Lab on apps development for tablets, smartphones and smartwatches

Week 0:
Course presentation & Introduction to Kotlin

Giovanni Ansaloni

Dimitra Tatli, Riselda Kodra, Yuxuan Wang
Qunyou Liu, Amirhossein Shahbazinia, Christodoulos Kechris

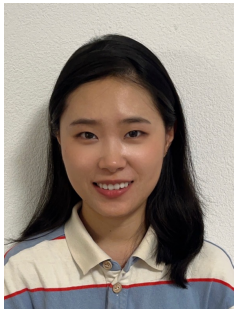
School of Engineering (STI) – Institute of Electrical and Micro Engineering (IEM)

Who

- Giovanni Ansaloni (giovanni.ansaloni@epfl.ch)

- Teaching Assistants
  - Dimitra Tatli (dimitra.tatli@epfl.ch)
  - Riselda Kodra (riselda.kodra@epfl.ch)
  - Yuxuan Wang (yuxuan.wang@epfl.ch)
  - Christodoulos Kechris (christodoulos.kechris@epfl.ch)
  - Qunyou Liu (qunyou.liu@epfl.ch)
  - Amirhossein Shahbazinia (amirhossein.shahbazinia@epfl.ch)






What

Lab on apps development for tablets, smartphones and smartwatches

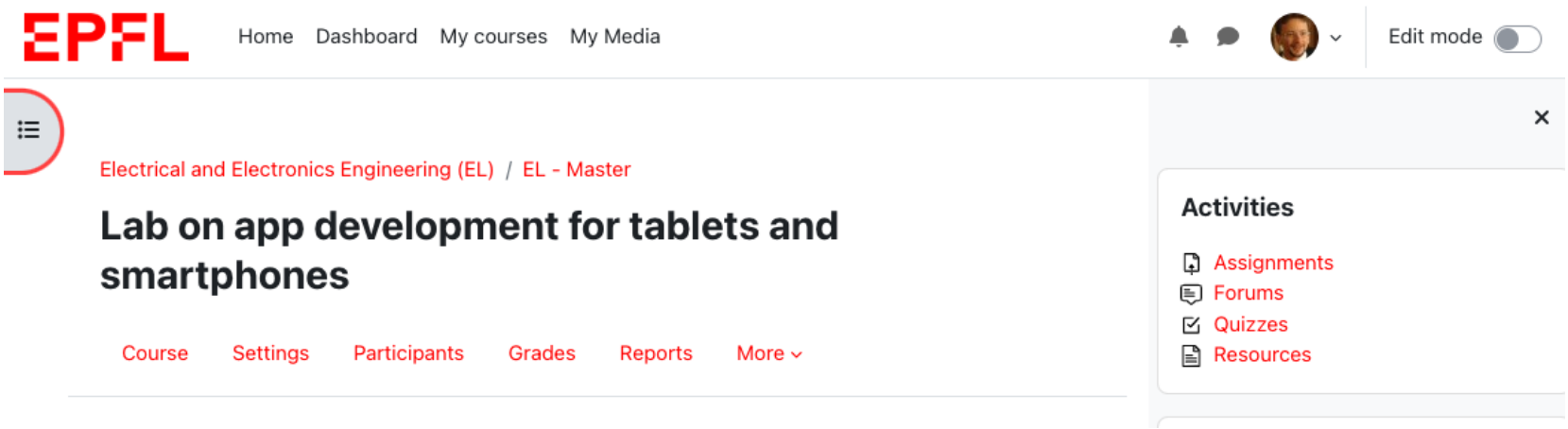
Course organization

- Classes: 4 hours per week – Tuesday, 14h to 18h
- First 9 weeks:
  - Lecture → 14h to 15h
    - 1 hour theory lesson to explain main concepts
  - Lab → 15h to 18h
    - 3 hours of practice of the concepts explained during the lectures, in groups of 3 students
    - solution available on Moodle before the start of the next lecture
- Next 5 weeks:
  - Midterm exam → (35% grade)
  - Development of projects in groups of 3 people → (65% grade)
- Apart from the 4-hour lab, you're supposed to devote another 4 hours per week
- Lectures+lab sessions provide you with the basis to develop your projects
  → but you can start project anytime you want!

