**BASICS**

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

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

Single-expression function
```kotlin
fun sum(a: Int, b: Int) = a + b
```

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

Variables with nullable types
```kotlin
var name: String? = null
val length: Int
length = name?.length ?: 0 // length, or 0 if name is null
length = name?.length ?: return // length, or return when name is null
length = name?.length ?: throw Error() // length, or throw error when name is null
```

**CONTROL STRUCTURES**

If as an expression
```kotlin
fun bigger(a: Int, b: Int) = if (a > b) a else b
```

For loop
```kotlin
val list = listOf("A", "B", "C")
for (element in list) {
    println(element)
}
```

When expression
```kotlin
fun numberTypeName(x: Number) = when{x {
    0 -> "Zero" // Equality check
    in 1..4 -> "Four or less" // Range check
    in 5, 6, 7 -> "Five to seven" // Multiple values
    is Byte -> "Byte" // Type check
    else -> "Some number"
}
```

When expression with predicates
```kotlin
fun signAsString(x: Int) = when {
    x < 0 -> "Negative"
    x == 0 -> "Zero"
    else -> "Positive"
}
```

**CLASSES**

Primary constructor
```kotlin
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
```kotlin
open class Person(val name: String) {
    open fun hello() = "Hello, I am $name"
    // Final by default so we need open
}
```

```kotlin
class PolishPerson(name: String) : Person(name) {
    override fun hello() = "Dzień dobry, jestem $name"
}
```

Properties with assessors
```kotlin
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
        }
}
```

Data classes
```kotlin
data class Person(val name: String, var age: Int) {
    val mike = Person("Mike", 23)
}
```

Modifier data adds:
1. `toString` that displays all primary constructor properties
2. `equals` that compares all primary constructor properties
3. `hashCode` that is based on all primary constructor properties
4. `component1, component2` etc. that allows deconstruction
5. `copy` that returns copy of object with concrete properties changed

```kotlin
val jake = mike.copy(name = "Jake")
```
COLLECTION LITERALS

- `listOf(1,2,3,4) // List<Int>`
- `mutableListOf(1,2,3,4) // MutableList<Int>`

- `setOf("A", "B", "C") // Set<String>`
- `mutableSetOf("A", "B", "C") // HashSet<String>`

- `arrayOf(1, 2, 3)`

- `mapOf(1 to "A", 2 to "B") // Map<Int, String>`
- `mutableMapOf(1 to "A", 2 to "B") // MutableMap<Int, String>`

- `generateSequence(4) // Sequnce<Int>`
- `sequenceOf(1..10) // Sequence<Int>`

- `List(4) { it * 2 } // List<Int>`
- `generateSequence(4) { it + 2 } // Sequence<Int>`

COLLECTION PROCESSING

- `students`:
  - `.filter { it.passing && it.averageGrade > 4.0 }` (only passing students)
  - `.sortedByDescending { it.averageGrade }` (starting from ones with biggest grades)
  - `.take(10)` (take first 10)
  - `.sortedWith(compareBy { it.surname, it.name })` (sort by surname and then name)

- `generateSequence(0) { it + 1 }` (infinite sequence of next numbers starting on 0)
  - `.filter { it % 2 == 0 }` (keep only even)
  - `.map { it * 3 }` (triple every one)
  - `.take(100)` (take first 100)
  - `.average()` (average)

Most important functions for collection processing

- `val l = listOf(1,2,3,4)`
  - `.filter { it % 2 == 0 }` (returns elements matched by predicate)
  - `.map { it * 2 }` (returns elements after transformation)
  - `.flatMap` (returns elements yielded from results of transformation)
  - `.fold()`, `.reduce()` (accumulate elements)
  - `.forEach()` (performs an action on every element)

EXTENSION FUNCTIONS TO ANY OBJECT

<table>
<thead>
<tr>
<th>Reference to receiver</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>it</code></td>
<td>also</td>
</tr>
<tr>
<td><code>this</code></td>
<td>apply</td>
</tr>
</tbody>
</table>

- `val l = Dialog().apply {
  title = "Dialog title"
  onClick { print("Clicked") }
}`
FUNCTIONS

Function types

()` -> Unit` - takes no arguments and returns nothing (Unit).
`(Int, Int) -> Int` - takes two arguments of type Int and returns Int.
`() -> Unit` - takes another function and returns Unit.
`(Int) -> Int` - takes argument of type Int and returns function.

Function literals

`val add: (Int, Int) -> Int = { i, j -> i + j }` // Simple lambda expression

`val printAndDouble: (Int) -> Int = { println(it) // When single parameter, we can reference it using `it` it * 2 // In lambda, last expression is returned }

// Anonymous function alternative
`val printAndDoubleFun: (Int) -> Int = fun(i: Int): Int { println(i) it * 2 // Needs return like any function }`

`val i = printAndDouble(10) // 10 print(i) // 20`

Extension functions

`fun Int.isEven() = this % 2 == 0 println(2.isEven()) // true`

`fun List<Int>.average() = 1.0 * sum() / size println(listOf(1, 2, 3, 4).average()) // 2.5`

DELEGATES

Lazy - calculates value before first usage

`val i by lazy { print("init "); 10 } print(i) // Prints: init 10 print(i) // Prints: 10`

notNull - returns last setted value, or throws error if no value has been set

observable/vetoable - calls function every time value changes. In vetoable function also decides if new value should be set.

`var name by observable("Unset") { p, old, new -> println("$p.name changed $old -> $new") } name = "Marcin" println("name changed Unset -> Marcin"

Map/MutableMap - finds value on map by property name

`val map = mapOf("a" to 10) val a by map { println(it) it * 2 // In lambda, last expression is returned }

// Anonymous function alternative
`val printAndDoubleFun: (Int) -> Int = fun(i: Int): Int {
println(i) it * 2 // Needs return like any function
}

print(a) // Prints: 10`

fun Int.isEven() = this % 2 == 0

print(2.isEven()) // true

fun List<Int>.average() = 1.0 * sum() / size

print(listOf(1, 2, 3, 4).average()) // 2.5

VISIBILITY MODIFIERS

<table>
<thead>
<tr>
<th>Modifier</th>
<th>Class members</th>
<th>Top-level</th>
</tr>
</thead>
<tbody>
<tr>
<td>Public (default)</td>
<td>Visible everywhere</td>
<td>Visible everywhere</td>
</tr>
<tr>
<td>Private</td>
<td>Visible only in the same class</td>
<td>Visible in the same file</td>
</tr>
<tr>
<td>Protected</td>
<td>Visible only in the same class and subclasses</td>
<td>Not allowed</td>
</tr>
<tr>
<td>Internal</td>
<td>Visible in the same module if class is accessible</td>
<td>Visible in the same module</td>
</tr>
</tbody>
</table>

VARIANCE MODIFIERS

**Invariance**

class Box `<T>`

Box `Number`

Box `<Number>`

Box `<Number>`

**Covariance**

class Box `<out T>`

Box `<Number>`

Box `<Number>`

Box `<Int>`

**Contravariance**

class Box `<in T>`

Box `<Number>`

Box `<Number>`

Box `<Int>`

Learn Kotlin with us: www.kt.academy