Mobile Device Architecture And Kotlin Programming I

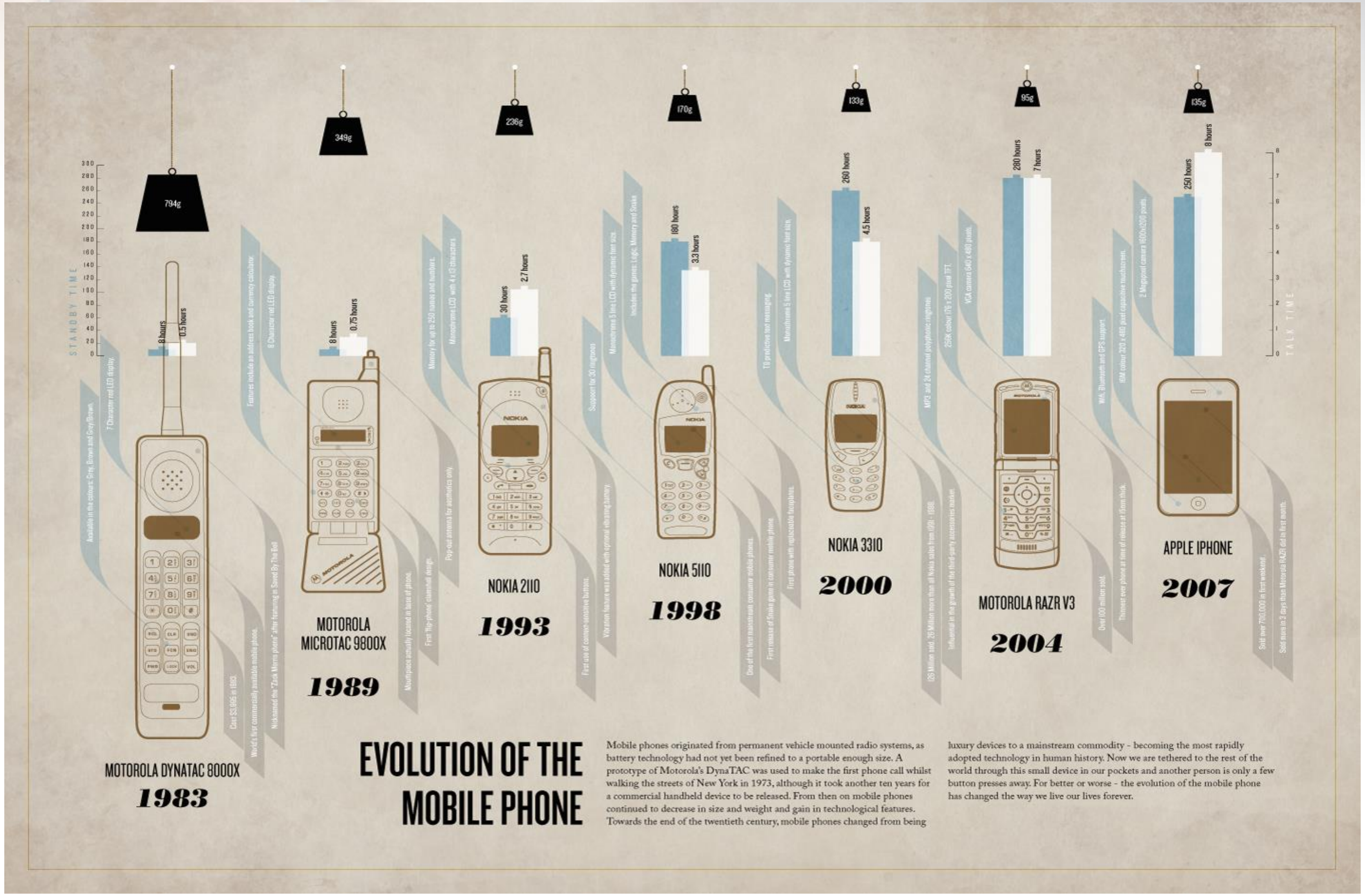
Asst. Prof. Monlica Wattana, Ph.D
Department of Computer Science,
Khon Kaen University

SC 362 007 Mobile Device Programming

Outline

- Evolution of Mobile
- Smartphone hardware architecture
- Mobile Development
- Android Operating System
- Kotlin Programming
  - Variable
  - Array
  - Operator
  - Type Conversion
  - If...Else
  - Loop
  - Function

