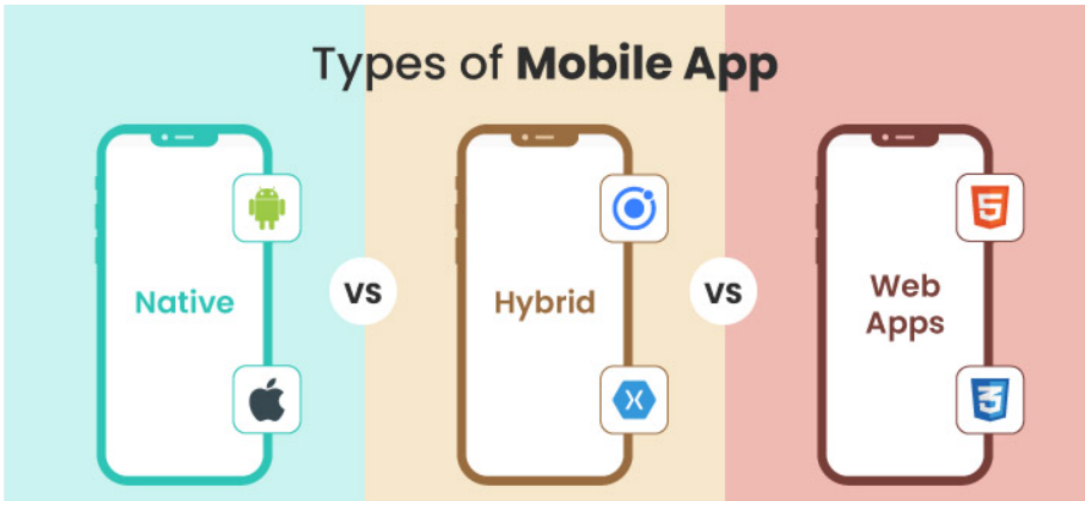
Android Developemnt Camp - by GDSC

Day - 1

- App Development Overview
  - 

  -

  - Native Apps
    - The codebase is different for different platforms.
    - Android -> Kotlin
    - iOS -> Swift
    - Example -> Whatsapp

  - Hybrid Apps
    - The Codebase is same for both android and iOS
    - Flutter, React Native
    - Example -> Google Pay

  - Why to make native apps when we can make hybrid apps?

    - Pros of Native Apps:
      - **Performance:** Native apps are typically faster and more responsive compared to hybrid apps. They are optimized for the specific platform (iOS or Android), allowing

developers to leverage platform-specific features and take full advantage of the device's capabilities.

  - **User Experience**: Native apps can provide a seamless and native user experience because they adhere to the design guidelines and user interface conventions of the respective platforms. This leads to a more intuitive and familiar interface for users.
  - **Access to Platform Features:** Native apps have direct access to the device's hardware and software features. Developers can leverage platform-specific APIs, sensors, and functionalities to create richer and more integrated applications.
  - **App Store Optimization:** Native apps are often preferred for app store optimization. They can take advantage of platform-specific features and design guidelines, which can positively impact app discoverability and user ratings.
  - **Offline Functionality:** Native apps can offer better support for offline functionality, allowing users to access certain features even when they are not connected to the internet.

- Pros of Hybrid Apps:
  - **Cross-Platform Development:** One of the main advantages of hybrid apps is that they allow developers to write code once and deploy it across multiple platforms. This can save time and resources compared to developing separate native apps for iOS and Android.
  - **Cost-Effectiveness:** Developing a single codebase for multiple platforms can be more cost-effective than maintaining separate codebases for iOS and Android. This is particularly beneficial for small to medium-sized businesses with limited resources.
  - **Faster Development:** Hybrid apps often have shorter development cycles since developers can write code once

    and deploy it across platforms. This can be advantageous when time-to-market is a critical factor.

  - **Web Technologies:** Hybrid apps are typically developed using web technologies such as HTML, CSS, and JavaScript. Developers with web development skills can leverage their existing knowledge to build mobile apps.

