Control Flow .. continuation

In this lesson we look at;

1. when
2. while
3. for
4. break
5. continue

Practice codes in this document at https://play.kotlinlang.org or in your local machine

Kotlin when expression

Kotlin's `when` expression is used to evaluate multiple conditions, it works more similar to an else if statements done in previous lesson

The `when` keyword matches its argument against all branches sequentially until some branch condition is satisfied. It can be used either as an expression or as a statement.

Below is an an example using WHEN , to check if a number is less/ *greater/* equal to 10

```
fun main(args: Array<String>) {
   var a:Int = 2 //set a variable
   when{
      a<10 -> println("$a is less than 10")
      a==10 -> println("$a is equal to 10")
      a>10 -> println("$a is greater than 10")
      else-> println("Invalid") //means above condition were not met
   }
}
```

Above we check;

when a < 10 we point (->)   **println**("$a is less than 10")

when a > 10 we point (->)   **println**("$a is greater than 10")

when a == 10 we point (->)   **println**("$a is equal to 10")

Ouput

2 is less than 10

In above example please note the else statement

```
else-> println("Invalid") //means above condition were not met
```

else works if the conditions set were not met, then it lands to an else statement, more similar to **else if statements**

The *cool* thing about `when` in Kotlin, is that we can use ranges with **`when`**. This functionality is useful in a lot of scenarios and we do not even need to use if-else for the purpose. Check out how simple it is to use ranges with `when`, in below code example:

in below example notice that the variable is passed in **when(variable){}**

```
fun main(args: Array<String>) {
    var marks:Int = 90 //we set some marks here
    when(marks){
        in 1..30 -> println("You have $marks - Below average")
        in 31..50 -> println("You have $marks  - Average")
        in 51..75 -> println("You have $marks  - Above average")
        in 76..99 -> println("You have $marks -  Excellent")
        else -> println("You have $marks - Invalid Marks try again")
    }
}
```

Ouput

You have 90 – Excellent

when(marks) check what is the value of marks, then check where that value lies in conditions I.e **in 0..50 ,** if that the case then points(->) **println**("You have $marks - Below average") and so on….

Assume you want to multiply marks by 2, after a given condition is met?

So we can extend our program to look like below, please observe the braces {} in red

and more calculation inside.

```
fun main(args: Array<String>) {
    var marks: Int = 10 //we set some marks here
    when(marks){
        in 1..30 -> {
            //assume we need to multiply marks by 2
            println("You have $marks - Below average")
            var new: Int = marks * 2
            println("You have new marks is $new")
        }

        in 31..50 -> println("You have $marks  - Average")
        in 51..75 -> println("You have $marks  - Above average")
        in 76..99 -> println("You have $marks -  Excellent")
        else -> println("You have $marks - Invalid Marks try again")

    }
}
```

Output

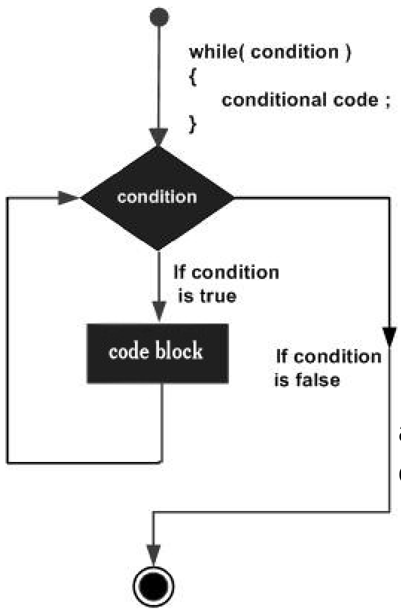
```
You have 10 - Below average
 You have new marks is 20
```

Above code we needed to multiply marks by 2 hence put everything in {} and do more *calculations* inside, the rest of conditions scan follow the same suite where needed

Exploring while loop

Looping is process of repeating a task n – times, we will look at a **while** and a **for** loop

