

---

# Kotlin Dasar

Eko Kurniawan Khannedy

---

# License

- Dokumen ini boleh Anda gunakan atau ubah untuk keperluan non komersial
- Tapi Anda wajib mencantumkan sumber dan pemilik dokumen ini
- Untuk keperluan komersial, silahkan hubungi pemilik dokumen ini

# Eko Kurniawan Khannedy

- Technical architect at one of the biggest ecommerce company in Indonesia
- 10+ years experiences
- [youtube.com/c/ProgrammerZamanNow](https://youtube.com/c/ProgrammerZamanNow)



---

# Pengenalan Kotlin

---

# Sejarah Kotlin

- Bahasa pemrograman Kotlin dikenalkan oleh Perusahaan JetBrains pada tahun 2011
- Awalnya Kotlin adalah bahasa pemrograman yang berjalan diatas JVM (Java Virtual Machine)
- Kotlin di desain agar terintegrasi dengan Java
- Tahun 2017, Google mengumumkan bahwa Kotlin adalah bahasa pemrograman yang direkomendasikan untuk pengembangan aplikasi Android

---

# Kenapa Belajar Kotlin?

- Java adalah bahasa pemrograman paling populer di dunia
- Ekosistem teknologi pendukung Java sudah sangat besar dan dewasa
- Kotlin adalah bahasa pemrograman yang dapat berjalan diatas JVM
- Bahasa pemrograman Kotlin lebih elegan dan sederhana dibanding Java
- Kotlin menjadi bahasa pemrograman utama untuk pengembangan aplikasi Android
- Spring (framework backend Java terpopuler) sekarang sudah mendukung Kotlin

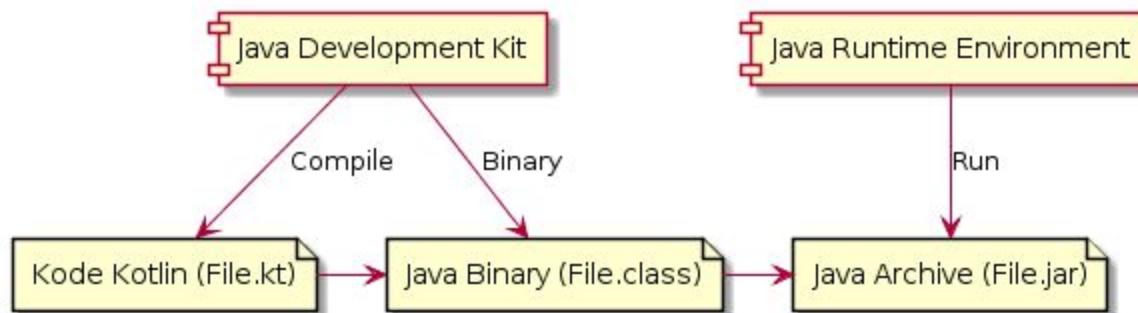
---

# Software Development Kit

- SDK adalah perangkat lunak yang digunakan untuk proses development
- SDK digunakan untuk melakukan kompilasi kode program Kotlin dan menjalankan kode program Kotlin
- Java Development Kit versi 8 keatas
- <https://jdk.java.net/>

---

# Proses Development Program Kotlin



---

# Integrated Development Environment

- IDE adalah smart editor yang digunakan untuk mengedit kode program Kotlin
- IDE juga digunakan untuk melakukan otomatisasi proses kompilasi kode program Kotlin dan otomatisasi proses menjalankan program Kotlin
- JetBrains IntelliJ IDEA
- <https://www.jetbrains.com/idea/>



# Instalasi JDK

- Windows : <https://medium.com/programmer-zaman-now/setting-java-path-di-windows-4da2c65d8298>
- Linux atau Mac :

```
# Add to .bashrc or .zshrc

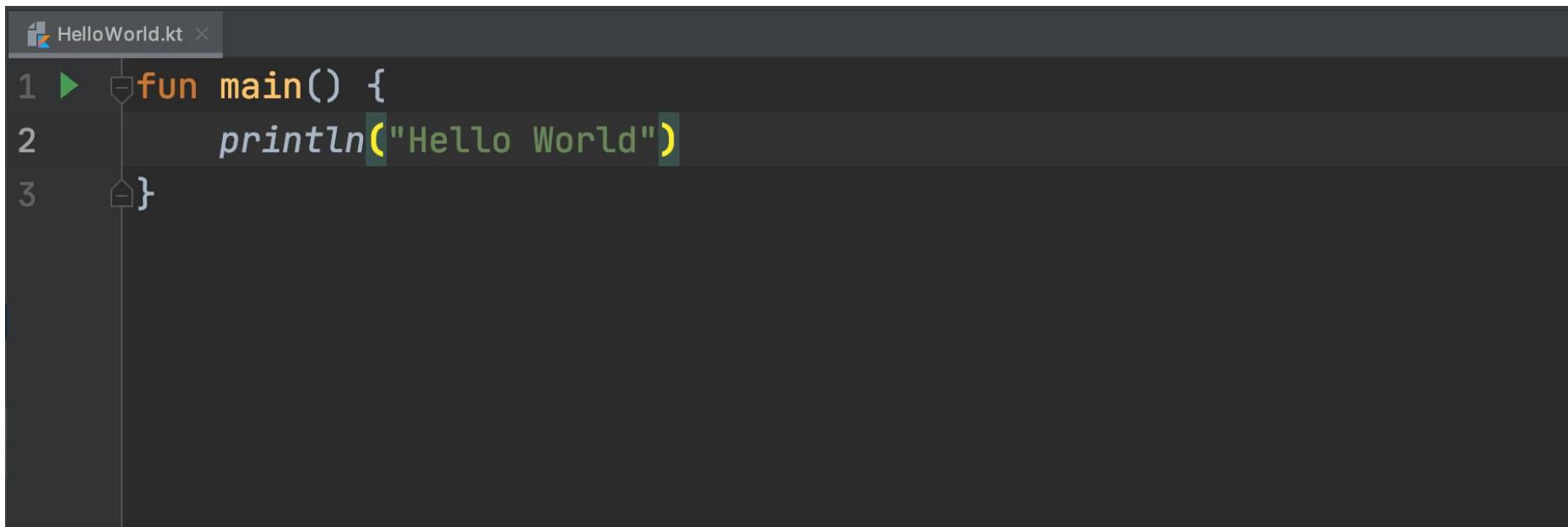
export JAVA_HOME="/Library/Java/JavaVirtualMachines/jdk1.8.0_241.jdk/Contents/Home"
export PATH="$JAVA_HOME/bin:$PATH"
```

---

# Program Hello World

---

# Kode : Hello World



A screenshot of a code editor window titled "HelloWorld.kt". The code editor displays the following Kotlin code:

```
1 > fun main() {  
2     println("Hello World")  
3 }
```

The code editor has a dark theme with syntax highlighting. The file tab shows "HelloWorld.kt" with a Kotlin icon. Line numbers 1, 2, and 3 are visible on the left. The code itself is in the main function body, with the println statement highlighted in green.

---

# Tipe Data Number

---

# Tipe Data Number

- Integer Number
- Floating Point Number

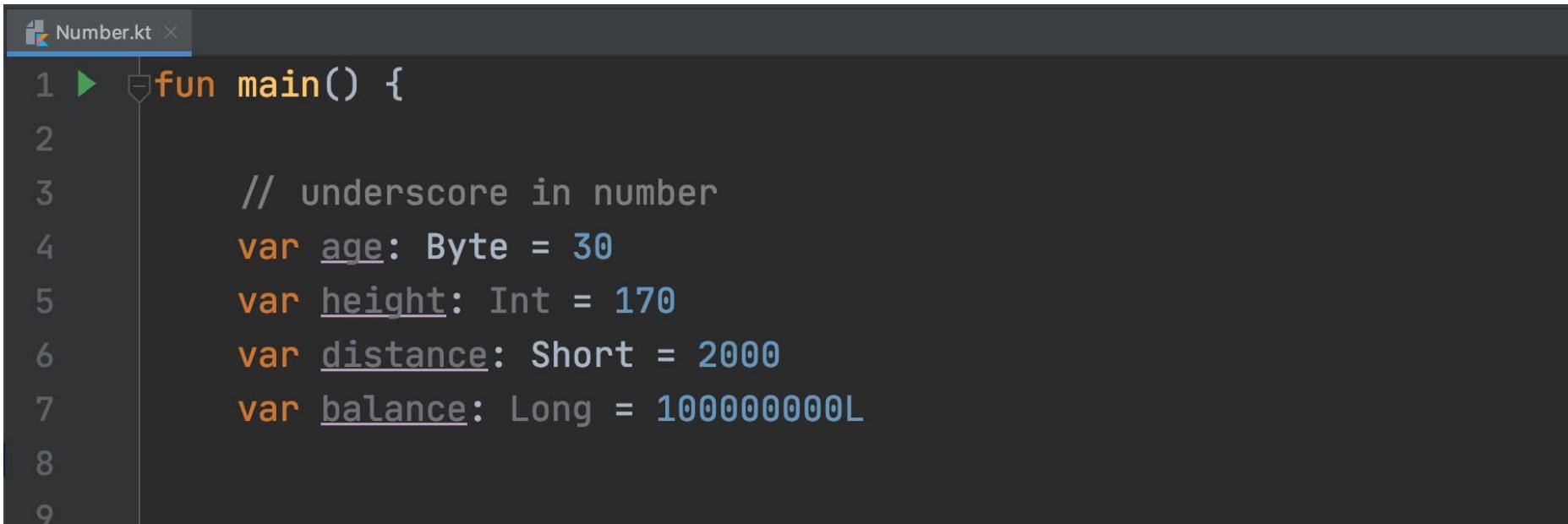


# Integer Number

Type	Size (bits)	Min value	Max value
Byte	8	-128	127
Short	16	-32768	32767
Int	32	$-2,147,483,648 (-2^{31})$	$2,147,483,647 (2^{31} - 1)$
Long	64	$-9,223,372,036,854,775,808 (-2^{63})$	$9,223,372,036,854,775,807 (2^{63} - 1)$



# Integer Number



A screenshot of a code editor showing a file named "Number.kt". The code defines a main function with several integer variables using underscores in their names:

```
1 fun main() {  
2  
3     // underscore in number  
4     var age: Byte = 30  
5     var height: Int = 170  
6     var distance: Short = 2000  
7     var balance: Long = 100000000L  
8  
9 }
```

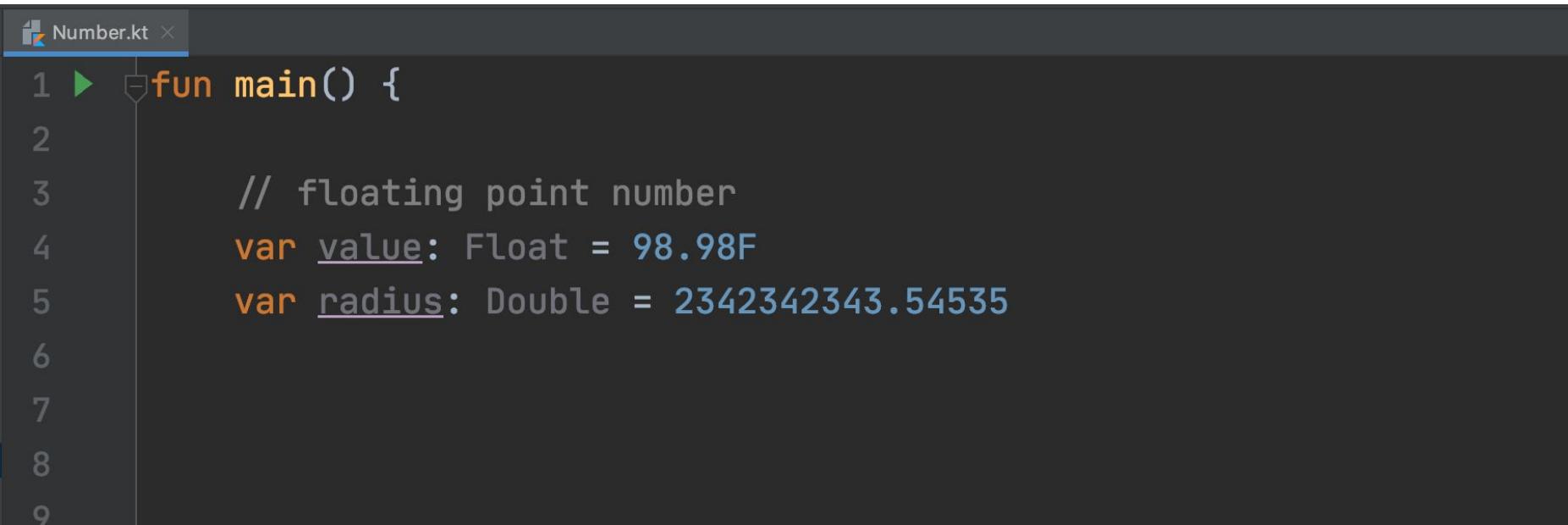


# Floating Point Number

Type	Size (bits)	Significant bits	Exponent bits	Decimal digits
Float	32	24	8	6-7
Double	64	53	11	15-16

---

# Floating Point Number

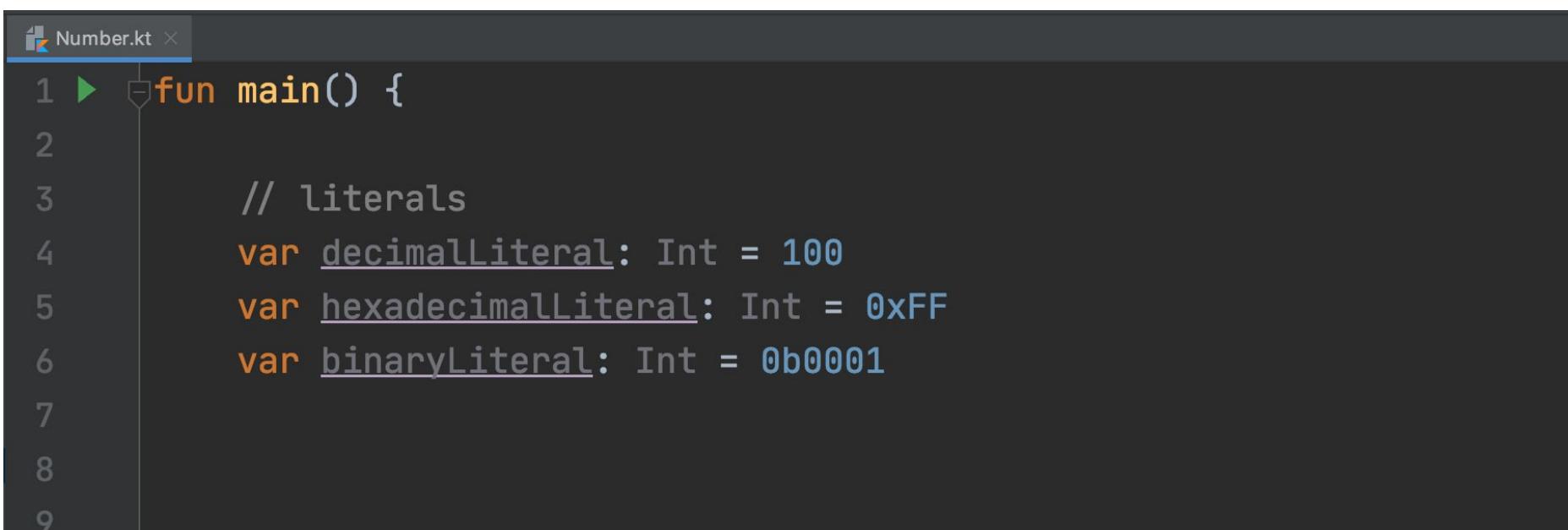


The screenshot shows a code editor window with a dark theme. The title bar says "Number.kt". The code area contains the following Kotlin code:

```
1 fun main() {  
2  
3     // floating point number  
4     var value: Float = 98.98F  
5     var radius: Double = 2342342343.54535  
6  
7  
8  
9}
```

The code defines a main function that declares two variables: `value` of type `Float` initialized to `98.98F`, and `radius` of type `Double` initialized to `2342342343.54535`. Lines 1 through 9 are numbered on the left.

# Literals

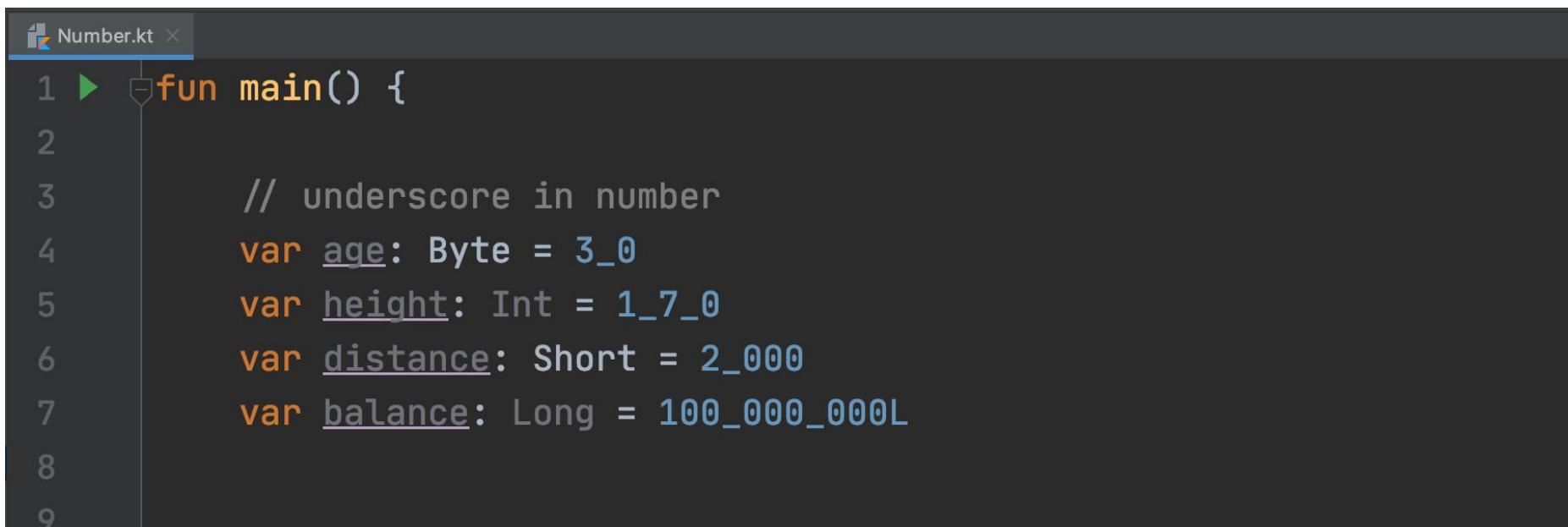


A screenshot of a code editor showing a file named "Number.kt". The code defines a main function with several integer literals:

```
1 fun main() {  
2  
3     // literals  
4     var decimalLiteral: Int = 100  
5     var hexadecimalLiteral: Int = 0xFF  
6     var binaryLiteral: Int = 0b0001  
7  
8  
9 }
```

---

# Underscore



A screenshot of a code editor showing a file named "Number.kt". The code contains several variables with underscores in their names:

```
1 fun main() {  
2  
3     // underscore in number  
4     var age: Byte = 3_0  
5     var height: Int = 1_7_0  
6     var distance: Short = 2_000  
7     var balance: Long = 100_000_000L  
8  
9 }
```

---

# Conversion

```
1 ► ↴ fun main() {  
2     var number: Int = 100  
3     // conversion  
4     var byte: Byte = number.toByte()  
5     var short: Short = number.toShort()  
6     var int: Int = number.toInt()  
7     var long: Long = number.toLong()  
8     var float: Float = number.toFloat()  
9     var double: Double = number.toDouble()  
10}
```

---

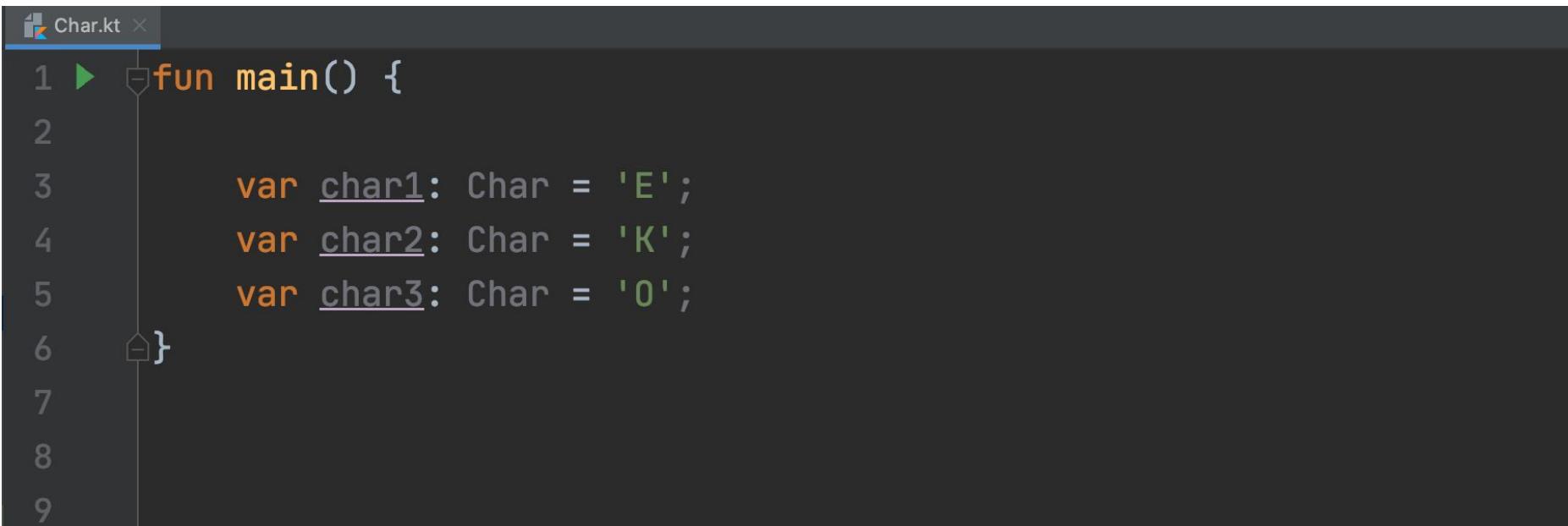
# Tipe Data Character

---

# Tipe Data Character

- Data karakter (huruf), di Kotlin direpresentasikan oleh tipe Char.
- Untuk membuat data Char, di Kotlin kita bisa menggunakan tanda ' (petik satu)

# Kode : Character



```
Char.kt
1 ►  fun main() {
2
3     var char1: Char = 'E';
4     var char2: Char = 'K';
5     var char3: Char = 'O';
6 }
```

The image shows a screenshot of a code editor with a dark theme. A file named "Char.kt" is open. The code within the file is as follows:

```
Char.kt
1 ►  fun main() {
2
3     var char1: Char = 'E';
4     var char2: Char = 'K';
5     var char3: Char = 'O';
6 }
```

The code defines a main function that declares three character variables: char1, char2, and char3, each initialized to the characters 'E', 'K', and 'O' respectively. The file has 9 lines of code, with lines 1 through 6 visible. Line 1 starts with a green play button icon followed by the function definition. Lines 3, 4, and 5 are preceded by orange underlined identifiers. Line 6 ends with a white brace icon.

---

# Tipe Data Boolean

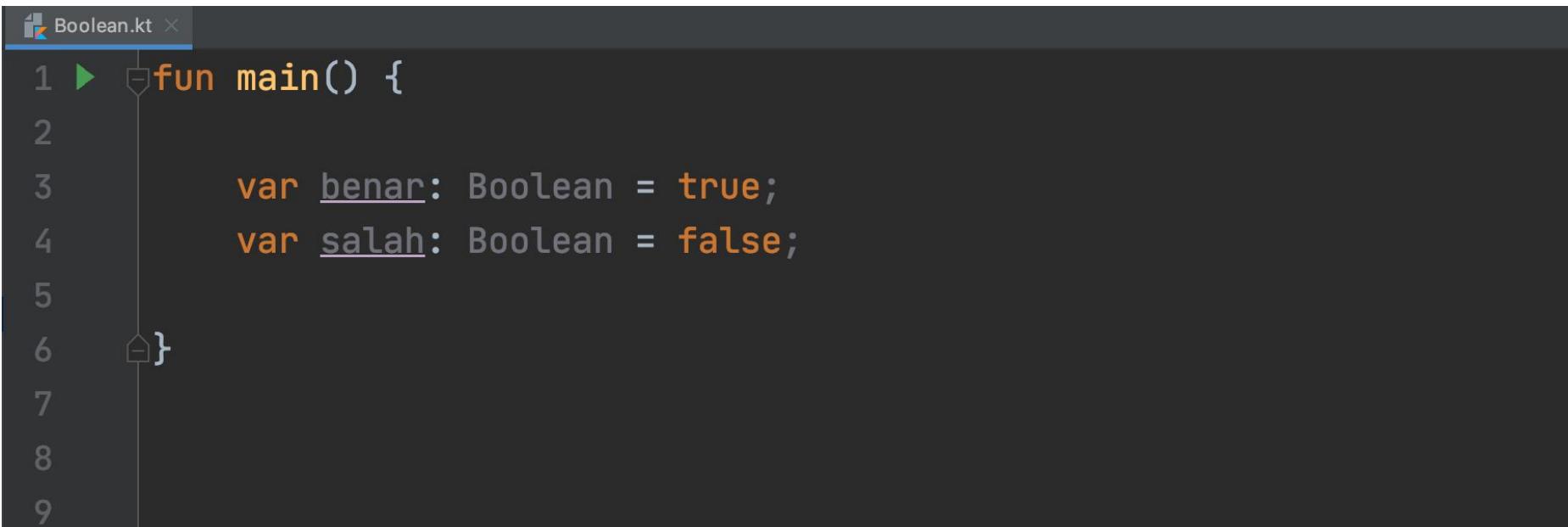
---

# Tipe Data Boolean

- Tipe data boolean adalah tipe data yang hanya memiliki 2 nilai, yaitu benar atau salah
- Tipe data boolean di kotlin direpresentasikan dengan kata kunci Boolean
- Nilai benar direpresentasikan dengan kata kunci true
- Nilai salah direpresentasikan dengan kata kunci false

---

# Kode : Boolean



A screenshot of a code editor window titled "Boolean.kt". The code editor shows the following Kotlin code:

```
1 ►  fun main() {  
2  
3     var benar: Boolean = true;  
4     var salah: Boolean = false;  
5  
6 }
```

The code defines a main function that declares two Boolean variables, `benar` and `salah`, both initialized to their respective boolean values.