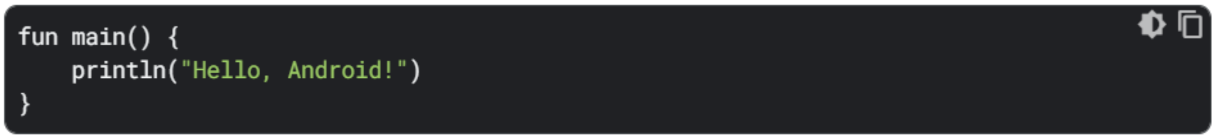
- In summary, the choice between native and hybrid app development depends on the specific needs of the project, including performance requirements, user experience goals, target audience, and development resources. Each approach has its own strengths and weaknesses, and the decision should be based on a careful evaluation of these factors.

- ## Android Studio:

- Android Studio is the official Integrated Development Environment (IDE) for Android app development.
- Recommended Version for the Camp: **Android Studio Hedgehog | 2023.1.1**
- Link to Download and Documentation: https://developer.android.com/studio
- Lets Open Android Studio

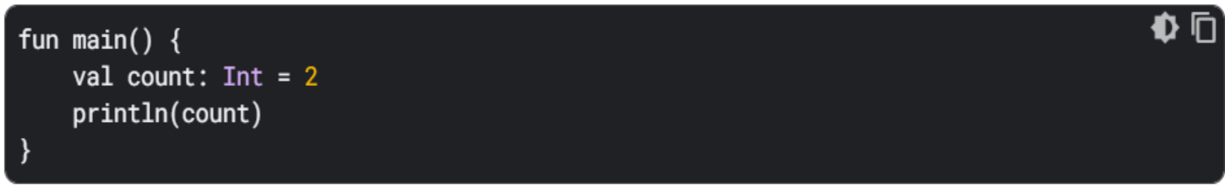
Basics of Kotlin Programming

- Kotlin Playground: https://developer.android.com/training/kotlinplayground
    - Print Hello Android



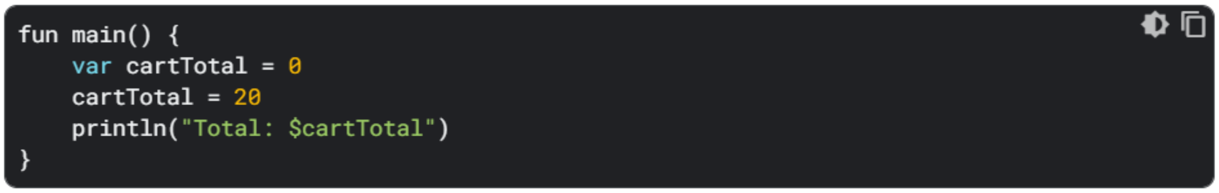
```kotlin
fun main() {
    println("Hello, Android!")
}
```

    - Variables



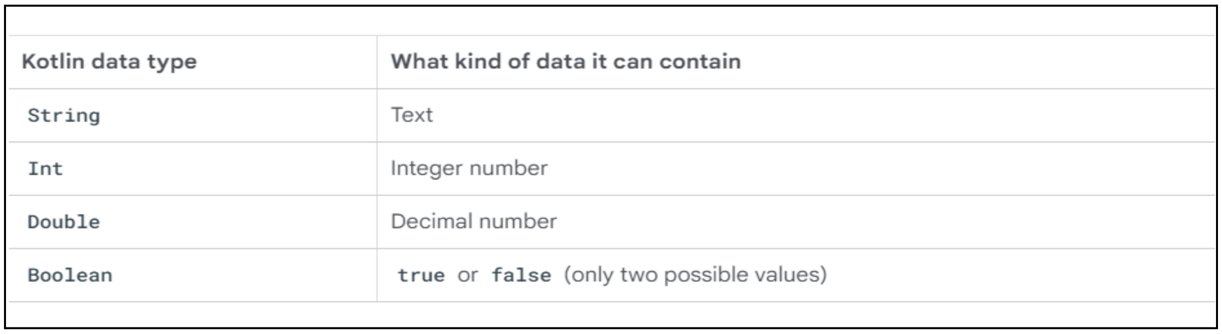
```kotlin
fun main() {
    val count: Int = 2
    println(count)
}
```

