MSIB magang dan studi independen bersertifikat | INFINITE LEARNING

Kotlin Fundamental

How to write Kotlin code

INFINITE LEARNING

What are Fundamentals?

Why efficient?

Should only use the fundamentals?

At the end of the framework, why learn fundamentals?

https://medium.com/purwadhikaconnect/haruskah-belajar-coding-dari-fundamental-bcaf434b032b

INFINITE
LEARNING

Table of Content

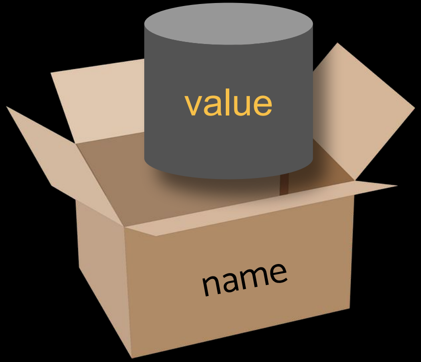
- Variable & Data types
- Functions
- If Expressions
- Nullable Types
- Safe calls & Elvis Operator
- String templates

Variable & Data types

Variable

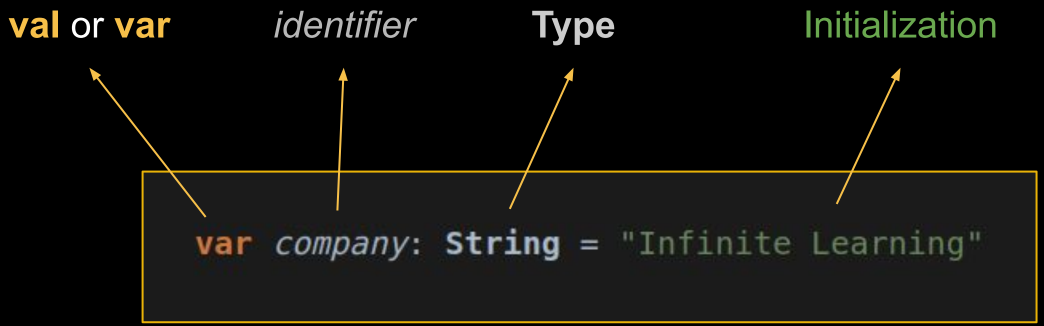
Variables are an important part of any programming.

A **variable** is a name that is entered into a computer memory location that is used to store a value in a computer program, then the name is used to retrieve the stored value and use it in the program.



Variable in Kotlin

In Kotlin variables will require the keywords **var** or **val**, *identifier*, *type* and *initialization*.



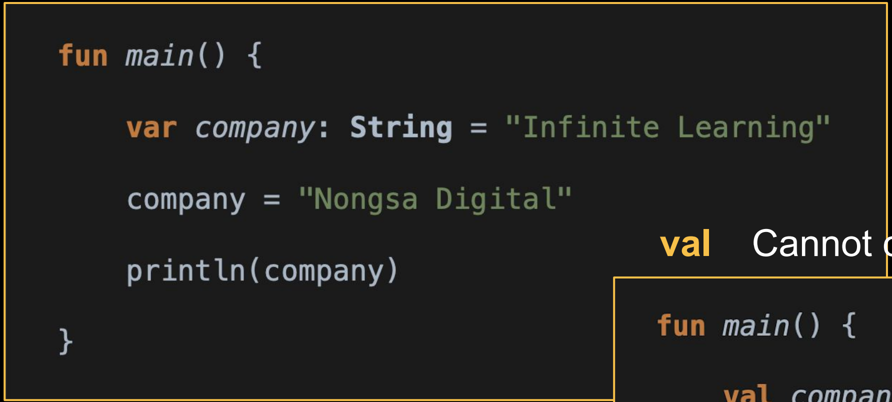
```kotlin
var company: String = "Infinite Learning"
```

Kotlin supports *type inference* so we are allowed to not write data types explicitly.

```kotlin
var company = "Infinite Learning"
```

Variable in Kotlin

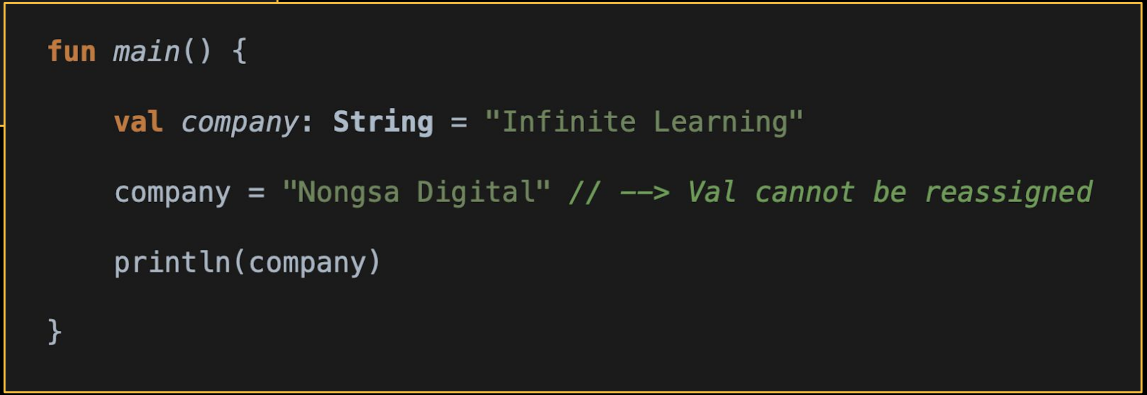
var Can change the initialized value. **1**



```kotlin
fun main() {

    var company: String = "Infinite Learning"

    company = "Nongsa Digital"

    println(company)

}
```

Let's code…

val Cannot change the value that was previously initialized. **2**



```kotlin
fun main() {

    val company: String = "Infinite Learning"

    company = "Nongsa Digital" // --> Val cannot be reassigned

    println(company)

}
```

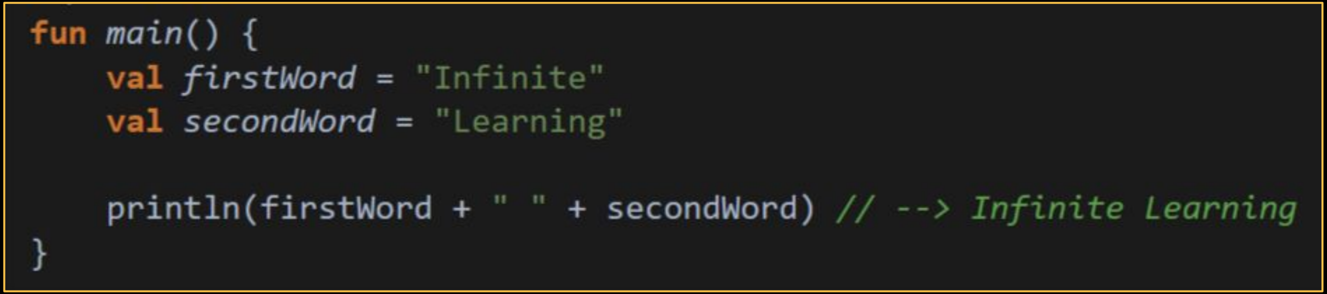
Any question?