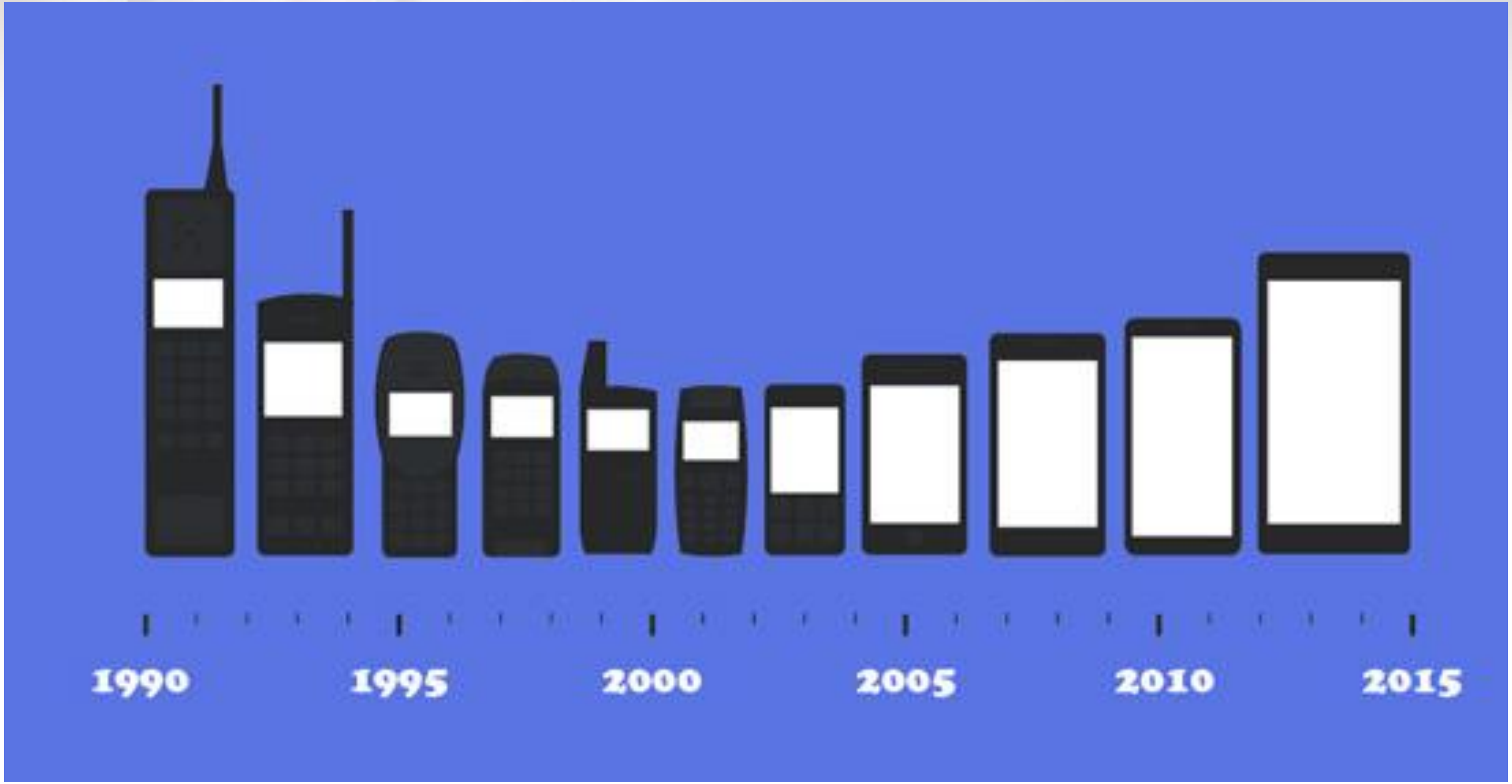
Evolution of Mobile Hardware



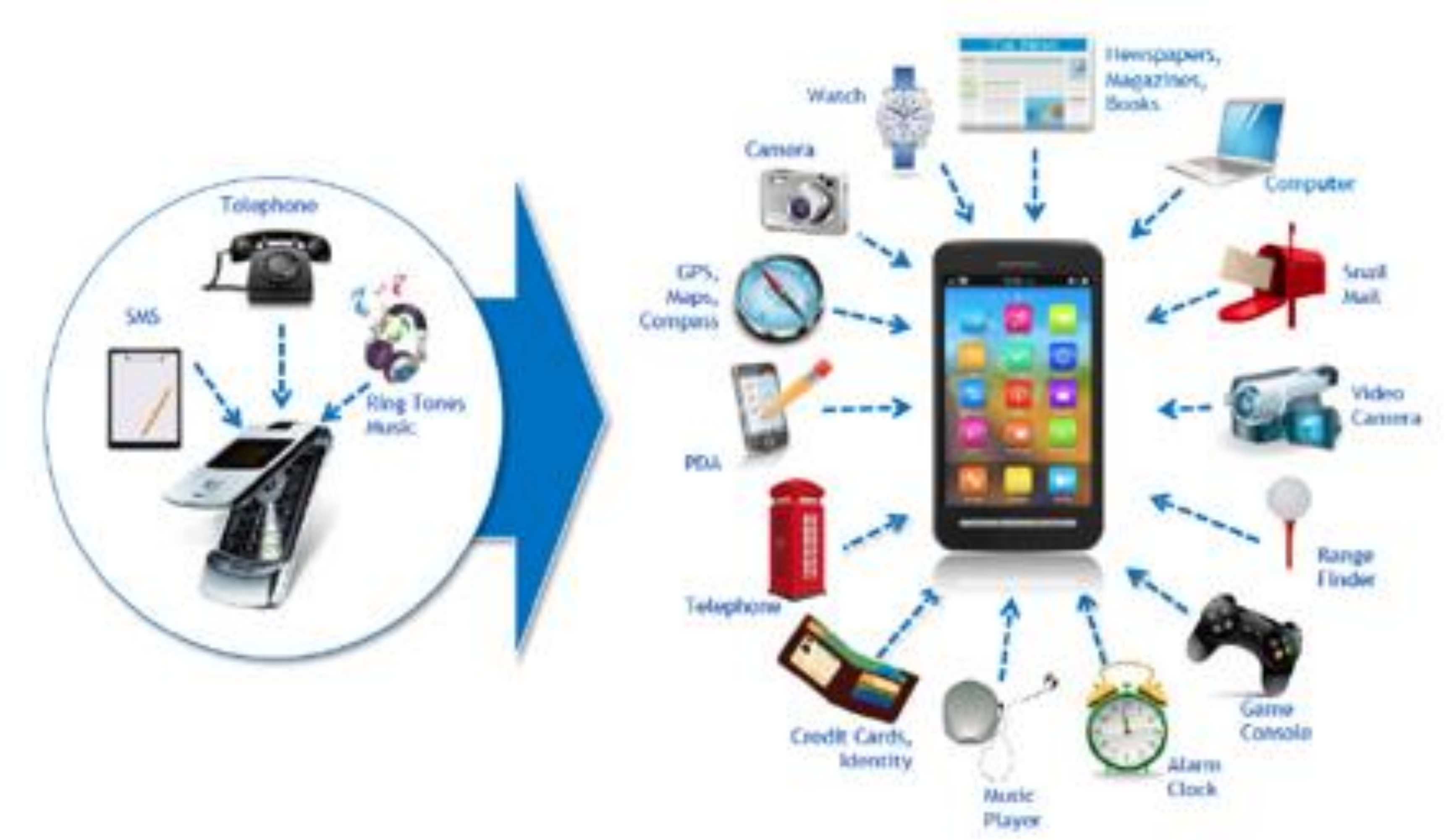
EVOLUTION OF THE MOBILE PHONE

Mobile phones originated from permanent vehicle mounted radio systems, as battery technology had not yet been refined to a portable enough size. A prototype of Motorola's DynaTAC was used to make the first phone call whilst walking the streets of New York in 1973, although it took another ten years for a commercial handheld device to be released. From then on mobile phones continued to decrease in size and weight and gain in technological features. Towards the end of the twentieth century, mobile phones changed from being luxury devices to a mainstream commodity - becoming the most rapidly adopted technology in human history. Now we are tethered to the rest of the world through this small device in our pockets and another person is only a few button presses away. For better or worse - the evolution of the mobile phone has changed the way we live our lives forever.

