Closures

Semana 06 - Abner Castro

@abnerabbey, abner.castro@wizeline.com

AGENDA

- Introducción a los closures
- Trailing closures
- Sintaxis de los closures
- Infiriendo tipos dentro de closures
- Nombres de argumentos cortos

Introducción a los closures

Son bloques de funcionalidades, operaciones que pueden ser usados o pasados a través del código

Los closures pueden capturar y guardar referencias strong a cualquier constante o variable del contexto en donde están definidos

Las funciones en realidad son casos especiales de closures. Los closures tienen tres formas:

1. Funciones globales, que son closures con nombres y no capturan ningún valor
2. Funciones anidadas, que tienen nombre y capturan valores de la función que las contiene
3. Expresiones sin nombre que son escritas con una sintaxis ligera para capturar valores de su alrededor

Los closures son limpios, tienen un estilo claro, con optimizaciones que incitan a que sean breves. Las optimizaciones incluyen:

- Infieren parámetros y regresan los valores de acuerdo a su contexto
- Returns implícitos para closures con una sola línea de expresión
- Argumentos cortos
- Sintaxis para trailing closures

Sintaxis de los closures

```
{ (parameters) -> return type in
    statements
}
```

Los parámetros pueden ser *in-out*, pero no pueden tener un valor por default

Infiriendo el Tipo por el contexto

```
let names = ["Hola", "Este", "Es", "Un", "Nombres"]

let reversedNames = names.sorted(by: { s1, s2 in
    return s1 > s2
})
```

Los closures pueden inferir los tipos de los parámetros y el tipo del valor que regresa. Esto es posible cuando a una función se le pasa un closure.

Valores de retorno implícitos

```
let names = ["Hola", "Este", "Es", "Un", "Nombres"]

let reversedNames = names.sorted(by: { s1, s2 in
    s1 > s2
})
```

Los closures pueden regresar valores omitiendo la palabra *return* para una sola línea

Nombres de argumentos cortos

```
let names = ["Hola", "Este", "Es", "Un", "Nombres"]

let reversedNames = names.sorted(by: { $0 > $1 })
```

Swift automáticamente provee nombres de argumentos y estos son usados para referir los valores con los nombres *$0, $1, $2*, y así.

Trailing Closures

```
func someFunctionThatTakesAClosure(closure: () -> ())

someFunctionThatTakesAClosure {

}
```

Esto sucede cuando una función recibe closures como parámetros y estos, son los últimos.

Thank you

Retroalimentación

¡Déjanos saber cómo podemos mejorar!

https://forms.gle/6vZLAv7j7X6FRdF79