-
- Mutable Varibales



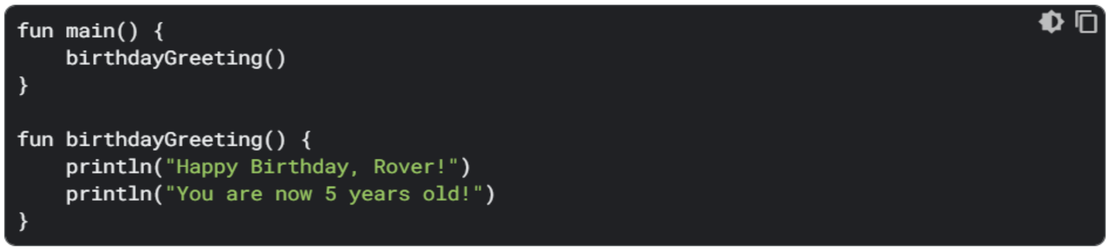
```kotlin
fun main() {
    var cartTotal = 0
    cartTotal = 20
    println("Total: $cartTotal")
}
```

-
- Other Data Types



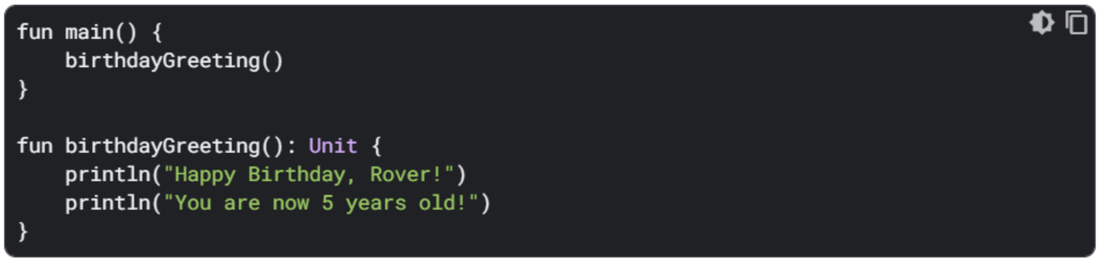
Kotlin data type	What kind of data it can contain
`String`	Text
`Int`	Integer number
`Double`	Decimal number
`Boolean`	`true` or `false` (only two possible values)

-
- Basic Function



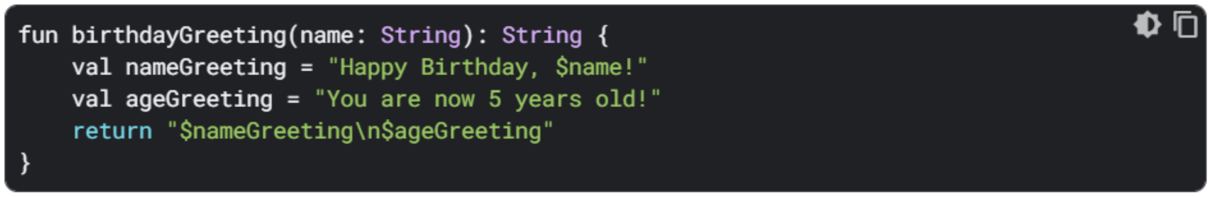
```
fun main() {
    birthdayGreeting()
}

fun birthdayGreeting() {
    println("Happy Birthday, Rover!")
    println("You are now 5 years old!")
}
```

