KOTLIN CONTROL FLOW

A programming language uses control statements to control the flow of execution of a program based on certain conditions. If the condition is true then it enters into the conditional block and executes the instructions.

There are different types of if-else expressions in Kotlin:

- if expression
- if-else expression
- if-else-if ladder expression
- nested if expression

if statement:

It is used to specify whether a block of statements will be executed or not, i.e. if a condition is true, then only the statement or block of statements will be executed otherwise it fails to execute.

Syntax:

```
if(condition) {


    // code to run if condition is true
}
```

Example:

```
fun main(args: Array<String>) {
   var a = 3
   if(a > 0){
      print("Yes,number is positive")
   }
}
```

Output: Yes, number is positive

if-else statement:

if-else statement contains two blocks of statements. ‘if’ statement is used to execute the block of code when the condition becomes true and ‘else’ statement is used to execute a block of code when the condition becomes false.

Syntax:

```
 if(condition) {
       // code to run if condition is true
}
else {
      // code to run if condition is false
}
```

Example:

```
fun main(args: Array<String>) {
       var a = 5
       var b = 10
       if(a > b){
          print("Number 5 is larger than 10")
       }
       else{
          println("Number 10 is larger than 5")
       }
   }
```

Output:

Number 10 is larger than 5

Kotlin if-else expression as ternary operator:

In Kotlin, if-else can be used as an expression because it returns a value. Unlike java, there is no ternary operator in Kotlin because if-else returns the value according to the condition and works exactly similar to ternary.

Below is the Kotlin program to find the greater value between two numbers using if-else expression.

```
fun main(args: Array<String>) {
    var a = 50
    var b = 40


    // here if-else returns a value which
    // is to be stored in max variable
    var max = if(a > b){
        print("Greater number is: ")
        a
    }
    else{
        print("Greater number is:")
        b
    }
    print(max)
}
```

Output:

Greater number is: 50

if-else-if ladder expression:

Here, a user can put multiple conditions. All the ‘if’ statements are executed from the top to bottom. One by one all the conditions are checked and if any of the conditions is found to be true then the code associated with the if statement will be executed and all other statements bypassed to the end of the block. If none of the conditions is true, then by default the final else statement will be executed.

Syntax:

```
if(Firstcondition) {
   // code to run if condition is true
}
else if(Secondcondition) {
   // code to run if condition is true
}
else{
}
```

Below is the Kotlin program to determine whether the number is positive, negative, or equal to zero.

```
import java.util.Scanner

fun main(args: Array<String>) {

   // create an object for scanner class
   val reader = Scanner(System.`in`)
   print("Enter any number: ")

   // read the next Integer value
```

```
    var num = reader.nextInt()
    var result  = if ( num > 0){
        "$num is positive number"
    }
    else if( num < 0){
        "$num is negative number"
    }
    else{
        "$num is equal to zero"
    }
    println(result)

}
```

Output:

```
Enter any number: 12
12 is positive number

Enter any number: -11
-11 is negative number

Enter any number: 0
0 is zero
```

nested if expression:

Nested if statements mean an if statement inside another if statement. If the first condition is true then code the associated block to be executed, and again check for the if condition nested in the first block and if it is also true then execute the code associated with it. It will go on until the last condition is true.

Syntax:

```
if(condition1){
        // code 1
    if(condition2){
            // code2
    }
}
```

Below is the Kotlin program to determine the largest value among the three Integers.

```
import java.util.Scanner


fun main(args: Array<String>) {


   // create an object for scanner class
   val reader = Scanner(System.`in`)
   print("Enter three numbers: ")


   var num1 = reader.nextInt()
   var num2 = reader.nextInt()
   var num3 = reader.nextInt()
```

```
    var max  = if ( num1 > num2) {
        if (num1 > num3) {
            "$num1 is the largest number"
        }
        else {
            "$num3 is the largest number"
        }
    }
    else if( num2 > num3){
        "$num2 is the largest number"
    }
    else{
        "$num3 is the largest number"
    }
    println(max)

}
```

Output:

```
Enter three numbers: 123 231 321
321 is the largest number
```

While loop

It consists of a block of code and a condition. First of all the condition is evaluated and if it is true then execute the code within the block. It repeats until the condition becomes false because every time the condition is checked before entering into the block. The while loop can be thought of as repeating of if statements.

