Closures

by Nedinic Nikola

Syntax & Definition

- anonyme\r Codeblock || Funktion

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

```
func backward(_ s1: String, _ s2: String) -> Bool {
    return s1 > s2
}
var reversedNames = names.sorted(by: backward)
// reversedNames is equal to ["Ewa", "Daniella", "Chris", "Barry", "Alex"]
```

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

Eigenschaften

- können Datentyp ableiten/erben
- verkürzte Schreibweisen möglich (Shorthand Arguments, Implicit Returns)
- Capture-fähig

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

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

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

Trailing Closures

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

```
reversedNames = names.sorted { $0 > $1 }
```

```
1  func someFunctionThatTakesAClosure(closure: () -> Void) {
2      // function body goes here
3  }
4
5  // Here's how you call this function without using a trailing closure:
6
7  someFunctionThatTakesAClosure(closure: {
8      // closure's body goes here
9  })
10
11 // Here's how you call this function with a trailing closure instead:
12
13 someFunctionThatTakesAClosure() {
14     // trailing closure's body goes here
15 }
```

Capture

```swift
func makeIncrementer(forIncrement amount: Int) -> () -> Int {
    var runningTotal = 0
    func incrementer() -> Int {
        runningTotal += amount
        return runningTotal
    }
    return incrementer
}
```

- Konstanten & Variablen aus umliegenden Scope
- Referenz und Manipulation dieser Variablen ohne innere Definition

```swift
let incrementByTen = makeIncrementer(forIncrement: 10)
```

```swift
incrementByTen()
// returns a value of 10
incrementByTen()
// returns a value of 20
incrementByTen()
// returns a value of 30
```

```swift
let incrementBySeven = makeIncrementer(forIncrement: 7)
incrementBySeven()
// returns a value of 7
```

Escaping

Auf Variablen zugreifen außerhalb des umliegenden Scopes: @escaping

```
1  var completionHandlers: [() -> Void] = []
2  func someFunctionWithEscapingClosure(completionHandler: @escaping () ->
     Void) {
3      completionHandlers.append(completionHandler)
4  }
```

Autoclosures

```
1  var customersInLine = ["Chris", "Alex", "Ewa", "Barry", "Daniella"]
2  print(customersInLine.count)
3  // Prints "5"
4
5  let customerProvider = { customersInLine.remove(at: 0) }
6  print(customersInLine.count)
7  // Prints "5"
8
9  print("Now serving \(customerProvider())!")
10 // Prints "Now serving Chris!"
11 print(customersInLine.count)
12 // Prints "4"
```

Beispiel

Funktion - mit 2 Ints und 1 closure als Übergabeparameter

Funktion summiert über den Rückgabewert der Closure returns (Capturen)

Closures (Shorthand)

- Quadrat der Zahl
- Summe der Zahlen
- Falls $0 durch $1 teilbar -> Quotient ansonsten 0