



# Kotlin Workshop

QUB Google Dev Student Club  
November 2022



build

teach

talk

instil.co



AMERICAN  
EXPRESS

SHOPKEEP<sup>®</sup>

IBM.<sup>®</sup>

 Liberty  
Mutual<sup>™</sup>

citi

Bell

intel<sup>®</sup>

ATLASSIAN



INSTITUTIONAL  
PARTNERS

Johnson  
Controls



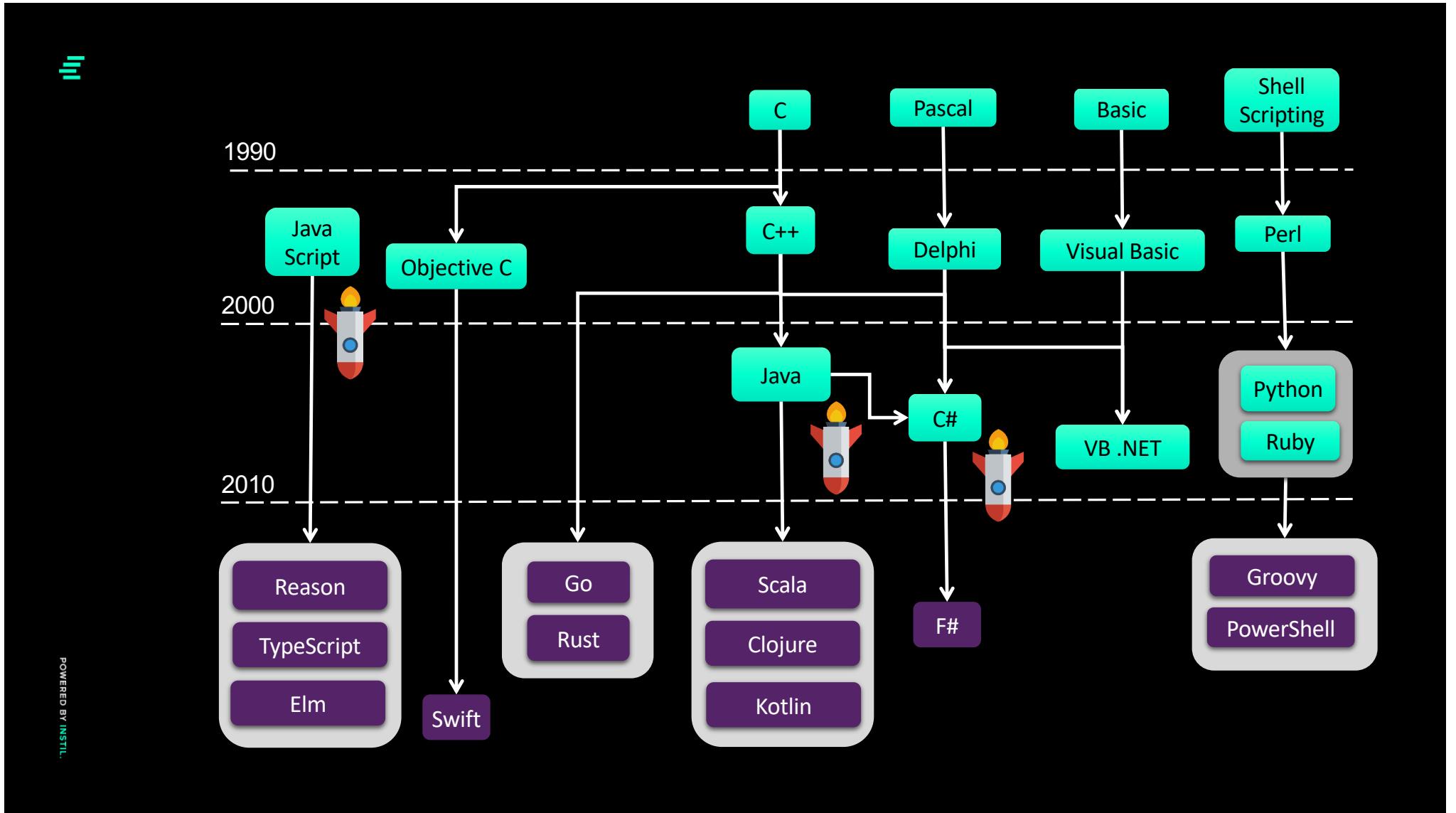
# Garth (@GarthGilmour)



≡

# Introduction

1





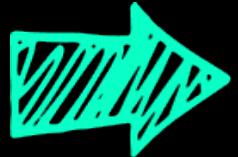
# Java and the JVM





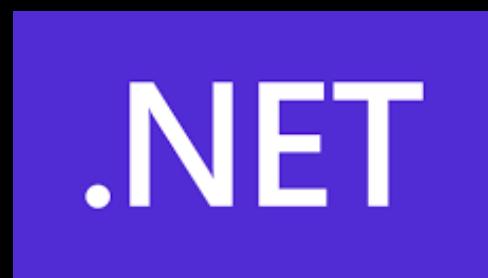


# New Kids on the (JVM) Block



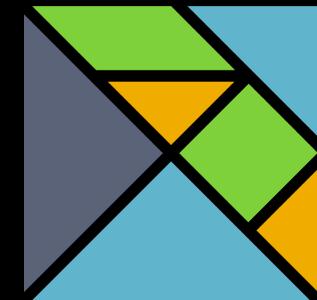
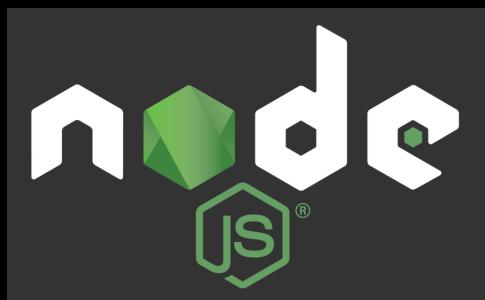