Course organization

- Course material on Moodle
  - EE-490(g)

- Lecture slides
- Assignments handouts
- Assignments solutions
- Projects description
- News & discussion forums
- …



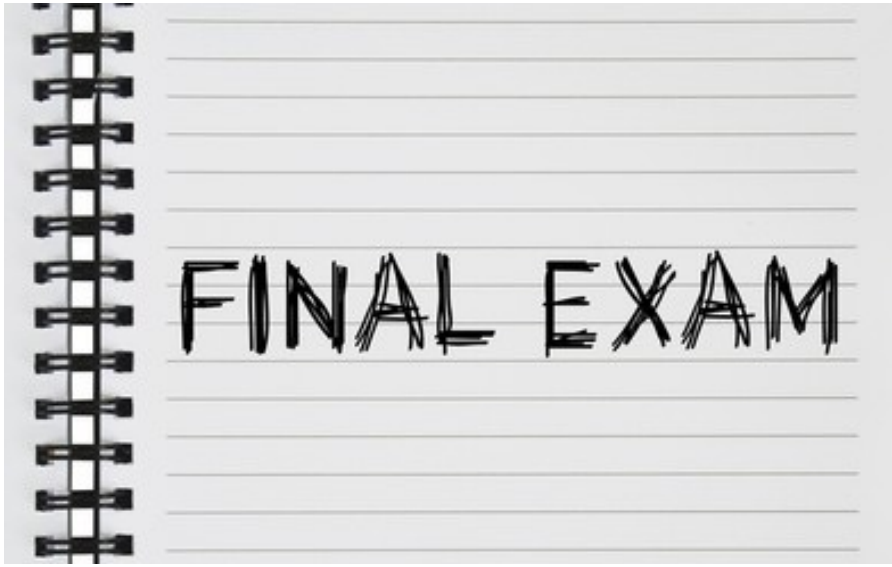
Course evaluation: Mid-term

- November 18th
- **35%** of the grade
- ~1h30m duration
- Individual, in class (using PC)

- 2 parts
  - mini-project to be completed
  - multiple-choices questions

- Content covered by lectures and labs
  → if you followed classes & did lab exercises, you will pass the exam

