1. Introduction to Kotlin Programming language

In Google I/O 2017, Kotlin programming language was announced as an official programming language for android application development.

Kotlin is statically typed programming language for modern multi-platform applications. It is developed by JetBrains.

It's syntax is more expressive and concise than that of Java. However, Kotlin and Java is interoperable i.e. you can use both language in same application.

Major features of Kotlin Programming language

a. Concise

We are just comparing between java and kotlin language. Our job is to create a simple class that contains getters and setters.

+ Java code:

https://onlinegdb.com/QgoKtpach

+ Kotlin code:

https://onlinegdb.com/9Npshhf-_

There is a huge difference in number of lines of code between kotlin and java programming language to perform the same task. Single line in Kotlin is equivalent to entire class written in java

b. Safe

This language avoids entire classes of errors such as null pointer exceptions.

+ Java code:

https://onlinegdb.com/weDlUYggk

+ Kotlin code:

https://onlinegdb.com/1h8HRJTfW

Note: You must declare explicitly if a value may be null.

c. Interoperable

Java and Kotlin are 100% compatible. You can also use any existing library on the JVM. You can use both languages in the same project. You can convert java code into Kotlin and vice-versa.

For example, Both code has meaning.

+ Java code:

https://onlinegdb.com/XMwBiHG0-

+ Kotlin code:

https://onlinegdb.com/TTxoj7ENe

d. Tool-friendly

You can use any Java IDE to do programming in Kotlin. You can use IntelliJ, Android Studio or Eclipse etc. to develop application using Kotlin.

2. Variables Declaration

A kotlin variable is name given to memory location so that we can manipulate the data within it in a program.

Each kotlin variable has a specific data type that determines the size and layout of the variable's memory.

The name of the kotlin variables can be composed of letters, digits and the underscore character.

+ To declare a kotlin variable, either var or val keyword is used.

For example,

– Variable using **val** Keyword

```
val marks:Int = 40;
```

– Variable using **var** Keyword

```
var name:String = "Jonson"
```

Note: In Kotlin, you do not need to specify the type of the variables explicitly, if value is being initialised while declaring. Compiler knows the type of variable by the initialiser expression.

ex: var name = "Jonson"

+ Difference between var and val in kotlin

There are 2 types of variables in kotlin.

a. Mutable variable: a variable declared using var keyword can be re-assigned.

b. Immutable variable: a variable declared using val keyword can not be re-assigned.

ex:

Mutable kotlin variable

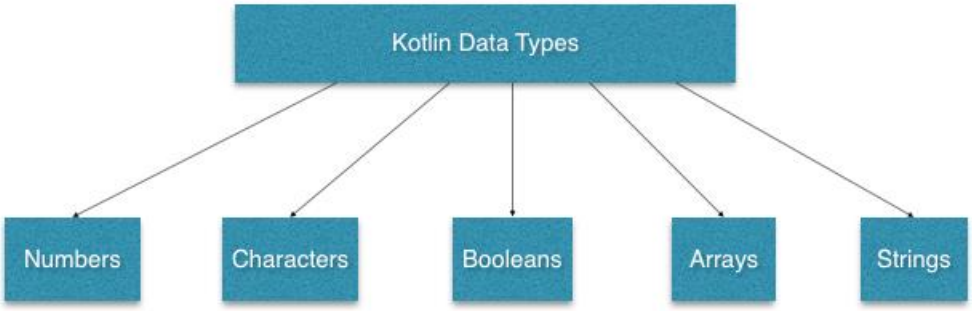
```
var x=10      // ok
x = 20        // ok
```