# New Kids on the (CLR) Block





# New Kids on the (JS) Block



TypeScript



# New Kids on the (Mac) Block

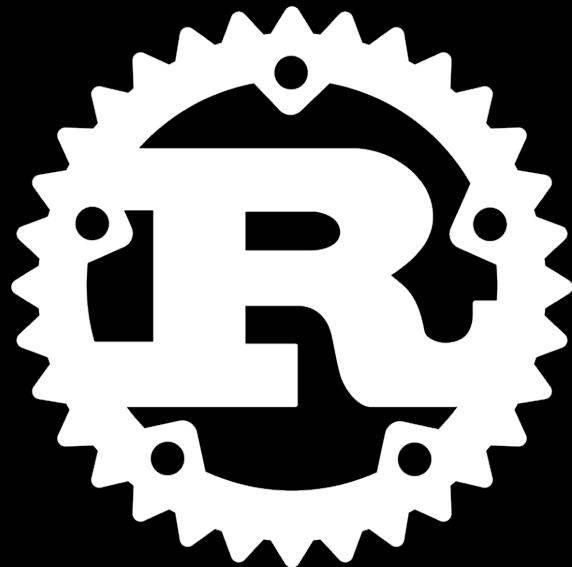


Swift



New Kids on the CPU

=GO



**DON'T  
PANIC**



# There Are Only So Many Paradigms

```
while(orders.hasNext()) {  
    Order o = orders.next();  
    if(o.priority == URGENT) {  
        o.ship();  
    }  
}
```

Procedural

Object Oriented

```
f1(List<? extends Order> data) {  
    Optional<Client> opt = f2();  
    for (Order o : data) {  
        o.send(opt.getorElse(me));  
    }  
}
```

Generic

Functional

```
class Order {  
    // ...  
}  
Order o1 = new Order();  
Order o2 = new Order();
```

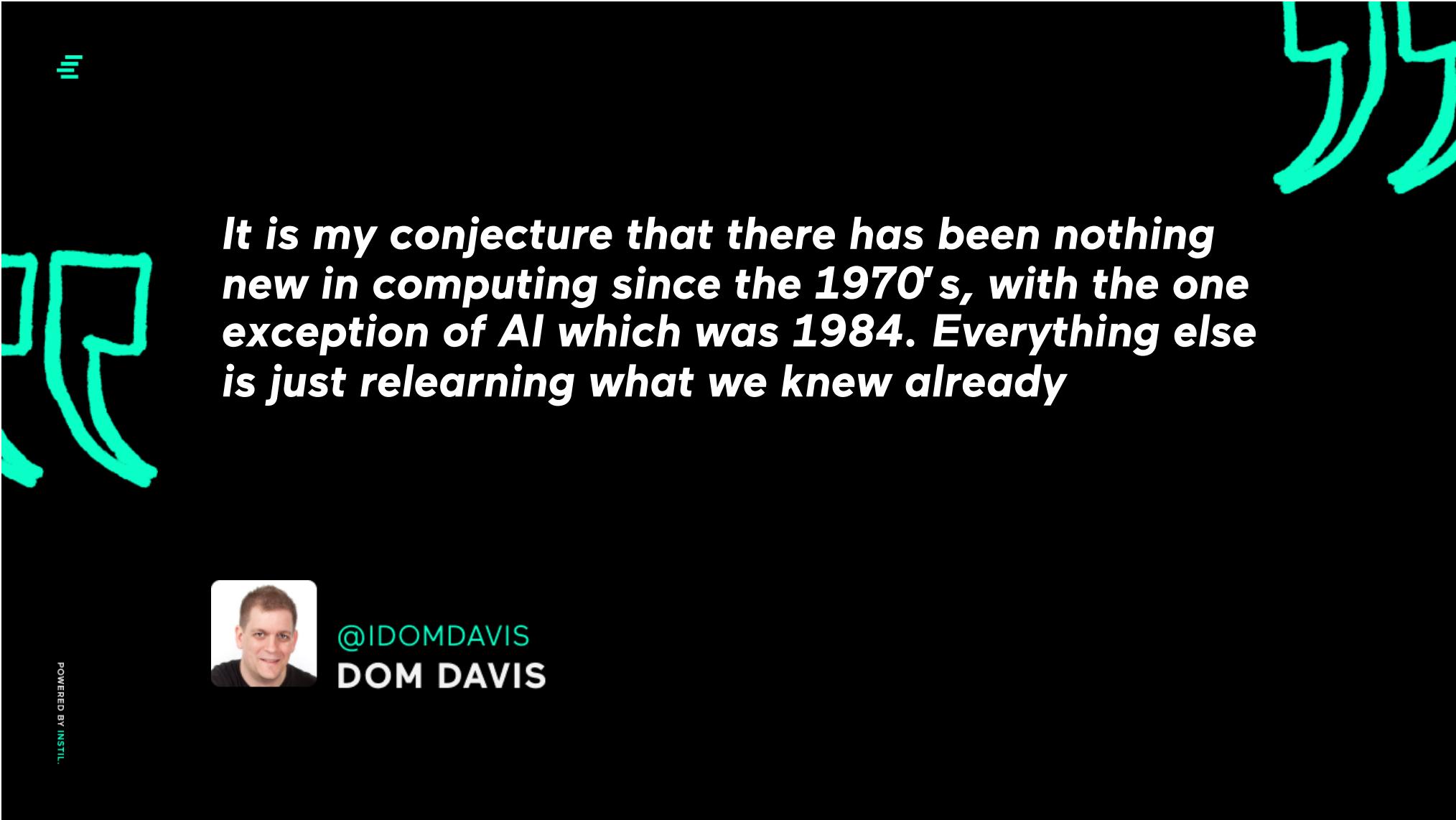
```
Orders  
.filter(o -> o.value > 500)  
.map(o -> o.customer)  
.each(c -> print(c.name));
```



*It is my conjecture that there has been nothing new in computing since the 1970's, with the one exception of AI*