---

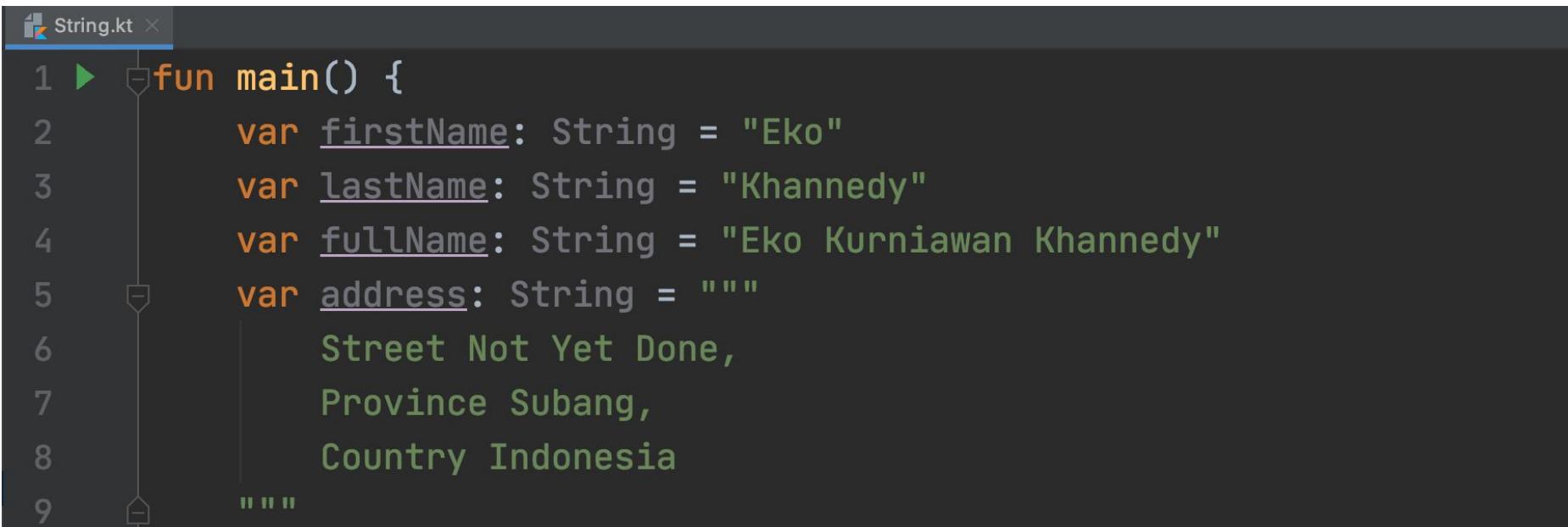
# Tipe Data String

---

# Tipe Data String

- Tipe data string adalah tipe data yang berisikan data kumpulan karakter atau sederhananya adalah teks.
- Di kotlin, tipe data string direpresentasikan dengan kata kunci String.
- Untuk membuat string di kotlin, kita bisa menggunakan
  - “ (tanda petik 2) untuk teks satu baris
  - “”” (tanda petik 2 sebanyak 3 kali) untuk teks lebih dari satu baris

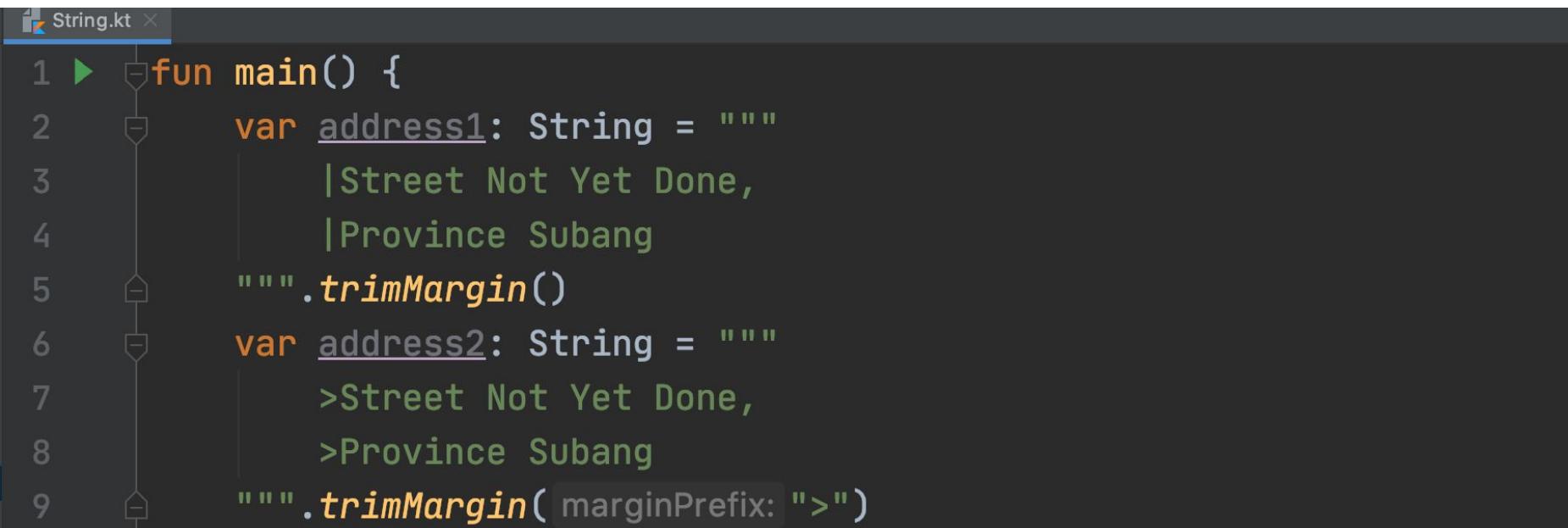
# Kode : String



```
String.kt
1 ►  ↴ fun main() {
2     var firstName: String = "Eko"
3     var lastName: String = "Khannedy"
4     var fullName: String = "Eko Kurniawan Khannedy"
5     var address: String = """
6         Street Not Yet Done,
7         Province Subang,
8         Country Indonesia
9     """
```

---

# Kode : String Trim Margin



The screenshot shows a code editor window with a dark theme. The file is named "String.kt". The code demonstrates two ways to use the `trimMargin` function on a multi-line string:

```
1 ► fun main() {
2     var address1: String = """
3         |Street Not Yet Done,
4         |Province Subang
5 """.trimMargin()
6     var address2: String = """
7         >Street Not Yet Done,
8         >Province Subang
9 """.trimMargin(marginPrefix: ">")
```

The first example uses the standard `trimMargin` function, which removes leading whitespace from each line. The second example uses the `trimMargin(marginPrefix: ">")` overload, which removes the specified prefix from each line.

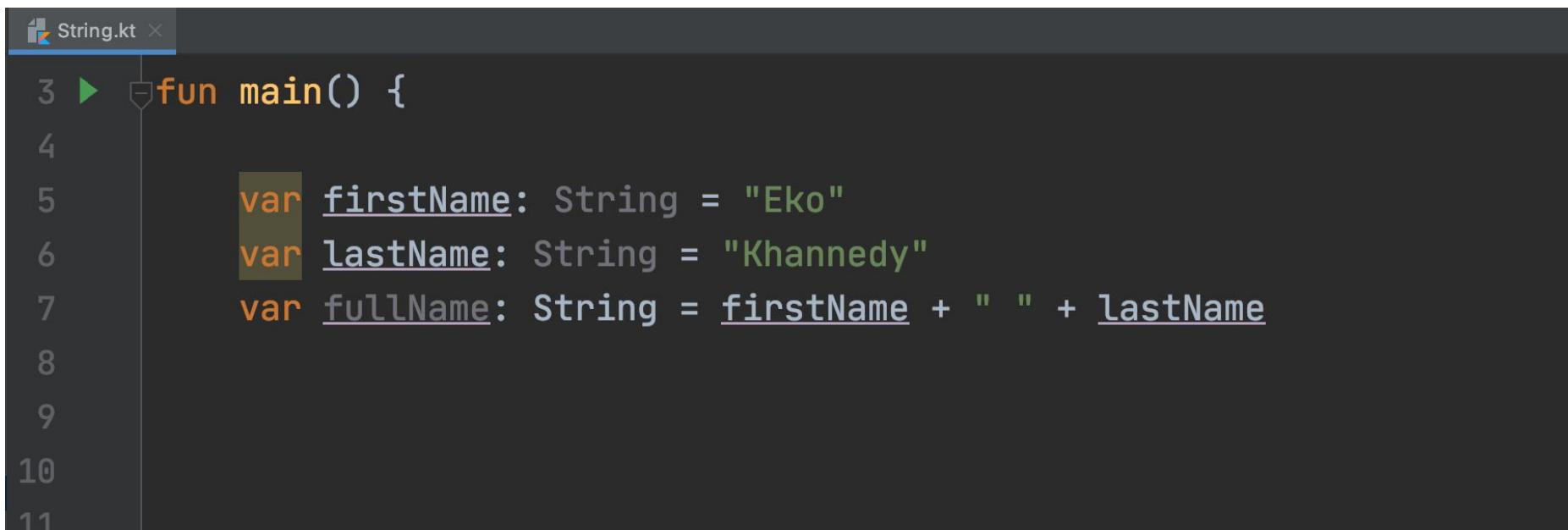
---

# Menggabungkan String

- Kadang kita butuh melakukan menggabungkan data String
- Untuk melakukan penggabungan data String, kita bisa menggunakan operator +

---

# Kode : Menggabungkan String



The screenshot shows a code editor window with a dark theme. The title bar says "String.kt". The code is as follows:

```
1 String.kt
2
3 fun main() {
4
5     var firstName: String = "Eko"
6     var lastName: String = "Khannedy"
7     var fullName: String = firstName + " " + lastName
8
9
10
11 }
```

The variables `firstName`, `lastName`, and `fullName` are highlighted with orange boxes. The code editor has a vertical scrollbar on the left.

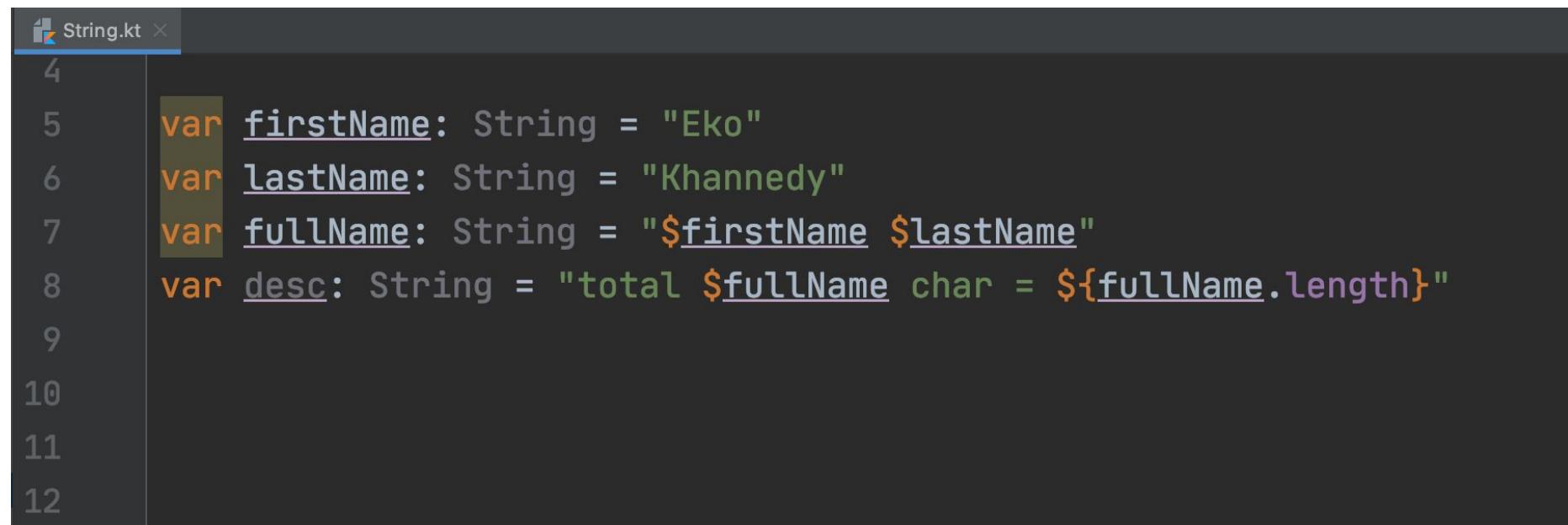
---

# String Template

- String template adalah kemampuan String di kotlin yang mendukung ekspresi template
- Dengan string template, secara otomatis kita bisa mengakses data dari luar teks string.
- \$ adalah tanda yang digunakan untuk template ekspresi sederhana, seperti mengakses variable lain
- \${ isi ekspresi } adalah tanda yang digunakan untuk template ekspresi yang kompleks

---

# Kode : String Template



```
String.kt ×
4
5  var firstName: String = "Eko"
6  var lastName: String = "Khannedy"
7  var fullName: String = "$firstName lastName"
8  var desc: String = "total fullName char = ${fullName.length}"
9
10
11
12
```

---

# Variable



# Variable

- Variable adalah tempat untuk menyimpan data
- Kotlin mendukung 2 jenis variabel; Mutable (bisa diubah) dan Immutable (tidak bisa diubah).
- Untuk membuat variable Mutable, di kotlin bisa menggunakan kata kunci `var`
- Untuk membuat variable Immutable, di kotlin bisa menggunakan kata kunci `val`

---

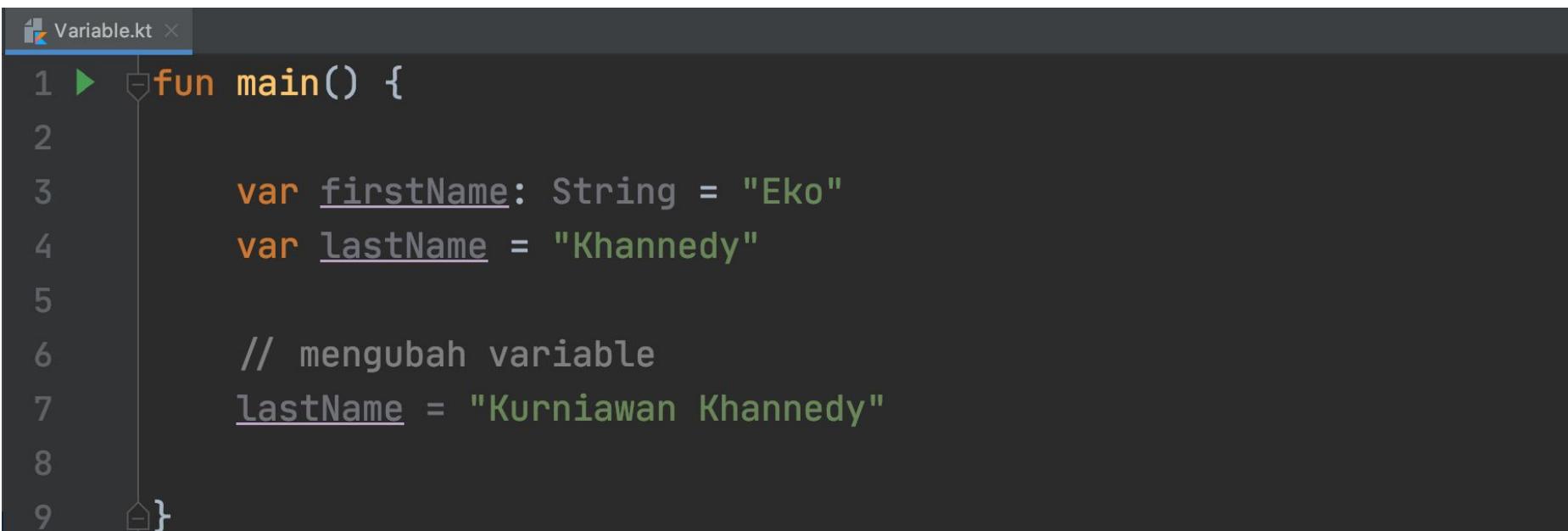
## Deklarasi Variable

val/var namaVariable : TipeData = data

**Direkomendasikan  
menggunakan Immutable  
dibanding Mutable data**

---

# Kode : Variable Mutable

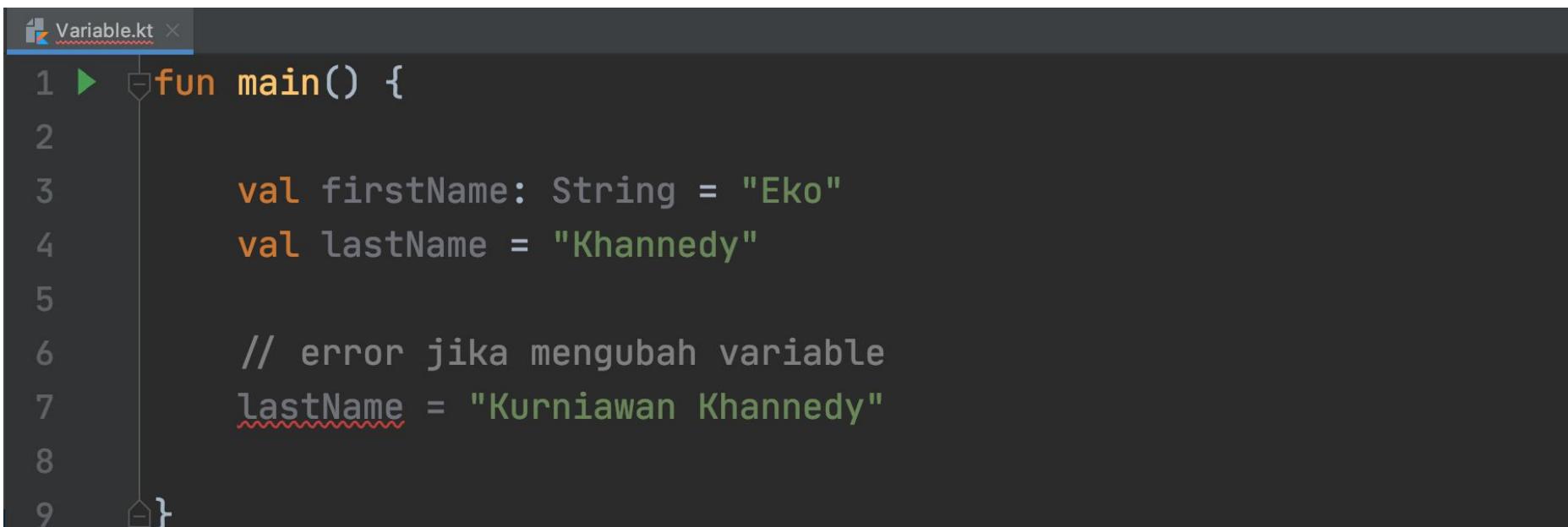


The image shows a screenshot of a code editor with a dark theme. The file is named "Variable.kt". The code defines a main function with two mutable variables: `firstName` and `lastName`. It then demonstrates changing the value of `lastName`.

```
1 fun main() {
2
3     var firstName: String = "Eko"
4     var lastName = "Khannedy"
5
6     // mengubah variable
7     lastName = "Kurniawan Khannedy"
8
9 }
```

---

# Kode : Variable Immutable



The image shows a screenshot of a code editor with a dark theme. The file is named "Variable.kt". The code defines a main function with two immutable variable declarations. It then attempts to change one of the variables, which results in a red underline error.

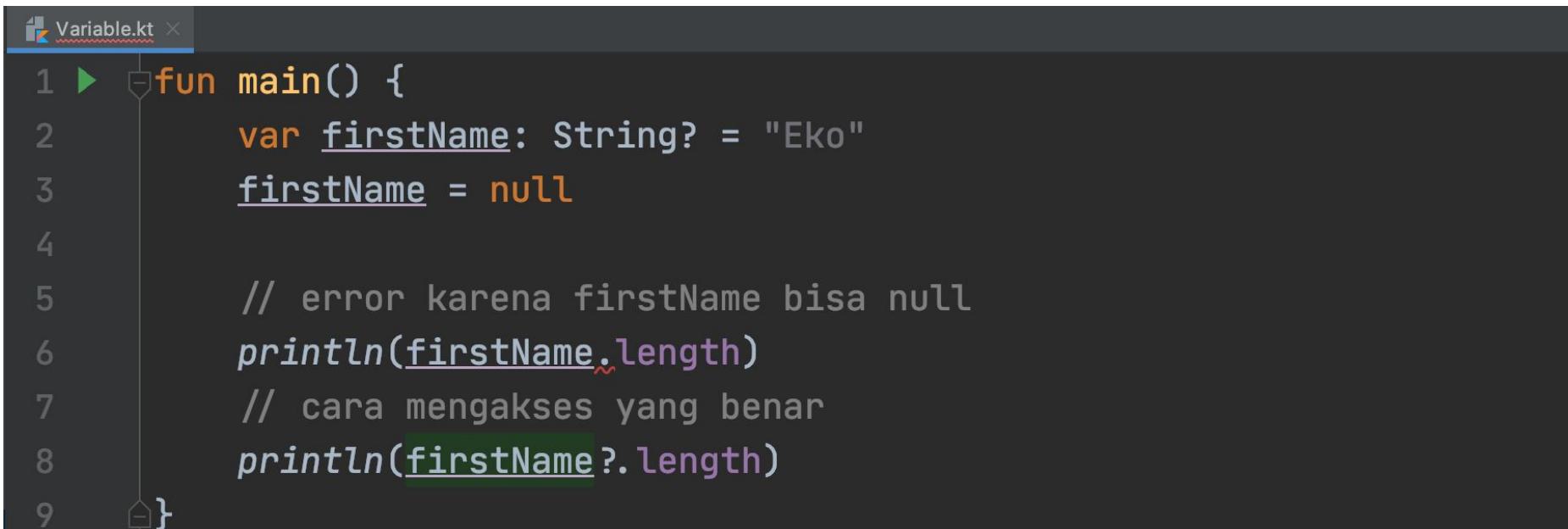
```
1 fun main() {  
2  
3     val firstName: String = "Eko"  
4     val lastName = "Khannedy"  
5  
6     // error jika mengubah variable  
7     lastName = "Kurniawan Khannedy"  
8  
9 }
```

---

# Nullable

- Secara standar, variable di Kotlin harus dideklarasikan / diinisialisasikan nilai nya
- Jika saat membuat variable, tidak diberi nilai, maka akan error
- Kotlin mendukung variable yang boleh null (tidak memiliki data)
- Ini dikarenakan Kotlin bisa mengakses Java, dan kebanyakan di Java, semua variable bisa null
- Untuk membuat variable bisa bernilai null, di Kotlin bisa menggunakan ? (tanda tanya) setelah tipe datanya.
- Penggunaan fitur ini tidak direkomendasikan untuk dilakukan di kotlin, hanya sebagai jalan akhir jika misal mengakses kode Java

## Kode : Nullable



The screenshot shows a code editor window with a dark theme. The title bar says "Variable.kt". The code is as follows:

```
1 fun main() {
2     var firstName: String? = "Eko"
3     firstName = null
4
5     // error karena firstName bisa null
6     println(firstName.length)
7     // cara mengakses yang benar
8     println(firstName?.length)
9 }
```

Line 1: `fun main()` {  
Line 2: `var firstName: String? = "Eko"`  
Line 3: `firstName = null`  
Line 4:  
Line 5: `// error karena firstName bisa null`  
Line 6: `println(firstName.length)`  
Line 7: `// cara mengakses yang benar`  
Line 8: `println(firstName?.length)`  
Line 9: }

