무료 전자 책

배우기 Kotlin

Free unaffiliated eBook created from
Stack Overflow contributors.

#kotlin

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: kotlin

It is an unofficial and free Kotlin ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official Kotlin.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

1: Kotlin

Kotlin JVM JetBrains . Kotlin , , , Java 100 % . Kotlin Java . Kotlin JVM .

Kotlin

Kotlin Eclipse IntelliJ IDE . Kotlin Ant Gradle Maven .

`$ kotlinc Main.kt` java (`MainKt.class` (Kt). `$ java MainKt` java throw.

```
Exception in thread "main" java.lang.NoClassDefFoundError: kotlin/jvm/internal/Intrinsics
    at MainKt.main(Main.kt)
Caused by: java.lang.ClassNotFoundException: kotlin.jvm.internal.Intrinsics
    at java.net.URLClassLoader.findClass(URLClassLoader.java:381)
    at java.lang.ClassLoader.loadClass(ClassLoader.java:424)
    at sun.misc.Launcher$AppClassLoader.loadClass(Launcher.java:335)
    at java.lang.ClassLoader.loadClass(ClassLoader.java:357)
    ... 1 more
```

Java Kotlin jar .

```
java -cp .:/path/to/kotlin/runtime/jar/kotlin-runtime.jar MainKt
```

1.0.0	2016-02-15
1.0.1	2016-03-16
1.0.2	2016-05-13
1.0.3	2016-06-30
1.0.4	2016-09-22
1.0.5	2016-11-08
1.0.6	2016-12-27
1.1.0	2017-03-01
1.1.1	2017-03-14
1.1.2	2017-04-25
1.1.3	2017-06-23

Examples

Kotlin `main` . Kotlin "Hello World" .

```
package my.program

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

`Main.kt` `Main.kt` ( )

JVM . `my.program.MainKt` .

.

```
@file:JvmName("MyApp")
```

`my.program.MyApp` .

:

- `@JvmName` .
-

Hello Hello World

Kotlin Object Declaration .

```
package my.program

object App {
    @JvmStatic fun main(args: Array<String>) {
        println("Hello World")
    }
}
```

, `my.program.App` .

`App` . `App` .

:

- `@JvmStatic`

Hello World

Object Declaration , Companion Object Kotlin `main` .

```
package my.program

class App {
    companion object {
        @JvmStatic fun main(args: Array<String>) {
```

```
            println("Hello World")
        }
    }
}
```

, my.program.App .

App . Object Declaration .

"hello" .

```
class App {
    companion object {
        @JvmStatic fun main(args: Array<String>) {
            App().run()
        }
    }

    fun run() {
        println("Hello World")
    }
}
```

:

- @JvmStatic

varargs

varargs .

```
package my.program

fun main(vararg args: String) {
    println("Hello, world!")
}
```

Kotlin

java Java . Kotlin .

javac . java java .

kotlinc kotlin .

Kotlin . N (1 2 3) .

Kotlin .

```
fun main(args: Array<String>) {

    println("Enter Two number")
    var (a, b) = readLine()!!.split(' ') // !! this operator use for
```

```
NPE(NullPointerException).

    println("Max number is : ${maxNum(a.toInt(), b.toInt())}")
}



fun maxNum(a: Int, b: Int): Int {

    var max = if (a > b) {
        println("The value of a is $a");
        a
    } else {
        println("The value of b is $b")
        b
    }

    return max;

}
```

. :

```
Enter Two number
71 89 // Enter two number from command line

The value of b is 89
Max number is: 89
```

For !! Null .

: Intellij .

Kotlin : https://riptutorial.com/ko/kotlin/topic/490/kotlin-

2: DSL

Kotlin DSL .

Examples

DSL

:

```
infix fun <T> T?.shouldBe(expected: T?) = assertEquals(expected, this)
```

DSL .

```
@Test
fun test() {
  100.plusOne() shouldBe 101
}
```

DSL invoke .

:

```
class MyExample(val i: Int) {
  operator fun <R> invoke(block: MyExample.() -> R) = block()
  fun Int.bigger() = this > i
}
```

DSL .

```
fun main2(args: Array<String>) {
    val ex = MyExample(233)
    ex {
        // bigger is defined in the context of `ex`
        // you can only call this method inside this context
        if (777.bigger()) kotlin.io.println("why")
    }
}
```

:

```
val r = Random(233)
infix inline operator fun Int.rem(block: () -> Unit) {
  if (r.nextInt(100) < this) block()
}
```

DSL .

```
20 % { println("The possibility you see this message is 20%") }
```

:

```
operator fun <R> String.invoke(block: () -> R) = {
  try { block.invoke() }
  catch (e: AssertException) { System.err.println("$this\n${e.message}") }
}
```

DSL .

```
"it should return 2" {
   parse("1 + 1").buildAST().evaluate() shouldBe 2
}
```

shouldBe Infix approach to build DSL .

DSL : https://riptutorial.com/ko/kotlin/topic/10042/dsl-

3: Java 8 Stream Equivalents

Kotlin iterable . `Sequence` .

chain `asSequence()` `Sequence` . `Sequence` . `toList()` , `toSet()` , `toMap()` `Sequence` .

```
// switch to and from lazy
val someList = items.asSequence().filter { ... }.take(10).map { ... }.toList()

// switch to lazy, but sorted() brings us out again at the end
val someList = items.asSequence().filter { ... }.take(10).map { ... }.sorted()
```

?!?

Kotlin . Kotlin . Null NPE Java . Kotlin :

```
val someList = people.filter { it.age <= 30 }.map { it.name }
```

:

```
val someList: List<String> = people.filter { it.age <= 30 }.map { it.name }
```

`people` , `people.age` `Int` `Int` , `people.name` A `String` `map` `List<String>` (`List` `String`).

`List<People>?` `people` `null` `List<People>?` :

```
val someList = people?.filter { it.age <= 30 }?.map { it.name }
```

`List<String>?` `List<String>?` null (null *Kotlin* . null *Kotlin* . *Kotlin nullable*)

Kotlin . `Sequence` iterator "once use" , . Java 8 Kotlin .

```
// Java:
Stream<String> stream =
Stream.of("d2", "a2", "b1", "b3", "c").filter(s -> s.startsWith("b"));

stream.anyMatch(s -> true);    // ok
stream.noneMatch(s -> true);   // exception
```

```
// Kotlin:
val stream = listOf("d2", "a2", "b1", "b3", "c").asSequence().filter { it.startsWith('b' ) }

stream.forEach(::println) // b1, b2

println("Any B ${stream.any { it.startsWith('b') }}") // Any B true
println("Any C ${stream.any { it.startsWith('c') }}") // Any C false

stream.forEach(::println) // b1, b2
```

:

```
// Java:
Supplier<Stream<String>> streamSupplier =
    () -> Stream.of("d2", "a2", "b1", "b3", "c")
          .filter(s -> s.startsWith("a"));

streamSupplier.get().anyMatch(s -> true);   // ok
streamSupplier.get().noneMatch(s -> true);  // ok
```

Kotlin . `Sequence` `constrainOnce()` `Sequence` `constrainOnce()` .

```
val stream = listOf("d2", "a2", "b1", "b3", "c").asSequence().filter { it.startsWith('b' ) }
        .constrainOnce()

stream.forEach(::println) // b1, b2
stream.forEach(::println) // Error:java.lang.IllegalStateException: This sequence can be
consumed only once.
```

:

- Iterable API
- API
- List API
- Map API

Examples

```
// Java:
List<String> list = people.stream().map(Person::getName).collect(Collectors.toList());
```

```
// Kotlin:
val list = people.map { it.name }  // toList() not needed
```

```
// Java:
String joined = things.stream()
                      .map(Object::toString)
                      .collect(Collectors.joining(", "));
```

```
// Kotlin:
val joined = things.joinToString() // ", " is used as separator, by default
```

```
// Java:
int total = employees.stream()
                     .collect(Collectors.summingInt(Employee::getSalary)));
```

```
// Kotlin:
val total = employees.sumBy { it.salary }
```

```
// Java:
Map<Department, List<Employee>> byDept
     = employees.stream()
                .collect(Collectors.groupingBy(Employee::getDepartment));
```

```
// Kotlin:
val byDept = employees.groupBy { it.department }
```

```
// Java:
Map<Department, Integer> totalByDept
     = employees.stream()
                .collect(Collectors.groupingBy(Employee::getDepartment,
                     Collectors.summingInt(Employee::getSalary)));
```

```
// Kotlin:
val totalByDept = employees.groupBy { it.dept }.mapValues { it.value.sumBy { it.salary }}
```

.

```
// Java:
Map<Boolean, List<Student>> passingFailing =
     students.stream()
             .collect(Collectors.partitioningBy(s -> s.getGrade() >= PASS_THRESHOLD));
```

```
// Kotlin:
val passingFailing = students.partition { it.grade >= PASS_THRESHOLD }
```

```
// Java:
List<String> namesOfMaleMembersCollect = roster
    .stream()
    .filter(p -> p.getGender() == Person.Sex.MALE)
    .map(p -> p.getName())
    .collect(Collectors.toList());
```

```
// Kotlin:
val namesOfMaleMembers = roster.filter { it.gender == Person.Sex.MALE }.map { it.name }
```

```
// Java:
Map<Person.Sex, List<String>> namesByGender =
      roster.stream().collect(
        Collectors.groupingBy(
            Person::getGender,
            Collectors.mapping(
                Person::getName,
                Collectors.toList())));
```

```
// Kotlin:
val namesByGender = roster.groupBy { it.gender }.mapValues { it.value.map { it.name } }
```

```
// Java:
List<String> filtered = items.stream()
```

```
    .filter( item -> item.startsWith("o") )
    .collect(Collectors.toList());
```

```
// Kotlin:
val filtered = items.filter { item.startsWith('o') }
```

```
// Java:
String shortest = items.stream()
    .min(Comparator.comparing(item -> item.length()))
    .get();
```

```
// Kotlin:
val shortest = items.minBy { it.length }
```

2 -

```
// Java:
Stream.of("a1", "a2", "a3")
    .findFirst()
    .ifPresent(System.out::println);
```

```
// Kotlin:
sequenceOf("a1", "a2", "a3").firstOrNull()?.apply(::println)
```

3 -

```
// Java:
IntStream.range(1, 4).forEach(System.out::println);
```

```
// Kotlin:  (inclusive range)
(1..3).forEach(::println)
```

4 - , , .