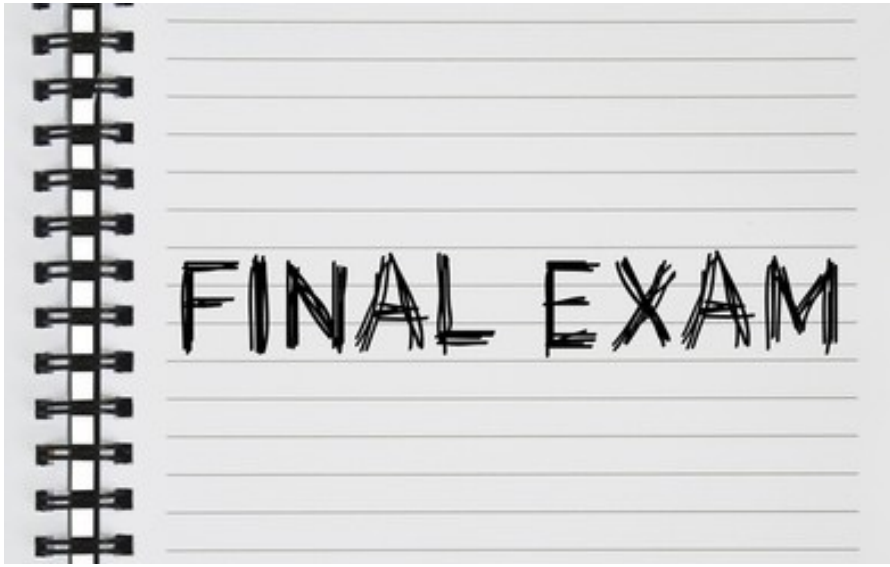
Course evaluation: Final exam



- Oral exam related to final project
  - ***65%*** of the grade
  - Grade may vary among members of the same group
  - Duration of the exam per group: 25 minutes
    - Including app demo
  - Documentation to upload via Moodle, **7 days** before the exam date:
    - Slides of the presentation of the project (typically 8-10 slides)
    - Working code of the project
    - *Video* (optional)

Course evaluation: Final exam

- Final grade determined by
  - Developed app
    - compliance to specification
    - features above minimum requirements
    - UI look&feel, user friendliness
    - code quality
    - difficulty of the project
  - Presentation
  - Individual assessment
    - % of project developed
    - understanding of implemented features
    - Q&A



Lectures: Outline of the course

- 9 weeks of theory/lab sessions (+ 5 weeks for projects)

0. Course presentation. Introduction to Kotlin.
1. Android overview. Defining a GUI.
2. Dynamic applications: State and interactivity.
3. Complex GUIs: Screens and menus.
4. Apps under the hood: Life cycles
   Communication between Android and AndroidWear devices: Wear APIs.
5. Separating concerns: UI controllers and viewModels.
   Interfacing with sensors: System Services.
6. Interfacing with the cloud: Firebase.
   Displaying structured data: Lists.
7. Local databases: Room library.
   Integrating Google maps.
8. Bluetooth Low Energy.

- Android courses:
https://developer.android.com/courses

- Android Basics with Compose:
https://developer.android.com/courses/android-basics-compose/course

Material used for labs & project

- Programming using several devices:
  - Tablet: Huawei MediaPad T3 10
    - Or your Android phone, if you prefer
  - Huawei Watch Sport 2
  - Interacting with external peripherals
    (Geonaute Dual HR band, Parrot Anafi drones, etc.)

- You'll be given the required material
  - you need to give it back by the end of the course

Material used for labs & project

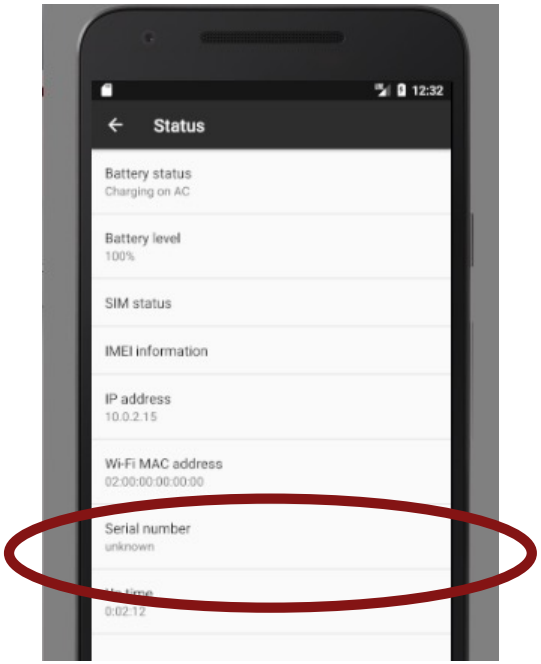
- Each group of three students receives a Tablet, a Smartwatch and a HR sensor at the start of the course
  - today after the lecture
  - ...or during the upcoming lab hours
- *Form your team as soon as possible!*

- Further material assigned depending on the chosen project

Filling-in the Lending Material Form

- Students in a group are **fully and equally responsible** for the material.

