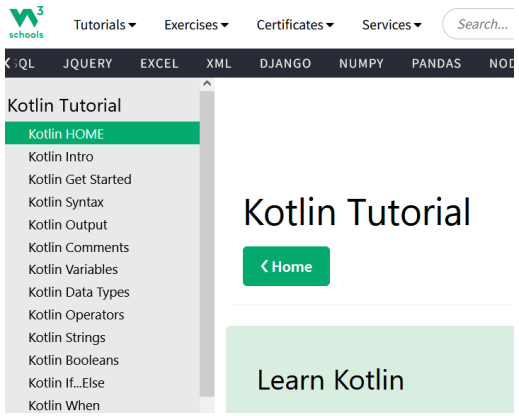
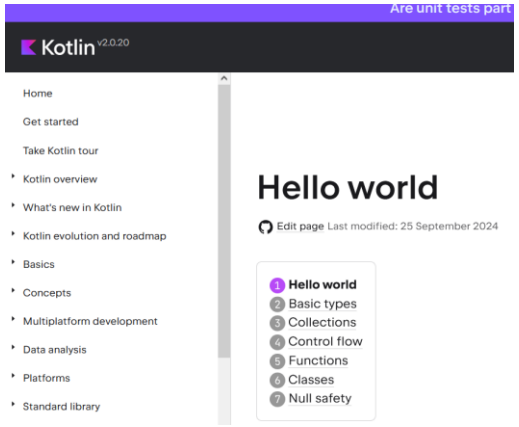
kotlin

