What is the difference between "const" and "val"?

Asked 8 years, 6 months ago Modified 5 months ago

Viewed 122k times

480

I have recently read about the `const` keyword, and I'm so confused! I can't find any difference between `const` and the `val` keyword, I mean we can use both of them to make an immutable variable, is there anything else that I'm missing?

constants kotlin

Share

Improve this question

Follow

edited Oct 16, 2019 at 4:38

chancyWu

14.4k • 11 • 66 • 82

asked Jun 2, 2016 at 15:20

Mathew Hany

14k • 5 • 20 • 16

4 kotlinlang.org/docs/reference/… – Michael Jun 2, 2016 at 15:34

9 Answers

Sorted by: Highest score (default)

640

`const`s are compile time constants. Meaning that their value has to be assigned during compile time, unlike `val`s, where it can be done at runtime.

This means that only a String or primitive can be assigned to a `const`, not the result of a function or class constructor invocation.

For example:

```
const val foo = complexFunctionCall()   //Not okay
val fooVal = complexFunctionCall()      //Okay

const val bar = "Hello world"           //Also okay
```

Share Improve this answer Follow

edited Dec 3, 2023 at 2:42
Scott
99 1 9

answered Jun 2, 2016 at 15:24
Luka Jacobowitz
23.4k 5 40 58

5 What about something like this: `const val foo = "Hello world"` and `val bar = "Hello world"`? Are they the same? – Mathew Hany Jun 2, 2016 at 15:32

5 @MathewHany, at least not in terms of bytecode, see: stackoverflow.com/questions/37482378/static-data-in-kotlin/… – hotkey Jun 2, 2016 at 15:34

6 I think `const` values will just be completely inlined during compilation. – Luka Jacobowitz Jun 2, 2016 at 15:35

187 This begs another question: Why does Kotlin require `const val` instead of just `const`? It seems to me the `val` keyword is totally superfluous in this context, since `const var` would be absurd on its face. – Eric Lloyd Jun 8, 2017 at 15:51

51 @EricLloyd With `const val`, `const` is a modifier on `val` rather than a keyword. Modifiers > keywords. More examples of this same design are, `annotation/enum/data class`, `private val`, `inline fun`, etc. – Aro Apr 2, 2018 at 19:36

59

Just to add to Luka's answer:

> **Compile-Time Constants**
>
> Properties the value of which is known at compile time can be marked as compile time constants using the const modifier. Such properties need to fulfill the following requirements:
>
> - Top-level or member of an object declaration or a companion object.
> - Initialized with a value of type String or a primitive type
> - No custom getter
>
> Such properties can be used in annotations.

Source: Official documentation

Share Improve this answer Follow

edited Jun 20, 2020 at 9:12

Community Bot

1 ● 1

answered Jun 3, 2016 at 14:48

EPadronU

1,813 ● 1 ● 16 ● 15

43

You can transform the Kotlin to Java. Then you can see **const** has one more **static** modifier than **val**. The simple code like this.

Kotlin:

```
const val str = "hello"
class SimplePerson(val name: String, var age: Int)
```

To Java(Portion):

```
@NotNull
public static final String str = "hello";

public final class SimplePerson {
   @NotNull
   private final String name;
   private int age;

   @NotNull
   public final String getName() {
      return this.name;
   }

   public final int getAge() {
      return this.age;
   }
```

```
    public final void setAge(int var1) {
        this.age = var1;
    }

    public SimplePerson(@NotNull String name, int age)
        Intrinsics.checkParameterIsNotNull(name, "name")
        super();
        this.name = name;
        this.age = age;
    }
}
```

Share Improve this answer Follow

answered Aug 12, 2018 at 8:37

Jin Wang
721 ● 7 ● 8

3 Could someone state in a comment why this answer was downvoted to oblivion? – James Jordan Taylor Nov 4, 2018 at 19:39

3 @JamesJordanTaylor I upvoted. But I assume it's because some people didn't read it carefully, and at a quick glance this answer seems to be talking about how to convert from java to kotlin, which would be off-topic. – WSBT Nov 7, 2018 at 23:19

2 What if `const` is removed, will it yield a different Java file? – DYS Feb 13, 2019 at 4:22

3 @DYS: I think it will remove the "static" and it will be just public final String str = "hello"; – Varun Ajay Gupta Mar 3, 2020 at 15:25

@DYS compare it to `SimplePerson`'s `private final String name;` which doesn't have the const and then is private as well, but that's because it's a member val instead of a top-level/package val and not because of the `const`. – nobled Jul 18, 2021 at 6:18

31

const kotlin to Java

```
const val Car_1 = "BUGATTI" // final static String Car
```

val kotlin to Java

```
val Car_1 = "BUGATTI"   // final String Car_1 = "BUGAT
```

In simple Language

1. The value of the const variable is known at compile time.
2. The value of val is used to define constants at run time.

Example 1-

```
const val Car_1 = "BUGATTI" ✔
val Car_2 = getCar() ✔
const val Car_3 = getCar() ❌

//Because the function will not get executed at the co
through error

fun getCar(): String {
    return "BUGATTI"
}
```

This is because getCar() is evaluated at run time and assigns the value to Car.

Additionally -

1. **val** is read-only means immutable that is known at run-time
2. **var** is mutable that is known at run-time
3. **const** are immutable and variables that are known at compile-time

Share Improve this answer Follow

answered Feb 25, 2020 at 20:02

Shivam Tripathi
648 ● 8 ● 7

24

Both `val` and `const` are immutable.

