

The background of the slide is a grayscale image of a circuit board. It features a complex network of black lines representing traces, with several large black circular pads or vias. The background has a subtle, repeating pattern of concentric circles, giving it a technical, high-tech appearance.

Kotlin.. Some interesting bits! (For a Java Developer)

Extension Functions, Destructuring declarations and Scope Functions

Everything that I'm about
to tell you has already
been explained by people
much smarter than me.
Read the Kotlin Docs!

<https://kotlinlang.org/docs/home.html>

Thank you for
coming to my
TED Talk

Who am I?



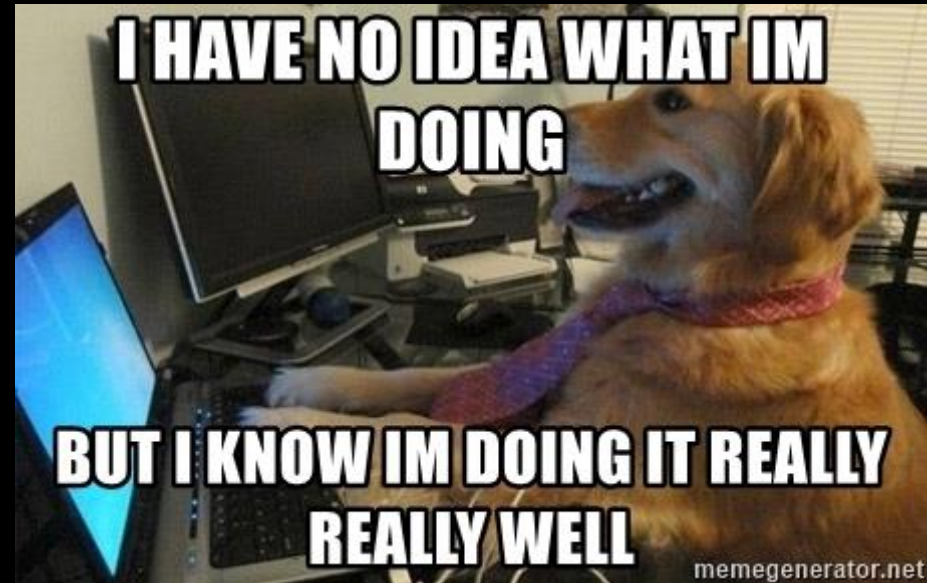
- Some random person Ben Dale once met



- I'm now working for myself, so:
 - No disclaimers necessary
 - I can describe myself as a Software Engineer (no fancy titles necessary)

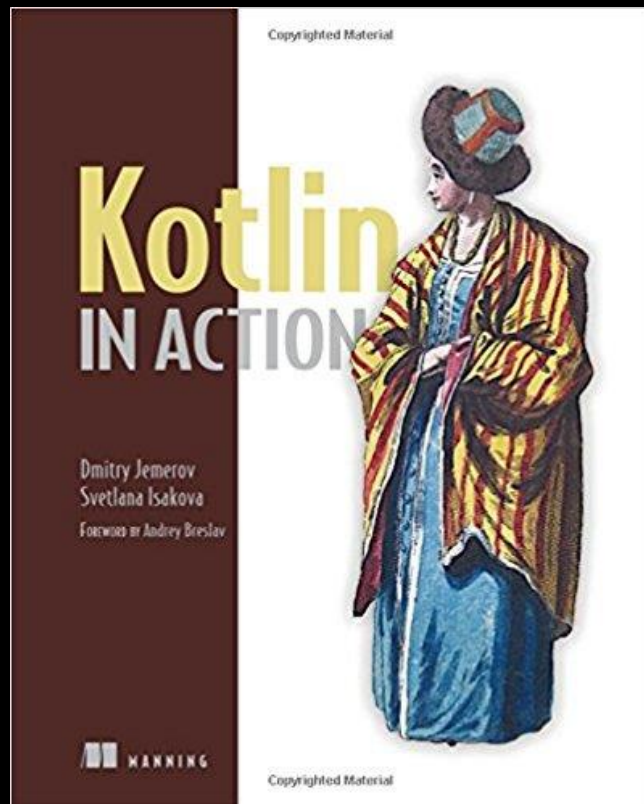
I'm not a Kotlin Developer

- I've passed a Kotlin technical interview
- And then another
- And another
- And I'm currently working with Kotlin
- But I'm not a Kotlin Developer