Immutable kotlin variable

```
val y = 20    //ok
y=30          //error
```

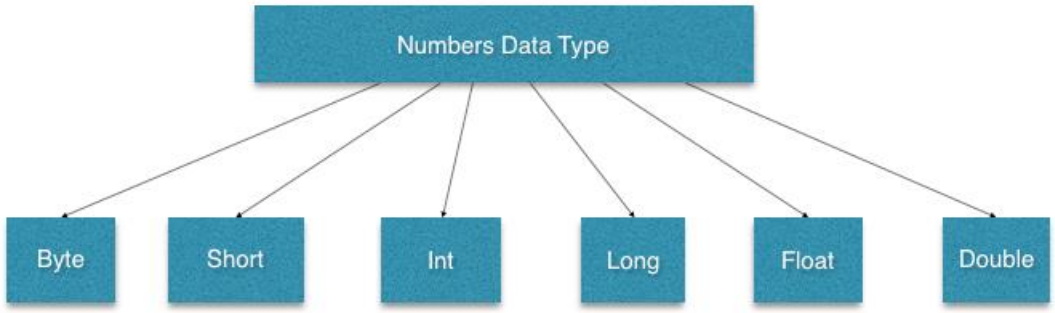
3. Kotlin Basic Data Types

The **built-in** types in Kotlin can be categorised as:



Kotlin Basic Data Types

3.1. Numbers Data Type



There are 6 built-in types representing numbers.

TYPE	BIT WIDTH
Byte	8
Short	16
Int	32
Long	64
Float	32
Double	64

+ **Byte**: this data type can have values from -128 to 127.
+ **Short**: it can have value from -32768 to 32767.
+ **Int**: it can have value from -2^31 to 2^31–1.
+ **Long**: it can have value from -2^63 to 2^63–1.
ex: var a:Long = 10L
+ **Double**
ex: var a:Double = 3.8
+ **Float**
ex: var a:Float = 3.8F

Every number type supports the following conversions:
There are some functions that are used frequently for data type conversion. They are:
* toByte(): Byte
* toShort(): Short

* toInt(): Int
* toLong(): Long
* toFloat(): Float
* toDouble(): Double
* toChar(): Char

ex:

```
fun main(args: Array<String>) {
    var numb: Number = 60.0
    print("Number is $numb \n")
    println(numb::class.java.typeName)
    var numbByte = numb.toByte()
    print("Number in byte is $numbByte\n")
    println(numbByte::class.java.typeName)
}
```

3.2. Character Data Type

- Characters are represented by the type Char.
- You need to use single quotes ‘’ to denote a character.

For example,

```
var letter:Char = 'm'
```

3.3. Boolean Data Type: it has two values True or False

3.4. Arrays Data Type: An array is a container that holds value of single type. For example, an array of Int that holds 100 integer values, an array of Float that holds 58 floating point values etc.

In Kotlin, you can create array in 2 ways.

a. Create Array Using Library Function

You can create an array in kotlin using library function **arrayOf()** and pass the values to it. **For example**,

```
fun main(args: Array<String>) {
   var numbList = arrayOf(1,2,3)
    for(numb in numbList) {
        print(numb + " ")
    }
}
```

=> 1 2 3

b. Create Array Using Factory Function

You can create an array in Kotlin using factory function that takes size of array and the function that can return initial value of each element, present in the array, given it’s index. **For example,**

```
fun main(args: Array<String>) {
    var numbList = Array(3, { i -> (i*i).toString() })
    for(numb in numbList) {
        print(numb + " ")
    }
}
```

3.5. String Data Type

ex1:

```
val s = "Hello, world!"
```

ex2:

```
val text = """
This is
      Apple
"""
```

4. Input and Output in Kotlin Programming language

4.1. How To Print Output in Kotlin

a. Using print() method

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

b. Using println() method

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

4.2. How to Take Input From User in Kotlin

Using **readLine()** method: to read a line of input from the standard input stream, you can use **readLine()** function.

ex1:

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

ex2:

```
fun main(args: Array<String>) {
    var a:Int? = null
    var b:Int? = null
```

```
    print("Input a:")
    a = readLine()!!.toInt()
    print("Input b:")
    b = readLine()!!.toInt()
    print("Sum : $a + $b = ${a+b}")
}
```

ex3:

```
fun main(args: Array<String>) {
    var (a, b) = readLine()!!.split(' ')
    println(a.toInt() + b.toInt())
}
```

ex4:

```
fun main(args: Array<String>) {
    var (a, b) = readLine()!!.split(' ').map(String::toInt)
    println(a + b)
}
```

5. Operators in Kotlin

5.1. Arithmetic operator: Arithmetic operators in Kotlin are used to carry out the basic mathematical operations like addition, subtraction, multiplication, division and modulus.