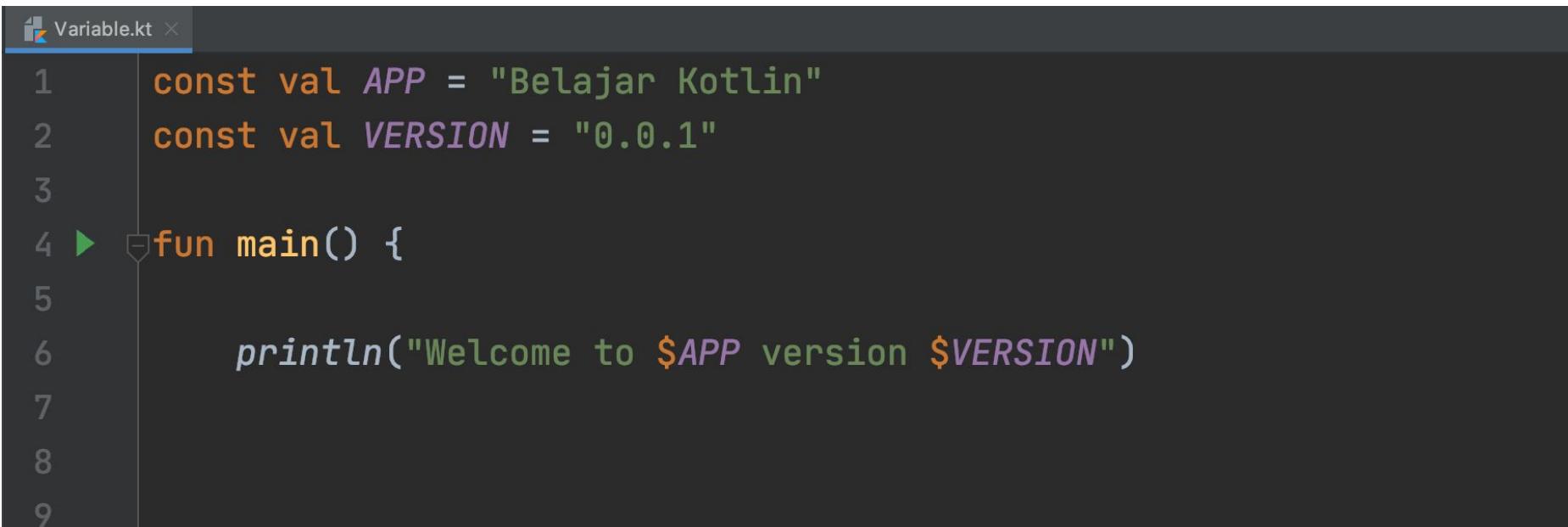
---

# Variable Constant

- Constant adalah Immutable data, yang biasanya diakses untuk keperluan global.
- Global artinya bisa diakses dimanapun
- Untuk menandai bahwa variable tersebut adalah constant, biasanya menggunakan UPPER\_CASE dalam pembuatan nama variable constant nya

---

# Kode : Variable Constant



The image shows a screenshot of a code editor with a dark theme. The file is named "Variable.kt". The code defines two constants: APP and VERSION, both set to strings. It then defines a main function that prints a welcome message using these constants.

```
1 const val APP = "Belajar Kotlin"
2 const val VERSION = "0.0.1"
3
4 ► fun main() {
5
6     println("Welcome to $APP version $VERSION")
7
8
9 }
```

---

# Tipe Data Array

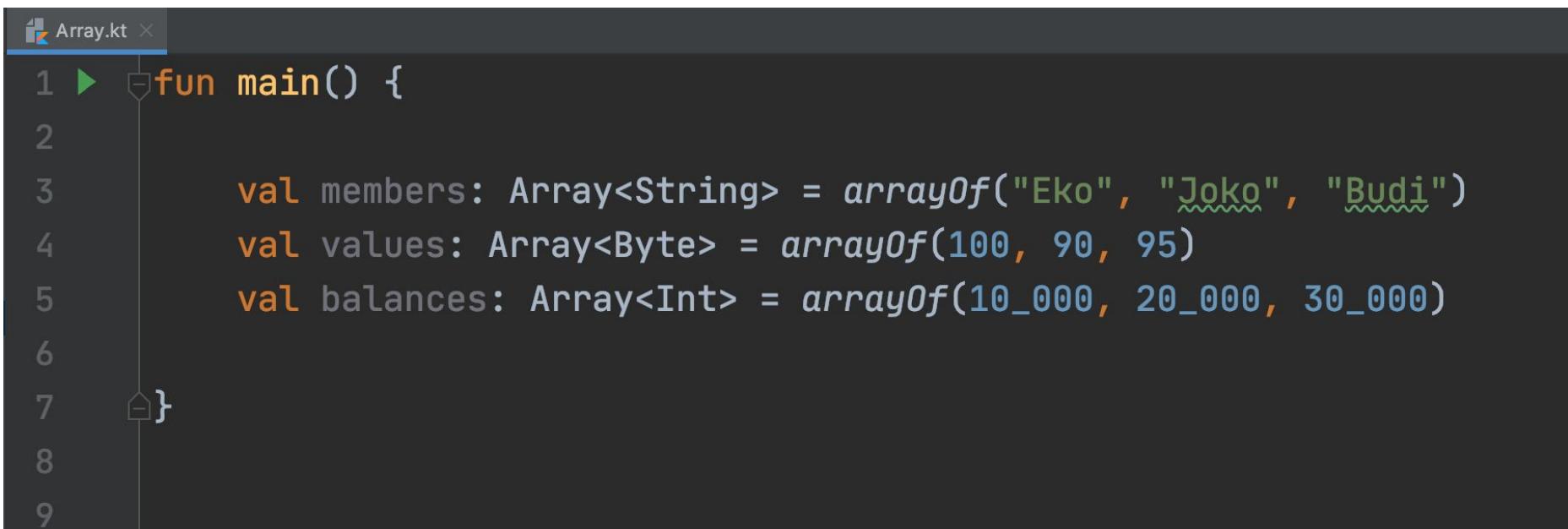
---

# Tipe Data Array

- Array adalah tipe data yang berisikan kumpulan data dengan tipe yang sama
- Tipe data array di Kotlin direpresentasikan dengan kata kunci Array

---

# Kode : Membuat Array



The image shows a screenshot of a code editor with a dark theme. The file is named "Array.kt". The code defines a main function that declares three arrays: "members" (an array of strings), "values" (an array of bytes), and "balances" (an array of integers). The code is numbered from 1 to 9 on the left side.

```
1 fun main() {  
2  
3     val members: Array<String> = arrayOf("Eko", "Joko", "Budi")  
4     val values: Array<Byte> = arrayOf(100, 90, 95)  
5     val balances: Array<Int> = arrayOf(10_000, 20_000, 30_000)  
6  
7 }  
8  
9
```



# Index di Array

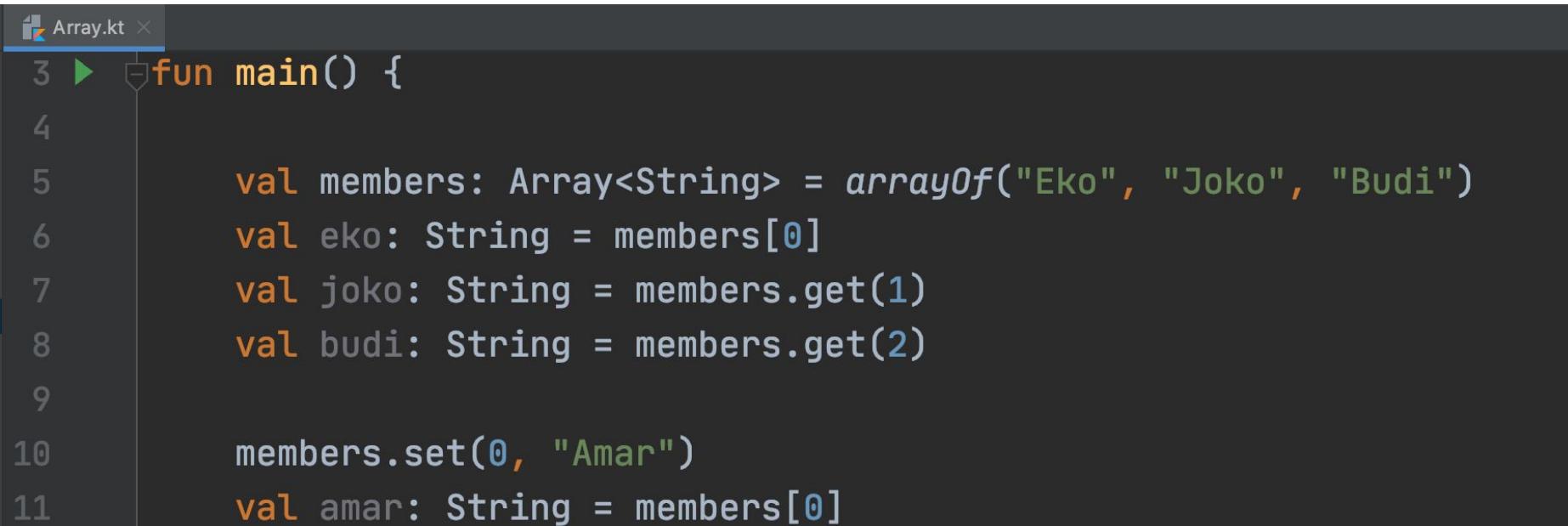
Index	Data
0	Eko
1	Kurniawan
2	Khannedy

---

# Operasi Array

Operasi	Keterangan
size	Untuk mendapatkan panjang Array
get(index)	Mendapat data di posisi index
[index]	Mendapat data di posisi index
set(index, value)	Mengubah data di posisi index
[index] = value	Mengubah data di posisi index

# Kode : Operasi Array



The screenshot shows a code editor window with a dark theme. The file is named "Array.kt". The code demonstrates various array operations in Kotlin:

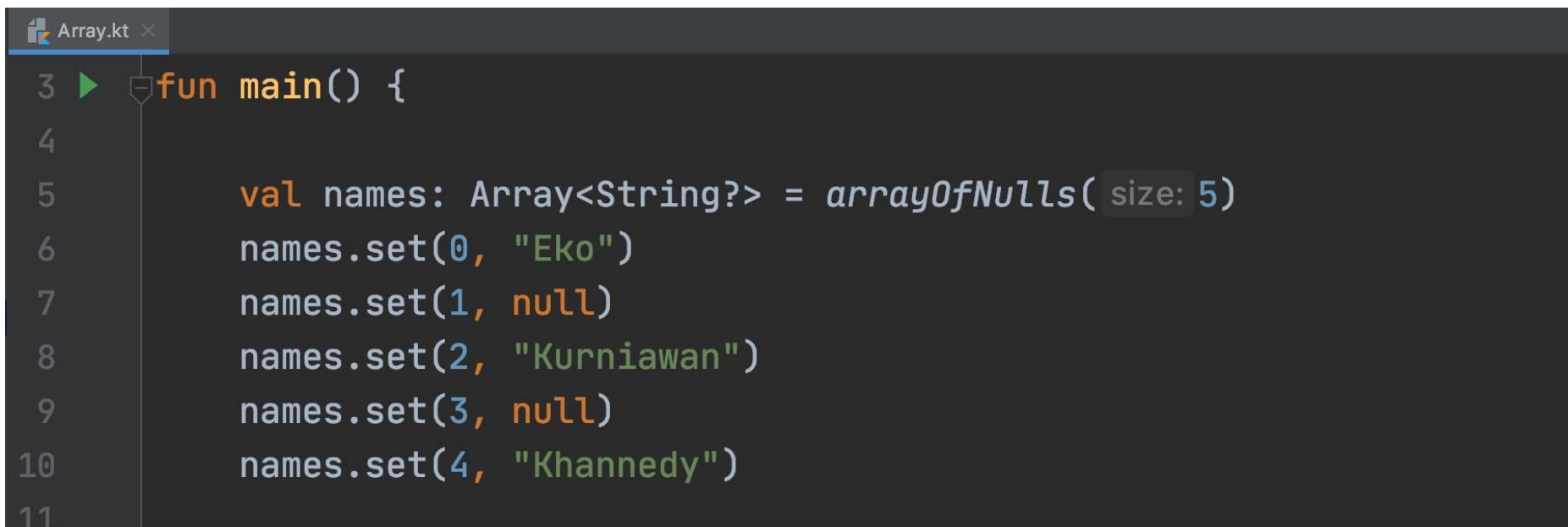
```
1 // Array.kt
2
3 fun main() {
4
5     val members: Array<String> = arrayOf("Eko", "Joko", "Budi")
6     val eko: String = members[0]
7     val joko: String = members.get(1)
8     val budi: String = members.get(2)
9
10    members.set(0, "Amar")
11    val amar: String = members[0]
```

---

# Array Nullable

- Secara standard data di Array di Kotlin tidak boleh null
- Jika kita butuh membuat Array yang datanya boleh null, kita bisa menggunakan ? (tanda tanya)

# Kode : Array Nullable



The screenshot shows a code editor window with a dark theme. The title bar says "Array.kt". The code is as follows:

```
1 // Array.kt
2
3 fun main() {
4
5     val names: Array<String?> = arrayOfNulls(size: 5)
6     names.set(0, "Eko")
7     names.set(1, null)
8     names.set(2, "Kurniawan")
9     names.set(3, null)
10    names.set(4, "Khannedy")
11}
```

---

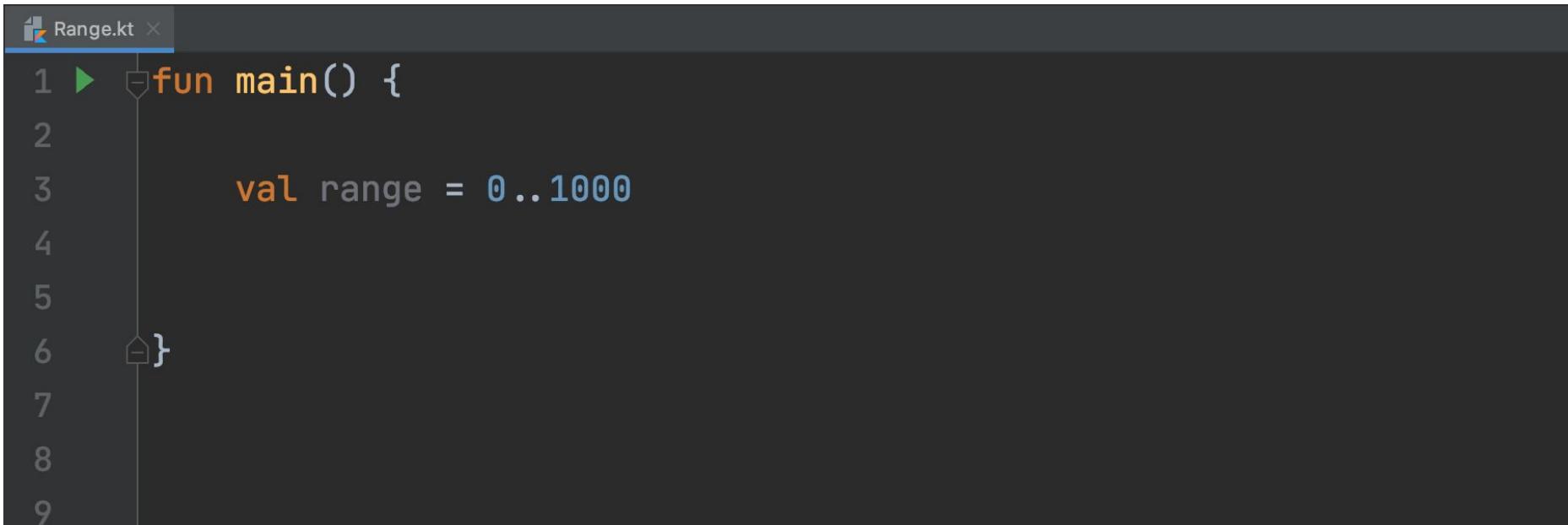
# Tipe Data Range

---

# Tipe Data Range

- Kadang kita ingin membuat array yang berisi data yang angka berurutan
- Membuat array dengan jumlah data sedikit mungkin mudah, tapi bagaimana jika data angka yang berurutannya sangat banyak, misal dari 1 sampai 1000
- Kotlin mendukung tipe data range, yang digunakan untuk kebutuhan seperti ini
- Cara membuat range di Kotlin sangat mudah cukup menggunakan tanda .. (titik dua kali) :
  - 0..10 : Range dari 0 sampai 10
  - 1..100 : Range dari 1 sampai 100

# Kode : Range



```
Range.kt
1 ►  fun main() {
2
3     val range = 0..1000
4
5
6     }
7
8
9
```

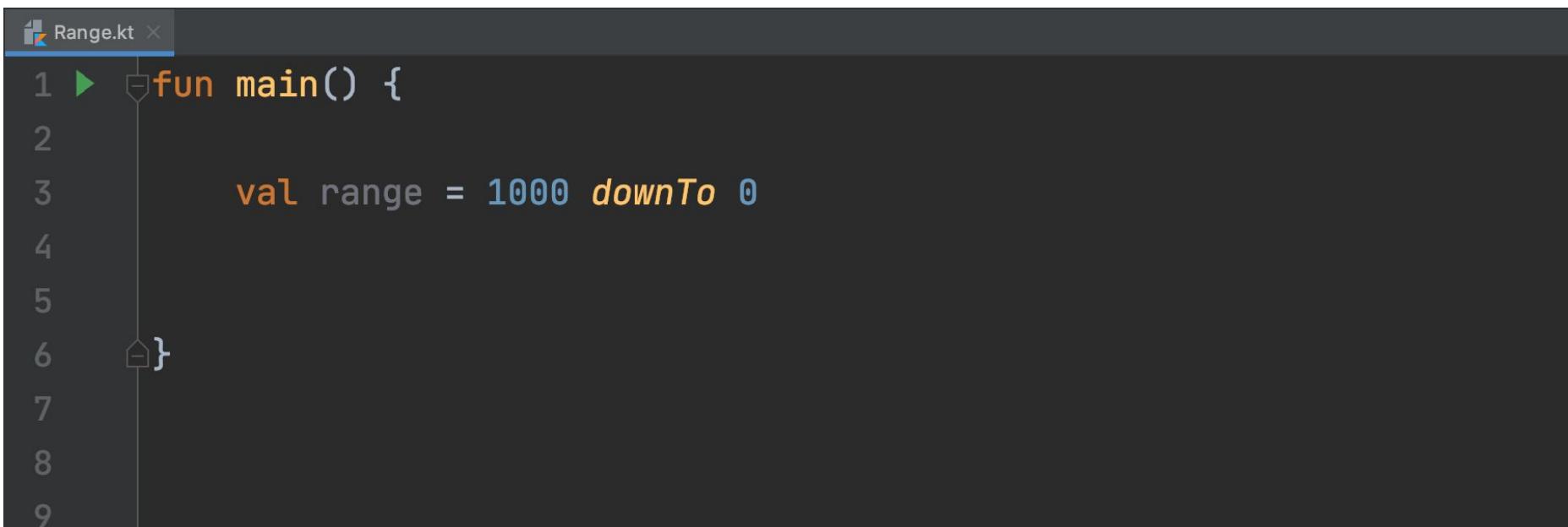
---

# Operasi Range

Operasi	Keterangan
count()	Mendapatkan total data di range
contains(value)	Mengecek apakah terdapat value tersebut
first	Mendapatkan nilai pertama
last	Mendapatkan nilai terakhir
step	Mendapatkan nilai tiap kenaikan

---

# Kode : Range Terbalik



```
Range.kt
1 ►  fun main() {
2
3     val range = 1000 downTo 0
4
5
6     }
7
8
9
```

The image shows a screenshot of a code editor with a dark theme. The title bar says "Range.kt". The code editor displays the following Kotlin code:

```
fun main() {
    val range = 1000 downTo 0
}
```

The code defines a main function that declares a variable range with a value of 1000, using the `downTo` operator to create an inverted range from 1000 down to 0.

---

# Kode : Range Dengan Step



The screenshot shows a code editor window with a dark theme. The file is named "Range.kt". The code defines a main function that creates two ranges: "range1" from 0 to 1000 with a step of 5, and "range2" from 1000 down to 0 with a step of 5.

```
Range.kt
1 fun main() {
2
3     val range1 = 0..1000 step 5
4     val range2 = 1000 downTo 0 step 5
5
6
7 }
```

---

# Operasi Matematika

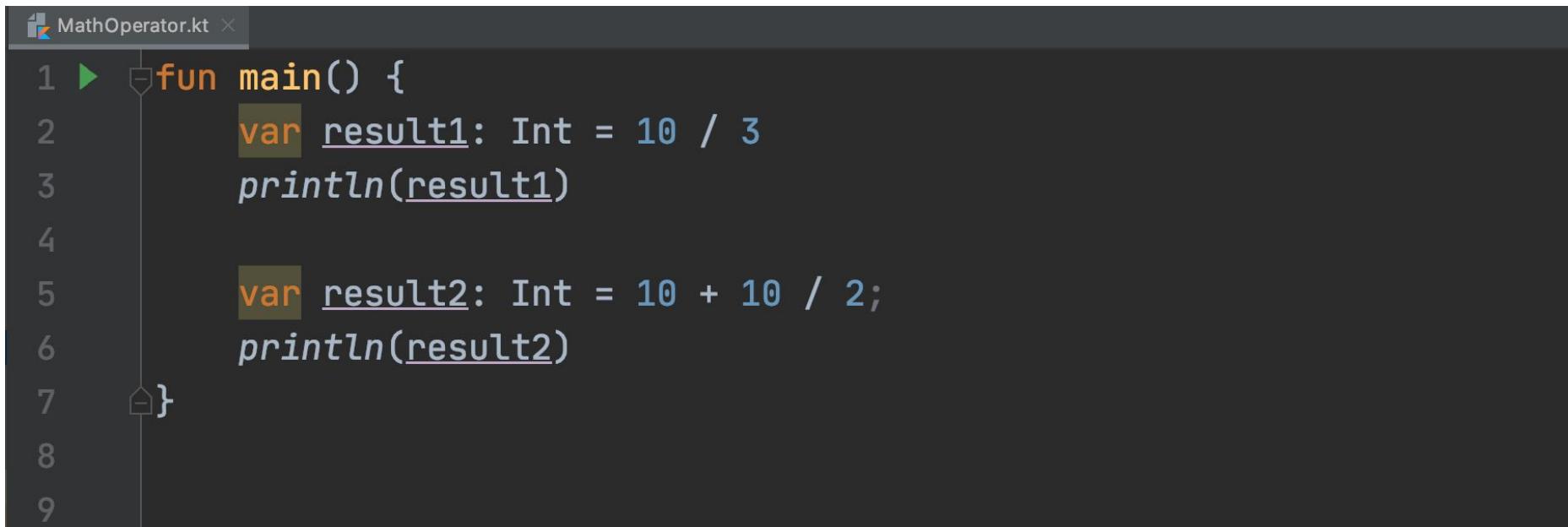
---

# Operasi Matematika

Operator	Keterangan
+	Penjumlahan
-	Pengurangan
*	Perkalian
/	Pembagian
%	Sisa Pembagian

---

# Contoh Operasi Matematika



```
MathOperator.kt ✘
1 ►  ↴ fun main() {
2     var result1: Int = 10 / 3
3     println(result1)
4
5     var result2: Int = 10 + 10 / 2;
6     println(result2)
7
8
9 }
```

---

# Augmented Assignments

Operasi Matematika	Augmented Assignments
$a = a + 10$	$a += 10$
$a = a - 10$	$a -= 10$
$a = a * 10$	$a *= 10$
$a = a / 10$	$a /= 10$
$a = a \% 10$	$a \%= 10$

---

# Unary Operator

Operator	Keterangan
<code>++</code>	$a = a + 1$
<code>--</code>	$a = a - 1$
<code>-</code>	Negative
<code>+</code>	Positive
<code>!</code>	Boolean kebalikan

---

# Operasi Perbandingan

---

# Operasi Perbandingan

- Operasi perbandingan adalah operasi untuk membandingkan dua buah data
- Operasi perbandingan adalah operasi yang menghasilkan nilai boolean (benar atau salah)
- Jika hasil operasinya adalah benar, maka nilainya adalah true
- Jika hasil operasinya adalah salah, maka nilainya adalah false

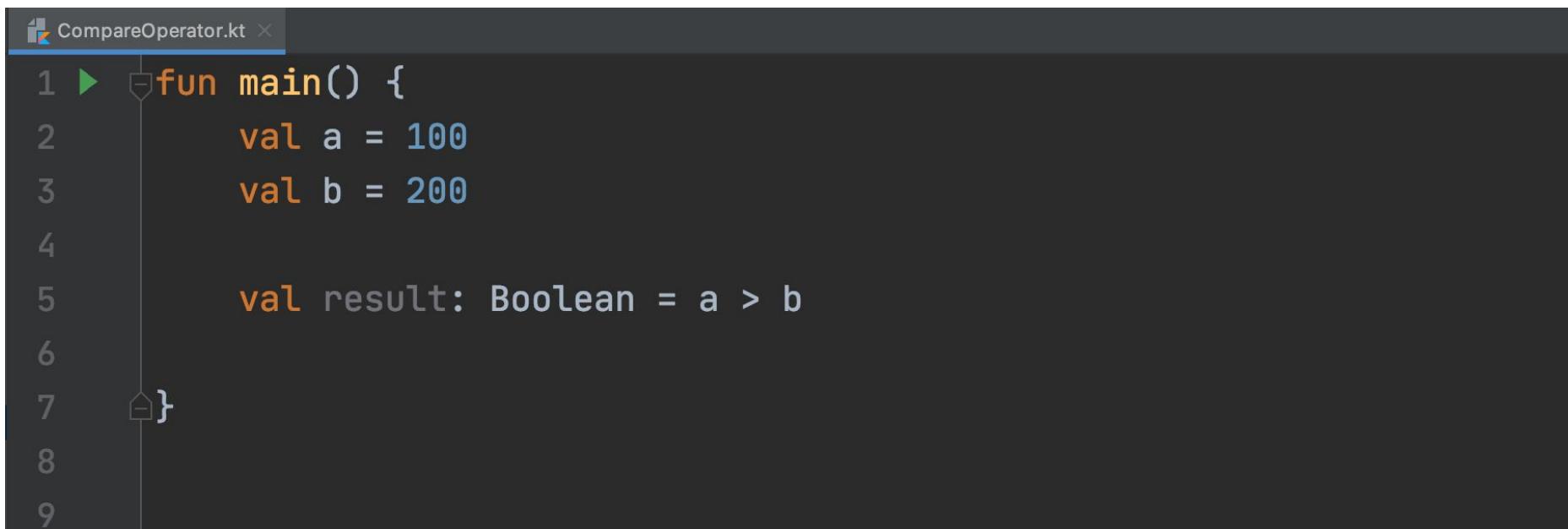
---

# Operator Perbandingan

Operator	Keterangan
>	Lebih Dari
<	Kurang Dari
>=	Lebih Dari Sama Dengan
<=	Kurang Dari Sama Dengan
==	Sama Dengan
!=	Tidak Sama Dengan