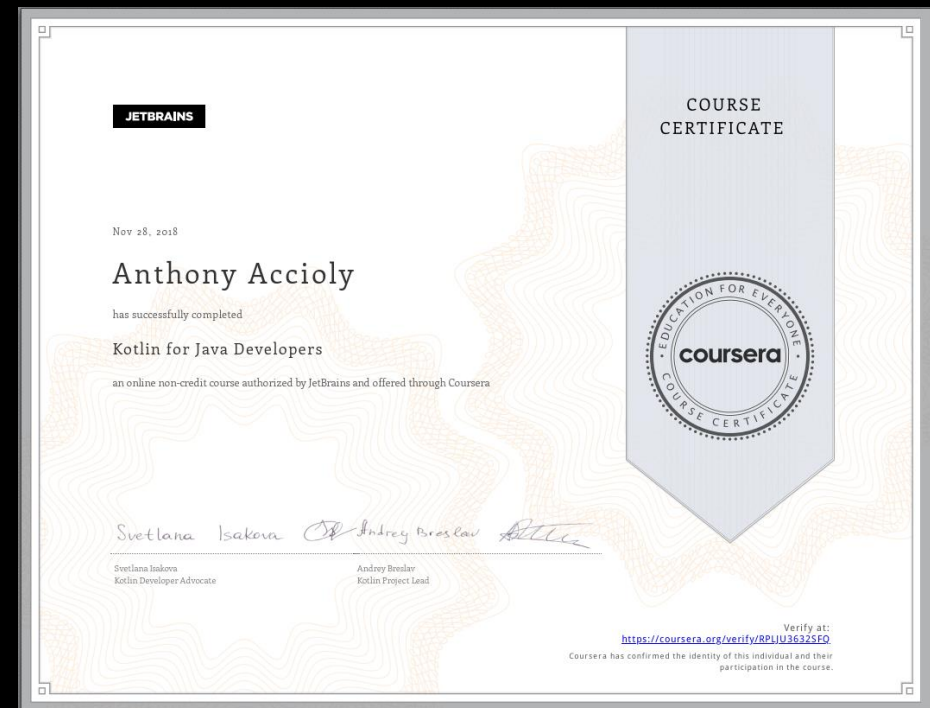


My path as a non-Kotlin developer

- I've read [the book](#)



- I took an online course by JetBrains:



<https://www.coursera.org/learn/kotlin-for-java-developers>

But I'm kind of a Java Developer

- I also wrote a lot of Scala code
- And, as everyone knows:

(Long unnecessary explanation about obviously ironic statement)

- Andrey's slides:
<https://www.slideshare.net/abreslav/2022-may-shoulders-of-giants-amsterdam-kotlin-dev-daypdf>
- Kotlin Channel:
<https://www.youtube.com/c/Kotlin>



#kotlin is just...



Andrey Breslav



Okay... Kotlin

Or actually: Java



How would you implement
“racecar”.isPalindrome()?

You can't! Well...

```
public final class StringUtils {  
  
    private StringUtils() {  
        throw new UnsupportedOperationException(  
            "This is a utility class and cannot be instantiated");  
    }  
    public static String reverse(String s) {  
        return new StringBuilder(s).reverse().toString();  
    }  
    public static boolean isPalindrome(String s) {  
        return s.equals(reverse(s));  
    }  
}
```

Or use [Project Lombok's @UtilityClass](#)



Trouble?

