LECTURE 2.1: KOTLIN BASIC OUTPUT

BY AHMAD BARGHASH

In this lecture, you will learn to display output to the screen, in Kotlin.

LECTURE 3: VARIABLES AND DATA TYPES IN KOTLIN

BY AHMAD BARGHASH

You will learn about Kotlin comments, and why and how to use them. In addition to, you will learn to write Hello World program in Kotlin. And you will learn about variables, how to create them, and basic data types that Kotlin supports for creating variables.

PROGRAM ENTRY POINT

- An entry point of a Kotlin application is the main function.

```
fun main() {
    println("Hello world!")
}
```

The output is:

```
Hello World
```

THE MAIN FUNCTION

- A function is a named block of code that performs some well-defined set of operations.
- Every Koltlin program **must** contain a function named main. This is where execution begins when you run the program.
- Complex programs contain many functions, but for now most of your programs will only contain the main function.

CODE DISCUSSION

- The fun keyword designates a function followed by the name of the function right here.
- This is followed by a set of parenthesis where the arguments go.
- This function doesn't have any arguments.
- Next, we have the curly braces and they frame the function body.
- Inside a function, we can make calls. call is for println, which prints one line of text.
- We are printing the string, "Hello World". You do not have to specify a return type because we're not returning anything in this function, and you may notice that there no punctuation at the end of your statements.
- No semicolons.

PRINT() VS PRINTLN()

- Print():
  - Print output on the screen or standard output.
  - Does not add a new break line.
  - You can print all type of variables (like Int, Float, Double, Long, Char, and String) value to print the console.

- Println():
  - Does the same job of the print() function and output screen cursor goes to the beginning of the next line.
  - Println = print + newline.
  - You can print all type of variables (like Int, Float, Double, Long, Char, and String) value to print the console

EXAMPLE 1: PRINT() VS PRINTLN()

- Find the difference between the following code:

```
fun main() {
    println("Hello, World!")
    helloKotlin()
}

fun helloKotlin(){
    println("Hello, Kotlin!")
}
```

The output is:

```
Hello, World!
Hello, World!
```

```
fun main() {
    print("Hello, World!")
    helloKotlin()
}

fun helloKotlin(){
    println("Hello, Kotlin!")
}
```

The output is:

```
Hello, World! Hello, World!
```

EXAMPLE 2: PRINT() AND PRINTLN()

```
fun main() {
    println("1. println ")
    println("2. println ")
    print("1. print ")
    print("2. print")
}
```

The output is:

```
1. println
2. println
1. print 2. print
```

COMMENTS IN KOTLIN

- In programming, comments are portion of the program intended for you and your fellow programmers to understand the code. They are completely ignored by the compiler.
- there are two types of comments in Kotlin:

1. /* ... */

   This is a multiline comment that can span over multiple lines. The compiler ignores everything from /* to */.

2. // ....

   The compiler ignores everything from // to the end of the line.

COMMENTS IN KOTLIN

- Kotlin supports single-line (or end-of-line) and multi-line (block) comments.

```
// This is an end-of-line comment

/* This is a block comment
   on multiple lines. */
```

- Block comments in Kotlin can be nested.

```
/* The comment starts here
/* contains a nested comment */
and ends here. */
```

VARIABLES

```
fun main() {
    println("Hello, World! My name is Lina.")
}
```

The output is:

```
Hello, World! My name is Lina.
```

- This program is inflexible because it always print my name is Lina.
- If you want to replace the name dynamically, you should use the **variables**.

VARIABLES

- The variable is container that that holds data values.
- A variable refers to **a memory location** that stores some data. It has a name and an associated type.
- The type of a variable defines the range of values that the variable can hold, and the operations that can be done on those values.
- The variable can be:
  - Name that the user typed into the application.
  - Result of mathematical calculations.
  - Results of database query that we want to store and then use later.
  - Value that you want to use in multiple different places.
- You can declare a variable in Kotlin using var and val keywords.

VARIABLES

- There are two types of variables in Kotlin; **changeable** and **unchangeable**.