Operator	Description	Expression
+	Addition of two variables	a + b
-	Subtraction of two variables	a - b
*	Multiplication of two variables	a * b
/	Division of two variables	a / b
%	Modulus of two variables. It will return remainder when a is divided by b.	a % b

5.2. Comparison operator: Comparison operators in Kotlin are used to compare variables. The comparison operators return either true or false, based on the result of the comparison.

Operator	Description	Expression
<	Checks if one variable is less than other. It returns **true** if the operand on the left is less than the operand on the right, otherwise returns **false**.	a < b
>	Checks if one variable is greater than other. It returns **true** if the operand on the left is greater than the operand on the right, otherwise returns **false**.	a > b
<=	Checks if one variable is less than or equal to other. It returns **true** if the operand on the left is less than or equal to the operand on the right, otherwise returns **false**.	a <= b
>=	Checks if one variable is greater than or equal to other. It returns **true** if the operand on the left is greater than or equal to the operand on the right, otherwise returns **false**.	a >= b

5.3. Assignment operators

Operator	Description	Expression
=	Assign a value to a variable	a = 10
+=	Add and assign a value to a variable	a+=b
-=	Subtract and assign a value to a variable	a-=b
*=	Multiply and assign a value to a variable	a*=b
/=	Divide and assign a value to a variable	a/=b
%=	Mudulus and assign a value to a variable	a%=b

5.4. Equality operators

Operator	Description	Expression
==	Checks if two objects are equal and returns true	a == b
!=	Checks if two objects are not equal and returns true	a != b
===	Checks if two variable refer to same object	a === b
!==	Checks if two variable does not refer to same object	a !== b

5.5. Unary operators

Operator	Description	Expression
+	Unary plus, returns positive value	+a
-	Unary minus, returns negative value	-b
++	Increment operator, increase value by 1	a++, ++a
--	Decrement operator, decrease value by 1	b--, --b

5.6. Logical operators

Operator	Description	Expression
&&	AND operator, return true if both inputs are true	a && b
||	OR operator, return true if any one is true	a || b
!	NOT operator, return negation of variable	!a

5.7. Miscellaneous Operators: Kotlin also supports some more miscellaneous operators.

a. Nullable operator (?)

Kotlin doesn't allow a field to be null by default. To make a field nullable, we need to assign **?** operator.

ex:

```
var message: String = null       // Error
var nullableMessage: String? = null   // No error
```

b. The !! operator

The null pointer assertion operator(**!!**) will convert any value to **non-null type** and throws an exception if the value is null.

ex:

```
val message: String? = null
println(message!!.length)
```

=> Note: This code will not throw any error during compile time but it will throw error at runtime. This operator is not preferred to use.

c. Elvis operator

The Elvis operator in Kotlin is used to assign some other value if the reference is null.

ex:

```
var message: String? = null
println(message?:"Message is null")
message = "Hello World"
println(message?:"Message is null")
```

Output:

```
Message is null
Hello World
```

6. Kotlin Comments

6.1. Single line comment

ex:

```
// This is a comment
println("Enter string: ")       // Enter a string here
var inputString = readLine()
println("Input in string format is: $inputString")     // Input is printed here
```