- Serial Number:
  - EPFL Inventory number sticker (if available)
  - Serial number of the tablet/watch:
    - Setting → About Phone → Status → **Serial Number**
    - ...or printed on the box



Projects

- Project in groups of 3 students
  - We propose several topics for projects
    - We ensure these projects cover the objectives in the course
  - If you want to develop your own project, discuss objectives with me

- Teams need to request a project before **November 11th** at the latest
  - Register your group and project preference using Google Forms
    - Form opens on **Tuesday *16th* at 15:00h → Link sent by Moodle**
  - Please state 1st/2nd/3rd choice
  - If you don't pick a group and project, you'll be assigned one

- You can start working on the project before theory/labs finish
  - As soon as you have confirmation that it has been assigned to you
  - We will provide a gitLab sub-repository for each group → *upload regularly!*

When

- Lectures + labs
- Project
- Choose project
- Mid-term
- Tentative exam dates
- Meet-up
- Return material

September 2025

Mon	Tue	Wed	Thu	Fri	Sat	Sun
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30					

October 2025

Mon	Tue	Wed	Thu	Fri	Sat	Sun
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31		

November 2025

Mon	Tue	Wed	Thu	Fri	Sat	Sun
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30

December 2025

Mon	Tue	Wed	Thu	Fri	Sat	Sun
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

January 2026

Mon	Tue	Wed	Thu	Fri	Sat	Sun
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	

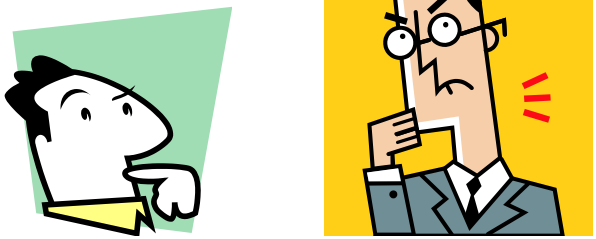
February 2026

Mon	Tue	Wed	Thu	Fri	Sat	Sun
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	

Meet-up

- Let’s demo them in a meet-up at the beginning of next semester
  - Tentatively: February 27th, room ELG123
- Groups can showcase their projects
  - to each other
  - to their friends/colleagues
  - …
- Snacks and prizes!

Questions?



Class outline

- Course presentation
- Projects
- Introduction to Kotlin
- Today's lab

Why Kotlin (and not Java)?

- Android applications are developed using ~~Java~~ → ***Kotlin***

- Since 2019, Kotlin is the recommended language for Android
  - new tools/content, documentation and training Kotlin-first

- Good news: Kotlin is easy to learn
  (especially if you know OOP already)
  - Concise and expressive
  - Android studio will help you a lot

- Extensive documentation here:
  https://developer.android.com/courses/kotlin-bootcamp/overview

Kotlin in a nutshell

- Kotlin can be used for:

- Functional programming

```
val messages = listOf(
    "Hey! Where are you?",
    "Everything going according to plan today?",
    "Please reply. I've lost you!"
)

fun main() {
    val senders = messages
    println(messages[1])
}
```

- Object-orientated programming

```
class Person(val name: String) {
    fun greet() = println("It's me, $name.")
}

fun main() {
    val sam = Person("Sam")
    sam.greet()
}
```

- *We will use both!*

Kotlin as functional language

- Based on functions
  - modules of code that accomplish a specific task

```
fun feedFish (hungry : Boolean) : String {
    var food: String
    if (hungry) {
        food = ”yes”
    } else {
        food = “no”
    }
    return food
}

fun main(){
   ...
   val result = feedFish(true)
}
```

return type

arguments

local variable

Function call

Lambdas

- In Kotlin (almost) every expression has a value

```
var isHot = if (temperature > 50) true else false
```

- A lambda is an incapsulated expression
  - Last statement is the return value

```
var waterFilter : (Int) -> Int = { dirty : Int    ->   dirty / 2}
```

parameters

implementation

Inputs types (comma separated)