Data Types

Data Types

Data type is a data classifier based on the **type** of data owned by the **variable**

The data type also determines the **operations** that can be performed on a variable and how the **value** of a variable is **stored**



```kotlin
fun main() {
    val firstWord = "Infinite"
    val secondWord = "Learning"

    println(firstWord + " " + secondWord) // --> Infinite Learning
}
```

Data Types

Data type is a classification of data based on the type of data.

Data Types	Description	Example
Character	Just one character	'A'
String	Just Text	"Kampus Merdeka"
Boolean	Just two values	true / false
Numbers	Double, Long, Int, Short, Byte	128, -15, 2.7182818284
Array	Save multiple objects	arrayOf(2, 4, 6, 9, "Infinite Learning" , true)

Data Types : Character

- Char is a data type whose value type is text.
- Characters type are represented using the **Char**.
- Char type variable define can use single quotes (**' '**)
- Char can only be used to store **single** characters

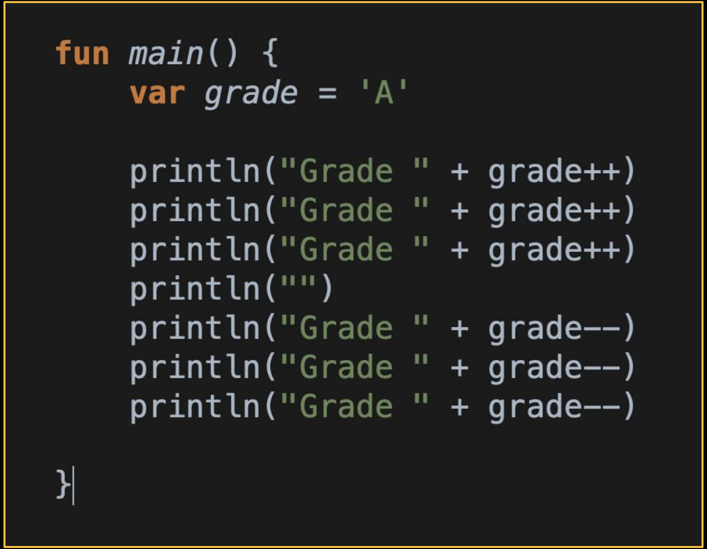
```
var grade: Char = 'A'
```

Data Types : Character

INFINITE LEARNING

Increment (++) and decrement (--) operations on the Char data type

Let’s code…



```
fun main() {
    var grade = 'A'

    println("Grade " + grade++)
    println("Grade " + grade++)
    println("Grade " + grade++)
    println("")
    println("Grade " + grade--)
    println("Grade " + grade--)
    println("Grade " + grade--)

}
```

Data Types : String

INFINITE LEARNING

- Not only Char, the **String** data type is also a text value. The difference is, String can hold several characters in it.
- String data type are represented using the **String**.
- Defines a variable with double quotes (**" "**).

```
val stringText  = "Kotlin Language"
```

Data Types : String

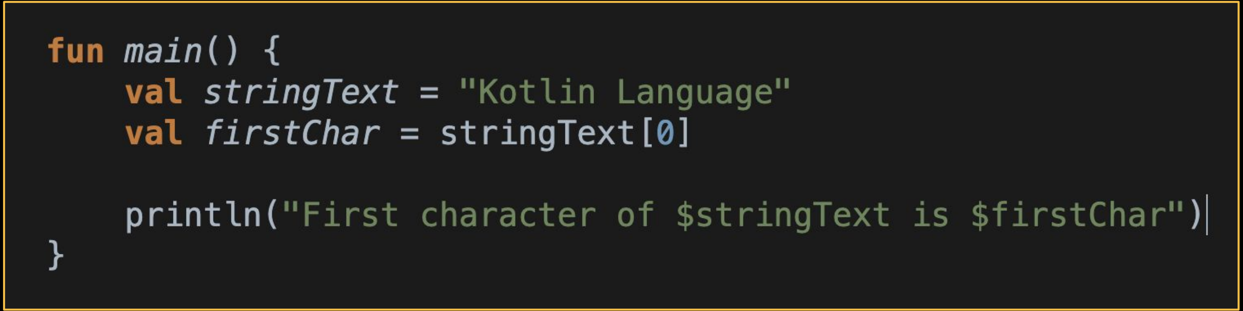
INFINITE LEARNING

Basically a set of characters in a String value is in the form of an Array, so we can retrieve a single character by using **indexing**.

What is Indexing?

Indexing is an easy way to get elements in a collection by using the index or position of the element. The position of an element starts from 0.

Let's code…



```kotlin
fun main() {
    val stringText = "Kotlin Language"
    val firstChar = stringText[0]

    println("First character of $stringText is $firstChar")
}
```

Data Types : String

Kotlin have two kinds of string, Escaped String and Raw String.

Escaped string is declared within double quote (" ") and may contain escape characters like:

\t	: add tabs to the text.
\n	: create a new line in the text.
\'	: adds a single quote character to the text.
\"	: adds a double quote character to the text.
\\	: adds a backslash character to the text.

Data Types : String

INFINITE LEARNING

Escaped string is declared within double quote (" ") and may contain escape characters like **\t**, **\n**, **\'**, **\"**, \.

Let's code…

```
fun main() {

    val stringText  = "Kampus Merdeka \nby \"Infininte Learning\""
    println(stringText)

}
```

Data Types : String

INFINITE LEARNING

Raw string is declared within triple quote (""" """) and may contain multiple lines of text without any escape characters.

