LET'S TALK ABOUT KOTLIN

LEARN ALL ABOUT IT

BY: NISHANT GUPTA

KOTLIN

INDEX

KOTLIN

INTRODUCTION

Kotlin is a cross-platform, statically-typed, general-purpose programming language with type inference. It is designed to improve productivity in an enterprise setting and is fully interoperable with Java. Kotlin is an officially supported language for Android development, and can be used to write both Android apps and server-side code. It is known for its safety features, such as null safety and immutability, as well as its concise and expressive syntax.

Kotlin is an object-oriented language, and a “better language” than Java, but still be fully interoperable with Java code.

It was announced in 2017 that Kotlin is an official language for Android development, sponsored by Google.

KOTLIN

KOTLIN ENVIRONMENT SETUP FOR COMMAND LINE

To set up a Kotlin development environment for the command line, you will need to have the Java Development Kit (JDK) installed on your computer. You can download the JDK from the Oracle website. Once the JDK is installed, you can download the latest version of the Kotlin compiler from the Kotlin website.

Once you have downloaded the Kotlin compiler, you will need to add it to your system's PATH environment variable so that you can use the Kotlin command-line tools.

On Windows, you can do this by opening the System Properties dialog **(press Win + Pause/Break)**, clicking on **"Advanced system settings"**, and then clicking on **"Environment Variables"**.

On macOS or Linux, you can add the Kotlin compiler to your PATH by editing your **.bash**_profile or **.bashrc** file.

After adding the Kotlin compiler to your PATH, you can open a command prompt or terminal and type **"kotlinc"** to check if it's working.

To write and run Kotlin code, you can use a simple text editor like Notepad or TextEdit to create a file with the .kt extension, and then use the "kotlinc" command to compile the file.

```
kotlinc Hello.kt -include-runtime -d Hello.jar
```

This command will compile the Kotlin code in the "Hello.kt" file and produce a JAR file containing the compiled code and the Kotlin runtime libraries.
You can then run the JAR file using the "java" command.

```
java -jar Hello.jar
```

You can also use an IDE like IntelliJ IDEA or Eclipse, which have built-in support for Kotlin development and provide a more advanced development environment, with features such as code completion, error checking, and debugging.

KOTLIN

HELLO WORLD PROGRAM

Hello, World! is the first basic program in any programming language. Let's write the first program in Kotlin programming language.

The "Hello, World!" program in Kotlin:

Step 1:Open your favorite editor notepad or notepad++ and create a file named firstapp.kt with the following code.

You can compile the program in the command-line compiler.

$ kotlinc firstapp.kt

Now, Run the program to see the output in a command-line compiler.

$ kotlin firstapp.kt

Hello, World!

KOTLIN

VARIABLES

In Kotlin, variables are used to store data. There are two types of variables in Kotlin: var and val.

- **var**: A variable declared using the keyword var is a mutable variable. This means its value can be changed after it is declared.

```
var x = 5
x = 6
```

- **val:** A variable declared using the keyword val is an immutable variable. This means its value cannot be changed after it is declared.

```
val y = 5
y = 6 // This will give an error
```

In addition to var and val, Kotlin also supports type inference. This means that the compiler can automatically deduce the type of a variable based on its value.

```
var x = 5 // x is inferred to be of type Int
val y = "Hello" // y is inferred to be of type String
```

KOTLIN

Kotlin also supports type-safe null handling. It differentiates between nullable and non-nullable types. A variable that can hold a null value is called nullable and is defined by adding a question mark (?) after the type.

```
var name: String? = null
val age: Int = 25
```

To access the value of a nullable variable, you need to use the safe call operator (?.) or the Elvis operator (?:).

Overall, Kotlin variables are designed to be type-safe and null-safe, which helps in avoiding common programming errors.

DATA TYPES

There are different data types in Kotlin:

1. Integer Data type
2. Floating-point Data Type
3. Boolean Data Type
4. Character Data Type

Integer Data Type

These data types contain integer values.

