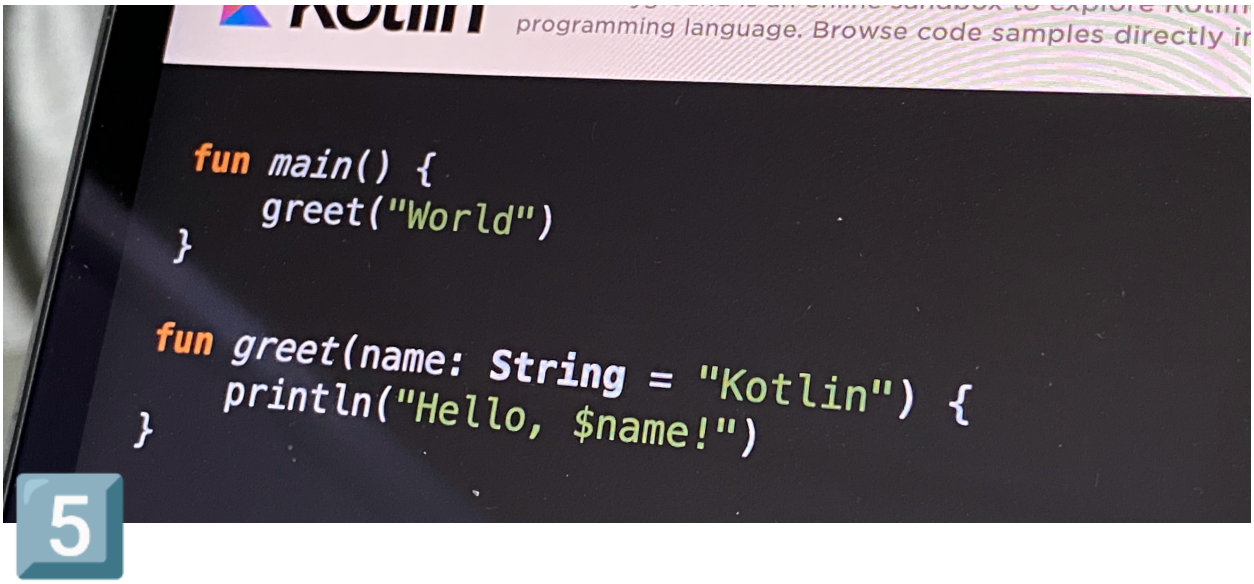


# Class 5(Operators)

## 1. Arithmetic Operators

Here's a list of arithmetic operators in Kotlin:

| Operator | Meaning |
|---|---|
| + | Addition (also used for string concatenation) |
| - | Subtraction Operator |
| * | Multiplication Operator |
| / | Division Operator |
| % | Modulus Operator |

### Example: Arithmetic Operators

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

    val number1 = 12.5
    val number2 = 3.5
    var result: Double

    result = number1 + number2
    println("number1 + number2 = $result")
```

```
    result = number1 - number2
    println("number1 - number2 = $result")

    result = number1 * number2
    println("number1 * number2 = $result")

    result = number1 / number2
    println("number1 / number2 = $result")

    result = number1 % number2
    println("number1 % number2 = $result")
}
```

When you run the program, the output will be:

```
number1 + number2 = 16.0
number1 - number2 = 9.0
number1 * number2 = 43.75
number1 / number2 = 3.5714285714285716
number1 % number2 = 2.0
```

The `+` operator is also used for the concatenation of `String` values.

---

## Example: Concatenation of Strings

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

    val start = "Talk is cheap. "
    val middle = "Show me the code. "
    val end = "- Linus Torvalds"

    val result = start + middle + end
    println(result)
}
```

When you run the program, the output will be:

```
Talk is cheap. Show me the code. - Linus Torvalds
```

---

## How arithmetic operators actually work?

Suppose, you are using `+` arithmetic operator to add two numbers a and b.