---

# Kode : Operator Perbandingan



```
CompareOperator.kt ✘
1 ►  ↴ fun main() {
2     val a = 100
3     val b = 200
4
5     val result: Boolean = a > b
6
7     }
8
9
```

---

# Operasi Boolean

---

# Operator Logika

- Operator logika adalah operator untuk dua buah data boolean
- Hasil dari operator logika adalah boolean lagi

---

# Operasi Boolean

Operator	Keterangan
&&	Dan
	Atau
!	Kebalikan

---

# Operasi &&

Nilai 1	Operator	Nilai 2	Hasil
true	&&	true	true
true	&&	false	false
false	&&	true	false
false	&&	false	false

---

# Operasi ||

Nilai 1	Operator	Nilai 2	Hasil
true		true	true
true		false	true
false		true	true
false		false	false

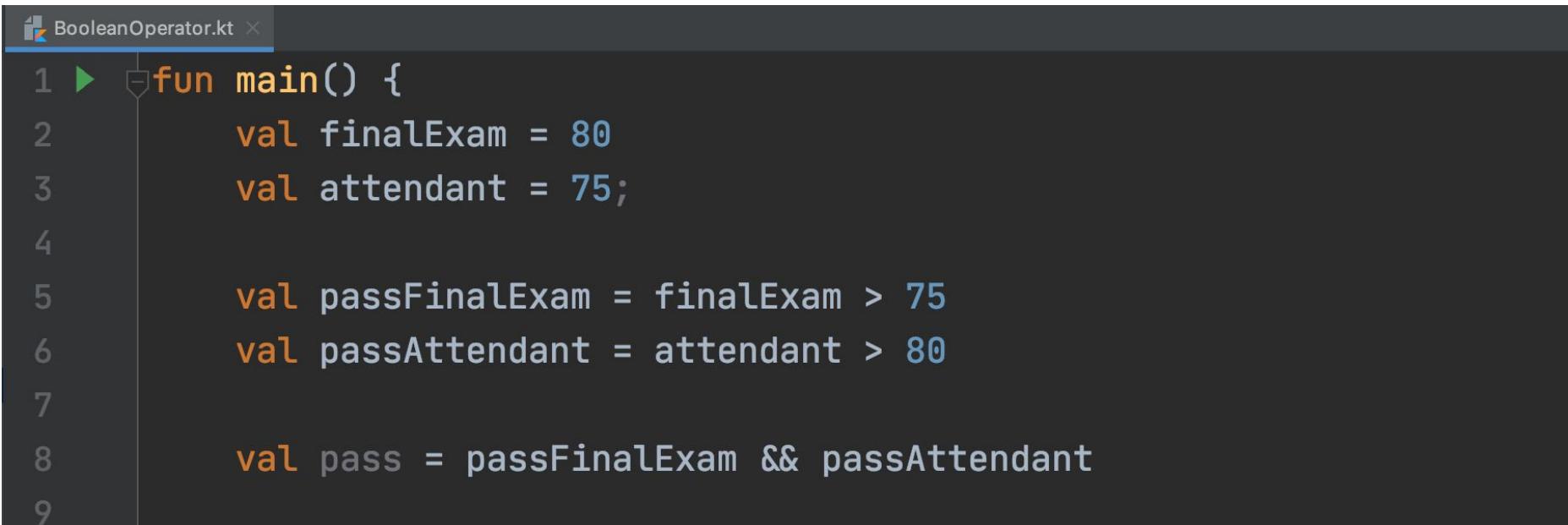
---

# Operasi !

Operator	Nilai 2	Hasil
!	true	false
!	false	true



# Kode Operasi Boolean



The image shows a screenshot of a code editor with a dark theme. The title bar of the window is labeled "BooleanOperator.kt". The code itself is written in Kotlin and defines a main function that declares two variables, finalExam and attendant, and then calculates passFinalExam and passAttendant based on their values. Finally, it calculates a pass variable using the logical AND operator (&&). The code is numbered from 1 to 9 on the left side.

```
1 fun main() {
2     val finalExam = 80
3     val attendant = 75;
4
5     val passFinalExam = finalExam > 75
6     val passAttendant = attendant > 80
7
8     val pass = passFinalExam && passAttendant
9 }
```

---

# If Expression

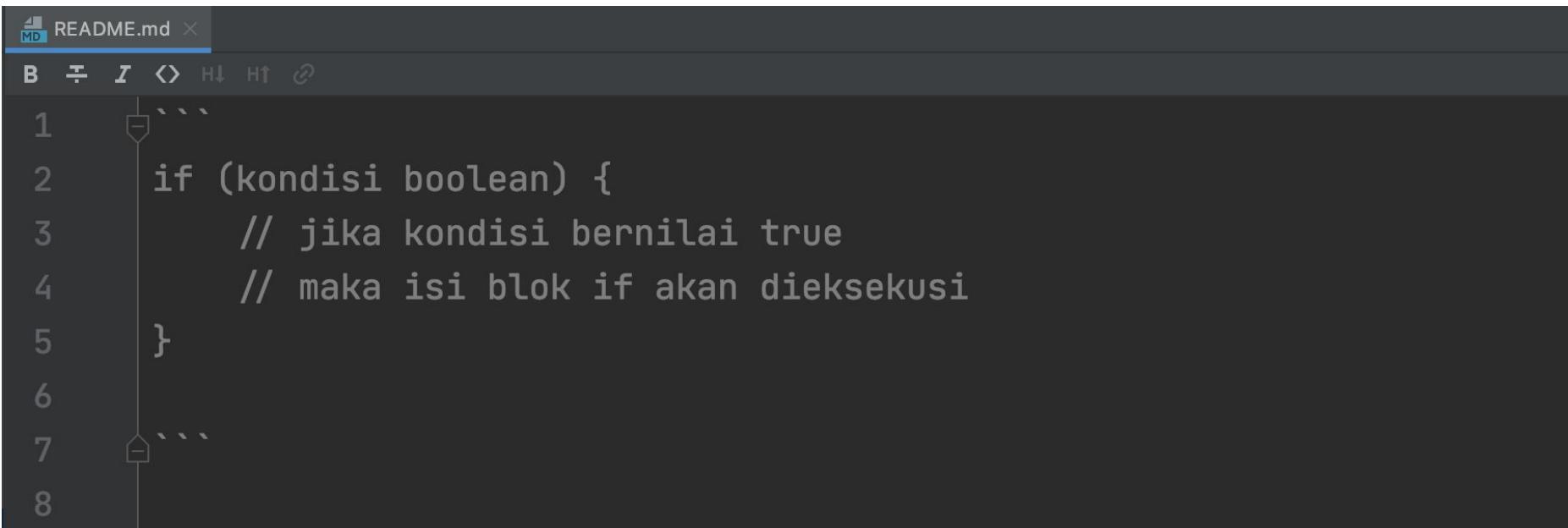
---

# If Expression

- Dalam Kotlin, if adalah salah satu kata kunci yang digunakan untuk percabangan
- Percabangan artinya kita bisa mengeksekusi kode program tertentu ketika suatu kondisi terpenuhi
- Hampir di semua bahasa pemrograman mendukung if expression



# If Expression



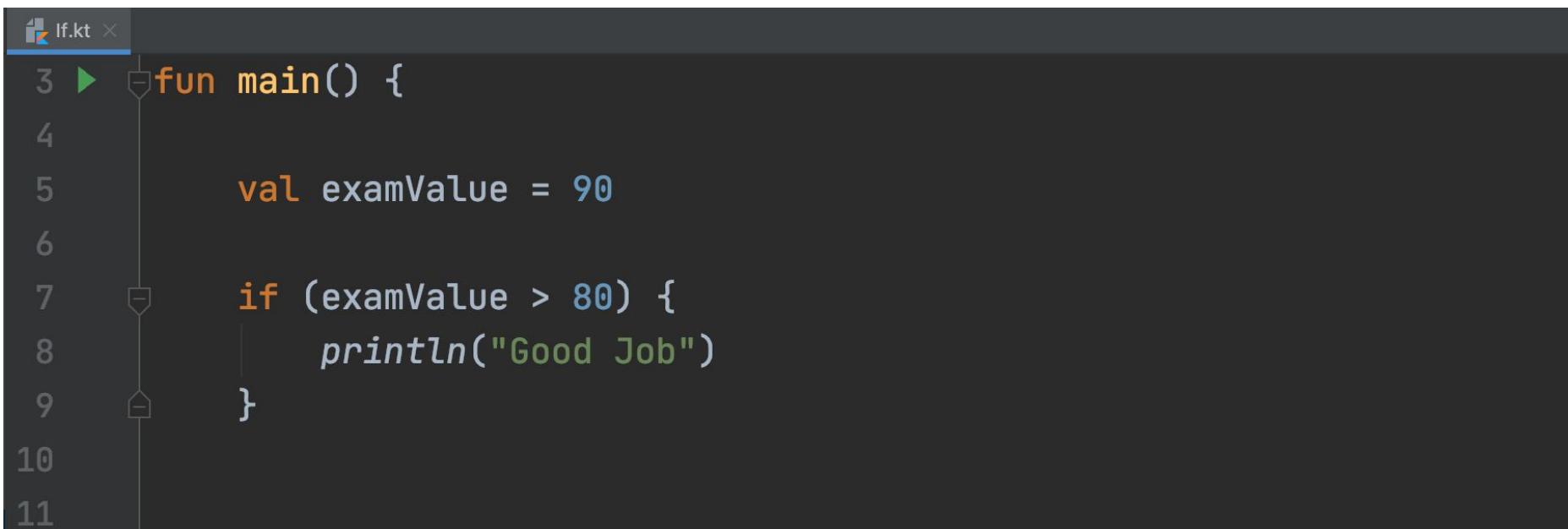
The screenshot shows a code editor window with a dark theme. The title bar says "README.md". The toolbar includes icons for bold, italic, code, and other document operations. The code area displays the following text:

```
1  if (kondisi boolean) {
2      // jika kondisi bernilai true
3      // maka isi blok if akan dieksekusi
4
5  }
6
7
8
```

Line 1 has a small icon above it, likely indicating a collapsed section or a specific code folding state. Lines 2 and 3 contain explanatory comments. Line 5 ends with a closing brace. Lines 7 and 8 are blank.



# Kode : If Expression



A screenshot of an IDE showing a Kotlin file named `If.kt`. The code contains a main function that prints a message based on a value:

```
1 If.kt ×
2
3 fun main() {
4
5     val examValue = 90
6
7     if (examValue > 80) {
8         println("Good Job")
9     }
10
11 }
```

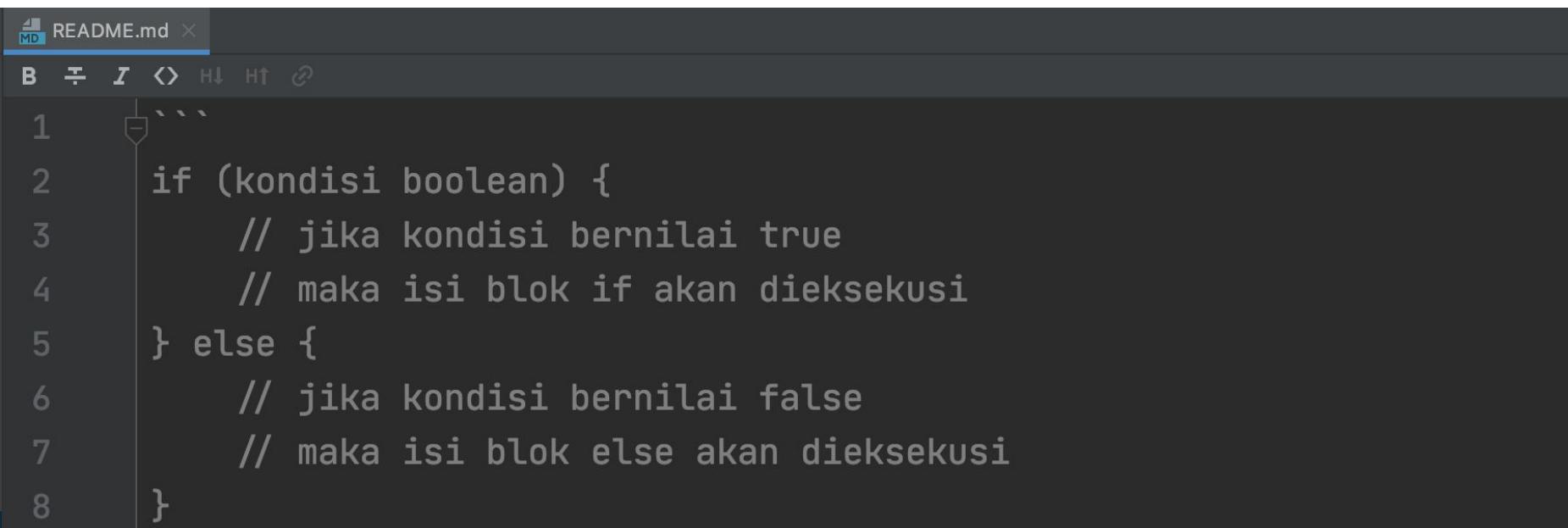
The code is annotated with several small icons: a green triangle at line 3, a yellow open bracket at line 4, a yellow open brace at line 7, a yellow minus sign at line 8, and a yellow close brace at line 9.

---

# Else Expression

- Blok if akan dieksekusi ketika kondisi if bernilai true
- Kadang kita ingin melakukan eksekusi program tertentu jika kondisi if bernilai false
- Hal ini bisa dilakukan menggunakan else expression

# Else Expression



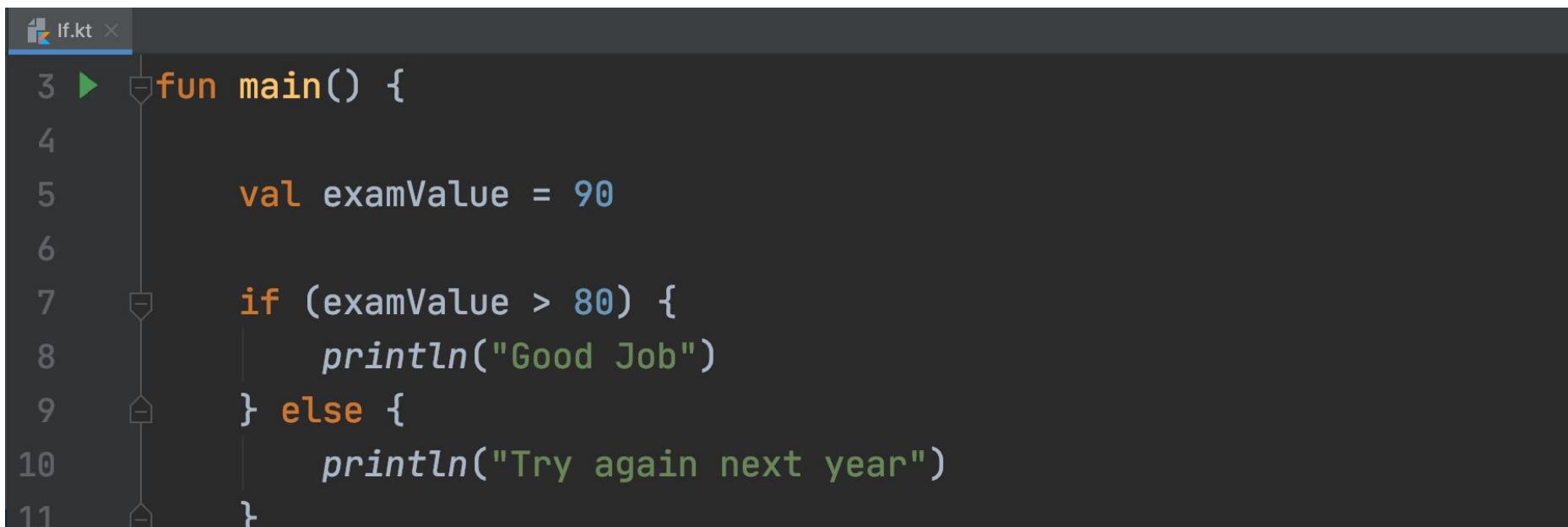
The image shows a screenshot of a code editor with a dark theme. The title bar says "README.md". The code area contains the following Java code:

```
1  if (kondisi boolean) {
2      // jika kondisi bernilai true
3      // maka isi blok if akan dieksekusi
4  } else {
5      // jika kondisi bernilai false
6      // maka isi blok else akan dieksekusi
7
8 }
```

The first line of code (if) has a small icon with a minus sign and a plus sign next to it, indicating it can be collapsed or expanded. The code is written in Indonesian, explaining the purpose of the if-else statement.

---

# Kode : Else Expression



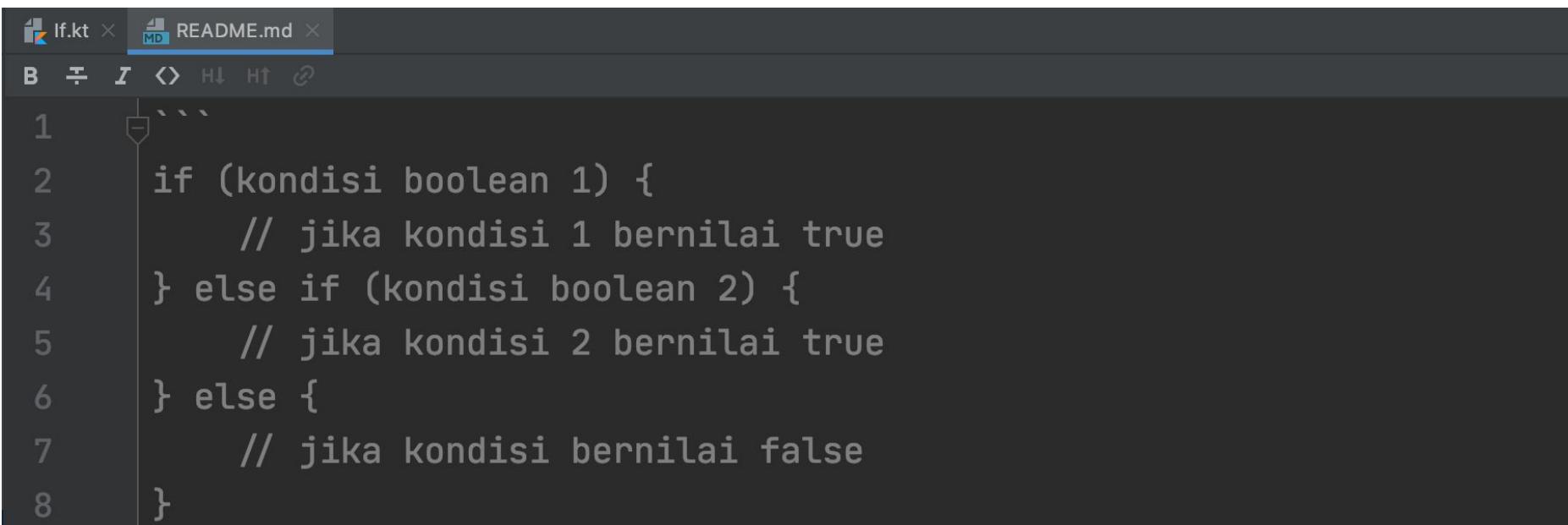
```
1 If.kt x
2
3 fun main() {
4
5     val examValue = 90
6
7     if (examValue > 80) {
8         println("Good Job")
9     } else {
10        println("Try again next year")
11    }
12 }
```

---

# Else If Expression

- Kada dalam If, kita butuh membuat beberapa kondisi
- Kasus seperti ini, di Kotlin kita bisa menggunakan Else If expression

# Else If Expression



The screenshot shows a code editor interface with a dark theme. At the top, there are two tabs: 'If.kt' and 'README.md'. The 'README.md' tab is currently active, indicated by a blue background and white text. Below the tabs is a toolbar with various icons for file operations. The main area of the editor displays the following Java code:

```
1  ``````  
2  if (kondisi boolean 1) {  
3      // jika kondisi 1 bernilai true  
4  } else if (kondisi boolean 2) {  
5      // jika kondisi 2 bernilai true  
6  } else {  
7      // jika kondisi bernilai false  
8 }
```

The code uses triple backticks at the beginning to indicate a multi-line string or a block of code. The code itself is a simple conditional statement using the 'if' and 'else if' keywords, with explanatory comments in Indonesian.

# Kode : Else If Expression



```
1 If.kt x
2
3 fun main() {
4     val examValue = 90
5     if (examValue > 80) {
6         println("Good Job")
7     } else if (examValue > 60) {
8         println("Not Bad")
9     } else {
10        println("Try again next year")
11    }
12 }
```

The image shows a screenshot of an IDE interface, specifically the Kotlin code editor. The file is named 'If.kt'. The code itself is a simple 'if-else if-else' expression. It starts with a function declaration 'fun main()', followed by a variable declaration 'val examValue = 90'. Then it uses an 'if' statement to check if 'examValue' is greater than 80, printing 'Good Job' if true. If false, it checks the next condition with an 'else if' statement for values greater than 60, printing 'Not Bad' if true. If neither condition is met, it falls through to the final 'else' block, which prints 'Try again next year'. The code is color-coded: orange for keywords like 'fun', 'val', 'if', 'else', and 'else if'; green for strings like 'Good Job', 'Not Bad', and 'Try again next year'; and blue for numbers like '90' and '60'. There are also small gray diamond icons on the left margin, likely indicating collapsed sections of code.

---

# When Expression

---

# When Expression

- Selain if expression, untuk melakukan percabangan di Kotlin, kita juga bisa menggunakan When Expression
- When expression sangat sederhana dibandingkan if
- Biasanya when expression digunakan untuk melakukan pengecekan ke kondisi dalam satu variable

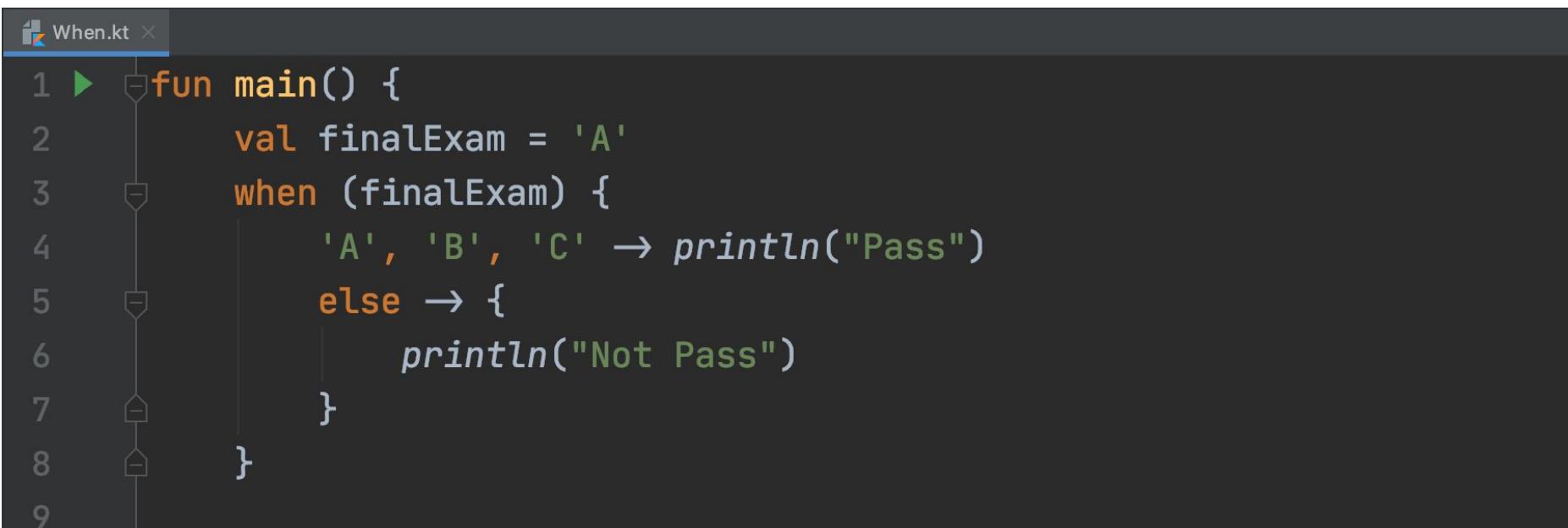


# Kode : When Expression

```
1 ►  ↗ fun main() {  
2     val finalExam = 'A'  
3     when (finalExam) {  
4         'A' → println("Amazing")  
5         'B' → println("Good")  
6         'C' → println("Not Bad")  
7         'D' → println("Bad")  
8         'E' → println("Try Again Next Year")  
9         else → println("Ups")  
10    } }
```

