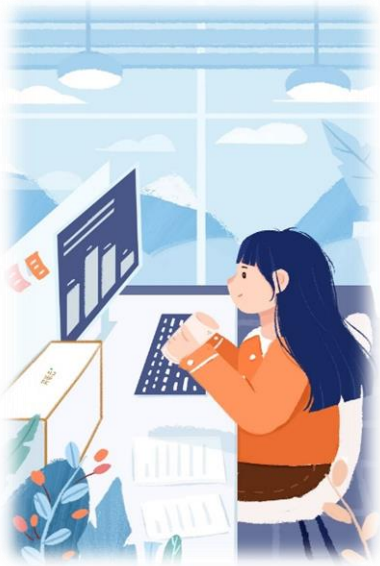


Let's Learn about,

Kotlin . . .

By~ Ruzal Kathiriya

What is Kotlin ?

- Kotlin is a statically typed, general-purpose programming language developed by **JetBrains,** that has built world-class IDEs like IntelliJ IDEA, PhpStorm, Appcode, etc.
- It was first introduced by JetBrains in 2011 and is a new language for the JVM.
- Kotlin is an object-oriented language, and a “better language” than Java, but still be fully interoperable with Java code.
- Kotlin is sponsored by **Google**, announced as one of the official languages for **Android** Development in 2017.
- Kotlin offers object-oriented and functional features to developers. In contrast, Java only offers object-oriented programming.

Features Kotlin ?

- **Conciseness :** Kotlin is a concise language, meaning that you can write less code to achieve the same results as in other languages.
- **Null safety**: Kotlin is a null-safe language, meaning that it helps you to avoid NullPointerExceptions. This is done through features such as non-nullable types, nullable types with default values, and the Elvis operator.
- **Expressiveness**: Kotlin is an expressive language, meaning that it allows you to write code that is easy to read and understand.
- **Interoperability with Java**:Kotlin is fully interoperable with Java, meaning that you can use Kotlin code in Java projects and vice versa.
- **Community**: Kotlin has a large and active community, which provides support and resources for Kotlin developers.

Datatype in Kotlin ?

- A data type is a collection of data values that specifies the type of value a variable has and the types of operations that can be performed on it without causing an error. Data types can also represent values as machine types.
- Data types are divided into different groups like as below..

  (Numbers [Integer, Float], Booleans, Characters, Strings, Arrays)

1) **<u>Numbers Datatype :</u>**

   (int => These data types contain integer values.)

   (float => These data type used to store decimal value or fractional part.)

Data Type	Bits	Min Value	Max Value
byte	8 bits	-128	127
short	16 bits	-32768	32767
int	32 bits	-2147483648	2147483647
long	64 bits	-9223372036854775808	9223372036854775807
float	32 bits	1.40129846432481707e-45	3.40282346638528860e+38
double	64 bits	4.94065645841246544e-324	1.79769313486231570e+308

2) **Boolean Data type :**

Boolean data type represents only one bit of information either **true or false**. The Boolean type in Kotlin is the same as in Java. These operations disjunction (||) or conjunction (&&) can be performed on boolean types.

Data Type	Bits	Data Range
boolean	1 bit	true or false

3) **Character Data Type :**

Character data type represents the small letters(a-z), Capital letters(A-Z), digits(0-9) and other symbols.

Data Type	Bits	Min Value	Max Value
char	16 bits	'\u0000' (0)	'\uFFFF' (65535)

4) **String Data Type :**

Strings are used for storing text. A string contains a collection of characters surrounded by double quotes.

Variables in Kotlin ?

- every variable should be declared before it's used. Without declaring a variable, an attempt to use the variable gives a syntax error.
- Declaration of the variable type also decides the kind of data you are allowed to store in the memory location. In case of local variables, the type of variable can be inferred from the initialized value.

