INTRODUCTION TO KOTLIN

Kotlin is a statically typed, general-purpose programming language developed by JetBrains, that has built world-class IDEs like IntelliJ IDEA, PhpStorm, Appcode, etc. It was first introduced by JetBrains in 2011 and a new language for the JVM. Kotlin is object-oriented language, and a "better language" than Java, but still be fully interoperable with Java code.

Kotlin is sponsored by Google, announced as one of the official languages for Android Development in 2017.

Key Features of Kotlin:

- Statically typed – Statically typed is a programming language characteristic that means the type of every variable and expression is known at compile time. Although it is statically typed language, it does not require you to explicitly specify the type of every variable you declare.
- Data Classes– In Kotlin, there are Data Classes which lead to auto-generation of boilerplate like equals, hashCode, toString, getters/setters and much more.

Consider the following example –

```
/*     Java Code     */
class Book {
   private String title;
   private Author author;
   public String getTitle()
   {
      return title;
   }
   public void setTitle(String title)
   {
      this.title = title;
   }
```

```
    public Author getAuthor()
    {
        return author;
    }
    public void setAuthor(Author author)
    {
        this.author = author;
    }
}
```

But in Kotlin only one line used to define the above class –

```
/* Kotlin Code */
data class Book(var title:String, var author:Author)
```

- Concise – It drastically reduces the extra code written in other object-oriented programming languages.
- Safe – It provides the safety from most annoying and irritating NullPointerExceptions by supporting nullability as part of its system.

Every variable in Kotlin is non-null by default.

```
String s = "Hello Geeks"    // Non-null
```

If we try to assign s null value then it gives compile time error.So,

```
s = null                    // compile time error
```

To assign null value to any string it should be declared as nullable.

```
string nullableStr? = null  // compiles successfully
```

length() function also disabled on the nullable strings.

- Interoperable with Java – Kotlin runs on Java Virtual Machine(JVM) so it is totally interoperable with java. We can easily access use java code from kotlin and kotlin code from java.
- Functional and Object Oriented Capabilities – Kotlin has rich set of many useful methods which includes higher-order functions, lambda expressions, operator overloading, lazy evaluation, operator overloading and much more.

Higher order function is a function which accepts function as a parameter or returns a function or can do both.

- Smart Cast – It explicitly typecasts the immutable values and inserts the value in its safe cast automatically.

If we try to access a nullable type of String (String? = “BYE”) without safe cast it will generate a compile error.

```
fun main(args: Array){
   var string: String? = "BYE"
      print(string.length)       // compile time error
   }
}
fun main(args: Array){
   var string: String? = "BYE"
   if(string != null) {              // smart cast
      print(string.length)
   }
}
```

- Compilation time – It has higher performance and fast compilation time.
- Tool- Friendly – It has excellent tooling support. Any of the Java IDEs – IntelliJ IDEA, Eclipse and Android Studio can be used for Kotlin. We can also be run Kotlin program from command line.

Advantages of Kotlin language:

- Easy to learn – Basic is almost similar to java.If anybody worked in java then easily understand in no time.
- Kotlin is multi-platform – Kotlin is supported by all IDEs of java so you can write your program and execute them on any machine which supports JVM.
- It's much safer than Java.
- It allows using the Java frameworks and libraries in your new Kotlin projects by using advanced frameworks without any need to change the whole project in Java.

Applications of Kotlin language:

- You can use Kotlin to build Android Application.
- Kotlin can also compile to JavaScript, and making it available for the frontend.
- It is also designed to work well for web development and server-side development.

Kotlin Hello World Program

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

To compile: kotlinc firstapp.kt

To run: kotlin firstapp.kt

Hello, World!

KOTLIN DATA TYPES

The most fundamental data type in Kotlin is the Primitive data type and all others are reference types like array and string. Java needs to use wrappers (java.lang.Integer) for primitive data types to behave like objects but Kotlin already has all data types as objects.

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)

KOTLIN VARIABLES

In Kotlin, every variable should be declared before it's used. Without declaring a variable, an attempt to use the variable gives a syntax error. Declaration of the variable type also decides the kind of data you are allowed to store in the memory location. In case of local variables, the type of variable can be inferred from the initialized value.

```
var rollno = 55

var name = "Praveen"

println(rollno)

println(name)
```

Above we have the local variable *rollno* whose value is 55 and it's type is Integer because the literal type is Int and another variable is *name* whose type is String. In Kotlin, variables are declared using two types –

1. Immutable using val keyword
2. Mutable using var keyword

Immutable Variables –

Immutable is also called read-only variables. Hence, we can not change the value of the variable declared using *val* keyword.

```
val myName = "Gaurav"

myName = "Praveen"    // compile time error

// It gives error Kotlin Val cannot be reassigned
```