The syntax of while loop-

```
while(condition) {
        // code to run
}
```

Kotlin program to print numbers from 1 to 10 using while loop:

```
fun main(args: Array<String>) {
   var number = 1

   while(number <= 10) {
      println(number)
      number++;
   }
}
```

Output:

```
1
2
3
4
5
```

6

7

8

9

10

Kotlin program to print the elements of an array using while loop:

```
fun main(args: Array<String>) {
    var names = arrayOf("Praveen","Gaurav","Akash","Sidhant","Abhi","Mayank")
    var index = 0

    while(index < names.size) {
        println(names[index])
        index++
    }
}
```

Output:

Praveen

Gaurav

Akash

Sidhant

Abhi

Mayank

Do-while loop

Like Java, do-while loop is a control flow statement which executes a block of code at least once without checking the condition, and then repeatedly executes the block, or not, it totally depends upon a Boolean condition at the end of do-while block.

It contrast with the while loop because while loop executes the block only when condition becomes true but do-while loop executes the code first and then the expression or test condition is evaluated.

do-while loop working – First of the all the statements within the block is executed, and then the condition is evaluated. If the condition is true the block of code is executed again. The process of execution of code block repeated as long as the expression evaluates to true. If the expression becomes false, the loop terminates and transfers control to the statement next to do-while loop. It is also known as post-test loop because it checks the condition after the block is executed.

Syntax of the do-while loop-

```
do {
    // code to run
}
while(condition)
```

Kotlin program to find the factorial of a number using do-while loop –

```
fun main(args: Array<String>) {
   var number = 6
   var factorial = 1
   do {
      factorial *= number
      number--
   }while(number > 0)
   println("Factorial of 6 is $factorial")
}
```

Output:

Factorial of 6 is 720

Kotlin program to print table of 2 using do-while loop –

```
fun main(args: Array<String>) {
    var num = 2
    var i = 1
    do {
        println("$num * $i = "+ num * i)
        i++
    }while(i <= 10)
}
```

Output:

2 * 1 = 2

2 * 2 = 4

2 * 3 = 6

2 * 4 = 8

2 * 5 = 10

2 * 6 = 12

2 * 7 = 14

2 * 8 = 16

2 * 9 = 18

2 * 10 = 20

For loop:

In Kotlin, for loop is equivalent to foreach loop of other languages like C#. Here for loop is used to traverse through any data structure which provides an iterator. It is used very differently then the for loop of other programming languages like Java or C.

The syntax of for loop in Kotlin:

```
for(item in collection) {
    // code to execute
}
```

In Kotlin, for loop is used to iterate through the following because all of them provides iterator.

- Range
- Array
- String
- Collection

Iterate through range using for loop –

You can traverse through Range because it provides iterator. There are many ways you can iterate through Range. The in operator used in for loop to check value lies within the Range or not.

Below programs are example of traversing the range in different ways and in is the operator to check the value in the range. If value lies in between range then it returns true and prints the value.

Iterate through range to print the values:

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

```
        print("$i ")
    }
}
```

Output:

1 2 3 4 5 6

Iterate through range to jump using step-3 :

```
fun main(args: Array<String>)
{
    for (i in 1..10 step 3) {
        print("$i ")
    }
}
```

Output:

1 4 7 10

You can not iterate through Range from top to down without using DownTo :

```
fun main(args: Array<String>)
{
    for (i in 5..1) {
        print("$i ")
    }
    println("It prints nothing")
}
```

Output: It prints nothing

Iterate through Range from top to down with using downTo :

```
fun main(args: Array<String>)
{
   for (i in 5 downTo 1) {
      print("$i ")
   }
}
```

Output:

5 4 3 2 1

Iterate through Range from top to down with using downTo and step 3:

```
fun main(args: Array<String>)
{
   for (i in 10 downTo 1 step 3) {
      print("$i ")
   }
}
```

Output:

10 7 4 1

Iterate through array using for loop –

An array is a data structure which contains same data type like Integer or String. Array can be traversed using for loop because it also provides iterator. Each array has a starting index and by default, it is 0.