@IDOMDAVIS  
**DOM DAVIS**



***It is my conjecture that there has been nothing new in computing since the 1970's, with the one exception of AI which was 1984. Everything else is just relearning what we knew already***



@IDOMDAVIS  
**DOM DAVIS**



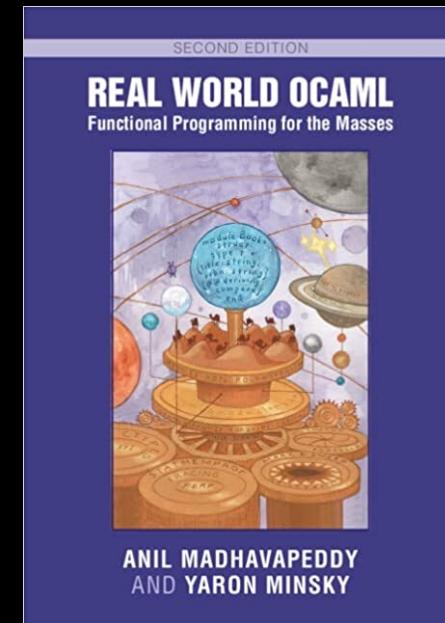
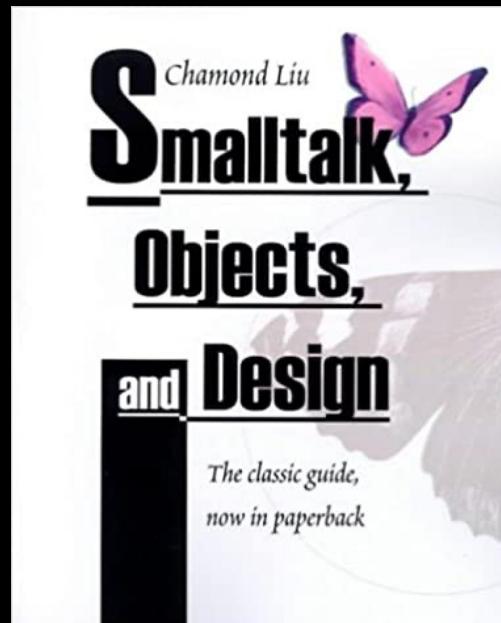
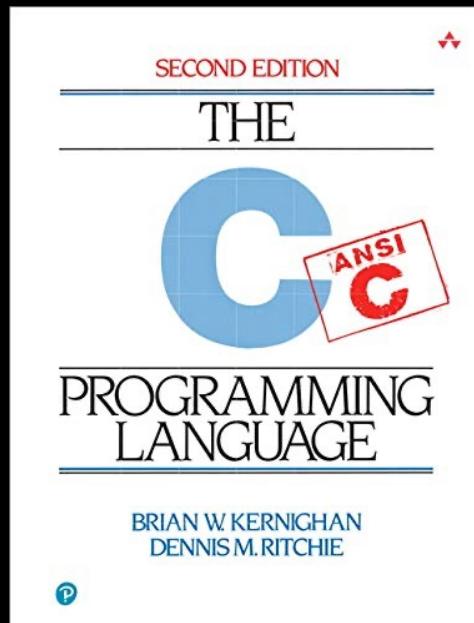
# There Are Only So Many Paradigms take the 'Bruce Wayne approach'

- **Focus on 'Root Styles' not 'Shiny Things'**
  - Procedural (Pull Based)
  - Declarative (Push Based)
  - Functional (Lambda Calculus)
  - Object Oriented (Behavior)
  - Generic (Type Based)



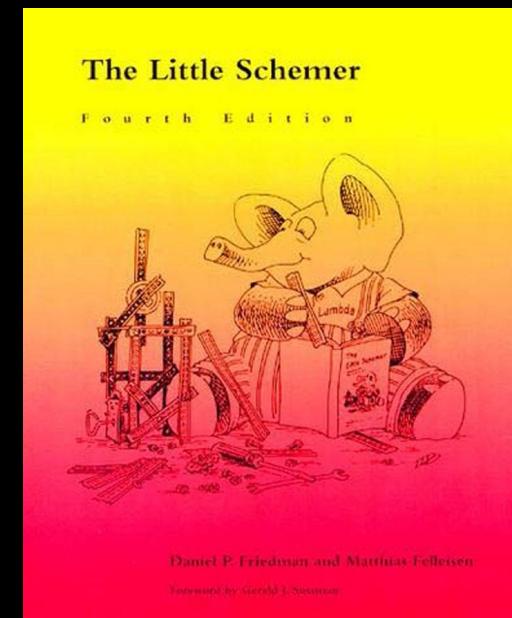
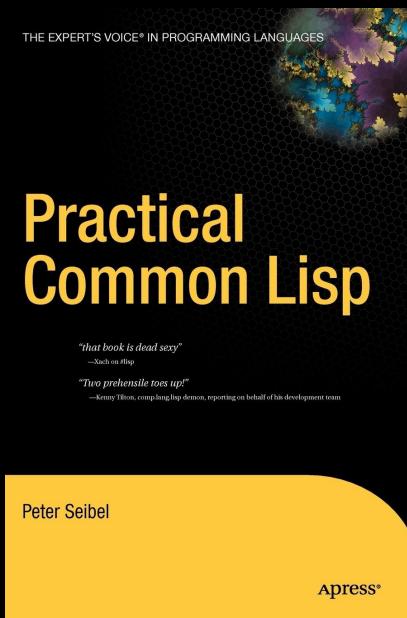
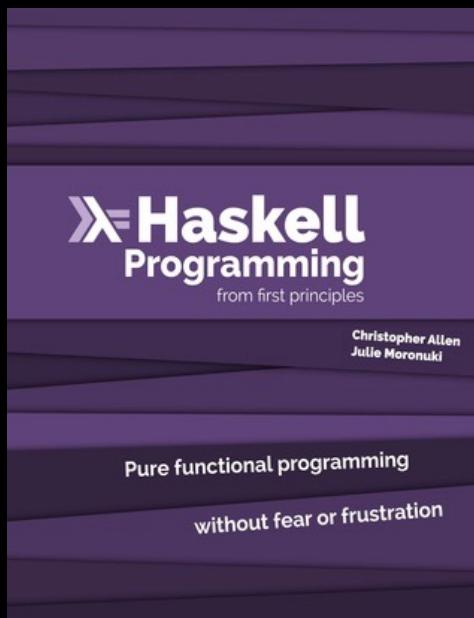