Data Type	Bits	Min Value	Max Value
byte	8 bits	-128	127
short	16 bits	-32768	32767
int	32 bits	-2147483648	2147483647
long	64 bits	-9223372036854775808	9223372036854775807

Floating-Point Data Type

These data type used to store decimal value or fractional part.

Data Type	Bits	Min Value	Max Value
float	32 bits	1.40129846432481707e-45	3.40282346638528860e+38
double	64 bits	4.94065645841246544e-324	1.79769313486231570e+308

Boolean Data Type

Boolean data type represents only one bit of information either **true** or **false**.

The Boolean type in Kotlin is the same as in Java. These operations disjunction (||) or conjunction (&&) can be performed on boolean types.

Data Type	Bits	Data Range
boolean	1 bit	true or false

Character Data Type

Character data type represents the small letters(a-z), Capital letters(A-Z), digits(0-9) and other symbols.

Data Type	Bits	Min Value	Max Value
char	16 bits	'\u0000' (0)	'\uFFFF' (65535)

CONTROL FLOW

Control flow statements are used to control the order of execution of statements in a program. Kotlin supports several control flow statements, including:

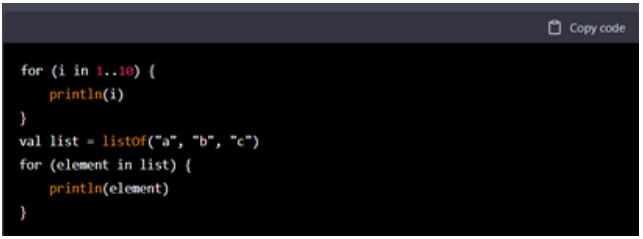
1. **If/else:** The if/else statement is used to evaluate a condition and execute a certain block of code if the condition is true, and another block of code if the condition is false.

```
val x = 5
if (x > 0) {
    println("x is positive")
} else {
    println("x is non-positive")
}
```

2.**When:** The when statement is used as a replacement for a switch statement in other languages. It allows you to evaluate an expression and execute a block of code corresponding to the first matching condition.

```
val x = 5
when (x) {
    0 -> println("x is 0")
    1, 2, 3 -> println("x is 1, 2 or 3")
    in 4..10 -> println("x is in the range 4 to 10")
    else -> println("x is something else")
}
```

3. **For loops:** The for loop is used to iterate over a range or a collection of elements.



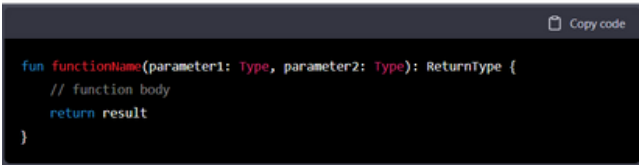
4.**While loops:** The while loop is used to execute a block of code repeatedly as long as a certain condition is true.

5. **Do-while loops:** The do-while loop is similar to the while loop, but it guarantees that the body of the loop will be executed at least once, before the condition is evaluated.

FUNCTIONS

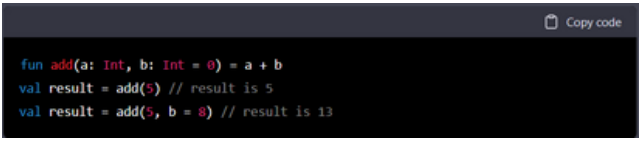
Functions are a fundamental building block in Kotlin. They are used to group a set of statements together to perform a specific task.

1. **Function Declaration:** Functions are declared using the keyword fun, followed by the function name, a list of parameters and the function body. The basic syntax is:

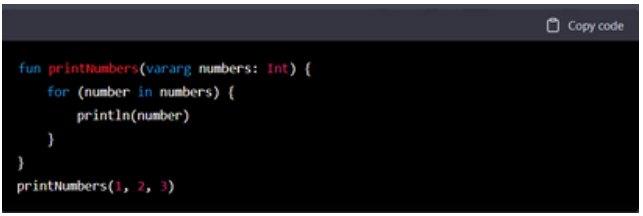


2. **Function Call:** To call a function, you simply use the function name followed by the list of arguments in parentheses.