---

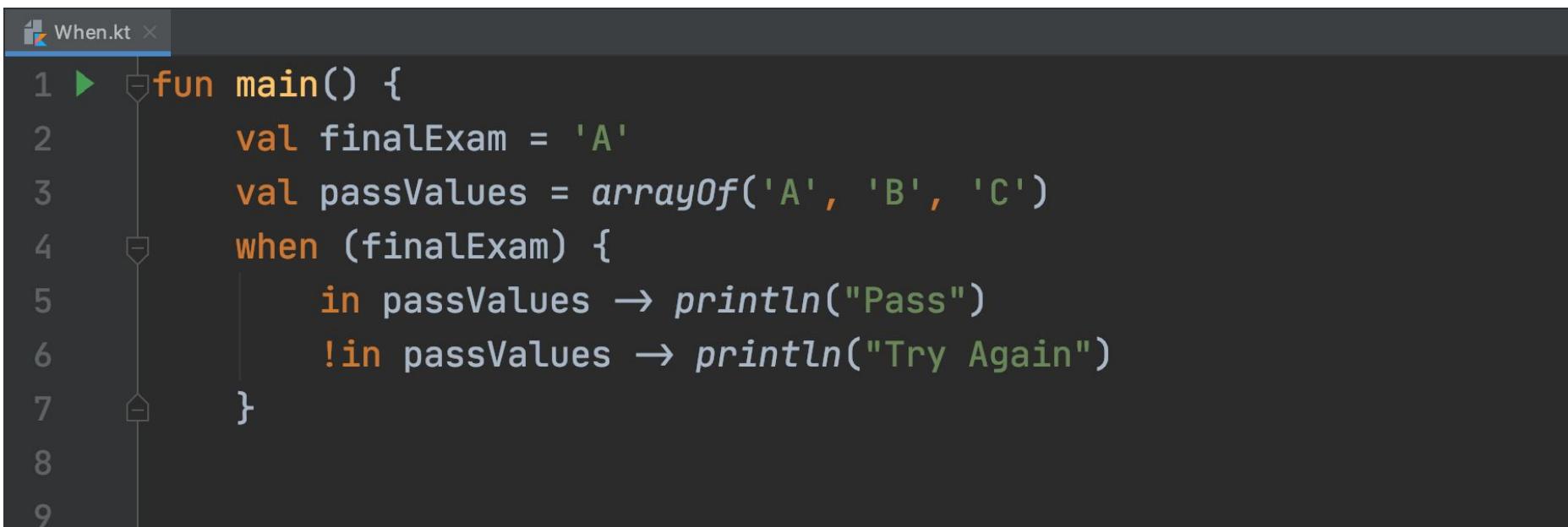
# Kode : When Expression Multiple Option



```
When.kt
1 ►  fun main() {
2     val finalExam = 'A'
3     when (finalExam) {
4         'A', 'B', 'C' → println("Pass")
5         else → {
6             println("Not Pass")
7         }
8     }
9 }
```

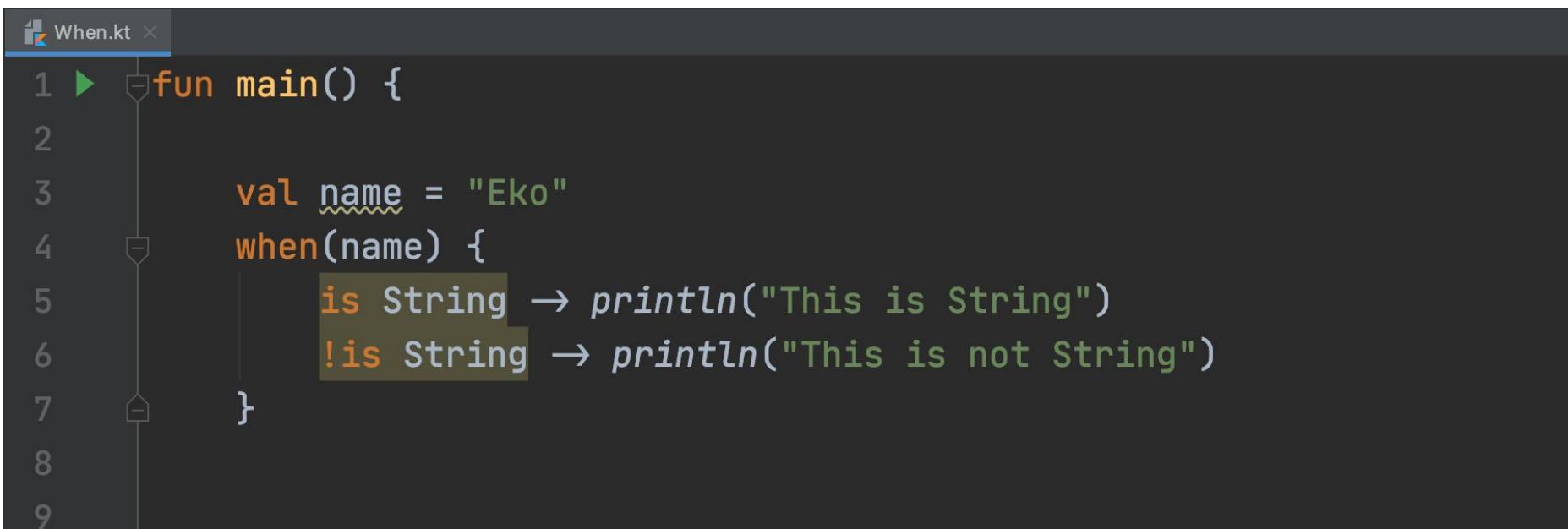
The image shows a screenshot of a code editor with a dark theme. A file named "When.kt" is open. The code within the file is a Kotlin function named "main". It declares a variable "finalExam" and uses a "when" expression to check its value. If "finalExam" is 'A', 'B', or 'C', it prints "Pass". Otherwise, it prints "Not Pass". The code is numbered from 1 to 9 on the left side. There are small icons above each line number, likely indicating code folding or selection status.

# Kode : When Expression In



```
When.kt
1 ► fun main() {
2     val finalExam = 'A'
3     val passValues = arrayOf('A', 'B', 'C')
4     when (finalExam) {
5         in passValues → println("Pass")
6         !in passValues → println("Try Again")
7     }
8
9 }
```

# Kode : When Expression Is



```
When.kt
1 fun main() {
2
3     val name = "Eko"
4     when(name) {
5         is String → println("This is String")
6         !is String → println("This is not String")
7     }
8
9 }
```

The image shows a screenshot of a code editor with a dark theme. The file tab at the top is labeled "When.kt". The code itself is as follows:

```
fun main() {
    val name = "Eko"
    when(name) {
        is String → println("This is String")
        !is String → println("This is not String")
    }
}
```

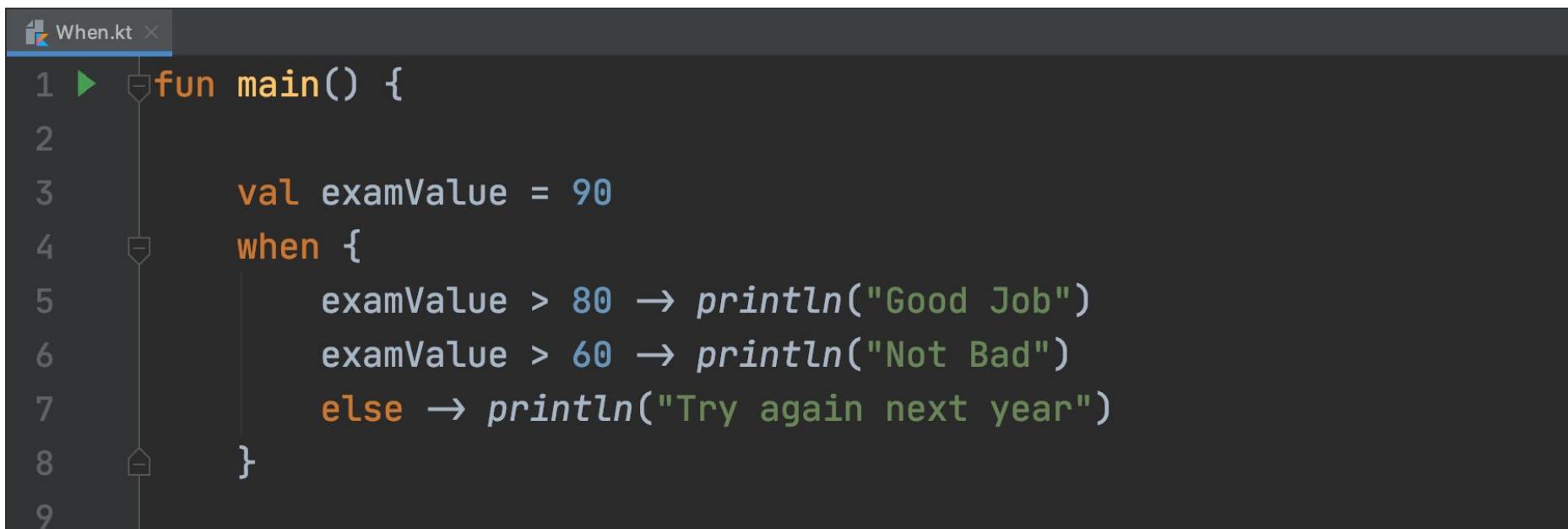
The code uses a when expression to check if the variable `name` is a string or not. The first branch, where `name` is a string, is highlighted with a green background. The entire when expression block is enclosed in a light gray box.

---

## When Sebagai Pengganti If Else

- Selain pengecekan terhadap variable
- When juga dapat digunakan sebagai pengganti if else
- Untuk mengganti if else dengan when, kita tidak perlu menggunakan variable dalam when

# Kode : When Tanda Variable



```
When.kt
1 fun main() {
2
3     val examValue = 90
4     when {
5         examValue > 80 -> println("Good Job")
6         examValue > 60 -> println("Not Bad")
7         else -> println("Try again next year")
8     }
9 }
```

The image shows a screenshot of a code editor with a dark theme. A file named "When.kt" is open. The code within the file is a Kotlin function named "main". Inside "main", there is a declaration of a variable "examValue" set to 90. A "when" expression is used to print different messages based on the value of "examValue". If "examValue" is greater than 80, it prints "Good Job". If it is greater than 60, it prints "Not Bad". Otherwise, it prints "Try again next year". The code editor has line numbers from 1 to 9 on the left, and there are small icons above each line number indicating code completion or similar information.

---

# For Loops

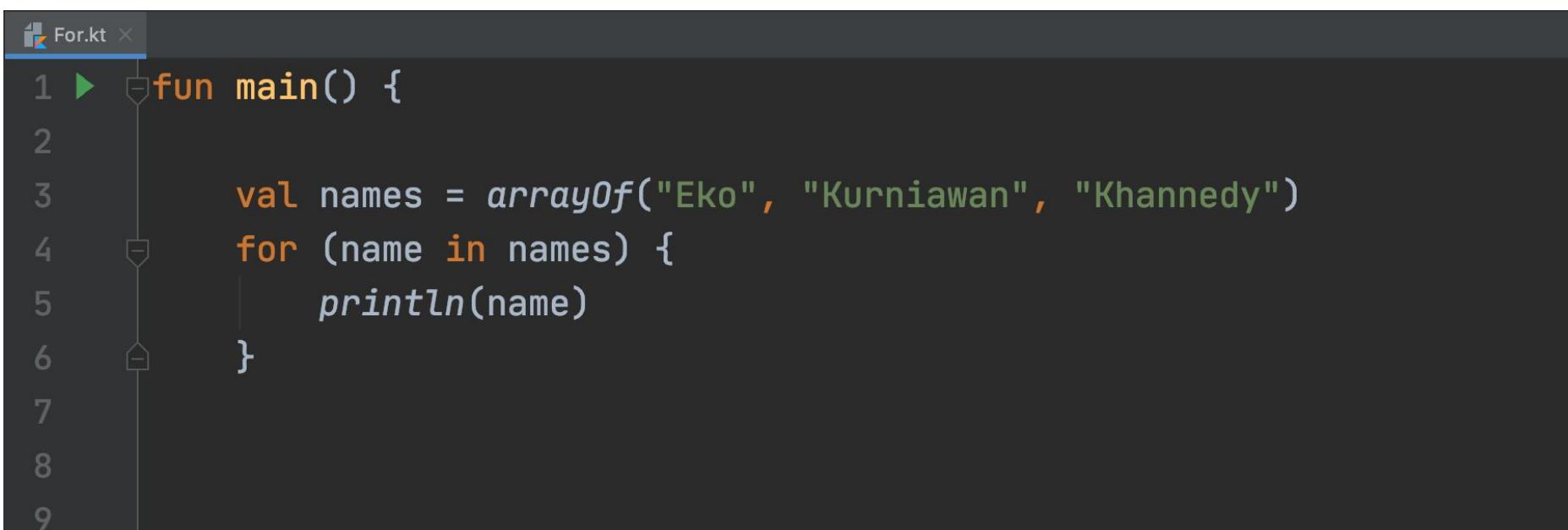
---

# For Loops

- Dalam bahasa pemrograman, biasanya ada fitur yang bernama perulangan
- Salah satu fitur perulangan di Kotlin adalah for
- For digunakan untuk melakukan perulangan iterasi dari data iterator (Array, Range, dan lain-lain)

---

# Kode : For Array



```
For.kt x
1 ►  ↴ fun main() {
2
3     ↴ val names = arrayOf("Eko", "Kurniawan", "Khannedy")
4     ↴ for (name in names) {
5         ↴     println(name)
6     ↴ }
7
8
9 }
```

# Kode : For Range



The screenshot shows a code editor window with a dark theme. The file is named "For.kt". The code contains two for loops:

```
1 ► fun main() {  
2     for (value in 0..100) {  
3         println(value)  
4     }  
5     for (value in 1000 downTo 0 step 5) {  
6         println(value)  
7     }  
8 }  
9 }
```

The first for loop iterates from 0 to 100, printing each value. The second for loop iterates from 1000 down to 0, printing each value in steps of 5. The code editor has syntax highlighting and line numbers.

---

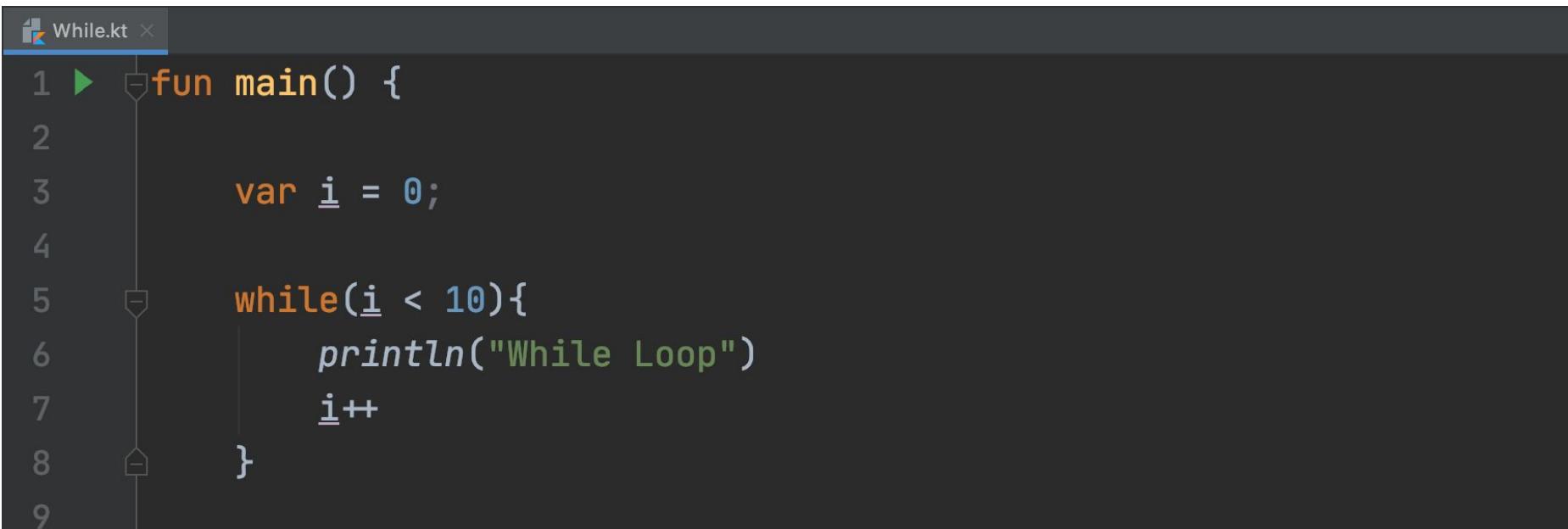
# While Loops

---

# While Loops

- While adalah salah satu perulangan lain yang ada di Kotlin
- While adalah salah satu perulangan yang sangat flexible, dimana kode while akan melakukan pengecekan kondisi, jika kondisi bernilai true, maka dia akan menjalankan blok while, dan terus diulangi sampai kondisi while bernilai false

# Kode : While



The screenshot shows a code editor window with a dark theme. The title bar says "While.kt". The code is as follows:

```
1 ►  fun main() {  
2  
3     var i = 0;  
4  
5     while(i < 10){  
6         println("While Loop")  
7         i++  
8     }  
9 }
```

The code consists of 9 lines. Lines 1 through 4 are part of the main function definition. Lines 5 through 8 form the body of the while loop. Line 5 initializes the variable `i` to 0. Line 6 prints the string "While Loop" to the console. Line 7 increments the value of `i`. Line 8 marks the end of the loop body. Line 9 marks the end of the main function.

---

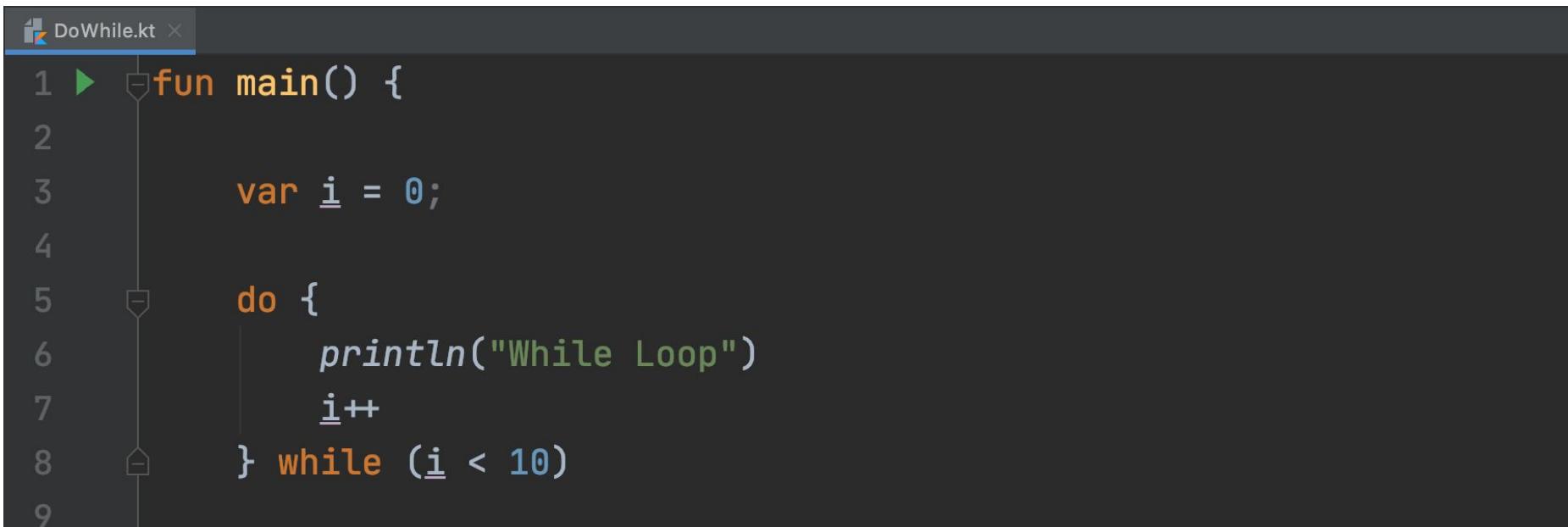
# Do While Loops

---

# Do While Loop

- Do While Loop adalah perulangan yang hampir sama dengan While Loop
- Yang membedakan adalah, pada Do While Loop, kode blok akan dijalankan dahulu, baru diakhiri dilakukan pengecekan kondisi

# Kode : Do While



The screenshot shows a code editor window with a dark theme. At the top, there is a tab labeled "DoWhile.kt". Below the tabs, there is a toolbar with icons for file operations. The main area displays the following code:

```
1 ►  ↴ fun main() {  
2  
3     var i = 0;  
4  
5     ↴ do {  
6         println("While Loop")  
7         i++  
8     } while (i < 10)  
9
```

The code is a Kotlin script named "DoWhile.kt". It contains a main function that initializes a variable "i" to 0. A "do" loop is then entered, which prints the string "While Loop" to the console and increments "i" by 1. This loop continues as long as "i" is less than 10. The code editor uses color coding for syntax: orange for keywords like "fun", "main", "var", "do", and "while", blue for the variable "i", and green for the string "While Loop". Line numbers are shown on the left side of the code.

---

# Break & Continue

---

# Break & Continue

- Break & continue adalah kata kunci yang bisa digunakan dalam semua perulangan di Kotlin
- Break digunakan untuk menghentikan seluruh perulangan
- Continue adalah digunakan untuk menghentikan perulangan yang berjalan, dan langsung melanjutkan ke perulangan selanjutnya

# Kode : Break

The screenshot shows a code editor window titled "BreakContinue.kt". The code is annotated with several small icons:

- A green play button icon is at the top left.
- At the start of the first line ("fun main() {"), there is a yellow square icon with a black outline.
- On the second line ("var i = 0"), there is a yellow square icon with a black outline.
- On the third line ("while (true) {"), there is a yellow square icon with a black outline.
- On the fourth line (" println("Break Me")"), there is a yellow square icon with a black outline.
- On the fifth line (" i++"), there is a yellow square icon with a black outline.
- On the sixth line (" if (i > 1000) {"), there is a yellow square icon with a black outline.
- On the seventh line (" break"), there is a yellow square icon with a black outline.
- On the eighth line (" }"), there is a yellow square icon with a black outline.
- On the ninth line ("}"), there is a yellow square icon with a black outline.

```
1 ►  ↗ fun main() {
2     ↗ var i = 0
3     ↗ while (true) {
4         ↗     println("Break Me")
5         ↗     i++
6         ↗     if (i > 1000) {
7             ↗         break
8         ↗     }
9     ↗ }
```

# Kode : Continue



```
BreakContinue.kt
```

```
1 ►  ↗ fun main() {  
2  
3     ↗ for (i in 1..1000) {  
4         ↗ if (i % 2 == 0) {  
5             ↗     continue  
6         }  
7         ↗     println(i)  
8     }  
9 }
```

The image shows a screenshot of a code editor with a dark theme. A file named "BreakContinue.kt" is open. The code within the file is a Kotlin function "main". Inside "main", there is a "for" loop that iterates from 1 to 1000. Inside the loop, there is an "if" condition that checks if the current value "i" is even (i.e., "i % 2 == 0"). If it is, the "continue" keyword is used, which causes the loop to skip the rest of the current iteration and move to the next iteration. Otherwise, the "println(i)" statement is executed, printing the value of "i" to the console.

---

# Function

---

# Function

- Sebelumnya kita sudah mengenal sebuah function yang wajib dibuat agar program Kotlin bisa berjalan, yaitu function main
- Function adalah sebuah blok kode yang sengaja dibuat dalam program agar bisa digunakan berulang-ulang
- Cara membuat function di Kotlin sangat sederhana, hanya dengan menggunakan kata kunci fun lalu diikuti dengan nama function nya dan blok kode isi function nya
- Setelah membuat function, kita bisa mengeksekusi function tersebut dengan memanggilnya menggunakan kata kunci nama function nya

# Kode : Function

```
Function.kt
1 fun helloWorld(){
2     println("Hello World")
3     println("Programmer Zaman Now")
4 }
5
6 ► fun main() {
7     helloWorld()
8 }
9
```

---

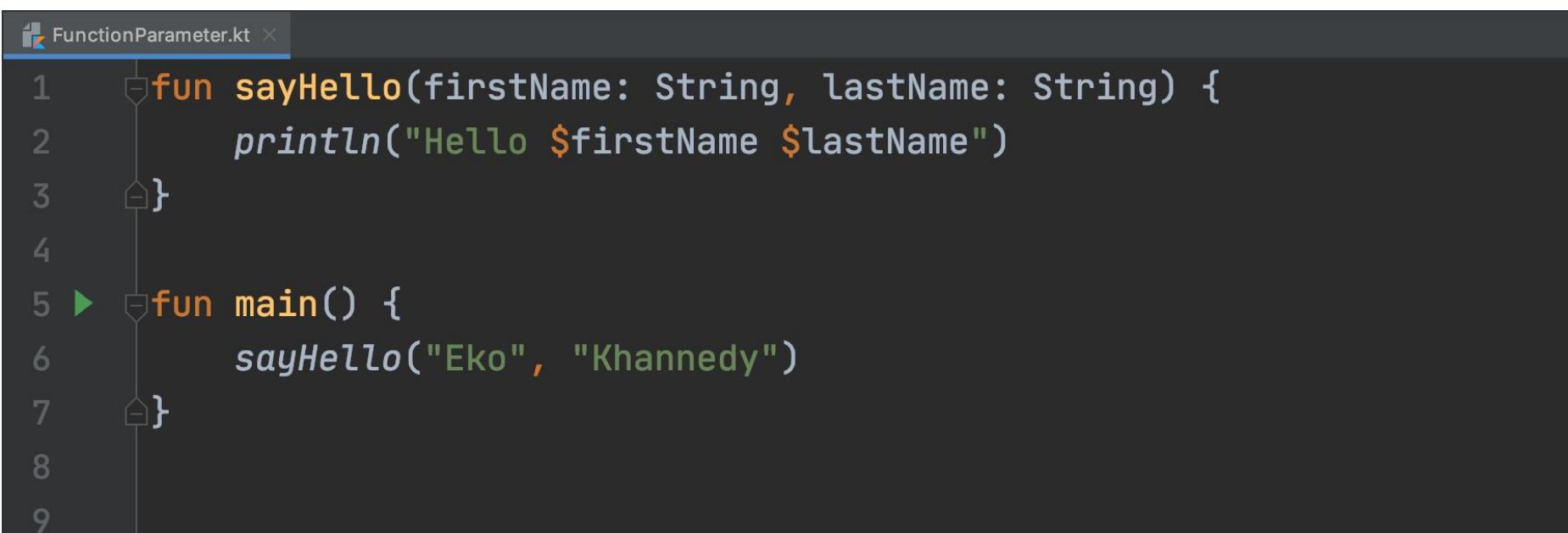
# Function Parameter

---

# Function Parameter

- Saat membuat function, kadang-kadang kita membutuhkan data dari luar, atau kita sebut parameter.
- Di Kotlin, kita bisa menambahkan parameter di function, bisa lebih dari satu
- Parameter tidaklah wajib, jadi kita bisa membuat function tanpa parameter seperti sebelumnya yang sudah kita buat
- Namun jika kita menambahkan parameter di function, maka ketika memanggil function tersebut, kita wajib memasukkan data ke parameternya

# Kode : Function Parameter



The screenshot shows a code editor window with a dark theme. The file tab at the top is labeled "FunctionParameter.kt". The code itself is as follows:

```
1  fun sayHello(firstName: String, lastName: String) {
2      println("Hello $firstName $lastName")
3  }
4
5 ▶ fun main() {
6     sayHello("Eko", "Khannedy")
7 }
```

The code defines a function named `sayHello` that takes two parameters: `firstName` and `lastName`, both of type `String`. It prints a greeting message using string interpolation. Below it, the `main` function calls `sayHello` with the arguments "Eko" and "Khannedy". The editor uses color coding for syntax: orange for keywords like `fun`, `String`, and `println`, green for strings, and grey for comments and numbers. Line numbers are visible on the left side.

