



Week 1 : Session 3

Typecasting, Operations and Control flow Statements

→ Condition
 ↳ if-else
 ↳ when }

Agenda

- What Kotlin Type Conversion
 - Explicit Type Casting.
- Kotlin Operators
 - Arithmetic
 - Assignment
 - Unary prefix and Increment / Decrement Operators ✓
 - Comparison and Equality Operators ✓
 - Logical Operators ✓
 - in Operator ✓
 - as and as? Type Casting Operator ✓
- Kotlin Control Flow
 - **if - else** Expression ✓
 - **when** Expression ✓



Kotlin Type Conversion

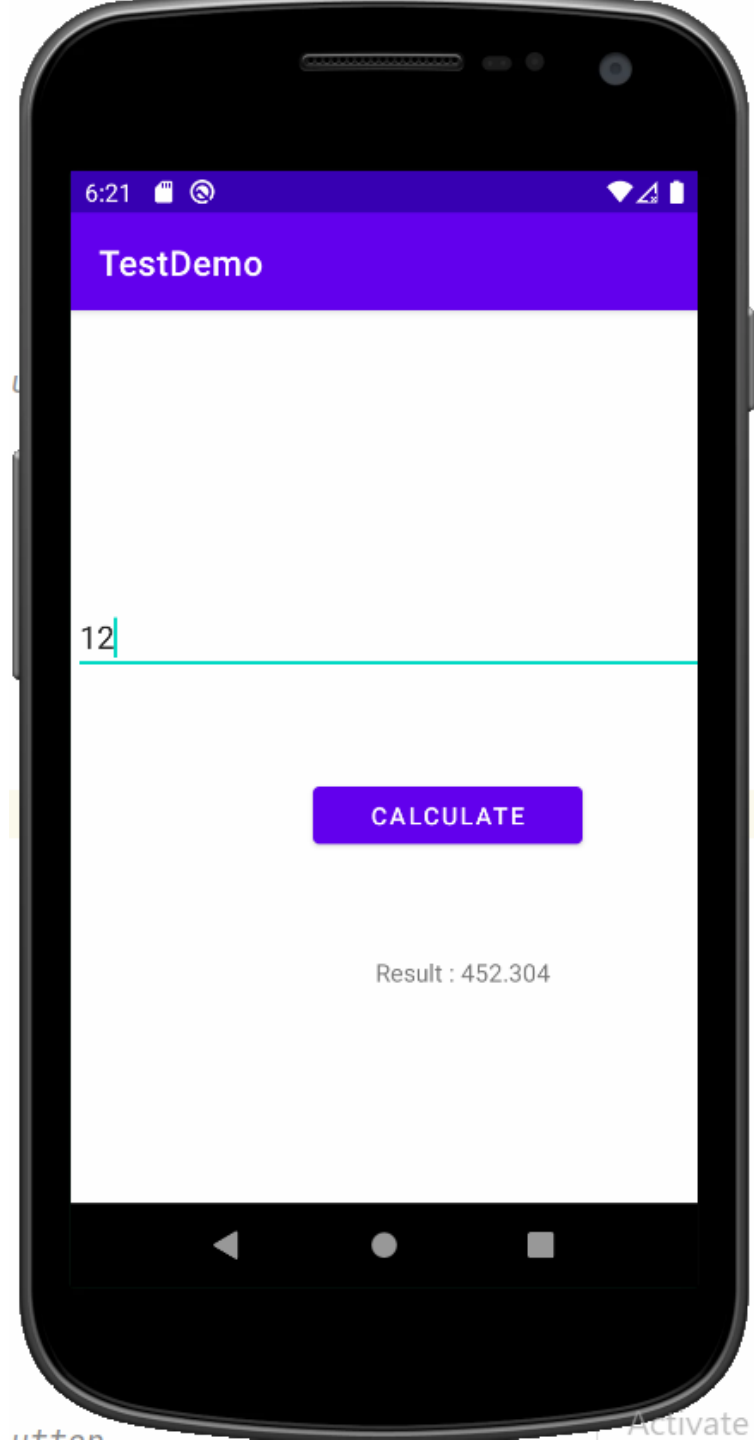
- Type conversion is a process in which one data type variable is converted into another data type.

```
1 package com.example.testdemo
2
3 fun main(){
4     var value1 = 10
5     val value2: Long = value1 //Compile error, type mismatch
6     print("$value2")
7 }
```

