Kotlin Cheat Sheet

1. Klassen

Klassen sind per Default `final` und `public`.

```
abstract class Car(val name: String)

class Seat(val model: String, val color: String) : Car("Seat")

val focus = Seat("Leon", "Blue")

println(focus.name()         -> prints "Seat"
println(focus.model()        -> prints "Leon"
println(focus.color()        -> prints "Blue"

val alhambra = Seat(color = "Silver", model = "Alhambra")
```

2. if, when, for, while

`if` Anweisung

```
// Traditional usage
var max = a
if (a < b) max = b

// With else
var max: Int
if (a > b) {
    max = a
} else {
    max = b
}

// As expression
val max = if (a > b) a else b
```

Die letzte Anweisung in einem Block ist der Rückgabewert.

```
val max = if (a > b) {
    print("Choose a")
    a
} else {
    print("Choose b")
    b
}
```

`when` Anweisung

`when` ist Kotlins Variante des `switch`-Statements.

```
when (x) {
    in 1..10 -> print("x is in the range")
    in validNumbers -> print("x is valid")
    !in 10..20 -> print("x is outside the range")
    else -> print("none of the above")
}
```

`when` lässt sich sehr gut mit Single-Line-Expressions kombinieren.

```
fun hasPrefix(x: Any) = when(x) {
    is String -> x.startsWith("prefix")
    else -> false
}
```

`for` Schleife

`for` iteriert durch alles, was einen `Iterator` anbietet.

```
// Ohne Block
for (item in collection) print(item)


// Mit Block
for (item: Int in ints) {
    // ...
}
```

Verwendung von *Range Expressions*:

```
for (i in 1..3) {
    println(i)
}
for (i in 6 downTo 0 step 2) {
    println(i)
}
```

Verwendung des Index eines Arrays:

```
for (i in array.indices) {
    println(array[i])
}

// oder:
for ((index, value) in array.withIndex()) {
    println("the element at $index is $value")
}
```

3. Funktionen

```
fun double(x: Int): Int {
    return 2 * x
}

val result = double(2)

Sample().foo() // create instance of class Sample and call foo
```

Man kann Default-Werte setzen:

```
fun read(b: Array<Byte>, off: Int = 0, len: Int = b.size) { ... }
```

Überschreiben von Funktionen. Dabei darf der Default-Wert der überschriebenen Funktion nicht verändert werden.

```
open class A {
    open fun foo(i: Int = 10) { ... }
```

```
}

class B : A() {
    override fun foo(i: Int) { ... }  // no default value allowed
}
```

Argumente können in ihrer Reihenfolge vertauscht werden, wenn man beim Aufruf ihre Namen angibt.

```
fun reformat(str: String,
             divideByCamelHumps: Boolean = false,
             upperCaseFirstLetter: Boolean = true,
             normalizeCase: Boolean = true,
             wordSeparator: Char = ' ') {
...

reformat(str,
    normalizeCase = true,
    upperCaseFirstLetter = true,
    divideByCamelHumps = false,
    wordSeparator = '_'
)
}
```

4. Spring Boot

`runApplication()`

Das Starten einer Spring-Boot-Anwengung kann mit einer *Package-Level-Funktion* `main(args: Array<String>)` implementiert werden. Diese muss die Funktion `runApplication()` aus dem Spring-Boot-Framework aufrufen.

```
package com.example.demo

import org.springframework.boot.autoconfigure.SpringBootApplication
import org.springframework.boot.runApplication

@SpringBootApplication
open class Application

fun main(args: Array<String>) {
    runApplication<Application>(*args)
}
```