

## Unit-III

# Dart Programming Language



## What is Dart?

Dart is a general-purpose, high-level modern programming language which is originally developed by Google. It is the new programming language which is emerged in 2011, but its stable version was released in June 2017. Dart is not so popular at that time, but It gains popularity when it is used by the Flutter.

Dart is a dynamic, class-based, object-oriented programming language with closure and lexical scope. Syntactically, it is quite similar to Java, C, and JavaScript. If you know any of these programming languages, you can easily learn the Dart programming language.

Dart is an open-source programming language which is widely used to develop the mobile application, modern web-applications, desktop application, and the Internet of Things (IoT) using by Flutter framework. It also supports a few advance concepts such as interfaces, mixins, abstract classes, refield generics, and type interface. It is a compiled language and supports two types of compilation techniques.

- **AOT (Ahead of Time)** - It converts the Dart code in the optimized JavaScript code with the help of the dar2js compiler and runs on all modern web-browser. It compiles the code at build time.
- **JIT (Just-In-Time)** - It converts the byte code in the machine code (native code), but only code that is necessary.

# History

Dart was revealed for the first time in the GOTO conference in the month of 10<sup>th</sup> - 12<sup>th</sup> October 2011 at Aarhus, Denmark. It is initially designed by the **Lars bark and Kesper** and developed by Google.

The first version 1.0 of Dart was released on November 14th, 2013, intended as a replacement of JavaScript.

In July 2014, the first edition of Dart language was approved by Ecma International approved at its 107<sup>th</sup> General Assembly.

The first version was criticized due to a malfunction on the web and this plan was dropped in 2015 with the 1.9 release of Dart.

The second version of Dart 2.0 was released in August, including a sound type system.

The recent version Dart 2.7 is supplemented with the extension method, which enables us to add any type of functionality.

## Why Dart?

We define the characteristics of Dart in the following point.

- Dart is a platform-independent language and supports all operating systems such as Windows, Mac, Linux, etc.
- It is an open-source language, which means it available free for everyone.
- It is an object-oriented programming language and supports all features of oops such as inheritance, interfaces, and optional type features.
- Dart is very useful in building real-time applications because of its stability.
- Dart with the dar2js comes compiler which transmits the Dart code into JavaScript code that runs on all modern web browser.
- The stand-alone Dart VM permits Dart code to run in a command-line interface environment.

## Key Points to Remember

Before learning the Dart, we should keep these concepts in mind. These concepts are given below.

- Everything in Dart is treated as an object including, numbers, Boolean, function, etc. like Python. All objects inherit from the Object class.
- Dart tools can report two types of problems while coding, warnings and errors. Warnings are the indication that your code may have some problem, but it doesn't interrupt the code's execution, whereas error can prevent the execution of code.
- Dart supports generic types, like **List<int>** (a list of integers) or **List<dynamic>** (a list of objects of any type).

## Dart Features

The Dart is an object-oriented, open-source programming language which contains many useful features. It is the new programming language and supports an extensive range of programming utilities such as interface, collections, classes, dynamic and optional typing. It is developed for the server as well as the browser. Below is the list of the important Dart features.



## Open Source

Dart is an open-source programming language, which means it is freely available. It is developed by Google, approved by the ECMA( **European Computer Manufacturer's Association**) standard, and comes with a BSD(Berkeley Source Distribution) license.

## Platform Independent

Dart supports all primary operating systems such as Windows, Linux, Macintosh, etc. The Dart has its own Virtual Machine which known as Dart VM, that allows us to run the Dart code in every operating system.

## Object-Oriented

Dart is an object-oriented programming language and supports all oops concepts such as classes, inheritance, interfaces and optional typing features. It also supports advance concepts like mixin, abstract, classes, reified generic, and robust type system.

## Concurrency

Dart is an asynchronous programming language, which means it supports multithreading using Isolates. The isolates are the independent entities that are related to threads but don't share memory and establish the communication between the processes by the message passing. The message should be serialized to make effective communication. The serialization of the message is done by using a snapshot that is generated by the given object and then transmits to another isolate for desterilizing.

## Extensive Libraries

Dart consists of many useful inbuilt libraries including SDK (Software Development Kit), core

As we discussed in the previous section, learning the Dart is not the Hercules task as we know that Dart's syntax is similar to Java, C#, JavaScript, kotlin, etc. if you know any of these languages then you can learn easily the Dart.

## Flexible Compilation

Dart provides the flexibility to compile the code and fast as well. It supports two types of compilation processes, AOT (Ahead of Time) and JIT (Just-in-Time). The Dart code is transmitted in the other language that can run in the modern web-browsers.