---

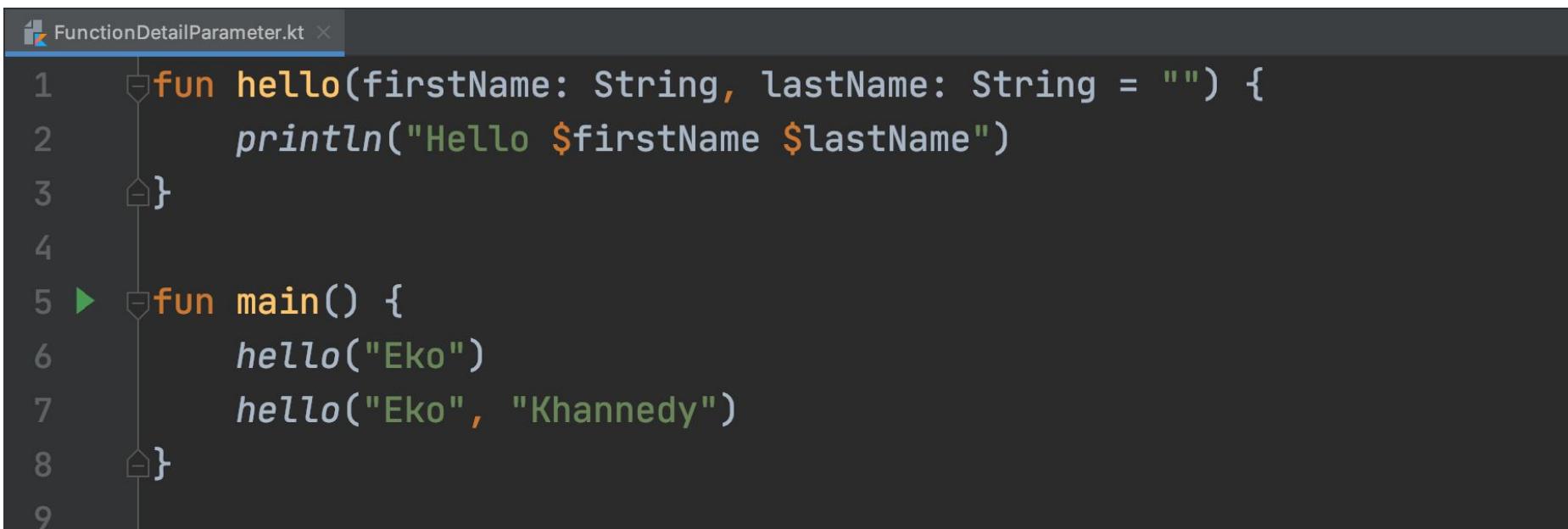
# Function Default Argument

---

# Function Default Parameter

- Di Kotlin, function parameter wajib diisi ketika memanggil function
- Namun kita juga bisa memasukkan nilai default pada parameter, dengan demikian saat memanggil function tersebut, kita tidak wajib memasukkan nilai pada parameter nya
- Default parameter ini cocok pada jenis parameter yang sekiranya memang tidak wajib untuk diisi

# Kode : Function Default Parameter



The screenshot shows a code editor window with a dark theme. The title bar says "FunctionDetailParameter.kt". The code is as follows:

```
1  fun hello(firstName: String, lastName: String = "") {
2      println("Hello $firstName $lastName")
3  }
4
5 ▶ fun main() {
6     hello("Eko")
7     hello("Eko", "Khannedy")
8 }
9
```

The code defines a function named `hello` that takes two parameters: `firstName` and `lastName`. The `lastName` parameter has a default value of an empty string. The `main` function calls `hello` with one argument ("Eko") and with two arguments ("Eko", "Khannedy").

---

# Function Named Argument

---

# Function Named Argument

- Kadang ada function yang parameternya banyak sekali
- Hal ini sangat menyulitkan saat kita akan memanggil function tersebut, kita harus benar-benar tahu urutan parameter di function tersebut
- Untungnya kotlin memiliki fitur yang namanya Named Argument
- Named Argument adalah fitur dimana kita bisa menyebutkan nama parameter saat memanggil function, dengan demikian kita tidak wajib tahu posisi tiap parameter

---

# Kode : Function Named Argument

```
1  fun fullName(firstName: String,  
2                  middleName: String,  
3                  lastName: String) {  
4      println("Hello $firstName $middleName $lastName")  
5  }  
6 ▶  fun main() {  
7      fullName(firstName = "Eko",  
8                  lastName = "Khannedy",  
9                  middleName = "Kurniawan")  
10 }
```

---

# Unit Returning Function

---

# Unit Returning Function

- Function ada 2 jenis, pertama tidak mengembalikan nilai, yang kedua mengembalikan nilai
- Function-function yang sebelumnya sudah kita buat adalah function yang tidak mengembalikan nilai
- Sebenarnya, function-function yang sudah kita buat sebelumnya, dia mengembalikan tipe data Unit, dimana Unit adalah tanda bahwa function tersebut tidak mengembalikan apa-apa
- Penulisan Unit adalah tidak wajib, namun jika kita menulis tipe data pengembalian sebuah function, maka secara otomatis dia adalah Unit

---

# Kode : Unit Returning Function

```
1  ↗ fun printHello(name: String?): Unit {  
2      ↗ if (name == null) {  
3          println("Hello Bro")  
4      } else {  
5          println("Hello $name")  
6      }  
7  ↘ }  
8  
9 ▶ ↗ fun main() {  
10    ↗ printHello(name: "Eko")
```

---

# Function Return Type

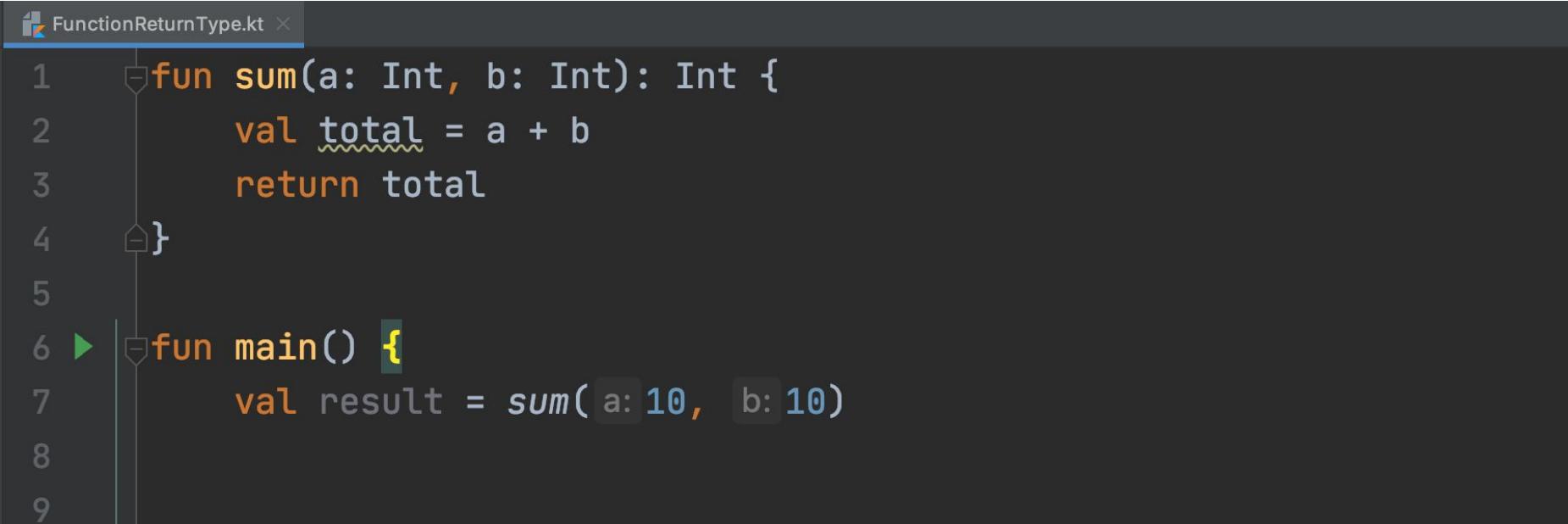
---

# Function Return Type

- Seperti yang sudah dibahas sebelumnya, bahwa function itu bisa mengembalikan data
- Untuk memberitahu bahwa function mengembalikan data, kita harus menuliskan tipe data kembalian dari function tersebut
- Jika function tersebut kita deklarasikan dengan tipe data pengembalian, maka wajib di dalam function nya kita harus mengembalikan data
- Untuk mengembalikan data dari function, kita bisa menggunakan kata kunci return, diikuti dengan datanya

---

# Function Return Type



A screenshot of a code editor showing a file named `FunctionReturnType.kt`. The code defines a `sum` function that takes two `Int` parameters and returns an `Int`. It calculates the total by adding the two parameters and returns it. Below it, a `main` function is defined, which calls the `sum` function with both parameters set to `10`.

```
1  fun sum(a: Int, b: Int): Int {  
2      val total = a + b  
3      return total  
4  }  
5  
6  fun main() {  
7      val result = sum(a: 10, b: 10)  
8  }  
9
```

---

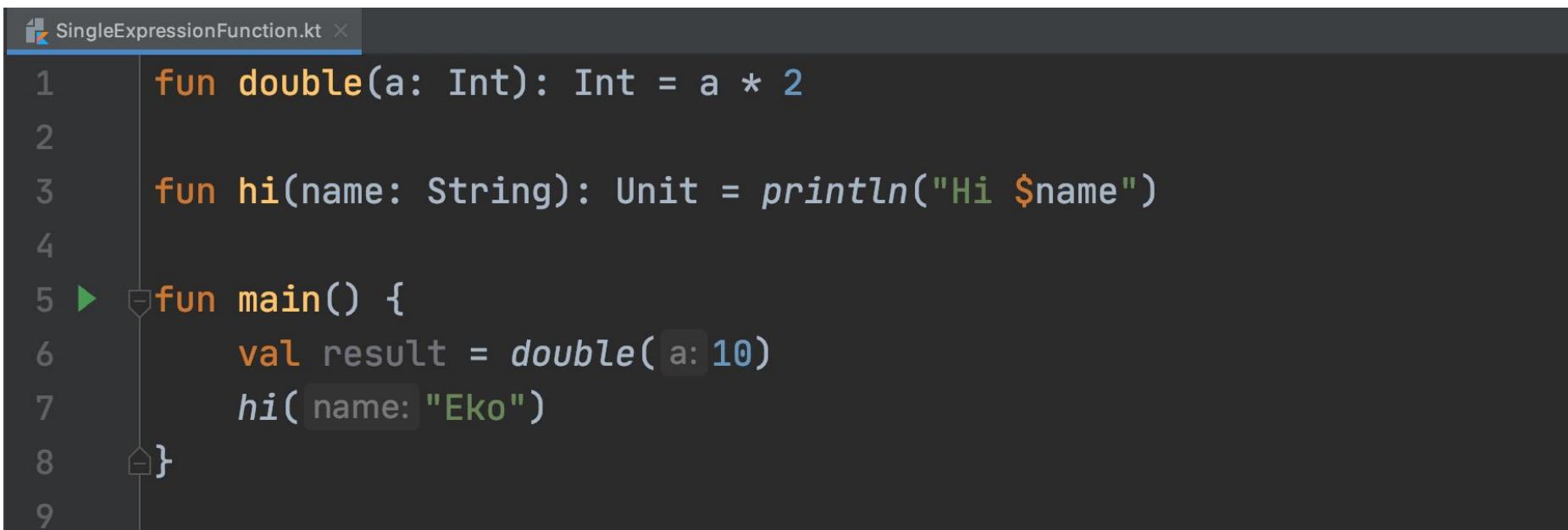
# Single Expression Function

---

# Single Expression Function

- Kadang kita sering membuat function yang sangat sederhana
- Misal function yang hanya berisikan kode blok sederhana, misal hanya 1 baris
- Jika kita mengalami hal seperti ini, kita bisa mengubahnya menjadi single expression function
- Single expression function adalah deklarasi function hanya dengan 1 baris kode
- Untuk membuat single expression function, kita hanya butuh tanda = (sama dengan) setelah deklarasi nama function dan tipe data pengembalian function

# Kode : Single Expression Function



The screenshot shows a code editor window with a dark theme. The title bar says "SingleExpressionFunction.kt". The code is as follows:

```
1  fun double(a: Int): Int = a * 2
2
3  fun hi(name: String): Unit = println("Hi $name")
4
5 ▶  ↴ fun main() {
6      val result = double( a: 10)
7      hi( name: "Eko")
8  }
```

The code defines two functions: `double` which takes an `Int` and returns an `Int` by multiplying it by 2; and `hi` which takes a `String` and prints "Hi \$name" using `println`. The `main` function calls `double` with the argument 10 and `hi` with the argument "Eko". Lines 1 through 4 are blank. Line 5 starts with a green triangle icon followed by the `fun main()` declaration. Line 6 shows the assignment of `result` to the return value of `double`. Line 7 shows the call to `hi`. Line 8 ends the `main` function with a closing brace.

---

# Function Varargs Parameter

---

# Function Varargs Parameter

- Parameter yang berada di posisi terakhir, memiliki kemampuan dijadikan sebuah varargs
- Varargs artinya datanya bisa menerima lebih dari satu input, atau anggap saja semacam Array.
- Apa bedanya dengan parameter biasa dengan tipe data Array?
  - Jika parameter tipe Array, kita wajib membuat array terlebih dahulu sebelum mengirimkan ke function
  - Jika parameter menggunakan varargs, kita bisa langsung mengirim data nya, jika lebih dari satu, cukup gunakan tanda koma

# Function Varargs Parameter

FunctionVarargsParameter.kt

```
1  fun finalValue(name: String, vararg values: Int) {
2      var total = 0.0
3      for (value in values) {
4          total += value
5      }
6      total /= values.size
7      println("Final Value $name = total")
8  }
```

---

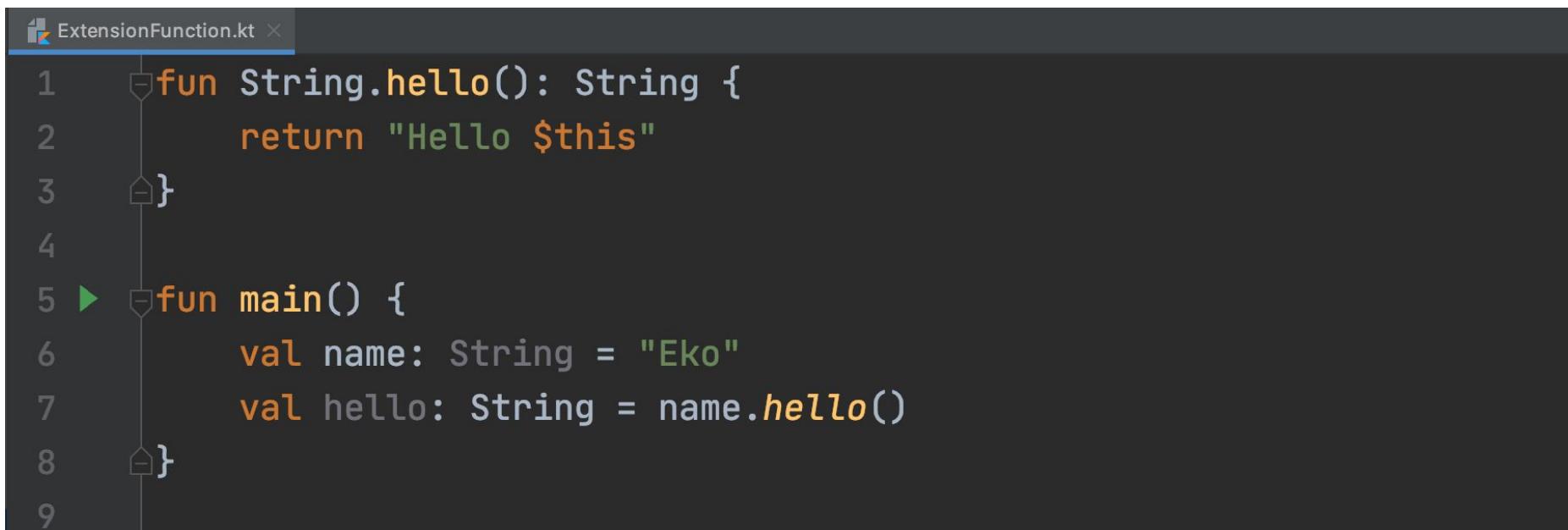
# Extension Function

---

# Extension Function

- Extension function adalah kemampuan menambahkan function pada tipe data yang sudah ada
- Extension function adalah salah satu fitur yang sangat powerfull, namun harap bijak menggunakannya, karena jika terlalu banyak digunakan, akan membuat program sulit dimengerti, karena terlihat seperti magic
- Untuk membuat extension function, kita cukup menambahkan tipe data pada nama function nya, lalu diikuti dengan tanda . (titik)
- Untuk mengakses data nya di extension function, kita bisa menggunakan kata kunci this

# Kode : Extension Function



The screenshot shows a code editor window with a dark theme. The file is named "ExtensionFunction.kt". The code defines an extension function "hello" on the String class and a main function.

```
1  fun String.hello(): String {  
2      return "Hello $this"  
3  }  
4  
5 ▶ fun main() {  
6     val name: String = "Eko"  
7     val hello: String = name.hello()  
8 }  
9
```

---

# Function Infix Notation

---

# Function Infix Notation

- Infix Notation adalah operasi yang biasa dilakukan di operasi matematika, dimana dia melakukan operasi terhadap dua data
- Hampir semua operasi matematika adalah infix notation
- Di kotlin, kita bisa membuat function infix notation juga
- Untuk menggunakan function infix notation, tidak wajib menggunakan tanda . (titik)

---

## Syarat Function Infix Notation

- Harus sebagai function member (akan dibahas di OOP) atau function extension
- Harus memiliki 1 parameter
- Parameter tidak boleh varargs dan tidak boleh memiliki nilai default

# Kode : Function Infix Notation

```
FunctionInfixNotation.kt ×
1  infix fun String.to(type: String): String {
2      if (type == "UP") {
3          return this.toUpperCase()
4      } else {
5          return this.toLowerCase()
6      }
7  }
8 ► fun main() {
9     val result: String = "Eko" to "UP"
```

---

# Function Scope

---

# Function Scope

- Function scope adalah ruang lingkup dimana sebuah function bisa diakses
- Saat kita membuat function di dalam file kotlin, maka secara otomatis function tersebut bisa diakses dari file kotlin manapun
- Jika kita ingin membatasi misalnya sebuah function hanya bisa diakses dalam function tertentu, maka kita bisa membuat function di dalam function



# Function Scope



A screenshot of a code editor showing a file named `FunctionScope.kt`. The code demonstrates function scope in Kotlin:

```
1 >  fun main() {  
2     >     fun sayHello(name: String): Unit {  
3         >         println("Hello $name")  
4     }  
5  
6         sayHello(name: "Eko")  
7     }  
8  
9 }
```

The code consists of two nested functions: `main` and `sayHello`. The `sayHello` function is defined within the `main` function. Inside `sayHello`, the variable `name` is used without being explicitly declared, demonstrating the function's local scope. The code editor shows syntax highlighting for keywords like `fun`, `main`, and `println`, and for identifiers like `sayHello` and `name`.

---

# Return If & When

---

# Return If & When

- Kadang dalam sebuah function, kita sering menggunakan If expression atau when expression, lalu di dalam blok nya kita mengembalikan nilai untuk sebuah function
- Kotlin mendukung return if atau when, dimana fitur ini bisa mempermudah kita ketika ingin mengembalikan nilai dalam if atau when

# Kode : Return If



The screenshot shows a code editor window with a dark theme. The file tab at the top is labeled "ReturnIfWhen.kt". The code itself is as follows:

```
1  fun sayHello(name: String = ""): String {  
2      return if (name == "") {  
3          "Hello Bro"  
4      } else {  
5          "Hello $name"  
6      }  
7  }  
8  
9  sayHello( name: "Eko")  
10  
11
```

The code defines a function named `sayHello` that takes an optional `String` parameter `name`. If `name` is empty, it returns the string "Hello Bro". Otherwise, it returns "Hello \$name". The function is then called with the argument "Eko". The code editor uses color coding for syntax: orange for functions and conditionals, green for strings and variables, and grey for comments and the file tab.

# Kode : Return When



```
ReturnIfWhen.kt
4     fun sayHello(name: String = ""): String {
5         return when (name) {
6             "" -> "Hello Bro"
7             else -> "Hello $name"
8     }
9 }
10 sayHello( name: "Eko")
11
12
```

The image shows a screenshot of a code editor with a dark theme. The file tab at the top is labeled 'ReturnIfWhen.kt'. The code itself is a simple function named 'sayHello' that returns a string based on the value of 'name'. It uses the 'when' expression with an 'else' branch. The code editor has line numbers on the left and syntax highlighting for Java-like constructs like 'String' and 'else'. There are also small icons above each line, likely indicating code quality or specific annotations.

---

# Recursive Function

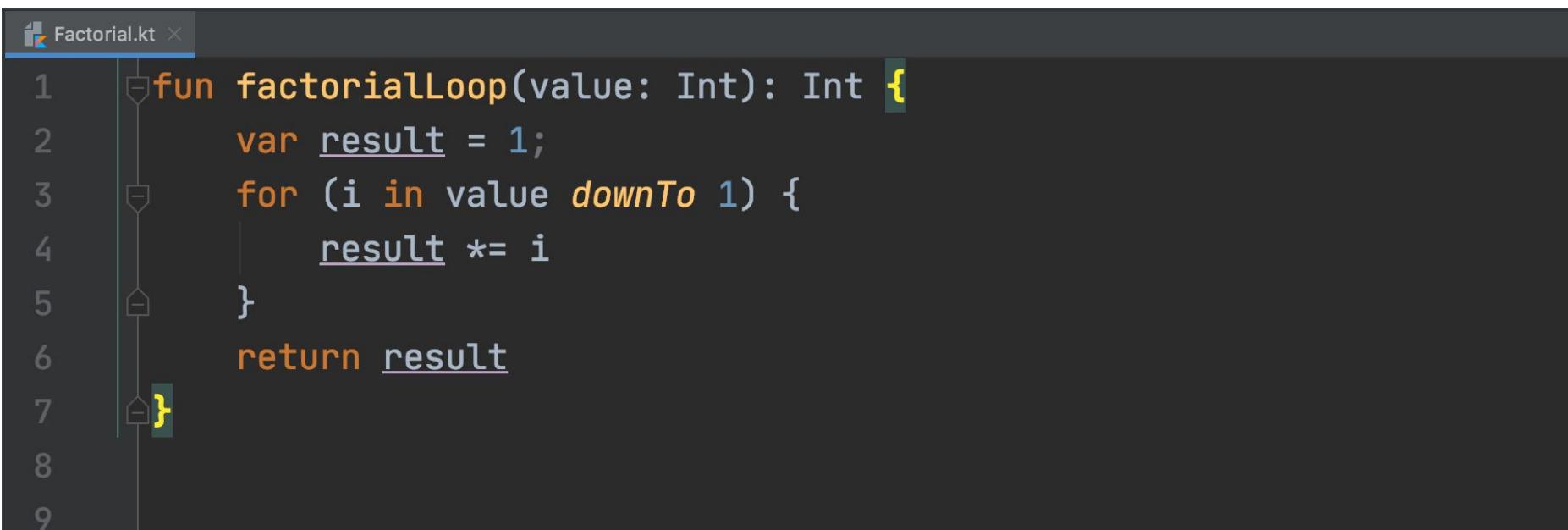
---

# Recursive Function

- Recursive function adalah function yang memanggil function dirinya sendiri
- Kadang dalam pekerjaan, kita sering menemui kasus dimana menggunakan recursive function lebih mudah dibandingkan tidak menggunakan recursive function
- Contoh kasus yang lebih mudah diselesaikan menggunakan recursive adalah Factorial

---

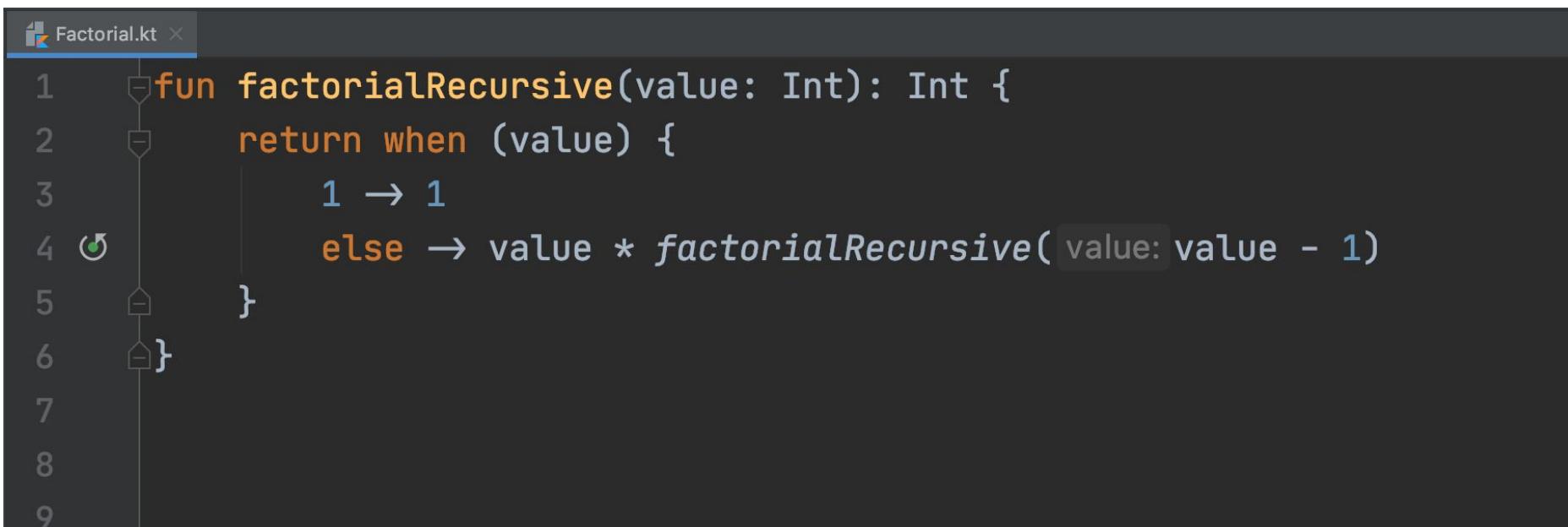
# Kode : Factorial For Loop



```
Factorial.kt ×
1 fun factorialLoop(value: Int): Int {
2     var result = 1;
3     for (i in value downTo 1) {
4         result *= i
5     }
6     return result
7 }
8
9
```

The image shows a screenshot of a code editor with a dark theme. A file named "Factorial.kt" is open. The code defines a function "factorialLoop" that takes an integer "value" and returns an integer. Inside the function, a variable "result" is initialized to 1. A "for" loop then iterates from "value" down to 1, multiplying "result" by the current value "i" in each iteration. Finally, the function returns the calculated "result". The code editor interface includes a tab bar at the top with the file name, and a vertical scrollbar on the left side of the code area.

