# Mobile Apps Development

# COMP-304
# Fall 2018

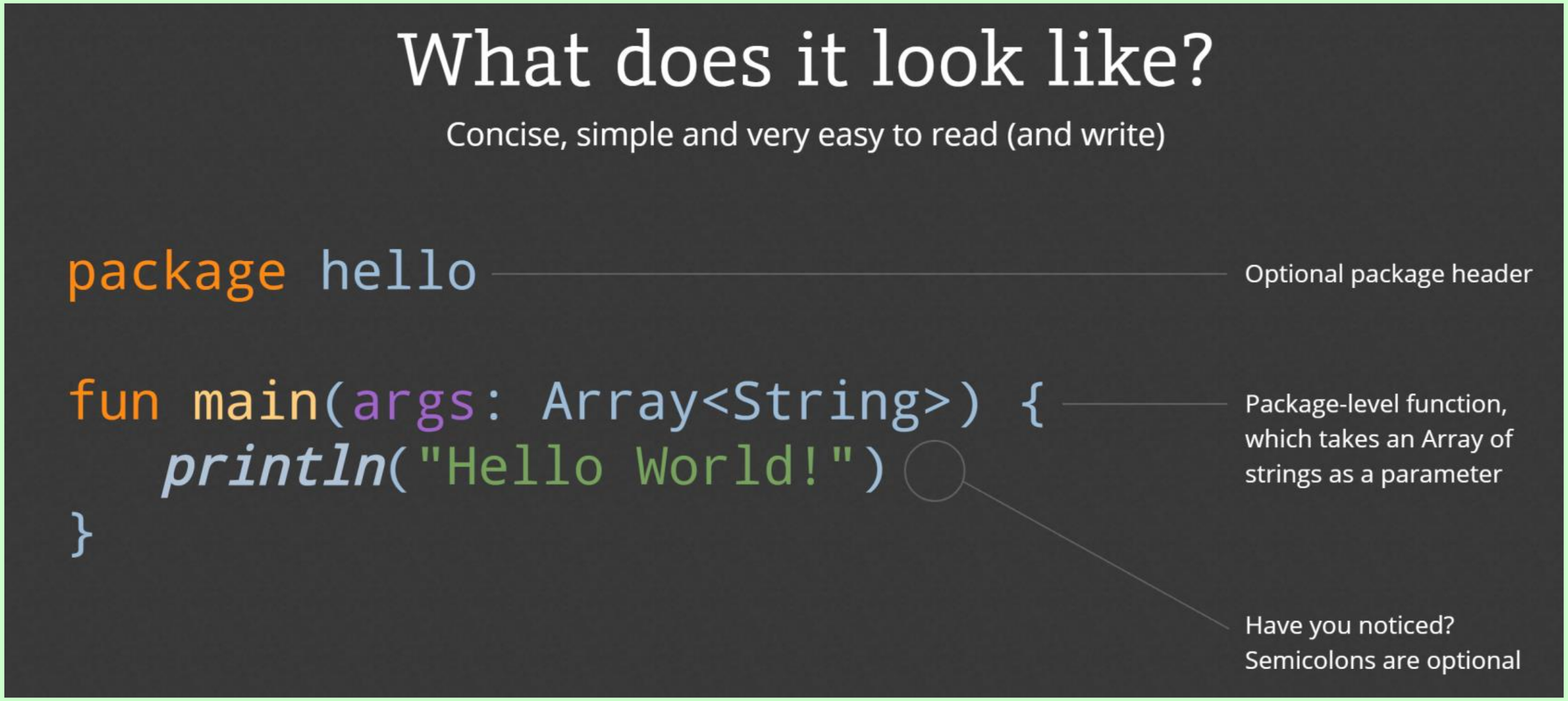
## Introduction to Kotlin

**Objectives:**

- ❑ **What's Kotlin?**
- ❑ **Android and Kotlin**
- ❑ **Developing Android Apps Using Kotlin**

# What is Kotlin

- Statically typed programming language for modern multiplatform applications - **type** checking is done at compile-time
- 100% interoperable with Java and Android



What does it look like?