tutorial version 1.0

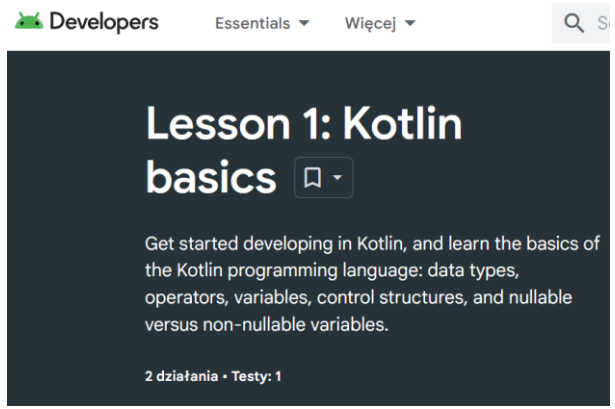
Kotlin tutorial



https://www.w3schools.com/kotlin/index.php

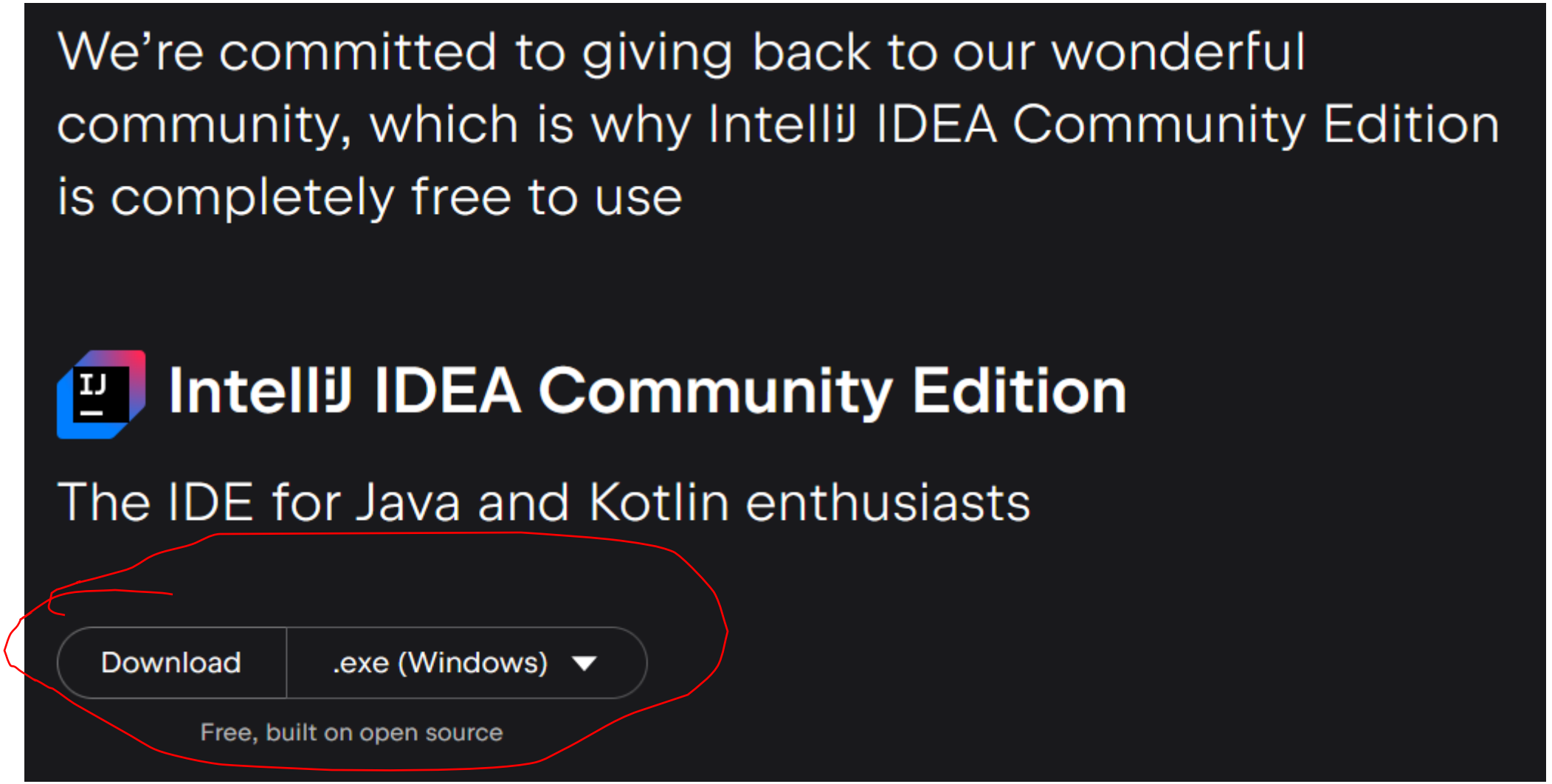


https://kotlinlang.org/docs/kotlin-tour-hello-world.html



https://developer.android.com/courses/pathways/android-development-with-kotlin-1#codelab-https://developer.android.com/codelabs/android-development-kotlin-1.1

