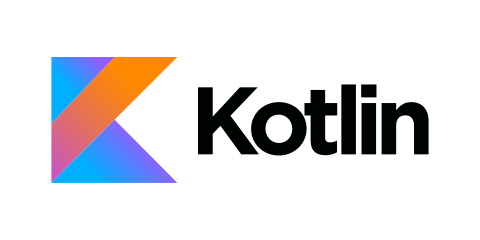
**502-02 : Advance Mobile Computing**

**Unit-1: Introduction to Kotlin**

**1.1 Concepts of Kotlin and its introduction.**

Kotlin is a statically-typed, general-purpose programming language. It is widely used to develop android applications.

**What is Kotlin**

**Kotlin** is a general-purpose, statically typed, and open-source programming language. It runs on JVM and can be used anywhere Java is used today. It can be used to develop Android apps, server-side apps and much more.

**History of Kotlin**

**Kotlin** was developed by **JetBrains** team. A project was started in 2010 to develop the language and officially, first released in February 2016. Kotlin was developed under the Apache 2.0 license.

Features of Kotlin

* **Concise:**Kotlin reduces writing the extra codes. This makes Kotlin more concise.
* **Null safety:**Kotlin is null safety language. Kotlin aimed to eliminate the NullPointerException (null reference) from the code.
* **Interoperable:**Kotlin easily calls the Java code in a natural way as well as Kotlin code can be used by Java.
* **Smart cast:**It explicitly typecasts the immutable values and inserts the value in its safe cast automatically.
* **Compilation Time:**It has better performance and fast compilation time.
* **Tool-friendly:**Kotlin programs are build using the command line as well as any of Java IDE.
* **Extension function:**Kotlin supports extension functions and extension properties which means it helps to extend the functionality of classes without touching their code.

**Kotlin Environment Setup (Command line)**

**Prerequisite**

Since Kotlin runs on JVM, it is necessary to install JDK and setup the JDK and JRE path in local system environment variable.

To setup Kotlin for command line, you have to pre install JDK 1.6+ or above. To install JDK and set path of JDK and JRE refer link Set Path in Java .

Setup Kotlin for Command Line

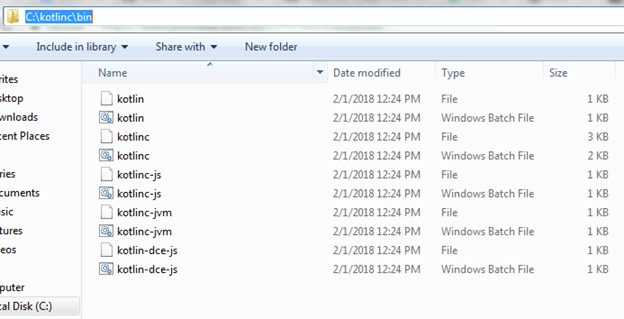
To setup Kotlin for command line, we need to go through following steps:

1. Download the Kotlin Compiler from GitHub Releases [**https://github.com/JetBrains/kotlin/releases/tag/v1.2.21**](https://github.com/JetBrains/kotlin/releases/tag/v1.2.21) .

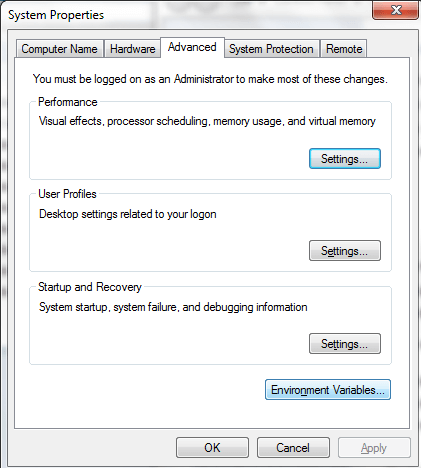


2. Extract downloaded zip in any of system location (in my case it is in C drive).

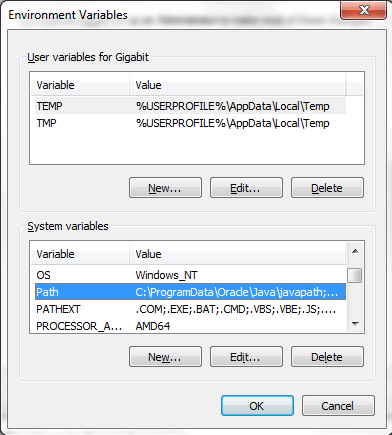
3. Copy the path up to bin directory of kotlinc.