```
// Java:
Arrays.stream(new int[] {1, 2, 3})
    .map(n -> 2 * n + 1)
    .average()
    .ifPresent(System.out::println); // 5.0
```

```
// Kotlin:
arrayOf(1,2,3).map { 2 * it + 1}.average().apply(::println)
```

5 - , , Int , .

```
// Java:
Stream.of("a1", "a2", "a3")
    .map(s -> s.substring(1))
    .mapToInt(Integer::parseInt)
```

```
    .max()
    .ifPresent(System.out::println);  // 3
```

```
// Kotlin:
sequenceOf("a1", "a2", "a3")
    .map { it.substring(1) }
    .map(String::toInt)
    .max().apply(::println)
```

6 - Ints , ,

```
// Java:
IntStream.range(1, 4)
    .mapToObj(i -> "a" + i)
    .forEach(System.out::println);

// a1
// a2
// a3
```

```
// Kotlin:  (inclusive range)
(1..3).map { "a$it" }.forEach(::println)
```

7 - , Int , String , .

```
// Java:
Stream.of(1.0, 2.0, 3.0)
    .mapToInt(Double::intValue)
    .mapToObj(i -> "a" + i)
    .forEach(System.out::println);

// a1
// a2
// a3
```

```
// Kotlin:
sequenceOf(1.0, 2.0, 3.0).map(Double::toInt).map { "a$it" }.forEach(::println)
```

```
// Java:
long count = items.stream().filter( item -> item.startsWith("t")).count();
```

```
// Kotlin:
val count = items.filter { it.startsWith('t') }.size
// but better to not filter, but count with a predicate
val count = items.count { it.startsWith('t') }
```

- , ,

```
// Java:
List<String> myList = Arrays.asList("a1", "a2", "b1", "c2", "c1");
```

```
myList.stream()
      .filter(s -> s.startsWith("c"))
      .map(String::toUpperCase)
     .sorted()
     .forEach(System.out::println);

// C1
// C2
```

```
// Kotlin:
val list = listOf("a1", "a2", "b1", "c2", "c1")
list.filter { it.startsWith('c') }.map (String::toUpperCase).sorted()
        .forEach (::println)
```

1 - .

```
// Java:
Arrays.asList("a1", "a2", "a3")
    .stream()
    .findFirst()
    .ifPresent(System.out::println);
```

```
// Kotlin:
listOf("a1", "a2", "a3").firstOrNull()?.apply(::println)
```

ifPresent String .

```
// Kotlin:
inline fun String?.ifPresent(thenDo: (String)->Unit) = this?.apply { thenDo(this) }

// now use the new extension function:
listOf("a1", "a2", "a3").firstOrNull().ifPresent(::println)
```

`apply()` : `apply()`

:

: `?.` null : http://stackoverflow.com/questions/34498562/in-kotlin-what-is-the-idiomatic-way-to-deal-with-nullable-values-referencing-o/34498563 # 34498563

5 - , .

```
// Java:
String phrase = persons
        .stream()
        .filter(p -> p.age >= 18)
        .map(p -> p.name)
        .collect(Collectors.joining(" and ", "In Germany ", " are of legal age."));

System.out.println(phrase);
// In Germany Max and Peter and Pamela are of legal age.
```

```
// Kotlin:
```

```
val phrase = persons
        .filter { it.age >= 18 }
        .map { it.name }
        .joinToString(" and ", "In Germany ", " are of legal age.")

println(phrase)
// In Germany Max and Peter and Pamela are of legal age.
```

Kotlin .

```
// Kotlin:
// data class has equals, hashcode, toString, and copy methods automagically
data class Person(val name: String, val age: Int)

val persons = listOf(Person("Tod", 5), Person("Max", 33),
                     Person("Frank", 13), Person("Peter", 80),
                     Person("Pamela", 18))
```

6 - ,

```
// Java:
Map<Integer, String> map = persons
        .stream()
        .collect(Collectors.toMap(
                p -> p.age,
                p -> p.name,
                (name1, name2) -> name1 + ";" + name2));

System.out.println(map);
// {18=Max, 23=Peter;Pamela, 12=David}
```

, . / `Map` :

```
// Kotlin:
val map1 = persons.map { it.age to it.name }.toMap()
println(map1)
// output: {18=Max, 23=Pamela, 12=David}
// Result: duplicates overridden, no exception similar to Java 8

val map2 = persons.toMap({ it.age }, { it.name })
println(map2)
// output: {18=Max, 23=Pamela, 12=David}
// Result: same as above, more verbose, duplicates overridden

val map3 = persons.toMapBy { it.age }
println(map3)
// output: {18=Person(name=Max, age=18), 23=Person(name=Pamela, age=23), 12=Person(name=David,
age=12)}
// Result: duplicates overridden again

val map4 = persons.groupBy { it.age }
println(map4)
// output: {18=[Person(name=Max, age=18)], 23=[Person(name=Peter, age=23), Person(name=Pamela,
age=23)], 12=[Person(name=David, age=12)]}
// Result: closer, but now have a Map<Int, List<Person>> instead of Map<Int, String>

val map5 = persons.groupBy { it.age }.mapValues { it.value.map { it.name } }
```

```
println(map5)
// output: {18=[Max], 23=[Peter, Pamela], 12=[David]}
// Result: closer, but now have a Map<Int, List<String>> instead of Map<Int, String>
```

:

```
// Kotlin:
val map6 = persons.groupBy { it.age }.mapValues { it.value.joinToString(";") { it.name } }

println(map6)
// output: {18=Max, 23=Peter;Pamela, 12=David}
// Result: YAY!!
```

Person joinToString joinToString Person.name .

7a - ,

```
// Java (verbose):
Collector<Person, StringJoiner, String> personNameCollector =
Collector.of(
        () -> new StringJoiner(" | "),          // supplier
        (j, p) -> j.add(p.name.toUpperCase()),  // accumulator
        (j1, j2) -> j1.merge(j2),               // combiner
        StringJoiner::toString);                // finisher

String names = persons
        .stream()
        .collect(personNameCollector);

System.out.println(names);  // MAX | PETER | PAMELA | DAVID

// Java (concise)
String names = persons.stream().map(p -> p.name.toUpperCase()).collect(Collectors.joining(" |
"));
```

```
// Kotlin:
val names = persons.map { it.name.toUpperCase() }.joinToString(" | ")
```

7b - SummarizingInt

```
// Java:
IntSummaryStatistics ageSummary =
    persons.stream()
           .collect(Collectors.summarizingInt(p -> p.age));

System.out.println(ageSummary);
// IntSummaryStatistics{count=4, sum=76, min=12, average=19.000000, max=23}
```

```
// Kotlin:

// something to hold the stats...
data class SummaryStatisticsInt(var count: Int = 0,
                                var sum: Int = 0,
                                var min: Int = Int.MAX_VALUE,
```

```
                                var max: Int = Int.MIN_VALUE,
                                var avg: Double = 0.0) {
    fun accumulate(newInt: Int): SummaryStatisticsInt {
        count++
        sum += newInt
        min = min.coerceAtMost(newInt)
        max = max.coerceAtLeast(newInt)
        avg = sum.toDouble() / count
        return this
    }
}

// Now manually doing a fold, since Stream.collect is really just a fold
val stats = persons.fold(SummaryStatisticsInt()) { stats, person ->
stats.accumulate(person.age) }

println(stats)
// output: SummaryStatisticsInt(count=4, sum=76, min=12, max=23, avg=19.0)
```

Kotlin stdlib .

```
// Kotlin:
inline fun Collection<Int>.summarizingInt(): SummaryStatisticsInt
        = this.fold(SummaryStatisticsInt()) { stats, num -> stats.accumulate(num) }

inline fun <T: Any> Collection<T>.summarizingInt(transform: (T)->Int): SummaryStatisticsInt =
        this.fold(SummaryStatisticsInt()) { stats, item -> stats.accumulate(transform(item)) }
```

summarizingInt .

```
val stats2 = persons.map { it.age }.summarizingInt()

// or

val stats3 = persons.summarizingInt { it.age }
```

. Sequence .

Java 8 Stream Equivalents : https://riptutorial.com/ko/kotlin/topic/707/java-8-stream-equivalents

4: Java Kotlin

Kotlin .

Java Kotlin Kotlin Java .

Examples

Kotlin Java .

```
val i : Int = 42
```

- `val` `var` , `final` (**" ")** **VAR** iable.
- `:`
- Kotlin obmitted

`int i = 42;`	`var i = 42` (`var i : Int = 42`)
`final int i = 42;`	`val i = 42`

- `;`
- Kotlin **null-safe.**
- Kotlin **100 % Java**
- Kotlin (JVM).
- Kotlin **.**
- Kotlin `equals` / `hashCode` `hashCode` .
- Kotlin **.**
- Kotlin **`new`** . .
- Kotlin () . . `val a = someMap["key"]`
- Kotlin JVM **Java Script** Kotlin .
- Kotlin **Java 6** Android (:) .
- Kotlin **Android** **.**
- Kotlin .
- ().

Kotlin `==` (, `equals` `equals`) ID `===` .

`a.equals(b);`	`a == b`
`a == b;`	`a === b`
`a != b;`	`a !== b`

: https://kotlinlang.org/docs/reference/equality.html

IF, TRY .

Kotlin `if` , `try` others expression .

, Kotlin Java .

```
val i = if (someBoolean) 33 else 42
```

`try` .

```
val i = try {
    Integer.parseInt(someString)
}
catch (ex : Exception)
{
    42
}
```

Java Kotlin : https://riptutorial.com/ko/kotlin/topic/10099/java--kotlin

5: JUnit

Examples

JUnit ,

```
@Rule @JvmField val myRule = TemporaryFolder()
```

`@JvmField myRule` ( ) () . JUnit .

JUnit : https://riptutorial.com/ko/kotlin/topic/6973/junit

6: Kotlin Android

Kotlin Android ButterKnife . , .

Examples

gradle .

- ( ) `build.gradle` Kotlin .