- String is (and should remain) immutable
- Utility classes all over the place
- Code without context
- Apache Commons Lang [StringUtils](#) class has ~10k lines



Extension functions to
the rescue

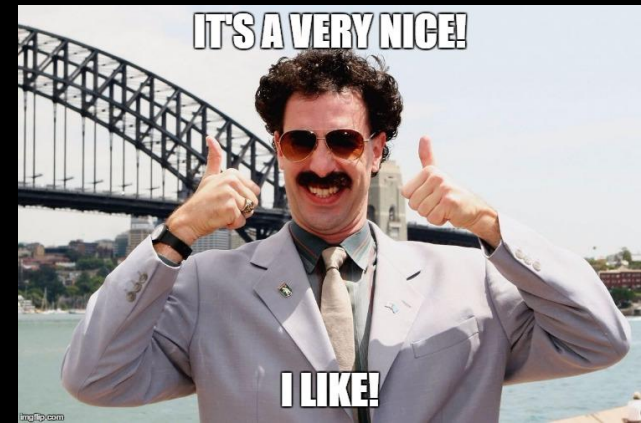
Kotlin functions!

```
fun isPalindrome(string: String): Boolean {  
    return string == string.reversed()  
}
```

```
fun isPalindrome(string: String): Boolean =  
    string == string.reversed()
```

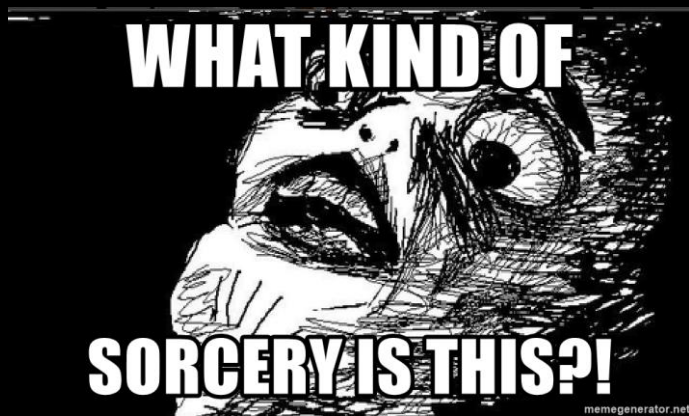
```
fun isPalindrome(string: String) =  
    string == string.reversed()
```

- Top-level function
- Single-expression function
- Return type is optional



But we can still do better!

```
fun String.isPalindrome() =  
    this == this.reversed()
```



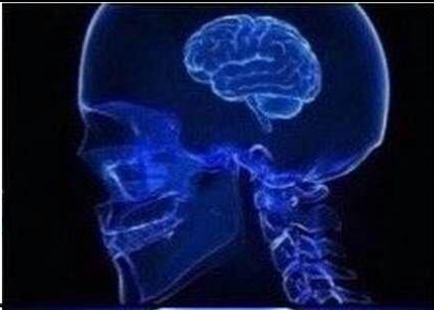
```
@file:JvmName("KotlinStringUtils")
```

- Extension functions!
- Call direct on a string
- Compiles to a static method
- FileStrings.kt → Default class StringsKt
- Can be customized
- Can be instantiated from Java ([reference](#)). Well, nothing is perfect!

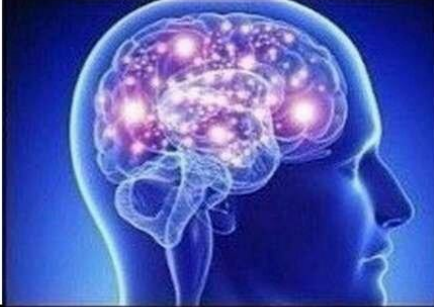


How would you implement
`String.firstAndLast()`?


```
char[2]  
List.of(first, last)  
Collections.singletonMap  
javafx.util.Pair
```



```
Map.Entry<Character,  
Character>
```



Custom class?
Custom class + Lombok?
External Library (javatuples, vavr)