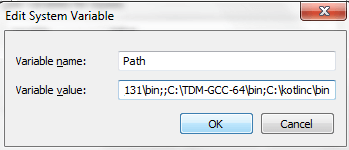
4. Open Computer properties and click Environment variables.



5. Click on edit path



6. Past the path of kotlinc bin directory in variable value.



**Kotlin Hello World Program in Command line.**

To write Kotlin program, we can use any text editor like: Notepad++. Put the following code into any text file and save.

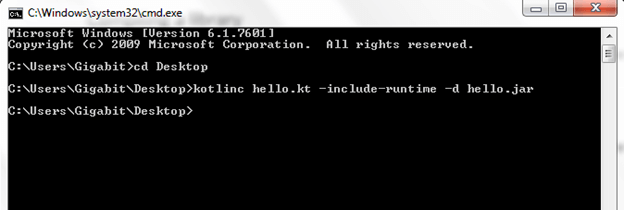
1. **fun** main(args: Array<String>){
2. println("Hello World!")
3. }

Save the file with name hello.kt, **.kt** extension is used for Kotlin file.

**Compile Kotlin File**

Open command prompt and go to directory location where file is stored. Compile **hello.kt** file with following command.

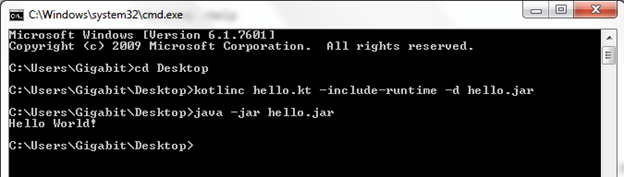
1. kotlinc hello.kt -include-runtime -d hello.jar



**Run Kotlin File**

To run the Kotlin .jar (hello.jar) file run the following command.

1. java -jar hello.jar



**Kotlin First Program Concept**

Let's understand the concepts and keywords of Kotlin program 'Hello World.kt'.

1. **fun** main(args: Array<String>) {
2. println("Hello World!")
3. }

**1.** The first line of program defines a function called **main()**. In Kotlin, function is a group of statements that performs a group of tasks. Functions start with a keyword **fun** followed by function name (main in this case).

The main () function takes an array of string **(Array<String>)** as a parameter and returns Unit. Unit is used to indicate the function and does not return any value (void as in Java). Declaring Unit is an optional, we do not declare it explicitly.

1. **fun** main(args: Array<String>): Unit {
2. //
3. }

The main() function is the entry point of the program, it is called first when Kotlin program starts execution.

**2.** The second line used to print a String "Hello World!". To print standard output we use wrapper println() over standard Java library functions (System.out.println()).

1. println("Hello World!")

Note: Semicolons are optional in Kotlin.

1.2 Downloading IntelliJ and its settings.

**Kotlin Environment Setup (IDE)**

**Install JDK and Setup JDK path**

Since, Kotlin runs on JVM, it is necessary to install JDK and setup the JDK and JRE path in local system environment variable.