Let’s code…

```
fun main() {

    val stringText  = """
    Belajar Bahasa Kotlin
    ---------------------
    Di Infinite Learning
    Bersama Kampus Merdeka
    """

    println(stringText)

}
```

INFINITE LEARNING

Any question?

Data Types : Boolean

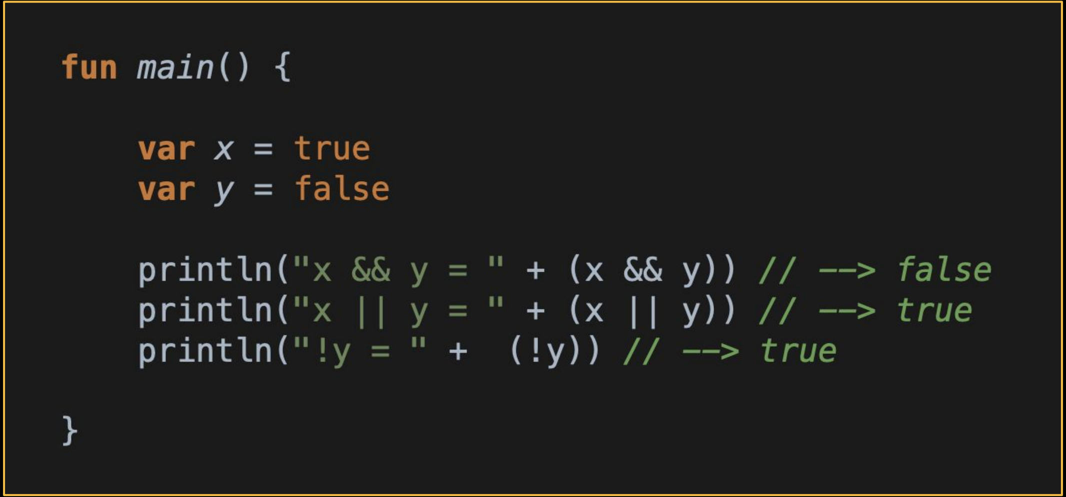
INFINITE LEARNING

Boolean is a data type that has only two values, **true** and **false**. There are 3 (three) operators that can be used in Boolean.

Operator	Name	Description	Example
&&	Logical and	Returns true if both operands are true	x && y
||	Logical or	Returns true if either of the operands is true	x || y
!	Logical not	Reverse the result, returns false if the operand is true	!x

Data Types : Boolean

Let’s code…



```
fun main() {

    var x = true
    var y = false

    println("x && y = " + (x && y)) // --> false
    println("x || y = " + (x || y)) // --> true
    println("!y = " +  (!y)) // --> true

}
```

Data Types : Boolean Expression

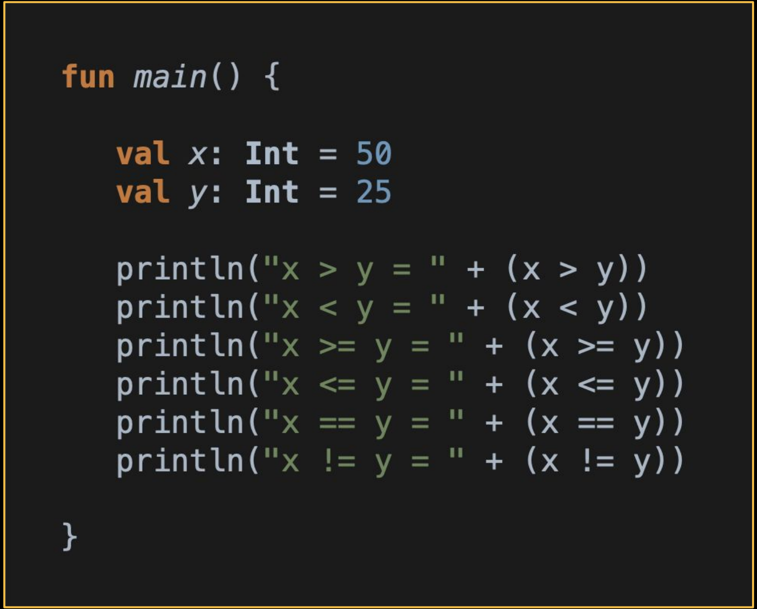
Boolean variable mostly used with checking conditions with **if...else expressions**. A boolean expression makes use of relational operators, for example:

> **<=**

< **==**

>= **!=**

Let’s code…



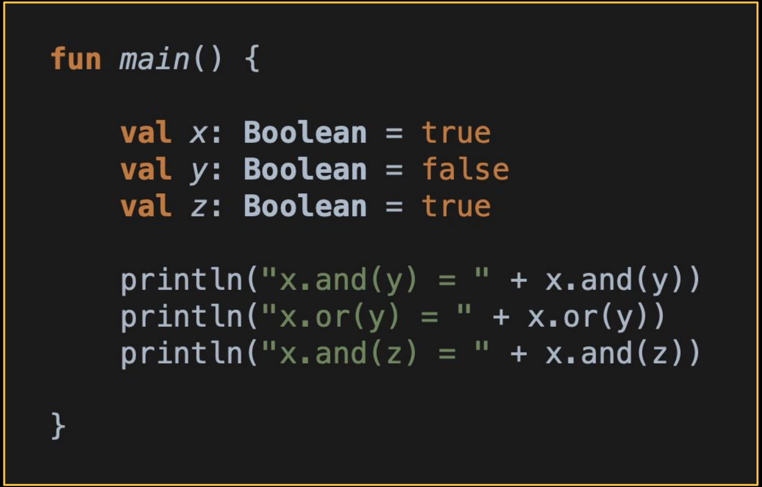
```
fun main() {

    val x: Int = 50
    val y: Int = 25

    println("x > y = " + (x > y))
    println("x < y = " + (x < y))
    println("x >= y = " + (x >= y))
    println("x <= y = " + (x <= y))
    println("x == y = " + (x == y))
    println("x != y = " + (x != y))

}
```

Data Types : Boolean Functions

INFINITE LEARNING

Kotlin provides **and()** and **or()** functions to perform logical AND and logical OR operations between two boolean operands.