3. **Default and Named Arguments:** Functions can have default values for some or all of its arguments, which allows calling the function without providing values for those arguments. Named arguments allow the caller to specify which argument they are passing the value to.



4.**Variable Number of Arguments (varargs):** Functions can also take a variable number of arguments using the vararg keyword.



5. **Higher-order Functions:** Functions in Kotlin can be passed as arguments to other functions or can be returned as the result of a function. These are called higher-order functions. Higher-order functions are useful for creating more generic, reusable code.

```
fun operation(x: Int, y: Int, op: (Int, Int) -> Int): Int {
    return op(x, y)
}
val result = operation(2, 3, ::add)
```

6. **Lambdas:** They are anonymous functions that can be passed as arguments to higher-order functions.

```
val result = operation(2, 3) { x, y -> x * y }
```

Functions in Kotlin provide a powerful and expressive way of organizing code, making it more readable, maintainable and reusable.

CLASSES AND OBJECTS

Kotlin is an object-oriented programming language, which means it supports the creation of classes and objects.

1. **Class Declaration:** A class is a blueprint for creating objects (a particular data structure), providing initial values for state (member variables or properties) and implementations of behavior (member functions or methods). The basic syntax for creating a class is:

```
class ClassName {
    // properties
    // methods
}
```

2. **Object Declaration:** An object is an instance of a class. To create an object of a class, you use the keyword "new" followed by the class name.

```
val obj = ClassName()
```

3. **Properties:** A property is a variable that is a member of a class. Properties can be defined in a class using the keywords "var" or "val".

```
class Person {
    var name: String = ""
    var age: Int = 0
}
```

4. **Methods:** Methods are functions that are a member of a class. Methods are defined using the keyword "fun" followed by the method name.

```
class Person {
    fun display() {
        println("Name: $name")
        println("Age: $age")
    }
}
```

5.**Constructors:** A constructor is a special kind of method that is used to create and initialize an object when it is created. Kotlin has two types of constructors: primary and secondary constructors.

```
class Person(val name: String, var age: Int) {
    // secondary constructor
    constructor(name: String) : this(name, 0)
}
```

6. **Inheritance:** Inheritance is a mechanism by which one class can inherit properties and methods from another class. To inherit from a class, use the ":" followed by the parent class name.

```
open class Shape {
    // properties and methods
}
class Rectangle : Shape() {
    // properties and methods
}
```

7. **Polymorphism:** Polymorphism is the ability of an object to take on multiple forms.

```
open class Shape {
    open fun draw() {
        // implementation
    }
}
class Rectangle : Shape() {
    override fun draw() {
        // implementation
    }
}
```

Classes and objects in Kotlin provide a powerful and expressive way of organizing code, creating reusable and modular code, and encapsulating data and behavior.

KOTLIN

COLLECTIONS AND GENERICS

Collections and generics are important concepts in Kotlin programming.

- **Collections:** Kotlin provides a rich set of collection classes, such as List, Set and Map. These classes provide a variety of useful operations for working with collections of data, such as adding, removing, and searching for elements.

```
val list = listOf("a", "b", "c")
val set = setOf(1, 2, 3)
val map = mapOf(1 to "one", 2 to "two", 3 to "three")
```

- **Generics:** Generics are used to provide type safety in collections. The collection classes in Kotlin are all generic, which means you can specify the type of elements they will contain.

```
val list: List<String> = listOf("a", "b", "c")
val set: Set<Int> = setOf(1, 2, 3)
val map: Map<Int, String> = mapOf(1 to "one", 2 to "two", 3 to "three")
```

- **Filter, map, and reduce:** These are higher-order functions that are available on all collection classes. They allow you to perform common operations on collections, such as filtering elements, transforming elements and reducing the collection to a single value.

```
val numbers = listOf(1, 2, 3, 4, 5)
val evenNumbers = numbers.filter { it % 2 == 0 }
val doubledNumbers = numbers.map { it * 2 }
val sum = numbers.reduce { acc, i -> acc + i }
```