Concise, simple and very easy to read (and write)

```kotlin
package hello

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

Optional package header
Package-level function, which takes an Array of strings as a parameter
Have you noticed? Semicolons are optional

## Basic Syntax

- Package specification should be at the top of the source file:

```
package my.demo

import java.util.*

// ...
```

- It is not required to match directories and packages: source files can be placed arbitrarily in the file system.

# Defining Functions

- Function having two Int parameters with Int return type:

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

- Function with an expression body and inferred return type:

```
fun sum(a: Int, b: Int) = a + b
```

- Function returning no meaningful value:

```
fun printSum(a: Int, b: Int): Unit {
    println("sum of $a and $b is ${a + b}")
}
```

- Unit return type can be omitted:

```
fun printSum(a: Int, b: Int) {
    println("sum of $a and $b is ${a + b}")
}
```

## Defining variables

- Assign-once (read-only) local variable:

```
val a: Int = 1  // immediate assignment
val b = 2   // `Int` type is inferred
val c: Int  // Type required when no initializer is provided
c = 3       // deferred assignment
```

- Mutable variable:

```
var x = 5 // `Int` type is inferred
x += 1
```

- Top-level variables:

```
val PI = 3.14
var x = 0

fun incrementX() {
    x += 1
}
```

## Comments

- Just like Java and JavaScript, Kotlin supports end-of-line and block comments.

```
// This is an end-of-line comment


/* This is a block comment
   on multiple lines. */
```

- Unlike Java, block comments in Kotlin can be nested.

## Using string templates

```
var a = 1
// simple name in template:
val s1 = "a is $a"

a = 2
// arbitrary expression in template:
val s2 = "${s1.replace("is", "was")}, but now is $a"
```

## Using conditional expressions

```
fun maxOf(a: Int, b: Int): Int {
    if (a > b) {
        return a
    } else {
        return b
    }
}
```

- Using if as an expression:

```
fun maxOf(a: Int, b: Int) = if (a > b) a else b
```

# Using nullable values and checking for null

- A reference must be explicitly marked as nullable when null value is possible.
- Return null if str does not hold an integer:

```
fun parseInt(str: String): Int? {
    // ...
}
```

- Use a function returning nullable value:

```
fun printProduct(arg1: String, arg2: String) {
    val x = parseInt(arg1)
    val y = parseInt(arg2)
    // Using `x * y` yields error because they may hold nulls.
    if (x != null && y != null) {
        // x and y are automatically cast to non-nullable after null check
        println(x * y)
    }
    else {
        println("either '$arg1' or '$arg2' is not a number")
    }
}
```

## Using type checks and automatic casts

- The is operator checks if an expression is an instance of a type.
  - If an immutable local variable or property is checked for a specific type, there's no need to cast it explicitly:

```
fun getStringLength(obj: Any): Int? {
    if (obj is String) {
        // `obj` is automatically cast to `String` in this branch
        return obj.length
    }

    // `obj` is still of type `Any` outside of the type-checked branch
    return null
}
```

## Using a for loop

```
val items = listOf("apple", "banana", "kiwi")
for (item in items) {
    println(item)
}
```

or

```
val items = listOf("apple", "banana", "kiwi")
for (index in items.indices) {
    println("item at $index is ${items[index]}")
}
```

## Using a while loop

```
val items = listOf("apple", "banana", "kiwi")
var index = 0
while (index < items.size) {
    println("item at $index is ${items[index]}")
    index++
}
```

## Using when expression

```
fun describe(obj: Any): String =
when (obj) {
    1          -> "One"
    "Hello"    -> "Greeting"
    is Long    -> "Long"
    !is String -> "Not a string"
    else       -> "Unknown"
}
```

# Using ranges

- Check if a number is within a range using in operator:

```
val x = 10
val y = 9
if (x in 1..y+1) {
    println("fits in range")
}
```

- Check if a number is out of range:

```
val list = listOf("a", "b", "c")

if (-1 !in 0..list.lastIndex) {
    println("-1 is out of range")
}
if (list.size !in list.indices) {
    println("list size is out of valid list indices range too")
}
```

## Iterating over a range

```
for (x in 1..5) {
    print(x)
}
```

or over a progression:

```
for (x in 1..10 step 2) {
    print(x)
}
println()
for (x in 9 downTo 0 step 3) {
    print(x)
}
```

# Using collections

- Iterating over a collection:

```
for (item in items) {
    println(item)
}
```

- Checking if a collection contains an object using in operator:

```
when {
    "orange" in items -> println("juicy")
    "apple" in items -> println("apple is fine too")
}
```

## Using Lampda Expressions

- Using lambda expressions to filter and map collections:

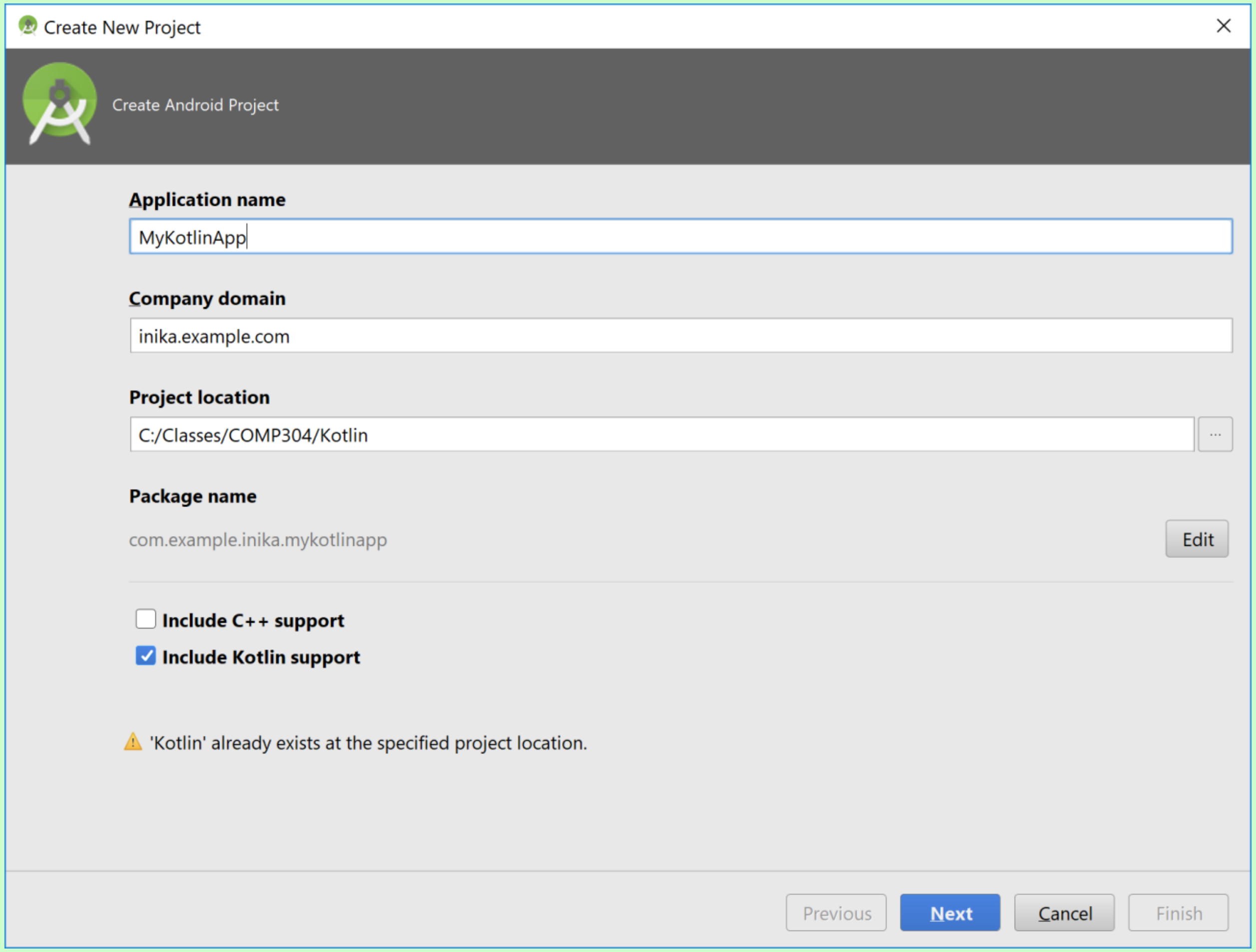
```
fruits
.filter { it.startsWith("a") }
.sortedBy { it }
.map { it.toUpperCase() }
.forEach { println(it) }
```

- Creating basic classes and their instances:

```
val rectangle = Rectangle(5.0, 2.0) //no 'new' keyword required
val triangle = Triangle(3.0, 4.0, 5.0)
```

# Build Your First Android App in Kotlin

❑ Create a new Project and include Kotlin support:



Application name
MyKotlinApp

Company domain
inika.example.com

Project location
C:/Classes/COMP304/Kotlin

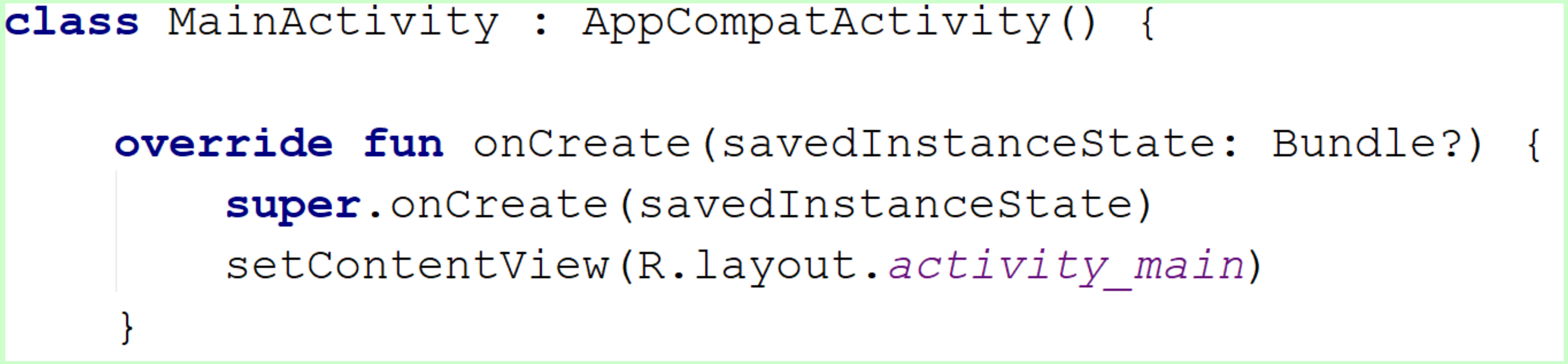
Package name
com.example.inika.mykotlinapp

☐ Include C++ support
☑ Include Kotlin support

⚠ 'Kotlin' already exists at the specified project location.

## Build Your First Android App in Kotlin

- Android Studio will generate the following Kotlin code:



```
class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
    }
```

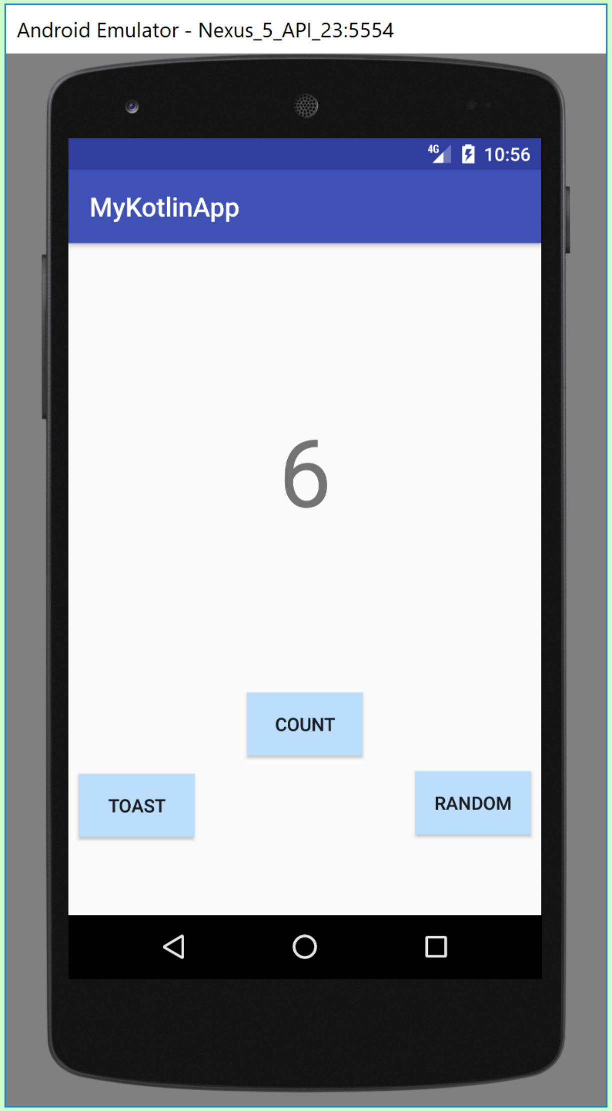
# Creating a function in Kotlin

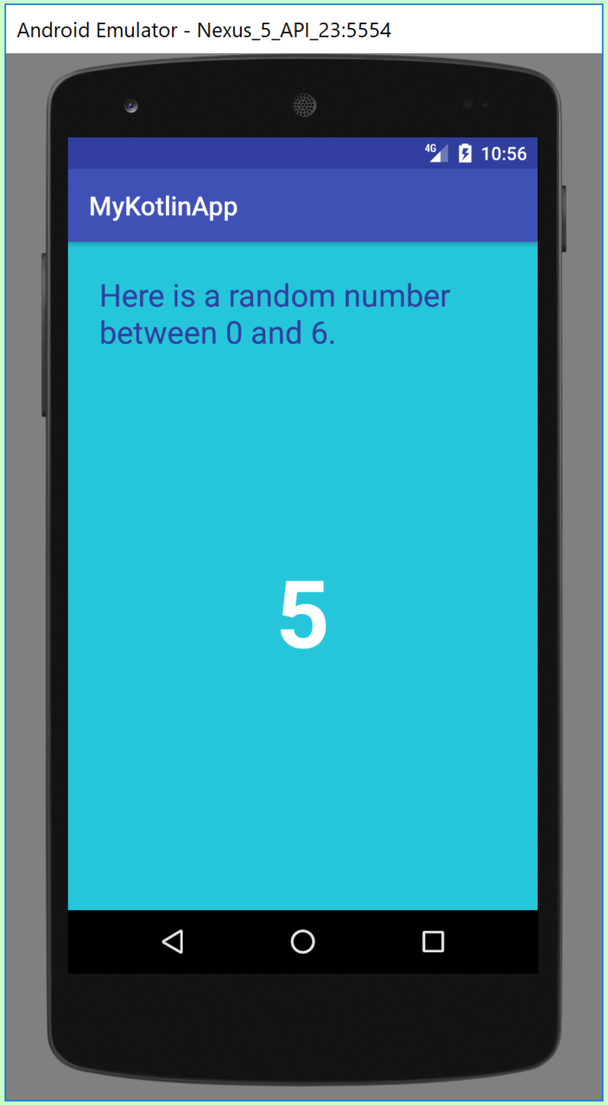
```
fun toastMe(view: View) {
        // val myToast = Toast.makeText(this, message, duration);
        val myToast = Toast.makeText(this, "Hello Toast!", Toast.LENGTH_SHORT)
        myToast.show()
    }

fun countMe (view: View) {

        // Get the value of the text view.
        val countString = textView.text.toString()

        // Convert value to a number and increment it
        var count: Int = Integer.parseInt(countString)
        count++

        // Display the new value in the text view.
        textView.text = count.toString();
    }
```

# Creating a function in Kotlin

```
fun randomMe (view: View) {
        // Create an Intent to start the second activity
        val randomIntent = Intent(this, SecondActivity::class.java)

        // Get the current value of the text view.
        val countString = textView.text.toString()

        // Convert the count to an int
        val count = Integer.parseInt(countString)

        // Add the count to the extras for the Intent.
        randomIntent.putExtra(SecondActivity.TOTAL_COUNT, count)

        // Start the new activity.
        startActivity(randomIntent)
    }
```

# Running the Project


Android Emulator - Nexus_5_API_23:5554


Android Emulator - Nexus_5_API_23:5554

Here is a random number between 0 and 6.

## References

- https://codelabs.developers.google.com/codelabs/build-your-first-android-app-kotlin/index.html#0
- https://kotlinlang.org/
- https://kotlinlang.org/docs/reference/basic-syntax.html
- https://kotlinlang.org/docs/reference/coding-conventions.html
- https://kotlinlang.org/docs/reference/basic-types.html
- https://kotlinlang.org/docs/reference/classes.html

- Android Documentation