Example of declare variables :

```
var rollno = 5
var name = "sita"

println(rollno)
println(name)
```

- variables are declared using two types –
  1) Immutable using val keyword
  2) Mutable using var keyword

✓ **Immutable Variables:**

Immutable is also called read-only variables. Hence, we can not change the value of the variable declared using *val* keyword.

```
val name = “Ram”
 name = “shyam”  //error was generated on compile time

//error – kotlin val cannot be reassigned
```

✓ **Mutable Variables:**

Mutable variable we can change the value of the variable.

```
val age = 19
age= 20              //successfully compile and run

println("My new Age is ${Age}")
```

Control statments in Kotlin ?

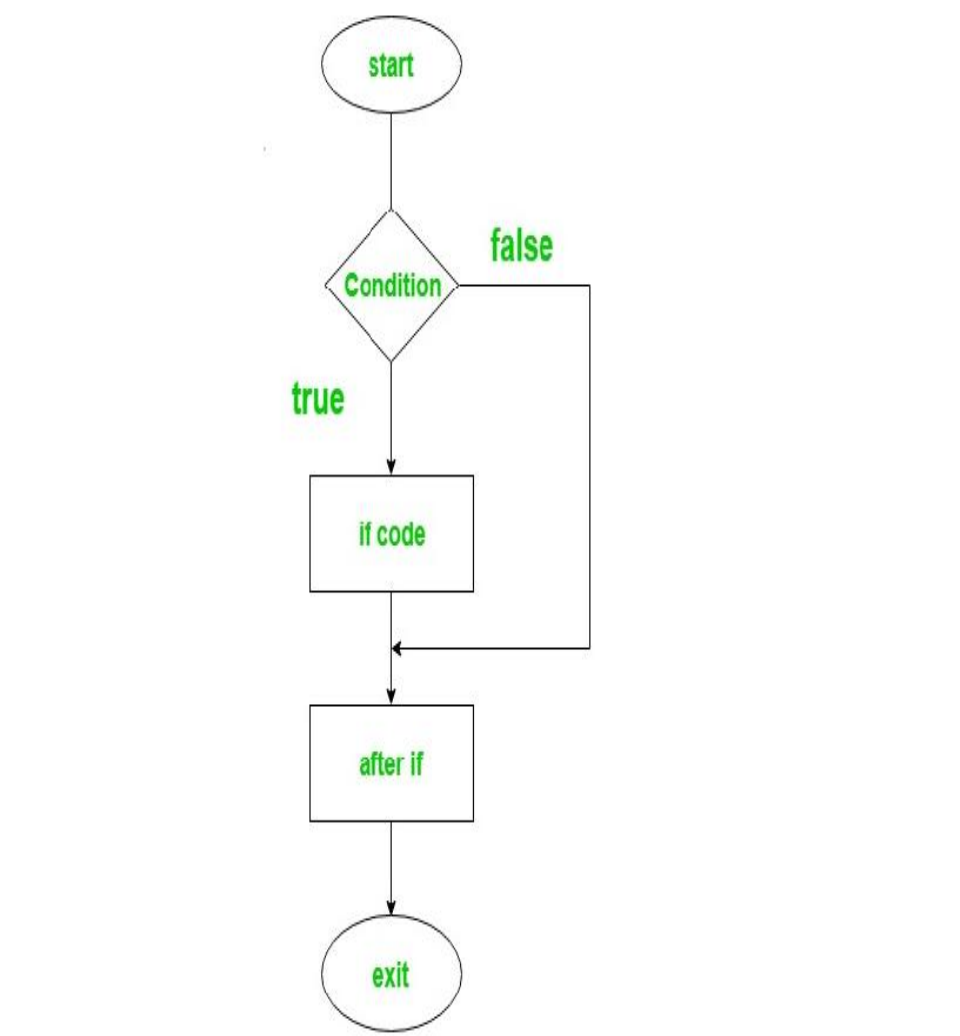
- A programming language uses control statements to control the flow of execution of a program based on certain conditions. If the condition is true then it enters into the conditional block and executes the instructions.
- There are different types of if-else expressions in Kotlin:
    1. if expression
    2. if-else expression
    3. if-else-if ladder expression
    4. nested if expression