6.2. Multi-line comment

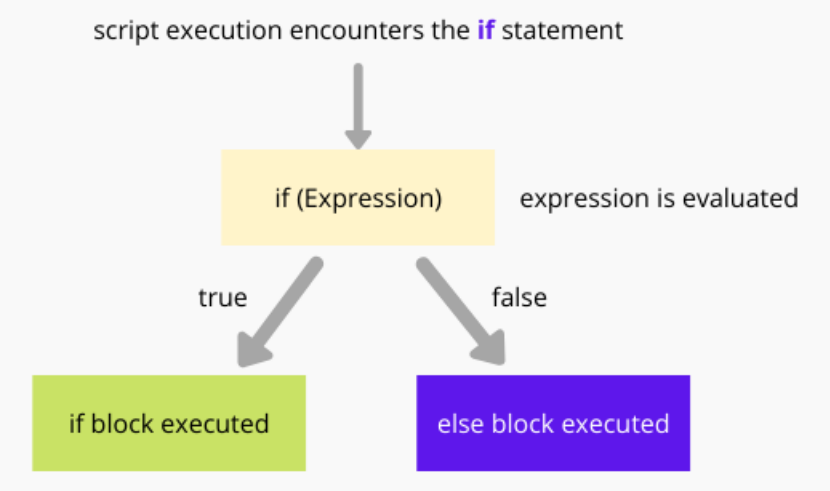
ex:

```
/*
  This is a multi-line comment
  This is second line
  Last line of comment
 */
```

```
println("Enter string: ")
var inputString = readLine()
println("Input in string format is: $inputString")
```

7. Kotlin if else Expression

7.1. Simple if-else expression



In most of the languages, if-else block is used to select a condition based on an expression. If the if expression returns true, if block will be executed. Otherwise, the else block will be executed. For example, if we want to know whether a student passed the examination or not using the score, we can write the following code:

ex:

```
fun main() {
   val marks = 70
   val result:String
   if(marks > 33){
      result = "Hurrah! You passed"
   }
   else{
      result ="Sorry! You failed"
   }
   println(result)
}
```

But in Kotlin, if-else is not a statement, it is an expression. It means the if-else block can return a value with a slight syntax change.Let's see the code below:

ex:

```
fun main() {
   val marks = 70
   val result:String
   result = if(marks > 33) {
      "Hurrah! You passed"
```

```
    }
    else{
        "Sorry! You failed"
    }
    println(result)
}
```

Here, based on the condition, the value will be returned from the if-else block and will be assigned to the result variable.

Note: While using the **if-else** as an expression, it is compulsory to have an else block with if block.

7.2. if..else..if Ladder

You can use else if condition to specify a new condition if the first condition is false.

Syntax

```
if (condition1) {
  // code block A to be executed if condition1 is true
} else if (condition2) {
  // code block B to be executed if condition2 is true
} else {
  // code block C to be executed if condition1 and condition2 are false
}
```

ex:

```
fun main(args: Array<String>) {
    val age:Int = 13

    val result = if (age > 19) {
        "Adult"
    } else if ( age > 12 && age  < 20 ){
        "Teen"
    } else {
        "Minor"
    }
    print("The value of result : ")
    println(result)
}
```

7.3. Nested if Expression

Kotlin allows to put an if expression inside another if expression. This is called nested if expression

Syntax

```
if(condition1) {
   // code block A to be executed if condition1 is true
   if( (condition2) {
```

```
    // code block B to be executed if condition2 is true
  }else{
    // code block C to be executed if condition2 is fals
  }
} else {
 // code block D to be executed if condition1 is false
}
```

ex:

```
fun main(args: Array<String>) {
   val age:Int = 20

   val result = if (age > 12) {
     if ( age > 12 && age  < 20 ){
        "Teen"
     }else{
        "Adult"
     }
   } else {
      "Minor"
   }
   print("The value of result : ")
   println(result)
}
```

8. When Expression

Consider a situation when you have large number of conditions to check. Though you can use if..else if expression to handle the situation, but Kotlin provides when expression to handle the situation in nicer way.

Kotlin when expression is similar to the switch statement in C, C++ and Java. Kotlin when can be used either as an expression or as a statement.