Output type

- Inputs and outputs type declarations are optional

```
var waterFilter = { dirty : Int    ->   dirty / 2}
```

lambda in curly braces

Lambdas

- Lambdas are commonly passed in-line as the last parameter
  → trailing lambda

```
fun main() {
  dirtyLevel = updateDirty(dirtyLevel) {dirty : Int   ->   dirty / 2}
}
```

lambda

```
fun updateDirty(dirtyLevel: Int, operation: (Int) -> Int): Int {
  return operation(dirtyLevel)
}
```

Higher order functions

- A higher-order function is a function that
  - takes functions (including lambdas) as parameters

  and/or
  - returns a function as result

lambda →

```
var waterFilter = { dirty : Int    ->   dirty / 2}

fun main(){
    dirtyLevel = updateDirty(dirtyLevel, waterFilter)
}

fun updateDirty(dirtyLevel: Int, operation: (Int) -> Int): Int {
  return operation(dirtyLevel)
}
```

higher order function

function that takes an Int and returns an Int

Higher order functions

- A higher-order function is a function that
  - takes functions (including lambdas) as parameters

  and/or
  - returns a function as result

function →

```
fun waterFilter(dirty: Int): Int { return dirty / 2 }

fun main(){
    dirtyLevel = updateDirty(dirtyLevel, ::waterFilter)
}

fun updateDirty(dirtyLevel: Int, operation: (Int) -> Int): Int {
  return operation(dirtyLevel)
}
```

Method reference

higher order function

function that takes an Int and returns an Int

Kotlin as an OOP Language

- OOP → Object Oriented Programming

- One with…
  - **Abstract data types**:
    - Classes → types
    - Objects → instances of a class
  - **Encapsulation**
    - Properties → characteristic of a class
    - Methods → functionality of a class
  - **Inheritance**

Kotlin Classes

- A class defines a type:
  - An abstraction represented as a set of features/members
    - Properties (aka Attributes/Fields/Instance variables)
    - Methods (aka Routines/(Member) functions)
  - A mold for all its objects

- Example:

Default values (constructor)

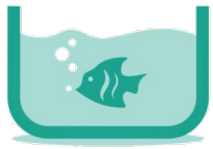
```
class Aquarium(length: Int = 100, width: Int = 20, height: Int = 40) {
    var length: Int = length
    var width: Int = width
    var height: Int = height

    fun printSize() {
        println("Width: $width cm " + "Length: $length cm " +  "Height: $height cm ")
    }
}
```

properties

Method

var can be assigned multiple times

val can only be assigned once



Constructors

- Kotlin allows to directly assign values to property in constructors
- Constructor code (other than default values assignments) in *init{}* block

```
class Aquarium (var length: Int = 100, var width: Int = 20, var height: Int = 40) {
    init {
        println("Aquarium Initializing")
    }

    fun printSize() {…}
    var hungryFishes : Boolean = false
    fun feedFish () { …}
}
```

Constructor code

Default values assigned to properties assigned in constructor

Methods

- Functions inside of a class
  - A functionality offered by a class

```
class careForFish (...){

  fun feedFish (hungry : Boolean) : String {
     var food: String
     if (hungry) {
          food = "yes"
     }else{
          food = "no"
     }
     return food
  }
...
}
```

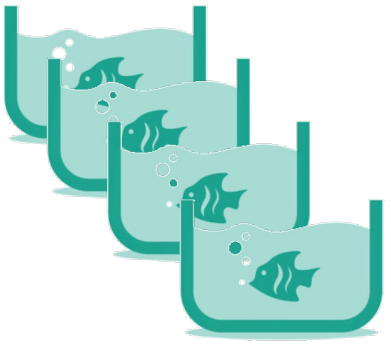
return type

arguments

local variable

Classes and objects

- Creating an objects, referencing properties/methods
  → client relationship



```
class Aquarium (var length: Int = 100,
                var width: Int = 20,
                var height: Int = 40) {

    init {... }
    fun printSize() {...}
    ...
}
```

```
fun buildAquarium() {
    val aquarium1 = Aquarium()
    aquarium1.printSize()
    // default height and length
    val aquarium2 = Aquarium(width = 25)
    aquarium2.printSize()
    // default width
    val aquarium3 = Aquarium(height = 35, length = 110)
    aquarium3.printSize()
}
```

