--- Betty's script ---

Good morning everyone. We are from group 1 and today we are going to present on Kotlin.
My group consists of me (Betty), Eric, Horatiu & Ashiq. I will start off with an introduction of kotlin, demand of Kotlin, Eric will compare kotlin with java & community responses, Horatiu will talk about language performance and productivity, lastly Ashiq will talk about the future of Kotlin and conclude the presentation.

[Click]

Kotlin is an open-source statically typed, general purpose programming language developed by JetBrains that targets the Java Virtual Machine and Android with object oriented and functional programming features. The project for Kotlin started in 2010, and its first official release was in February 2016. It is an alternative to the Java language and supports Java interoperability, allowing developers to go back and forth between these two languages seamlessly. It introduces improved syntax as well as concise expressions and abstractions. It was made out of the aspiration for greater productivity and efficiency, and the goal was to improve the coding experience in a way that was both practical and effective for developers.

[Click]

Next, I will discuss the demand of Kotlin.
In 2017, when Google announced Kotlin as the official programming language of the Android platform, it led to the increased popularity of Kotlin among developers, university students and enterprise communities. Furthermore, according to a Stack Overflow Annual Developer Survey, Kotlin is the 4th most wanted and most loved language in 2018 and 2019. It is one of the fastest growing programming languages on GitHub. Companies like Coursera, Uber, Pinterest, and Netflix has publicly stated that they are switching to Kotlin for their Android applications. Even though the adoption of cross platform Kotlin development has not been explosive, major industry players are taking note of the many benefits Kotlin has to offer. Moreover, In 2019, more enterprise leaders are migrating to Kotlin or planning to do so in the future. It provides an alternative approach in programming as some of the object oriented and functional programming features available are not available in Java or not suitable for Android development. In addition, from May 2018 to May 2019, it was observed that the overall job postings for mobile app developers rose by 4.93%, and job postings looking for specifically Android developers increased by 10.61%. A recent research by Indeed revealed that companies looking for developers skilled in Kotlin skyrocketed by nearly 90% over the past year. This could indicate an element of a larger trend, that is, the iOS developer demand is falling, while the demand for Android developers is rising. When Kotlin was announced to be Google's official language, general postings for developers skilled in Swift fell by 3.46%. Even though that number might not be very significant, with the demand of Kotlin on the rise, it might be an indicator of a huge future shift in employer demand towards Android and pose a threat to Swift developers.

Now, I will hand it over to Eric who will share more about Kotlin and the comparison with Java.

+++ Eric's Section +++ DO NOT ENTER! ++

Thank you Betty for the introduction to Kotlin and the increase in demand.

Kotlin's rising popularity has created a significant attention among programming community. Increasing demand allowed many skilled programmers to catch onto Kotlin very quickly and create one of the fastest growing community in the field of mobile app development. The community base actively promotes the benefits of the language and have created multiple individual-basis code snippets, libraries and repositories that help other developers use the language. In this section, I will talk about first, similarities to Java Framework, next, about Kotlin's quirky language features, and close it off with currently available community created libraries and resources online.

Similarities to Java Framework

Kotlin was able to rise in demand due to its similarities with Java. There is an abundant amount of languages to choose from when developing software but the popularity of a language is decided based on the community base, how easy it is to learn, and the functionality it provides. In the scene of Android application development, Java has been the dominating language since 1995 (Gill). Integrated development environment, or IDE such as Android Studio, adopted Java as its primary language. With this popularity of Java in mind, programming conventions and style of Kotlin is very similar to Java. Examine the code block: This shows Java's explicit declaration and Kotlin's declaration, in which the syntax is very identical.
The similarity in style allows programmers to quickly pick up the syntax and use it in real-world implementation. The transition is seamless, and they both use the same programming paradigm, which is Object Oriented programming.

One of the benefits of object-oriented programming is inheritance. There are many online code blocks that aid code development processes, which can inherit from already existing code. Unfortunately, new language usually does not provide many choices for libraries. This usually means the developers are forced to implement missing libraries that are commonly used in application development. However, Kotlin is fully compatible with Java, meaning existing Java frameworks and libraries can be used in Kotlin. Therefore, in addition to an easy transition for developers who are versatile in Java, libraries and methods in Java are also accessible and used interchangeably in Kotlin. Compilation and building are just as easy as Java as well with build

automation systems including Gradle and Maven, which helps reduce the learning curve that new language generally introduces. **[click]**

Kotlin's Easy Syntax

Secondly, I will talk about Kotlin's language features. Kotlin introduces easy syntax and quick-to-learn programming conventions that are clear and intuitive, while having a level of sophistication that Java provides and more. It entails features such as type inference which allows developers to omit the type of declared variables, and semicolons are not necessary, which adds to the conciseness and cleanness of the final product. To demonstrate this in action, the implementation of keyword *switch* in Java is translated in Kotlin.
[click]
Switch keyword diminishes the readability of the code due to its syntax, but this is easily implemented as *when* in Kotlin. This code snippet is outputting the messages based on the numbers you obtain before this control flow statement. As you can see from the example, the number of lines and the length of each condition for Kotlin is significantly small compared to Java.
[click]

Null Protection

Kotlin Also Introduces a very useful feature called "Null Protection", but since it is a complicated topic, I will not be covering in this presentation. If you are curious about this feature, feel free to check out our report for more detail.