There are the following you can traverse array:

- Without using Index property
- With Using Index property
- Using withIndex Library Function

Traverse an array without using index property:

```
fun main(args: Array<String>) {
    var numbers = arrayOf(1,2,3,4,5,6,7,8,9,10)
    for (num in numbers){
        if(num%2 == 0){
            print("$num ")
        }
    }
}
```

Output:

2 4 6 8 10

Traverse an array with using index property:

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

    var planets = arrayOf("Earth", "Mars", "Venus", "Jupiter", "Saturn")

    for (i in planets.indices) {
        println(planets[i])
    }
}
```

Output:

Earth

Mars

Venus

Jupiter

Saturn

Traverse an array using withIndex() Library Function:

```
fun main(args: Array<String>) {
   var planets = arrayOf("Earth", "Mars", "Venus", "Jupiter", "Saturn")


   for ((index,value) in planets.withIndex()) {
      println("Element at $index th index is $value")
   }
}
```

Output:

Element at 0 th index is Earth

Element at 1 th index is Mars

Element at 2 th index is Venus

Element at 3 th index is Jupiter

Element at 4 th index is Saturn

Iterate through string using for loop –

A string can be traversed using the for loop because it also provides iterator.

There are following ways to traverse the string:

- Without using Index property
- With Using Index property
- Using withIndex Library Function

```
fun main(args: Array<String>) {
    var name = "Geeks"
    var name2 = "forGeeks"

    // traversing string without using index property
    for (alphabet in name)   print("$alphabet ")

    // traversing string with using index property
    for (i in name2.indices) print(name2[i]+" ")
    println(" ")

    // traversing string using withIndex() library function
    for ((index,value) in name.withIndex())
    println("Element at $index th index is $value")
}
```

Output:

G e e k s f o r G e e k s

Element at 0 th index is G

Element at 1 th index is e

Element at 2 th index is e

Element at 3 th index is k

Element at 4 th index is s

Iterate through collection using for loop –

You can traverse the collection using the for loop. There are three types of collections list, map and set.

In the listOf() function we can pass the different data types at the same time.

Below is the program to traverse the list using for loop.

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


    // read only, fix-size
    var collection = listOf(1,2,3,"listOf", "mapOf", "setOf")


    for (element in collection) {
        println(element)
    }
}
```

Output:

1

2

3

listOf

mapOf

setOf

when :

In Kotlin, when replaces the switch operator of other languages like Java. A certain block of code needs to be executed when some condition is fulfilled. The argument of when expression compares with all the branches one by one until some match is found. After the first match is found, it reaches to end of the when block and executes the code next to the when block. Unlike switch cases in Java or any other programming language, we do not require a break statement at the end of each case.

In Kotlin, when can be used in two ways:

- when as a statement
- when as an expression

Using when as a statement with else

when can be used as a statement with or without else branch. If it is used as a statement, the values of all individual branches are compared sequentially with the argument and execute the corresponding branch where the condition matches. If none of the branches is satisfied with the condition then it will execute the else branch.

```
fun main (args : Array<String>) {
    print("Enter the name of heavenly body: ")
    var name= readLine()!!.toString()
    when(name) {
        "Sun" -> print("Sun is a Star")
        "Moon" -> print("Moon is a Satellite")
        "Earth" -> print("Earth is a planet")
```

```
        else -> print("I don't know anything about it")
    }
}
```

Output:

```
Enter the name of heavenly body: Sun
Sun is a Star
Enter the name of heavenly body: Mars
I don't know anything about it
```

Using when as a statement without else

We can use when as a statement without else branch. If it is used as a statement, the values of all individual branches are compared sequentially with the argument and execute the corresponding branch where condition matches. If none of the branches are satisfied with the condition then it simply exits the block without printing anything to system output.

```
fun main (args : Array<String>) {
    print("Enter the name of heavenly body: ")
    var name= readLine()!!.toString()
    when(name) {
        "Sun" -> print("Sun is a Star")
        "Moon" -> print("Moon is a Satellite")
        "Earth" -> print("Earth is a planet")

    }
}
```