**Install IDE for Kotlin**

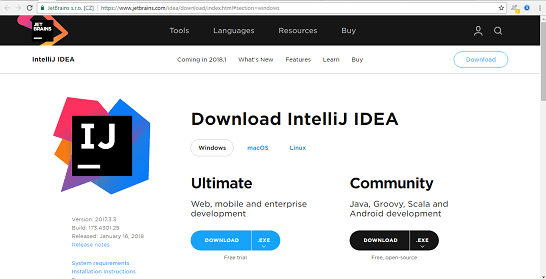
There are various Java IDE available which supports Kotlin project development. We can choose these IDE according to our compatibility. The download links of these IDE's are given below.

|  |  |
| --- | --- |
| **IDE Name** | **Download links** |
| IntelliJ IDEA | <https://www.jetbrains.com/idea/download/> |
| Android Studio | <https://developer.android.com/studio/preview/index.html> |
| Eclipse | <https://www.eclipse.org/downloads/> |

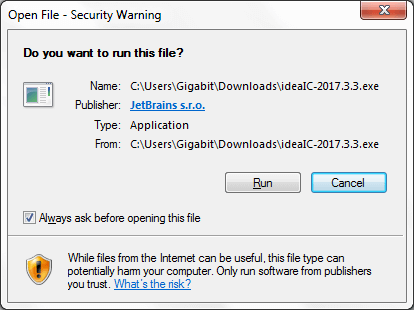
In this tutorial, we are going to use IntelliJ IDEA for our Kotlin program development.

Steps to Setup IntelliJ IDEA

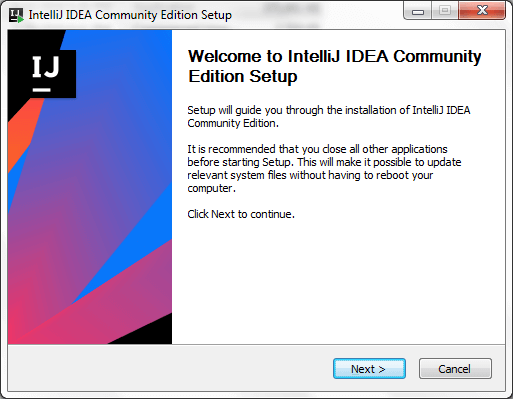
1. Download IntelliJ IDEA.



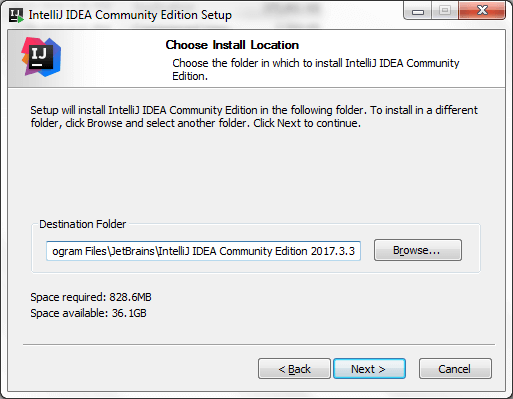
2. Run the downloaded setup.



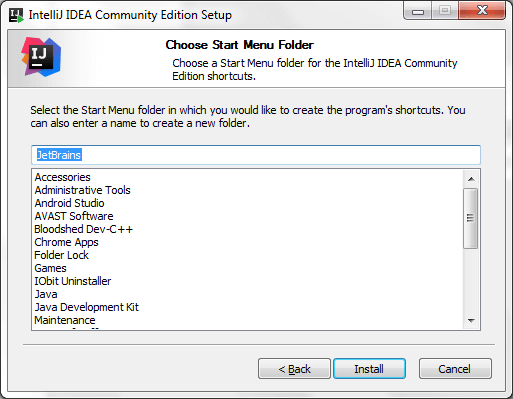
3. Click next to continue.



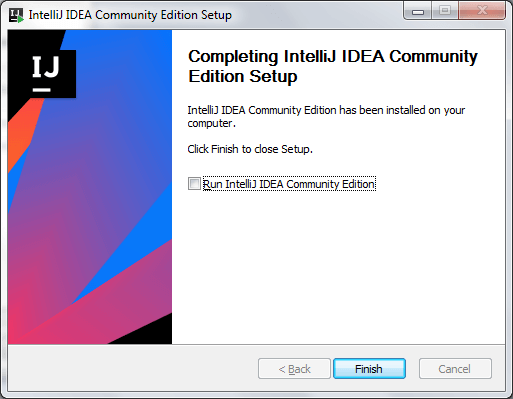
4. Choose installation location.