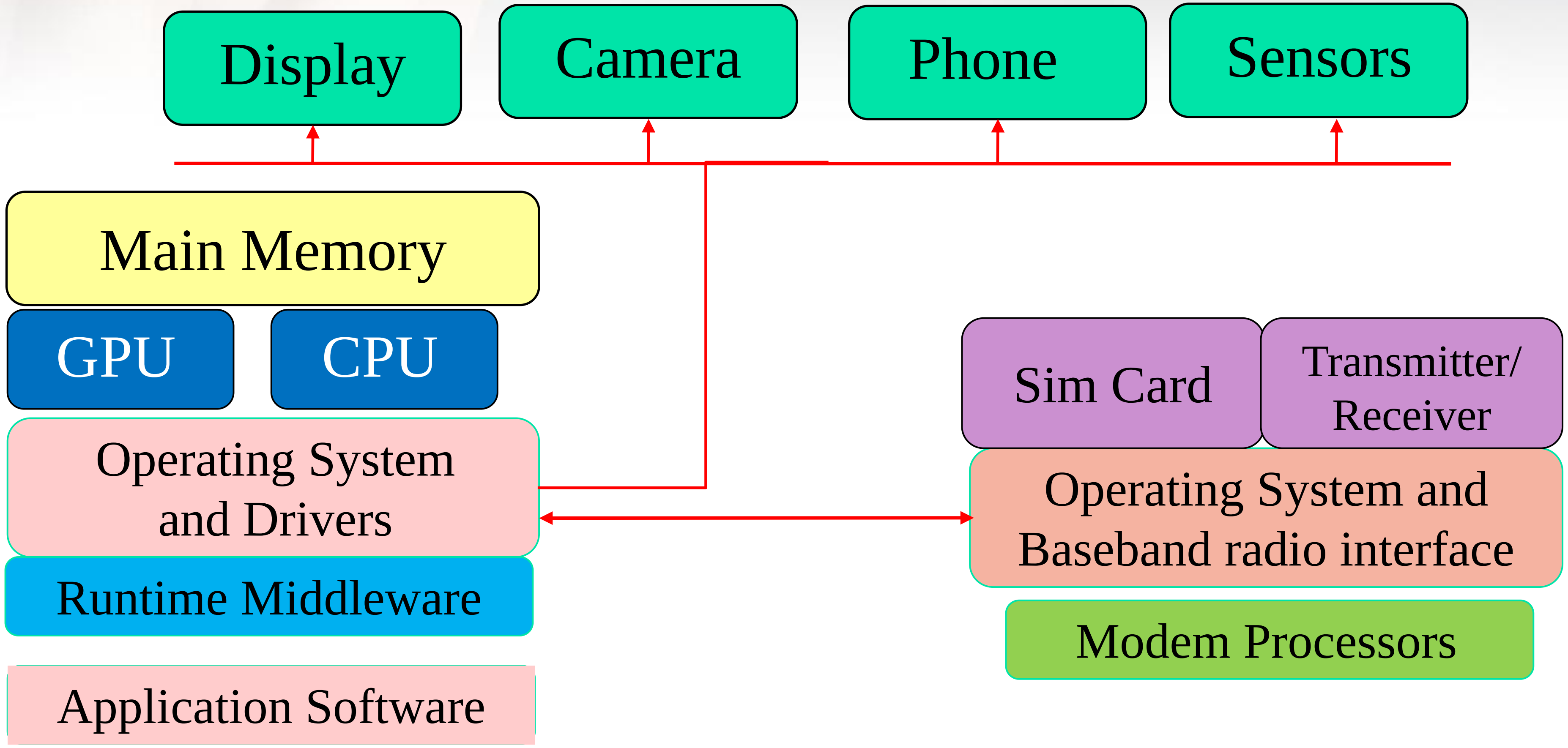
Evolution of Mobile Hardware



Evolution of Mobile Functionality/Software



Smartphone Hardware Architecture



Mobile Operating System

- Windows Phone

Windows Phone

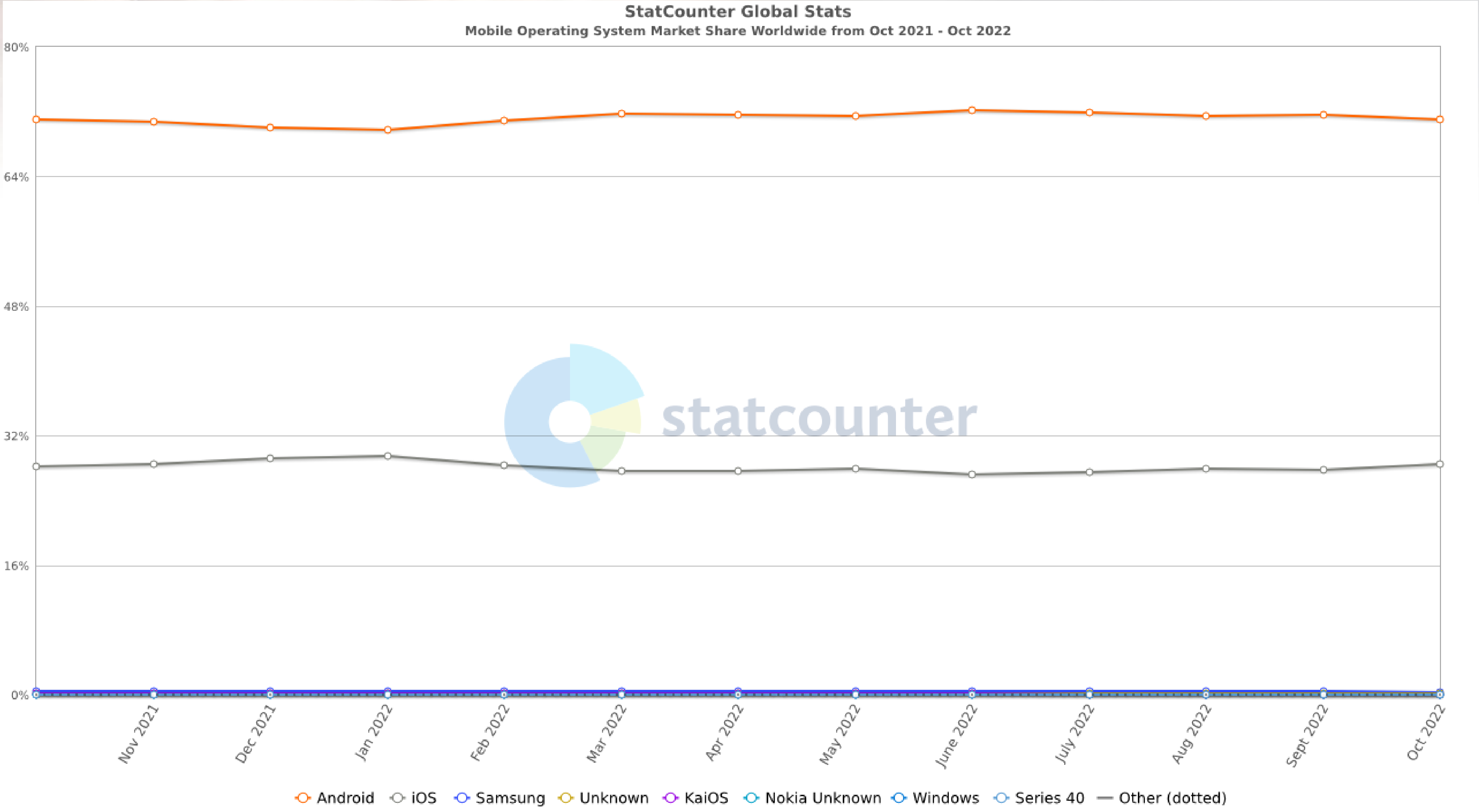
- iOS (iPhone OS): Apple Inc.

iOS

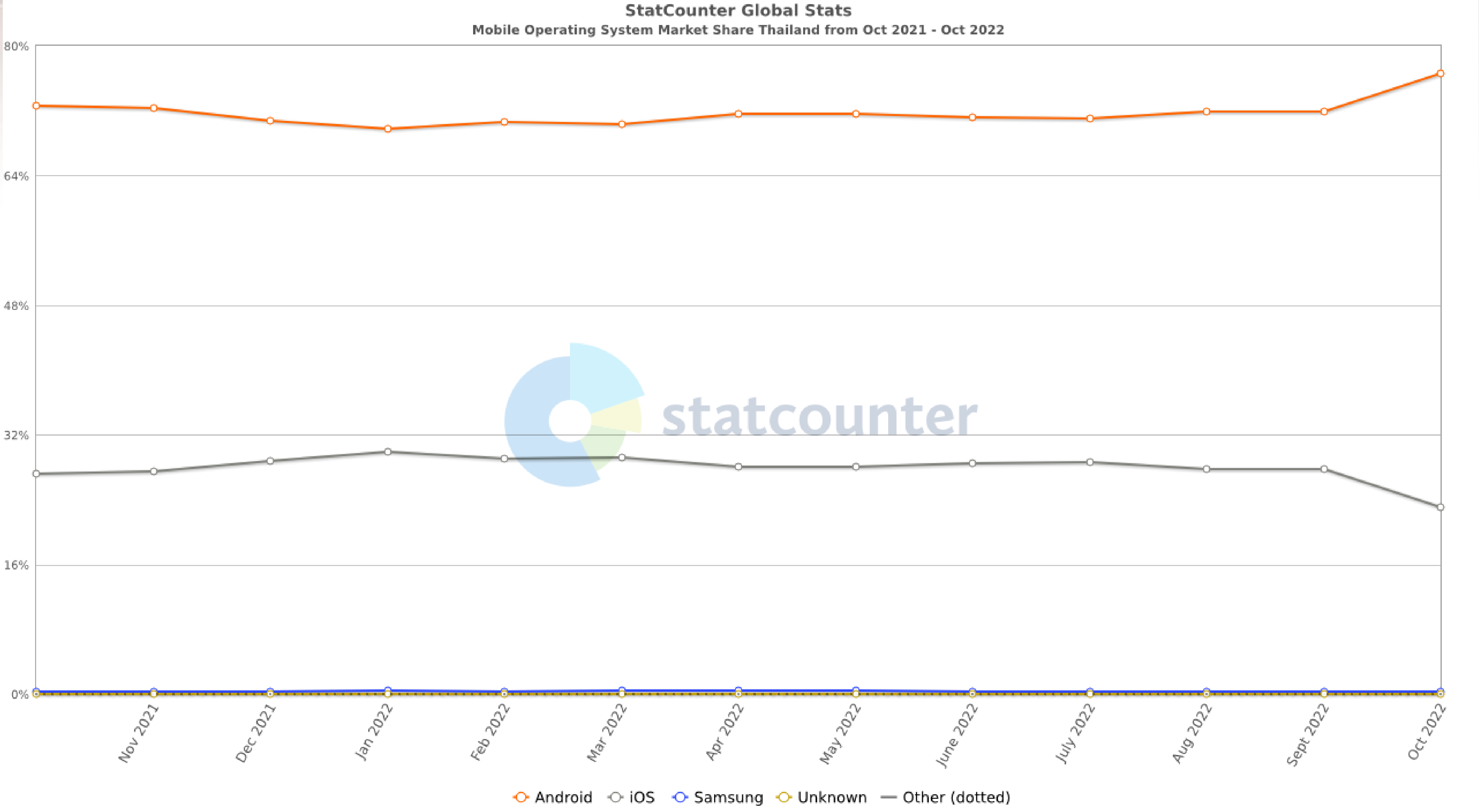
- Android: Google (Open Handset Alliance(OHA))

