Cheat Sheet - KOTLIN

Basics

Declaring variables

```
val name = "Marcin"  // Can't be changed
var age = 5          // Can be changed
age++
```

"Hello, World" program

```
fun main(args: Array<String>) {
    println("Hello, World")
}
```

Declaring function

```
fun sum(a: Int, b: Int): Int {
    return a + b
}
```

Classes

Primary constructor

```
val declares a read-only property, var a mutable one
class Person(val name: String, var age: Int)
// name is read-only, age is mutable
```

Inheritance

```
open class Person(val name: String) {
    open fun hello() = "Hello, I am $name"
    // Final by default so we need open
}
class PolishPerson(name: String) : Person(name) {
    override fun hello() = "Dzień dobry, jestem $name"
}
```

Properties with assessors

```
class Person(var name: String, var surname: String) {
    var fullName: String
        get() = "$name $surname"
        set(value) {
            val (first, rest) = value.split(" ", limit = 2)
            name = first
            surname = rest
        }
}
```

Comentários em bloco são usados para gerar documentação, descrevem classes, métodos e atributos: **/** TEXTO */**

Tipos de dados

CONVERSÃO DE TIPOS

- A conversão de tipos precisa ser **explícita**
- Conversão através de **funções** de **suporte**

```
.toByte()
.toShort()
.toInt()
.toLong()
.toFloat()
.toDouble()
.toChar()
```

Operadores

RELACIONAIS

número (operador) número → booleano

Operador	Descrição
==	Igualdade
!=	Diferença
>	Maior
>=	Maior ou igual
<	Menor
<=	Menor ou igual

Entrada e saída

LEITURA DE VALORES

```
print("Digite uma frase: ")
val texto = readLine()
println(texto)

print("Digite um número inteiro: ")
val numero = readLine()!!.toInt()
println(numero)
```

Estruturas de decisão

COMANDOS EM LINHA

```
if (numero > 0) println("$numero é positivo")

val podeBeber = if (idade ⩾ 18) "sim" else "não"
println("Você $podeBeber beber")
```

Estruturas de decisão

```
when (operador) {
    "+" → println("$a + $b = ${a + b}")
    "-" → println("$a - $b = ${a - b}")
    "*" → println("$a * $b = ${a * b}")
    "/" → println("$a / $b = ${a / b}")
    else → println("Operador inválido")
}
```

```
var valor: String

do {
    println("Digite 'sair' para sair: ")
    valor = readLine()!!
} while (valor ≠ "sair")
```

Cheat Sheet - KOTLIN

Listas imutáveis

```kotlin
// Listas imutáveis
val cores: List<String> = listOf("azul", "verde", "laranja")

// Acessando um elemento
print(cores[0])

// Percorrendo uma lista
for (cor in cores) {
    print("${cor}, ")
}

// Tamanho da lista
print(cores.size)
```

Listas mutáveis

```kotlin
// Listas mutáveis
val frutas: MutableList<String> = mutableListOf("banana", "uva", "maçã")

// Adicionando um elemento
frutas.add("laranja")

// Ordenado uma lista
frutas.sort()

// Removendo um elemento
frutas.removeAt(0)
frutas.remove("uva")
```

Objetos

- O **universo** é formado por objetos
- Cada objeto possui:
  - **Características**
  - **Funções**

Classes

- Os **objetos** são **classificados** de acordo com
  - **Características** semelhantes
  - **Funcionalidades** semelhantes

Relacion

- Associação
- Agregação
- Composição
- Herança

Encapsulamento

- Todo objeto é responsável pelos seus atributos
- Esconder do mundo externo:
  - Estrutura interna dos objetos
  - Detalhes de implementação
- Interação através de uma interface pública

Objetos

```kotlin
val pessoa = Pessoa("Batman", 32)

pessoa.correr()
pessoa.andar()
pessoa.imprimir()
pessoa.fazerAniversario()
pessoa.imprimir()
```

Herança

- Estabelece a relação **é um** entre duas classes
- Permite
  - **Abstração**
  - **Reaproveitamento** de código

Polimorfismo

- Um objeto pode assumir **diferentes formas**
- Tipos de polimorfismo:
  - **por inclusão** via herança
  - **paramétrico** tipos genéricos
  - **sobrescrita** redefinição de métodos
  - **sobrecarga** métodos com mesmo nome, parâmetros diferentes

Herança

```kotlin
open class Funcionario(var nome: String, var matricula: Int) {
    fun imprimir() = println("[$matricula]: $nome")
}

class Programador(nome: String, matricula: Int) : Funcionario(nome, matricula) {
    fun programar() = println("$nome está programando...")
}

class Gerente(nome: String, matricula: Int) : Funcionario(nome, matricula) {
    fun gerenciar() = println("$nome está gerenciando...")
}
```

Interfaces

```kotlin
interface Imprimivel {
    fun imprime()
}

class Cliente(val nome:String, val sobrenome:String) : Imprimivel {
    override fun imprime() {
        println("$nome $sobrenome")
    }
}
```