5. Choose start menu folder and click Install.



6. Click Finish to complete Installation.

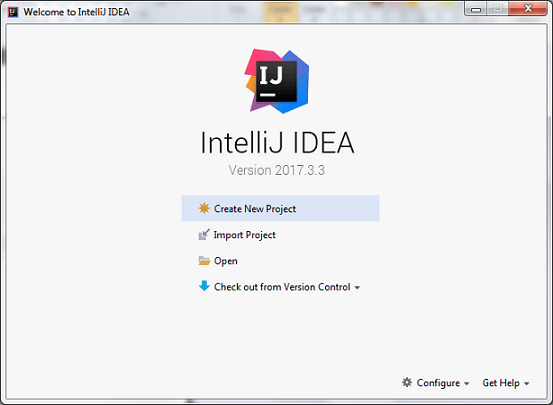


**Kotlin First Program Printing 'HelloWorld'**

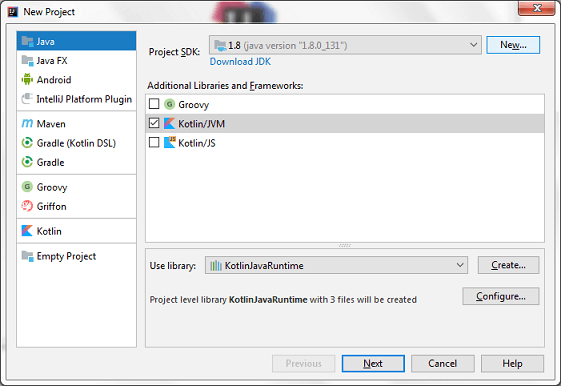
Let's create a Kotlin first example using IntelliJ IDEA IDE.

**Steps to Create First Example**

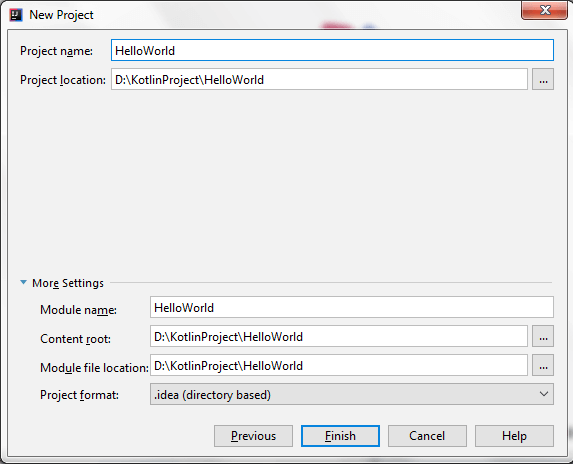
**1.** Open IntelliJ IDEA and click on Create New Project'.



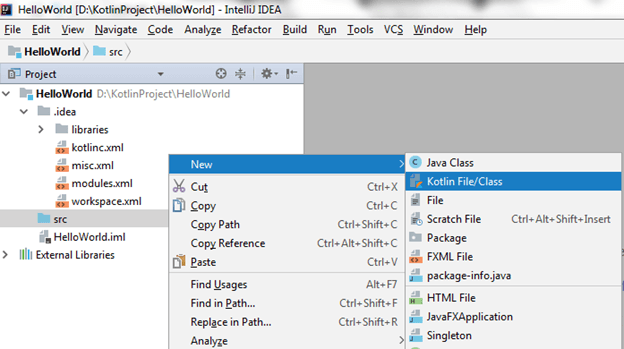
**2.** Select Java option, provide project SDK path and mark check on Kotlin/JVM frameworks.



