Recap

Recap what you learned about Kotlin, its principles, and how they drive language features.

WE'LL COVER THE FOLLOWING

You now have a solid understanding of the basics of Kotlin. Before we explore ways to continue your Kotlin journey, let's briefly recap all that you learned in this course. After all, repetition is one of the keys to learning.

Variables and Mutable Types

- Kotlin favors the use of read-only variables.
  - Use `val` to create a read-only variable.
  - Use `var` to create a mutable variable.
  - First, note how declaring a read-only variable takes no more characters than a mutable one, in contrast to many other languages (e.g., Java requiring the `final` modifier).
  - Second, note that variable names in your code are usually nicely aligned because the two keywords have the same number of characters.
- In alignment with this, Kotlin also differentiates between read-only and mutable collection types, again favoring the use of read-only ones.
  - For instance, a simple `listOf(...)` will give you a read-only list.

    - To get a mutable list, you have to explicitly write `mutableListOf(...)`, making read-only the default case.

Nullable Types and Type Inference

- One of Kotlin's main features regarding safety is nullability anchored in the type system.
    - Every type is non-nullable by default.
    - Thus, every variable is either explicitly nullable or cannot be `null` (to be precise, there are hacky ways to set `null`, but you won't usually come across these).
    - You should prefer non-nullable types over nullable ones because it simplifies code and prevents the common null pointer exception.
- The Kotlin compiler can infer the type of most expressions.
    - For instance, declaring the type of a variable is almost always optional.
    - You can always choose to declare types explicitly to improve readability or to program against a more abstract type.

Conditions and Expressions

- Kotlin has `if` and `when` to define conditions.
    - `when` can match against a variety of conditions and is therefore more powerful than `switch` from other languages.
- Both `if` and `when` are expressions in Kotlin.
    - Thus, you can assign a variable to the result of a condition.
        - The value is defined by the value in the last line of each condition block.
    - If a function contains only a condition, you can rewrite it using shorthand notation when using the condition as an expression.
- As a side note, most constructs in Kotlin are expressions because it allows for a more functional style of programming.

Collections

- The Kotlin Collections API directly builds upon the Java Collections API.
    - In contrast to Java, it clearly separates between read-only and

mutable collections.

- The common collection types are `List`, `Set`, and `Map`.
  - A **list** is an ordered linear data structure to store multiple elements and can contain duplicates.
  - A **set** is an unordered linear data structure to store multiple elements and cannot contain duplicates.
  - A **map** is an unordered nonlinear data structure to store multiple key-value pairs and can contain each key only once.

Loops

- Kotlin offers `for` and `while` loops.
  - `while` allows repeating a block of code until a given condition is no longer true.
  - `for` allows iterating over any `Iterable` object.
    - This includes collections, string, ranges (`1..10`), arrays, etc.
    - This differs from the `for` loop you may know from other languages where you define a loop counter variable.
      - It acts like a “for-each” loop, which is also available in many other languages.

Functions

- Kotlin offers several interesting features regarding function declarations.
  - For functions with a single expression in their body, you can use shorthand notation to avoid clutter.
  - Function parameters can have default values, eliminating the need for function overloading 90% of the time.
  - When calling a function, you can explicitly denote the parameter names for each argument to increase readability in certain cases.
  - Extension functions allow virtually extending the API of a type.
    - This is extremely useful to smoothen rough edges in third-party APIs.
  - You can define infix functions to write the function name between its two arguments, which can improve readability.
  - Lastly, you can implement a predefined list of operator functions on any type you want.

any type you want.

- Since it focuses on functional programming, Kotlin offers even more interesting features with regards to lambda functions, which were not covered in this course.

Summary

Congratulations! 🎉 Having finished this course, you're now able to write simple Kotlin scripts and applications. Also, with this solid understanding of the language basics, you'll be able to tackle more advanced material, such as functional and object-oriented programming (see the next lesson).

Thank you for making it through the entire course! I hope you enjoyed the journey and learned a lot along the way.

In the final lesson, I'll leave you with a hand-selected list of further resources about Kotlin that I'd recommend as the next step if you want to learn more about this fantastic programming language.