1.) <u>if statement:</u>

It is used to specify whether a block of statements will be executed or not, i.e. if a condition is true, then only the statement or block of statements will be executed otherwise it fails to execute.

```
if(condition)
{
   // code to run if condition is true
}
```

- Flow chart of if statements:



- Example of if statements:

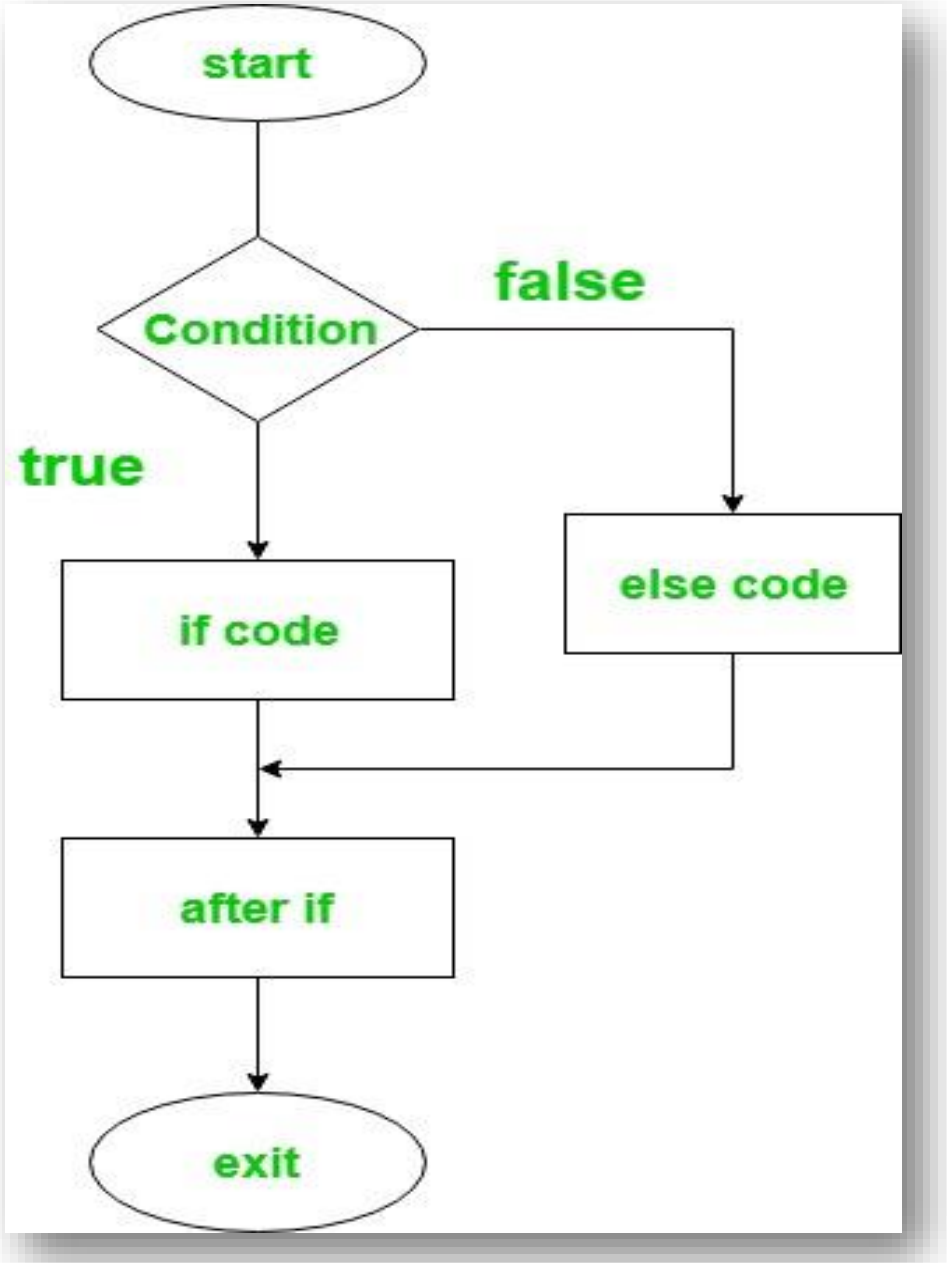
```
fun main()
{
    var a  = 3
     if(a > 0)
    {
          print("This number is positive.")
     }
}
```

