Jvm

Kotlin

Kerem Akkaya

IntelliJ IDEA

JET BRAINS

Kotlin Powers

- Concise
- Interoperable
- Tool Friendly
  - Ant
  - Maven
  - Gradle

Server Side Development

- Spring supports from version 5.0
  - Spring Boot
- Vert.x for reactive Web Applications
- Amazon Web Services
- Google Cloud Platform
- Heroku
- HttpServlet

Android

- JDK 6 support on Kotlin
- Android Studio Plugin
- Performance
  - Inline functions
  - Lambdas
- Incremental Builds Faster
- Kotlin Koan

Javascript

- DOM elements
- Graphics such as WebGL
- Server-side technology
  - Node.js

Native Interoperability

- Currently in development ☹
- Without any VM
- IOS, embedded targets
- Target Platforms
  - Windows (x86_64 only)
  - Linux
  - Mac OS
  - iOS
  - Android

Hello World - Main

```
1  /**
2   * We declare a package-level function main which returns Unit and takes
3   * an Array of strings as a parameter. Note that semicolons are optional.
4   */
5
6  fun main(args: Array<String>) {
7      println("Hello, Devnot!")
8  }
9
```

```
Compilation completed successfully
Hello, Devnot!
```

Hello World – Call Function

```
1
2  fun printMessageWithParameters (param1: String, param2: String): Unit {
3      println("${param1} on ${param2}!")
4      return Unit
5  }
6
7  fun main (args: Array<String>) {
8      printMessageWithParameters("Devnot Istanbul", "9th of December")
9  }
```

```
1
2  fun printMessageWithParameters (param1: String, param2: String) {
3      println("${param1} on ${param2}!")
4  }
5
6  fun main (args: Array<String>) {
7      printMessageWithParameters("Devnot Istanbul", "9th of December")
8  }
```

```
Compilation completed successfully
Devnot Istanbul on 9th of December!
```

Hello World - Class

```kotlin
/**
 * Here we have a class with a primary constructor and a member function.
 * Note that there's no `new` keyword used to create an object.
 * See http://kotlinlang.org/docs/reference/classes.html#classes
 */

class Greeter(val name: String) {
    fun greet() {
        println("Hello, ${name}");
    }
}

fun main(args: Array<String>) {
    Greeter(args[0]).greet()
}
```

```
Compilation completed successfully
Hello, Kerem
```

Variables
Assign-once (read-only)

```
1  fun main(args: Array<String>) {
2      val a: Int = 1  // immediate assignment
3      val b = 2   // `Int` type is inferred
4      val c: Int  // Type required when no initializer is provided
5      c = 3       // deferred assignment
6      println("a = $a, b = $b, c = $c")
7  }
```

```
Compilation completed successfully
a = 1, b = 2, c = 3
```

Variables - Mutable

```
1  fun main(args: Array<String>) {
2      var x = 5 // `Int` type is inferred
3      x += 1
4      println("x = $x")
5  }
```

```
Compilation completed successfully
x = 6
```

String Templates

```
1  fun main(args: Array<String>) {
2      var a = 1
3      // simple name in template:
4      val s1 = "a is $a"
5
6      a = 2
7      // arbitrary expression in template:
8      val s2 = "${s1.replace("is", "was")}, but now is $a"
9      println(s2)
10 }
```

```
Compilation completed successfully
a was 1, but now is 2
```

Conditional Expressions - If

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

fun main(args: Array<String>) {
    println("max of 0 and 11 is ${maxOf(0, 11)}")
}
```

```
Compilation completed successfully
max of 0 and 11 is 11
```

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

fun main(args: Array<String>) {
    println("max of 0 and 11 is ${maxOf(0, 11)}")
}
```

```
Compilation completed successfully
max of 0 and 11 is 11
```

```
fun maxOf(a: Int, b: Int) = if (a > b) {
    println("max is 1st arg = $a")
    a
} else {
    println("max is 2nd arg = $b")
    b
}

fun main(args: Array<String>) {
    println("max of 0 and 11 is ${maxOf(0, 11)}")
}
```

```
Compilation completed successfully
max is 2nd arg = 11
max of 0 and 11 is 11
```

