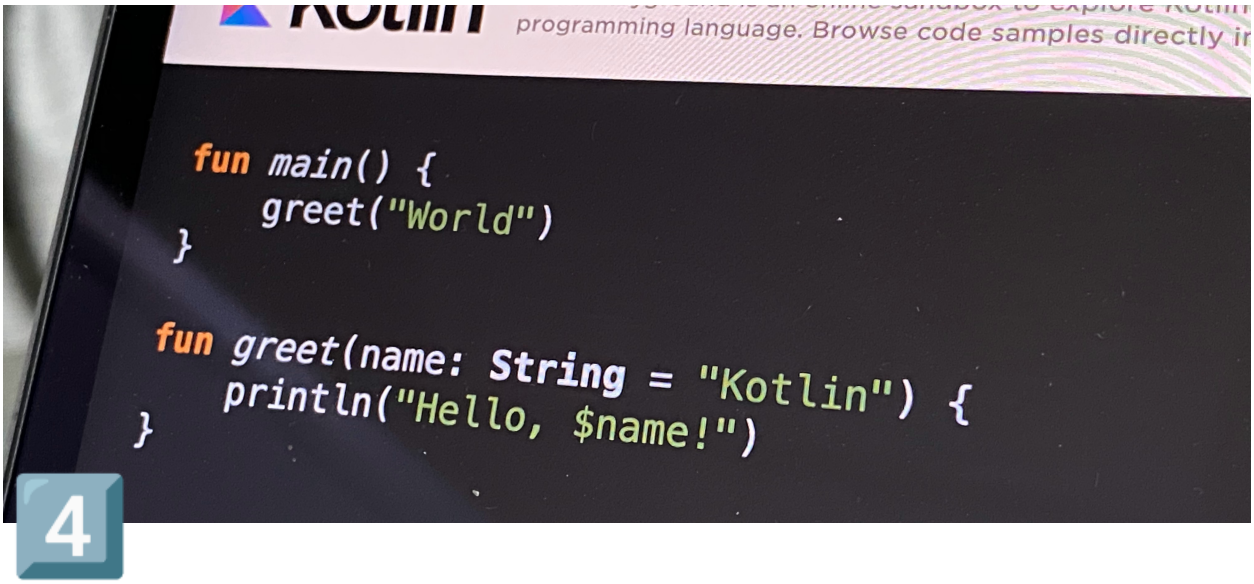


Class 4

Kotlin Hello World - You First Kotlin Program

A "Hello, World!" is a simple program that outputs `Hello, World!` on the screen. Since it's a very simple program, it's often used to introduce a new programming language.

Before you write the program, make sure your computer can run Kotlin. For that visit: How to Run Kotlin on Your Computer?

Let's explore how "Hello, World!" program works in Kotlin.

Kotlin "Hello, World!" Program

```
// Hello World Program

fun main(args : Array<String>) {
    println("Hello, World!")
}
```

When you run the program, the output will be:

```
Hello, World!
```

How this program works?

1. `// Hello World Program` Any line starting with `//` is a comment in Kotlin (similar to Java). Comments are ignored by the compiler. They are intended for person reading the code to better understand the intent and functionality of the program. To learn more, visit *Kotlin comments*.
2. `fun main(args : Array<String>) { ... }` This is the `main` function, which is mandatory in every Kotlin application. The Kotlin compiler starts executing the code from the `main` function.The function takes array of strings as a parameter and returns Unit. You will learn about functions and parameters in later chapters.For now, just remember that `main` function is a mandatory function which is the entry point of every Kotlin program. The signature of `main` function is:

   ```
   fun main(args : Array<String>) {
       ... .. ...
   }
   ```

3. `println("Hello, World!")` The `println()` function prints the given message inside the quotation marks and newline to the standard output stream. In this program, it prints `Hello, World!` and new line.

Comparison With Java "Hello, World!" program

As you know, Kotlin is 100% interoperable with Java. Here's an equivalent Java "Hello, World!" program.