# The Root Languages





# The Root Languages



III

# Convergence

2



```
let m = new Mutant("Raven");
```

```
Mutant m = new Mutant("Raven");
```

```
val m = Mutant("Raven")
```



```
1 class Person(val name: String)
2
3 def display(p: Person) = println(s"Person called ${p.name}")
4
5 @main def demo() = {
6   val things = List("Dave", "Jane", "Pete", "Lucy")
7
8   things
9     .map(Person(_))
10    .foreach(display)
11 }
```

Person called Dave  
Person called Jane  
Person called Pete  
Person called Lucy



```
class Person(val name: String)

fun display(p: Person) = println("Person called ${p.name}")

fun main() {
    val things = listOf("Dave", "Jane", "Pete", "Lucy")

    things
        .map { Person(it) }
        .forEach(::display)
}
```

Person called Dave  
Person called Jane  
Person called Pete  
Person called Lucy





```
1  class Person{  
2    |  constructor(public name: String) {}  
3  }  
4  
5  const display = (p: Person) => console.log(`Person called ${p.name}`)  
6  
7  const things = ["Dave", "Jane", "Pete", "Lucy"]  
8  
9  things  
10 |  .map(str => new Person(str))  
11 |  .forEach(display)
```

Person called Dave  
Person called Jane  
Person called Pete  
Person called Lucy





```
public class Program {  
    record Person(String name) {}  
  
    static void display(Person p) {  
        final var msg : String = "Person called %s".formatted(p.name);  
        System.out.println(msg);  
    }  
  
    public static void main(String[] args) {  
        var things : List<String> = List.of("Dave", "Jane", "Pete", "Lucy");  
  
        things.stream() Stream<String>  
            .map(Person::new) Stream<Program.Person>  
            .forEach(Program::display);  
    }  
}
```