```
buildscript {
    ...
}

apply plugin: "com.android.application"
...
apply plugin: "kotlin-android"
apply plugin: "kotlin-android-extensions"
...
```

activity_main.xml `activity_main.xml` `activity_main.xml` .

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <Button
        android:id="@+id/my_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="My button"/>
</LinearLayout>
```

Kotlin .

```
import kotlinx.android.synthetic.main.activity_main.my_button

class MainActivity: Activity() {
    override fun onCreate(savedInstanceBundle: Bundle?) {
        super.onCreate(savedInstanceBundle)
        setContentView(R.layout.activity_main)
        // my_button is already casted to a proper type of "Button"
        // instead of being a "View"
        my_button.setText("Kotlin rocks!")
    }
}
```

* ID

```
// my_button can be used the same way as before
import kotlinx.android.synthetic.main.activity_main.*
```

Activities / Fragments / Views .

```
import kotlinx.android.synthetic.main.activity_main.my_button

class NotAView {
    init {
        // This sample won't compile!
        my_button.setText("Kotlin rocks!")
    }
}
```

Android Android Product Flavors . `build.gradle` `build.gradle` :

```
android {
    productFlavors {
        paid {
            ...
        }
        free {
            ...
        }
    }
}
```

, .

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <Button
        android:id="@+id/buy_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Buy full version"/>
</LinearLayout>
```

.

```
import kotlinx.android.synthetic.free.main_activity.buy_button
```

, .

```
mView.afterMeasured {
  // inside this block the view is completely drawn
  // you can get view's height/width, it.height / it.width
}
```

```
inline fun View.afterMeasured(crossinline f: View.() -> Unit) {
viewTreeObserver.addOnGlobalLayoutListener(object : ViewTreeObserver.OnGlobalLayoutListener {
    override fun onGlobalLayout() {
        if (measuredHeight > 0 && measuredWidth > 0) {
            viewTreeObserver.removeOnGlobalLayoutListener(this)
```

```
                f()
            }
        }
})
}
```

Kotlin Android : https://riptutorial.com/ko/kotlin/topic/9474/kotlin-android--

7: Kotlin

Examples

`kotlin-gradle-plugin` Kotlin Gradle . Kotlin . Kotlin `1.0.3` `kotlin-gradle-plugin 1.0.3` .

`gradle.properties` `ExtraPropertiesExtension` :

```
buildscript {
   ext.kotlin_version = '1.0.3'

   repositories {
     mavenCentral()
   }

   dependencies {
     classpath "org.jetbrains.kotlin:kotlin-gradle-plugin:$kotlin_version"
   }
}
```

. .

JVM

```
apply plugin: 'kotlin'
```

Android

```
apply plugin: 'kotlin-android'
```

JS

```
apply plugin: 'kotlin2js'
```

.

- kotlin : `src/main/kotlin`
- : `src/main/java`
- kotlin : `src/test/kotlin`
- java tests : `src/test/java`
- : `src/main/resources`
- : `src/test/resources`

`SourceSets` .

Kotlin .

```
dependencies {
    compile "org.jetbrains.kotlin:kotlin-stdlib:$kotlin_version"
}
```

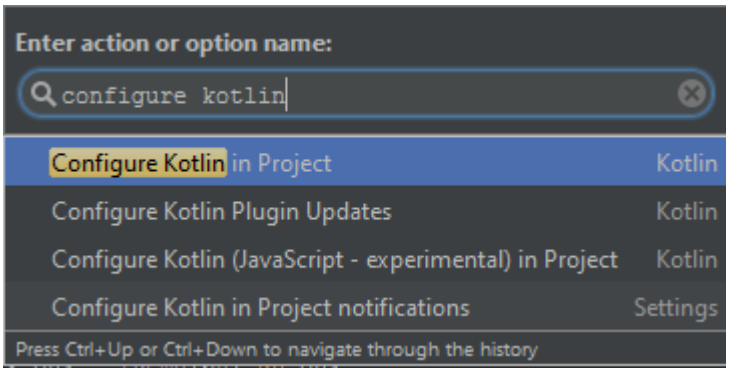
Kotlin Reflection `compile "org.jetbrains.kotlin:kotlin-reflect:$kotlin_version"` `compile "org.jetbrains.kotlin:kotlin-reflect:$kotlin_version"`

Android Studio

Android Studio Android Kotlin .

Kotlin > > > > JetBrains ...> Kotlin> Android Studio .

Android Studio `Ctrl` + `Shift` + `A` . " Kotlin " Enter .



Android Studio Gradle .

Java Kotlin `Ctrl` + `Shift` + `A` "Java Kotlin " . `.kt` Kotlin .

```java
package com.orangeflash81.myapplication;

public class Foo {
    private String name = "Joe Bloggs";

    String getName() { return name; }

    void setName(String value) { name = value; }
}
```

Groovy Gradle Kotlin

:

- gradle-script-kotlin
- / :
  - `build.gradle.kts`
  - `gradlew`
  - `gradlew.bat`
  - `settings.gradle`
- `build.gradle.kts` .
- Intellij Gradle .
- Intellij `build.gradle.kts` . Intellij .
- Gradle .

Windows . Gradle 3.3 . .

OSX Ubuntu .

, publicing , Jitpack , . .

Kotlin : https://riptutorial.com/ko/kotlin/topic/2501/kotlin--

8: kotlin

: Kotlin

Examples

kotlin.logging

```
class FooWithLogging {
  companion object: KLogging()

  fun bar() {
    logger.info { "hello $name" }
  }

  fun logException(e: Exception) {
    logger.error(e) { "Error occured" }
  }
}
```

kotlin.logging

kotlin : https://riptutorial.com/ko/kotlin/topic/3258/kotlin-

9: Kotlin RecyclerView

Kotlin RecyclerView .

Examples

Kotlin , activity_main.xml RecyclerView .

```
class MainActivity : AppCompatActivity(){

        lateinit var mRecyclerView : RecyclerView
        val mAdapter : RecyclerAdapter = RecyclerAdapter()

        override fun onCreate(savedInstanceState: Bundle?) {
            super.onCreate(savedInstanceState)
            setContentView(R.layout.activity_main)
            val toolbar = findViewById(R.id.toolbar) as Toolbar
            setSupportActionBar(toolbar)

            mRecyclerView = findViewById(R.id.recycler_view) as RecyclerView

            mRecyclerView.setHasFixedSize(true)
            mRecyclerView.layoutManager = LinearLayoutManager(this)
            mAdapter.RecyclerAdapter(getList(), this)
            mRecyclerView.adapter = mAdapter
        }

        private fun getList(): ArrayList<String> {
            var list : ArrayList<String> = ArrayList()
            for  (i in 1..10) { // equivalent of 1 <= i && i <= 10
                println(i)
                list.add("$i")
            }
            return list
        }
    }
```

main_item.xml .

```
class RecyclerAdapter : RecyclerView.Adapter<RecyclerAdapter.ViewHolder>() {

    var mItems: ArrayList<String>  = ArrayList()
    lateinit var mClick : OnClick

    fun RecyclerAdapter(item : ArrayList<String>, mClick : OnClick){
        this.mItems = item
        this.mClick = mClick;
    }

    override fun onBindViewHolder(holder: ViewHolder, position: Int) {
        val item = mItems[position]
        holder.bind(item, mClick, position)
    }

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): ViewHolder {
```

```
        val layoutInflater = LayoutInflater.from(parent.context)
        return ViewHolder(layoutInflater.inflate(R.layout.main_item, parent, false))
    }

    override fun getItemCount(): Int {
        return mItems.size
    }

    class ViewHolder(view: View) : RecyclerView.ViewHolder(view) {
        val card = view.findViewById(R.id.card) as TextView
        fun bind(str: String, mClick: OnClick, position: Int){
            card.text = str
            card.setOnClickListener { view ->
                mClick.onClickListner(position)
            }
        }
    }
}
```

Kotlin RecyclerView : https://riptutorial.com/ko/kotlin/topic/10143/kotlin-recyclerview

10: Kotlin

Kotlin .

1. Kotlin .kt.
2. Kotlin Any (Java Object).
3. val (immutable- assign once) var (mutables-) .
4. .
5. Unit. . 6. === . a === b a b true .

Examples

1. . .

```
    fun printHello(name: String?): Unit {
        if (name != null)
            println("Hello ${name}")
    }

    fun printHello(name: String?) {
        ...
    }
```

2.Single-Expression functions : = .

```
  fun double(x: Int): Int = x * 2
```

.

```
  fun double(x: Int) = x * 2
```

3. : .

```
In java:
    int num=10
    String s  = "i =" + i;

In Kotlin
    val num = 10
    val s = "i = $num"
```

4.In Kotlin null (nullable) (null) . , String null .

```
var a: String = "abc"
a = null // compilation error
```

null nullable . String? :

```
var b: String? = "abc"
b = null // ok
```

5.In Kotlin == . a == b .

```
   a?.equals(b) ?: (b === null)
```

Kotlin : https://riptutorial.com/ko/kotlin/topic/10648/kotlin-

11: Kotlin

Kotlin . , int .

Examples

x

```
repeat(10) { i ->
    println("This line will be printed 10 times")
    println("We are on the ${i + 1}. loop iteration")
}
```

for-loop :

```
val list = listOf("Hello", "World", "!")
for(str in list) {
    print(str)
}
```

Kotlin :

```
for(i in 0..9) {
    print(i)
}
```

iterating :

```
for((index, element) in iterable.withIndex()) {
    print("$element at index $index")
}
```

forEach .

```
iterable.forEach {
    print(it.toString())
}
```

```
it
```

While

while do-while .

```
while(condition) {
    doSomething()
}
```

```
do {
    doSomething()
} while (condition)
```

do-while .

.

```
while(true) {
    if(condition1) {
        continue // Will immediately start the next iteration, without executing the rest of
the loop body
    }
    if(condition2) {
        break // Will exit the loop completely
    }
}
```

break continue .

```
outer@ for(i in 0..10) {
    inner@ for(j in 0..10) {
        break       // Will break the inner loop
        break@inner // Will break the inner loop
        break@outer // Will break the outer loop
    }
}
```

forEach .

kotlin

```
//iterates over a map, getting the key and value at once

var map = hashMapOf(1 to "foo", 2 to "bar", 3 to "baz")

for ((key, value) in map) {
    println("Map[$key] = $value")
}
```

Kotlin .