android

Mobile Operating System Market Share Worldwide



Mobile Operating System Market Share Thailand



StatCounter Global Stats

Mobile Operating System Market Share Thailand from Oct 2021 - Oct 2022

Mobile Development Language

- **Native development**

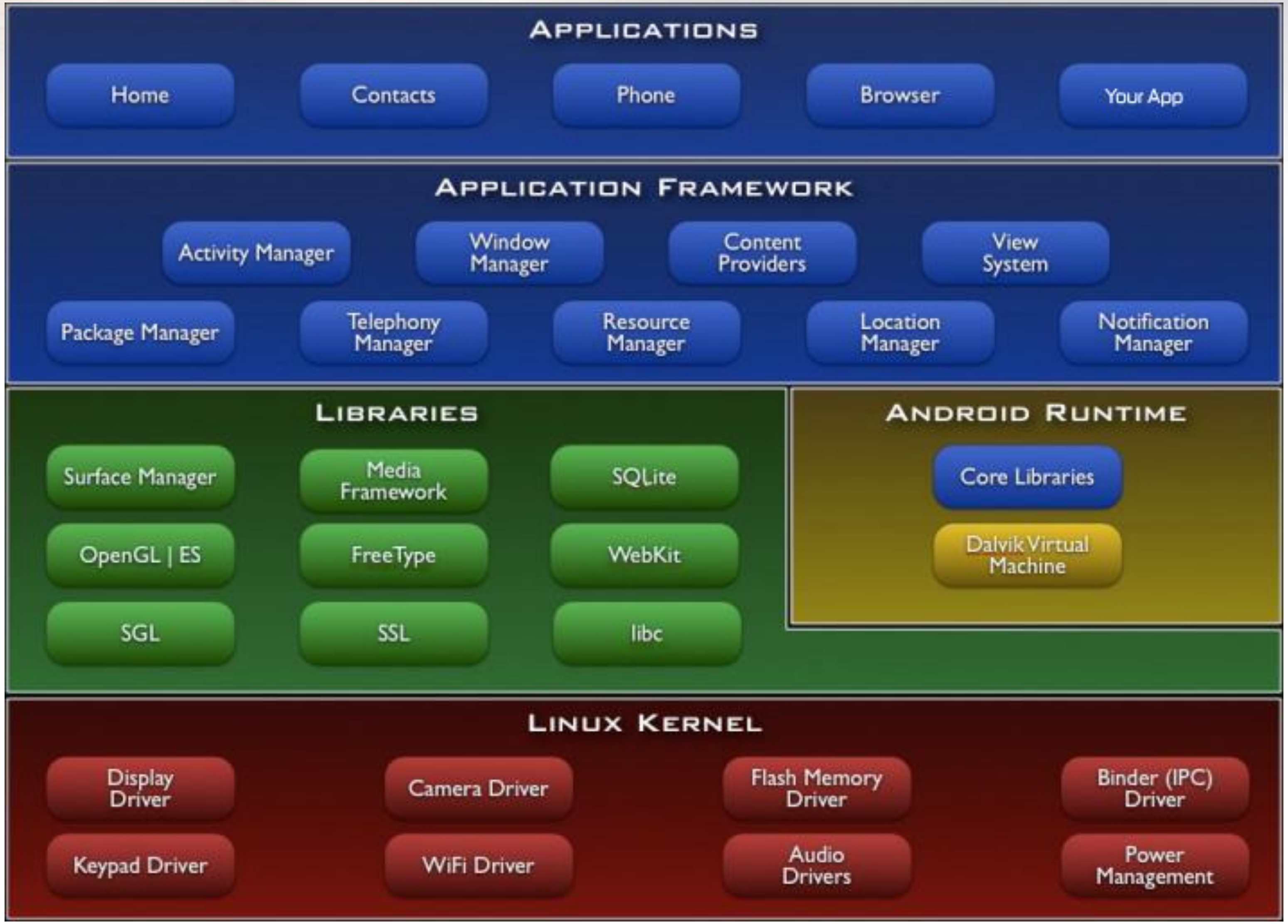
  - Android development: JAVA, Kotlin

  - iOS app development: Objective-C, Swift

- **Cross-platform development**

  React Native (JavaScript), Flutter (Google), Ionic, Xamarin (Microsoft)

Android Architecture



Applications

Home | Contacts | Phone | Browser | Your App

Application Framework

Activity Manager | Window Manager | Content Providers | View System

Package Manager | Telephony Manager | Resource Manager | Location Manager | Notification Manager

Libraries

Surface Manager | Media Framework | SQLite

OpenGL | ES | FreeType | WebKit

SGL | SSL | libc

Android Runtime

Core Libraries

Dalvik Virtual Machine

Linux Kernel

Display Driver | Camera Driver | Flash Memory Driver | Binder (IPC) Driver

Keypad Driver | WiFi Driver | Audio Drivers | Power Management

Application Framework

- Activity Manager android.activity

  - Manages the activity life cycle of applications
- Content Providers android.provider

  - Manage the data sharing between applications
- Telephony Manager android.telephony

  - Manages all voice calls.
- Location Manager android.location

  - Location management, using GPS or cell tower
- Resource Manager

  - Manage the various types of resources we use in our App

Libraries

- OpenGL ES android.opengl

  The OpenGL ES is a 3D graphics library.
- SQLite android.database.sqlite

  Contains the SQLite database management classes
- Media Framework

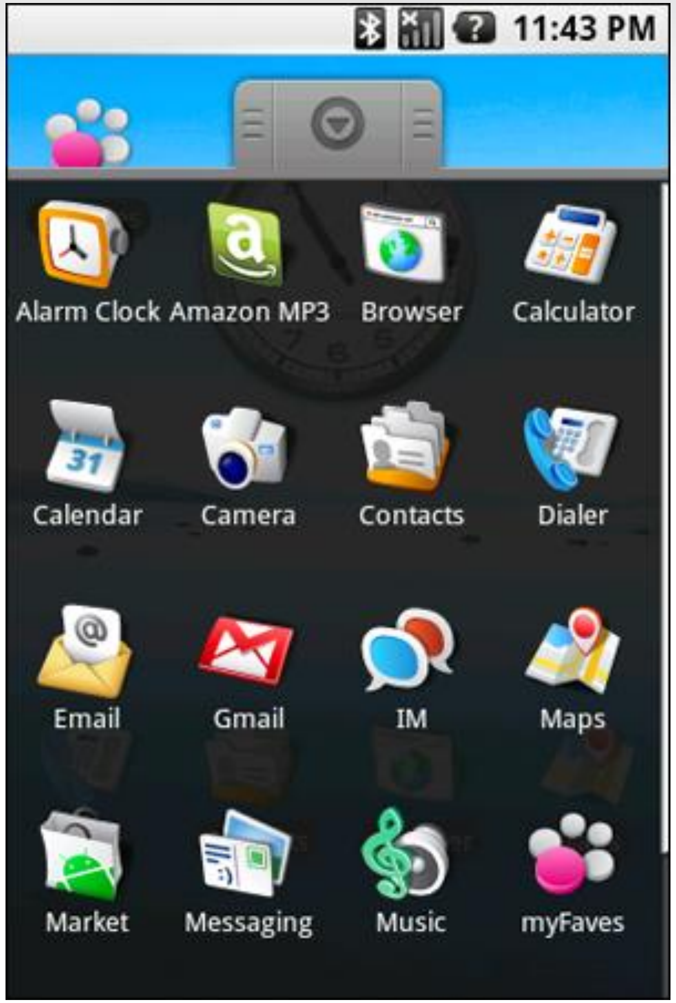
  The media framework contains all of the codecs that are required for multimedia experience.
- FreeType: used to render the fonts
- SSL: used for internet security
- WebKit: open source browser engine

Linux Kernel

- Based on Linux 2.6 kernel but Android is not Linux.
- Does not include the full set of standard Linux utilities
- No native windowing system
- Performs important power management activities
- Open Source: provide libraries to modify hardware drivers
- It is possible to make your own version of Android

First version of android

- Initial release: Oct. 2008
- NO on-screen keyboard
- NO multitouch capability
- NO paid apps
- The pull-down notification window
- Deep, rich Gmail integration
- Home screen widgets



Early versions of Android

- Version 1.5 cupcake and Version 1.6 Donut
  - An on-screen keyboard
  - Extensible widgets
  - Video capture and playback
- Version 2.1 – Version 2.3
  - multitouch capability
  - Support for front-facing cameras
  - Screen PIN protection
- Version 3.x
  - Targeted exclusively at tablets
  - No physical buttons
  - Improved multitasking

Recent versions of android

- Version 4.0 Ice Cream Sandwich
  - NFC support
  - Face unlock
  - Data usage analysis