Note: Immutable variable is not a constant because it can be initialized with the value of a variable. It means the value of immutable variable doesn't need to be known at compile-time, and if it is declared inside a construct that is called repeatedly, it can take on a different value on each function call.

```
var myBirthDate = "02/12/1993"

val myNewBirthDate = myBirthDate

println(myNewBirthDate)
```

Mutable Variables –

In **Mutable** variable we can change the value of the variable.

```
var myAge = 25

myAge = 26          // compiles successfully

println("My new Age is ${myAge}")
```

Output:

```
My new Age is 26
```

Scope of a variable – A variable exists only inside the block of code({…….}) where it has been declared. You can not access the variable outside the loop. Same variable can be declared inside the nested loop – so if a function contains an argument x and we declare a new variable x inside the same loop, then x inside the loop is different than the argument.

KOTLIN OPERATORS

Operators are the special symbols that perform different operation on operands. For example + and – are operators that perform addition and subtraction respectively. Like Java, Kotlin contains different kinds of operators.

- Arithmetic operator
- Relation operator
- Assignment operator
- Unary operator
- Logical operator
- Bitwise operator

Arithmetic Operators –

Operators	Meaning	Expression	Translate to
+	Addition	a + b	a.plus(b)
–	Subtraction	a – b	a.minus(b)
*	Multiplication	a * b	a.times(b)
/	Division	a / b	a.div(b)
%	Modulus	a % b	a.rem(b)

```
fun main(args: Array<String>)
{
        var a = 20 var b = 4 println("a + b = " + (a + b))
                println("a - b = " + (a - b))
                println("a * b = " + (a.times(b)))
                println("a / b = " + (a / b))
```

```
                println("a % b = " + (a.rem(b)))
}
```

a + b = 24

a - b = 16

a * b = 80

a / b = 5

a % b = 0

Relational Operators –

Operators	Meaning	Expression	Translate to
>	greater than	a > b	a.compareTo(b) > 0
<	less than	a < b	a.compareTo(b) < 0
>=	greater than or equal to	a >= b	a.compareTo(b) >= 0
<=	less than or equal to	a <= b	a.compareTo(b) <= 0
==	is equal to	a == b	a?.equals(b) ?: (b === null)
!=	not equal to	a != b	!(a?.equals(b) ?: (b === null)) > 0

```
fun main(args: Array<String>)
{
    var c = 30
    var d = 40
```

```
    println("c > d = "+(c>d))
    println("c < d = "+(c.compareTo(d) < 0))
    println("c >= d = "+(c>=d))
    println("c <= d = "+(c.compareTo(d) <= 0))
    println("c == d = "+(c==d))
    println("c != d = "+(!(c?.equals(d) ?: (d === null))))
}
```

Output:

c > d = false

c < d = true

c >= d = false

c <= d = true

c == d = false

c != d = true

Assignment Operators –

Operators	Expression	Translate to
+=	a = a + b	a.plusAssign(b) > 0
-=	a = a – b	a.minusAssign(b) < 0
*=	a = a * b	a.timesAssign(b)>= 0
/=	a = a / b	a.divAssign(b) <= 0
%=	a = a % b	a.remAssign(b)

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

```
 var a = 10
    var b = 5
    a+=b
    println(a)
    a-=b
    println(a)
    a*=b
    println(a)
    a/=b
    println(a)
    a%=b
    println(a)

}
```

Output:

15

10

50

10

0

Unary Operators –

Operators	Expression	Translate to
++	++a or a++	a.inc()
—	–a or a–	a.dec()

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

```
    var e=10
    var flag = true
    println("First print then increment: "+ e++)
    println("First increment then print: "+ ++e)
    println("First print then decrement: "+ e--)
    println("First decrement then print: "+ --e)
}
```

Output:

First print then increment: 10

First increment then print: 12

First print then decrement: 12

First decrement then print: 10

Logical Operators –

Operators	Meaning	Expression
&&	return true if all expressions are true	(a>b) && (a>c)
||	return true if any of expression is true	(a>b) || (a>c)
!	return complement of the expression	a.not()

```
fun main(args : Array<String>){
    var x = 100
    var y = 25
    var z = 10
    var result = false
    if(x > y && x > z)
     println(x)
```

```
    if(x < y || x > z)
     println(y)
    if( result.not())
     println("Logical operators")
}
```

Output:

100

25

Logical operators

Bitwise Operators –

Operators	Meaning	Expression
shl	signed shift left	a.shl(b)
shr	signed shift right	a.shr(b)
ushr	unsigned shift right	a.ushr()
and	bitwise and	a.and(b)
or	bitwise or	a.or()
xor	bitwise xor	a.xor()
inv	bitwise inverse	a.inv()