Ex1: using when expression form

```
fun main(args: Array<String>) {
  val day = 2
  val result = when (day) {
    1 -> "Monday"
    2 -> "Tuesday"
    3 -> "Wednesday"
    4 -> "Thursday"
    5 -> "Friday"
    6 -> "Saturday"
    7 -> "Sunday"
    else -> "Invalid day."
  }
```

```
  println(result)
}
```

8.1. Kotlin when as Statement

Ex: Let's write above example once again without using expression form

```
fun main(args: Array<String>) {
  val day = 2
  when (day) {
    1 -> println("Monday")
    2 -> println("Tuesday")
    3 -> println("Wednesday")
    4 -> println("Thursday")
    5 -> println("Friday")
    6 -> println("Saturday")
    7 -> println("Sunday")
    else -> println("Invalid day.")
  }
}
```

8.2. Combine when Conditions

We can combine multiple when conditions into a single condition.

Ex:

```
fun main(args: Array<String>) {
  val day = 2
  when (day) {
    1, 2, 3, 4, 5 -> println("Weekday")
    else -> println("Weekend")
  }
}
```

8.3. Range in when Conditions

Kotlin ranges are created using double dots **..** and we can use them while checking when condition with the help of in operator.

Ex:

```
fun main(args: Array<String>) {
  val day = 2
  when (day) {
    in 1..5 -> println("Weekday")
    else -> println("Weekend")
  }
}
```

8.4. Expression in when Conditions

Kotlin when can use expressions instead of a constant as branch condition.

Ex:

```
fun main(args: Array<String>) {
  val x = 20
  val y = 10
  val z = 10
  when (x) {
    (y+z) -> print("y + z = x = $x")
    else -> print("Condition is not satisfied")
  }
}
```

8.5. Kotlin when with block of code

Kotlin when can be put as block of code enclosed within curly braces.

Ex:

```
fun main(args: Array<String>) {
  val day = 2

  when (day) {
    1 -> {
      println("First day of the week")
      println("Monday")
    }
    2 -> {
      println("Second day of the week")
      println("Tuesday")
    }
    3 -> {
      println("Third day of the week")
      println("Wednesday")
    }
    4 -> println("Thursday")
    5 -> println("Friday")
    6 -> println("Saturday")
    7 -> println("Sunday")
    else -> println("Invalid day.")
  }
}
```

9. Kotlin For Loop

Unlike c or java or other programming languages, for loop is used differently in kotlin.

9.1. Syntax to use Kotlin for loop:

```
for(item in collection) {
   // Code to perform any action in item of the collection.
}
```

In Kotlin, you can use for loop to iterate through following things:

+ Range
+ Array
+ String
+ Collection

9.2. Iterate through Range using for loop in Kotlin

Ex:

```
fun main(args: Array<String>) {
   for (i in 1..10) {
      println(i)
   }
}
```

Different ways to iterate through Range using for loop in Kotlin

Ex1:

```
fun main(args: Array<String>) {
   for (item in 1..6) print(item)
}
```

Ex2:

```
fun main(args: Array<String>) {
   for (item in 6..1) print(item)
}
```

Ex3:

```
fun main(args: Array<String>) {
   for (item in 6 downTo 1) print(item)
}
```

Ex4:

```
fun main(args: Array<String>) {
   for (i in 1..6 step 2) print(i)
}
```

Ex5:

```
fun main(args: Array<String>) {
   for (i in 6 downTo 1 step 2) print(i)
}
```

9.3. Iterate Through An Array Using For Loop In Kotlin

In Kotlin, you can iterate through an array in following ways:

a. Without using Index property

Syntax:

```
for(itemInArray in Array) {
   // Write code to perform the action with item
}
```

Ex:

```
fun main(args: Array<String>) {
   var nameList = arrayOf("Xuan", "Ha", "Thu", "Dong")
   for (name in nameList)
      print(name + " ")
}
```

b. Using Index property

Syntax:

```
fun main(args: Array<String>) {
   for (index in array.indices) {
     // Write code to perform some action. You can access array element at position index
using array[index]
   }
}
```

Ex:

```
fun main(args: Array<String>) {
   var nameList = arrayOf("Xuan", "Ha", "Thu", "Dong")
   for (index in nameList.indices) {
         println(nameList[index])
   }
}
```

c. Using withIndex Library Function

Syntax:

```
fun main(args: Array<String>) {
for ((index,value) in array.withIndex()) {
      // Write code to perform some action
   }
```

```
}
```