- Version 4.1 to 4.3
  - Support panoramic image
  - Predictive text
  - support OpenGL ES 3.0
- Version 4.4 KitKat (SDK 19)
  - Full screen apps
  - Google Cloud Print
  - Improved Quickoffice app

Current versions of Android

- Version 5.0 – 5.1 Lollipop (SDK 22)

- Version 6.0 Marshmallow (SDK 23)

- Version 7.0 Nougat (SDK 24)

http://socialcompare.com/en/comparison/android-versions-comparison

Current versions of Android

- Version 8.0 Oreo (API 26)

- Version 9 Pie (API 28)

- Version 10 Android Q (API 29)

http://socialcompare.com/en/comparison/android-versions-comparison

Current versions of Android

- Version 11 (API 30) (2020)

- Version 12 (API 31) (2021)

- Version 13 (Beta 3) (release after July 2022)

http://socialcompare.com/en/comparison/android-versions-comparison

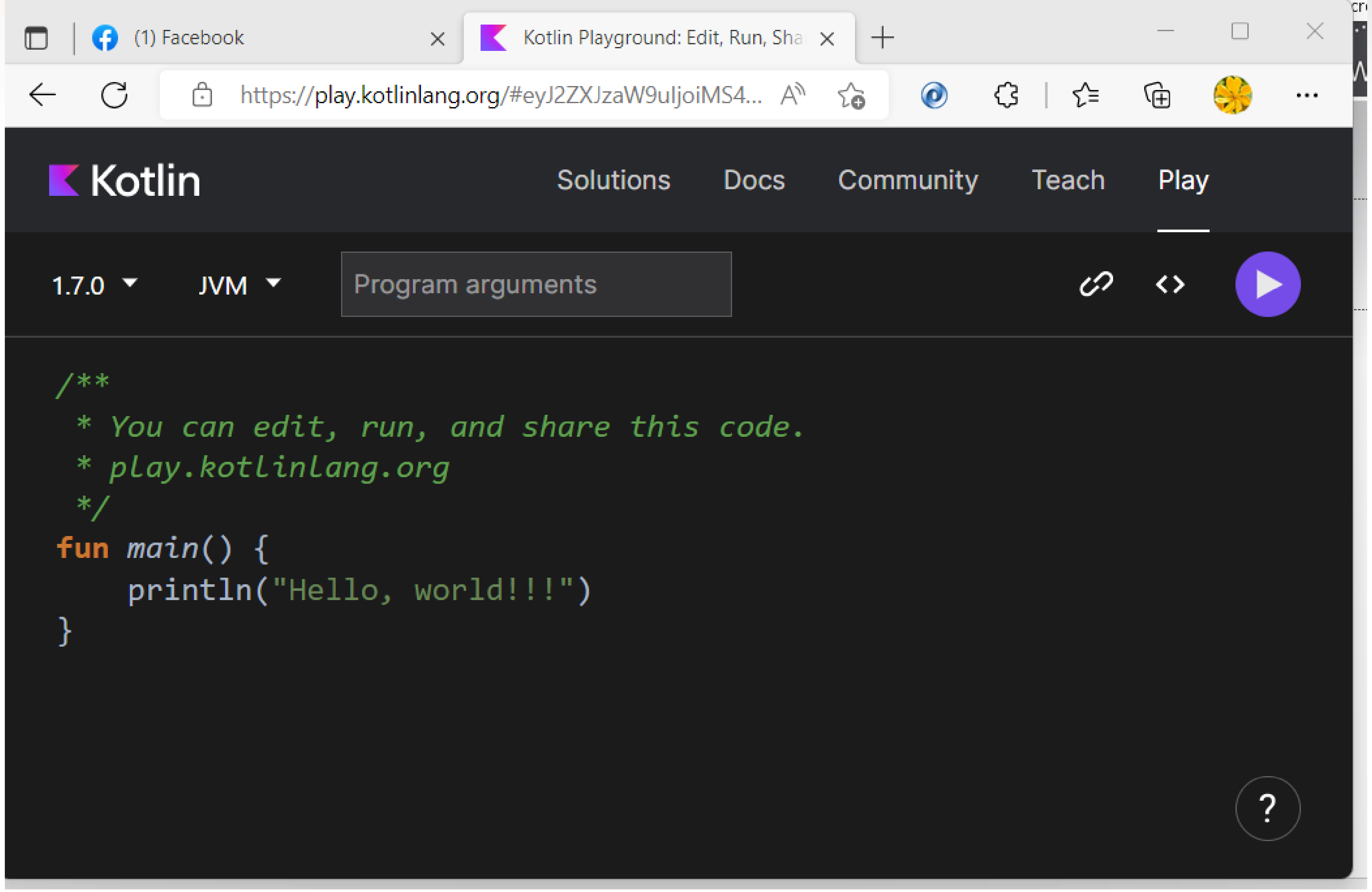
Introduction to Kotlin Programming

Kotlin

- Kotlin is a programming language introduced by JetBrains, the official designer of the most intelligent Java IDE, named IntelliJ IDEA.
- This is a strongly statically typed language that runs on JVM.
- In 2017, Google announced Kotlin is an official language for android development.
- Kotlin is an open source programming language that combines object-oriented programming and functional features into a unique platform.

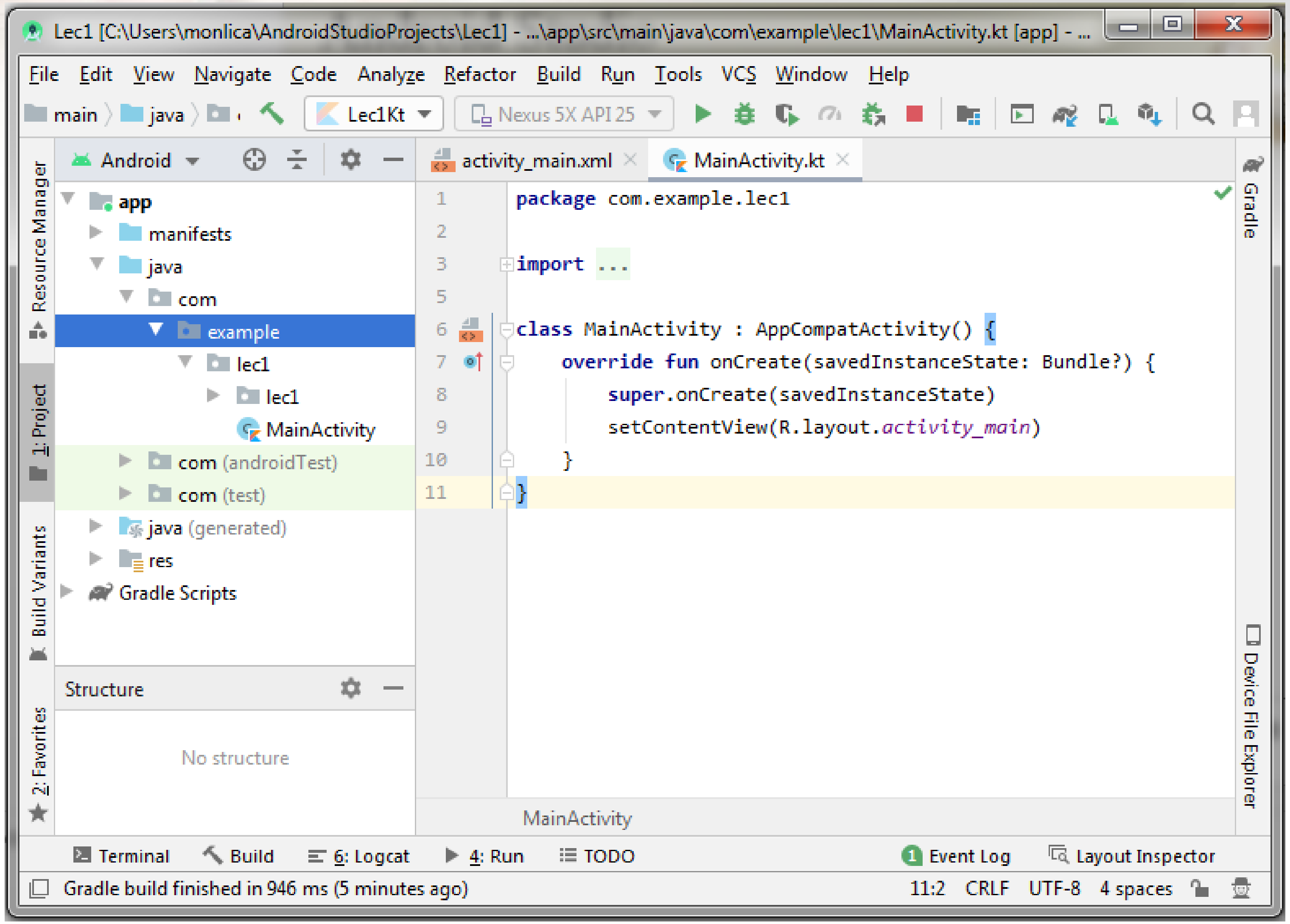
Try Kotlin Programming

https://try.kotlinlang.org


(1) Facebook
Kotlin Playground: Edit, Run, Sha
https://play.kotlinlang.org/#eyJ2ZXJzaW9uIjoiMS4...
Kotlin
Solutions
Docs
Community
Teach
Play
1.7.0
JVM
Program arguments
?

```
/**
 * You can edit, run, and share this code.
 * play.kotlinlang.org
 */
fun main() {
    println("Hello, world!!!")
}
```