`const` is used to declare compile-time constants, whereas `val` for run-time constants.

```
const val VENDOR_NAME = "Kifayat Pashteen"  // Assignm

val PICon = getIP()  // Assignment done at run-time
```

Share Improve this answer Follow

edited Feb 29, 2020 at 13:31

answered Feb 29, 2020 at 10:06

Kifayat Ullah
619 ● 1 ● 6 ● 14

Compile-time happens before run-time, right?
– whatwhatwhat Dec 3, 2020 at 22:10

2 @whatwhatwhat yes. The code is compiled before being sent for execution. The point of time when the code executes is what is essentially known as run-time execution.
– Arpan Sircar Mar 1, 2021 at 15:43

1 @whatwhatwhat Yes compile-time happen before runtime.
– androminor Jan 7, 2022 at 14:48

1 `val` is not necessarily immutable. – Tenfour04 Feb 28, 2022 at 19:43

Because I read a lot, that "val" means immutable: This is definitely not the case, just see this example:

12

```
class Test {
    var x: Int = 2
    val y
        get() = x
}

fun main(args: Array<String>) {
    val test = Test()
    println("test.y = ${test.y}") // prints 2
    test.x = 4
    println("test.y = ${test.y}") // prints 4
}
```

Sadly, true immutability you can currently only expect with const - but this only at compile time. At runtime you can't create true immutability.

val just means "readonly", you can't change this variable directly, only indirect like I have shown in the example above.

Share Improve this answer Follow

answered Dec 26, 2021 at 12:12

henry86

159 ● 1 ● 8

wow, remarkable! – Sheldon Wei Feb 9, 2022 at 3:31

2 it *is* immutable even in your example. you defined y as a *function* which returns whatever is in x. this function cannot be re-assinged to another function – Marko Bjelac Jan 20, 2023 at 9:45

You are just returning a value of another variable through overriden getter method! You are NOT MUTATING Y i.e not reassigning it! – Mohammed Junaid Jan 25, 2023 at 11:15

10

Let's learn this by an example.

```
object Constants {

    val NAME = "Amit"

}
```

Note: We are not using `const`.

And, we are accessing this `NAME` as below:

```
fun testValWithoutConst() {
    val name = Constants.NAME
}
```

Now, we need to decompile this code. For that, we will have to convert this Kotlin source file to a Java source file.

We will get the following output:

```
public final void testValWithoutConst() {
  String name = Constants.INSTANCE.getNAME();
}
```

The output is as expected.

The above example was without the `const` keyword. Now, let's use the `const` keyword.

For that, we will modify our object class `Constants` in Kotlin as below:

```
object Constants {

    const val NAME = "Amit"

}
```

Note: We are using `const`.

And, we are accessing this `NAME` as below:

```
fun testValWithConst() {
    val name = Constants.NAME
}
```

Now, when we decompile this code, we will get the following output:

```
public final void testValWithConst() {
  String name = "Amit";
}
```

Here, we can see that the variable `NAME` has been replaced by its value which is `Amit`.

As the value has been inlined, there will be **no overhead to access that variable at runtime**. And hence, it will lead to a better performance of the application.

This is the advantage of using `const` in Kotlin.

Reference from my blog: [Advantage of using const in Kotlin]

Share Improve this answer Follow

edited Jul 7 at 4:27

answered Feb 6, 2023 at 5:48

Amit Shekhar
3,195 ● 2 ● 18 ● 18

this answer is a bit long, but it actually gives the real reason for using `const`, which is that the value will be inlined (whereas the fact that it must be a compile-time constant, while true, is only a *consequence* of that benefit)
– Andrew Spencer Aug 14 at 11:51

7

val

Kotlin `val` keyword is for **read-only** properties in comparison with Kotlin `var` keyword. The other name for `read-only` property is `immutable`.

Kotlin code:

```
val variation: Long = 100L
```

Java equivalent looks like this:

```
final Long variation = 100L;
```

const val

We use `const` keyword for immutable properties too. `const` is used for properties that are known at compile-time. That's the difference. Take into consideration that `const` property must be declared **`globally`**.

Kotlin code (in playground):

```
const val WEBSITE_NAME: String = "Google"

fun main() {
    println(WEBSITE_NAME)
}
```

Java code (in playground):

```
class Playground {

    final static String WEBSITE_NAME = "Google";

    public static void main(String[ ] args) {
        System.out.println(WEBSITE_NAME);
    }
}
```

Share Improve this answer Follow

edited Nov 21, 2020 at 12:28

answered Nov 21, 2020 at 12:21

Andy Jazz
57.7k • 18 • 160 • 252

Read-only is not the same thing as immutable so the second sentence of this answer is false. You can have a read-only `val` that produces different results on multiple calls through a custom getter or because it's a delegated property, or because it's open and has a setter in a subclass – Tenfour04 Feb 28, 2022 at 19:42

"Kotlin val keyword is for read-only properties" if so then why do you write to it in your example? – Marian Paździoch Apr 26, 2022 at 13:16

@Tenfour04 read-only === immutable, see my comment: stackoverflow.com/questions/37595936/… – Marko Bjelac Jan 20, 2023 at 9:47

@MarkoBjelec Counterexamples to what you're saying: pl.kotl.in/DZ-c6drq0 – Tenfour04 Jan 20, 2023 at 14:03

7

For those who are looking which is more appropriate or efficient between `val` and `const`:

For the String or any primitive data type `const val` is recommended to use instead of `val`. Because `val` will be known at runtime, so when your app is running then it will process all the values. On other hand `const val` will do this earlier at compile time. So performance wise `const val` will give better result.

Share Improve this answer Follow

edited Aug 15, 2022 at 5:48

answered May 19, 2021 at 12:50

Rahul Sharma

6,169 ● 5 ● 38 ● 47