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

```
    println("5 signed shift left by 1 bit: " + 5.shl(1))
    println("10 signed shift right by 2 bits: : " + 10.shr(2))
    println("12 unsigned shift right by 2 bits:  " + 12.ushr(2))
    println("36 bitwise and 22: " + 36.and(22))
    println("36 bitwise or 22: " + 36.or(22))
    println("36 bitwise xor 22: " + 36.xor(22))
    println("14 bitwise inverse is: " + 14.inv())
}
```

Output:

```
5 signed shift left by 1 bit: 10
10 signed shift right by 2 bits: : 2
12 unsigned shift right by 2 bits:  3
36 bitwise and 22: 4
36 bitwise or 22: 54
36 bitwise xor 22: 50
14 bitwise inverse is: -15
```

KOTLIN STANDARD INPUT OUTPUT

Kotlin standard I/O operations are performed to flow sequence of bytes or byte streams from input device such as Keyboard to the main memory of the system and from main memory to output device such as Monitor.

In Java, we use System.out.println(message) to print output on the screen but, in Kotlin println(message) is used to print.

Kotlin Output –

Kotlin standard output is the basic operation performed to flow byte streams from main memory to the output device. You can output any of the data types integer, float and any patterns or strings on the screen of the system.

You can use any one of the following function to display output on the screen.

print() function

println() function

Here is the Kotlin program for standard output.

```
fun main(args: Array<String>)
{
   print("Hello! ")
   println("This is Kotlin tutorial.")
}
```

Output:

Hello! This is Kotlin tutorial.

Difference between println() and print() –

print() function prints the message inside the double quotes.

println() function prints the message inside the double quotes and moves to the beginning of the next line.

kotlin program to print string:

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

```
    println("HARMAN")
    println("A Samsung company")

    print("HARMAN-  ")
    print("A Samsung company")
}
```

Output:

HARMAN

A Samsung company

HARMAN- A Samsung company

Actually print() internally calls System.out.print() and println() internally calls System.out.println().

Print literals and Variables –

```
fun sum(a: Int,b: Int) : Int{
        return a + b
}
fun main(args: Array<String>){
        var a = 10
        var b = 20
        var c = 30L
        var marks = 40.4

        println("Sum of {$a} and {$b} is : ${sum(a,b)}")
        println("Long value is: $c")
        println("marks")
        println("$marks")
}
```

Output:

Sum of {10} and {20} is : 30

Long value is: 30

marks

40.4

Kotlin Input –

Kotlin standard input is the basic operation performed to flow byte streams from input device such as Keyboard to the main memory of the system.

You can take input from user with the help of the following function:

- readline() method
- Scanner class

Take input from user using readline() method –

```
fun main(args : Array<String>) {
   print("Enter text: ")
   var input = readLine()
   print("You entered: $input")
}
```

Output:

Enter text: Hello, employee!

You entered: Hello, employee!

Take input from user using Scanner class –

If you are taking input from user other than String data type, you need to use Scanner class. To use Scanner first of all you have to import the Scanner on the top of the program.

```
import java.util.Scanner;
```

Create an object for the scanner class and use it to take input from the user. In the below program we will show you how to take input of integer, float and boolean data type.

```
import java.util.Scanner
 fun main(args: Array<String>) {
```

```
    // create an object for scanner class
    val number1 = Scanner(System.`in`)
    print("Enter an integer: ")
    // nextInt() method is used to take
    // next integer value and store in enteredNumber1 variable
    var enteredNumber1:Int = number1.nextInt()
    println("You entered: $enteredNumber1")

    val number2 = Scanner(System.`in`)
    print("Enter a float value: ")
    // nextFloat() method is used to take next
    // Float value and store in enteredNumber2 variable
    var enteredNumber2:Float = number2.nextFloat()
    println("You entered: $enteredNumber2")

    val booleanValue = Scanner(System.`in`)
    print("Enter a boolean: ")
    // nextBoolean() method is used to take
    // next boolean value and store in enteredBoolean variable
    var enteredBoolean:Boolean = booleanValue.nextBoolean()
    println("You entered: $enteredBoolean")
}
```

Output:

```
Enter an integer: 123
You entered: 123
Enter a float value:  40.45
You entered: 40.45
Enter a boolean:  true
You entered: true
```

Take input from user without using the Scanner class –

Here, we will use readline() to take input from the user and no need to import Scanner class.

readline()!! take the input as a string and followed by (!!) to ensure that the input value is not null.

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


    print("Enter an Integer value: ")
    val string1 = readLine()!!


    // .toInt() function converts the string into Integer
    var integerValue: Int = string1.toInt()
    println("You entered: $integerValue")


    print("Enter a double value: ")
    val string2= readLine()!!


    // .toDouble() function converts the string into Double
    var doubleValue: Double = string2.toDouble()
    println("You entered: $doubleValue")
}
```

Output:

```
Enter an Integer value: 123
You entered: 123
Enter a double value:  22.22222
You entered: 22.22222
```