```
fun factorial(n: Long): Long = if (n == 0) 1 else n * factorial(n - 1)

println(factorial(10)) // 3628800
```

factorial .

Kotlin .

map .

```
val numbers = listOf(1, 2, 3, 4, 5, 6, 7, 8, 9, 0)
val numberStrings = numbers.map { "Number $it" }
```

. . filter .

```
val numbers = listOf(1, 2, 3, 4, 5, 6, 7, 8, 9, 0)
val numberStrings = numbers.filter { it % 2 == 0 }.map { "Number $it" }
```

Kotlin : https://riptutorial.com/ko/kotlin/topic/2727/kotlin-

12:

Kotlin 4 .

: .

: .

Protected : .

: .

- `<visibility modifier> val/var <variable name> = <value>`

Examples

: `public val name = "Avijit"`

: `private val name = "Avijit"`

: `protected val name = "Avijit"`

: `internal val name = "Avijit"`

: https://riptutorial.com/ko/kotlin/topic/10019/--

13:

- (***Params***) = ...
- (***Params***) {...}
- (***Params***) : {...}
- < > (***Params***) : {...}
- (***Params***) : {...}
- { ***ArgName*** : ***ArgType*** -> ...}
- { ***ArgName*** -> ...}
- { ***ArgName*** -> ...}
- {(***ArgName*** : ***ArgType***) : -> ...}

Params	: ***Name : Type***
	( )
ArgName	
ArgType	***ArgName***
ArgName	ArgName

Examples

"Lambda Functions" . " " .

```
# Takes no parameters and returns anything
() -> Any?

# Takes a string and an integer and returns ReturnType
(arg1: String, arg2: Int) -> ReturnType
```

, vaguest `() -> Any?` , .

```
fun twice(x: () -> Any?) {
    x(); x();
}

fun main() {
    twice {
        println("Foo")
    } # => Foo
      # => Foo
}
```

. , - {} -> .

```
{ name: String ->
    "Your name is $name" //This is returned
}
```

.

.

:

```
{ argumentOne:String, argumentTwo:String ->
    "$argumentOne - $argumentTwo"
}
```

, it .

```
{ "Your name is $it" }
```

.

```
# These are identical
listOf(1, 2, 3, 4).map { it + 2 }
listOf(1, 2, 3, 4).map({ it + 2 })
```

:: . .

```
fun addTwo(x: Int) = x + 2
listOf(1, 2, 3, 4).map(::addTwo) # => [3, 4, 5, 6]
```

(ParamTypeA, ParamTypeB, ...) -> ReturnType . ParamTypeA , ParamTypeB ... `ReturnType1 .

```
fun foo(p0: Foo0, p1: Foo1, p2: Foo2): Bar {
    //...
}
println(::foo::class.java.genericInterfaces[0])
// kotlin.jvm.functions.Function3<Foo0, Foo1, Foo2, Bar>
// Human readable type: (Foo0, Foo1, Foo2) -> Bar
```

( ) . .

```
class Foo
fun Foo.foo(p0: Foo0, p1: Foo1, p2: Foo2): Bar {
    //...
}
val ref = Foo::foo
println(ref::class.java.genericInterfaces[0])
// kotlin.jvm.functions.Function4<Foo, Foo0, Foo1, Foo2, Bar>
// Human readable type: (Foo, Foo0, Foo1, Foo2) -> Bar
// takes 4 parameters, with receiver as first and actual parameters following, in their order
```

```
// this function can't be called like an extension function, though
val ref = Foo::foo
Foo().ref(Foo0(), Foo1(), Foo2()) // compile error

class Bar {
    fun bar()
}
print(Bar::bar) // works on member functions, too.
```

. .

```
object Foo
fun Foo.foo(p0: Foo0, p1: Foo1, p2: Foo2): Bar {
    //...
}
val ref = Foo::foo
println(ref::class.java.genericInterfaces[0])
// kotlin.jvm.functions.Function3<Foo0, Foo1, Foo2, Bar>
// Human readable type: (Foo0, Foo1, Foo2) -> Bar
// takes 3 parameters, receiver not needed

object Bar {
    fun bar()
}
print(Bar::bar) // works on member functions, too.
```

kotlin 1.1 , .

1.1.0

```
fun makeList(last: String?): List<String> {
    val list = mutableListOf("a", "b", "c")
    last?.let(list::add)
    return list
}
```

. .

. .

```
class Foo
class Bar {
    fun Foo.foo() {}
    val ref = Foo::foo // compile error
}
```

`fun` . . `Unit` . `{}` . `Unit` return .

```
fun sayMyName(name: String): String {
    return "Your name is $name"
}
```

:

```
fun sayMyName(name: String): String = "Your name is $name"
```

.

```
fun sayMyName(name: String) = "Your name is $name"
```

equals . .

```
fun sayMyName(name: String): String = "Your name is $name"
```

`inline` `inline` . C . . lambda .

```
inline fun sayMyName(name: String) = "Your name is $name"
```

C .

```
inline fun sayMyName() = "Your name is $name"

fun main() {
    val name = "Foo"
    sayMyName() # => Unresolved reference: name
}
```

Kotlin fixed symbolic representation (`+` `*`) . . `operator` :

```
data class IntListWrapper (val wrapped: List<Int>) {
    operator fun get(position: Int): Int = wrapped[position]
}

val a = IntListWrapper(listOf(1, 2, 3))
a[1] // == 2
```

.

: https://riptutorial.com/ko/kotlin/topic/1280/

14:

- :
- {parameterName : ParameterType, otherParameterName : OtherParameterType -> anExpression ()}
- :
- : (Int, Int) -> Int = {a, b -> a + b}
- `it`
- : (Int) -> Int = {it * it}
- :
- () -> ResultType
- (InputType) -> ResultType
- (InputType1, InputType2) -> ResultType

. .

```
data class User(val fistName: String, val lastName: String) {
    fun username(userNameGenerator: (String, String) -> String) =
        userNameGenerator(firstName, secondName)
}
```

.

```
val user = User("foo", "bar")
println(user.userName { firstName, secondName ->
     "${firstName.toUppercase}"_"${secondName.toUppercase}"
 }) // prints FOO_BAR
```

:

```
//valid:
val addition: (Int, Int) = { a, b -> a + b }
//valid:
val addition = { a: Int, b: Int -> a + b }
//error (type inference failure):
val addition = { a, b -> a + b }
```

, , `it` .

```
listOf(1,2,3).map { it * 2 } // [2,4,6]
```

Examples

```
val allowedUsers = users.filter { it.age > MINIMUM_AGE }
```

```
val isOfAllowedAge = { user: User -> user.age > MINIMUM_AGE }
val allowedUsers = users.filter(isOfAllowedAge)
```

:

```
object Benchmark {
    fun realtime(body: () -> Unit): Duration {
        val start = Instant.now()
        try {
            body()
        } finally {
            val end = Instant.now()
            return Duration.between(start, end)
        }
    }
}
```

:

```
val time = Benchmark.realtime({
    // some long-running code goes here ...
})
println("Executed the code in $time")
```

: https://riptutorial.com/ko/kotlin/topic/5878/-

15:

Examples

Null Null

`String` Null . Null `?` : `String?`

```
var string        : String = "Hello World!"
var nullableString: String? = null

string = nullableString    // Compiler error: Can't assign nullable to non-nullable type.
nullableString = string    // This will work however!
```

nullable .

, `?.` , null null .

```
val string: String? = "Hello World!"
print(string.length)   // Compile error: Can't directly access property of nullable type.
print(string?.length)  // Will print the string's length, or "null" if the string is null.
```

Idiom : null-checked

null-checked Kotlin `apply` .

```
obj?.apply {
    foo()
    bar()
}
```

`foo bar obj` (`this apply`) `obj` , , null.

nullable nullable `apply let let apply` .

```
nullable?.let { notnull ->
    notnull.foo()
    notnull.bar()
}
```

`notnull` lambda `it` .

null .

```
var string: String? = "Hello!"
print(string.length)  // Compile error
if(string != null) {
    // The compiler now knows that string can't be null
```

```
    print(string.length)  // It works now!
}
```

: null .

(:), .

null .

`Collection<T?> Collections<T> . filterNotNull .`

```
val a: List<Int?> = listOf(1, 2, 3, null)
val b: List<Int> = a.filterNotNull()
```

Null Coalescing / Elvis Operator

null if-else . elvis `?:` Kotlin .

:

```
val value: String = data?.first() ?: "Nothing here."
```

`data?.first() data null "Nothing here" data?.first() data?.first() .`

throw .

```
val value: String = data?.second()
    ?: throw IllegalArgumentException("Value can't be null!")
```

: (: `data!!.second()!!`) NullPointerExceptions `data!!.second()!!`

`!!` null null . `null KotlinNullPointerException` Throw.

```
val message: String? = null
println(message!!) //KotlinNullPointerException thrown, app crashes
```

(? :)

Kotlin `null reference` . nullable `a` . " `a` , `x` " .

```
var a: String? = "Nullable String Value"
```

`a` null . `a` , , . `if...else` .

```
val b: Int = if (a != null) a.length else -1
```

`Elvis` (: `?: Elvis . if...else` Elvis .

```
val b = a?.length ?: -1
```

`?:` (: `a?.length`) null elvis (`-1`) . .

: https://riptutorial.com/ko/kotlin/topic/2080/-

16:

Examples

String `string[index]` .

```
val str = "Hello, World!"
println(str[1]) // Prints e
```

String for .

```
for (c in str) {
    println(c)
}
```

Kotlin .

-
-

. . \t , \b , \n , \r , \' , \" , \\ \$. \uFF00 .

```
val s = "Hello, world!\n"
```

"""

```
val text = """
    for (c in "foo")
        print(c)
"""
```

trimMargin () .

```
val text = """
    |Tell me and I forget.
    |Teach me and I remember.
    |Involve me and I learn.
    |(Benjamin Franklin)
    """.trimMargin()
```

| trimMargin . : `trimMargin(">")` .

. . $.

```
val i = 10
val s = "i = $i" // evaluates to "i = 10"
```

:

```
val s = "abc"
val str = "$s.length is ${s.length}" // evaluates to "abc.length is 3"
```

.