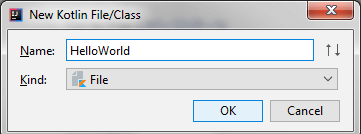
**3.** Provide the project details in new frame and click 'Finish'.



**4.** Create a new Kotlin file to run Kotlin first example. Go to src ->New->Kotlin File/Class.



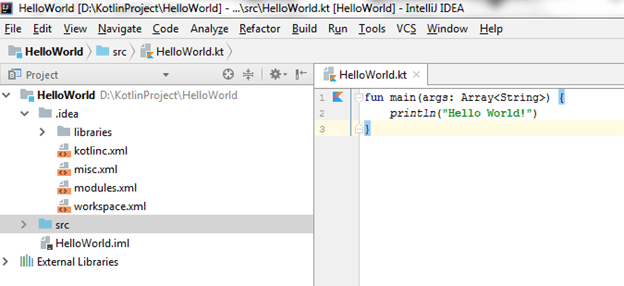
**5.** Enter the file name 'HelloWorld' and click 'OK'.



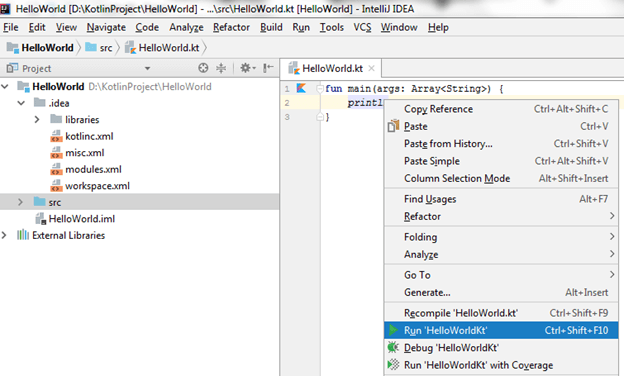
**6.** Write the following code in 'HelloWorld.kt' file. Kotlin files and classes are saved with ".kt" extension.

1. **fun** main(args: Array<String>) {
2. println("Hello World!")
3. }

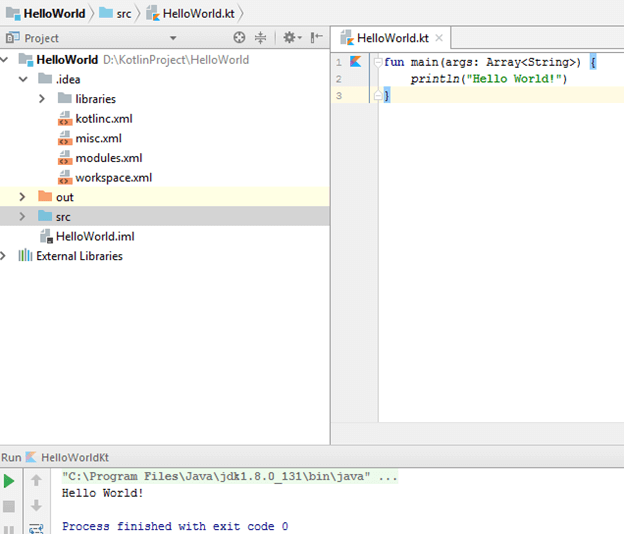
We will discuss the detail of this code later in upcoming tutorial.



**7.** Now we can run this program by right clicking on file and select Run option.



**8.** Finally, we got the output of program on console, displaying 'HelloWorld' message.



1.3 Variables:

1.3.1 val vs. var, Byte, Short, Int, Long, Float, Double, Boolean, and Char.

**Kotlin Variable**

Variable refers to a memory location. It is used to store data. The data of variable can be changed and reused depending on condition or on information passed to the program.

**Variable Declaration**

Kotlin variable is declared using keyword **var** and **val**.

1. var language ="Java"
2. val salary = 30000

The difference between var and val is specified later on this page.

Here, variable language is String type and variable salary is Int type. We don't require specifying the type of variable explicitly. Kotlin complier knows this by initilizer expression ("Java" is a String and 30000 is an Int value). This is called type inference in programming.

