Swift 4 Kurzreferenz

Konstanten und Variablen

```
let constant = 42
var value = 5
value += 5
```

Explizite Typangaben

```
var anotherValue : Int = 5
```

Ausgaben

```
print("Value", value)
print("Value \(value)")
NSLog("%@", "Value \(value)")
NSLog("Value %i", value)
```

Kontrollfluss

Bedingte Anweisung: if

```
if temperatur < 20 { print("kalt") }
else if temperatur < 0 { print("kälter") }
else { print("warm") }
```

Konditionaler Ausdruck

```
msg = temperatur < 20 ? "zu kalt" : "zu warm"
```

Fallunterscheidung: switch

```
switch value {
case 0:
  print("Keine")
case 0..<8:
  print("Wenige")
default:
  print("Unbekannter Wert")
}
```

for

```
for i in 5..<10 { print(i) }
```

while, repeat ... while

```
var i = 0
while i < 10 {
  print(i); i+=1
}
repeat {
  print(i); i+=1
} while i < 20
```

Optionale Typen

Erzwungenes Auspacken

```
var parsedNumber : Int? = Int(someString)
var number     : Int  = parsedNumber!
```

Binden von optionalen Werten

```
if let number = parsedNumber {
  print("Ergebnis: ", number)
}

if let a = Int("2"), let b = Int("3"), a<b {
  print("\(a) ist kleiner als \(b)")
}
```

Implicitly Unwrapped Optionals

```
var istr : String! = "foo" // wie String?
istr.contains("foo") // wie istr!
```

Zeichenketten

Zeichenketten erstellen

```
var str = "Ich ♥ Swift / Ich \u{2665} Swift"
let emptyString = String()
msg = String(repeating: "*", count: 5)
msg = "r = \(r), area = \(Double.pi * pow(r,
2))"
```

Länge

```
msg.count
msg.isEmpty
```

Einfügen von Inhalten

```
str += msg
str.insert(contentsOf: msg, at:
str.startIndex)
```

Bereiche in Strings

```
let start = str.startIndex
let end = str.index(start, offsetBy: 10)
let range = start..<end
let substring = str[range]
```

Suchen & Ersetzen

```
while let range = str.range(of: "***") {
  str.removeSubrange(range)
}
str.replacingOccurrences(of: "/", with: "-")
```

Listen

Listen erzeugen

```
var numbers : [Int] = [1, 2, 3, 5, 8]

let emptyArray = [Int]()
let emptyArray2 : [Int] = []

let reps = [Int](repeating: 22, count: 4)
```

Listen: Elementzugriff

```
let count = numbers.count
let empty = numbers.isEmpty

let firstElement  = numbers.first
let secondElement = numbers[1]
let lastElement   = numbers.last
```

Suchen in Listen

```
let matchingElement = numbers.index(of:8)
```

Teilbereiche von Listen

```
numbers[0...2] == numbers[0..<3] // [1, 2, 3]
```

Listen: Iteration

```
for number in numbers {
  print(number)
}
for (index, number) in numbers.enumerated() {
  print("#", index + 1, ": ", number)
}
numbers.forEach { (number) in
  print(number)
}
```

Listenmanipulation

```
numbers[0] = 0

numbers.append(13)
numbers += [ 21, 34 ]

let first = numbers.remove(at: 0)
numbers.insert(1, at: 0)
let last = numbers.removeLast()
```


Ralf Ebert
www.ios-schulung.de
www.ralfebert.de/ios/

Blockbasierte Listenoperationen

map und flatMap

```
let text = ["Hello world", "Foo bar"]

text.map { (str) in
  str.components(separatedBy: " ")
} // => [["Hello", "world"], ["Foo", "bar"]]

text.flatMap { (str) in
  str.components(separatedBy: " ")
} // => ["Hello", "world", "Foo", "bar"]
```

Filtern

```
numbers.filter { $0 % 2 == 0 }
```

Sortieren

```
numbers.sort()
numbers.sort { $0 > $1 }
numbers.sort(by: >)
```

Summieren

```
numbers.reduce(0) { (sum, e) in sum + e }
```

Dictionaries und Sets

Assoziative Dictionaries (Schlüssel : Wert)

```
var dictionary = ["Apple": 10, "Kiwi": 15]
let emptyDict : [String:Int] = [:]

let amount = dictionary["Apple"]
dictionary["Orange"] = 20

for (key, value) in dictionary {
  print(key, " => ", value)
}
```