Adding onto Kotlin's easy and readable syntactic conventions, Kotlin has an advantage over Java in null protection**.** Java is known as a *Statically typed language* which means the types of variables are decided before run-time with the use of keywords. This is very beneficial since it prevents any type-errors that may occur, for example adding a string to an integer. In Java, you are required to check for variables of the type Null, in order to avoid null point exceptions. This is counterintuitive since it disregards the main benefit of using statically typed language. This problem is solved in Kotlin by distinguishing the variable types into nullable types and non-nullable types by adding "?" after the type declaration. What this means is that the compiler does not allow variables to have null assignment unless specified. Also, when using a nullable type variable, Kotlin forces you to guard against null-references . This is simply done by a built-in command "?".

In case x is null, ? operator makes the return value null from the call before running the called function's body, which prevents any undefined behaviours regards to null value.

[click]

Libraries and Community Support

Lastly, I will talk about Libraries and community support. Kotlin has an incredible number of libraries considering its recent introduction to the field. There are more than 2000 GitHub repositories found in “kotlin.link” website. **(Navigate to Kotlin.link)** With the benefits mentioned above, many developers adopted Kotlin as their primary language. A huge amount of online support was created such as tutorials, videos, books and very active communication channel (using Slack, over 21 K+ users). This is another factor in terms of Kotlin’s rising popularity since community involvement is very high and fresh.

[click]

One of the most popular repositories for Kotlin is Design-Patterns-In-Kotlin by Dariusz Baciński. This is a library that contains useful design patterns in OOP such as behavioural, creational and structural patterns. The code is very well documented with its intended usage outlined in README.md file.

[click]

One of the examples for behavioural design pattern is the observer pattern. Notice how readable the code is with a specific example that it shows. This gives Kotlin an edge in competing with other popular languages. This ends the section, and now I will pass it onto Horatiu for the deeper insight on Kotlin’s performance and developer productivity.

=== Horatiu's presentation ===

Thank you Eric for the overview of the Kotlin programming language and the open-source community around it. I will now be talking about the performance and productivity in the Kotlin programming language.

[Click]

Performance and developer productivity are important aspects in deciding whether a new language should be adopted by an organization. Specific characteristics of performance that should be considered include compilation and execution time. Moreover, particular language features can change the way applications are written including shorter code and easier to use functions.

[Click]

Compilation Time - Uber Case Study

The compilation time of Kotlin is similar to other popular languages such as Java. Uber conducted a case study where they benchmarked over 20 Android apps containing over 2000

modules. Their individual modules use an open-source RPC (remote procedure call) library called Apache Thrift which allows for inter-process communication. The benefit of using Apache Thrift is that as part of making the procedure calls, the library automatically generates object files for each type allowing Uber engineers to easily switch languages with only simple configuration file changes.

The results from the case study showed that pure Java projects compiled slightly quicker than pure Kotlin, while mixing both languages resulted in poorer performance. However, annotation processors, which are optionally used by developers to provide code suggestions in their development environments, made Kotlin significantly slower to compile than the Java counterpart which has a more mature “Java apt” (annotation processor tool). As a result, for an enterprise-level company where annotation processors are used Kotlin gave a productivity hit, but for general-purpose programming the performance was nearly equivalent.

[Click]
As you can see, this is an example of annotations in Java.
[Click]

Developer Productivity - KeepSafe Case Study

In addition, Kotlin provides various changes in language syntax and provides constructs/facilities that make developers more efficient in development. KeepSafe, the company behind the 100,000+ downloads app "Keepsafe Unlisted - Second Phone Number", converted their Android codebase from Java to entirely Kotlin and found a 10% reduction in the number of methods, and reduced total lines of code by 30% (Keepsafe). Moreover, they found it also reduced the need for third-party libraries such as Guava, Retrolambda and RxJava. Smaller codebases are easier to maintain, and lack dependency on external libraries making it easier for developers to contribute because there is no need to understand language constructs beyond the original language. However, despite the reduced codebase size, Kotlin still compiled 10-15% slower than Java for clean builds but is similar in performance to Java for incremental builds. Therefore, the results from the benchmarks agree with the case study presented by Uber.

[Click]
As you can see, this chart shows the lines of code in the autogenerated code for both Java and Kotlin.
[Click]

Specific Language Features

Specific language features that make Kotlin more efficient and easier to write code in than Java include 4 things: the lack of checked typed exceptions, operator overloading, coroutines along with string templates (Kotlin).