We can also explicitly specify the type of variable while declaring it.

1. var language: String ="Java"
2. val salary: Int = 30000

It is not necessary to initialize variable at the time of its declaration. Variable can be initialized later on when the program is executed.

1. var language: String
2. ... ... ...
3. language = "Java"
4. val salary: Int
5. ... ... ...
6. salary = 30000

Difference between var and val

* **var** (Mutable variable): We can change the value of variable declared using **var** keyword later in the program.
* **val** (Immutable variable): We cannot change the value of variable which is declared using **val** keyword.

**Example**

1. var salary = 30000
2. salary = 40000 //execute

Here, the value of variable salary can be changed (from 30000 to 40000) because variable salary is declared using **var** keyword.

1. val language = "Java"
2. language = "Kotlin" //Error

Here, we cannot re-assign the variable language from "Java" to "Kotlin" because the variable is declared using **val** keyword.

**Kotlin Data Type**

**Data type** (basic type) refers to type and size of data associated with variables and functions. Data type is used for declaration of memory location of variable which determines the features of data.

In Kotlin, everything is an object, which means we can call member function and properties on any variable.

Kotlin built in data type are categorized as following different categories:

* Number
* Character
* Boolean
* Array
* String

Number Types

Number types of data are those which hold only number type data variables. It is further categorized into different Integer and Floating point.

|  |  |  |
| --- | --- | --- |
| **Data Type** | **Bit Width (Size)** | **Data Range** |
| Byte | 8 bit | -128 to 127 |
| Short | 16 bit | -32768 to 32767 |
| Int | 32 bit | -2,147,483,648 to 2,147,483,647 |
| Long | 64 bit | -9,223,372,036,854,775,808 to +9,223,372,036,854,775,807 |
| Float | 32 bit | 1.40129846432481707e-45 to 3.40282346638528860e+38 |
| Double | 64 bit | 4.94065645841246544e-324 to 1.79769313486231570e+308 |

Character (Char) Data Type

Characters are represented using the keyword **Char**. Char types are declared using single quotes ('').

|  |  |  |
| --- | --- | --- |
| **Data Type** | **Bit Width (Size)** | **Data Range** |
| Char | 4 bit | -128 to 127 |

**Example**

1. val value1 = 'A'
2. //or
3. val  value2: Char
4. value2= 'A'

Boolean Data Types

Boolean data is represented using the type **Boolean**. It contains values either true or false.

|  |  |  |
| --- | --- | --- |
| **Data Type** | **Bit Width (Size)** | **Data Value** |
| Boolean | 1 bit | true or false |

**Example**

1. val flag = **true**

Array

Arrays in Kotlin are represented by the Array class. Arrays are created using library function arrayOf() and Array() constructor. Array has get (), set() function, size property as well as some other useful member functions.

Creating Array using library function arrayOf()

The arrayOf() function creates array of wrapper types. The item value are passed inside arrayOf() function like arrayOf(1,2,3) which creates an array[1,2,3].

The elements of array are accessed through their index values (array[index]). Array index are start from zero.

1. val id = arrayOf(1,2,3,4,5)
2. val firstId = id[0]
3. val lasted = id[id.size-1]

Creating Array using Array() constructor

Creating array using Array() constructor takes two arguments in Array() constructor:

1. First argument as a size of array, and
2. Second argument as the function, which is used to initialize and return the value of array element given its index.
3. val asc = Array(5, { i -> i \* 2 }) //asc[0,2,4,6,8]

String

String in Kotlin is represented by String class. String is immutable, which means we cannot change the elements in String.

**String declaration:**

1. val text ="Hello, JavaTpoint"

Types of String

String are categorize into two types. These are:

1. **Escaped String:**Escape String is declared within double quote (" ") and may contain escape characters like '\n', '\t', '\b' etc.

