

Functions

Bauer Philipp

Syntax

```
func greet(person: String) -> String {  
    let greeting = "Hello, " + person + "!"  
    return greeting  
    // return "Hello, " + person + "!"  
}
```

```
print(greet(person: "Anna"))  
// Prints "Hello, Anna!"  
print(greet(person: "Brian"))  
// Prints "Hello, Brian!"
```

- Ausführbarer benannter Codeblock mit optionalen Parametern und Rückgabewerten
- Parameter Namen müssen grundsätzlich beim Aufruf angeführt werden

Eigenschaften

- Funktionen ohne Parameter natürlich auch möglich
- Oder mit mehreren
- Ohne Rückgabewert

```
func sayHelloWorld() -> String {  
    return "hello, world"  
}  
print(sayHelloWorld())  
// Prints "hello, world"
```

```
func greetMore(person: String, alreadyGreeted: Bool) -> String {  
    if !alreadyGreeted {  
        return greet(person: person)  
    }  
}
```

Multiple Returns

- Tupel zur Verwendung von mehreren Rückgabewerten
- Diese können auch optionals sein

```
func minMax(array: [Int]) -> (min: Int, max: Int) {  
    //... calc min, max of param-array  
    return (currentMin, currentMax)  
}
```

```
let bounds = minMax(array: [8, -6, 2, 109, 3, 71])  
print("min is \ \(bounds.min) and max is \ \(bounds.max)")
```

```
func minMax(array: [Int]) -> (min: Int, max: Int)? {}  
  
if let bounds = minMax(array: [8, -6]) {  
    print("\ (bounds.min) ; \ (bounds.max)")  
}
```

Argument Labels

- Vereinfachen Aufrufe und geben dem Code Lesbarkeit
- `'_'` als AL erlaubt es einen Parameter ohne Parameternamen zu übergeben
 - -> "Omitting Argument Labels"

```
func greet(person: String, from hometown: String) -> String {  
    return "Hello \((person)! Glad you could visit from \((hometown))."  
}  
print(greet(person: "Bill", from: "Cupertino"))
```

```
func someFunction(_ first: Int, second: Int) { }  
someFunction(1, secondParameterName: 2)
```

Default Parameter Values

- Wenn dieser nicht übergeben wird nimmt dieser den gesetzten Wert an

```
func someFunction(parameterWithoutDefault: Int, parameterWithDefault: Int = 12) {  
    // If you omit the second argument when calling this function, then  
    // the value of parameterWithDefault is 12 inside the function body.  
}  
  
someFunction(parameterWithoutDefault: 3, parameterWithDefault: 6)  
// parameterWithDefault is 6  
  
someFunction(parameterWithoutDefault: 4)  
// parameterWithDefault is 12
```

Variadic Parameters

- Akzeptiert 0 oder mehr Parameter eines Datentyps
- Eine Funktion darf maximal 1 davon haben
- Zusätzliche Parameter an die vordere Stelle

```
func arithmeticMean(_ numbers: Double...) -> Double {  
    //calculate and return  
}  
arithmeticMean(1, 2, 3, 4, 5)  
arithmeticMean(3, 8.25, 18.75)
```

In-Out Parameters

- Parameter sind standardmäßig Konstanten -> nicht veränderbar
- Parameter mit dem “inout” keyword
- Mit ‘&’ werden Referenzen als Parameter übergeben -> haben nach dem Beenden der Funktion den in der Funktion gesetzten Wert

```
func swapTwoInts(_ a: inout Int, _ b: inout Int) {  
    let temporaryA = a  
    a = b  
    b = temporaryA  
}
```

```
var someInt = 3  
var anotherInt = 107  
swapTwoInts(&someInt, &anotherInt)  
print("\(someInt) ; \(anotherInt)")  
// Prints "someInt is now 107, and anotherInt is now 3"
```


Functions Types

- Typ aus der Definition einer Funktion

```
func addTwoInts(_ a: Int, _ b: Int) -> Int  
(Int, Int) -> Int
```

```
func printHelloWorld()  
() -> Void
```

- Variablen aus Funktionstypen anlegen möglich

```
var mathFunction: (Int, Int) -> Int =  
addTwoInts
```

```
mathFunction = multiplyTwoInts
```

```
print("Result: \(\mathFunction(2, 3))")
```

Function Types as Parameter Types

- Jede Funktion mit dem selben Function Type kann hier übergeben werden

```
func printRes(_ mathFunction: (Int, Int) -> Int, _ a: Int, _ b: Int) {  
    print("Result: \(\mathFunction(a, b))")  
}  
  
printRes(addTwoInts, 3, 5)  
  
// Prints "Result: 8"
```

Function Types as Return Types

```
func stepForward(_ input: Int) -> Int {  
    return input + 1  
}
```

```
func stepBackward(_ input: Int) -> Int {  
    return input - 1  
}
```

```
func chooseStepFunction(backward: Bool) -> (Int) -> Int {  
    return backward ? stepBackward : stepForward  
}
```

```
var currentValue = 3
```

```
let moveNearerToZero = chooseStepFunction(backward: currentValue > 0)
```

Nested Functions

- Alle Funktionen bis jetzt sind globale Funktionen
- Nested Functions = Funktionen in Funktionen -> sind “private”

```
func chooseStepFunction(backward: Bool) -> (Int) -> Int {  
    func stepForward(input: Int) -> Int { return input + 1 }  
    func stepBackward(input: Int) -> Int { return input - 1 }  
    return backward ? stepBackward : stepForward  
}
```

Übung

- Funktion die eine Funktion und einen Bool wert übergeben bekommt
 - Parameter- Funktion soll aus beliebig vielen Integer Werten ein Tupel aus 2 zusammenhanglosen, aus den Parametern, resultierenden Werten bestehen
 - Der Bool Wert (soll ohne Namen übergeben werden können) bestimmt ob das Return Tupel ausgegeben werden soll oder nicht

Functions

Bauer Philipp