# Kode : Factorial Recursive Function



The screenshot shows a code editor window with a dark theme. The file tab at the top is labeled "Factorial.kt". The code itself is a recursive function named "factorialRecursive". The code is as follows:

```
1 fun factorialRecursive(value: Int): Int {
2     return when (value) {
3         1 → 1
4     else → value * factorialRecursive(value - 1)
5 }
6 }
```

The code editor highlights the word "else" in orange. There are also some small green and grey circular markers near the line numbers 4 and 5.

---

# Tail Recursive Function

---

# Recursive Function

- Recursive function adalah salah satu kemampuan bagus di Kotlin, namun sayangnya ada keterbatasan dalam penggunaan recursive
- Jika recursive function yang kita buat, saat dijalankan memanggil function dirinya sendiri terlalu dalam, maka bisa dimungkinkan akan terjadi error stack overflow



# Stack Overflow Error

factorial(5)

factorial(5) => 5 \* factorial(4)

factorial(5) => 5 \* factorial(4) => 4 \* factorial(3)

factorial(5) => 5 \* factorial(4) => 4 \* factorial(3) => 3 \* factorial(2)

factorial(5) => 5 \* factorial(4) => 4 \* factorial(3) => 3 \* factorial(2) => 2 \* factorial(1)

factorial(5) => 5 \* factorial(4) => 4 \* factorial(3) => 3 \* factorial(2) => 2 \* factorial(1) => 1

---

# Tail Recursive Function

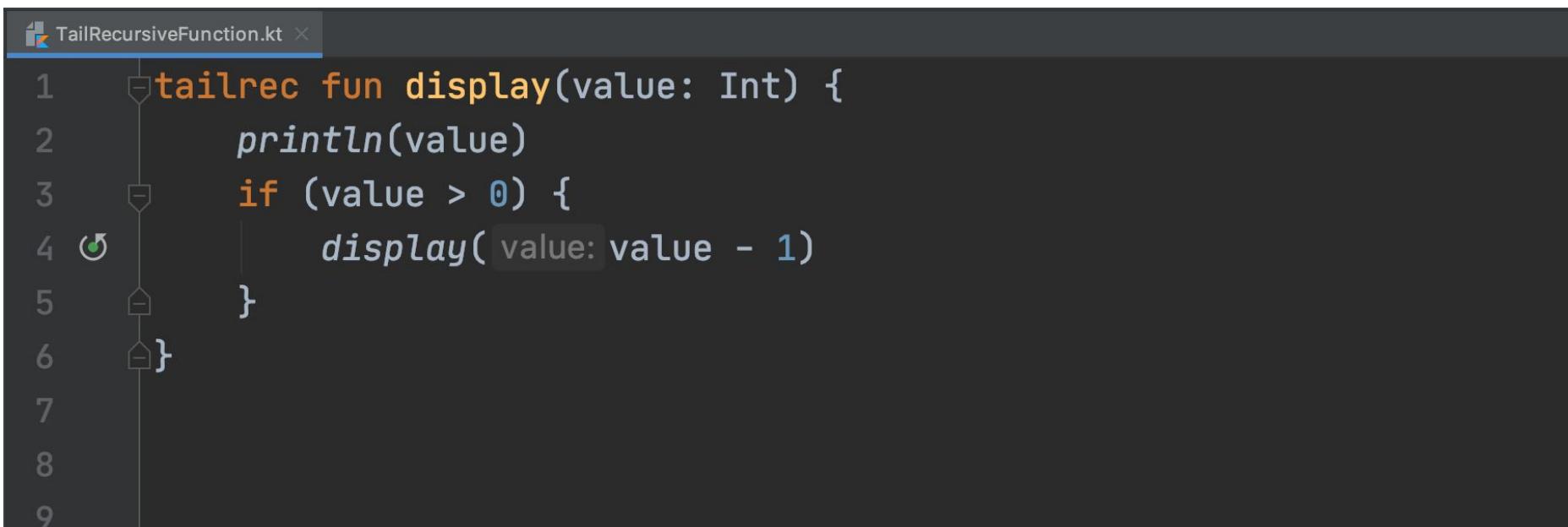
- Permasalahan ini di bahasa pemrograman Java tidak bisa ditangani
- Namun di Kotlin, masalah ini ada solusinya, yaitu dengan menggunakan tail recursive function
- Tail recursive function adalah teknik mengubah recursive function yang kita buat, menjadi looping biasa ketika dijalankan
- Tidak semua recursive function bisa secara otomatis dibuat menjadi tail recursive function, ada syarat-syarat nya

---

## Syarat Tail Recursive Function

- Tambahkan tailrec di functionnya
- Saat memanggil function dirinya sendiri, hanya boleh memanggil function dirinya sendiri, tanpa embel-embel operasi dengan data lain

# Kode : Tail Recursive Function

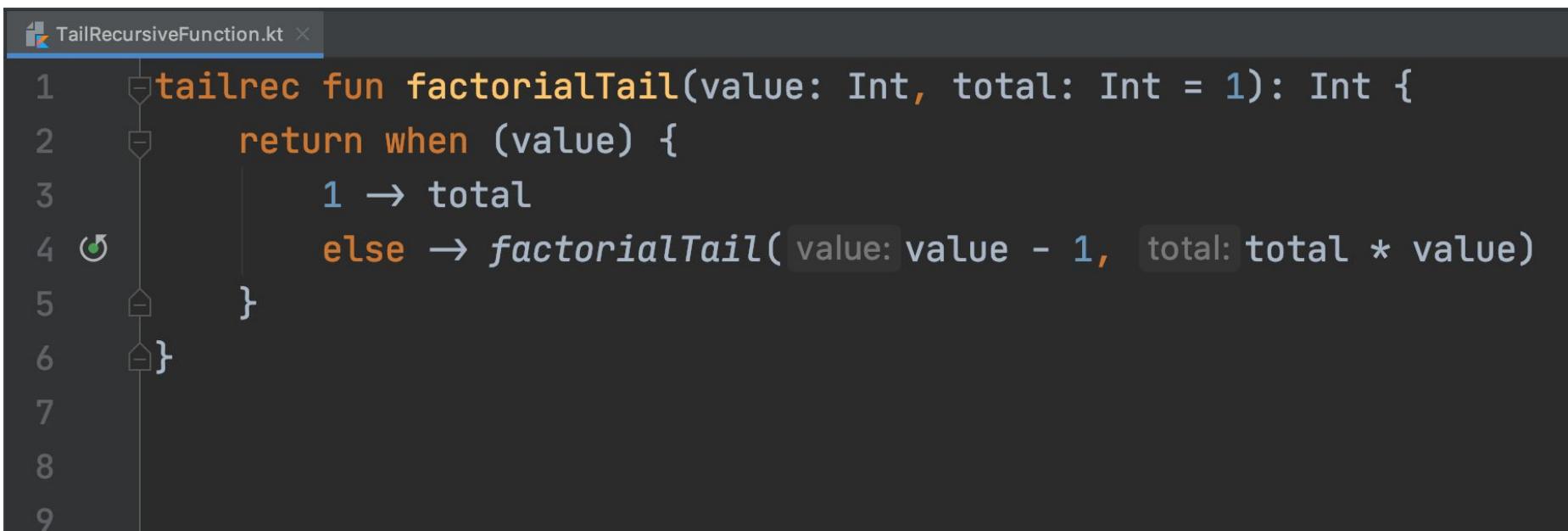


The screenshot shows a code editor window with a dark theme. The title bar says "TailRecursiveFunction.kt". The code is a tail-recursive function named "display". The code is annotated with several small icons: a green circle with a dot at line 4, a green arrow at line 5, and a red arrow at line 6.

```
1 tailrec fun display(value: Int) {
2     println(value)
3     if (value > 0) {
4         display(value - 1)
5     }
6 }
```

---

# Kode : Tail Recursive Factorial Function



The screenshot shows a code editor window with a dark theme. The title bar says "TailRecursiveFunction.kt". The code is a tail-recursive factorial function:

```
tailrec fun factorialTail(value: Int, total: Int = 1): Int {  
    return when (value) {  
        1 → total  
        else → factorialTail(value - 1, total * value)  
    }  
}
```

The code editor highlights the word "factorialTail" in orange. Lines 1 through 9 are numbered on the left. A green circular icon is located next to line 4.



# Tail Recursive Factorial

factorialTail(5, 1)

factorialTail(4, 5)

factorialTail(3, 20)

factorialTail(2, 60)

factorialTail(1, 120)

120

---

# Lambda Expression

---

# Lambda Expression

- Function di kotlin adalah first-class citizens, artinya dianggap seperti tipe data yang lainnya.
- Bisa disimpan di variable, array, bahkan bisa dikirim ke parameter function itu sendiri
- Lambda expression secara sederhana adalah function yang tidak memiliki nama
- Biasanya saat kita membuat function, kita akan selalu membuat menggunakan kata kunci fun dan mendeklarasikan nama function nya
- Dengan lambda expression, kita bisa membuat function tanpa harus mendeskripsikan function nya

---

# Kode : Lambda Expression di Variable



The screenshot shows a code editor window with a dark theme. The file is named "Lambda.kt". The code demonstrates how to define and use a lambda expression as a variable:

```
1 ► ┌ fun main() {
2   // membuat lambda
3   val lambdaName: (String) -> String = { value: String ->
4     value.toUpperCase()
5   }
6
7   // mengeksekusi lambda
8   val name = lambdaName("eko")
9 }
```

The code consists of 9 lines. Lines 1 through 5 define the main function and its body, which contains a lambda expression assigned to the variable `lambdaName`. This lambda takes a `String` parameter `value` and returns its uppercase version. Line 6 is an empty line. Lines 7 through 9 complete the main function body and declare a variable `name` that holds the result of calling `lambdaName` with the argument "eko".

# Kode : It

```
1 ►  ↗ fun main() {  
2     // membuat lambda  
3     ↗ val lambdaName: (String) → String = { it: String  
4         |     it.toUpperCase()  
5     }  
6     // mengeksekusi lambda  
7     ↗ val name = lambdaName("eko")  
8  
9 }
```

# Kode : Method References

```
1  fun toUpper(value: String): String = value.toUpperCase()
2
3 ▶  ↴ fun main() {
4      // membuat lambda
5      val lambdaName: (String) → String = ::toUpper
6
7      // mengeksekusi lambda
8      val name = lambdaName("eko")
9 }
```

---

# Higher-Order Functions

---

# Higher-Order Functions

- Higher-Order Function adalah function yang menggunakan function sebagai parameter atau mengembalikan function
- Penggunaan Higher-Order Function kadang berguna ketika kita ingin membuat function yang general dan ingin mendapatkan input yang flexible berupa lambda, yang bisa dideklarasikan oleh si user ketika memanggil function tersebut

# Kode : Higher-Order Function

```
1  fun hello(value: String, transformer: (String) → String): String {  
2      return "Hello ${transformer(value)}";  
3  }  
4  
5  ▶ fun main() {  
6      val upperTransformer = { value: String → value.toUpperCase() }  
7      val lowerTransformer = { value: String → value.toLowerCase() }  
8      println(hello( value: "Eko", upperTransformer))  
9      println(hello( value: "Eko", lowerTransformer))
```



# Kode : Trailing Lambda

```
4
5 ▶  ⌂ fun main() {
6    ⌂ val result1 = hello( value: "Eko") { value: String →
7        value.toUpperCase()
8    }
9    ⌂ val result2 = hello( value: "Eko") { value: String →
10       value.toUpperCase()
11    }
12  ⌂ }
```

---

# Anonymous Functions

---

# Anonymous Function

- Lambda akan menganggap baris terakhir di blok lambda sebagai hasil kembalian
- Kadang kita butuh membuat lambda yang se flexible function, dimana kita bisa mengembalikan hasil dimanapun
- Untuk hal ini, kita bisa menggunakan Anonymous Function
- Anonymous Function sebenarnya mirip dengan lambda, hanya cara membuatnya saja yang sedikit beda, masih menggunakan kata kunci func

# Kode : Anonymous Function

```
1 ►  fun main() {  
2     val anonymousUpper = fun(value: String): String {  
3         if (value.isBlank()) {  
4             return "Ups"  
5         }  
6         return value.toUpperCase()  
7     }  
8  
9     val result1 = hello( value: "Eko", anonymousUpper)  
10 }
```

---

# Kode : Anonymous Function as Parameter

```
1 ► fun main() {  
2     val result1 = hello(value: "Eko", fun(value: String): String {  
3         if (value.isBlank()) {  
4             return "Ups"  
5         }  
6         return value.toUpperCase()  
7     })  
8 }  
9
```

---

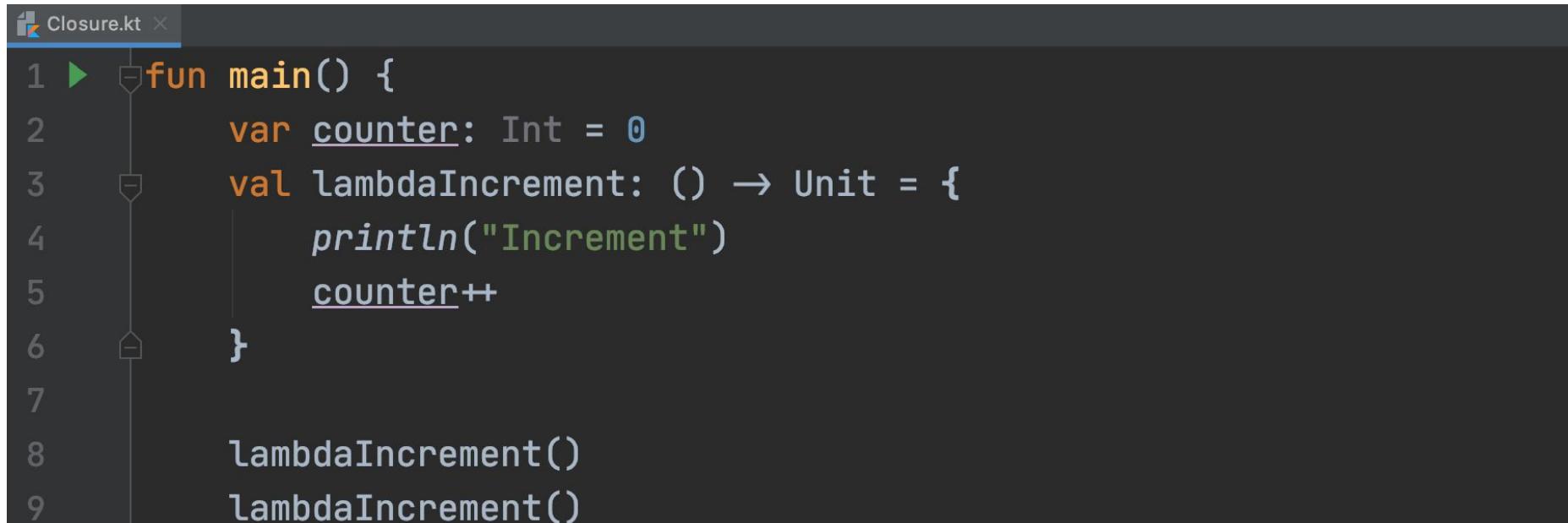
# Closure

---

# Closures

- Closure adalah kemampuan sebuah function, lambda atau anonymous function berinteraksi dengan data-data disekitarnya dalam scope yang sama
- Harap gunakan fitur closure ini dengan bijak saat kita membuat aplikasi

# Kode : Closure



The screenshot shows a code editor window titled "Closure.kt". The code is as follows:

```
1 fun main() {
2     var counter: Int = 0
3     val lambdaIncrement: () -> Unit = {
4         println("Increment")
5         counter++
6     }
7     lambdaIncrement()
8     lambdaIncrement()
9 }
```

The code defines a main function that initializes a variable `counter` to 0. It then creates a closure `lambdaIncrement` that prints "Increment" and increments the `counter`. Finally, it calls the `lambdaIncrement` function twice.

---

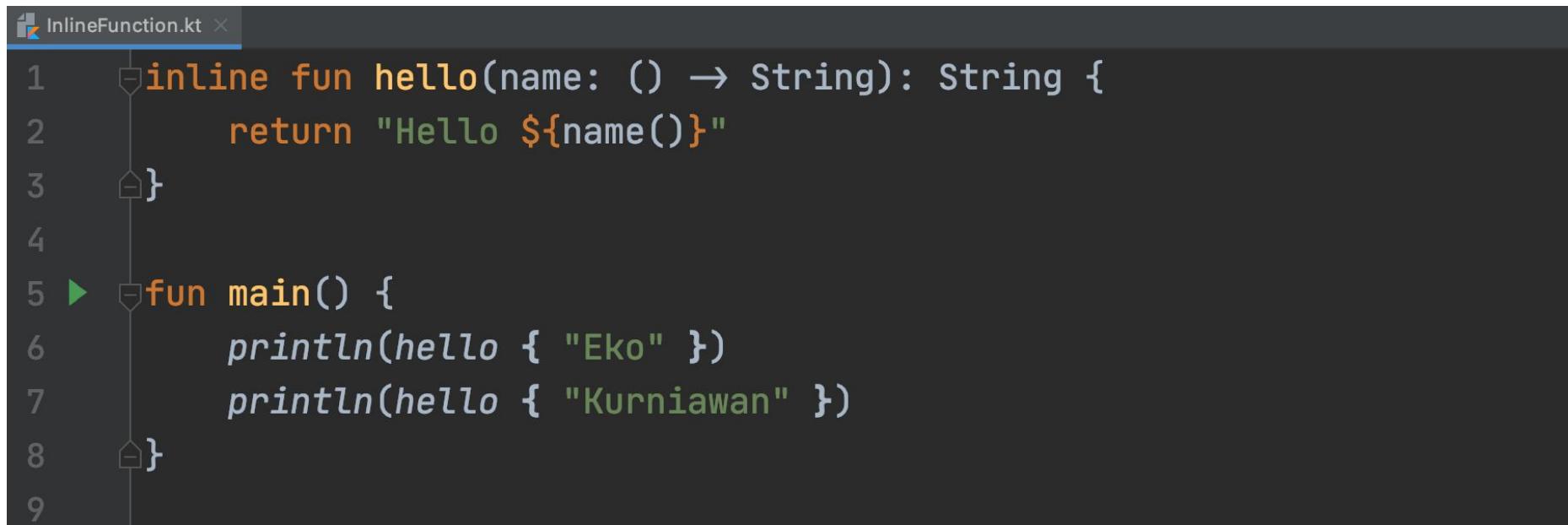
# Inline Function

---

# Inline Functions

- Menggunakan Higher-Order Function adalah salah satu fitur yang sangat berguna.
- Namun penggunaan Higher-Order Function akan berdampak terhadap performa saat aplikasi berjalan, karena harus membuat object lambda berulang-ulang. Jika penggunaannya tidak terlalu banyak mungkin tidak akan terasa, tapi jika banyak sekali, maka akan terasa impact nya
- Inline Functions adalah kemampuan di Kotlin yang mengubah Higher-Order Function menjadi Inline Function
- Dimana dengan Inline Function, code di dalam Higher-Order Function akan di duplicate agar pada saat runtime, aplikasi tidak perlu membuat object lambda berulang-ulang

# Kode : Inline Function

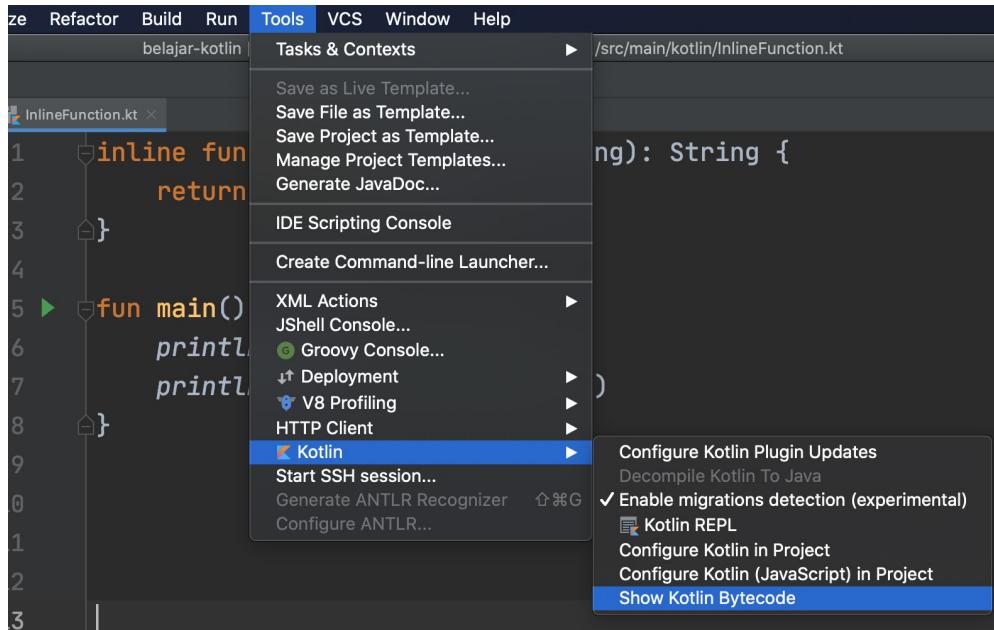


The screenshot shows a code editor window with a dark theme. The file tab at the top is labeled "InlineFunction.kt". The code itself is as follows:

```
1  inline fun hello(name: () -> String): String {  
2      return "Hello ${name()}"  
3  }  
4  
5 ▶ fun main() {  
6     println(hello { "Eko" })  
7     println(hello { "Kurniawan" })  
8  }
```

The code defines an inline function named `hello` that takes a lambda expression as a parameter. It returns a string concatenating "Hello" and the result of the lambda. The `main` function then calls `hello` with two different arguments: "Eko" and "Kurniawan". The code editor uses syntax highlighting to distinguish between keywords, strings, and comments.