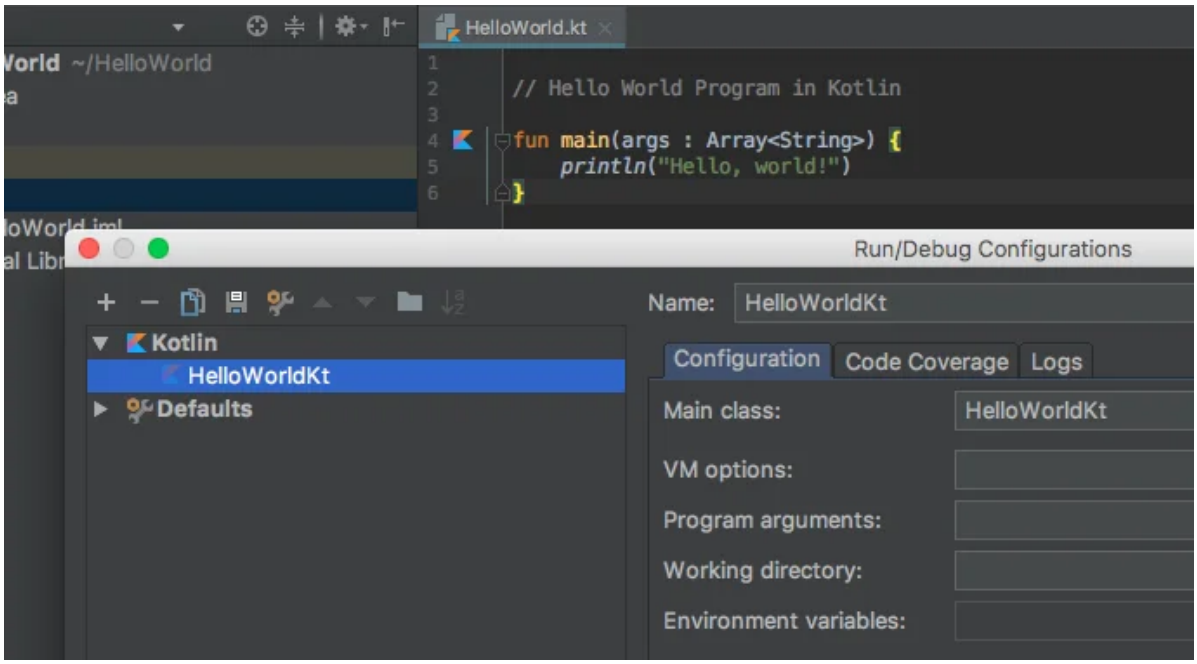
```
// Hello World Program

class HelloWorldKt {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
```

```
    }
}
```

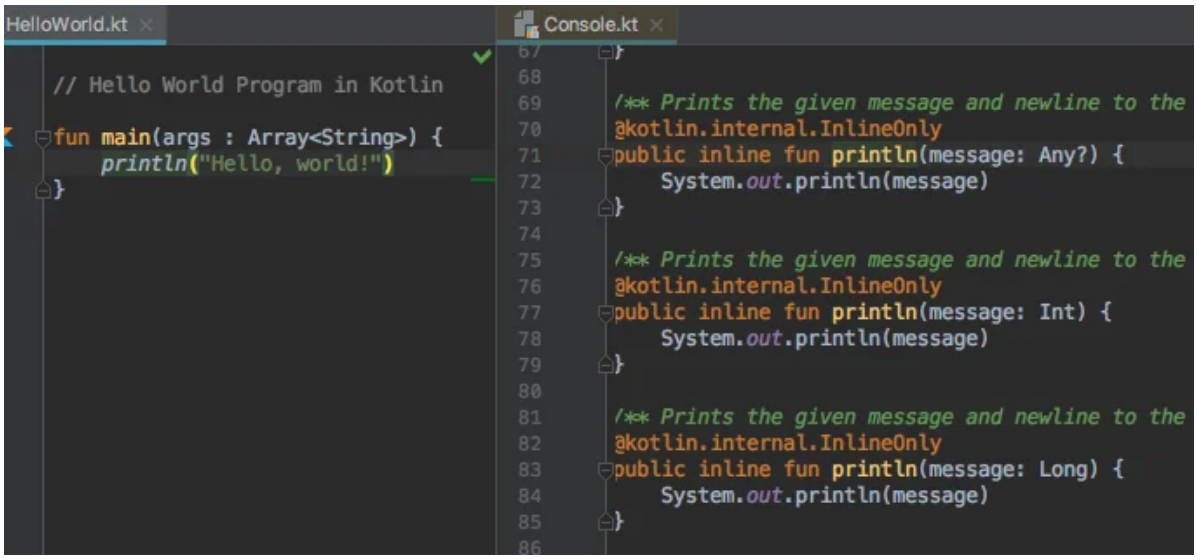
Few Important Notes

1. Unlike Java, it is not mandatory to create a `class` in every Kotlin program. It's because the Kotlin compiler creates the class for us.If you are using IntelliJ IDEA, go to `Run` > `Edit Configurations` to view this class. If you named your Kotlin file **HelloWorld.kt**, the compiler creates class.