`while` work just in C or Python. Let us try to code above example using `while`



```
First we need to set a condition
if the condition is true
the loop works
if the condition is false
the loop fails to run
NB: The loop will always work for
a true condition and fail for false
ones
```

```
fun main(args: Array<String>) {
    var count = 0 //we set some initial variable
    //works if condition is true
    while(count<=10){
        println("It Looping $count")
        count = count + 1// or count++ means add 1 to
        //increment/decrement count so as one ...time it will be
11 and make condition false
    }
}
```

Output

```
It Looping 0 It Looping 1 It Looping 2 It Looping 3 It Looping 4
It Looping 5 It Looping 6 It Looping 7 It Looping 8 It Looping 9
It Looping 10
```

Exploring for:

`for` works like usual, `for` iterates through anything that provides an iterator. Check out the example below:

In Kotlin, `for` loop is used to iterate through ranges, arrays, maps and so on (anything that provides an iterator).

The syntax of `for` loop in Kotlin is:

```
for (item in collection) {
    // body of loop
}
```

Example: Iterate Through a Range

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

Here, the loop iterates through the range and prints individual item.

When you run the program, the **output** will be:

```
1
2
3
4
5
```

If the body of the loop contains only one statement (like above example), it's not necessary to use curly braces `{ }`.

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

Below code shows different ways to use **for loop** in a range

Example: Different Ways to Iterate Through a Range

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

    print("for (i in 1..5) print(i) = ")
    for (i in 1..5) print(i)

    println() //new line
    print("for (i in 5..1) print(i) = ")
    for (i in 5..1) print(i)              // prints nothing

    println()

    print("for (i in 5 downTo 1) print(i) = ")
    for (i in 5 downTo 1) print(i)

    println()

    print("for (i in 1..4 step 2) print(i) = ")
    for (i in 1..5 step 2) print(i)

    println()

    print("for (i in 4 downTo 1 step 2) print(i) = ")
    for (i in 5 downTo 1 step 2) print(i)
}
```

When you run the program, the **output** will be:

```
for (i in 1..5) print(i) = 12345
for (i in 5..1) print(i) =
for (i in 5 downTo 1) print(i) = 54321
for (i in 1..4 step 2) print(i) = 135
for (i in 4 downTo 1 step 2) print(i) = 531
```

Iterating Through an Array

Here's an example to iterate through a `String` array.

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

    var language = arrayOf("Ruby", "Koltin", "Python" ,"Java")

    for (item in language)
        println(item)
}
```

When you run the program, the **output** will be:

```
Ruby
Koltin
Python
Java
```

Iterating Through an Array

Here's an example to iterate through a `Int` array.

```
fun main(args: Array<String>) {
    //loop through ints....
    var numbers = arrayOf(34,66,77,55,66,55,44)

    for (number in numbers)
        println(number)
}
```

When you run the program, the **output** will be:

```
34

66

77

55

66

55

44
```

Iterating Through a String

```
fun main(args: Array<String>) {
    //loops each letter in a given text
    var text= "Kotlin"

    for (letter in text) {
        println(letter)
    }
}
```

When you run the program, the output will be:

```
K
o
t
l
i
n
```

Jumps

Kotlin has three structural jump expressions:

- *return*. By default returns from the nearest enclosing function or anonymous function.
- *break*. Terminates the nearest enclosing loop.
- *continue*. Proceeds to the next step of the nearest enclosing loop.

Below is an example using break and continue, in this example the loop runs from 1...10 it breaks if it reaches 4.

```
 fun main(args: Array<String>) {
   for (i in 1..10) {
       if (i==4){ //if i reaches 4 in a loop , we stop/break
           println("$i")
           break;
       }

       else {  //i has not reached 4 we keep looping
           println("$i")
           continue;
       }//end else
   } //end for
} //end main fun
```

Output

1 2 3 4

Useful links – Read more on below links

https://www.tutorialspoint.com/kotlin/kotlin_control_flow.htm

https://www.javatpoint.com/kotlin-for-loop https://www.javatpoint.com/kotlin-while-loop

https://www.javatpoint.com/kotlin-when-expression

https://www.javatpoint.com/kotlin-continue-structure

Practice codes in this document at https://play.kotlinlang.org or in your local machine