Ex:

```
fun main(args:Array<String>) {
    var nameList = arrayOf("Xuan", "Ha", "Thu", "Dong")
    for ((index,value) in nameList.withIndex()) {
        println("$index - $value")
    }
}
```

9.4. Iterate Through a String using For Loop in Kotiln

You can iterate through a String using kotlin for loop in following ways:

a. Without using Index property

Syntax:

```
for(letter in String) {
    // Write code to perform the action with letter
}
```

Ex:

```
fun main(args:Array<String>) {
    var name = "Hello World"
    for (letter in name) {
        print(letter + " ")
    }
}
```

b. Using Index property

Syntax:

```
fun main(args: Array<String>) {
    for (index in string.indices) {
        // Write code to perform some action. You can access string element at position index
using string[index]
    }
}
```

Ex:

```
fun main(args:Array<String>) {
    var name = "Hello World"
    for (index in name.indices) {
        print(name[index] + " ")
    }
}
```

c. Using withIndex() library function.

Syntax:

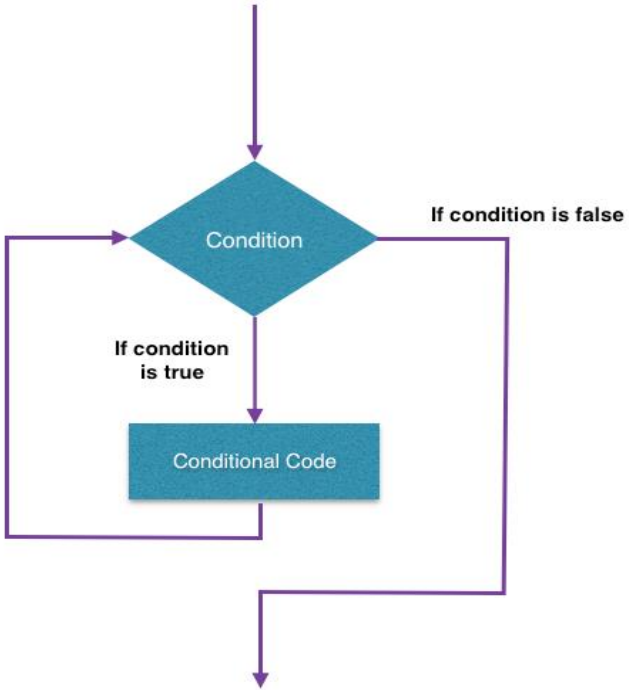
```
fun main(args: Array<String>) {
    for ((index,value) in string.withIndex()) {
        // Write code to perform some action
    }
}
```

Ex:

```
fun main(args:Array<String>) {
    var name = "Hello World"
    for ((index, value) in name.withIndex()) {
        println("$index => $value ")
    }
}
```

10. While Loop in Kotlin

Kotlin while loop executes its body continuously as long as the specified condition is true.



Syntax:

```
while (condition) {
    // body of the loop
}
```

Ex1:

```
fun main(args:Array<String>) {
    var i = 5;
    while (i > 0) {
        println(i)
        i--
    }
}
```

Ex2:

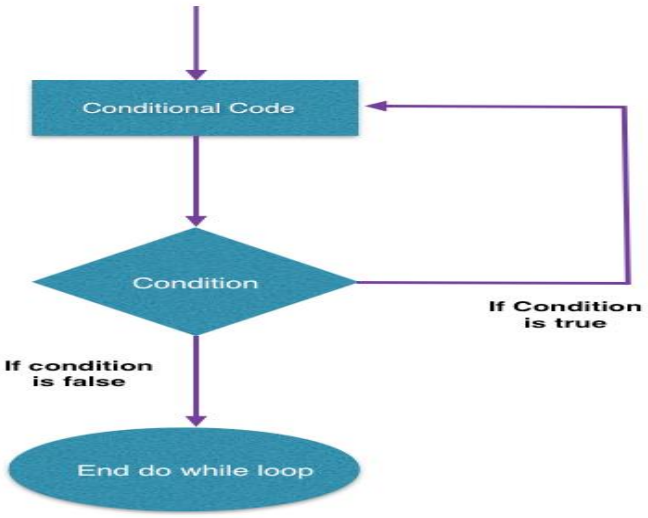
```
fun main(args:Array<String>) {
    var index = 1
    while(index <= 5) {
        println(index * index)
        index++
    }
}
```