Android Studio



```
package com.example.lec1

import ...

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

Gradle build finished in 946 ms (5 minutes ago)

Basic Syntax

- Defining packages

```
package my.demo

import java.util.*
```

- Defining variables

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

```
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
val myNull : Int? = null     //null values
```

Basic Syntax

- Defining variables

  Mutable variable:

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

Difference between var and val

- **var** (Mutable variable): We can change the value of variable declared using **var** keyword later in the program.

- **val** (Immutable variable): We cannot change the value of variable which is declared using **val** keyword.

Basic Syntax

Basic Types

Types	Instruction	Result

Numbers	val a: Int = 10000
	val d: Double = 100.00
	val f: Float = 100.00f
	val l: Long = 1000000004
	val s: Short = 10
	val b: Byte = 1
	
	println("Your Int Value is "+a)
	println("Your Double Value is "+d)
	println("Your Float Value is "+f)
	println("Your Long Value is "+l)
	println("Your Short Value is "+s)
	println("Your Byte Value is "+b)
		Your Int Value is 10000
		Your Double Value is 100.0
		Your Float Value is 100.0
		Your Long Value is 1000000004
		Your Short Value is 10
		Your Byte Value is 1

Basic Syntax

Basic Types

Types	Instruction	Result

Characters	val letter: Char // defining a variable
	letter = 'A' // assigning a value to it
	println("$letter")
		A

Boolean	val letter: Boolean // defining a variable
	letter = true // assinging a value to it
	println(“Character is "+"$letter")
		Character is true

Strings	var rawString :String = "I am Raw String!"
	val escapedString : String = "I am escaped String!\n"
	println("Hello!"+ escapedString)
	println("Hey!! $rawString")
		Hello!I am escaped String!
		
		Hey!!I am Raw String!

Basic Syntax

Basic Types

Types	Instruction	Result
Arrays	val anyThing = arrayOf(1, "A", 23.99) println("I am array example "+ anyThing[2]) val numbers= intArrayOf(1, 2, 3, 4, 5) println("I am int array example "+ numbers[2])	I am array example 23.99 I am int array example 3

List	val listIn= listOf ("A", "B", "C", "D")
	println(listIn)
	val listChange= mutableListOf ("A", "B", "C", "D")
	listChange.remove("D")
	println(listChange)
	println("I am list example " + listChange[2])
		[A, B, C, D]
		[A, B, C]
		I am list example C

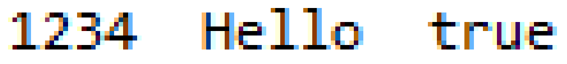
Basic Syntax

Arrays

- Array 1 dimensional

```
fun main() {
    val rows : Int  = 3
    val array1 = arrayOf(1234, "Hello", true)
    for (i in 0..rows - 1) {
          print( " " + array1[i] + " ")
        }
}
```

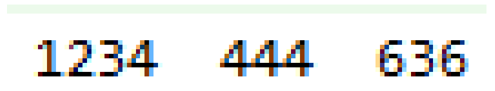
Result



```
1234  Hello  true
```

```
fun main() {
    val array2 = arrayOf<Int>(1234, 444, 636)
    for (i in 0 .. array2.size-1 ) {
          print( " " + array2[i] + " ")
        }
}
```

Result



```
1234  444  636
```

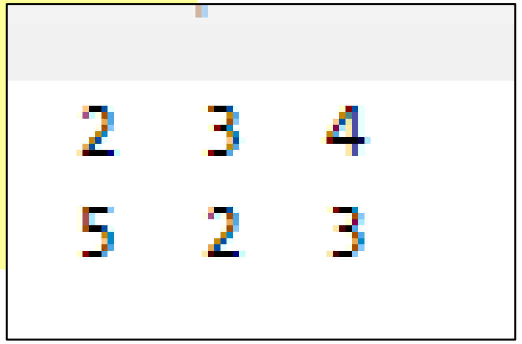
Basic Syntax

Arrays

- Array 2 dimensional

```
fun main() {
    val rows : Int  = 2
    val columns : Int = 3
    val firstMatrix = arrayOf(intArrayOf(2, 3, 4), intArrayOf(5, 2, 3))
    for (i in 0..rows - 1) {
         for (j in 0..columns - 1) {
              print( " " + firstMatrix[i][j] + " ")
         }
        println(" ")
    }
}
```

Result



```
 2  3  4
 5  2  3
```

Basic Syntax

Operators

- Arithmetic Operators

Operator	Meaning
+	Addition (also used for string concatenation)
-	Subtraction Operator
*	Multiplication Operator
/	Division Operator
%	Modulus Operator

Basic Syntax

Operators

- Arithmetic Operators and Function

Expression	Function name	Translates to
a + b	plus	a.plus(b)
a - b	minus	a.minus(b)
a * b	times	a.times(b)
a / b	div	a.div(b)
a % b	mod	a.mod(b)

```
fun main() {
    var a : Int = 5
    var b : Int = 3
    println("a+b = "+  a.plus(b))
}
```

Basic Syntax

Operators

- Assignment Operators

Expression	Equivalent to	Translates to
a +=b	a = a + b	a.plusAssign(b)
a -= b	a = a - b	a.minusAssign(b)
a *= b	a = a * b	a.timesAssign(b)
a /= b	a = a / b	a.divAssign(b)
a %= b	a = a % b	a.modAssign(b)

```
fun main() {
    var a  =5
     a  += 3
    println("a = " + a)
}
```

Basic Syntax

Operators

- Comparison and Equality Operators

Operator	Meaning	Expression
>	greater than	a > b
<	less than	a < b
>=	greater than or equals to	a >= b
<=	less than or equals to	a < = b
==	is equal to	a == b
!=	not equal to	a != b

Basic Syntax

Operators

- Logical Operators

Operator	Description	Expression	Corresponding Function
||	true if either of the Boolean expression is true	(a>b)||(a<c)	(a>b)or(a<c)
&&	true if all Boolean expressions are true	(a>b)&&(a<c)	(a>b)and(a<c)

Basic Syntax

Type Conversion

A numeric value of one type is not automatically converted to another type

- toByte()
- toShort()
- toInt()
- toLong()
- toFloat()
- toDouble()
- toChar()
- toString()

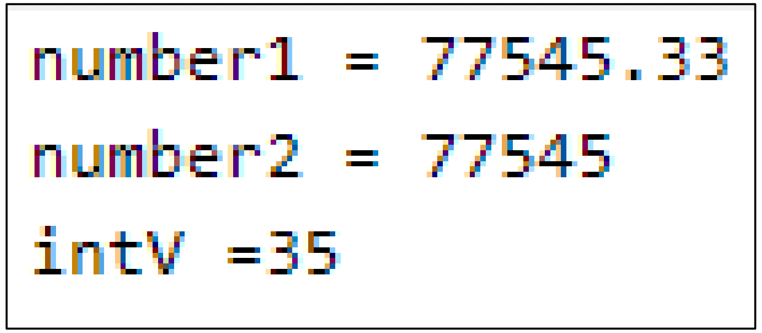
Basic Syntax

Type Conversion

Example

```
fun main() {
      val number1: Double = 77545.33
      val number2: Int = number1.toInt()
         println("number1 = "+ number1)
         println("number2 = "+ number2)
      val str : String = "35"
      val intV : Int = str.toInt()
         println( "intV =" + intV )
}
```

Result



```
number1 = 77545.33
number2 = 77545
intV =35
```

Basic Syntax

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. */
```

Basic Syntax

- Nullable Variable to solve NullPointerException or NPE

 - Use var and ?

```
var nullV   : Int    = null // Error
var nullV2 : Int?  =  null
```

 - !! Operator

This operator is used to explicitly tell the compiler that the property is not null and if it’s null, please throw a null pointer exception (NPE).

```
val s: String? = “”
val lowerS = s!!.toLowerCase()
```

Basic Syntax

?: (Elvis Operator)

```
val result = value1 ?: value2
```

If *value1* is NOT NULL, result is assigned its value.
But, if *value1* is NULL, result is assigned *value2*’s value.