Conditional Expressions – When (No Switch)

```
fun describe(x: Any): String =
when (x) {
    1           -> "One"
    "Hello"     -> "Greeting"
    is Long     -> "Long"
    in 2..15    -> "$x is in the range"
    !is String -> "Not a string"
    else        -> "Unknown"
}

fun main(args: Array<String>) {
    println(describe(1))
    println(describe("Hello"))
    println(describe(1000L))
    println(describe(2))
    println(describe(15))
    println(describe(16))
    println(describe("other"))
}
```

```
Compilation completed
One
Greeting
Long
2 is in the range
15 is in the range
Not a string
Unknown
```

```
fun main(args: Array<String>) {
    val items = setOf("apple", "banana", "kiwi")
    when {
        "orange" in items -> println("juicy")
        "apple" in items -> println("apple is fine too")
    }
}
```

```
Compilation comple
apple is fine too
```

Loops - For

```
1  fun main(args: Array<String>) {
2      val items = listOf("apple", "banana", "kiwi")
3      for (item in items) {
4          println(item)
5      }
6  }
```

```
Compilat
apple
banana
kiwi
```

```
8  fun main(args: Array<String>) {
9      val items = listOf("apple", "banana", "kiwi")
10     for (index in items.indices) {
11         println("item at $index is ${items[index]}")
12     }
13 }
```

```
Compilation completed s
item at 0 is apple
item at 1 is banana
item at 2 is kiwi
```

Loops - For

```
8  // Iterating on index and value
9  fun main(args: Array<String>) {
10     val items = listOf("apple", "banana", "kiwi")
11     for ((index, value) in items.withIndex()) {
12         println("item at $index is ${value}")
13     }
14 }
```

```
Compilation completed s
item at 0 is apple
item at 1 is banana
item at 2 is kiwi
```

```
1  fun main(args: Array<String>) {
2      val items = listOf("apple", "banana", "kiwi")
3      for (index in 0..items.lastIndex) {
4          println("$index = ${items[index]}")
5      }
6  }
```

```
Compilation c
0 = apple
1 = banana
2 = kiwi
```

Loops – While and Do

```
1  fun main(args: Array<String>) {
2      val items = listOf("apple", "banana", "kiwi")
3      var index = 0
4      while (index < items.size) {
5          println("$index = ${items[index]}")
6          index++
7      }
8  }
```

```
Compilation c
0 = apple
1 = banana
2 = kiwi
```

```
1  fun main(args: Array<String>) {
2      val items = listOf("apple", "banana", "kiwi")
3      var index = 0
4      do {
5          println("$index = ${items[index]}")
6          index++
7      } while (index < items.size)
8  }
```

Return, break, continue

```
1  fun main(args: Array<String>) {
2      label@ for (i in 1..10){
3          for (j in 1..10) {
4              if (i == j){
5                  println(j)
6                  continue@label
7              } else{
8                  print(j)
9              }
10         }
11     }
12 }
```

Compilation

```
1
12
123
1234
12345
123456
1234567
12345678
123456789
12345678910
```

Ranges

```
1  fun main(args: Array<String>) {
2      val list = listOf("a", "b", "c")
3
4      if (-1 !in 0..list.lastIndex) {
5          println("-1 is out of range")
6      }
7      if (list.size !in list.indices) {
8          println("list size is out of valid list indices range too")
9      }
10 }
```

```
Compilation completed successfully
-1 is out of range
list size is out of valid list indices range too
```

```
1  fun main(args: Array<String>) {
2      for (x in 1..10 step 2) {
3          print("$x ")
4      }
5      println()
6      for (x in 9 downTo 0 step 3) {
7          print("$x ")
8      }
9  }
```

```
Compilation
1 3 5 7 9
9 6 3 0
```

Nullables - ?

```
1  fun main(args: Array<String>) {
2      var a: String = "abc"
3      a = null // compilation error
4  }
```