Let's code…



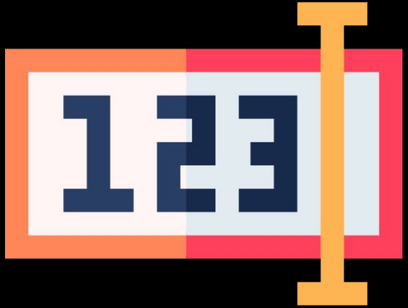
```
fun main() {

    val x: Boolean = true
    val y: Boolean = false
    val z: Boolean = true

    println("x.and(y) = " + x.and(y))
    println("x.or(y) = " + x.or(y))
    println("x.and(z) = " + x.and(z))

}
```

Data Types : Numbers

INFINITE LEARNING

Number data types are used to define variables which hold numeric values.

Some of the default types that represent Numbers are **Byte**, **Short**, **Int**, **Long**, **Float**, and **Double**.



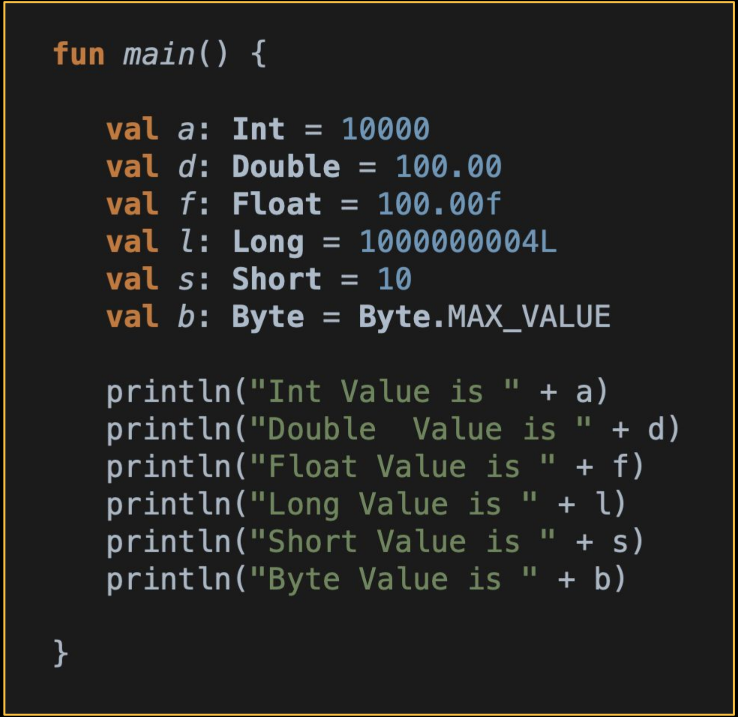
Data Types : Numbers

Each Number data type has a different size (bit unit), depending on the amount of value that can be stored. As shown in the following table:

Data Type	Size (bits)	Data Range
Byte	8 bit	-128 to 127
Short	16 bit	-32768 to 32767
Int	32 bit	-2,147,483,648 to 2,147,483,647
Long	64 bit	-9,223,372,036,854,775,808 to +9,223,372,036,854,775,807
Float	32 bit	1.40129846432481707e-45 to 3.40282346638528860e+38
Double	64 bit	4.94065645841246544e-324 to 1.79769313486231570e+308

Data Types : Numbers

Let’s code…



```kotlin
fun main() {

    val a: Int = 10000
    val d: Double = 100.00
    val f: Float = 100.00f
    val l: Long = 1000000004L
    val s: Short = 10
    val b: Byte = Byte.MAX_VALUE

    println("Int Value is " + a)
    println("Double  Value is " + d)
    println("Float Value is " + f)
    println("Long Value is " + l)
    println("Short Value is " + s)
    println("Byte Value is " + b)

}
```

INFINITE LEARNING

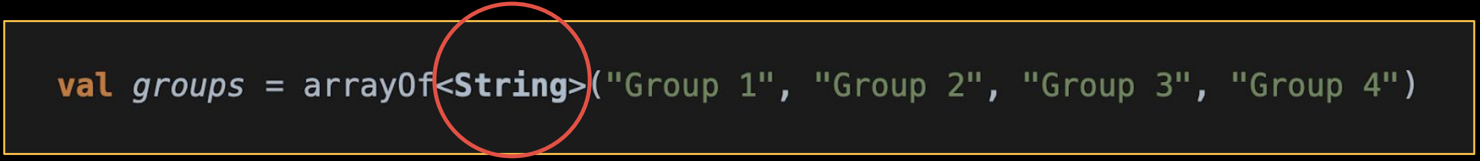
Any question?

Data Types : Array

INFINITE LEARNING

- Array is a data type that can store multiple values or objects in a variable.
- Arrays in Kotlin are represented by the **Array**.
- To create an Array, we can use a library function **arrayOf()**

Optionally with provide a data type as follows:



```kotlin
val groups = arrayOf<String>("Group 1", "Group 2", "Group 3", "Group 4")
```

Get and Set the Elements of an Array

INFINITE LEARNING

Use the index number inside square brackets (**[ ]**) for access an array element. Array index starts with zero (0). So if access 1th element of the array then give number 0 as the index.

Kotlin provides **get()** and **set()** member functions to get and set the value at a particular index.