Firstly, checked typed exceptions in Java require any statement that may throw an exception (such as division by zero) to be wrapped in a try/catch block or that the method specifically indicate that it can throw that value. While it provides a static compile-time check for any possible unhandled exceptions, it is very often neglected, results in code bloat, and reduces readability and performance.

Secondly, operator overloading is having the ability to have custom methods that are invoked on operations, for example someone can overload the > operator in a Person object, such that a > b returns true only when a is older than b. The advantages of operator overloading include vastly improved code readability.

[Click]

This slide shows Java and Kotlin functions for the division of two Number objects.

In the Java example, there must be a try/catch for the division because DivisionByZeroException can be thrown, and there must be a divide method invoked because the language doesn't support method overloading. In contrast, the divide method using Kotlin is cleaner and doesn't require the try/catch, in addition to using an overloaded division operator instead which improves code readability.

[Click]

Coroutines are an advanced language feature that allows for suspending methods such that progress can be continued later. They are commonly found in the Python programming language under the name ‘generators’ and allow for significantly improved performance in particular use-cases.

[Click]

For example, if a particular user repeatedly asks for the factorial of a sequence of numbers in an increasing order (ex: 2!, 3!, 10!, 200!), in Java all the computation would need to be computed first up to 200! and then results would be returned for each in order. In contrast, using coroutines, the results can be incrementally constructed, and it allows for more immediate responses.

[Click]

Lastly, string manipulation is an extremely common task and Kotlin brings string templates to simplify the insertion of values into strings. Similar to other languages such as Python and JavaScript, Kotlin allows for templating strings and then directly substituting values efficiently without needing to modify the strings using auxiliary data-structures. For example, “Hello my name is “+ name + “!” would be replaced by “Hello my name is {}!”.format(name). The benefits are improved code readability and performance. In Java, performing the + operator which concatenates two strings results in recreating the entire string each time because internally Java cannot modify a string - which makes performance extremely slow because for each append the entire string is reprocessed. In contrast, using string templating in Kotlin, the concatenation operator doesn’t involve reconstructing the entire string which significantly improves performance.

[Click]

====== Ashiq’s presentation =======

Thank you horatiu for that insightful information on kotlin.

Now i will be talking about the future of Kotlin .
I will be covering 3 main reasons why we believe that Kotlin will grow in the years to come:

1) The brilliant foundation that was set up for Kotlin by its founder, Jetbrains
2) The potential impact, Academia could have to Kotlin when and if it is implemented
3) And how the Pros and Cons of Kotlin will be able to propel Kotlin to greater heights.

[Click] -

Firstly, Jetbrain had created Kotlin to be an open source platform as compared to a closed interface. This set the foundation for kotlin to be always growing and readily observed by various developers around the globe. With the constant accessibility to thousands of developers, the number of bugs and errors in the systems are drastically decreased . And as the popularity of Kotlin increases, the benefits of being an open source platform will also increase due to the increased traffic of the language. Moreover, from a financial point of view, open source platforms are relatively cheaper hence allowing an easier adoption in various enterprises.

Next, we look into impact academia will have on kotlin.

[Click]

Currently, Kotlin has yet to be widely used in universities around the world as it is a relatively new language . However, there has been a start in debate whether it should be in the curriculum for university students.Due to the lack of research on this we are unable to provide concrete evidence on the benefits of Academia on Kotlin. However we can use other programming languages to form a correlation between Academia and popularity of the language.

[Click]

For this we looked further into python . In early 2010's python was the most thought programming language in the United States (US). From a study of 39 universities in the US, 27 of them thought Python as an introductory course. Moreover, accordingly to Stack Overflow, most of the visits to questions on python were by students.

With these 2 information combined, we can make a speculative analysis that an increase in the teaching of the programming language could lead to an increase in the usage of the language.

Many of us here can relate to this findings as it is true that many languages that are popular today were once thought in universities.

[Click]
Lastly , we will analyse the pros and cons of Kotlin and how it will propel kotlin into the future, Some of them are as shown on the screen. Looking through the table , the Cons of implementing the programme may somewhat negligible to the pros. And this is indeed true, if enterprises were willing to forgo the minimalist cons of Kotlin and embrace Kotlin for its potential, the popularity of Kotlin will boost drastically.

With all the 3 major points combined, we could potentially see Kotlin over take Java in the future.

[click]

let's move on to the conclusion of the report. Before I begin, do take a few seconds to read the quote written there.

……..

This quote could not be more relevant to Kotlin. Kotlin as been able to seamlessly enter the lives of application developers due to the assistance of Google. As Betty previously mentioned, the increase in demand for kotlin could be evidence to this statement. We also see how enterprises have begun to test and adopt kotlin from what Horatiu has shared.

If this trend continues, Kotlin will have a place as a major programming in the near future.

With that we end our presentation and thank you for listening.