Error:(3, 5) Null can not be a value of a non-null type String

```
1  fun main(args: Array<String>) {
2      var b: String? = "abc"
3      b = null // ok
4      val l = b.length // compilation error.
5  }
```

Error:(4, 13) Only safe (?.) or non-null asserted (!!.) calls are allowed on a nullable receiver of type String?

```
1  fun main(args: Array<String>) {
2      var b: String? = "abc"
3      b = null // ok
4      val l = b!!.length // runtime error
5  }
```

```
Exception in thread "main" kotlin.KotlinNullPointerException
    at NullablesKt.main(Nullables.kt:4)
```

Safe Calls – Safe Casts

```
1 fun main(args: Array<String>) {
2     var b: String? = "abc"
3     b = null
4     val l = b?.length
5     println("end")
6 }
```

```
Compila
end
```

```
1 fun main(args: Array<String>) {
2     val a: Any = 10
3     val aInt: Int? = a as? Int
4     println(aInt ?: "empty")
5     println("aInt = $aInt")
6 }
```

```
Compilation c
10
aInt = 10
```

Functions - fun

```
1
2  fun double(x: Int): Int {
3      return 2 * x
4  }
5
6  fun main (args: Array<String>) {
7      println(double(10))
8  }
```

```
Compilation completed successfully
20
```

```
1
2  fun double(x: Int): Int {
3      return 2 * x
4  }
5
6  fun main (args: Array<String>) {
7      val result = double(8)
8      println(result)
9  }
10
```

```
Compilation completed successfully
16
```

Functions – Default Arguments

```
1  fun foo(bar: Int = 0, baz: Int = 1, qux: () -> Unit) { /* ... */
2      println("Bar: $bar, baz: $baz,")
3      qux()
4  }
5
6  fun main (args: Array<String>) {
7      foo(5) { println("devnot 1") } // Uses the default value baz = 1
8      foo { println("devnot 2") }    // Uses both default values bar = 0 and baz = 1
9      foo(6,7) { println("devnot 3") }
10 }
```

```
Compilation completed successfully
Bar: 5, baz: 1,
devnot 1
Bar: 0, baz: 1,
devnot 2
Bar: 6, baz: 7,
devnot 3
```

Functions - Default Arguments

```kotlin
data class Money(val amount: Int, val currency: String)

// Functions can have default parameters 1st 2nd 4th
fun pay (money: Money, messages: String = "Empty String"){
     println("1 Payment with message $messages")
}

// And they can have multiple of them. 3rd
fun pay (money: Money, messages: String = "Empty String", commission: Int = 0) {
     println("2 Payment with message and commission amount $messages $commission")
}

// 5th
fun pay (money: Money, messages: String = "Empty String", commissionMessage: String = "") {
     println("3 Payment with message and commission message $messages $commissionMessage")
}

fun main (args: Array<String>) {
    val gift = Money(100,"$")

    pay(gift) // 1st call
    pay(gift, "For getting new Gift 2nd") // 2nd call

    pay(gift, "For getting new Gift 3rd", 5) // 3rd call

    pay(money = gift, messages = "For getting new Gift 4th") // 4th call
    pay(money = gift, commissionMessage = "For getting new Gift 6th") // 5th call

}
```

```
Compilation completed successfully
1 Payment with message Empty String
1 Payment with message For getting new Gift 2nd
2 Payment with message and commission amount For getting new Gift 3rd 5
1 Payment with message For getting new Gift 4th
3 Payment with message and commission message Empty String For getting new Gift 6th
```

Unit-returning functions

```
1  fun printHello(name: String?): Unit {
2      if (name != null)
3          println("Hello ${name}")
4      else
5          println("Hi there!")
6      // `return Unit` or `return` is optional
7  }
8
9  fun main (args: Array<String>) {
10     printHello("Kerem")
11 }
```

```
Compilation completed successfully
Hello Kerem
```

Single-Expression functions

```
1
2 fun double(x: Int) = x * 2 //Inferred return type
3
4 fun main (args: Array<String>) {
5     val result = double(9)
6     println(result)
7 }
8
```

```
Compilation completed successfully
18
```