- The list of helper functions used for numeric conversion in Kotlin are:
 - toByte()
 - toShort()
 - toInt()
 - toLong()
 - toFloat()
 - toDouble()
 - toChar()

```
1 package com.example.testdemo
2
3 fun main(){
4     var value1 = 10
5     val value2: Long = value1.toLong()
6     print("$value2")
7 }
```

Exercise - 1





Kotlin Operators

- Set of operators to perform arithmetic, assignment, comparison operators and more.
- Arithmetic Operator

Expression	Function name	Translates to
$a + b$	plus	<code>a.plus(b)</code>
$a - b$	minus	<code>a.minus(b)</code>
$a * b$	times	<code>a.times(b)</code>
a / b	div	<code>a.div(b)</code>
$a \% b$	mod	<code>a.mod(b)</code>

```
1 package com.example.testdemo
2
3 fun main(){
4     val number1 = 12.5
5     val number2 = 3.5
6     var result: Double
7
8     result = number1 + number2
9     println("number1 + number2 = $result")
10
11    result = number1 - number2
12    println("number1 - number2 = $result")
13
14    result = number1 * number2
15    println("number1 * number2 = $result")
16
17    result = number1 / number2
18    println("number1 / number2 = $result")
19
20    result = number1 % number2
21    println("number1 % number2 = $result")
22 }
```



Kotlin Operators

- Assignment Operators

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

```
1 package com.example.testdemo
2
3 ▶ fun main(){
4     var number = 12
5
6     number *= 5    // number = number*5
7     println("number = $number")
8 }
```



Kotlin Operators

- Unary prefix and Increment / Decrement Operators

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 by 1	++a	a.inc()
--	Decrement: decreases value by 1	--a	a.dec()

```
1 package com.example.testdemo
2
3 fun main(){
4     val a = 1
5     val b = true
6     var c = 1
7
8     var result: Int
9     var booleanResult: Boolean
10
11     result = -a
12     println("-a = $result")
13
14     booleanResult = !b
15     println("!b = $booleanResult")
16
17     --c
18     println("--c = $c")
19 }
```

Kotlin Operators

- Comparison and Equality Operators

boolean → true
false

if else

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	$a != b$	$!(a?.equals(b) ?: (b === null))$

```
1 package com.example.testdemo
2
3 fun main(){
4     val a = -12
5     val b = 12
6     println(a < b)
7
8     // use of greater than operator
9     val max = if (a > b) {
10         println("a is larger than b.")
11     } else {
12         println("b is larger than a.")
13     }
14
15     println("max = $max")
16
17
18 }
```

↓
12



Kotlin Operators

- Logical Operators
 - There are two logical operators in Kotlin: || and &&

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)

```
1 package com.example.testdemo
2
3 fun main(){
4     val a = 10
5     val b = 9
6     val c = -1
7     var result: Boolean
8
9     // result is true if a is largest
10    result = (a<b) && (a>c) // result = (a>b) and (a>c)
11    println(result)
12    result = (a<b) || (a>c) // result = (a>b) or (a>c)
13    println(result)
14 }
```

if-else

Kotlin Operators

- in Operator → range

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



Operator	Expression	Translates to
<u>in</u>	a in b	b.contains(a)
<u>!in</u>	a !in b	!b.contains(a)

!

```
1 package com.example.testdemo
2
3 ▶ fun main(){
4     val numbers = arrayOf<Int>(1, 4, 42, -3)
5
6     if (4 in numbers) {
7         println("numbers array contains 4.")
8     }
9 }
```

Kotlin Operators

as

toString()
toInt()

- Type Casting Operator
- Explicit type casting can be done using :
 - Unsafe cast operator as
 - Manually, we use type cast operator as to cast a variable to target type.

```
1 package com.example.testdemo
2
3 fun main(){
4     val str1: String = "It works fine"
5     val str2: String = str1 as String // Works
6     println(str1)
7 }
```