Get the value

```
group[0]
group.get(0)
```

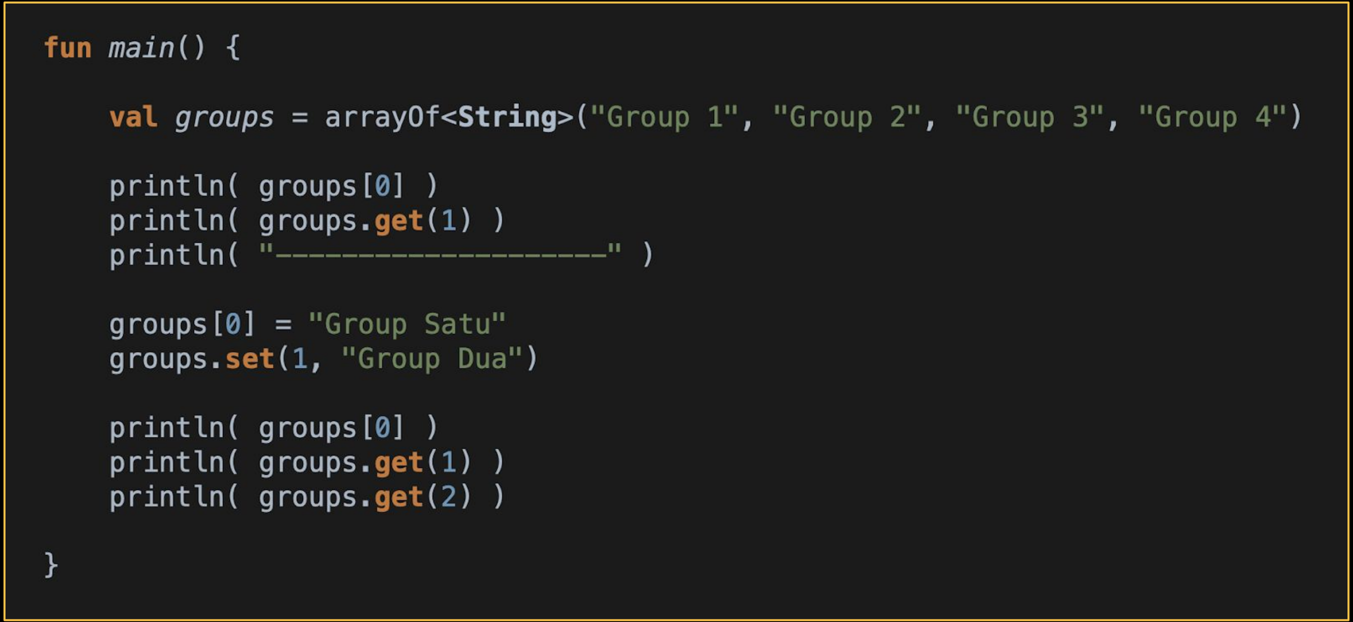
Set the value

```
groups[0] = "Group 1"
groups.set(1, "Group Dua")
```

Get and Set the Elements of an Array

INFINITE LEARNING

Let’s code…



```kotlin
fun main() {

    val groups = arrayOf<String>("Group 1", "Group 2", "Group 3", "Group 4")

    println( groups[0] )
    println( groups.get(1) )
    println( "--------------------" )

    groups[0] = "Group Satu"
    groups.set(1, "Group Dua")

    println( groups[0] )
    println( groups.get(1) )
    println( groups.get(2) )

}
```

Primitive type Arrays

INFINITE LEARNING

Kotlin also has built-in factory methods to create arrays of primitive data types.

intArrayOf() : **IntArray**

booleanArrayOf() : **BooleanArray**

charArrayOf() : **CharArray**

longArrayOf() : **LongArray**

shortArrayOf() : **ShortArray**

byteArrayOf() : **ByteArray**

Let's code…

```
val groups = intArrayOf(1, 2, 3, 4)
```

```
fun main() {
    val intArray = intArrayOf(2, 9, 11, 15)  // [2, 9, 11, 15]
    intArray[2] = 30                         // [2, 9, 30, 15]

    print(intArray[2])
}
```

INFINITE
LEARNING

Any question?

INFINITE LEARNING

Functions

Functions

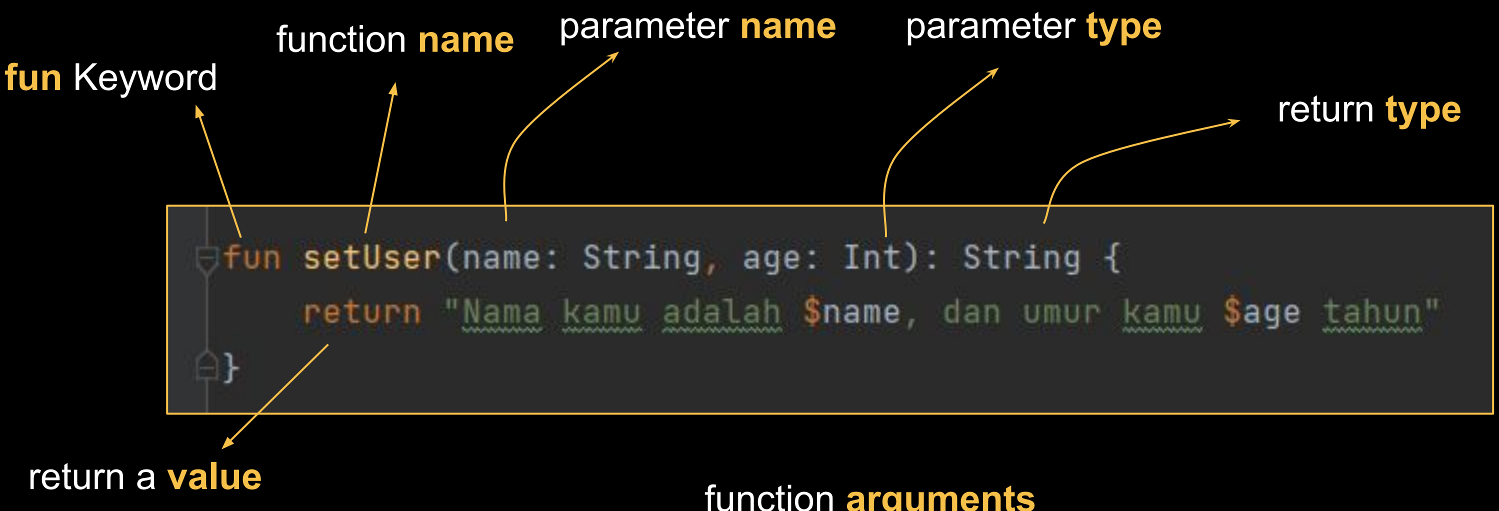
- A function is a procedure that is related to messages and objects
- Functions are created to perform specific tasks
- A function can be said to be a mini program that will run when called
- Reusable and decomposition

Functions : Create a function

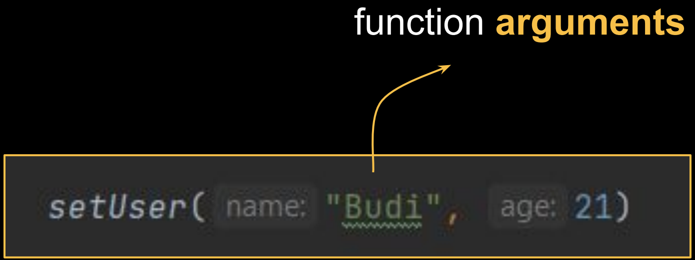
INFINITE LEARNING

- Function declarations with the keyword **fun**
- Continue the **name** of the desired function
- Setting **parameters** (optional)
  - Parameter **name**
  - **Tipe** parameter separated by (:)
  - Each parameter of a function is separated by a **comma** (,)
- Specify the return **type** of the function (optional)
- Function **body is inside { }** (in which there is an **expression** or **statement** to run)