Variable Number of Arguments

```
1  fun <T> asList(vararg ts: T): List<T> {
2      val result = ArrayList<T>()
3      for (t in ts) // ts is an Array
4          result.add(t)
5      return result
6  }
7
8  fun main (args: Array<String>) {
9      val a = arrayOf(1, 2, 3)
10     val list = asList(-1, 0, *a, 4) // * = spread operator
11     println(list)
12 }
13
```

```
Compilation completed successfully
[-1, 0, 1, 2, 3, 4]
```

Extension Function

```
1  fun Int.powerOfExtension(x: Int): Int {
2      return Math.pow(this.toDouble(), x.toDouble()).toInt()
3  }
4
5  fun main (args: Array<String>) {
6      println(5.powerOfExtension(3))
7  }
```

```
Compilation completed successfully
125
```

```
1  val <T> List<T>.lastIndex: Int
2      get() = size - 1
3
4  fun main (args: Array<String>) {
5      val l = listOf<Int>(1, 2, 3, 4, 5, 6)
6      println(l.lastIndex)
7  }
```

```
Compilation completed successfully
5
```

Infix Operator

```
1  infix fun Int.powerOf(x: Int): Int {
2      return Math.pow(this.toDouble(), x.toDouble()).toInt()
3  }
4
5  fun main (args: Array<String>) {
6      println(5 powerOf 3)
7  }
8
```

```
Compilation completed successfully
125
```

Lambda Functions

```
1  fun <T, R> List<T>.map(transform: (T) -> R): List<R> {
2      val result = arrayListOf<R>()
3      for (item in this)
4          result.add(transform(item))
5      return result
6  }
7
8  fun main (args: Array<String>) {
9      val numbers: List<Int> = arrayListOf<Int>(1, 2, 3)
10
11     val doubled = numbers.map { it * 2 } //inferred lambda same as { x -> x * 2 }
12     println(doubled)
13     val poweredLambda = numbers.map { x -> x * x } //inferred lambda same as { it * it }
14     println(poweredLambda)
15
16     val poweredLambdaPlusOne = numbers.map { x ->
17                                     print("- ${x * x} ")
18                                     x * x + 1 }
19     println()
20     println(poweredLambdaPlusOne)
21
22     val tripled = numbers.map (fun (x) = x * 3) // Inline function
23     println(tripled)
24 }
```

```
Compilation
[2, 4, 6]
[1, 4, 9]
- 1 - 4 - 9
[2, 5, 10]
[3, 6, 9]
```

Builder-style usage of methods that return Unit

```
1 fun arrayOfDefaultValue(size: Int, default: Int): IntArray {
2     return IntArray(size).apply { fill(default) }
3 }
4 fun main (args: Array<String>) {
5    println(arrayOfDefaultValue(5,3).asList())
6    println(arrayOfDefaultValue(5,2).asList())
7 }
```

```
Compilation completed successfully
[3, 3, 3, 3, 3]
[2, 2, 2, 2, 2]
```

Classes

```
1 open class DevNot
2 class SubDevNot: DevNot()
3 fun main (args: Array<String>) {
4     val devNot = SubDevNot()
5     println(devNot)
6 }
```

```
1 open class DevNot
2 class SubDevNot(var presenter: String = "Uğur"): DevNot()
3 fun main (args: Array<String>) {
4     val devNot = SubDevNot("Kerem")
5     println(devNot.presenter)
6     val devNotDefault = SubDevNot()
7     println(devNotDefault.presenter)
8 }
```

```
Compila
Kerem
Uğur
```

Classes - Constructors

```
1  package com.devnot.deneme
2  open class DevNot(var presenter: String = "Uğur")
3  class DevNotKerem: DevNot("Kerem")
4  class DevNotUgur: DevNot("Uğur")
5  fun main (args: Array<String>) {
6      println(DevNotKerem().presenter)
7      println(DevNotUgur().presenter)
8  }
```

```
Compila
Kerem
Uğur
```

```
1  package com.devnot.deneme
2  open class DevNot(open var presenter: String?)
3  class DevNotPresenter(override var presenter: String? = "Uğur"): DevNot(presenter)
4  fun main (args: Array<String>) {
5      println(DevNotPresenter("Kerem").presenter)
6      println(DevNotPresenter().presenter)
7  }
```

```
Compila
Kerem
Uğur
```