Set: ohne Reihenfolge, eindeutige Werte

```
let numberSet : Set<Int> = Set([3, 5, 8])
numberSet.subtracting(Set([5, 9])) // [3, 8]
numberSet.intersection(Set([5, 9])) // [5]
numberSet.union(Set([5, 9])) // [3, 5, 8, 9]
```

Objektorientierte Programmierung

Klassendefinition, Eigenschaften, Initializer, berechnete Eigenschaften

```
class Circle {
  var radius : Double {
    willSet { print(radius, "=>", newValue) }
    didSet { print(oldValue, "=>", radius) }
  }

  var diameter : Double {
    get { return radius * 2 }
    set { self.radius = newValue * 0.5 }
  }

  var area : Double {
    return Double.pi * pow(radius, 2)
  }

  init(radius : Double) {
    self.radius = radius
  }

  func scaled(factor : Double) -> Circle {
    let newRadius = self.radius * factor
    return Circle(radius: newRadius)
  }
}
```

Strukturtypen

```
struct Square {
    var length : Double

    mutating func scale(factor : Double) {
        self.length = self.length * factor
    }
}
```

Singleton, statische Eigenschaften

```
class Greeter {
    static var shared = Greeter()

    func sayHello() { print("Hello!") }
}

Greeter.shared.sayHello()
```

Typumwandlung

```
someValue as? String
// -> String?, nil wenn kein String

someValue as! String
// -> String, Crash wenn kein String
```

Protokolle

```
protocol Account {
  var balance : Int { get set }
  func withdraw(amount : Int)
}

class CheckingAccount : Account {
  var balance = 0

  func withdraw(amount : Int) {
    balance -= amount
  }
}
```

Extensions

```
extension Date {
  func addDays(days : Int) -> Date? {
    let cal = Calendar.current
    return cal.date(byAdding: .day,
        value: days, to: self)
  }
}
```

Enums

```
enum Direction {
  case north
  case south
  case east, west
}

var direction = Direction.north
direction = .south
```

Enums: Associated Values

```
enum Result {
  case success(String)
  case error(Error)
}

let result = Result.success("Test")

switch(result) {
case .success(let value):
  print("Sucess, got \(value)")
case .error(let error):
  print("Error, got \(error)")
}
```

Funktionale Programmierung

Swift-Typen für Funktionen

```
func triple(value : Int) -> Int {
  return value * 3
}

let operation : (Int) -> Int = triple

numbers.map(triple)
```

Closures

```
numbers.map({ (value : Int) -> Int in
  return value * 3
})
numbers.map({ (value) in value * 3 })
numbers.map({ $0 * 3 })
numbers.map { $0 * 3 }
```

CompletionHandler

```
class Example {
  typealias CompletionHandler = () -> ()

  func load(completion: CompletionHandler?) {
    // ... operation ...
    completion?()
  }
}

let example = Example()
example.load { print("completed!") }
```

Foundation APIs

JSON parsen

```
struct Person : Codable  {
    var name : String
}

try! JSONDecoder()
      .decode(Person.self, from: data)

// JSON erzeugen

try! JSONEncoder()
      .encode(Person(name: "Bob"))
```


Ralf Ebert
www.ios-schulung.de
www.ralfebert.de/ios/

Fehlerbehandlung

try? - nil im Fehlerfall

```
var opresult = try? dangerousOperation()
```

try! - Crash im Fehlerfall

```
opresult = try! dangerousOperation()
```

do ... try ... catch

```
do {
  opresult = try dangerousOperation()
} catch let error {
  print("Error: \(error)")
}
```

Eigene ErrorTypes

```
enum PurchaseError: Error {
  case unknownProduct
  case outOfStock
}

func purchase() throws -> String {
  throw PurchaseError.outOfStock
}

do {
  opresult = try purchase()
}
catch PurchaseError.outOfStock { print("out
of stock") }
catch is PurchaseError { print("purchase
error") }
catch { print("other error") }
```

Speicherverwaltung

Schwache Referenzen

```
weak var exampleRef : Example?
```

Schwache Referenzen in Blöcken

```
example.load { [weak self] in
  print("Completed \(self)!")
}
```