```
val str = "\$foo" // evaluates to "$foo"
```

. .

```
val price = """
${'$'}9.99
"""
```

Kotlin == .

```
val str1 = "Hello, World!"
val str2 = "Hello," + " World!"
println(str1 == str2) // Prints true
```

=== .

```
val str1 = """
    |Hello, World!
    """.trimMargin()

val str2 = """
    #Hello, World!
    """.trimMargin("#")

val str3 = str1

println(str1 == str2) // Prints true
println(str1 === str2) // Prints false
println(str1 === str3) // Prints true
```

: https://riptutorial.com/ko/kotlin/topic/8285/

17:

.

( ) (searrospect) .

JVM JAR (`kotlin-reflect.jar` . JS .

Examples

`KClass` .

```
val c1 = String::class
val c2 = MyClass::class
```

Kotlin . .

```
fun isPositive(x: Int) = x > 0

val numbers = listOf(-2, -1, 0, 1, 2)
println(numbers.filter(::isPositive)) // [1, 2]
```

Kotlin `KClass` Java `Class` `.java` .

```
val stringKClass: KClass<String> = String::class
val c1: Class<String> = stringKClass.java

val c2: Class<MyClass> = MyClass::class.java
```

`KClass` .

`Example` `BaseExample` :

```
open class BaseExample(val baseField: String)

class Example(val field1: String, val field2: Int, baseField: String):
    BaseExample(baseField) {

    val field3: String
        get() = "Property without backing field"

    val field4 by lazy { "Delegated value" }

    private val privateField: String = "Private value"
}
```

.

```
val example = Example(field1 = "abc", field2 = 1, baseField = "someText")
```

```
example::class.memberProperties.forEach { member ->
    println("${member.name} -> ${member.get(example)}")
}
```

. `private val privateField` private `member.get(example)` . . Java getter . `private val` getter .

.

```
fun isFieldAccessible(property: KProperty1<*, *>): Boolean {
    return property.javaGetter?.modifiers?.let { !Modifier.isPrivate(it) } ?: false
}

val example = Example(field1 = "abc", field2 = 1, baseField = "someText")

example::class.memberProperties.filter { isFieldAccessible(it) }.forEach { member ->
    println("${member.name} -> ${member.get(example)}")
}
```

.

```
example::class.memberProperties.forEach { member ->
    member.isAccessible = true
    println("${member.name} -> ${member.get(example)}")
}
```

, .

```
class TestClass {
    val readOnlyProperty: String
        get() = "Read only!"

    var readWriteString = "asd"
    var readWriteInt = 23

    var readWriteBackedStringProperty: String = ""
        get() = field + '5'
        set(value) { field = value + '5' }

    var readWriteBackedIntProperty: Int = 0
        get() = field + 1
        set(value) { field = value - 1 }

    var delegatedProperty: Int by TestDelegate()

    private var privateProperty = "This should be private"

    private class TestDelegate {
        private var backingField = 3

        operator fun getValue(thisRef: Any?, prop: KProperty<*>): Int {
            return backingField
        }

        operator fun setValue(thisRef: Any?, prop: KProperty<*>, value: Int) {
            backingField += value
```

```
        }
    }
}
```

. .

```
val instance = TestClass()
TestClass::class.memberProperties
        .filter{ prop.visibility == KVisibility.PUBLIC }
        .filterIsInstance<KMutableProperty<*>>()
        .forEach { prop ->
            System.out.println("${prop.name} -> ${prop.get(instance)}")
        }
```

`String "Our Value"` . Kotlin Java VM , Type Erasure , `List<String>` Properties `List<Any>` .

.

```
val instance = TestClass()
TestClass::class.memberProperties
        .filter{ prop.visibility == KVisibility.PUBLIC }
        // We only want strings
        .filter{ it.returnType.isSubtypeOf(String::class.starProjectedType) }
        .filterIsInstance<KMutableProperty<*>>()
        .forEach { prop ->
            // Instead of printing the property we set it to some value
            prop.setter.call(instance, "Our Value")
        }
```

: https://riptutorial.com/ko/kotlin/topic/2402/

18:

Examples

Kotlin `Array<T>` .

`emptyArray<T>()` .

```
val empty = emptyArray<String>()
```

:

```
var strings = Array<String>(size = 5, init = { index -> "Item #$index" })
print(Arrays.toString(a)) // prints "[Item #0, Item #1, Item #2, Item #3, Item #4]"
print(a.size) // prints 5
```

`get(index: Int): T` `set(index: Int, value: T)` .

```
strings.set(2, "ChangedItem")
print(strings.get(2)) // prints "ChangedItem"

// You can use subscription as well:
strings[2] = "ChangedItem"
print(strings[2]) // prints "ChangedItem"
```

`Array<T>` .

		JVM
BooleanArray	booleanArrayOf(true, false)	boolean[]
ByteArray	byteArrayOf(1, 2, 3)	byte[]
CharArray	charArrayOf('a', 'b', 'c')	char[]
DoubleArray	doubleArrayOf(1.2, 5.0)	double[]
FloatArray	floatArrayOf(1.2, 5.0)	float[]
IntArray	intArrayOf(1, 2, 3)	int[]
LongArray	longArrayOf(1, 2, 3)	long[]
ShortArray	shortArrayOf(1, 2, 3)	short[]

`average()` `Byte` , `Int` , `Long` , `Short` , `Double` , `Float` `Double` .

```
val doubles = doubleArrayOf(1.5, 3.0)
print(doubles.average()) // prints 2.25

val ints = intArrayOf(1, 4)
```

```
println(ints.average()) // prints 2.5
```

`component1()` , `component2()` , ... `component5()` .

`getOrNull(index: Int)` index null .

`first()` , `last()`

`toHashSet()` `HashSet<T>` .

`sortedArray()` , `sortedArrayDescending()` .

`sort()` , `sortDescending` .

`min()` , `max()`

Java enhanced `: in` .

```
val asc = Array(5, { i -> (i * i).toString() })
for(s : String in asc){
    println(s);
}
```

for .

```
val asc = Array(5, { i -> (i * i).toString() })
for(s in asc){
    println(s);
}
```

```
val a = arrayOf(1, 2, 3) // creates an Array<Int> of size 3 containing [1, 2, 3].
```

```
val a = Array(3) { i -> i * 2 } // creates an Array<Int> of size 3 containing [0, 2, 4]
```

```
val a = arrayOfNulls<Int>(3) // creates an Array<Int?> of [null, null, null]
```

null . Null .

: https://riptutorial.com/ko/kotlin/topic/5722/

19:

in ! in .. rangeTo . .

Examples

(IntRange, LongRange, CharRange) . . for-loop .

```
for (i in 1..4) print(i) // prints "1234"
for (i in 4..1) print(i) // prints nothing
```

downTo ()

? . downTo () .

```
for (i in 4 downTo 1) print(i) // prints "4321"
```

step ()

1 ? step () .

```
for (i in 1..4 step 2) print(i) // prints "13"
for (i in 4 downTo 1 step 2) print(i) // prints "42"
```

until .

```
for (i in 1 until 10) { // i in [1, 10), 10 is excluded
println(i)
}
```

: https://riptutorial.com/ko/kotlin/topic/10121/

20:

. `(String) -> Boolean` `Pair<Person, Person>` generic .

. .

- **typealias - =**

. JVM . .

Examples

```
typealias StringValidator = (String) -> Boolean
typealias Reductor<T, U, V> = (T, U) -> V
```

```
typealias Parents = Pair<Person, Person>
typealias Accounts = List<Account>
```

: https://riptutorial.com/ko/kotlin/topic/9453/-

21:

Examples

DTO (POJO / POCO)

kotlin . `data` .

```
data class User(var firstname: String, var lastname: String, var age: Int)
```

`User` .

- getter setter (`val` getter)
- `equals()`
- `hashcode()`
- `toString()`
- `copy()`
- `componentN()` (`N`)

.

```
data class User(var firstname: String = "Joe", var lastname: String = "Bloggs", var age: Int =
20)
```

.

```
val list = listOf(1,2,3,4,5,6)

//filter out even numbers

val even = list.filter { it % 2 == 0 }

println(even) //returns [2,4]
```

public .

. private . . , :

```
class MyTable private constructor(table: Table<Int, Int, Int>) : Table<Int, Int, Int> by table
{

    constructor() : this(TreeBasedTable.create()) // or a different type of table if desired

}
```

`MyTable` `MyTable()` . `MyTable` `Table<Int, Int, Int>` . .

SO .

Kotlin Serializable serialVersionUid

Kotlin `serialVersionUID` .

(`MySpecialCase` `private const val MySpecialCase` .

```
class MySpecialCase : Serializable {
    companion object {
        private const val serialVersionUID: Long = 123
    }
}
```

getter / setter .

```
class MySpecialCase : Serializable {
    companion object {
        private val serialVersionUID: Long = 123
    }
}
```

, `getSerialVersionUID` .

```
class MySpecialCase : Serializable {
    companion object {
        @JvmStatic private val serialVersionUID: Long = 123
    }
}
```

, `getSerialVersionUID` `MySpecialCase` getter .

`serialVersionUID Serializable` .

Kotlin

Kotlin Java .

```
fun doSomething() {
   someOtherAction()
   return this
}
```

.

```
fun <T: Any> T.fluently(func: ()->Unit): T {
    func()
    return this
}
```

.

```
fun doSomething() {
```

```
    return fluently { someOtherAction() }
}
```

let nullable .

let .:

```
val str = "foo"
str.let {
    println(it) // it
}
```

"foo" Unit .

let also let . also Unit reutrn .

. null null let also :

```
val str: String? = someFun()
str?.let {
    println(it)
}
```

str null let . null (?).

.

apply :

this this .

kdoc apply . apply this apply . .:

```
File(dir).apply { mkdirs() }
```

:

```
fun makeDir(String path): File {
    val result = new File(path)
    result.mkdirs()
    return result
}
```

: https://riptutorial.com/ko/kotlin/topic/2273/

22:

`object` . java Singleton ( ) . java . .

Examples

/ repalcement

```
object CommonUtils {

    var anyname: String ="Hello"

    fun dispMsg(message: String) {
        println(message)
    }
}
```

.

```
CommonUtils.anyname
CommonUtils.dispMsg("like static call")
```

Kotlin . `SomeSingleton.INSTANCE` .

.