III

# Kotlin

3



# kotlin's selling points

2011

Null Safety

2016

Shrinking Codebase

2017

Elegant DSL's

2018

Coroutines

2019

Multiplatform Libraries

**Null Safety**

**String Templates**

**Default parameters**

**Extensions**

**Free Functions**

**Coroutines**

**Single Expression Functions**

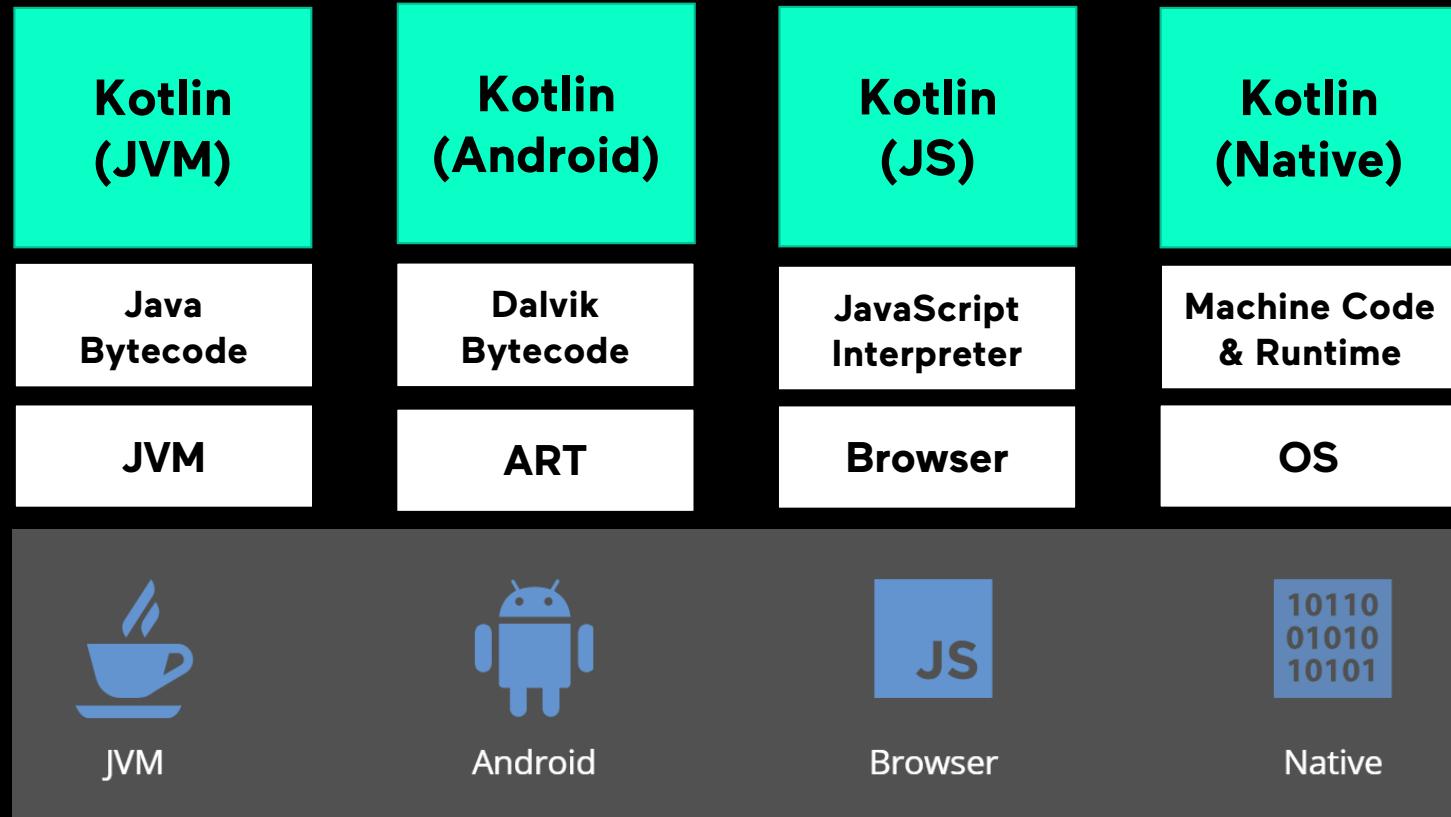
**Reified generics**

**Data classes and Properties**

**Type Inference**

**Smart Casts**

**Operator overloading**





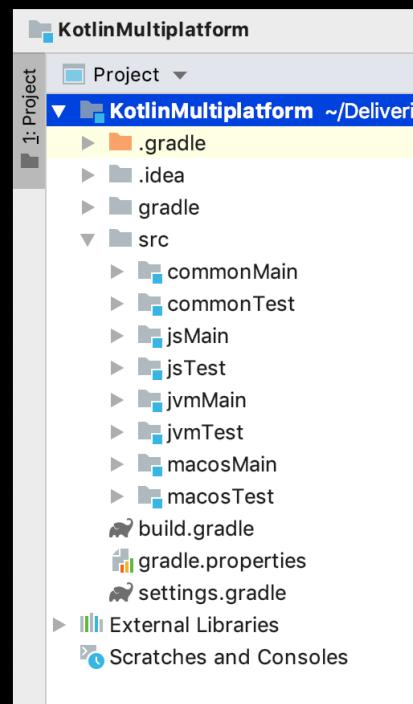
# multiplatform libraries

COMMON  
KOTLIN

KOTLIN JVM

KOTLIN JS

KOTLIN NATIVE

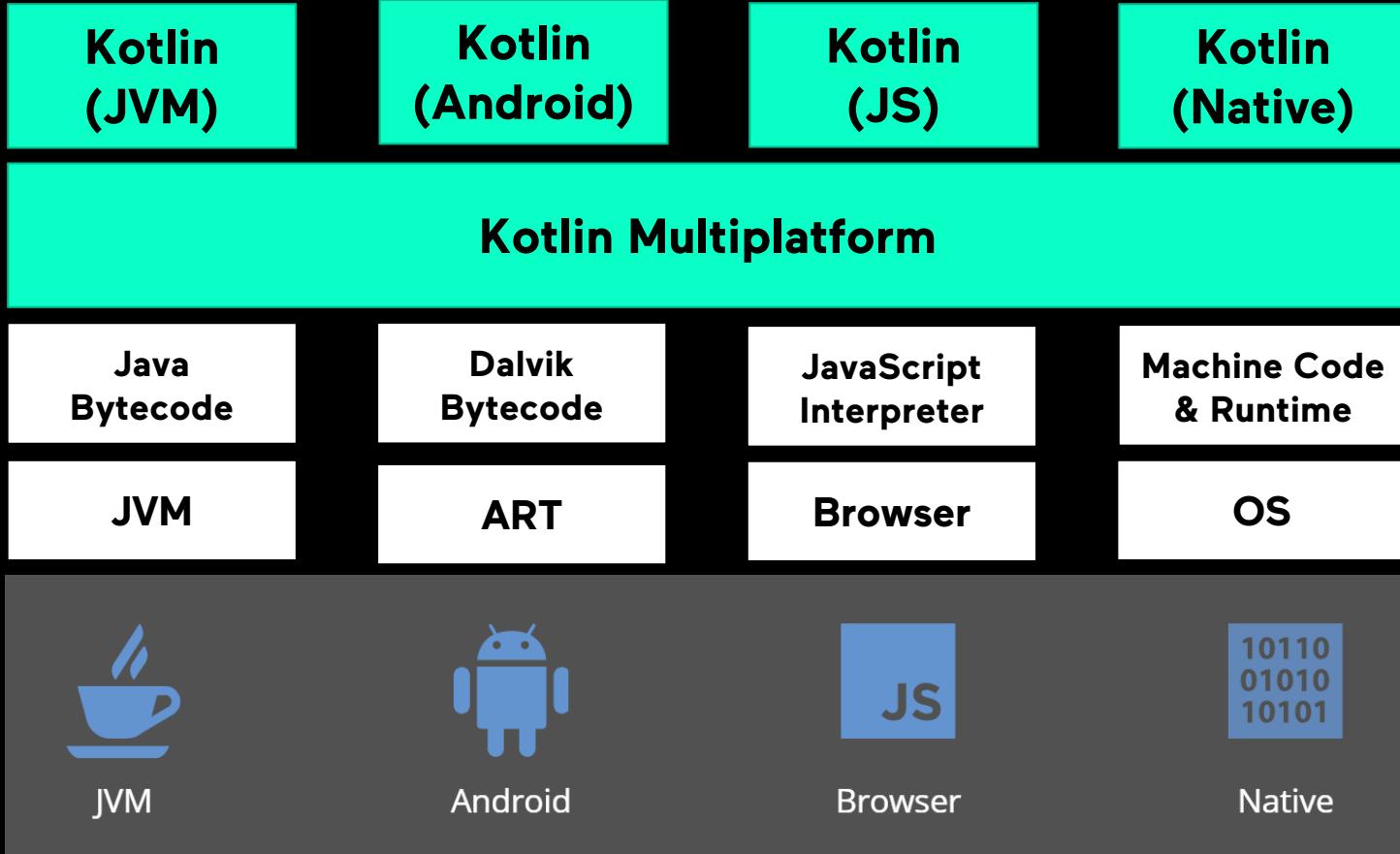


NATIVE  
ARTEFACT

JS BUNDLE

JAR





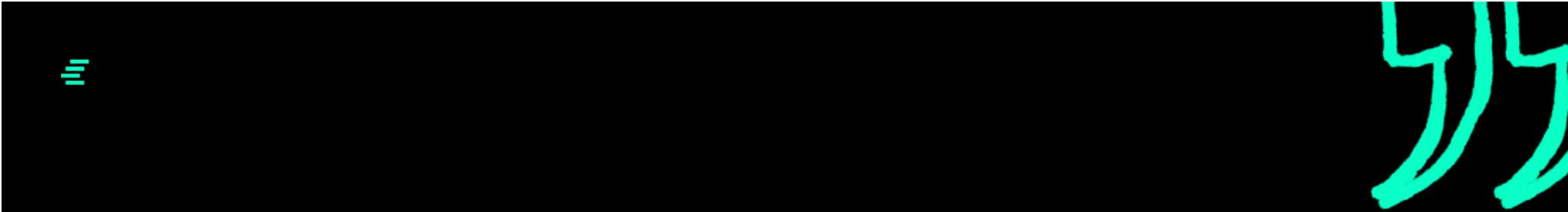


# why use kotlin on android?

## developer happiness

- **Codebase reduction by 40%**
  - (JetBrains figure)
- **Simplify the remaining code**
- **Plaster over Androids cracks**
- **Access FP features / patterns (Java 8+)**
- **Future proof your skill set**





***Today we're announcing another big step:  
Android development will become increasingly  
Kotlin-first. Many new Jetpack APIs and features  
will be offered first in Kotlin. If you're starting a  
new project, you should write it in Kotlin***



Google I/O 2019



# why use kotlin on android?

## it's better with kotlin

### Android Jetpack

Jetpack is a suite of libraries, tools, and guidance to help developers write high-quality apps easier. These components help you follow best practices, free you from writing boilerplate code, and simplify complex tasks, so you can focus on the code you care about.

Jetpack comprises the `androidx.*` package libraries, unbundled from the platform APIs. This means that it offers backward compatibility and is updated more frequently than the Android platform, making sure you always have access to the latest and greatest versions of the Jetpack components.