Kotlin IDE

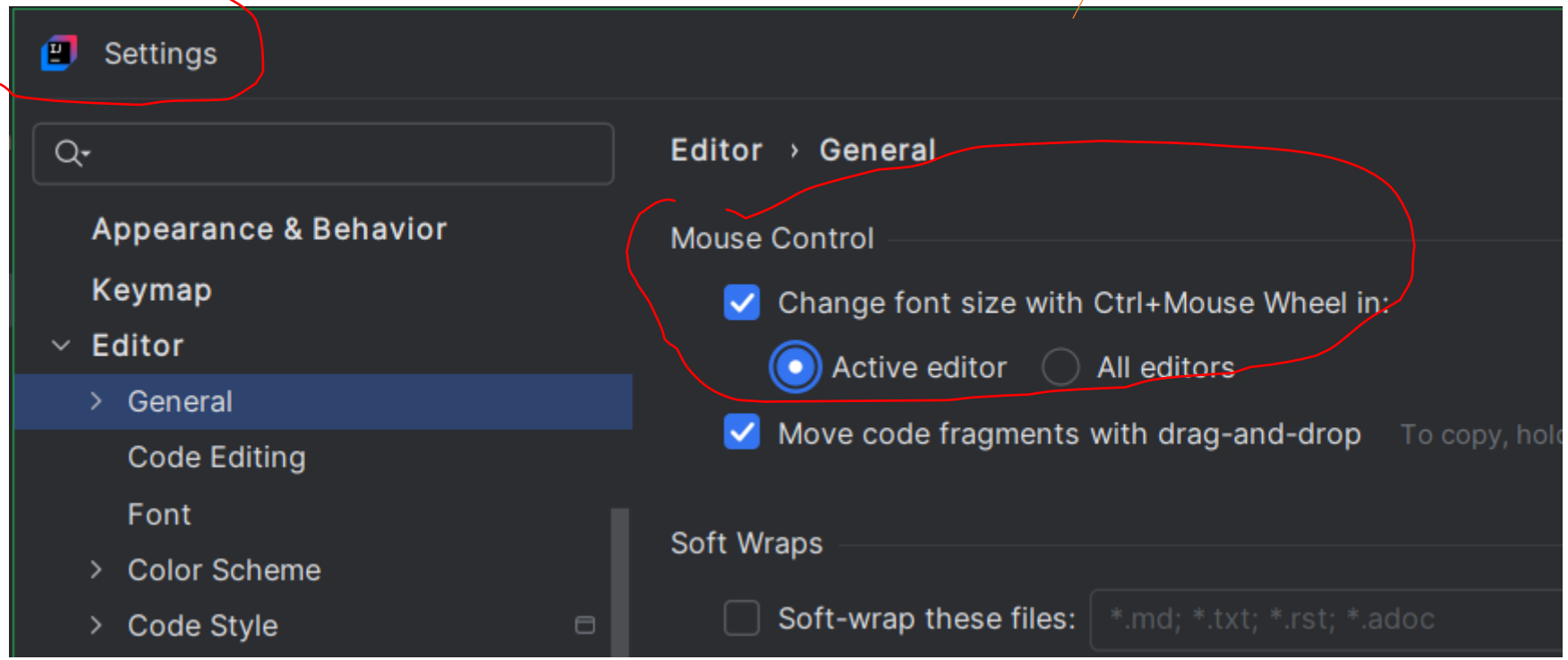


https://www.jetbrains.com/idea/download/?section=windows

Intellij IDEA Community Edition

ctrl + alt + s

zmiana wielkości czcionek
za pomocą kółka myszki



String Templates

In Kotlin, string templates allow you to create strings that include variables and expressions, making string manipulation more convenient and readable. You can insert variables or expressions directly into a string by using the `$` symbol for variables and `${}` for expressions that need evaluation.

Basic Usage

1. **Variable Substitution**: You can integrate a variable directly into a string using the `$` sign.

```
val name = "Alice"
val greeting = "Hello, $name!"
println(greeting)  // Output: Hello, Alice!
```

2. **Expressions**: If you want to perform an expression or use a complex calculation, use `${}`.

```
val x = 5
val y = 10
val result = "The sum of $x and $y is ${x + y}."
println(result)  // Output: The sum of 5 and 10 is 15.
```

https://deepai.org/chat

Kotlin ?

In Kotlin, the question mark (`?`) is used primarily to denote nullable types. Here's a brief overview of how it works:

1. Nullable Types

In Kotlin, by default, all types are non-nullable. This means you cannot assign `null` to a variable of a non-nullable type. If you want to allow a variable to hold `null`, you must declare it as nullable by adding a question mark after the type.

Example:

```
var nonNullableString: String = "Hello"
// nonNullableString = null // This will cause a compilation error

var nullableString: String? = "Hello"
nullableString = null // This is allowed
```

Kotlin ?.

2. Safe Calls

When working with nullable types, Kotlin provides a safe call operator (`?.`). This operator allows you to call methods or access properties on an object while safely handling the possibility of `null`.

Example:

```
val length: Int? = nullableString?.length // If nullableString is null, length will also be null
```

Kotlin ?:

3. Elvis Operator

The Elvis operator (`?:`) is used to provide a default value in case the left-hand side expression is null.

Example:

```
val length: Int = nullableString?.length ?: 0 // If nullableString is null, length will be 0
```

Kotlin !!

4. Not-null Assertion

If you are sure that a nullable variable is not null at a certain point in your code, you can use the non-null assertion operator (`!!`). However, if it is null, this will throw a `NullPointerException`.

Example:

```
val length: Int = nullableString!!.length // Make sure nullableString is not null to
avoid exceptions
```

nullable variables in Kotlin

Use nullability in Kotlin

Polski

Zaloguj się

1. Before you begin
2. **Use nullable variables**
3. Handle nullable variables
4. Conclusion

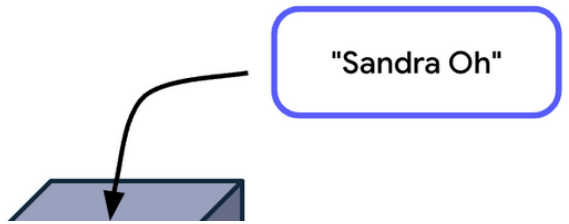
Zgłoś pomyłkę

2. Use nullable variables

What is `null`?