Classes Constructors

```
1  package com.devnot.deneme
2  class Person(val name: String) {
3      val children = mutableListOf<Person>()
4      constructor(name: String, parent: Person) : this(name) {
5          parent.children.add(this)
6      }
7  }
8  fun main (args: Array<String>) {
9      val baba = Person("Baba")
10     val child = Person("Kerem", baba)
11     println(baba.children[0].name)
12 }
```

```
Compilati
Kerem
```

```
1  package com.devnot.deneme
2  open class DevNot(open var presenter: String?)
3  class DevNotPresenter(var presenterName: String? = "Uğur"): DevNot(presenterName)
4  fun main (args: Array<String>) {
5      println(DevNotPresenter("Kerem").presenter)
6      println(DevNotPresenter().presenter)
7  }
```

```
Compila
Kerem
Uğur
```

Getters and Setters

```
1  open class DevNot(val name: String) {
2
3      var stringRepresentation: String = "Default Rep"
4      get() = name + " - " + field
5      set(value) {
6          field = value
7          println("setValue $value")
8      }
9
10     var time: Int = 0
11     get() = if (field >40) 40 else field
12
13 }
14 class SubDevNot(var presenter: String): DevNot(presenter)
15 fun main (args: Array<String>) {
16     val devNot = SubDevNot("Kerem")
17     devNot.stringRepresentation = "DevNot"
18     println(devNot.stringRepresentation)
19     devNot.time = 35
20     println(devNot.time)
21     devNot.time = 50
22     println(devNot.time)
23 }
```

```
Compilation completed successfully
setValue DevNot
Kerem - DevNot
35
40
```

Getters and Setters

```
1  open class DevNot(val name: String) {
2
3      var stringRepresentation: String = "Default Rep"
4      get() = name + " - " + field
5      set(value) {
6          field = value
7          println("setValue $value")
8      }
9
10     var time: Int = 0
11     get() = if (field >40) 40 else field
12     private set
13
14 }
15 class SubDevNot(var presenter: String): DevNot(presenter)
16 fun main (args: Array<String>) {
17     val devNot = SubDevNot("Kerem")
18     devNot.stringRepresentation = "DevNot"
19     println(devNot.stringRepresentation)
20     devNot.time = 35
21     println(devNot.time)
22     devNot.time = 50
23     println(devNot.time)
24 }
```

```
Error:(20, 4) Cannot assign to 'time': the setter is invisible (private in a supertype) in 'SubDevNot'
Error:(22, 4) Cannot assign to 'time': the setter is invisible (private in a supertype) in 'SubDevNot'
```

Data Class

```
1 package com.devnot.deneme
2 data class DevNot(var presenter: String = "Uğur")
3 fun main (args: Array<String>) {
4     println(DevNot("Kerem").presenter)
5     println(DevNot().presenter)
6 }
```

```
Compila
Kerem
Uğur
```

Object Oriented Support

```
1  open class Base {
2      open fun v() {println("Base.v()")}
3      fun nv() {println("Base.nv()")}
4  }
5  class Derived() : Base() {
6      override fun v() {println("Derived.v()")}
7  }
8  fun main (args: Array<String>) {
9      val obj = Derived()
10     obj.v()
11     obj.nv()
12 }
```

```
Compilation c
Derived.v()
Base.nv()
```

Interfaces

```
interface MyInterface {
    fun bar()
    fun foo() {
      // optional body
    }
}
```

```
class Child : MyInterface {
    override fun bar() {
        // body
    }
}
```

```
interface MyInterface {
    val prop: Int // abstract

    val propertyWithImplementation: String
        get() = "foo"

    fun foo() {
        print(prop)
    }
}

class Child : MyInterface {
    override val prop: Int = 29
}
```