## Type Safe

The Dart is the type safe language, which means it uses both static type checking and runtime checks to confirm that a variable's value always matches the variable's static type, sometimes it known as the sound typing.

Although *types* are required, type *annotations* are optional because of type inference. This makes it code more readable. The other advantage to being type-safe language is, when we change the part of code, the system warns us about that modification that we have modified earlier.

## Objects

The Dart treats everything as an object. The value which assigns to the variable is an object. The functions, numbers, and strings are also an object in Dart. All objects inherit from Object class.

## Browser Support

The Dart supports all modern web-browser. It comes with the dart2js compiler that converts the Dart code into optimized JavaScript code that is suitable for all type of web-browser.

## Community

Dart has a large community across the world. So if you face problem while coding then it is easy to find help. The dedicated developers' team is working towards enhancing its functionality.

Here we have discussed essential features of the Dart language. We will more concepts of Dart language in upcoming tutorials.

# Dart Installation

To learn the Dart, we need to set up the Dart programming environment to our local machine. We are describing the following instructions to install the Dart SDK (Software Development Kit) in various operating systems. If you have already installed it, then you can skip this part.

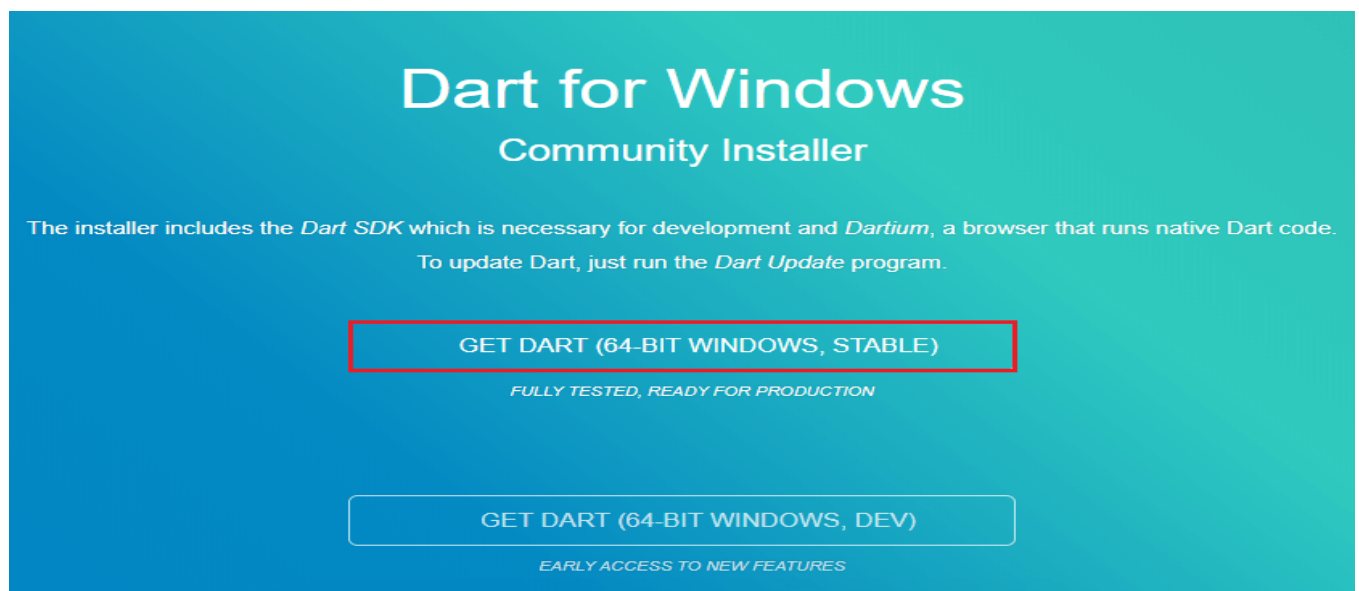
## Install the Dart SDK on Windows

Follow the below instructions to install Dart SDK in Windows.