- There might be possibility that we can not cast variable to target type and it throws an exception at runtime, thats why it is called as unsafe casting.

```
1 package com.example.testdemo
2
3 fun main(){
4     val str1: Any = 11
5     val str2: String = str1 as String // throw exception
6     println(str1)
7 }
```

Run: MainKt x

Exception in thread "main" java.lang.ClassCastException: class java.lang.Integer cannot be cast to class java.lang.String (java.lang.Integer is not assignable to java.lang.String)

at com.example.testdemo.MainKt.main(Main.kt:5)

at com.example.testdemo.MainKt.main(Main.kt)

- Safe cast operator: as?
 - If casting is not possible it returns null instead of throwing an ClassCastException exception.

```
1 package com.example.testdemo
2
3 fun main(){
4     val str1: Any = 11
5     val str2: String? = str1 as? String // prints null
6     println(str2)
7 }
```

Kotlin Control Flow – if - else Expression

- If is an expression which returns a value.
- Various type of if expression
 - if-else expression ✓
 - if-else if-else ladder expression
 - nested if expression
- The syntax of if...else is

```
1 package com.example.testdemo
2
3 fun main(){
4     val number = -10
5
6     val result = if (number > 0) {
7         "Positive number"
8     } else {
9         "Negative number"
10    }
11
12    println(result)
13 }
```

Handwritten notes: A red bracket on the left side of the code block, spanning from line 3 to line 13, is marked with a large red 'X'. Red checkmarks are placed next to the string literals "Positive number" and "Negative number". The value -10 in line 4 is circled in red.

- The syntax of if...else...if ladder is

```
1 package com.example.testdemo
2
3 fun main(){
4     val number = 0
5
6     val result = if (number > 0)
7         "positive number"
8     else if (number < 0)
9         "negative number"
10    else
11        "zero"
12
13    println("number is $result")
14 }
```

Handwritten notes: A red bracket on the left side of the code block, spanning from line 6 to line 11, is labeled "Ladder" in red. Red checkmarks are placed next to the string literals "positive number", "negative number", and "zero".

- The syntax of nested if is

```
1 package com.example.testdemo
2
3 fun main(){
4     val n1 = 3
5     val n2 = 5
6     val n3 = -2
7
8     val max = if (n1 > n2) {
9         if (n1 > n3)
10            n1
11        else
12            n3
13    } else {
14        if (n2 > n3)
15            n2
16        else
17            n3
18    }
19
20    println("max = $max")
21 }
```

Handwritten notes: A red bracket on the right side of the code block, spanning from line 8 to line 18, is labeled "Nested" in red. A red arrow points from the word "Nested" to the code block.

Kotlin Control Flow – **when** Expression

- The **when** in Kotlin is a replacement for switch Statement.
- It evaluates a section of code among many alternatives
- The syntax of when is

```
1 package com.example.testdemo
2
3 fun main(){
4     val a = 12
5     val b = 5
6
7     val operator = "*"
8
9     val result = when (operator) {
10         "+" -> a + b
11         "-" -> a - b
12         "*" -> a * b
13         "/" -> a / b
14         else -> "$operator operator is invalid operator."
15     }
16
17     println("result = $result")
18 }
```

Handwritten annotations on the left code block:

- A red circle around the `"*"` value of `operator`.
- A red arrow pointing from the circled `"*"` to the `"*" -> a * b` branch in the `when` expression.
- A red circle around the `a * b` expression.
- A red arrow pointing from the circled `a * b` to the handwritten text `60`.
- A red arrow pointing from the `when` expression to the handwritten text `executed`.

```
1 package com.example.testdemo
2
3 fun main(){
4     val n = 11
5
6     when (n) {
7         1, 2, 3 -> println("n is a positive integer less than 4.")
8         0 -> println("n is zero")
9         -1, -2 -> println("n is a negative integer greater than 3.")
10        in 1..10 -> println("A positive number less than 11.")
11        "11".toInt() -> println("Its 11")
12    }
13 }
```

Handwritten annotations on the right code block:

- A red circle around the value `11` assigned to `n`.
- A red bracket grouping the last two branches of the `when` expression: `in 1..10` and `"11".toInt()`.
- A red circle around the `else` branch (which is not explicitly labeled as such in the code, but the bracket implies it).
- A red arrow pointing from the circled `else` branch to the handwritten text `executed`.