Under the hood, the expression `a + b` calls `a.plus(b)` member function.
The `plus` operator is overloaded to work with `String` values and other basic data types (except Char and Boolean).

```
// + operator for basic types
operator fun plus(other: Byte): Int
operator fun plus(other: Short): Int
operator fun plus(other: Int): Int
operator fun plus(other: Long): Long
operator fun plus(other: Float): Float
operator fun plus(other: Double): Double

// for string concatenation
operator fun String?.plus(other: Any?): String
```

You can also use `+` operator to work with user-defined types (like objects) by overloading `plus()` function.

**Recommended Reading:** Kotlin Operator Overloading

Here's a table of arithmetic operators and their corresponding functions:

| Expression | Function name | Translates to |
|---|---|---|
| a + b | plus | a.plus(b) |
| a - b | minus | a.minus(b) |
| a * b | times | a.times(b) |
| a / b | div | a.div(b) |
| a % b | mod | a.mod(b) |

Class example:

```
fun main(args:Array<String>){
    var num1:Double = 5.0
    var num2:Double = 2.0
    //1. +(addition)
    var add = num1+num2
    println("Addition: ${add}")
    //2. -(subtraction)
```

```
    var sub = num1-num2
    println("Subtraction: ${sub}")
    //3. *(Multiplication)
    var malti = num1*num2
    println("Multiplication: ${malti}")
    //4. /(division)
    var div = num1/num2
    println("Division: ${div}")
    //5. %(Modulus)
    var mod = num1%num2
    println("Modulus: ${mod}")
}
```

# 2. Assignment Operators

Assignment operators are used to assign value to a variable. We have already used simple assignment operator `=` before.

```
val age = 5
```

Here, 5 is assigned to variable age using `=` operator.

Here's a list of all assignment operators and their corresponding functions:

| Expression | Equivalent to | Translates to |
|---|---|---|
| a +=b | a = a + b | a.plusAssign(b) |
| a -= b | a = a - b | a.minusAssign(b) |
| a *= b | a = a * b | a.timesAssign(b) |
| a /= b | a = a / b | a.divAssign(b) |
| a %= b | a = a % b | a.modAssign(b) |

## Example: Assignment Operators

```
fun main(args: Array<String>) {
    var number = 12

    number *= 5   // number = number*5
```

```
    println("number  = $number")
}
```

When you run the program, the output will be:

```
number = 60
```

Class Example:

```
fun main(args:Array<String>){
    var a = 11.0
    var b = 5.0
    //1. a+=b -> a = a+b
    a+=b
    println(a)
    //2. a-=b -> a = a-b
    //3. a*=b -> a = a*b
    //4. a/=b -> a = a/b
    //5. a%=b -> a = a%b
}
```

# 3. Unary prefix and Increment / Decrement Operators

Here's a table of unary operators, their meaning, and corresponding functions:

| Operator | Meaning | Expression | Translates to |
|---|---|---|---|
| + | Unary plus | +a | a.unaryPlus() |
| - | Unary minus (inverts sign) | -a | a.unaryMinus() |
| ! | not (inverts value) | !a | a.not() |
| ++ | Increment: increases value by1 | ++a | a.inc() |
| -- | Decrement: decreases value by 1 | --a | a.dec() |

## Example: Unary Operators

```
fun main(args: Array<String>) {
    val a = 1
    val b = true
    var c = 1

    var result: Int
    var booleanResult: Boolean

    result = -a
    println("-a = $result")

    booleanResult = !b
    println("!b = $booleanResult")

    --c
    println("--c = $c")
}
```

When you run the program, the output will be:

```
-a = -1
!b = false
--c = 0
```

Class Example:

```
fun main(args: Array<String>) {
    var a = 1
    //1. +a
    println(+a)

    //2. -a
    println(-a)

    var b = true
    //3.!b
    println(!b)

    //4. a++(increment)
    //a++
    println(++a)

    //5. a--(decrement)
//    println(a)
```

```
}
```

# 4. Comparison and Equality Operators

Here's a table of equality and comparison operators, their meaning, and corresponding functions:

| Operator | Meaning | Expression | Translates to |
|---|---|---|---|
| > | greater than | a > b | a.compareTo(b) > 0 |
| < | less than | a < b | a.compareTo(b) < 0 |
| >= | greater than or equals to | a >= b | a.compareTo(b) >= 0 |
| <= | less than or equals 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)) |

Comparison and equality operators are used in control flow such as *if expression*, *when expression*, and *loops*.