1- Changeable:

- With **var,** you can assign a value, and then you can change it.
- **Var** is mutable.

2- Unchangeable:

- With **val**, you can assign a value once. If you try to assign something again, you get an error.
- **Val** is immutable.

- You **can not change the type** of a variable in once its type has been determined.

VARIABLES IN KOTLIN- EXAMPLES

- Example 1

```
val number = 1
number = 2
error: val cannot be reassigned
number = 2
^
```

- Example 2

```
var number = 1
number = 2
```

Correct because var allows the changing in variable value.

- Example 3

```
var number = 1
number = 2
number = "5"
error: type mismatch: inferred type is String but Int was expected
number = "5"
     ^
```

PRINT VARIABLES

- To print a variable inside the print statement, we need to use the dollar symbol($) followed by the var/val name inside a double quoted string literal.
- To print the result of an expression we use `${ //expression goes here }`

VARIABLES IN KOTLIN

```
fun main() {
    val userName = "Florian"
    var age = 28

    println("Hello, world! My name is $userName. I am $age years old. In 2 years, I'll be ${age + 2}.")
}
```

The output is:

```
Hello, world! My name is Florian. I am 28 years old. In 2 years, I'll be
30.
```

USING THE + OPERATOR

- The simplest way of concatenating *Strings* in Kotlin is to use the + operator. As a result, we get a **new *String* object composed of *Strings* on the left and the right side of the operator:**

```
fun main()
{
    val a = "Hello"
    val b = "kotlin"
    val c = a + " " + b
    println("Hello kotlin " + c)
}
```

The output is:

```
Hello kotlin Hello kotlin
```

KOTLIN IDENTIFIERS

- Identifiers are the name given to variables, classes, methods etc.
- Here are the rules and conventions for naming a variable (identifier) in Kotlin:
  1. An identifier starts with a **letter** or underscore or have it within its name.
  2. **Whitespaces** are not allowed.
  3. An identifier cannot contain symbols such as @, $, # etc.
  4. Identifiers are **case sensitive**.
  5. When creating variables, choose a name that **makes sense**. For example, score, number, level makes more sense than variable name such as s, n, and l although they valid.
  6. If you choose a variable name having more than one word, it is suggested to use all **lowercase letters** for the first word and **capitalize** the first letter of each subsequent word. For example, speedLimit.
  7. Variable name should not be a Keyword.

KOTLIN IDENTIFIERS

- Valid or invalid?
  - Score .....Valid
  - Level .....Valid
  - Class .....InValid
  - 2number .....Invalid
  - highest Score .....InValid
  - @pple .....InValid
  - highestScore .....Valid
  - Number2 .....Valid
  - calculateTraffic .....Valid

KOTLIN KEYWORDS

- Keywords are predefined, reserved words used in Kotlin programming that have special meanings to the compiler. These words cannot be used as an identifier.

Kotlin keywords List

as	break	class	continue	do	else
false	for	fun	if	in	interface
is	null	object	package	return	super
this	throw	true	try	typealias	typeof
val	var	when	while		

DATA TYPES

1. Numbers
   - Integers
     - Byte - 8 bit
     - Short- 16 bit
     - Int - 32 bit
     - Long - 64 bit
   - Floating Point Numbers
     - Float - 32 bit
     - Double – 64 bit
2. Booleans
3. Characters
4. String

In case your program requires the direct definition of data types

NUMBERS

- Numeric types in Kotlin are similar to Java. They can be categorized into **integer** and **floating-point** types.
- Kotlin supports **underscores** in numbers. So you can specify long constants in a format that makes sense to you.
- Kotlin supports the declaration of the hexadecimal and binary values.
- Integer is a whole number (not a fractional number) that **can be positive, negative**. Ex: 1, -2, 0, 125, -852.
- Floating-point is a real number (that is, a number that can contain a fractional part). It can be **positive or negative.** Ex: 0.3, 125.500, -500.00, 12.5, 1.00, -400.625.