Functions : Structure



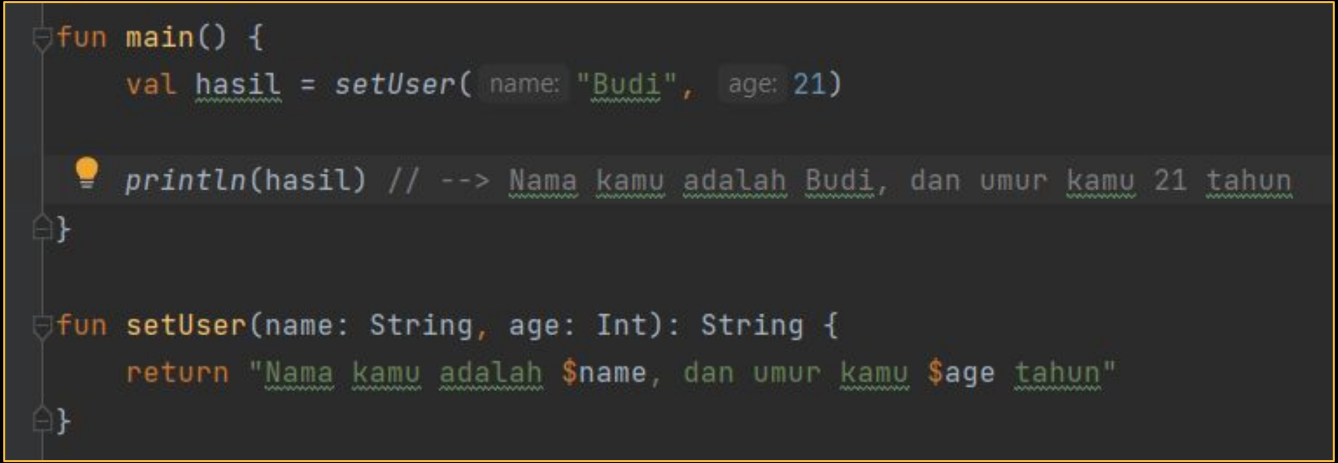
```
fun setUser(name: String, age: Int): String {
    return "Nama kamu adalah $name, dan umur kamu $age tahun"
}
```



```
setUser( name: "Budi", age: 21)
```

Functions

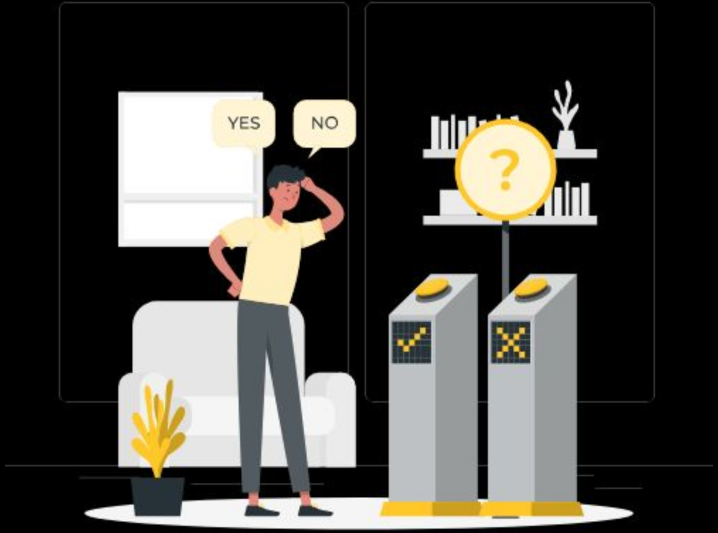
Let’s to code



```
fun main() {
    val hasil = setUser( name: "Budi",  age: 21)

    println(hasil) // --> Nama kamu adalah Budi, dan umur kamu 21 tahun
}

fun setUser(name: String, age: Int): String {
    return "Nama kamu adalah $name, dan umur kamu $age tahun"
}
```

INFINITE LEARNING

If Expressions



If Expressions

INFINITE LEARNING

- **If expression** is used when initializing the value of a variable based on a condition.
- If expression is represented by the keyword **if.**
- *if* is used to test a condition to run a process.
- *if* will execute a statement or expression if the evaluation result in the *if* block is **true**. Otherwise, it will be passed if it evaluates to **false**.

If Statement

INFINITE LEARNING

Let's start with traditional **if...else** statement.

Why is it called an **If statement**? because If **doesn't return any value**, it's a command to print the data to the screen. (Just branching)

```
val condition = true

if (condition) {
    println("code block to be executed if condition is true")
} else {
    println("code block to be executed if condition is false")
}
```

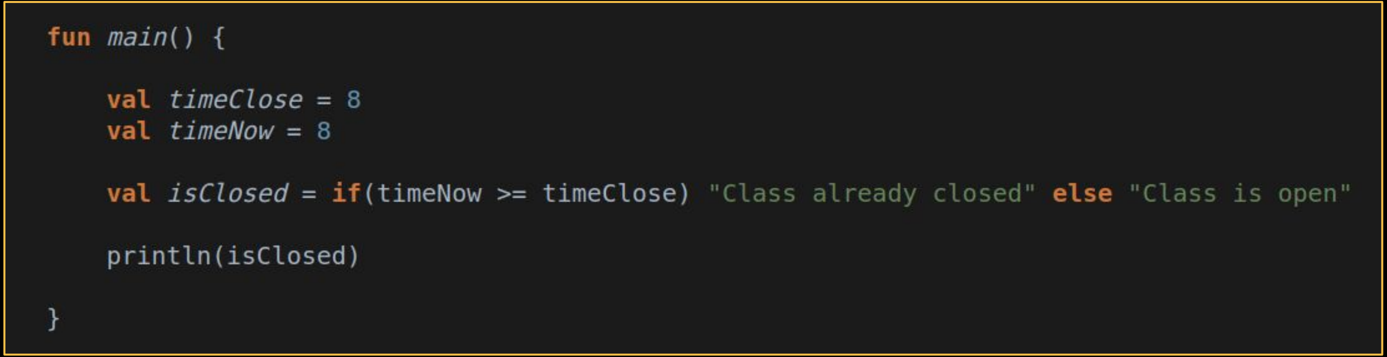
Come on, try the code!