Specialized classes

- An enum class that represents a group of constants

Class definition →

```
enum class Canton(val code: String) {
    VAUD("VD"),
    GENEVA("GE"),
    ...
}
```

Object initialization →

```
val canton = Canton.VAUD
...
val code = Canton.VAUD.code
```

- A data class are used to hold data
  - automatic generation of .equals(), .copy() methods

Class definition →

```
data class Person(
        val name: String,
        val age: Int
)
```

Object initialization →

```
val person = Person("John Doe", 25)
val samePerson = person.copy()

val isSamePerson = (person == samePerson)
...
val modifiedPerson = person.copy(age = 30)
```

Classes with type parameters

- Classes can leave the type of input parameters undetermined
  - usually, generic "T" type
  - type is resolved when initializing objects

Class definition →

```
class Box<T>(t: T) {
    var capacity = t
}
```

Object initialization →

```
val boxInt: Box<Int> =
                Box<Int>(1)

val boxFloat: Box<Float> =
                  Box<Float>(1.2f)
```

Object initialization (inferred types) →

```
val boxInt = Box(1)

val boxFloat = Box(1.2f)
```

Encapsulation

- Class attributes and methods can have different visibility/access levels
  - Public: visible outside the class.
    - Methods and attributes are public by default
  - Private: only visible in that class
  - Protected: same as private, but also visible by subclasses

```
public var length: Int = 30

private fun fishFood (hungry : Boolean) : String {...}
```

Getters and setters

- Getters and setters methods can (optionally) be defined for each property
  - Getter: called every time a property is accessed

```
var volume: Int
    get() = width * height * length / 1000
```

  - Setter: called every time a value is assigned to a property

```
var volume: Int
    set(value) {
        height = (value * 1000) / (width * length)
    }
```

Inheritance

- A class gets the properties/methods of another class by inheriting from it

- The derived class can
  - Introduce new features
  - Redefine features of the parent class but…
  - …only open Kotlin classes/methods/properties can be subclassed

    → everything closed by default

```
open class Aquarium (var length: Int = 100, var width: Int = 20, open var height: Int = 40) {

    open var volume: Int
        get() = width * height * length / 1000
        set(value) {
            height = (value * 1000) / (width * length)
        }
}
```

Inheritance

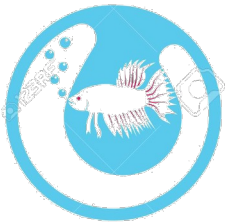
- Subclasses declaration

```
class TowerTank(override var height: Int, var diameter: Int): Aquarium(height = height, width = diameter,
                                                                       length = diameter) {

override var volume: Int
    get() = (width/2 * length/2 * height / 1000 * PI).toInt()
    set(value) {
        height = ((value * 1000 / PI) / (width/2 * length/2)).toInt()
    }
<OTHER PROPERTIES AND METHODS>
}
```

base class

- Objects of subclasses are created like any other objects

```
fun buildTowerTank() {
    val towerTank1 = TowerTank()
    println(towerTank1.volume)
}
```



Inheritance

- Abstract classes
  do not implement all methods
  - can only be sub-classed, they cannot be used to instantiate objects
  - implicitly “open”

```
abstract class AquariumFish {
    abstract val color: String
}
```

- Interfaces declare methods,
  but do not implement them
  - can’t have constructors
  - implicitly “open”

```
interface FishAction  {
    fun eat()
}
```

```
class Shark: AquariumFish(), FishAction {
    override val color = "gray"
    override fun eat() {
        println("hunt and eat fish")
    }
}
```