Elvis Operator

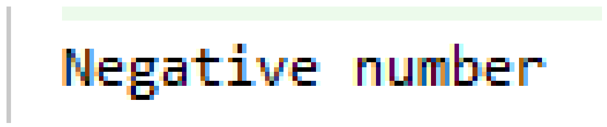
Basic Syntax

- If - Else

```
fun main() {
    val number : Int = -5
    if (number > 0) {
        print("Positive number")   }
    else { print("Negative number")  }
}
```

```
fun main() {
   val number : Int = -5
   val result = if (number > 0) {
        "Positive number“     }
     else { "Negative number“     }
    println(result)
}
```

Result



```
Negative number
```

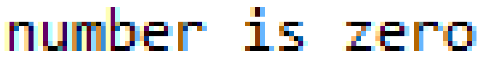
Basic Syntax

- if...else...if

```
fun main() {
    val number : Int = 0
    val result :String = if (number > 0)
        "positive number"
    else if (number < 0)
        "negative number"
    else
        "zero"

    println("number is $result")
}
```

Result



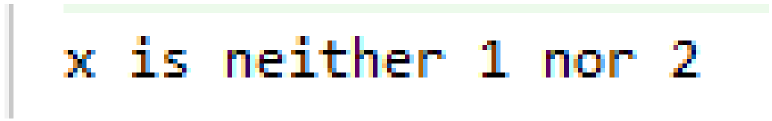
```
number is zero
```

Basic Syntax

- Use of When

```
fun main() {
   val x : Int = 5
   when (x) {
      1 -> print("x = 1")
      2 -> print("x =  2")
      else -> {
          print("x is neither 1 nor 2")
      }
   }
}
```

Result



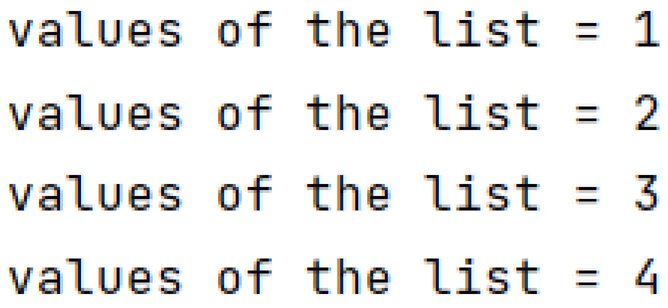
```
x is neither 1 nor 2
```

Basic Syntax

- For Loop

```
fun main() {
    val items = listOf(1, 2, 3, 4)
    for (i in items)
          println("values of the list = "+ i )
}
```

Result



```
values of the list = 1
values of the list = 2
values of the list = 3
values of the list = 4
```

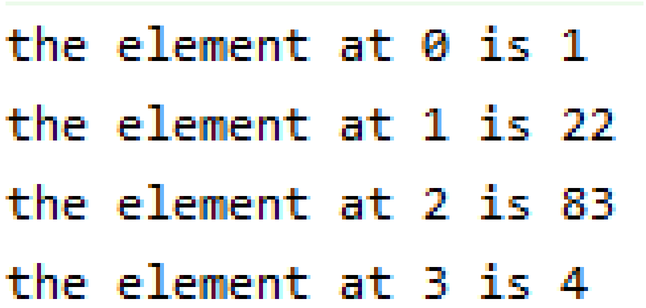
Basic Syntax

- For Loop

```
fun main() {
   val items = listOf(1, 22, 83, 4)

   for ((index, value) in items.withIndex()) {
      println("the element at $index is $value")
   }
}
```

Result



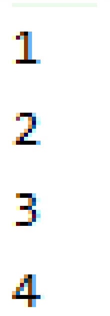
```
the element at 0 is 1
the element at 1 is 22
the element at 2 is 83
the element at 3 is 4
```

Basic Syntax

- For Loop : Range expressions

```
fun main() {
   for (i in 1..4)
         println(i)
}
```

Result

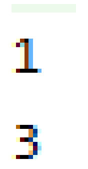


```
1
2
3
4
```

Arbitrary step :

```
fun main() {
   for (i in 1..4 step 2)
        println(i)
}
```

Result



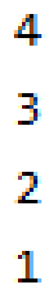
```
1
3
```

Basic Syntax

- For Loop : Reverse order

```
fun main() {
   for (i in 4 downTo 1)
         println(i)
}
```

Result

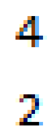


```
4
3
2
1
```

Arbitrary step :

```
fun main() {
   for (i in 4 downTo 1 step 2)
         println(i)
}
```

Result



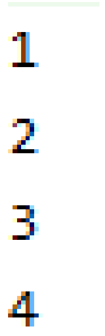
```
4
2
```

Basic Syntax

- For Loop : until

```
fun main() {
   for (i in 1 until 5) {
      // i in [1, 5), 5 is excluded
        println(i)
    }
}
```

Result



```
1
2
3
4
```

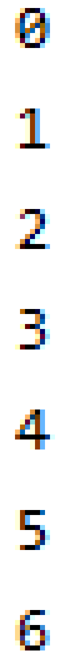
Basic Syntax

- While Loop

```
fun main() {
   var x:Int = 0

   while(x <= 6) {
      println(x)
      x++
   }
}
```

Result



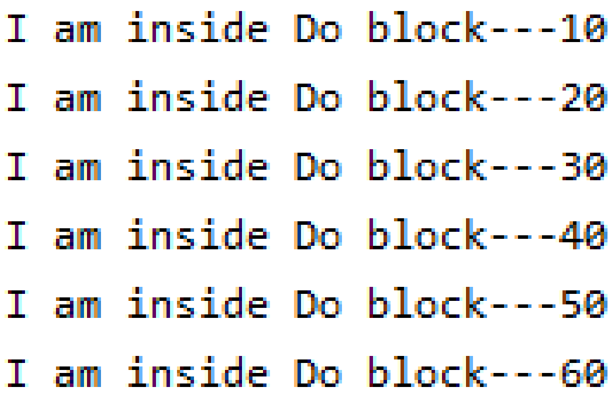
```
0
1
2
3
4
5
6
```

Basic Syntax

- Do-while loop

```
fun main() {
   var x:Int = 0
   do {
             x = x + 10
             println("I am inside Do block---"+ x)
   } while(x <= 50)
}
```

Result



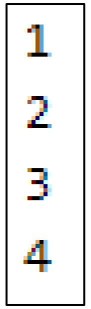
```
I am inside Do block---10
I am inside Do block---20
I am inside Do block---30
I am inside Do block---40
I am inside Do block---50
I am inside Do block---60
```