[GET STARTED](#)[WATCH THE INTRO VIDEO](#)

#### Accelerate development

Components are individually adoptable but built to work together while taking advantage of Kotlin language features that make you more productive.



#### Eliminate boilerplate code

Android Jetpack manages tedious activities like background tasks, navigation, and lifecycle management, so you can focus on what makes your app great.



#### Build high quality, robust apps

Built around modern design practices, Android Jetpack components enable fewer crashes and less memory leaked with backwards-compatibility baked in.



# why use kotlin and spring boot? it's better with kotlin

[◀ Back to index](#)

1. Spring WebFlux

2. WebClient

3. WebSockets

4. Testing

5. RSocket

5.1. Overview

5.2. RSocketRequester

**5.2.1. Client Requester**

Connection Setup

Strategies

Client Responders

Advanced

5.2.2. Server Requester

5.2.3. Requests

5.3. Annotated Responders

## 5.2.1. Client Requester

To obtain an `RSocketRequester` on the client side requires connecting to a server along with preparing and sending the initial RSocket `SETUP` frame. `RSocketRequester` provides a builder for that. Internally uses RSocket Java's `RSocketFactory`.

This is the most basic way to connect with default settings:

Java    Kotlin

KOTLIN

```
import org.springframework.messaging.rsocket.connectTcpAndAwait
import org.springframework.messaging.rsocket.connectWebSocketAndAwait

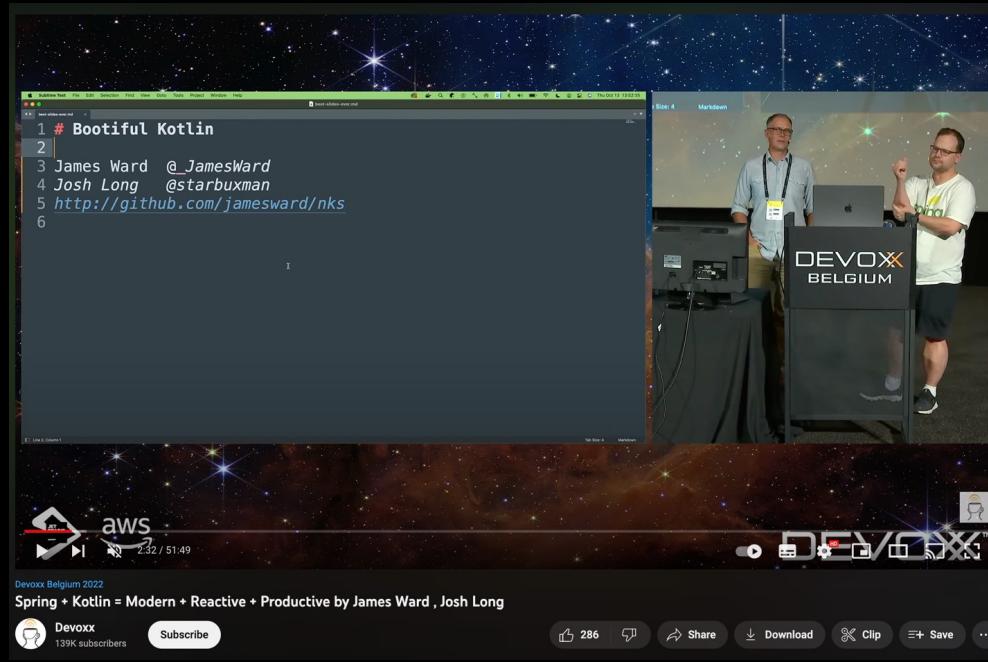
val requester = RSocketRequester.builder()
    .connectTcpAndAwait("localhost", 7000)

val requester = RSocketRequester.builder()
    .connectWebSocketAndAwait(URI.create("https://example.org:8080/rsocket"))
```

The above is deferred. To actually connect and use the requester:



# why use kotlin and spring boot? it's better with kotlin

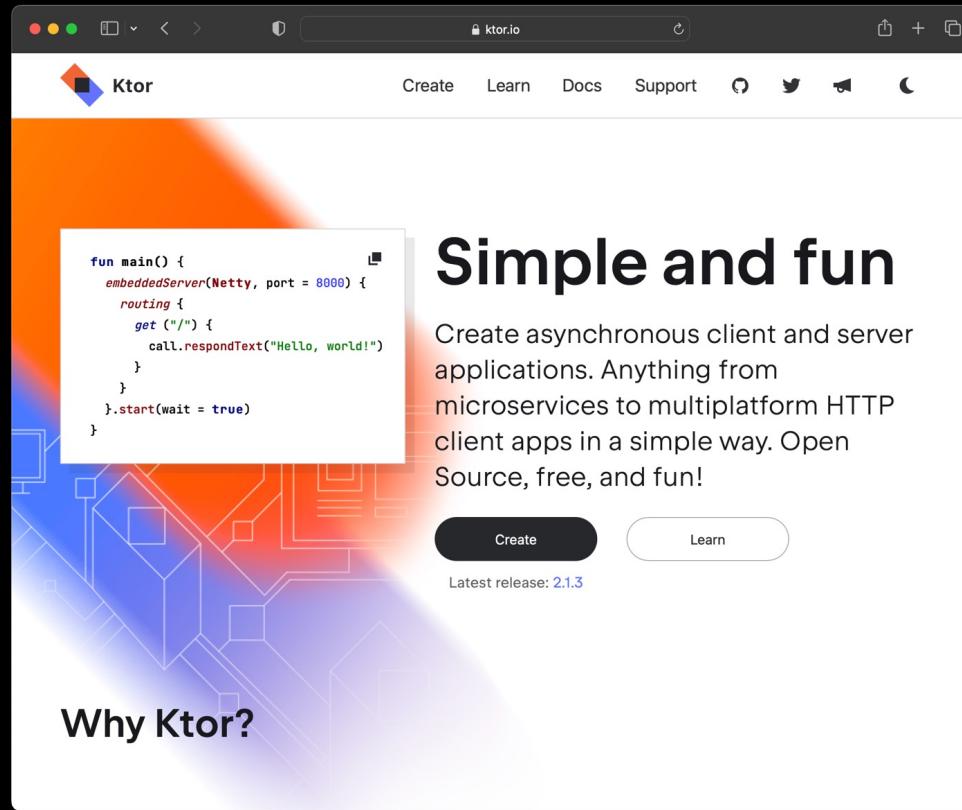




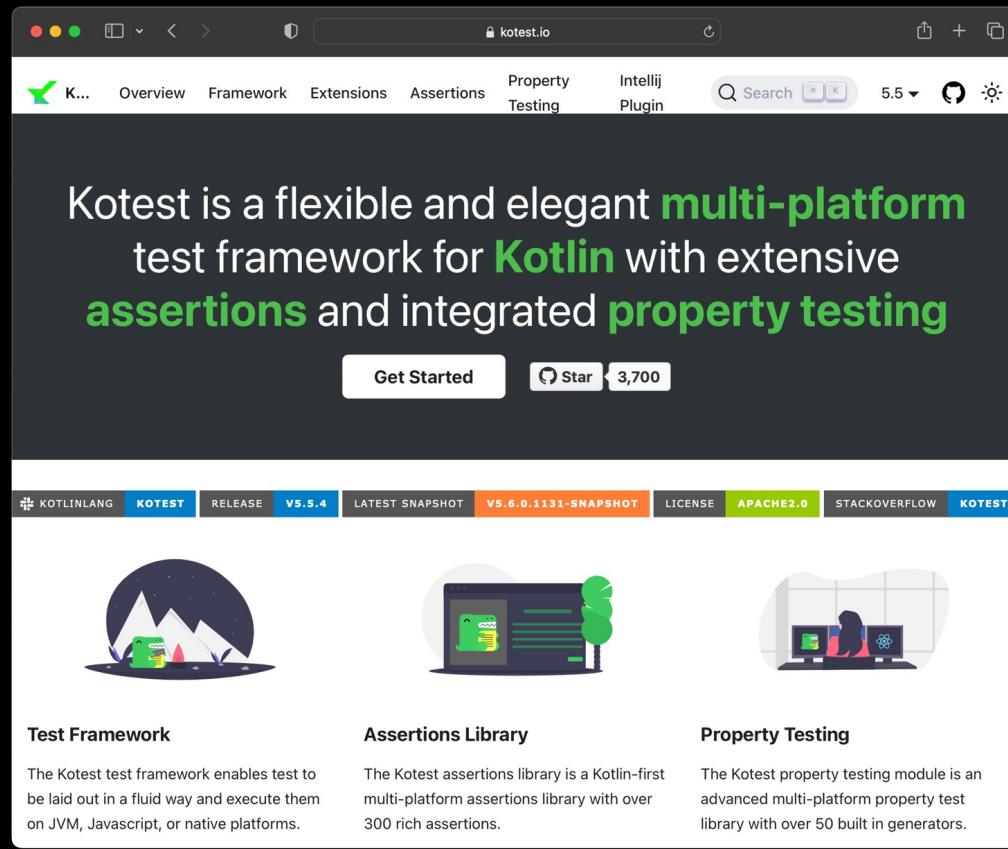
# why use kotlin and javascript? it's better with kotlin (maybe)

The screenshot shows a GitHub README.md page titled "Create React Kotlin App". At the top, there are status indicators: "TC build success" and "JB team". Below the title, the text reads: "Create React apps in Kotlin with no build configuration. Please note that this is an [early preview version](#). Make sure you have [JDK 8](#) installed before proceeding. Java 9 is not supported yet." It provides instructions for creating a new project: "Create a new project:  
`npx create-react-kotlin-app my-app`  
(`npx` comes with npm 5.2+ and higher. See [Installation](#) for older npm versions.)" and for running the project: "Run the project:  
`cd my-app/  
npm start`  
Then open <http://localhost:3000/> to see your app.  
When you're ready to deploy to production, create a minified bundle with `npm run build`.