```
public enum SharedRegistry {
    INSTANCE;
    public void register(String key, Object thing) {}
}

public static void main(String[] args) {
    SharedRegistry.INSTANCE.register("a", "apple");
    SharedRegistry.INSTANCE.register("b", "boy");
    SharedRegistry.INSTANCE.register("c", "cat");
    SharedRegistry.INSTANCE.register("d", "dog");
}
```

kotlin .

```
object SharedRegistry {
    fun register(key: String, thing: Object) {}
}

fun main(Array<String> args) {
    SharedRegistry.register("a", "apple")
    SharedRegistry.register("b", "boy")
    SharedRegistry.register("c", "cat")
    SharedRegistry.register("d", "dog")
}
```

obvoiusly .

: https://riptutorial.com/ko/kotlin/topic/10152/--

23:

Java Kotlin enum . (enum `EnumClass`).

```
EnumClass.valueOf(value: String): EnumClass
EnumClass.values(): Array<EnumClass>
```

`valueOf()` `IllegalArgumentException` .

enum enum .

```
val name: String
val ordinal: Int
```

Comparable .

Examples

Enum .

```
enum class Color(val rgb: Int) {
    RED(0xFF0000),
    GREEN(0x00FF00),
    BLUE(0x0000FF)
}
```

Enum (,) . (;) enum .

`abstract` enum .

```
enum class Color {
    RED {
        override val rgb: Int = 0xFF0000
    },
    GREEN {
        override val rgb: Int = 0x00FF00
    },
    BLUE {
        override val rgb: Int = 0x0000FF
    }

    ;

    abstract val rgb: Int

    fun colorString() = "#%06X".format(0xFFFFFF and rgb)
}
```

enum

```
enum class Color {
  RED, GREEN, BLUE
}
```

. Enum .

. .

```
enum class Planet(var population: Int = 0) {
    EARTH(7 * 100000000),
    MARS();

    override fun toString() = "$name[population=$population]"
}

 println(Planet.MARS) // MARS[population=0]
 Planet.MARS.population = 3
 println(Planet.MARS) // MARS[population=3]
```

: https://riptutorial.com/ko/kotlin/topic/2286/-

24:

Examples

try-catch-finally

Kotlin catch Java .

```
try {
    doSomething()
}
catch(e: MyException) {
    handle(e)
}
finally {
    cleanup()
}
```

catch .

```
try {
    doSomething()
}
catch(e: FileSystemException) {
    handle(e)
}
catch(e: NetworkException) {
    handle(e)
}
catch(e: MemoryException) {
    handle(e)
}
finally {
    cleanup()
}
```

try .

```
val s: String? = try { getString() } catch (e: Exception) { null }
```

Kotlin .

```
fun fileToString(file: File) : String {
    //readAllBytes throws IOException, but we can omit catching it
    fileContent = Files.readAllBytes(file)
    return String(fileContent)
}
```

: https://riptutorial.com/ko/kotlin/topic/7246/

25:

Kotlin . (:) . .

Examples

```
val foo : Int by lazy { 1 + 1 }
println(foo)
```

2 .

```
var foo : Int by Delegates.observable("1") { property, oldValue, newValue ->
    println("${property.name} was changed from $oldValue to $newValue")
}
foo = 2
```

```
foo was changed from 1 to 2
```

```
val map = mapOf("foo" to 1)
val foo : String by map
println(foo)
```

```
1
```

```
class MyDelegate {
    operator fun getValue(owner: Any?, property: KProperty<*>): String {
        return "Delegated value"
    }
}

val foo : String by MyDelegate()
println(foo)
```

```
Delegated value Delegated value
```

Delegate .

Kotlin Null Type `WeakReference<T>` .

`WeakReference` .

:

```
class MyMemoryExpensiveClass {
    companion object {
        var reference: WeakReference<MyMemoryExpensiveClass>? = null

        fun doWithReference(block: (MyMemoryExpensiveClass) -> Unit) {
            reference?.let {
                it.get()?.let(block)
```

```
            }
        }
    }

    init {
        reference = WeakReference(this)
    }
}
```

WeakReference . .

```
class WeakReferenceDelegate<T>(initialValue: T? = null) : ReadWriteProperty<Any, T?> {
    var reference = WeakReference(initialValue)
        private set

    override fun getValue(thisRef: Any, property: KProperty<*>): T? = reference.get()

    override fun setValue(thisRef: Any, property: KProperty<*>, value: T?) {
        reference = WeakReference(value)
    }
}
```

`WeakReference` nullable !

```
class MyMemoryExpensiveClass {
    companion object {
        var reference: MyMemoryExpensiveClass? by
WeakReferenceDelegate<MyMemoryExpensiveClass>()

        fun doWithReference(block: (MyMemoryExpensiveClass) -> Unit) {
            reference?.let(block)
        }
    }

    init {
        reference = this
    }
}
```

: https://riptutorial.com/ko/kotlin/topic/10571/--

26:

.

3 .

1. .
2. .
3. .

Kotlin

Kotlin . :

-
-
- Ktor
-

Examples

. JFrame .

```
import javax.swing.*

fun JFrame.menuBar(init: JMenuBar.() -> Unit) {
    val menuBar = JMenuBar()
    menuBar.init()
    setJMenuBar(menuBar)
}

fun JMenuBar.menu(caption: String, init: JMenu.() -> Unit) {
    val menu = JMenu(caption)
    menu.init()
    add(menu)
}

fun JMenu.menuItem(caption: String, init: JMenuItem.() -> Unit) {
    val menuItem = JMenuItem(caption)
    menuItem.init()
    add(menuItem)
}
```

.

```
class MyFrame : JFrame() {
    init {
        menuBar {
            menu("Menu1") {
                menuItem("Item1") {
                    // Initialize MenuItem with some Action
```

```
                }
                menuItem("Item2") {}
            }
            menu("Menu2") {
                menuItem("Item3") {}
                menuItem("Item4") {}
            }
        }
    }
}
```

: https://riptutorial.com/ko/kotlin/topic/6010/--

27:

: Kotlin :

Examples

Kotlin .

```
interface MyInterface {
    fun bar()
}
```

.

```
class Child : MyInterface {
   override fun bar() {
       print("bar() was called")
   }
}
```

Kotlin .

```
interface MyInterface {
    fun withImplementation() {
      print("withImplementation() was called")
    }
}
```

.

```
class MyClass: MyInterface {
    // No need to reimplement here
}
val instance = MyClass()
instance.withImplementation()
```

getter setter .

```
interface MyInterface2 {
    val helloWorld
        get() = "Hello World!"
}
```

.

```
interface MyInterface3 {
    // this property won't compile!
    var helloWorld: Int
        get() = field
        set(value) { field = value }
```

```
}
```

.

```
interface A {
    fun notImplemented()
    fun implementedOnlyInA() { print("only A") }
    fun implementedInBoth() { print("both, A") }
    fun implementedInOne() { print("implemented in A") }
}

interface B {
    fun implementedInBoth() { print("both, B") }
    fun implementedInOne() // only defined
}

class MyClass: A, B {
    override fun notImplemented() { print("Normal implementation") }

    // implementedOnlyInA() can by normally used in instances

    // class needs to define how to use interface functions
    override fun implementedInBoth() {
        super<B>.implementedInBoth()
        super<A>.implementedInBoth()
    }

    // even if there's only one implementation, there multiple definitions
    override fun implementedInOne() {
        super<A>.implementedInOne()
        print("implementedInOne class implementation")
    }
}
```

. .

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

    val propertyWithImplementation: String
        get() = "foo"

    fun foo() {
        print(property)
    }
}

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

2 , . . .

```
interface FirstTrait {
    fun foo() { print("first") }
    fun bar()
}
```

```
interface SecondTrait {
    fun foo() { print("second") }
    fun bar() { print("bar") }
}

class ClassWithConflict : FirstTrait, SecondTrait {
    override fun foo() {
        super<FirstTrait>.foo()  // delegate to the default implementation of FirstTrait
        super<SecondTrait>.foo() // delegate to the default implementation of SecondTrait
    }

    // function bar() only has a default implementation in one interface and therefore is ok.
}
```

```
interface MyInterface {
    fun funcOne() {
        //optional body
        print("Function with default implementation")
    }
}
```

super .

```
super.funcOne()
```

: https://riptutorial.com/ko/kotlin/topic/900/

28:

Examples

:

" " ." " `when` . regex .

```
import kotlin.text.regex

var string = /* some string */

val regex1 = Regex( /* pattern */ )
val regex2 = Regex( /* pattern */ )
/* etc */

when {
    regex1.matches(string) -> /* do stuff */
    regex2.matches(string) -> /* do stuff */
    /* etc */
}
```

:

" " . `when` .

```
import kotlin.text.regex

var string = /* some string */

when {
    Regex( /* pattern */ ).matches(string) -> /* do stuff */
    Regex( /* pattern */ ).matches(string) -> /* do stuff */
    /* etc */
}
```

:

syntax `when` " - " . `when` `whenEntry` `when` `whenEntry` . "immutable locals" "anonymous temporaries" .

```
import kotlin.text.regex

var string = /* some string */

when (RegexWhenArgument(string)) {
    Regex( /* pattern */ ) -> /* do stuff */
    Regex( /* pattern */ ) -> /* do stuff */
```

```
    /* etc */
}
```

`when` :

```
class RegexWhenArgument (val whenArgument: CharSequence) {
    operator fun equals(whenEntry: Regex) = whenEntry.matches(whenArgument)
    override operator fun equals(whenEntry: Any?) = (whenArgument == whenEntry)
}
```

Kotlin

`Regex` , `Regex` null , .

RegEx

Kotlin `Regex(pattern: String)` `find(..)` `replace(..)` .

`input` c d true `Regex` .

```
val regex = Regex(pattern = "c|d")
val matched = regex.containsMatchIn(input = "abc")    // matched: true
```

`Regex` `pattern` `input` . . `containsMatchIn(..)` .

Null

`find(..)` `matchEntire(..)` `MatchResult?` `MatchResult?` . ? `MatchResult` Kotlin null .

`find(..)` null Kotlin `Regex` null .

