int) -> Bool) -> Bool {
    for item in list {
        if condition(item) {
            return true
        }
    }
    return false
}
func lessThanTen(number: Int) -> Bool {
    return number < 10
}
var numbers = [20, 19, 7, 12]
let b = hasAnyMatches(list: numbers, condition: lessThanTen)
```

```
print(b)
```