2.) if-else statement:

if-else statement contains two blocks of statements. ‘if’ statement is used to execute the block of code when the condition becomes true and ‘else’ statement is used to execute a block of code when the condition becomes false.

```
if(condition)
{
    // code to run if condition is true
}
else
{
    // code to run if condition is false
}
```

- Flow chart of if-else statements:



- Example of if statements:

```
fun main()
 {
      var a  = 3
      var b = 10
       if(a > b)
      {
            print("a is grater than b.")
       }
        else
       {
            print("b is grater than a.")
        }
 }
```

Looping in Kotlin ?

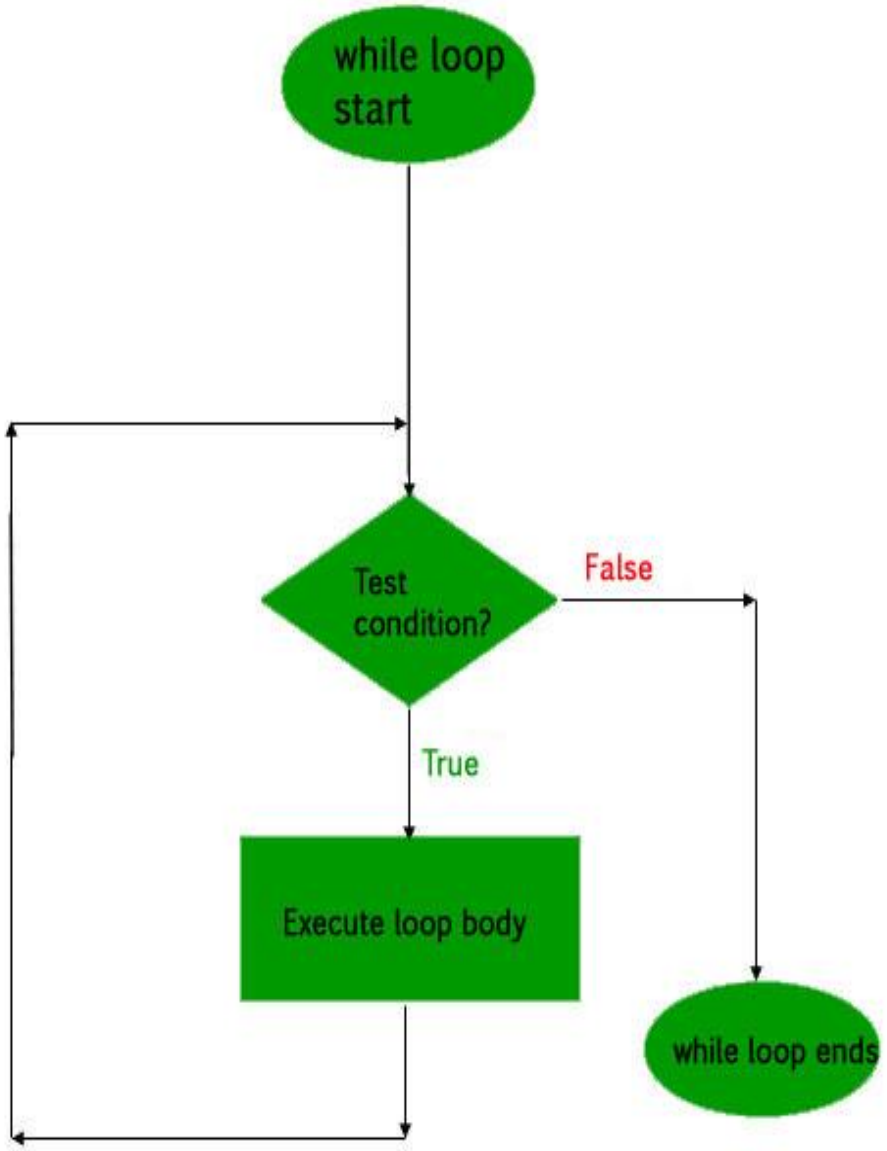
- A loop is a control flow statement that allows repeated execution of a block of code for a certain number of times or until a certain condition.
- There are different types of looping in Kotlin:
    1. While loop
    2. Do-while loop
    3. For loop