Basic Syntax

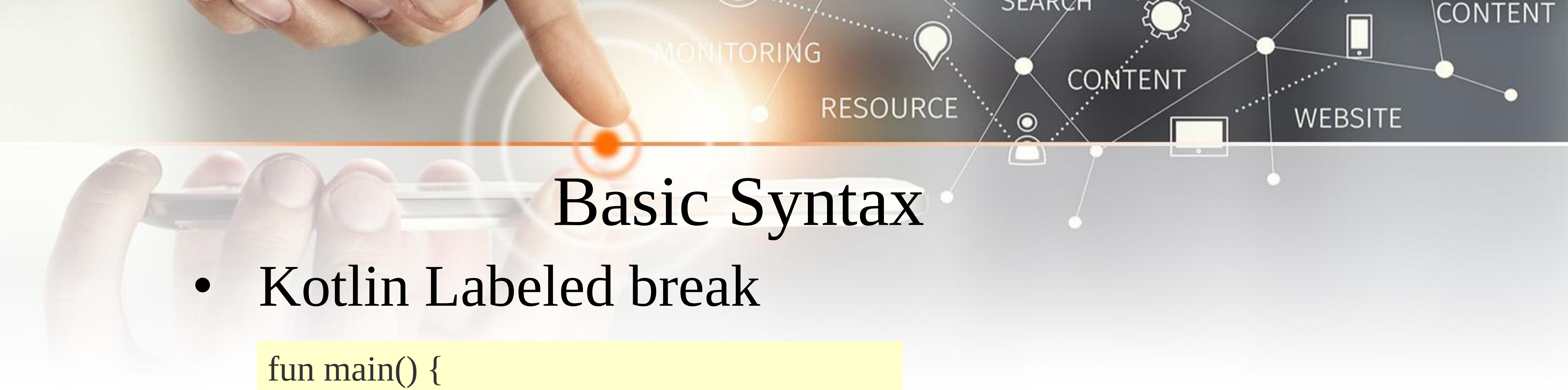
- Kotlin break

```
fun main() {
    for (i in 1..10) {
        if (i == 5) {
            break
        }
        println(i)
    }
}
```

Result



```
1
2
3
4
```

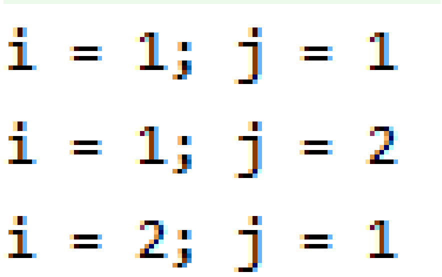


Basic Syntax

- Kotlin Labeled break

```
fun main() {
    first@ for (i in 1..4) {
        second@ for (j in 1..2) {
                    println("i = $i; j = $j")
                    if (i == 2)
                        break@first
                }
            }
}
```

Result



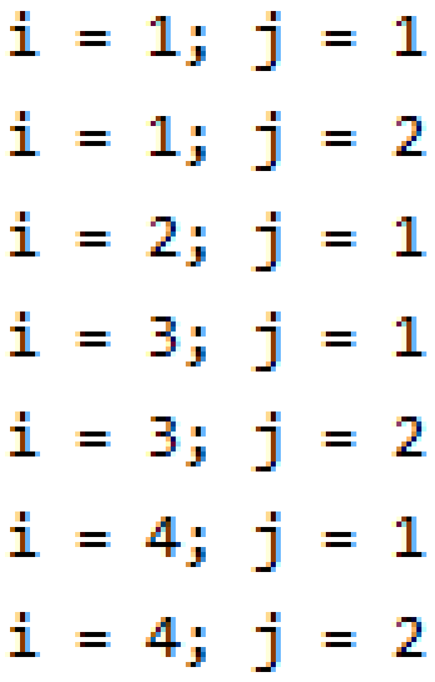
```
i = 1; j = 1
i = 1; j = 2
i = 2; j = 1
```

Basic Syntax

- Kotlin Labeled break

```
fun main() {
    first@ for (i in 1..4) {
        second@ for (j in 1..2) {
                        println("i = $i; j = $j")
                        if (i == 2)
                            break@second
                    }
                }
}
```

Result



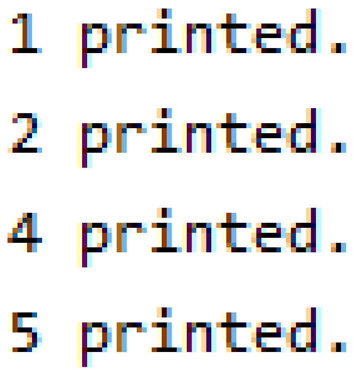
```
i = 1; j = 1
i = 1; j = 2
i = 2; j = 1
i = 3; j = 1
i = 3; j = 2
i = 4; j = 1
i = 4; j = 2
```

Basic Syntax

- Kotlin continue

```
fun main() {
    for (i in 1..5) {
        if (i == 3) {
            continue
        }
        println("$i printed.")
    }
}
```

Result



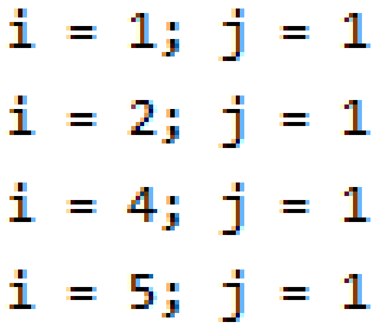
```
1 printed.
2 printed.
4 printed.
5 printed.
```

Basic Syntax

- Kotlin Labeled continue

```
fun main() {
    here@ for (i in 1..5) {
                for (j in 1..4) {
                    if (i == 3 || j == 2)
                        continue@here
                    println("i = $i; j = $j")
                }
            }
}
```

Result



```
i = 1; j = 1
i = 2; j = 1
i = 4; j = 1
i = 5; j = 1
```

Basic Syntax

- Functions

```
fun function_name (parameters) [  :  data_type] {

    //// statement

}
```

Example

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

Basic Syntax

- 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 (Compact Function):

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

Basic Syntax

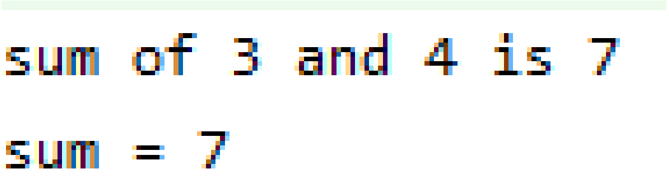
- Call Functions

```
fun main() {
     printSum(3,4)
     println("sum = " + sum(3,4) )
}

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


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

Result



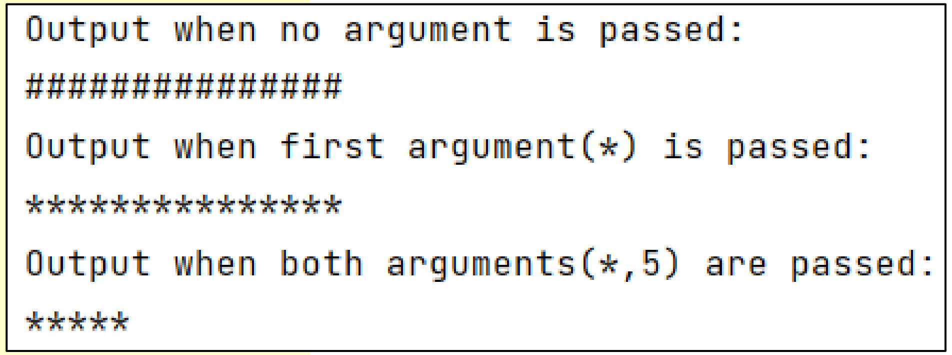
```
sum of 3 and 4 is 7
sum = 7
```

Basic Syntax

- Kotlin Default Argument

```
fun displayBorder( character : Char = '#', length : Int = 15) {
    for (i in 1..length) {
        print(character)
    }
    println()
}
fun main() {
    println("Output when no argument is passed:")
    displayBorder()
    println("Output when first argument is passed:")
    displayBorder('*')
     println("Output when both arguments are passed:")
    displayBorder('*', 5)
}
```

Result



```
Output when no argument is passed:
###############
Output when first argument(*) is passed:
***************
Output when both arguments(*,5) are passed:
*****
```

Basic Syntax

- Kotlin named argument

```
fun displayBorder( character : Char = '#', length : Int = 15) {
    for (i in 1..length) {
        print(character)
    }
    println()
}

fun main( ){
    displayBorder(length = 5)
}
```

Result

```
#####
```

End Of Chapter

References

- http://www.cems.uwe.ac.uk/~bk2dean/uwe/digitalmedia/mobiledevelopment/lectures/anatomy_of_a_mobile_device.ppt
- https://en.wikipedia.org/wiki/Windows_Phone#/media/File:Windows_10_Logo.svg
- https://www.cs.cmu.edu/~bam/uicourse/830spring09/BFeiginMobileApplicationDevelopment.pdf
- http://cs.joensuu.fi/~zhao/Courses/Location/Dariusz.ppt