Output:

```
Enter the name of heavenly body: : Mars
```

Process finished with exit code 0

Using when as an expression

If it is used as an expression, the value of the branch with which condition satisfied will be the value of overall expression. As an expression when returns a value with which the argument matches and we can store it in a variable or print directly.

```
fun main(args : Array<String>) {
    print("Enter number of the Month: ")
    var monthOfYear  = readLine()!!.toInt()
    var month= when(monthOfYear) {
        1->"January"
        2->"February"
        3->"March"
        4->"April"
        5->"May"
        6->"June"
        7->"July"
        8->"August"
        9->"September"
        10->"October"
        11->"November"
        12->"December"
        else-> "Not a month of year"
    }
    print(month)
}
```

Output:

Enter number of the Month: 8

August

If none of the branch conditions are satisfied with the argument, the else branch is executed. As an expression, the else branch is mandatory, unless the compiler can prove that all possible cases are covered with branch conditions. If we cannot use else branch it will give a compiler error.

Error:(6, 17) Kotlin: 'when' expression must be exhaustive, add necessary 'else' branch

Different ways to use when the block in Kotlin

Combine multiple branches in one using comma:

We can use multiple branches in a single one separated by a comma. When common logic is shared by some branches then we can combine them in a single branch. In the example below, we need to check the entered large body is a planet or not, so we combined all the names of planets in a single branch. Anything entered other than planet name will execute the else branch.

```
fun main (args :Array<String>) {
   print("Enter name of the planet: ")
   var name=readLine()!!.toString()
   when(name) {
      "Mercury","Earth","Mars","Jupiter"
         ,"Neptune","Saturn","Venus","Uranus" -> print("This is a planet")
      else -> print("This not a planet")
   }
}
```

Output:

Enter name of the planet: Earth

Planet

Check the input value in the range or not:

Using the in or !in operator, we can check the range of argument passed in when block. ‘in’ operator in Kotlin is used to check the existence of a particular variable or property in a range. If the argument lies in a particular range then in operator returns true and if the argument does not lie in a particular range then !in returns true.

```
fun main (args:Array<String>) {
    print("Enter the month number of year: ")
    var num= readLine()!!.toInt()
    when(num) {
        in 1..3 -> print("Spring season")
        in 4..6 -> print("Summer season")
        in 7..8 -> print("Rainy season")
        in 9..10 -> print("Autumn season")
        in 11..12 -> print("Winter season")
        !in 1..12 -> print("Enter valid month of year")
    }
}
```

Output:

Enter the month number of year: 5

It is summer season

Enter the month number of year: 14

Enter valid month of year

Check given variable is of a certain type or not:

Using is or !is operator we can check the type of variable passed as an argument in when block. If the variable is Integer type then is Int returns true else return false.

```
fun main(args: Array<String>) {
    var num: Any = "GeeksforGeeks"
    when(num){
        is Int -> println("It is an Integer")
        is String -> println("It is a String")
        is Double -> println("It is a Double")
    }
}
```

Output:

It is a String

Using when as a replacement for an if-else-if chain:

We can use when as a replacement for if-else-if. If no argument is supplied then the branch conditions are simply boolean expressions, and a branch is executed only when its condition is true:

```
fun isOdd(x: Int) = x % 2 != 0
fun isEven(x: Int) = x % 2 == 0

fun main(args: Array<String>) {
    var num = 8
    when{
        isOdd(num) ->println("Odd")
        isEven(num) -> println("Even")
        else -> println("Neither even nor odd")
    }
}
```

Output:

Even

Check that a string contains a particular prefix or suffix:

We can also check prefix or suffix in a given string by the below method. If the string contains the prefix or suffix then it will return the Boolean value true else return false.

```
fun hasPrefix(company: Any):Boolean{
return when (company) {
    is String -> company.startsWith("GeeksforGeeks")
    else -> false
  }
}

fun main(args: Array<String>) {
    var company = "GeeksforGeeks a computer science portal"
    var result = hasPrefix(company)
    if(result) {
        println("Yes, string started with GeeksforGeeks")
    }
    else {
        println("No, String does not started with GeeksforGeeks")
    }
}
```

Output:

Yes, string started with GeeksforGeeks