```
val matchResult =
    Regex("c|d").find("efg")           // matchResult: null
val a = matchResult?.value             // a: null
val b = matchResult?.value.orEmpty()   // b: ""
a?.toUpperCase()                       // Still needs question mark. => null
b.toUpperCase()                        // Accesses the function directly. => ""
```

`orEmpty()` `b` null `? b` .

null Kotlin Java null `!!` :

```
a!!.toUpperCase()                      // => KotlinNullPointerException
```

Kotlin Java Java . .

```
"""\d{3}-\d{3}-\d{4}""" // raw Kotlin string
"\\d{3}-\\d{3}-\\d{4}"  // standard Java string
```

(: CharSequence, startIndex : int) : MatchResult?

input Regex pattern . Matchresult? Matchresult? startIndex input null . MatchResult? MatchResult? value . startIndex 0.

.

```
val phoneNumber :String? = Regex(pattern = """\d{3}-\d{3}-\d{4}""")
    .find(input = "phone: 123-456-7890, e..")?.value // phoneNumber: 123-456-7890
```

input phoneNumber null .

findAll (input : CharSequence, startIndex : Int) : .

regex pattern input .

:

```
val matchedResults = Regex(pattern = """\d+""").findAll(input = "ab12cd34ef")
val result = StringBuilder()
for (matchedText in matchedResults) {
    result.append(matchedText.value + " ")
}

println(result) // => 12 34
```

matchedResults MatchResult . input findAll(..) .

matchEntire (: CharSequence) : MatchResult?

input pattern input . null .

.

```
val a = Regex("""\d+""").matchEntire("100")?.value             // a: 100
val b = Regex("""\d+""").matchEntire("100 dollars")?.value     // b: null
```

matches (input : CharSequence) : Boolean

true . .

.

```
val regex = Regex(pattern = """\d+""")
regex.matches(input = "50")              // => true
regex.matches(input = "50 dollars")      // => false
```

containsMatchIn (input : CharSequence) : Boolean

true . .

.

```
Regex("""\d+""").containsMatchIn("50 dollars")       // => true
Regex("""\d+""").containsMatchIn("Fifty dollars")    // => false
```

split (: CharSequence, limit : Int) :

.

:

```
val a = Regex("""\d+""").split("ab12cd34ef")     // a: [ab, cd, ef]
val b = Regex("""\d+""").split("This is a test") // b: [This is a test]
```

. input . .

(: CharSequence, :) :

input pattern .

x :

```
val result = Regex("""\d+""").replace("ab12cd34ef", "x") // result: abxcdxef
```

: https://riptutorial.com/ko/kotlin/topic/8364/

29:

, . List *generic* .

Generics .

- *ClassName* < ***TypeName*** >
- class *ClassName* <*>
- *ClassName* <in ***UpperBound*** >
- *ClassName* <out ***LowerBound*** >
- < ***TypeName*** : ***UpperBound*** >

TypeName
ClassName

Nullable.

Kotlin Generics `T Any?` . :

```
class Consumer<T>
```

`T T: Any?` . Null `T: Any` . :

```
class Consumer<T: Any>
```

Examples

- .

```
  class Consumer<in T> { fun consume(t: T) { ... } }

  fun charSequencesConsumer() : Consumer<CharSequence>() = ...

  val stringConsumer : Consumer<String> = charSequenceConsumer() // OK since in-projection
  val anyConsumer : Consumer<Any> = charSequenceConsumer() // Error, Any cannot be passed

  val outConsumer : Consumer<out CharSequence> = ... // Error, T is `in`-parameter
```

- `T List<out T> T Comparator<in T>` .

Java .

- :

```
val takeList : MutableList<out SomeType> = ... // Java: List<? extends SomeType>

val takenValue : SomeType = takeList[0] // OK, since upper bound is SomeType

takeList.add(takenValue) // Error, lower bound for generic is not specified
```

:

```
val putList : MutableList<in SomeType> = ... // Java: List<? super SomeType>

val valueToPut : SomeType = ...
putList.add(valueToPut) // OK, since lower bound is SomeType

putList[0] // This expression has type Any, since no upper bound is specified
```

```
val starList : MutableList<*> = ... // Java: List<?>

starList[0] // This expression has type Any, since no upper bound is specified
starList.add(someValue) // Error, lower bound for generic is not specified
```

:

- Java Kotlin Variant Generics .

: https://riptutorial.com/ko/kotlin/topic/1147/

30:

`switch` , `when` fall-through . , `break` . bahaviors .

```
when (x) {
    "foo", "bar" -> println("either foo or bar")
    else -> println("didn't match anything")
}
```

Examples

if

```
val str = "Hello!"
if (str.length == 0) {
    print("The string is empty!")
} else if (str.length > 5) {
    print("The string is short!")
} else {
    print("The string is long!")
}
```

else-branches if .

If

if .

```
val str = if (condition) "Condition met!" else "Condition not met!"
```

`if` -statement `else` -branch .

`else if` .

```
val str = if (condition1){
    "Condition1 met!"
   } else if (condition2) {
    "Condition2 met!"
   } else {
    "Conditions not met!"
   }
```

: Kotlin . `val str: String =` .

if-else-if when-

when else-if-branches if .

```
when {
    str.length == 0 -> print("The string is empty!")
    str.length > 5  -> print("The string is short!")
    else            -> print("The string is long!")
}
```

if-else-if :

```
if (str.length == 0) {
    print("The string is empty!")
} else if (str.length > 5) {
    print("The string is short!")
} else {
    print("The string is long!")
}
```

if else . :

```
when {
    condition -> {
        doSomething()
        doSomeMore()
    }
    else -> doSomethingElse()
}
```

when-statement

`when` statement . null `equals` `==` . .

```
when (x) {
    "English" -> print("How are you?")
    "German" -> print("Wie geht es dir?")
    else -> print("I don't know that language yet :(")
}
```

when .

```
val names = listOf("John", "Sarah", "Tim", "Maggie")
when (x) {
    in names -> print("I know that name!")
    !in 1..10 -> print("Argument was not in the range from 1 to 10")
    is String -> print(x.length) // Due to smart casting, you can use String-functions here
}
```

When

if when .

```
val greeting = when (x) {
    "English" -> "How are you?"
    "German" -> "Wie geht es dir?"
    else -> "I don't know that language yet :("
```

```
}
print(greeting)
```

when else .

when

`when` `enum` :

```
enum class Day {
    Sunday,
    Monday,
    Tuesday,
    Wednesday,
    Thursday,
    Friday,
    Saturday
}

fun doOnDay(day: Day) {
    when(day) {
        Day.Sunday ->      // Do something
        Day.Monday, Day.Tuesday ->      // Do other thing
        Day.Wednesday ->  // ...
        Day.Thursday ->   // ...
        Day.Friday ->     // ...
        Day.Saturday ->   // ...
    }
}
```

(`Monday` `Tuedsay`) `enum` .

`.else` .

```
fun doOnDay(day: Day) {
    when(day) {
        Day.Monday ->      // Work
        Day.Tuesday ->     // Work hard
        Day.Wednesday ->  // ...
        Day.Thursday ->   //
        Day.Friday ->     //
        else ->           // Party on weekend
    }
}
```

`if-then-else` , `when` `enum` .

kotlin `enum` .

: https://riptutorial.com/ko/kotlin/topic/2685/

31:

, Kotlin (, ,) . API .

- listOf, mapOf setOf .
- arrayListOf, hashMapOf, hashSetOf, linkedMapOf (LinkedHashMap), linkedSetOf (LinkedHashSet), mutableListOf (Kotlin MultableList), mutableMapOf (Kotlin MultableMap), mutableSetOf (Kotlin MultibleSet), sortedMapOf sortedSetOf
- first (), last (), get () filter, map, join, reduce .

Examples

```
// Create a new read-only List<String>
val list = listOf("Item 1", "Item 2", "Item 3")
println(list) // prints "[Item 1, Item 2, Item 3]"
```

```
// Create a new read-only Map<Integer, String>
val map = mapOf(Pair(1, "Item 1"), Pair(2, "Item 2"), Pair(3, "Item 3"))
println(map) // prints "{1=Item 1, 2=Item 2, 3=Item 3}"
```

```
// Create a new read-only Set<String>
val set = setOf(1, 3, 5)
println(set) // prints "[1, 3, 5]"
```

: https://riptutorial.com/ko/kotlin/topic/8846/

32:

()

Examples

1

( )

```
fun main(args: Array<String>) {
    launch(CommonPool) { // create new coroutine in common thread pool
        delay(1000L) // non-blocking delay for 1 second (default time unit is ms)
        println("World!") // print after delay
    }
    println("Hello,") // main function continues while coroutine is delayed
    Thread.sleep(2000L) // block main thread for 2 seconds to keep JVM alive
}
```

```
Hello,
World!
```

: https://riptutorial.com/ko/kotlin/topic/10936/-

33:

Examples

nullable toString () .

Kotlin `toString` `String?` null `String?` .

Android `EditText` `EditText` .

.

```
// Incorrect:
val text = view.textField?.text.toString() ?: ""
```

`"null"` .

```
// Correct:
val text = view.textField?.text?.toString() ?: ""
```

: https://riptutorial.com/ko/kotlin/topic/6608/--

34:

. :

, Apples , Oranges Pears . , , . .

. Fruit ? getFruit() ?

Fruit . !

- { }
- { } : { } ({ })
- override { }
- {DC-Object} {Base Class}. == true

Init	
- " "	

Examples

: "

Kotlin **final** .

open .

```
open class Thing {
    // I can now be extended!
}
```

: , open .

:

```
open class BaseClass {
    val x = 10
}
```

:

```
class DerivedClass: BaseClass() {
```

```
    fun foo() {
        println("x is equal to " + x)
    }
}
```

:

```
fun main(args: Array<String>) {
    val derivedClass = DerivedClass()
    derivedClass.foo() // prints: 'x is equal to 10'
}
```

:

```
open class Person {
    fun jump() {
        println("Jumping...")
    }
}
```

:

```
class Ninja: Person() {
    fun sneak() {
        println("Sneaking around...")
    }
}
```

Person .