if…else Expression

INFINITE LEARNING

If in Kotlin can also be used as an expression. Let's try using an if statement that can return a value to a variable.
Why is it called an **IF expression**? because the If statement that **returns a value** and is **stored** in a variable

And now let's try for if...else expression:



```
fun main() {

    val timeClose = 8
    val timeNow = 8

    val isClosed = if(timeNow >= timeClose) "Class already closed" else "Class is open"

    println(isClosed)

}
```

if…else Expression

INFINITE LEARNING

We can use else if condition to specify a new condition if the first condition is false.

```kotlin
fun main() {

    val timeOpen = 8
    val timeClose = 12
    val timeNow = 7

    val classStatus = if(timeNow >= timeClose) {
        "Class already closed"
    } else if(timeNow >= timeOpen) {
        "Class is open"
    } else {
        "Class is not open yet"
    }

    println(classStatus)

}
```

Let's code it!

```kotlin
fun main() {
    val time = 16 // waktu saat ini dalam format 24 jam

    val status = if (time < 8 || time >= 20) "Tutup" else "Buka"

    println("Cafe saat ini $status") // output: Cafe saat ini Tutup
}
```

Any question?

INFINITE LEARNING

Nullable Types

Kotlin makes it easy to manage nullable variables.

Kotlin is able to distinguish between null and non-null objects.

Nullable Types

INFINITE LEARNING

"The Billion Dollar Mistake"

Because it is very common and can be fatal, **NPE** is known by the term above.

NullPointerException (NPE) is an error that occurs when accessing or managing the value of an uninitialized variable or a null value variable.

Kotlin comes with easy nullability handling that minimizes the occurrence of NullPointerExceptions.

Nullable Types

Kotlin is able to distinguish between null and non-null objects when they are created.

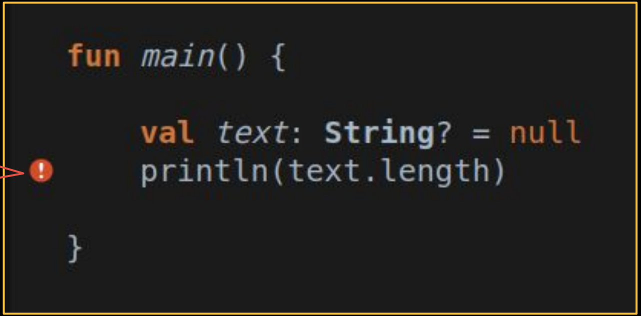
```kotlin
val text: String = null // Null can not be a value of a non-null type String
```

Add sign ? after specifying the object type so it can be null

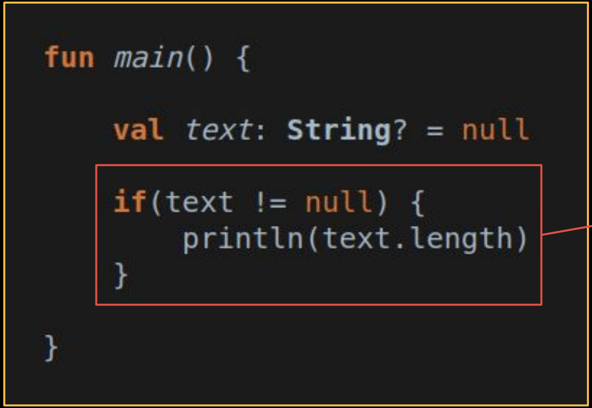
```kotlin
val text: String? = null
```

Nullable Types

Only safe (?.) or non-null asserted (!!.) calls are allowed on a nullable receiver of type String?



```kotlin
fun main() {

    val text: String? = null
    println(text.length)

}
```



```kotlin
fun main() {

    val text: String? = null

    if(text != null) {
        println(text.length)
    }

}
```

Check if the object is null or not.

INFINITE LEARNING

Safe calls & Elvis Operator

Safe calls & Elvis Operator

To use a safe call, replace the (.) with (?.) sign when accessing or managing values from nullable objects.

Safe Call will guarantee that the code we write is safe from **NullPointerException**.

```kotlin
val safeText: String? = null
safeText?.length
```

Handle nullable objects in an easier way using **Safe Calls** and **Elvis Operators**.

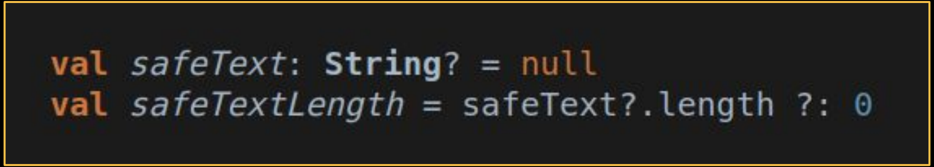
By implementing the safe call as above, the compiler will skip the process if the object is null.

Safe calls & Elvis Operator

INFINITE LEARNING

The **Elvis operator** allows you to set a default value if the object is null.

By adding a (**?:**) at the end of the object value that has used Safe Call, then continued by writing the **default value**.



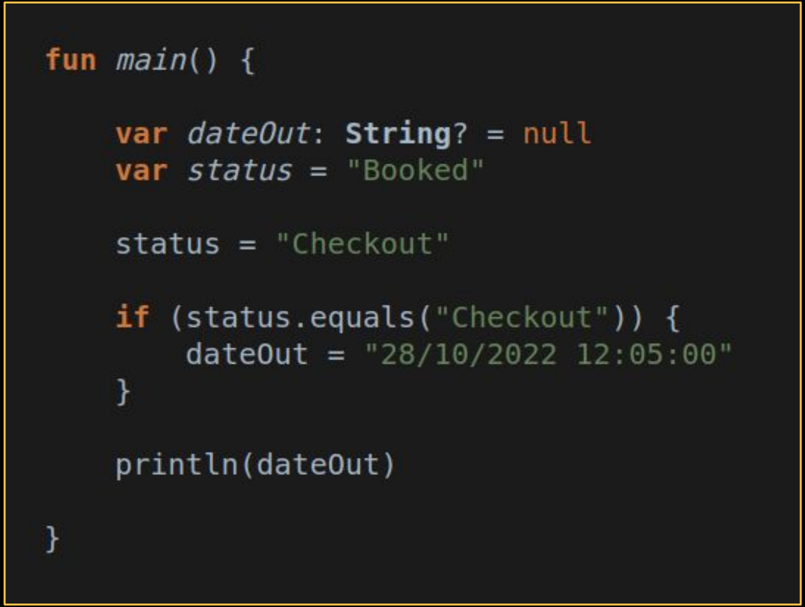
```kotlin
val safeText: String? = null
val safeTextLength = safeText?.length ?: 0
```

Code it!