Interfaces – Resolve Override Conflicts

```
interface A {
    fun foo() { print("A") }
    fun bar()
}

interface B {
    fun foo() { print("B") }
    fun bar() { print("bar") }
}

class C : A {
    override fun bar() { print("bar") }
}

class D : A, B {
    override fun foo() {
        super<A>.foo()
        super<B>.foo()
    }

    override fun bar() {
        super<B>.bar()
    }
}
```

try/catch expression

```
1  //'try/catch' expression
2  fun test() {
3      val result = try {
4          var divideByZero = 25 / 0
5      } catch (e: ArithmeticException) {
6          throw IllegalStateException(e)
7      }
8  }
9  fun main (args: Array<String>) {
10     test()
11 }
```

```
Exception in thread "main" java.lang.IllegalStateException: java.lang.ArithmeticException: / by zero
    at Try_catchKt.test(Try_catch.kt:6)
    at Try_catchKt.main(Try_catch.kt:10)
Caused by: java.lang.ArithmeticException: / by zero

    at Try_catchKt.test(Try_catch.kt:4)
    at Try_catchKt.main(Try_catch.kt:10)
```

With

```
1  class Turtle {
2      fun penDown(): Unit = println("pen down")
3      fun penUp(): Unit = println("pen up")
4      fun turn(degrees: Double): Unit = println("turned $degrees degrees")
5      fun forward(pixels: Double): Unit = println("forwarded $pixels pixels")
6  }
7  fun main (args: Array<String>) {
8      val myTurtle = Turtle()
9      with(myTurtle) { //draw a 100 pix square
10         penDown()
11         for(i in 1..4) {
12             forward(100.0)
13             turn(90.0)
14         }
15         penUp()
16     }
17 }
```

```
Compilation completed suc
pen down
forwarded 100.0 pixels
turned 90.0 degrees
forwarded 100.0 pixels
turned 90.0 degrees
forwarded 100.0 pixels
turned 90.0 degrees
forwarded 100.0 pixels
turned 90.0 degrees
pen up
```

Object Expressions

```
fun foo() {
    val adHoc = object {
        var x: Int = 0
        var y: Int = 0
    }
    print(adHoc.x + adHoc.y)
}
```

```
open class A(x: Int) {
    public open val y: Int = x
}

interface B {...}

val ab: A = object : A(1), B {
    override val y = 15
}
```

```
window.addMouseListener(object : MouseAdapter() {
    override fun mouseClicked(e: MouseEvent) {
        // ...
    }

    override fun mouseEntered(e: MouseEvent) {
        // ...
    }
})
```

Object Expressions

```
class C {
    // Private function, so the return type is the anonymous object type
    private fun foo() = object {
        val x: String = "x"
    }

    // Public function, so the return type is Any
    fun publicFoo() = object {
        val x: String = "x"
    }

    fun bar() {
        val x1 = foo().x        // Works
        val x2 = publicFoo().x  // ERROR: Unresolved reference 'x'
    }
}
```

No static - Companions

```
class MyClass {
    companion object Factory {
        fun create(): MyClass = MyClass()
    }
}
```

```
val instance = MyClass.create()
```

Java vs Kotlin

Kotlin:

```kotlin
class MyActivity : AppCompatActivity() {
  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity)
  }
}
```

Java:

```java
public class MyActivity extends AppCompatActivity {
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity);
  }
}
```

Java vs Kotlin

Kotlin:

```kotlin
val fab = findViewById(R.id.fab) as FloatingActionButton
fab.setOnClickListener {
    ...
}
```

Java:

```java
FloatingActionButton fab = (FloatingActionButton) findViewById
fab.setOnClickListener(new View.OnClickListener() {
  @Override
  public void onClick(View view) {
    ...
  }
});
```

References

- https://kotlinlang.org
- https://try.kotlinlang.org
- https://developer.android.com/samples/index.html?language=kotlin
- https://developer.android.com/kotlin/resources.html

Thanks - Questions

Introduction To Kotlin

Kerem Akkaya