Ex3:

```
fun main(args:Array<String>) {
    val nameList = arrayOf("Xuan", "Ha", "Thu", "Dong")
    var index = 0;
    while (index < nameList.size) {
        println(nameList[index])
        index++;
    }
}
```

11. Do...while Loop in Kotlin

The loop will always be executed at least once, even if the condition is false, because the code block is executed before the condition is tested.



Syntax:

```
do{
   // body of the loop
}while( condition )
```

Ex1:

```
fun main(args:Array<String>) {
   var i = 5;
   do{
      println(i)
      i--
   }while(i > 0)
}
```

Ex2:

```
fun main(args:Array<String>) {
   var i = 10
   do {
      println("Hello $i")
      i -= 1
   } while (i > 0)
}
```

12. Break and Continue in Kotlin

12.1. Kotlin Break Statement

Kotlin break statement is used to come out of a loop once a certain condition is met. This loop could be a for, while or do...while loop.

Syntax:

Let's check the syntax to terminate various types of loop to come out of them:

```
// Using break in for loop
for (...) {
  if(test){
    break
  }
}

// Using break in while loop
while (condition) {
  if(test){
    break
  }
}
```

```
// Using break in do...while loop
do {
  if(test){
    break
  }
}while(condition)
```

=> If test expression is evaluated to true, break statment is executed which terminates the loop and program continues to execute just after the loop statment.

Ex:

```
fun main(args:Array<String>) {
   var i = 0;
   while (i++ < 100) {
      println(i)
      if( i == 3 ){
         break
      }
   }
}
```

12.2. Kotlin Labeled Break Statement

Kotlin label is **the form of identifier followed by the @ sign**, for example test@ or outer@. To make any Kotlin Expression as labeled one, we just need to put a label in front of the expression.

```
outerLoop@ for (i in 1..100) {
   // ...
}
```

Kotlin labeled break statement is used to **terminate the specific loop**. This is done by using break expression with @ sign followed by label name (break@LabelName).

Ex:

```
fun main(args:Array<String>) {
   outerLoop@ for (i in 1..3) {
      innerLoop@ for (j in 1..3) {
         println("i = $i and j = $j")
         if (i == 2){
            break@outerLoop
         }
      }
   }
}
```

Output:
i = 1 and j = 1
i = 1 and j = 2
i = 1 and j = 3
i = 2 and j = 1

12.3. Kotlin Continue Statement

Syntax

Let's check the syntax to terminate various types of loop to come out of them:

```
// Using continue in for loop
for (...) {
  if(test){
    continue
  }
}

// Using continue in while loop
while (condition) {
  if(test){
    continue
  }
}

// Using continue in do...while loop
do {
  if(test){
    continue
  }
}while(condition)
```

If test expression is evaluated to true, continue statment is executed which skips remaning part of the loop and jump to the next iteration of the loop statment.

Ex:

```
fun main(args:Array<String>) {
   var i = 0;
   while (i++ < 6) {
      if( i == 3 ){
         continue
      }
      println(i)
   }
```

```
}
```

12.4. Kotlin Labeled Continue Statement

```
fun main(args:Array<String>) {
   outerLoop@ for (i in 1..3) {
      innerLoop@ for (j in 1..3) {
         if (i == 2){
            continue@outerLoop
         }
         println("i = $i and j = $j")
      }
   }
}
```

Output:

```
i = 1 and j = 1
i = 1 and j = 2
i = 1 and j = 3
i = 3 and j = 1
i = 3 and j = 2
i = 3 and j = 3
```