---

## Example: Comparison and Equality Operators

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

    val a = -12
    val b = 12

    // use of greater than operator
    val max = if (a > b) {
        println("a is larger than b.")
        a
    } else {
        println("b is larger than a.")
        b
    }

    println("max = $max")
}
```

When you run the program, the output will be:

```
b is larger than a.
max = 12
```

Class Example:

```
fun main(args: Array<String>) {
    var a = 10
    var b = 20

    //1. > (greater than)
    println(a>b)
    //2. < (Less than)
    println(a<b)
    //3. >= (greater than equal)
    println(a>=b)
    //4. <= (less than equal)
    println(a<=b)
    //5. == (equal)
    println(a==b)
    //6. != (not equal)
    println(a!=b)

}
```

# 5. Logical Operators

There are two logical operators in Kotlin: `||` and `&&`

Here's a table of logical operators, their meaning, and corresponding functions.

| Operator | Description | Expression | Corresponding Function |
|---|---|---|---|
| \|\| | `true` if either of the Boolean expression is `true` | `(a>b)\|\|(a<c)` | `(a>b)or(a<c)` |
| && | true if all Boolean expressions are `true` | `(a>b)&&(a<c)` | `(a>b)and(a<c)` |

Note that, `or` and `and` are functions that support infix notation.

Logical operators are used in control flow such as *if expression*, *when expression*, and *loops*.

### Example: Logical Operators

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

    val a = 10
    val b = 9
    val c = -1
    val result: Boolean

    // result is true is a is largest
    result = (a>b) && (a>c) // result = (a>b) and (a>c)
    println(result)
}
```

When you run the program, the output will be:

```
true
```

---

# 6. in Operator

The `in` operator is used to check whether an object belongs to a collection.

| Operator | Expression | Translates to |
|---|---|---|
| in | a in b | b.contains(a) |
| !in | a !in b | !b.contains(a) |

---

### Example: in Operator

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

    val numbers = intArrayOf(1, 4, 42, -3)

    if (4 in numbers) {
        println("numbers array contains 4.")
    }
}
```

When you run the program, the output will be:

```
numbers array contains 4.
```

# 7. Index access Operator

Here are some expressions using index access operator with corresponding functions in Kotlin.

| Expression | Translated to |
|---|---|
| `a[i]` | `a.get(i)` |
| `a[i, n]` | `a.get(i, n)` |
| `a[i1, i2, ..., in]` | `a.get(i1, i2, ..., in)` |
| `a[i] = b` | `a.set(i, b)` |
| `a[i, n] = b` | `a.set(i, n, b)` |
| `a[i1, i2, ..., in] = b` | `a.set(i1, i2, ..., in, b)` |

## Example: Index access Operator

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

    val a  = intArrayOf(1, 2, 3, 4, - 1)
    println(a[1])
    a[1]= 12
    println(a[1])
}
```

When you run the program, the output will be:

```
2
12
```

# 8. Invoke Operator

Here are some expressions using invoke operator with corresponding functions in Kotlin.

| Expression | Translated to |
| --- | --- |
| `a()` | `a.invoke()` |
| `a(i)` | `a.invoke(i)` |
| `a(i1, i2, ..., in)` | `a.inkove(i1, i2, ..., in)` |
| `a[i] = b` | `a.set(i, b)` |

In Kotlin, parenthesis are translated to call `invoke` member function.

---

## Bitwise Operation

Unlike Java, there are no bitwise and bitshift operators in Kotlin. To perform these task, various functions (supporting infix notation) are used:

- `shl` - Signed shift left
- `shr` - Signed shift right
- `ushr` - Unsigned shift right
- `and` - Bitwise and
- `or` - Bitwise or
- `xor` - Bitwise xor
- `inv` - Bitwise inversion