```
fun main(args: Array<String>) {
    val ninja = Ninja()
    ninja.jump()  // prints: 'Jumping...'
    ninja.sneak() // prints: 'Sneaking around...'
}
```

( ):

```
abstract class Car {
    abstract val name: String;
    open var speed: Int = 0;
}

class BrokenCar(override val name: String) : Car() {
    override var speed: Int
        get() = 0
        set(value) {
            throw UnsupportedOperationException("The car is bloken")
        }
}
```

```
fun main(args: Array<String>) {
    val car: Car = BrokenCar("Lada")
    car.speed = 10
}
```

:

```
interface Ship {
    fun sail()
    fun sink()
}

object Titanic : Ship {

    var canSail = true

    override fun sail() {
        sink()
    }

    override fun sink() {
        canSail = false
    }
}
```

: https://riptutorial.com/ko/kotlin/topic/5622/-

35:

Kotlin . .

Examples

.

```
interface Foo {
    fun example()
}

class Bar {
    fun example() {
        println("Hello, world!")
    }
}

class Baz(b : Bar) : Foo by b

Baz(Bar()).example()
```

```
Hello, world!
```

: https://riptutorial.com/ko/kotlin/topic/10575/-

36:

Examples

. .

```
annotation class Strippable
```

.

```
    @Target(AnnotationTarget.CLASS, AnnotationTarget.FUNCTION,
AnnotationTarget.VALUE_PARAMETER, AnnotationTarget.EXPRESSION)
    annotation class Strippable
```

.

```
annotation class Strippable(val importanceValue: Int)
```

.

- Java (Int, Long) .
-
- (Foo :: class)
-
-
-

.

- `@Target` : (, , ,) .
- `@Retention` (true).
- `@Repeatable` .
- `@MustBeDocumented` API API .

:

```
@Target(AnnotationTarget.CLASS, AnnotationTarget.FUNCTION,
        AnnotationTarget.VALUE_PARAMETER, AnnotationTarget.EXPRESSION)
@Retention(AnnotationRetention.SOURCE)
@MustBeDocumented
annotation class Fancy
```

: https://riptutorial.com/ko/kotlin/topic/4074/-

37:

- **Vararg** : `vararg` .
- : "" (`*`).

Examples

: vararg

`vararg` .

```
fun printNumbers(vararg numbers: Int) {
    for (number in numbers) {
        println(number)
    }
}
```

.

```
printNumbers(0, 1)                // Prints "0" "1"
printNumbers(10, 20, 30, 500)     // Prints "10" "20" "30" "500"
```

: Vararg .

: vararg

`*` vararg .

...

```
fun printNumbers(vararg numbers: Int) {
    for (number in numbers) {
        println(number)
    }
}
```

.

```
val numbers = intArrayOf(1, 2, 3)
printNumbers(*numbers)

// This is the same as passing in (1, 2, 3)
```

...

```
val numbers = intArrayOf(1, 2, 3)
printNumbers(10, 20, *numbers, 30, 40)

// This is the same as passing in (10, 20, 1, 2, 3, 30, 40)
```

: https://riptutorial.com/ko/kotlin/topic/5835/--

38:

- fun TypeName.extensionName (params, ...) {/ * body * /} //
- fun <T : Any> TypeNameWithGenerics <T> .extensionName (params, ...) {/ * body * /} //
- myObj.extensionName (args, ...) //

. , ...

Examples

.

```
fun IntArray.addTo(dest: IntArray) {
    for (i in 0 .. size - 1) {
        dest[i] += this[i]
    }
}
```

IntArray . ( ) this this .

.

```
val myArray = intArrayOf(1, 2, 3)
intArrayOf(4, 5, 6).addTo(myArray)
```

:

. .

```
open class Super

class Sub : Super()

fun Super.myExtension() = "Defined for Super"

fun Sub.myExtension() = "Defined for Sub"

fun callMyExtension(myVar: Super) {
    println(myVar.myExtension())
}

callMyExtension(Sub())
```

myVar Super "Defined for Super" .

Int Long .

```
fun Long.humanReadable(): String {
    if (this <= 0) return "0"
    val units = arrayOf("B", "KB", "MB", "GB", "TB", "EB")
```

```
    val digitGroups = (Math.log10(this.toDouble())/Math.log10(1024.0)).toInt();
    return DecimalFormat("#,##0.#").format(this/Math.pow(1024.0, digitGroups.toDouble())) + "
" + units[digitGroups];
}

fun Int.humanReadable(): String {
    return this.toLong().humanReadable()
}
```

.

```
println(1999549L.humanReadable())
println(someInt.humanReadable())
```

Java 7+ Path

API . Java 7+ `Path exist` , `notExists deleteRecursively exist` .

```
fun Path.exists(): Boolean = Files.exists(this)
fun Path.notExists(): Boolean = !this.exists()
fun Path.deleteRecursively(): Boolean = this.toFile().deleteRecursively()
```

.

```
val dir = Paths.get(dirName)
if (dir.exists()) dir.deleteRecursively()
```

Kotlin .

```
val x: Path = Paths.get("dirName").apply {
    if (Files.notExists(this)) throw IllegalStateException("The important file does not
exist")
}
```

apply . . , .

```
infix inline fun <T> T.verifiedBy(verifyWith: (T) -> Unit): T {
    verifyWith(this)
    return this
}

infix inline fun <T: Any> T.verifiedWith(verifyWith: T.() -> Unit): T {
    this.verifyWith()
    return this
}
```

.

```
val x: Path = Paths.get("dirName") verifiedWith {
    if (Files.notExists(this)) throw IllegalStateException("The important file does not
exist")
}
```

.

`verifiedBy` `T T: Any? T: Any?` null `.verifiedWith` nullable .

ISO Java 8 Temporal

:

```
fun Temporal.toIsoString(): String = DateTimeFormatter.ISO_INSTANT.format(this)
```

.

```
val dateAsString = someInstant.toIsoString()
```

( )

- , `Something fromString fromString` :

```
class Something {
    companion object {}
}

class SomethingElse {
}

fun Something.Companion.fromString(s: String): Something = ...

fun SomethingElse.fromString(s: String): SomethingElse = ...

fun main(args: Array<String>) {
    Something.fromString("") //valid as extension function declared upon the
                             //companion object

    SomethingElse().fromString("") //valid, function invoked on instance not
                                   //statically

    SomethingElse.fromString("") //invalid
}
```

. Kotlin KT-9686 KT-13053 (`this`) . .

`color . lazy this colorCache` :

```
class KColor(val value: Int)

private val colorCache = mutableMapOf<KColor, Color>()

val KColor.color: Color
    get() = colorCache.getOrPut(this) { Color(value, true) }
```

.

Anko

```
inline fun <reified T : View> View.find(id: Int): T = findViewById(id) as T
inline fun <reified T : View> Activity.find(id: Int): T = findViewById(id) as T
inline fun <reified T : View> Fragment.find(id: Int): T = view?.findViewById(id) as T
inline fun <reified T : View> RecyclerView.ViewHolder.find(id: Int): T =
itemView?.findViewById(id) as T

inline fun <reified T : View> View.findOptional(id: Int): T? = findViewById(id) as? T
inline fun <reified T : View> Activity.findOptional(id: Int): T? = findViewById(id) as? T
inline fun <reified T : View> Fragment.findOptional(id: Int): T? = view?.findViewById(id) as?
T
inline fun <reified T : View> RecyclerView.ViewHolder.findOptional(id: Int): T? =
itemView?.findViewById(id) as? T
```

```
val yourButton by lazy { find<Button>(R.id.yourButtonId) }
val yourText by lazy { find<TextView>(R.id.yourTextId) }
val yourEdittextOptional by lazy { findOptional<EditText>(R.id.yourOptionEdittextId) }
```

: https://riptutorial.com/ko/kotlin/topic/613/-

S. No		Contributors
1	Kotlin	babedev, Community, cyberscientist, ganesshkumar, Ihor Kucherenko, Jayson Minard, mnoronha, neworld, Parker Hoyes, Ruckus T-Boom, Sach, Sean Reilly, Sheigutn, Simón Oroño, UnKnown, Urko Pineda
2	DSL	Dmitriy L, ice1000
3	Java 8 Stream Equivalents	Brad, Gerson, Jayson Minard, Piero Divasto, Sam
4	Java Kotlin	Thorsten Schleinzer
5	JUnit	jenglert
6	Kotlin Android	Jemo Mgebrishvili, Ritave
7	Kotlin	Aaron Christiansen, elect, madhead
8	kotlin	Konrad Jamrozik, olivierlemasle, oshai
9	Kotlin RecyclerView	Mohit Suthar
10	Kotlin	Shinoo Goyal
11	Kotlin	Ben Leggiero, JaseAnderson, mayojava, razzledazzle, Robin
12		Avijit Karmakar
13		Aaron Christiansen, baha, Caelum, glee8e, Jayson Minard, KeksArmee, madhead, Spidfire
14		memoizr, Rich Kuzsma
15		KeksArmee, Kirill Rakhman, piotrek1543, razzledazzle, Robin, SerCe, Spidfire, technerd, Thorsten Schleinzer
16		Januson, Sam
17		atok, Kirill Rakhman, madhead, Ritave, Sup
18		egor.zhdan, Sam, UnKnown
19		Nihal Saxena
20		Kevin Robatel

21		Aaron Christiansen, Adam Arold, Brad Larson, Héctor, Jayson Minard, Konrad Jamrozik, madhead, mayojava, razzledazzle, Sapan Zaveri, Serge Nikitin, yole
22		Divya, glee8e
23		David Soroko, Kirill Rakhman, SerCe
24		Brad Larson, jereksel, Sapan Zaveri
25		Sam, Seaskyways
26		Slav
27		Divya, Jan Vladimir Mostert, Jayson Minard, Ritave, Robin
28		Espen, Travis
29		hotkey, Jayson Minard, KeksArmee
30		Abdullah, Alex Facciorusso, jpmcosta, Kirill Rakhman, Robin, Spidfire
31		Ascension
32		Jemo Mgebrishvili
33		Grigory Konushev, Spidfire
34		byxor, KeksArmee, piotrek1543, Slav
35		Sam
36		Brad Larson, Caelum, Héctor, Mood, piotrek1543, Sapan Zaveri
37		byxor, piotrek1543, Sam
38		Dávid Tímár, Jayson Minard, Kevin Robatel, Konrad Jamrozik, olivierlemasle, Parker Hoyes, razzledazzle