   HelloWorldKt



2. The `println()` function calls `System.out.println()` internally.If you are using IntelliJ IDEA, put your mouse cursor next to `println` and go to `Navigate` > `Declaration` (Shortcut: **Ctrl + B** . For Mac: **Cmd + B**), this will open `Console.kt` (declaration file). You can see that `println()` function is internally calling `System.out.println()`.



Kotlin Variables and Basic Types

As you know, a variable is a location in memory (storage area) to hold data.

To indicate the storage area, each variable should be given a unique name (identifier). Learn more about How to name a variable in Kotlin?

How to declare a variable in Kotlin?

To declare a variable in Kotlin, either `var` or `val` keyword is used. Here is an example:

```
var language = "French"
val score = 95
```

The difference in using var and val is discussed later in the article. For now, let's focus on variable declaration.

Here, language is a variable of type `String`, and `score` is a variable of type `Int`. You don't have to specify the type of variables; Kotlin implicitly does that for you. The compiler knows this by initializer expression ("French" is a `String`, and 95 is an integer value in the above program). This is called type inference in programming.

However, you can explicitly specify the type if you want to:

```
var language: String = "French"
val score: Int = 95
```

We have initialized variable during declaration in above examples. However, it's not necessary. You can declare variable and specify its type in one statement, and initialize the variable in another statement later in the program.

```
var language: String      // variable declaration of type String
... .. ...
language = "French"       // variable initialization

val score: Int          // variable declaration of type Int
... .. ...
score = 95             // variable initialization
```

Here are few examples that results into error.

```
var language           // Error
language = "French"
```

Here, the type of language variable is not explicitly specified, nor the variable is initialized during declaration.

```
var language: String
language = 14         // Error
```

Here, we are trying to assign 14 (integer value) to variable of different type (`String`).

Difference Between var and val

- **val** (Immutable reference) - The variable declared using `val` keyword cannot be changed once the value is assigned. It is similar to *final variable in Java*.
- **var** (Mutable reference) - The variable declared using `var` keyword can be changed later in the program. It corresponds to regular Java variable.

Here are few examples:

```
var language = "French"
language = "German"
```

Here, `language` variable is reassigned to German. Since, the variable is declared using `var`, this code work perfectly.

```
val language = "French"
language = "German"      // Error
```

You cannot reassign language variable to `German` in the above example because the variable is declared using `val`.

Now, you know what Kotlin variables are, it's time to learn different values a Kotlin variable can take.

Kotlin Basic Types

Kotlin is a statically typed language like Java. That is, the type of a variable is known during the compile time. For example,

```
val language: Int
val marks = 12.3
```

Here, the compiler knows that language is of type `Int`, and marks is of type `Double` before the compile time.

The built-in types in Kotlin can be categorized as:

- Numbers
- Characters
- Booleans
- Arrays

Number Type

Numbers in Kotlin are similar to Java. There are 6 built-in types representing numbers.

- Byte
- Short
- Int
- Long
- Float
- Double

1. Byte

- The `Byte` data type can have values from -128 to 127 (8-bit signed two's complement integer).
- It is used instead of `Int` or other integer data types to save memory if it's certain that the value of a variable will be within [-128, 127]
- Example:

```
fun main(args : Array<String>) {
    val range: Byte = 112
    println("$range")

    // The code below gives error. Why?
    // val range1: Byte = 200
}
```

When you run the program, the output will be:

```
112
```

Class Practice1

```
fun main(args: Array<String>) {
    var num:Byte = 127
    var num1:Byte = -128

    var a = Byte.MIN_VALUE

    println(num)
    println(num1)
    println(a)
}
```

2. Short

- The `Short` data type can have values from -32768 to 32767 (16-bit signed two's complement integer).
- It is used instead of other integer data types to save memory if it's certain that the value of the variable will be within [-32768, 32767].
- Example:

```
fun main(args : Array<String>) {

    val temperature: Short = -11245
    println("$temperature")
}
```

When you run the program, the output will be:

```
-11245
```

Class Practice

```
```

3. Int

- The `Int` data type can have values from `2 31` to `2 31 1` (32-bit signed two's complement integer).
- Example:

```
fun main(args : Array<String>) {

    val score: Int =  100000
    println("$score")
}
```