1. val text1 ="Hello, JavaTpoint"
2. //or
3. val text2 ="Hello, JavaTpoint\n"
4. //or
5. val text3 ="Hello, \nJavaTpoint"

2. **Raw String:**Row String is declared within triple quote (""" """). It provides facility to declare String in new lines and contain multiple lines. Row String cannot contain any escape character.

1. val text1 ="""
2. Welcome
3. To
4. JavaTpoint
5. """

**Kotlin Type Conversion**

Type conversion is a process in which one data type variable is converted into another data type. In Kotlin, implicit conversion of smaller data type into larger data type is not supported (as it supports in java). For example Int cannot be assigned into Long or Double.

In Java

1. **int** value1 = 10;
2. **long** value2 = value1;  //Valid code

In Kotlin

1. var value1 = 10
2. val value2: Long = value1  //Compile error, type mismatch

However in Kotlin, conversion is done by explicit in which smaller data type is converted into larger data type and vice-versa. This is done by using helper function.

1. var value1 = 10
2. val value2: Long = value1.toLong()

The list of helper functions used for numeric conversion in Kotlin is given below:

* toByte()
* toShort()
* toInt()
* toLong()
* toFloat()
* toDouble()
* toChar()

Kotlin Type Conversion Example

Let see an example to convert from Int to Long.

1. **fun** main(args : Array<String>) {
2. var value1 = 100
3. val value2: Long =value1.toLong()
4. println(value2)
5. }

We can also converse from larger data type to smaller data type.

1. **fun** main(args : Array<String>) {
2. var value1: Long = 200
3. val value2: Int =value1.toInt()
4. println(value2)
5. }

**Kotlin Operator**

**Operators** are special characters which perform operation on operands (values or variable).There are various kind of operators available in Kotlin.

* Arithmetic operator
* Relation operator
* Assignment operator
* Unary operator
* Bitwise operation
* Logical operator

**Arithmetic Operator**

Arithmetic operators are used to perform basic mathematical operations such as addition (+), subtraction (-), multiplication (\*), division (/) etc.

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Description** | **Expression** | **Translate to** |
| + | Addition | a+b | a.plus(b) |
| - | Subtraction | a-b | a.minus(b) |
| \* | Multiply | a\*b | a.times(b) |
| / | Division | a/b | a.div(b) |
| % | Modulus | a%b | a.rem(b) |

Example of Arithmetic Operator

1. **fun** main(args : Array<String>) {
2. var a=10;
3. var b=5;
4. println(a+b);
5. println(a-b);
6. println(a\*b);
7. println(a/b);
8. println(a%b);
9. }

**Output:**

15

5

50

2

0

**Relation Operator**

Relation operator shows the relation and compares between operands. Following are the different relational operators:

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Description** | **Expression** | **Translate to** |
| > | greater than | a>b | a.compateTo(b)>0 |
| < | Less than | a<b | a.compateTo(b)<0 |
| >= | greater than or equal to | a>=b | a.compateTo(b)>=0 |
| <= | less than or equal to | a<=b | a?.equals(b)?:(b===null) |
| == | is equal to | a==b | a?.equals(b)?:(b===null) |
| != | not equal to | a!=b | !(a?.equals(b)?:(b===null)) |

**Example of Relation Operator**

1. **fun** main(args : Array<String>) {
2. val a = 5
3. val b = 10
4. val max = **if** (a > b) {
5. println("a is greater than b.")
6. a
7. } **else**{
8. println("b is greater than a.")
9. b
10. }
11. println("max = $max")
12. }

Output:

b is greater than a.

max = 10

**Assignment operator**

Assignment operator **"="** is used to assign a value to another variable. The assignment of value takes from right to left.