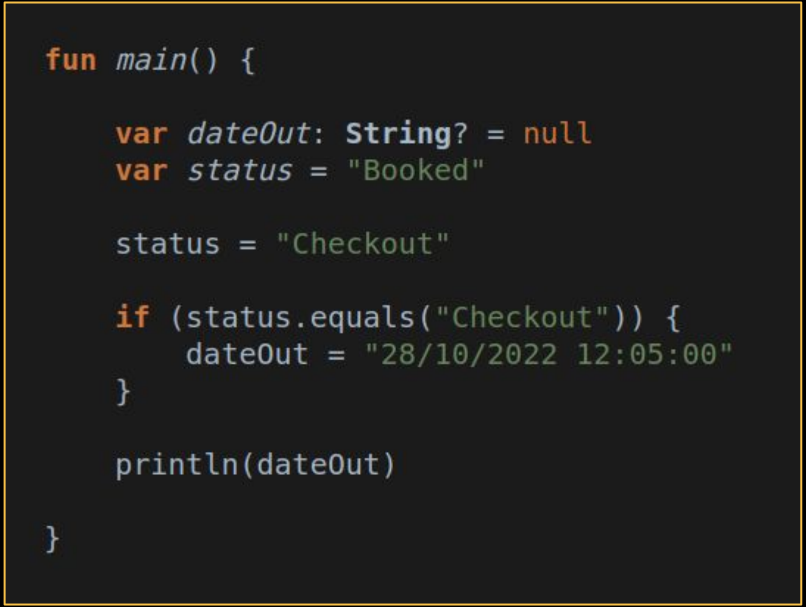
Elvis will return the default value (**0**) if ***safeText*** variable is null.

Safe calls & Elvis Operator

Let's code…



```kotlin
fun main() {

    var dateOut: String? = null
    var status = "Booked"

    status = "Checkout"

    if (status.equals("Checkout")) {
        dateOut = "28/10/2022 12:05:00"
    }

    println(dateOut)

}
```



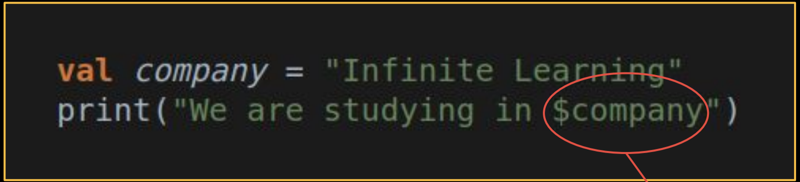
```kotlin
fun main() {

    var dateOut: String? = null
    var status = "Booked"

    status = "Checkout"

    if (status.equals("Checkout")) {
        dateOut = "28/10/2022 12:05:00"
    }

    println(dateOut)

}
```

INFINITE LEARNING

String Template

String Template

String Template is a feature that allows to insert variables into String without concatenation (merging String objects using +)



```kotlin
val company = "Infinite Learning"
print("We are studying in $company")
```

variable

String Template

Let's code…

Without String Template

```
fun main() {

    val company = "Infinite Learning"
    print("We are studying in " + $company)

}
```

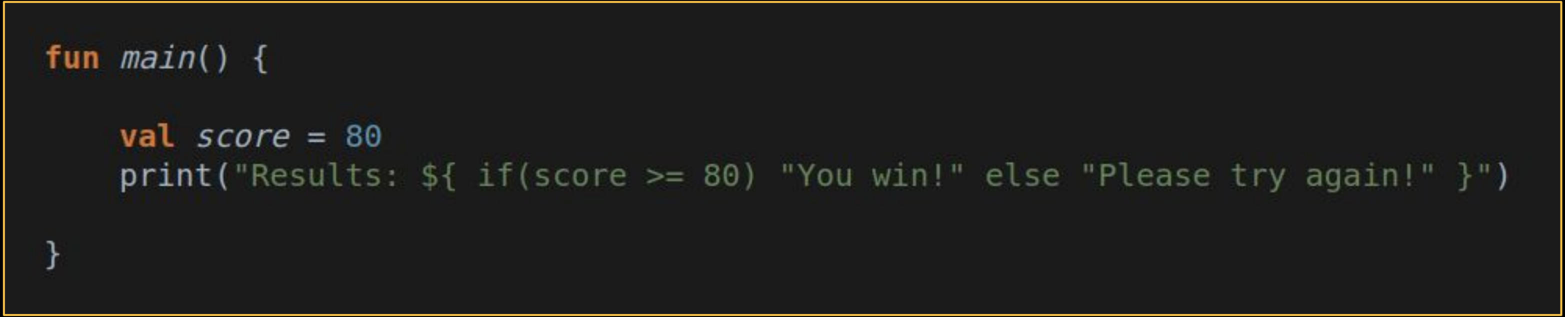
Using String Template

```
fun main() {

    val company = "Infinite Learning"
    print("We are studying in $company")

}
```

String Template

Kotlin also makes it possible to insert expressions into template strings. By inserting the expression into curly braces followed by the $ character.

Let’s code it!



```
fun main() {

    val score = 80
    print("Results: ${ if(score >= 80) "You win!" else "Please try again!" }")

}
```

GUIDING RESOURCE

INFINITE LEARNING

Guiding Resources:

1. https://kotlinlang.org/docs/home.html
2. https://www.w3schools.com/kotlin
3. https://www.tutorialspoint.com/kotlin
4. https://developer.android.com/codelabs/basic-android-kotlin-compose-variables#4

Design Asset:

https://storyset.com

INFINITE LEARNING

Cheers 🍻

@infinitelearning_id

Infinite Learning Indonesia

infinitelearning_id