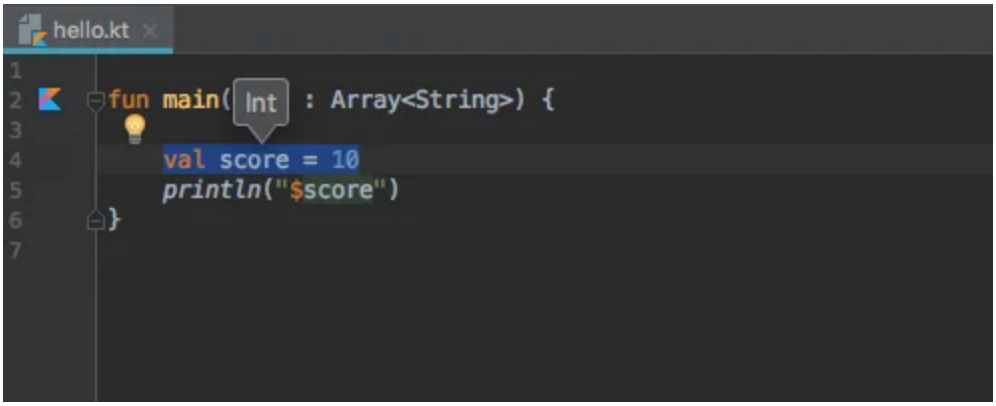
When you run the program, the output will be:

```
100000
```

If you assign an integer between `-2 31` to `2 31 -1` to a variable without explicitly specifying its type, the variable will be of `Int` type. For example,

```
fun main(args : Array<String>) {

   // score is of type Int
    val score = 10
    println("$score")
}
```

If you are using IntelliJ IDEA, you can place cursor inside the variable and press `Ctrl + Shift + P` to see its type.



4. Long

- The `Long` data type can have values from `2 63` to `2 63 1` (64-bit signed two's complement integer).
- Example:

```
fun main(args : Array<String>) {

    val highestScore: Long = 9999
    println("$highestScore")
}
```

When you run the program, the output will be:

```
9999
```

If you assign an integer value greater than `2 31 -1` or less than `-2 31` to a variable (without explicitly specifying its type), the variable will be of `Long` type. For example,

```
val distance = 10000000000  // distance variable of type Long
```

Similarly, you can use capital letter L to specify that the variable is of type `Long`. For example,

```
val distance = 100L    // distance value of type Long
```

5. Double

- The `Double` type is a double-precision 64-bit floating point.
- Example:

```
fun main(args : Array<String>) {

    // distance is of type Double
    val distance = 999.5
```

```
    println("$distance")
}
```

When you run the program, the output will be:

```
999.5
```

Float

- The `Float` data type is a single-precision 32-bit floating point. Learn more about single precision and double precision floating point if you are interested.
- Example:

```
fun main(args : Array<String>) {

    // distance is of type Float
    val distance = 19.5F
    println("$distance")
}
```

When you run the program, the output will be:

```
19.5
```

Notice that, we have used `19.5F` instead of `19.5` in the above program. It is because `19.5` is a `Double` literal, and you cannot assign `Double` value to a variable of type `Float`.

To tell compiler to treat `19.5` as `Float`, you need to use F at the end.

If you are not sure what number value a variable will be assigned in the program, you can specify it as `Number` type. This allows you to assign both integer and floating-point value to the variable (one at a time). For example:

```
fun main(args : Array<String>) {

    var test: Number = 12.2
```

```
    println("$test")

    test = 12
    // Int smart cast from Number
    println("$test")

    test = 120L
    // Long smart cast from Number
    println("$test")
}
```

When you run the program, the output will be:

```
12.2
12
120
```

To learn more, visit: *Kotlin smart casts*

Char

To represent a character in Kotlin, Char types are used.

Unlike Java, Char types cannot be treated as numbers. Visit this page to learn more about Java char Type.

```
fun main(args : Array<String>) {

    val letter: Char
    letter = 'k'
    println("$letter")
}
```

When you run the program, the output will be:

```
k
```

In Java, you can do something like:

```
char letter = 65;
```

However, the following code gives error in Kotlin.

```
var letter: Char = 65  // Error
```

Boolean

- The `Boolean` data type has two possible values, either `true` or `false`.
- Example:

```
fun main(args : Array<String>) {

    val flag = true
    println("$flag")
}
```

Booleans are used in decision making statements (will be discussed in later chapter).

Kotlin Arrays

An array is a container that holds data (values) of one single type. For example, you can create an array that can hold 100 values of `Int` type.

In Kotlin, arrays are represented by the `Array` class. The class has `get` and `set` functions, `size` property, and a few other useful member functions.

To learn in detail about arrays, visit: *Kotlin Arrays*

Kotlin Strings

In Kotlin, strings are represented by the `String` class. The string literals such as "this is a string" is implemented as an instance of this class.