-
- Function With a Return Type



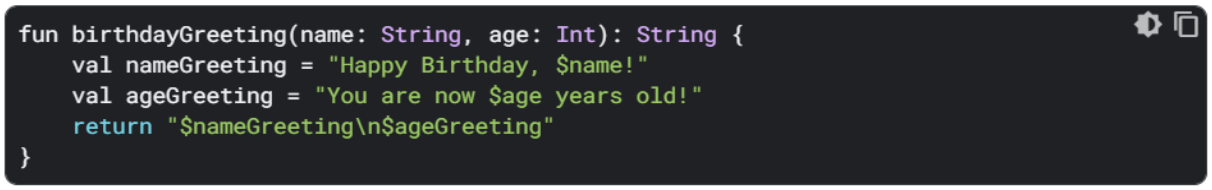
```
fun main() {
    birthdayGreeting()
}

fun birthdayGreeting(): Unit {
    println("Happy Birthday, Rover!")
    println("You are now 5 years old!")
}
```

-
- Passing Parameters to Function

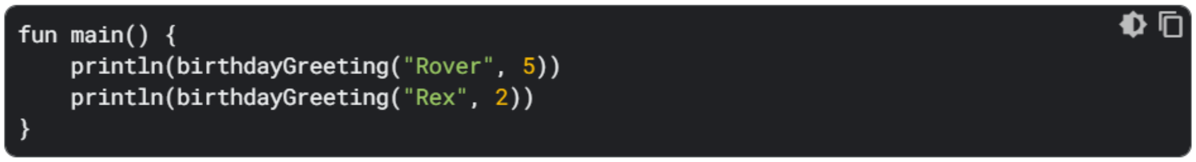


```
fun birthdayGreeting(name: String): String {
    val nameGreeting = "Happy Birthday, $name!"
    val ageGreeting = "You are now 5 years old!"
    return "$nameGreeting\n$ageGreeting"
}
```

-

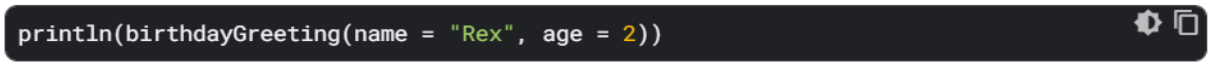


```
fun birthdayGreeting(name: String, age: Int): String {
    val nameGreeting = "Happy Birthday, $name!"
    val ageGreeting = "You are now $age years old!"
    return "$nameGreeting\n$ageGreeting"
}
```

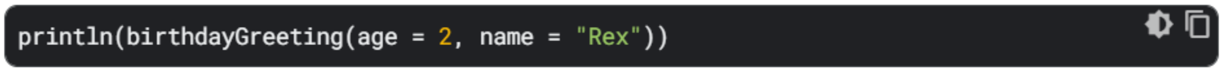


```
fun main() {
    println(birthdayGreeting("Rover", 5))
    println(birthdayGreeting("Rex", 2))
}
```

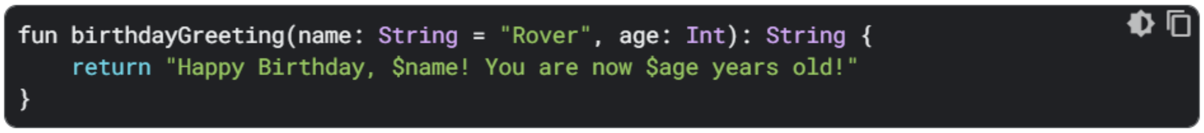
-

- Named Arguments, Order doesn't matter
- 

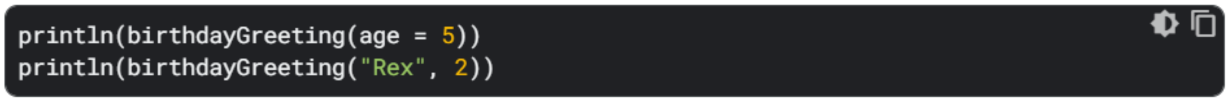
```kotlin
println(birthdayGreeting(name = "Rex", age = 2))
```



```kotlin
println(birthdayGreeting(age = 2, name = "Rex"))
```

- Default Arguments
- 

```kotlin
fun birthdayGreeting(name: String = "Rover", age: Int): String {
    return "Happy Birthday, $name! You are now $age years old!"
}
```



```kotlin
println(birthdayGreeting(age = 5))
println(birthdayGreeting("Rex", 2))
```

- Alright, we are all set to dive in!