Records

```
char[2]
```



Records are Okay...ish!

```
record FirstAndLast(char first, char last) {  
}
```

```
public static FirstAndLast firstAndLast(String s) {  
    return new FirstAndLast(  
        s.charAt(0),  
        s.charAt(s.length() - 1));  
}
```



Kotlin

Destructuring
declarations

Kotlin can do something similar

```
data class FirstAndLast(val first: Char, val last: Char)

fun String.firstAndLast(): FirstAndLast =
    FirstAndLast(first(), last())
```

But with better syntax

```
fun `returns first and last characters at once`()
{
    val input = "behind the scenes"
    val (first, last) = input.firstAndLast()
    first shouldBe 'b'
    last shouldBe 's'
}
```



Also...

- Backticks for method names with spaces
- Infix notation for matchers
- [Kotest](#)
- Great way to introduce Kotlin

Destructuring declarations

- Behind the scenes:

```
operator fun component1() = first
operator fun component2() = last
```
- You can implement your own
- Or use Pair

```
fun String.firstAndLast(): Pair<Char, Char> =
    first() to last()
```
- Kotlin used to have Tuples but...
Discontinued - RIP!
- Use underscore if you don't care:

```
val (first, _) = input.firstAndLast()
```
- Very useful with collections and lambdas

```
fun initials(firstName: String, lastName: String) =
    "${firstName.first()}.${lastName.first()}"

val names = mapOfNames.map { (fn, ln) ->
    initials(fn, ln)
}
```



Scope functions

Speaking of lambdas... Scope Functions

- Run a block of code against an object and lets you chain operations
- Idiomatic, but prone to abuse

Function	Object reference	Return value	Is extension function
<code>let</code>	<code>it</code>	Lambda result	Yes
<code>run</code>	<code>this</code>	Lambda result	Yes
<code>run</code>	-	Lambda result	No: called without the context object
<code>with</code>	<code>this</code>	Lambda result	No: takes the context object as an argument.
<code>apply</code>	<code>this</code>	Context object	Yes
<code>also</code>	<code>it</code>	Context object	Yes


```
val jackson = Person("Michael", "Jackson", 50)

val result = jackson.copy(lastName = "Jordan", age = 59)
    .let { jordan -> // Temporary variable for Michael Jordan
        jordan.firstName shouldBe "Michael"
        jordan.lastName shouldBe "Jordan"
        jordan.age shouldBe 59 // Michael Jordan is 59 as of 16/06/2022
        jackson // Forwards Michael Jackson, despite operating on Michael Jordan
    }.apply {
        age += 13
    }.also { // Useful for logging and additional effects
        it.age shouldBe 63 // Michael Jackson would be 63 as of 16/06/2022
    }.run {
        age += 1
        val mjInitials = initials(firstName, lastName)
        "By 29 August 2022 $mjInitials would be $age years old"
    }

with(result) {
    shouldBeSingleLine()
    shouldContain("64")
    shouldBe("By 29 August 2022 M.J. would be 64 years old")
}
```

Interesting usages and other goodies



- `?.let` to skip through null
- `run` without a receiver to give a scope to variables
- `also` for logging
- `takeIf` and `takeUnless` null based filtering
- use Kotlin equivalent of `try-with-resource`
- `Result Try` like type
- [Arrow](#) types



That's all Folks!

Questions?



<https://www.linkedin.com/in/aaccioly>



<https://stackoverflow.com/users/664577>



<https://a.accioly.7rtc.com>



[a.accioly \(at\) pm \(dot\) com](mailto:a.accioly(at)pm(dot)com)

Slides will be available at:

<https://github.com/aaccioly-demos/source-talks-on-kotlin-3> (Soon)



Kotlin... Some interesting bits!
(For a Java Developer) © 2022 by Anthony Accioly
is licensed under [CC BY-SA 4.0](https://creativecommons.org/licenses/by-sa/4.0/)