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Description** | **Expression** | **Convert to** |
| += | add and assign | a+=b | a.plusAssign(b) |
| -= | subtract and assign | a-=b | a.minusAssign(b) |
| \*= | multiply and assign | a\*=b | a.timesAssign(b) |
| /= | divide and assign | a/=b | a.divAssign(b) |
| %= | mod and assign | a%=b | a.remAssign(b) |

**Example of Assignment operator**

1. **fun** main(args : Array<String>) {
3. var a =20;var b=5
4. a+=b
5. println("a+=b :"+ a)
6. a-=b
7. println("a-=b :"+ a)
8. a\*=b
9. println("a\*=b :"+ a)
10. a/=b
11. println("a/=b :"+ a)
12. a%=b
13. println("a%=b :"+ a)
15. }

**Output:**

a+=b :25

a-=b :20

a\*=b :100

a/=b :20

a%=b :0

**Unary Operator**

Unary operator is used with only single operand. Following are some unary operator given below.

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Description** | **Expression** | **Convert to** |
| + | unary plus | +a | a.unaryPlus() |
| - | unary minus | -a | a.unaryMinus() |
| ++ | increment by 1 | ++a | a.inc() |
| -- | decrement by 1 | --a | a.dec() |
| ! | not | !a | a.not() |

**Example of Unary Operator**

1. **fun** main(args: Array<String>){
2. var a=10
3. var b=5
4. var flag = **true**
5. println("+a :"+ +a)
6. println("-b :"+ -b)
7. println("++a :"+ ++a)
8. println("--b :"+ --b)
9. println("!flag :"+ !flag)
10. }

**Output:**

+a :10

-b :-5

++a :11

--b :4

!flag :false

**Logical Operator**

Logical operators are used to check conditions between operands. List of logical operators are given below.

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Description** | **Expression** | **Convert to** |
| && | return true if all expression are true | (a>b) && (a>c) | (a>b) and (a>c) |
| || | return true if any expression are true | (a>b) || (a>c) | (a>b) or(a>c) |
| ! | return complement of expression | !a | a.not() |

**Example of Logical Operator**

1. **fun** main(args: Array<String>){
2. var a=10
3. var b=5
4. var c=15
5. var flag = **false**
6. var result: Boolean
7. result = (a>b) && (a>c)
8. println("(a>b) && (a>c) :"+ result)
9. result = (a>b) || (a>c)
10. println("(a>b) || (a>c) :"+ result)
11. result = !flag
12. println("!flag :"+ result)
14. }

**Output:**

(a>b) && (a>c) :false

(a>b) || (a>c) :true

!flag :true

**Bitwise Operation**

In Kotlin, there is not any special bitwise operator. Bitwise operation is done using named function.

|  |  |  |
| --- | --- | --- |
| **Named Function** | **Description** | **Expression** |
| shl (bits) | signed shift left | a.shl(b) |
| shr (bits) | signed shift right | a.shr(b) |
| ushr (bits) | unsigned shift right | a.ushr(b) |
| and (bits) | bitwise and | a.and(b) |
| or (bits) | bitwise or | a.or(b) |
| xor (bits) | bitwise xor | a.xor(b) |
| inv() | bitwise inverse | a.inv() |

**Example of Bitwise Operation**

1. **fun** main(args: Array<String>){
2. var a=10
3. var b=2
5. println("a.shl(b): "+a.shl(b))
6. println("a.shr(b): "+a.shr(b))
7. println("a.ushr(b:) "+a.ushr(b))
8. println("a.and(b): "+a.and(b))
9. println("a.or(b): "+a.or(b))
10. println("a.xor(b): "+a.xor(b))
11. println("a.inv(): "+a.inv())
13. }

**Output:**

a.shl(b): 40

a.shr(b): 2

a.ushr(b:) 2

a.and(b): 2

a.or(b): 10

a.xor(b): 8

a.inv(): -11

1.3.2 String, Nullable variables.

1.4 Conditional statements: if and when. Difference between if and when.

1.4.1 ranges, types, values of function calls

1.5 Arrays and Lists:

1.5.1 create, modify, and access arrays

1.5.2 creating, modifying, and accessing lists

1.6 Loops (Iterative statements)

1.6.1 for and while loop.

1.6.2 break, continue and return