- **Extension functions:** Kotlin provides a lot of useful extension functions for working with collections, such as forEach, find, groupBy, and many more. These functions are available on all collection classes, and they make working with collections more convenient and readable.

```
val numbers = listOf(1, 2, 3, 4, 5)
numbers.forEach { println(it) }
val firstEvenNumber = numbers.find { it % 2 == 0 }
val numbersByMod3 = numbers.groupBy { it % 3 }
```

Overall, collections and generics in Kotlin provide a powerful and expressive way of working with data, making your code more readable, maintainable and reusable.

FUNCTIONAL PROGRAMMING

Functional programming is a programming paradigm that emphasizes the use of functions to solve problems. Kotlin is a functional programming language and it provides several features for functional programming.

1. **Lambdas:** Lambdas are anonymous functions that can be passed as arguments to higher-order functions. They are written as expressions and have the following syntax:

```
val sum = { x: Int, y: Int -> x + y }
```

2. **Higher-order functions:** Higher-order functions are functions that take other functions as arguments or return a function as a result. Some examples of higher-order functions in Kotlin are filter, map and reduce.

```
val numbers = listOf(1, 2, 3, 4, 5)
val evenNumbers = numbers.filter { it % 2 == 0 }
val doubledNumbers = numbers.map { it * 2 }
val sum = numbers.reduce { acc, i -> acc + i }
```

3. **Closures:** Closures are a way to capture the state of a function and use it later. In Kotlin, closures are created using lambda expressions.

```
val greet = { "Hello " }
val name = "world"
println(greet() + name)
```

4. **Function composition:** Function composition is the process of combining multiple functions to create a new function. In Kotlin, function composition can be achieved using the pipe operator (..) or by chaining function calls.

```
val increment = { x: Int -> x + 1 }
val double = { x: Int -> x * 2 }
val incrementAndDouble = increment.andThen(double)
val result = incrementAndDouble(5)
```

5. **Currying:** Currying is the technique of transforming a function that takes multiple arguments into a chain of functions, each taking a single argument. In Kotlin, it can be achieved by using higher-order functions to partially apply function arguments.

```
fun add(x: Int) = { y: Int -> x + y }
val add5 = add(5)
val result = add5(3)
```

Functional programming in Kotlin provides a more declarative and expressive way of solving problems, making the code more readable and maintainable. It also makes it easy to parallelize the code, and helps in avoiding common programming errors.

CONCURRENCY AND MULTI-THREADING

Concurrency and multi-threading are important concepts in programming as they allow you to take full advantage of the capabilities of modern hardware, and improve the performance and responsiveness of your applications. Kotlin provides several features for concurrency and multi-threading, including:

1. **Threads:** The Thread class in Kotlin can be used to create and control threads. You can create a thread by instantiating the Thread class and passing a lambda to its constructor.

```
val thread = Thread {
    // code to run in the thread
}
thread.start()
```

2. **Coroutines:** Coroutines are a lightweight alternative to threads that allow you to write asynchronous and non-blocking code. Coroutines are launched using the '**launch**' function, and they can be cancelled or suspended using the '**cancel**' and '**suspend**' functions.

```
val job = GlobalScope.launch {
    // code to run in the coroutine
}
job.cancel()
```

3. **Concurrent Collections:** Kotlin provides thread-safe versions of the collection classes, such as ConcurrentMap and ConcurrentHashMap. These classes provide a way to access and modify their elements safely from multiple threads.

```
val map = ConcurrentHashMap<String, Int>()
map["a"] = 1
```

4. **Synchronized Blocks:** The **synchronized** function can be used to create a block of code that can only be executed by one thread at a time. This is useful for accessing shared resources safely from multiple threads.

```
val counter = 0
val lock = Any()
fun incrementCounter() {
    synchronized(lock) {
        counter++
    }
}
```

5. **Atomic Operations:** Kotlin provides the '**AtomicXXX**' classes such as AtomicInteger and AtomicLong, that provide atomic operations such as increment and compare-and-swap. These classes can be used to perform operations on shared variables in a thread-safe way.

```
val counter = AtomicInteger()
counter.incrementAndGet()
```

Kotlin's concurrency and multi-threading features provide a powerful and expressive way of writing concurrent and parallel code, making it easier to take full advantage of the capabilities of modern hardware and improve the performance and responsiveness of your applications.