INTEGER AND FLOATING POINTS NUMBER

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

Data types	bits	Min value	Max value
float	32	1.401298464324817070e-45	3.402823466385288598e+38
double	64	4.940656458412465441e-324	1.797693134862315708e+308

NUMBERS

```kotlin
// Kotlin Numeric Types Examples
val myByte: Byte = 10
val myShort: Short = 125

val myInt = 1000
val myLong = 1000L // The suffix 'L' is used to specify a long value

val myFloat = 126.78f   // The suffix 'f' or 'F' represents a Float
val myDouble = 325.49


val myHexa = 0x0A0F  // Hexadecimal values are prefixed with '0x' or '0X'
val myBinary = 0b1010  // Binary values are prefixed with '0b' or '0B'
```

```kotlin
val oneMillion = 1_000_000
val SSN = 999_999_9999L
val hexByte = 0xFF_EC_DE_5E
val bytes= 0b11001110_01011001_11101101_1000010101
```

BOOLEANS

- The type Boolean is used to represent logical values. It can have two possible values true and false.

```
val myBoolean = true
val anotherBoolean = false
```

CHARACTERS

- Characters are represented using the type Char. Unlike Java, Char types cannot be treated as numbers. They are declared using single quotes.

```
val letterChar  = 'A'
val digitChar   = '9'
val sepcialChar = '@'
```

STRINGS

- String in kotlin is pretty much like strings in any other language.
- Strings are represented using the String class. They are immutable, that means you cannot modify a String by changing some of its elements.

```
var myStr = "Hello, Kotlin"
```

- You can:

1. Concatenate strings using plus.

```
"Hello" + "fish" + "!"
```

2. Build strings by combining them with values. The dollar variable name is replaced by text representing its value.

```
val numberOfFish = 5
println("I have $numberOfFish fish")
```

The result is: `I have 5 fish`

STRINGS

- You can add the numeric result of an arithmetic operation if needed

```
val numberOfFish = 5
println("Print ${ numberOfFish + 5 } fish")
```

The output is: **Print 10 fish**

- You can access the character at a particular index in a String using **str[index].** The index starts from zero

```
var name = "John"
var firstCharInName = name[0]  // 'J'
var lastCharInName = name[name.length - 1]  // 'n'
```

- The length property is used to get the length of a String.
- In println please use braces

```
println("first char is ${name[0]}")
```

STRINGS

- Like other languages, Kotlin has a Boolean data type and operators.

```
val fish = "fish"
val plant = "plant"
println(fish == plant)
println(fish == fish)
println(fish != plant)
println(fish >= plant)
println(fish <= plant)
println(fish > plant)
println(fish < plant)
```

```
The output is:
false
true
true
false
true
false
true
```

TYPE INFERENCE

- **Did you notice** one thing about the variable declarations in the previous section? We didn't specify the type of variables.
- Although Kotlin is a statically typed language, It doesn't require you to explicitly specify the type of every variable you declare. It can infer the type of a variable from the initializer expression.
- **But**, If you want to explicitly specify the type of a variable, you can do that.
- **Note that:** the type declaration becomes **mandatory** if you're not initializing the variable at the time of declaration (see the example in the next slide).

TYPE INFERENCE

- Example 1

```
val greeting = "Hello, World"  // type inferred as `String`
val year = 2018                // type inferred as `Int`
```

- Example 2

```
// Explicitly defining the type of variables
val greeting: String = "Hello, World"
val year: Int = 2018
```

- Example 3

```
var language   // Error: The variable must either have a Type annotation or be initialized
language =  "French"
```

VARIABLES IN KOTLIN

- Read-only local variables are defined using the keyword val. They can be assigned a value only once.

```
val a: Int = 1  // immediate assignment
val b = 2   // `Int` type is inferred
val c: Int  // Type required when no initializer is provided
c = 3       // deferred assignment
```

- Variables that can be reassigned use the var keyword:

```
var x = 5 // `Int` type is inferred
x = x + 1
```

- Top-level variables:

```
val PI = 3.14
var x = 0

fun incrementX() {
    x = x + 1
}
```