1.) <u>While loop:</u>

The while loop is used to execute a block of code while a condition is true. The while loop will execute the code block as long as the condition is true.

```
while (condition)
{
    // code block
}
```

- Flow chart of while loop:



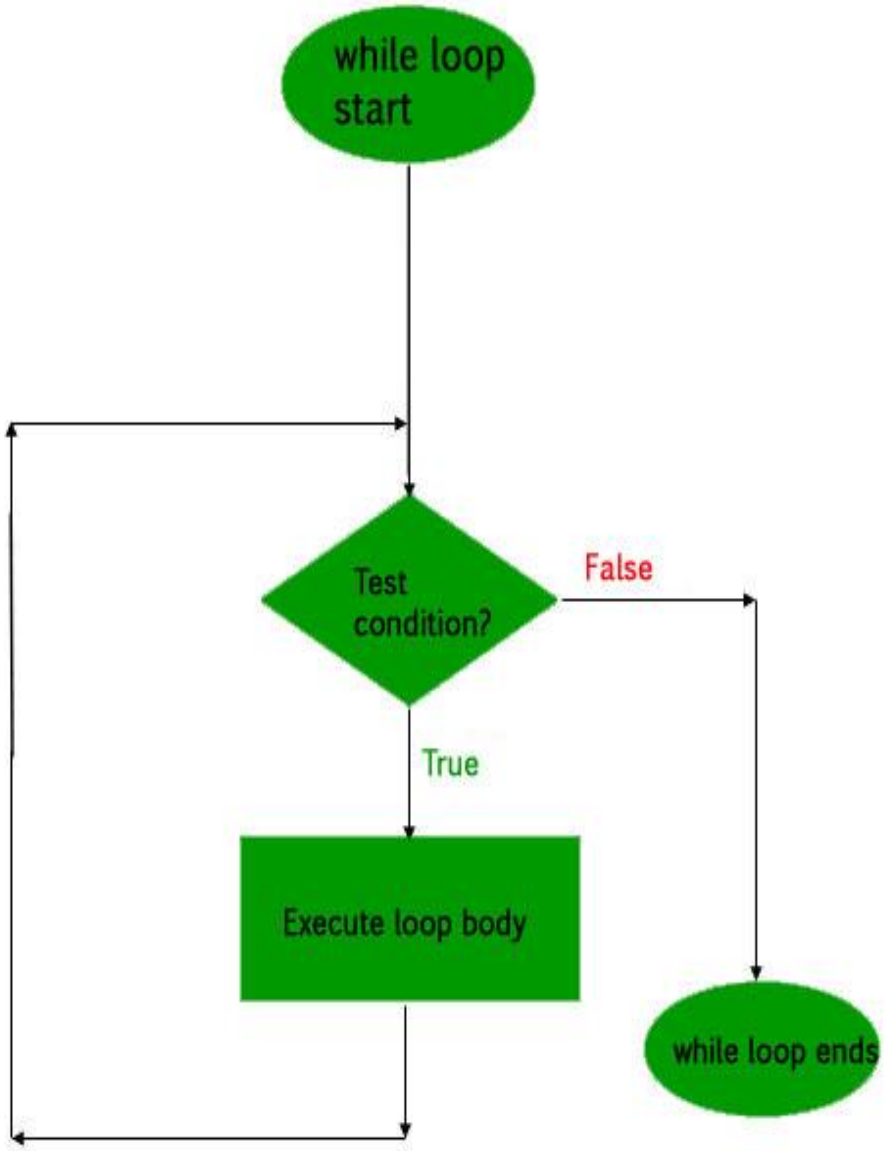
- Example of while loop:

(1.) using integer

```
fun main(args: Array<String>)
 {
     var num = 1

     while(num <= 10)
       {
           println(num)
           num++
       }
}
```

- Flow chart of while loop:



- Example of while loop:

(2.) using string array

```
fun main(args: Array<String>)
 {
     var name = arrayof (“ram”, “shyam”,
                         “mira”, “geeta”, “sita”)
     var i = 0
     while(i < name.size)
       {
           println(name[i])
           i++
       }
}
```

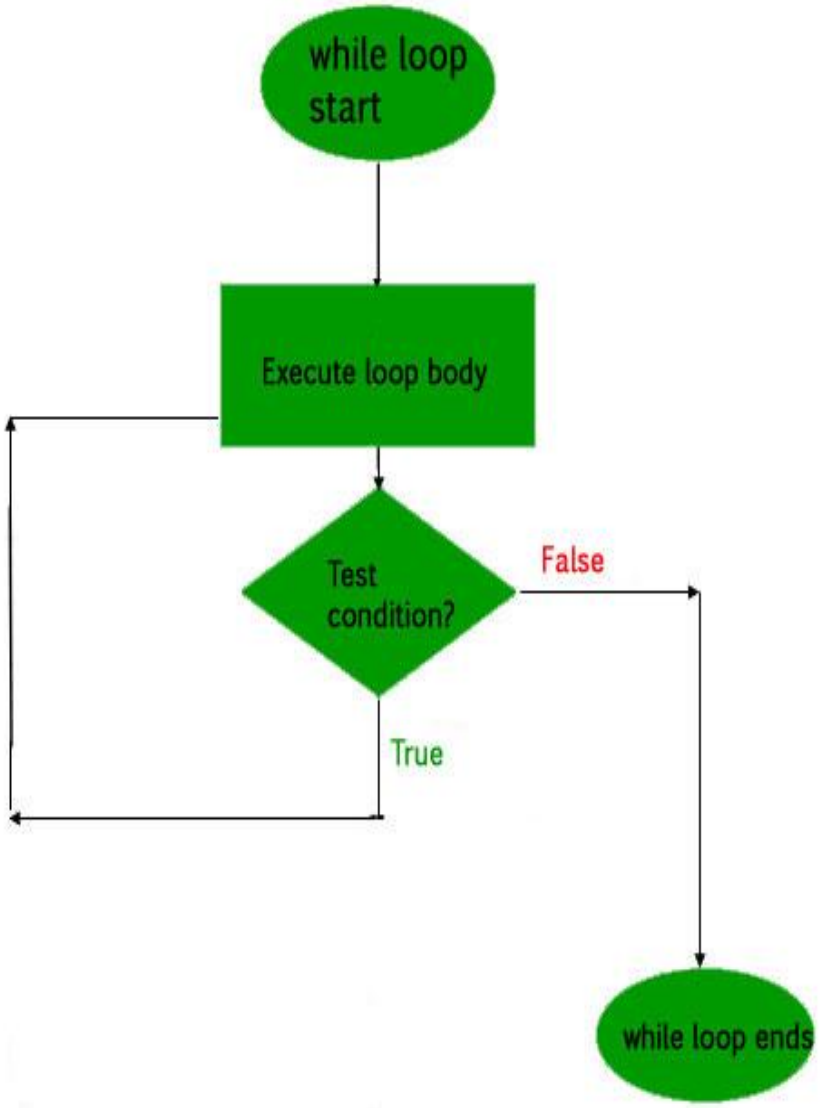
2.) Do-While loop:

Do-while loop is a control flow statement which executes a block of code at least once without checking the condition, and then repeatedly executes the block, or not, it totally depends upon a Boolean condition at the end of do-while block.

```
do
{
        // code to run
} while(condition)
```

- It contrast with the *while* loop because while loop executes the block only when condition becomes true but *do-while* loop executes the code first and then the expression or test condition is evaluated.
- ***do-while* loop working –** First of the all the statements within the block is executed, and then the condition is evaluated. If the condition is true the block of code is executed again.
- The process of execution of code block repeated as long as the expression evaluates to true. If the expression becomes false, the loop terminates and transfers control to the statement next to do-while loop.
- It is also known as **post-test loop** because it checks the condition after the block is executed.

- Flow chart of do-while loop:



- Example of do-while loop:

```
fun main(args: Array<String>)
 {
      var num = 12
      var i = 1

      do
        {
             println("$num * $i = " +num * i)
             i++
        }while(i <= 10)
}
```

3.) for loop:

for loop is used to *iterate* a part of program several times. It iterates through arrays, ranges, collections, or anything that provides for iterate. Kotlin for loop is equivalent to the **foreach** loop in languages like C#.

```
for(item in collection)
{
   // code to execute
}
```

- The forEach loop in Kotlin is a function that allows you to iterate over a collection, such as a list, set, or map. It takes a lambda function as an argument, which is executed for each element in the collection. The forEach loop is a concise and expressive way to iterate over collections, and it is often used in Kotlin code.
- for loop is used to iterate through the following because all of them provides iterator.
  - ✓ Range
  - ✓ Array
  - ✓ String
  - ✓ Collection

Class & Funcation in Kotlin ?

- classes and objects are used to represent objects in the real world. A class is a blueprint for creating objects (a particular data structure), providing initial values for state (member variables or fields), and implementations of behavior (member functions or methods).
- An object is an instance of a class and has its own state and behavior. You can create multiple objects from the same class, each with its own unique state.
- class is a blueprint for objects having similar properties. We need to define a class before creating an object and the **class** keyword is used to define a class. The class declaration consists of the class name, class header, and class body enclosed with curly braces.

Syntax of class :

```
class className { // class header
        // property
        // member function
}
```

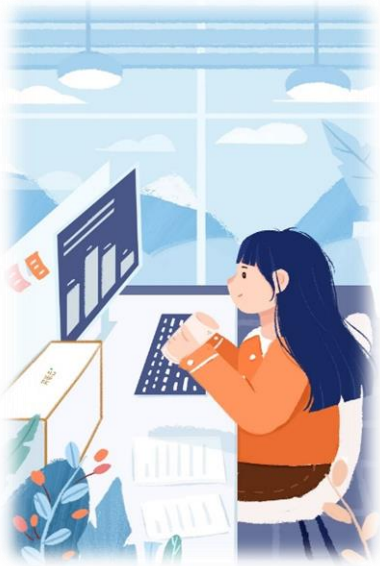
Syntax of funcation :

```
fun funName { // fun header

        //funcation data
}
```

❖ **Example of Class and funcation:**

```
class Car {
    var make : string = “”
    var modal : string = “”
    var year : Int = 0
     fun getInfo() : string {
         return “$make $modal, year $year”
     }
}
fun main() {
    val c = car()
    c.make = “Toyota”
    c.modal = “camry”
    c.year = 2020
    println(c.getInfo())
}
```



Let's Learn about,

ANDROID STUDIO

By~ Ruzal Kathiriya