In Unit 1, you learned that when you declare a variable, you need to assign it a value immediately. For example, when you declare a `favoriteActor` variable, you may assign it a `"Sandra Oh"` string value immediately.

```
val favoriteActor = "Sandra Oh"
```

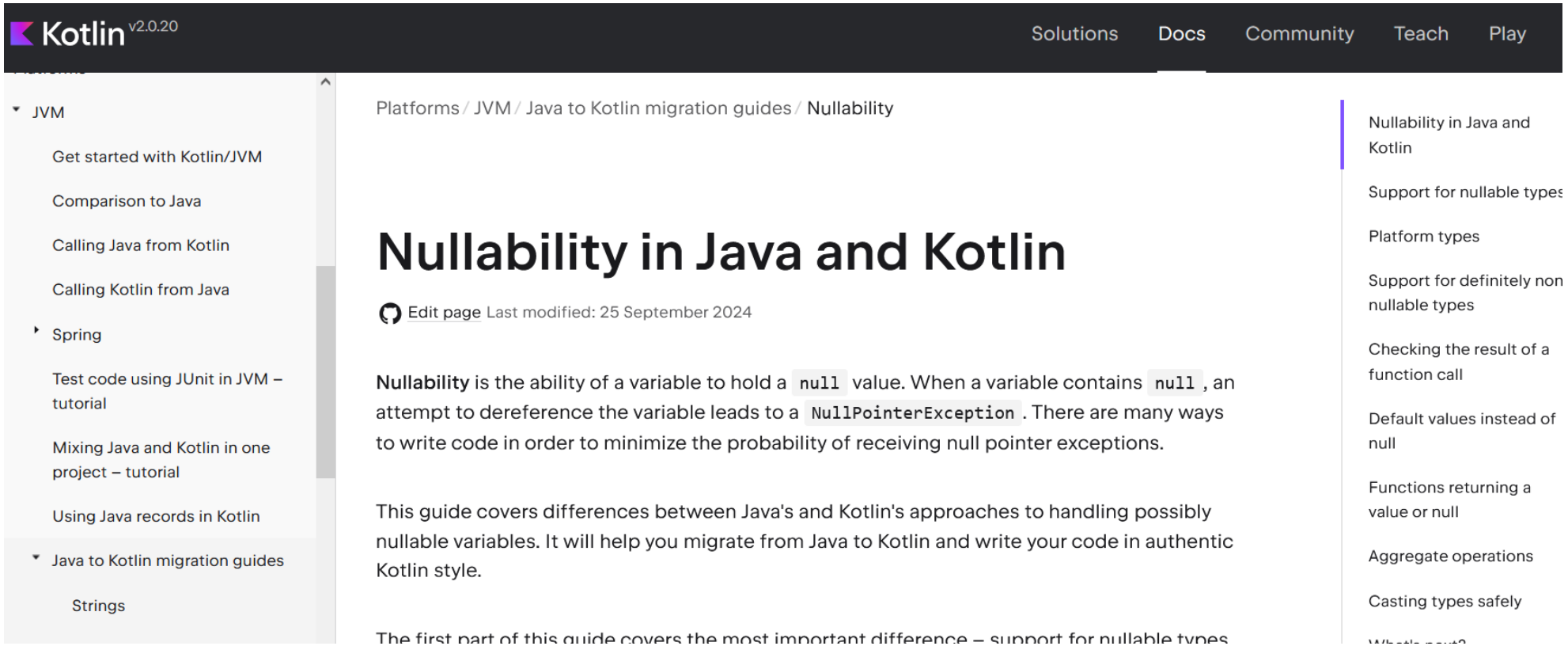


Wstecz

Dalej

https://developer.android.com/codelabs/basic-android-kotlin-compose-nullability#1

nullable variables in Kotlin



https://kotlinlang.org/docs/java-to-kotlin-nullability-guide.html

Kotlin Cheat Sheet

CHEAT SHEET

BASICS

"Hello, World" program

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

Declaring function

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

Single-expression function

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

Declaring variables

```
val name = "Marcin"  // Can't be changed
var age = 5          // Can be changed
```

CLASSES

Primary constructor

val declares a read-only property, var a mutable one

```
class Person(val name: String, var age: Int)
// name is read-only, age is mutable
```

Inheritance

```
open class Person(val name: String) {
    open fun hello() = "Hello, I am $name"
    // Final by default so we need open
}
class PolishPerson(name: String) : Person(name) {
    override fun hello() = "Dzień dobry, jestem $name"
}
```

Properties with assessors

```
class Person(var name: String, var surname: String) {
```

https://kt.academy/Kotlin_Cheat_Sheet.pdf