# Melihat Bytecode Kotlin

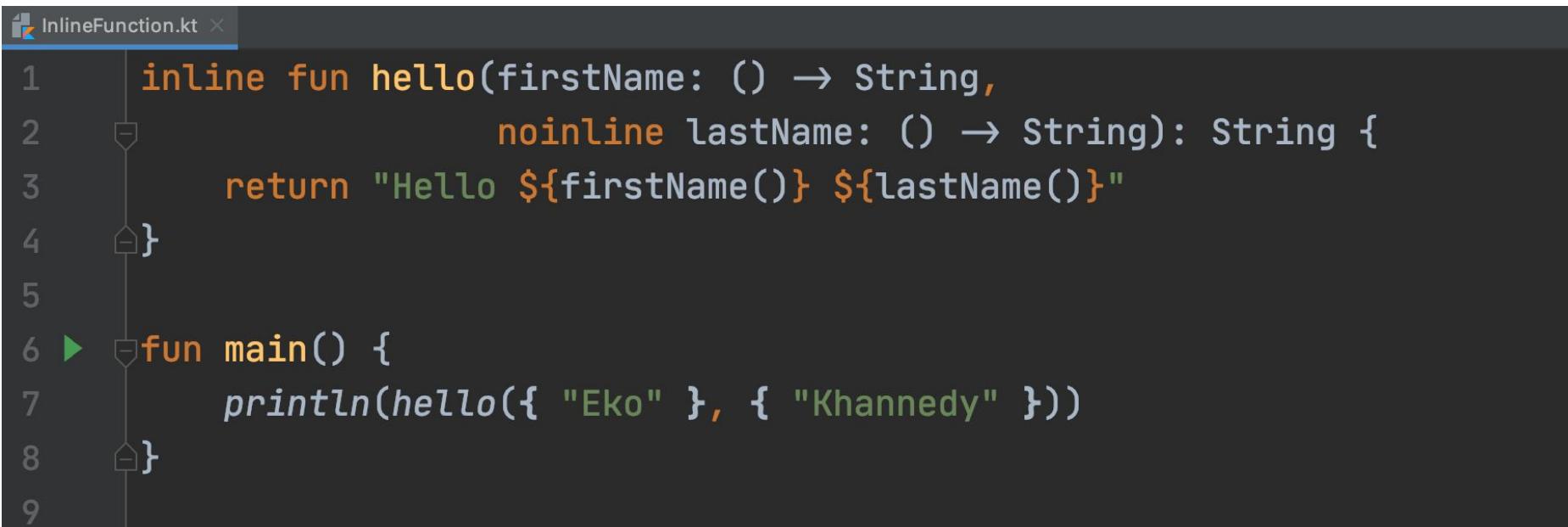


---

# Noinline

- Saat menandai bahwa function adalah inline, maka secara otomatis semua parameter akan menjadi inline
- Jika kita ingin memberi tahu bahwa jangan melakukan inline terhadap parameter, kita bisa menandai parameter tersebut dengan kata kunci noinline

# Kode : Noinline



The screenshot shows a code editor window with a dark theme. The title bar says "InlineFunction.kt". The code is as follows:

```
1  inline fun hello(firstName: () -> String,  
2                      noinline lastName: () -> String): String {  
3      return "Hello ${firstName()} ${lastName()}"  
4  }  
5  
6 ▶ fun main() {  
7     println(hello({ "Eko" }, { "Khannedy" }))  
8  }
```

The code defines an inline function `hello` that takes two parameters: `firstName` and `lastName`. The `firstName` parameter is marked with `inline`, and the `lastName` parameter is marked with `noinline`. The function returns a string concatenating the results of calling each function. Below the function definition, there is a main function `main` that prints the result of calling `hello` with the arguments "Eko" and "Khannedy".

---

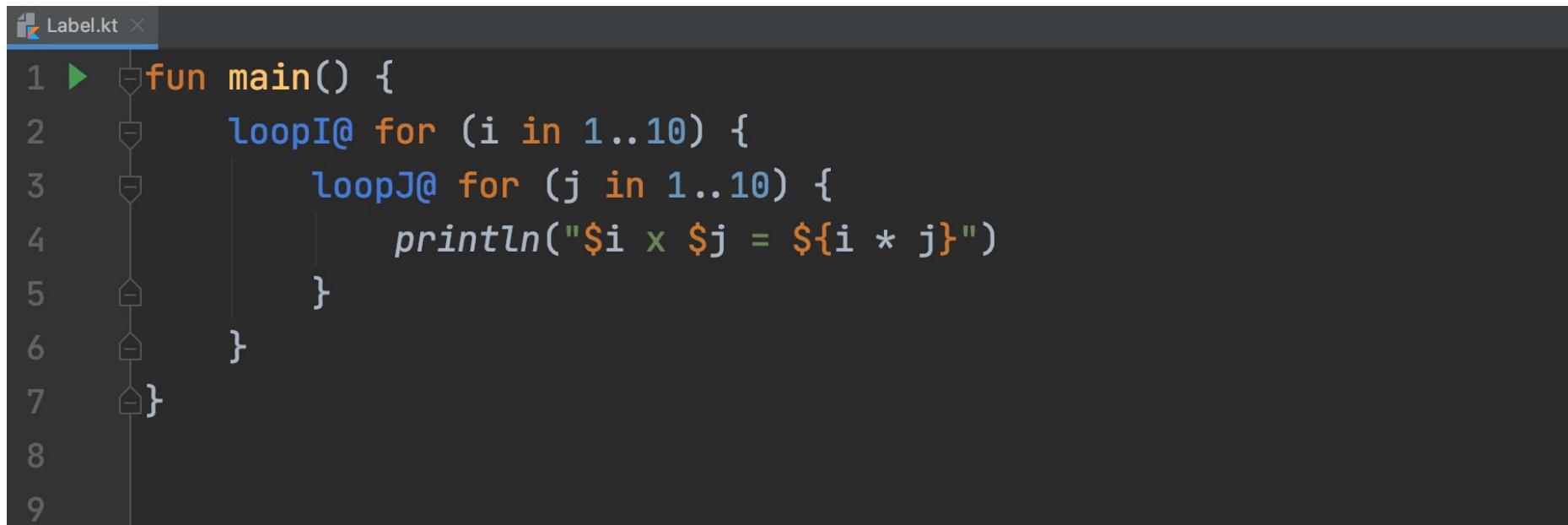
Label



# Label

- Label adalah penanda
- Semua expression di Kotlin bisa ditandai dengan label
- Untuk membuat label di Kotlin, cukup menggunakan nama label lalu diikuti dengan karakter @

# Kode : Label



The screenshot shows a code editor window with a dark theme. The title bar says "Label.kt". The code is a Kotlin program with the following structure:

```
1 ►  fun main() {  
2     loopI@ for (i in 1..10) {  
3         loopJ@ for (j in 1..10) {  
4             println("i x j = ${i * j}")  
5         }  
6     }  
7 }
```

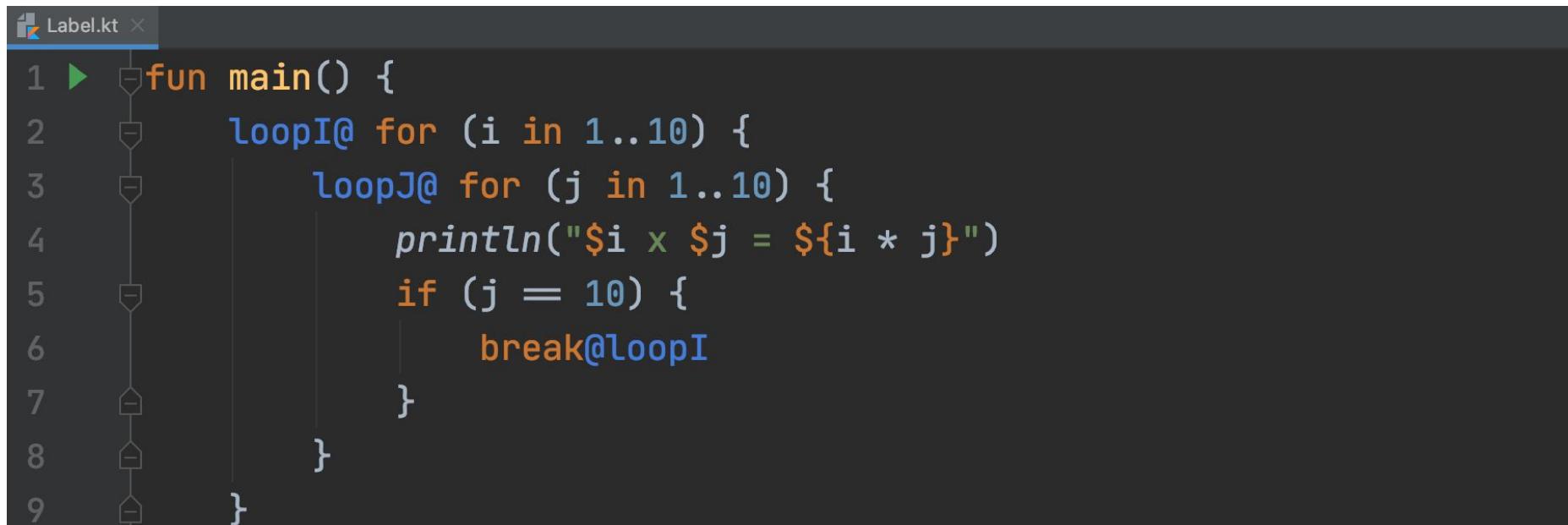
Line 1 has a green play button icon. The code uses labels "loopI@" and "loopJ@" to allow jumping between the two nested loops. The code is annotated with small gray diamond icons above each brace, likely indicating code folding or specific editor features.

---

# Break, Continue dan Return

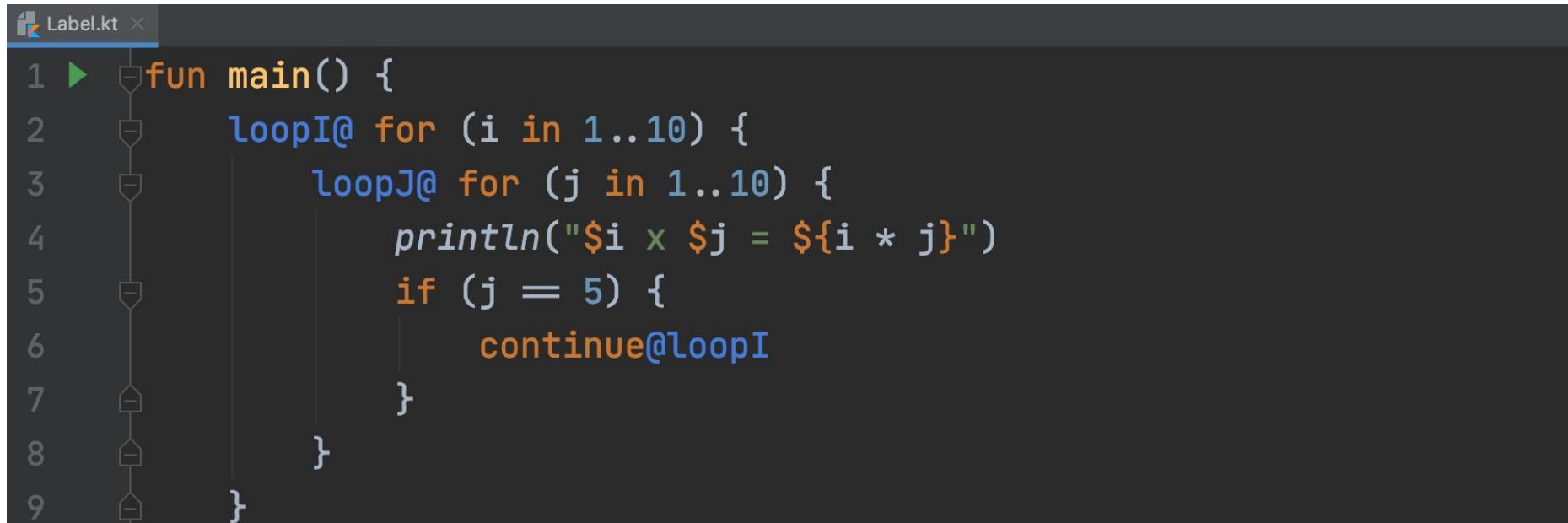
- Salah satu kegunaan label adalah, bisa diintegrasikan dengan break, continue dan return
- Biasanya break, continue dan return hanya bisa menghentikan proses di blok kode tempat mereka berada, namun dengan menggunakan label, kita bisa menentukan ke label mana kode akan berhenti

# Kode : Break to Label



```
Label.kt
1 ►  fun main() {
2     loopI@ for (i in 1..10) {
3         loopJ@ for (j in 1..10) {
4             println("i x j = ${i * j}")
5             if (j == 10) {
6                 break@loopI
7             }
8         }
9     }
}
```

# Kode : Continue to Label

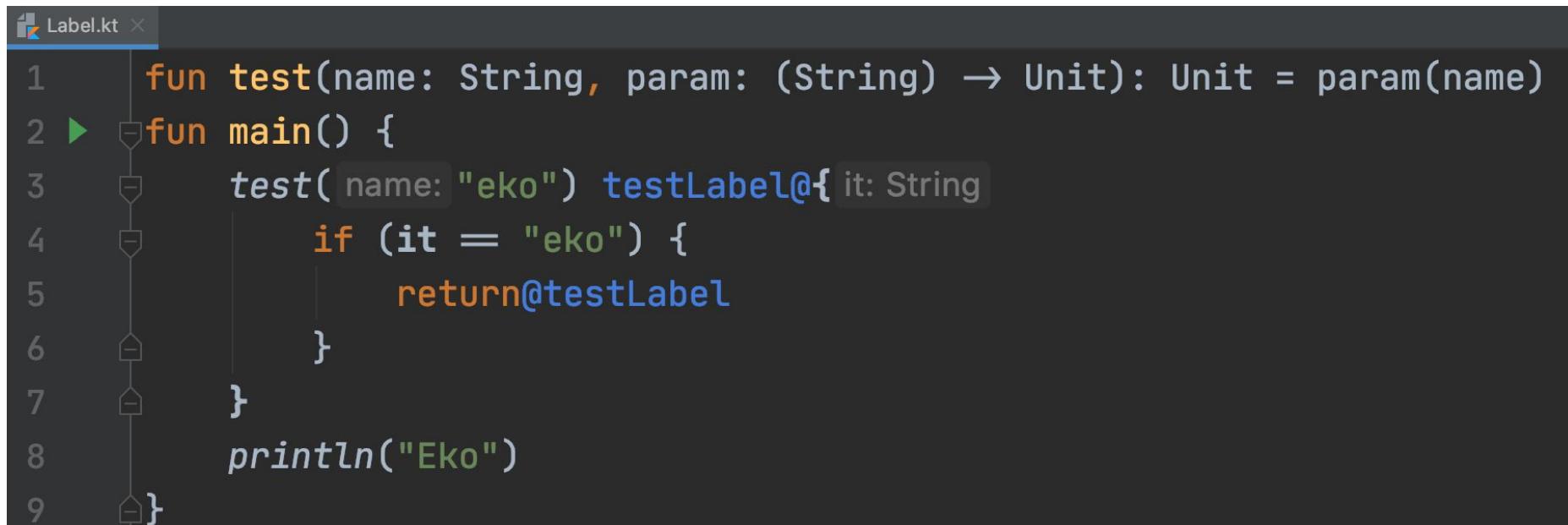


The screenshot shows a code editor window with a dark theme. The file is named "Label.kt". The code contains a nested loop structure with a "continue" statement:

```
1 fun main() {
2     loopI@ for (i in 1..10) {
3         loopJ@ for (j in 1..10) {
4             println("$i x $j = ${i * j}")
5             if (j == 5) {
6                 continue@loopI
7             }
8         }
9     }
}
```

The code uses labels "loopI@" and "loopJ@" to mark the start of the outer and inner loops respectively, and "continue@loopI" to skip the rest of the inner loop body when j equals 5.

# Kode : Return to Label



The image shows a screenshot of a code editor with a dark theme. A file named "Label.kt" is open. The code contains a function "test" and a main function "main". The "main" function calls "test" with the argument "eko". Inside "test", there is an if-statement that checks if "it" equals "eko". If it does, the code returns from the "test" function using the annotation "@testLabel". The code then continues with a closing brace for the "test" function and a call to "println" with the argument "Eko". The code editor highlights the "return@testLabel" statement with a blue background.

```
1 fun test(name: String, param: (String) → Unit): Unit = param(name)
2 ► fun main() {
3     test(name: "eko") testLabel@{ it: String
4         if (it == "eko") {
5             return@testLabel
6         }
7     }
8     println("Eko")
9 }
```

---

# Package & Import

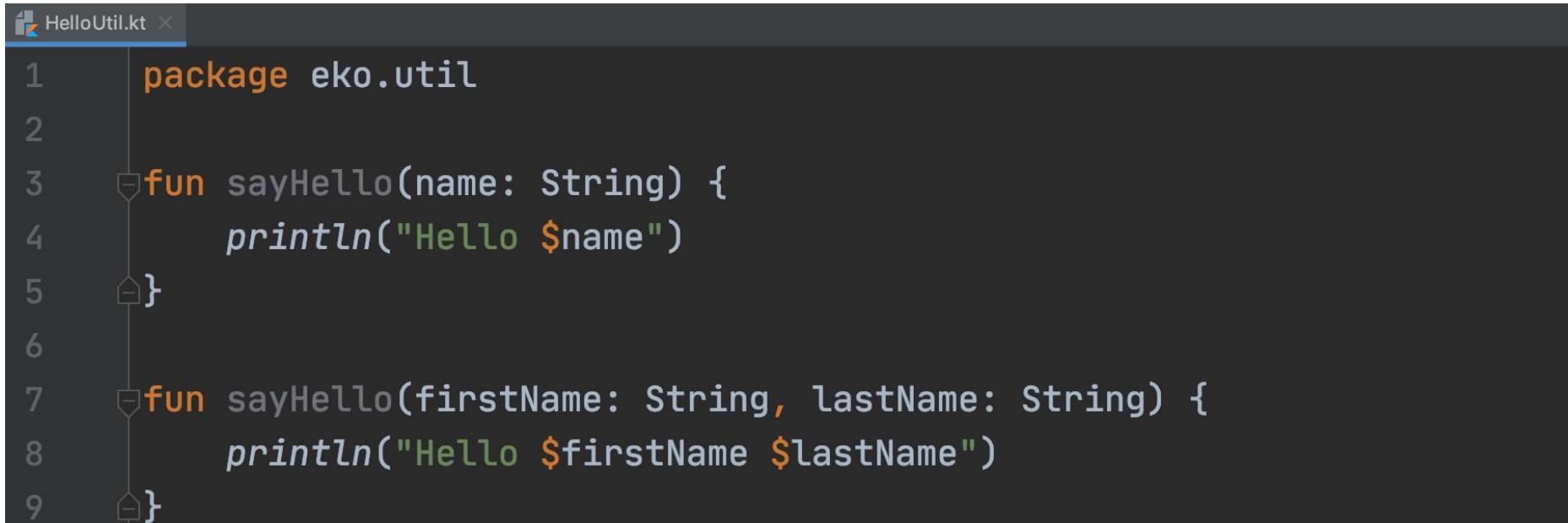
---

# Package

- Package adalah tempat yang bisa digunakan untuk mengorganisir kode program yang kita buat di Kotlin
- Dengan menggunakan package, kita bisa merapikan kode program Kotlin yang kita buat
- Penamaan package di Kotlin biasanya menggunakan huruf kecil semua
- Jika ingin membuat sub package bisa menggunakan tanda . (titik)

---

# Kode : Package



A screenshot of a code editor showing a file named `HelloUtil.kt`. The code defines two functions in the package `eko.util`:

```
1 package eko.util
2
3     fun sayHello(name: String) {
4         println("Hello $name")
5     }
6
7     fun sayHello(firstName: String, lastName: String) {
8         println("Hello $firstName $lastName")
9     }
```

---

# Import

- Secara standar, file Kotlin hanya bisa mengakses file Kotlin lainnya yang berada dalam package yang sama
- Jika kita ingin mengakses file Kotlin yang berada diluar package, maka kita bisa menggunakan Import
- Saat melakukan import, kita bisa memilih ingin meng-import bagian tertentu, atau semua file

# Kode : Import



The screenshot shows a code editor window with a dark theme. The title bar says "Import.kt". The code in the editor is:

```
1 import eko.util.sayHello
2
3 ► □ fun main() {
4     sayHello(name: "Eko")
5     eko.util.sayHello("Eko", "Kurniawan")
6 }
7
8
9
```

The code uses the `sayHello` function from the `eko.util` package. The first call to `sayHello` is a simple call with a single string argument. The second call is a standard function call with two arguments: a string and another string.

---

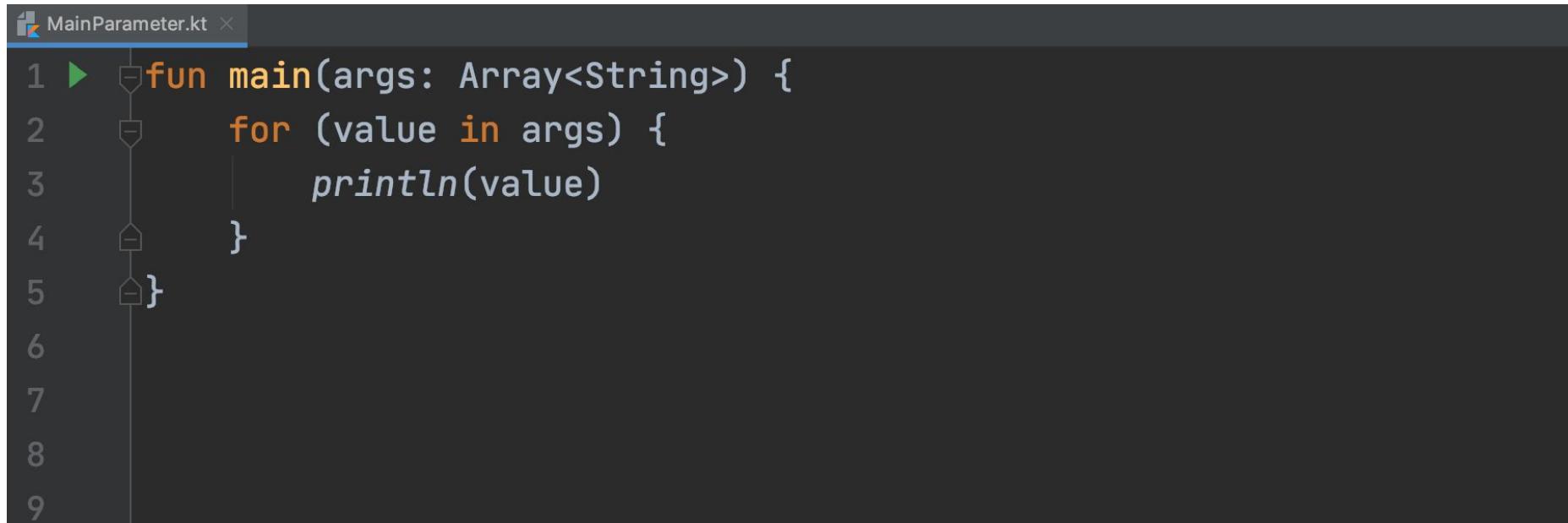
# Main Parameters

---

# Main Parameters

- Kadang saat menjalankan program, kita butuh input parameter dari luar
- Bisa untuk konfigurasi program, ataupun yang lainnya
- Kotlin mendukung parameter untuk main function

# Kode : Main Parameters



```
MainParameter.kt
1 ►  fun main(args: Array<String>) {
2     for (value in args) {
3         println(value)
4     }
5 }
```

---

# Komentar

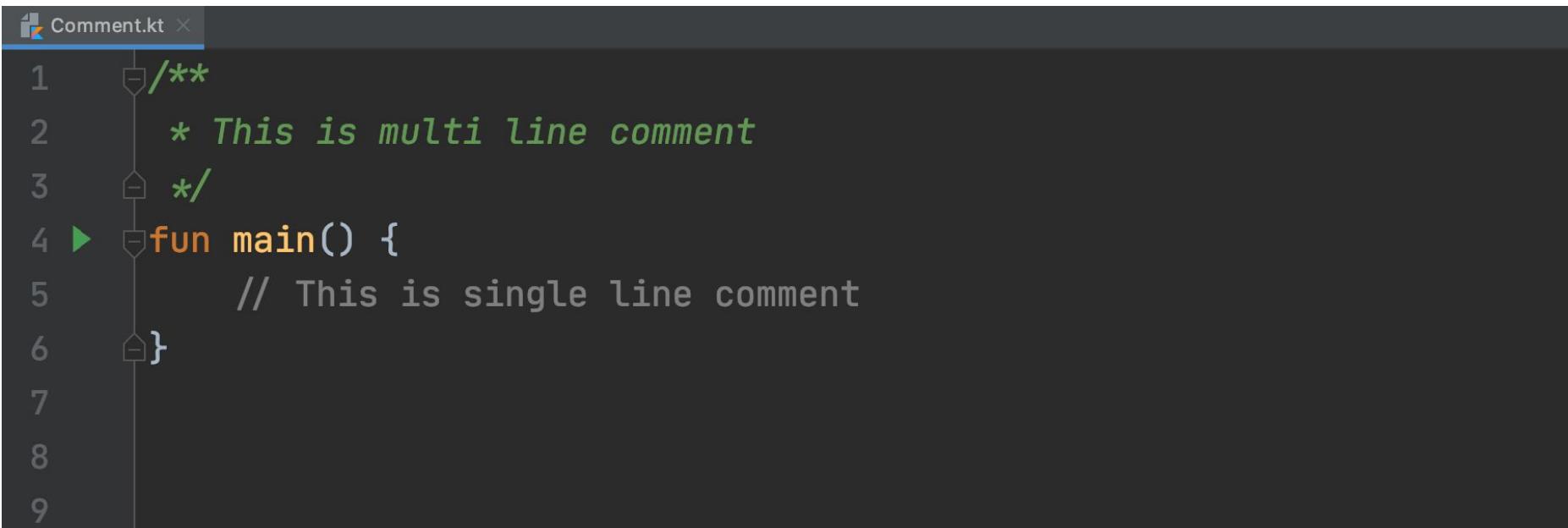
---

# Komentar

- Komentar terbaik pada kode adalah kode itu sendiri
- Saat membuat kode, kita perlu membuat kode semudah mungkin untuk dibaca
- Namun kadang juga kita butuh menambahkan komentar di kode Kotlin kita

---

# Kode : Komentar



```
Comment.kt
1  /**
2   * This is multi line comment
3   */
4  fun main() {
5      // This is single line comment
6  }
```

---

# Materi Selanjutnya?

---

# Materi Selanjutnya?

- Kotlin Object Oriented Programming
- Kotlin Generic
- Kotlin Collection
- Kotlin Coroutine

---

# Eko Kurniawan Khannedy

- Telegram : @khannedy
- Facebook : fb.com/khannedy
- Twitter : twitter.com/khannedy
- Instagram : instagram.com/programmerzamannow
- Youtube : youtube.com/c/ProgrammerZamanNow
- Email : echo.khannedy@gmail.com

---

Kotlin 1.4

# Mixing Named & Positional Arguments



# Kotlin 1.4

- <https://blog.jetbrains.com/kotlin/2020/03/kotlin-1-4-m1-released/>
- <https://blog.jetbrains.com/kotlin/2020/05/1-4-m2-standard-library/>