ADVANCED KOTLIN FEATURES

Kotlin provides several advanced features that allow for more expressive and powerful programming. Some examples of advanced features in Kotlin include:

1. **Reflection:** Kotlin provides built-in support for reflection, which allows you to examine and manipulate the structure of your code at runtime. This can be useful for generating code, creating custom serialization and deserialization, and many other use cases.

```
val cls = MyClass::class
val constructor = cls.constructors.first()
val instance = constructor.call()
```

2. **Annotation Processing:** Kotlin supports annotation processing, which allows you to define custom annotations and use them to generate code or perform other actions at compile-time. This can be used for tasks such as code generation, dependency injection, and more.

```
@Target(AnnotationTarget.CLASS)
annotation class MyAnnotation

@MyAnnotation
class MyClass
```

3.**Type Inference:** Kotlin has a powerful type inference system that allows the compiler to deduce the types of variables and expressions based on their values. This can make your code more concise and readable.

```
val x = 5 // x is inferred to be of type Int
val y = "Hello" // y is inferred to be of type String
```

4. **Extension Functions:** Kotlin allows you to extend the functionality of a class by adding new functions to it, without modifying the class itself. This can be useful for adding utility functions, or for creating custom DSLs.

```
fun String.repeat(n: Int): String {
    return this.repeat(n)
}
val result = "Hello".repeat(3)
```

5. **Type Aliases:** Kotlin allows you to define type aliases, which allow you to give a different name to

KOTLIN

BUILDING ANDROID APPS WITH KOTLIN

Building Android apps with Kotlin is a popular and efficient way to develop Android applications. Kotlin is fully compatible with Android and provides several features that make it a great choice for Android development.

1. **Interoperability with Java:** Kotlin is fully interoperable with Java, which means that you can use Java code in a Kotlin project and vice versa. This allows you to take advantage of existing Java libraries and frameworks when building your Android app.

2. **Android KTX:** Android KTX is a set of Kotlin extension functions that are designed to make Android development with Kotlin more concise and expressive. It provides a lot of useful extension functions that simplify working with Android APIs and make the code more readable.

```
val textView = findViewById<TextView>(R.id.text_view)
textView.text = "Hello Kotlin!"
```

3. **Android Studio Support:** Android Studio, the official integrated development environment (IDE) for Android, has built-in support for Kotlin. It provides advanced features such as code completion, debugging, and refactoring specifically designed for Kotlin, making it an efficient and productive development environment for building Android apps.

4. **Anko Library:** Anko is a library for Kotlin that provides a set of helpful extensions for Android development. It includes a wide range of utility functions for working with views, layouts, and more. It also provides a set of DSLs that simplify the creation of layouts and dialogs, making the code more readable.

```
verticalLayout {
    editText {
        hint = "Name"
    }
    button("OK")
}
```

5. **Data Binding**

CONCLUSION

In conclusion, Kotlin is a modern, powerful, and expressive programming language that provides a wide range of features for developing high-quality and maintainable software. It is fully interoperable with Java, which makes it easy to integrate with existing Java code and libraries. Kotlin also provides several features specifically designed for Android development such as Android KTX, Anko Library and Data Binding, which make it a great choice for building Android apps. With its concise syntax, powerful type inference, and support for functional programming, Kotlin provides a more efficient and productive development experience. Additionally, its growing popularity and support from major companies such as Google, make it a language worth considering for any new projects.