## ≡ Kotlin Specific Libraries



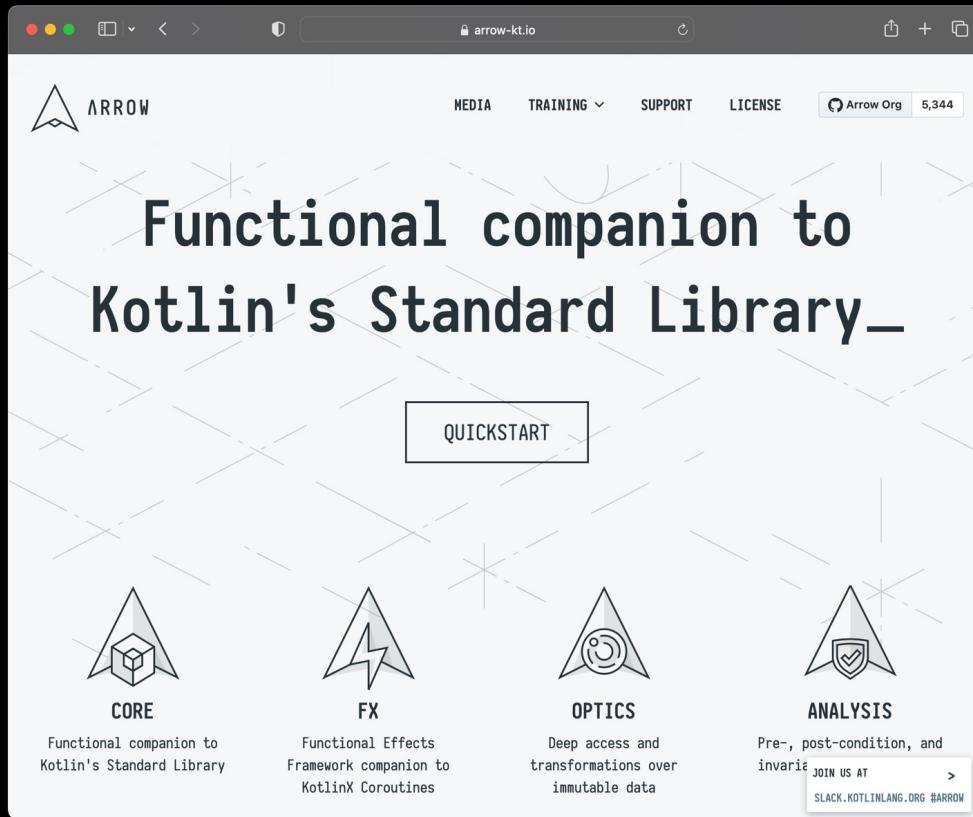
## ≡ Kotlin Specific Libraries



The screenshot shows the Kotest website homepage. At the top, there's a navigation bar with links for Overview, Framework, Extensions, Assertions, Property Testing, and IntelliJ Plugin. A search bar and a version indicator (5.5) are also present. The main heading reads: "Kotest is a flexible and elegant **multi-platform** test framework for **Kotlin** with extensive **assertions** and integrated **property testing**". Below this, there are three cards: "Test Framework" (with an icon of a mountain range), "Assertions Library" (with an icon of a laptop and tree), and "Property Testing" (with an icon of a person at a computer). A "Get Started" button and a GitHub star count of 3,700 are also visible.

POWERED BY INSTIL.

## ≡ Kotlin Specific Libraries



## ≡ Kotlin Specific Libraries

The screenshot shows a web browser window for developer.android.com. The navigation bar includes links for Platform, Jetpack (which is highlighted), Kotlin, More, Search, Language, and Sign in. Below the navigation bar, there are tabs for Overview, Get Started, Libraries, Community, and Compose (which is underlined). The main content area features a large heading "Build better apps faster with Jetpack Compose". A sub-section below it states: "Jetpack Compose is Android's modern toolkit for building native UI. It simplifies and accelerates UI development on Android. Quickly bring your app to life with less code, powerful tools, and intuitive Kotlin APIs." To the right of this text is a screenshot of a mobile application interface with a hexagonal logo. On the left side of the main content area, there is a code snippet:

```
@Composable
fun JetpackCompose() {
    Card {
        var expanded by remember { mutableStateOf(false) }
        Column(Modifier.clickable { expanded = !expanded }) {
            Image(painterResource(R.drawable.jetpack_compose))
            AnimatedVisibility(expanded) {
                Text(
                    text = "Jetpack Compose",
                    style = MaterialTheme.typography.h2,
                )
            }
        }
    }
}
```

The screenshot shows a web browser window for jetbrains.com. The navigation bar includes links for JET BRAINS (which is highlighted), English, and a search bar. The main content area features a large heading "Compose Multiplatform" with a "Beta" badge. Below the heading is a blue hexagonal icon. A sub-section states: "Fast reactive Desktop and Web UI framework for Kotlin, based on Google's [modern toolkit](#) and brought to you by JetBrains". A "Explore on GitHub" button is located below this section. The page is divided into two main sections: "Desktop app" and "Web app".

**Desktop app**  
Compose for Desktop provides a declarative and reactive approach to creating desktop user interfaces with Kotlin. Combine composable functions to build your user interface, and enjoy full tooling support from your IDE and build system – no XML or templating language required.

Compose for Desktop targets the JVM, and supports high-performance, hardware-accelerated UI rendering on all major desktop platforms (macOS, Windows, and Linux) by leveraging the powerful native [Skia](#) graphics library. Native application distributions can be created with a single click.

**Web app**  
Compose for Web allows you to build reactive user interfaces for the web in Kotlin, using the concepts and APIs of Jetpack Compose to express the state, behavior, and logic of your application.

Compose for Web provides multiple ways of declaring user interfaces in Kotlin code, allowing you to have full control over your website layout with a declarative DOM API.

4

# Let's Code!

4