- Subclasses can inherit from one superclass and/or multiple interfaces

Nullability

- By default, properties in Kotlin cannot be Null

  `var rocks: Int = null` → *ERROR*

- '?' must be added to the type to indicate that a property is nullable

  `var rocks: Int? = null` → *OK*

- '?.' operator → tests for null value

  `fishFoodTreats = fishFoodTreats?.dec()`

- '?:' (Elvis) operator → Handles null cases

  `fishFoodTreats = fishFoodTreats?.dec() ?: 0`

- '!!' operator → Rise exception at run time if value is null (discouraged)

  `val len = s!!.length`

Scope functions

- Functions that only execute a block of code
  - applied to an object
  - followed by a lambda expression
  - forms a temporary scope within the object
    - Objects methods/variables accessed without specifying the object name

- let
- apply
- run
- with
- also

Complete documentation:
https://kotlinlang.org/docs/scope-functions.html

Scope functions

- Functions that only execute a block of code
  - applied to an object
  - followed by a lambda expression
  - forms a temporary scope within the object
    - Objects methods/variables accessed without specifying the object name

- let
  - executes lambda on non-nullable object
- apply
- run
- with
- also

```
var name: String? = null
...
name?.let {
    val nameLength = it.length
    println(nameLength)
}
```

it refers to object inside of the scope

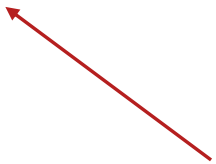
Only enter the scope if "*name*" is not null

Scope functions

- Functions that only execute a block of code
  - applied to an object
  - followed by a lambda expression
  - forms a temporary scope within the object
    - Objects methods/variables accessed without specifying the object name

- let
- apply
  - apply assignments to object
- run
- with
- also

```
val adam = Person("Adam").apply {
    this.age = 32
    this.city = "London"
}
println(adam)
```



"this" refers to object inside of the scope (optional)

Today's Lab – "Kotlin basics"

1. Creating your first Kotlin classes and objects
2. Basics of encapsulation
3. Inheriting from classes

Lab handout available on Moodle

Using Lab's Computers

1. Choose a computer for the whole semester

2. First Android Studio execution: run setup wizard
   - Use default configurations
   - Cancel the admin password request

Using your own computer

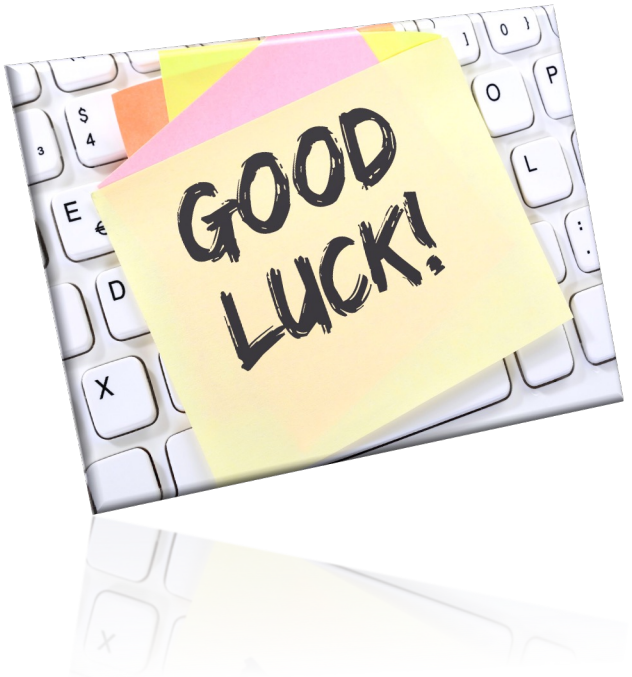
Android Studio is freely available for download here:

https://developer.android.com/studio/archive

IMPORTANT: we will use version

2024.3.1 Patch 2 (Meerkat) – April 2025

- same version as installed in lab desktops
- in general, Android Studio versions are ***not*** back-compatible



Detailed instructions in the handout