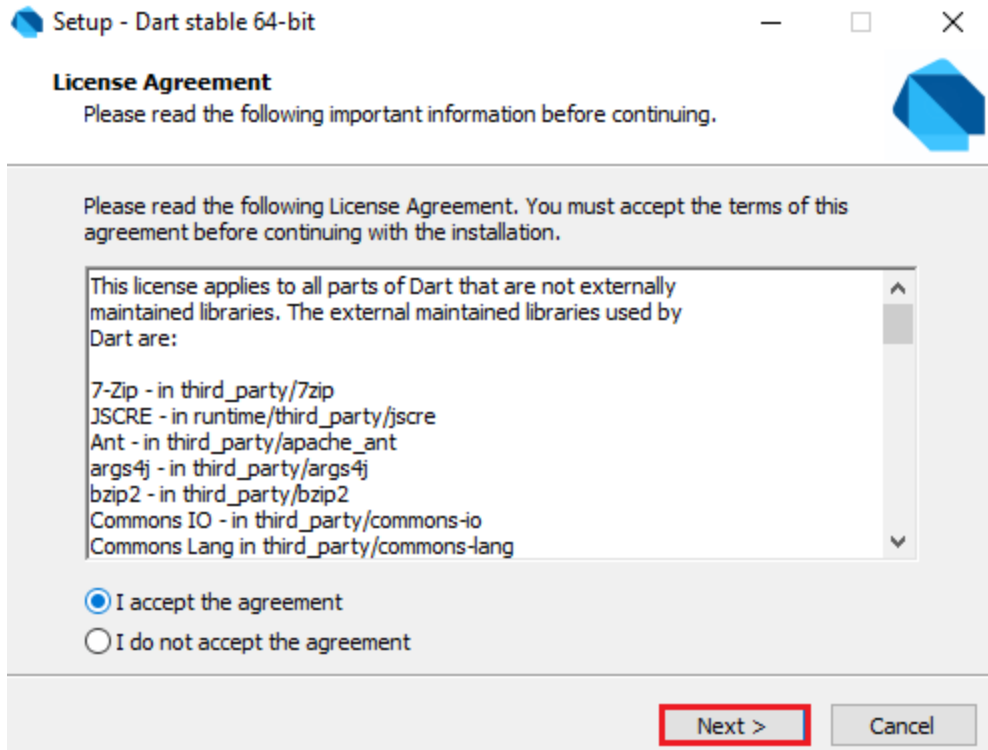
**Step -1:** Go to the browser and type the following link to download the SDK.

<http://www.gekorm.com/dart-windows/>

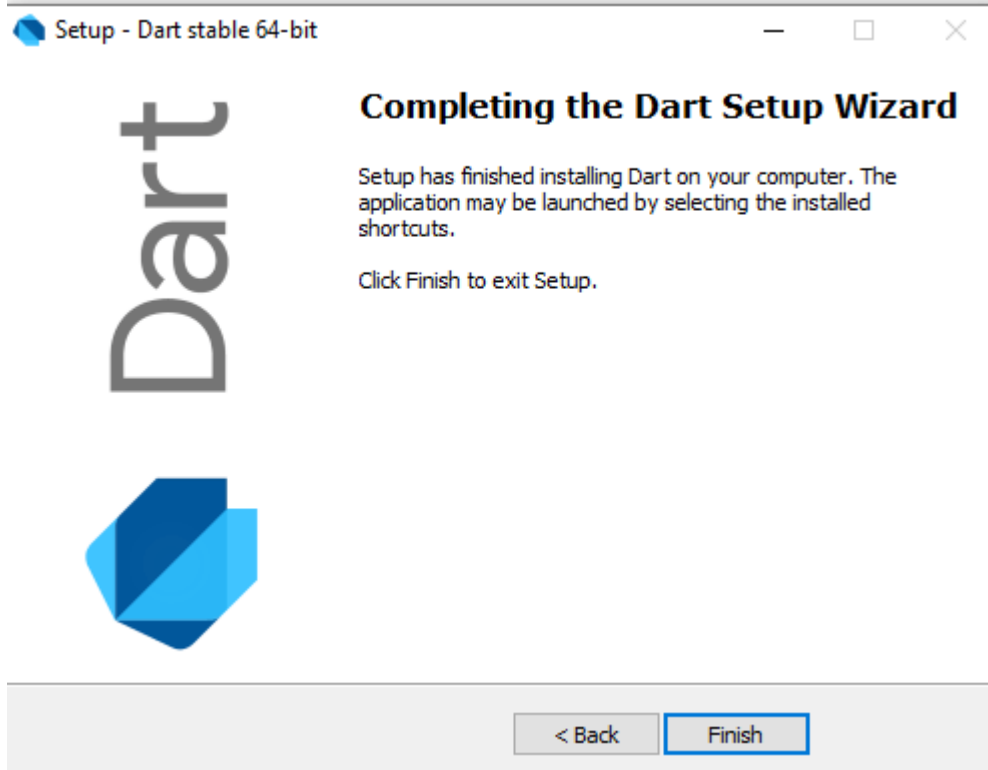
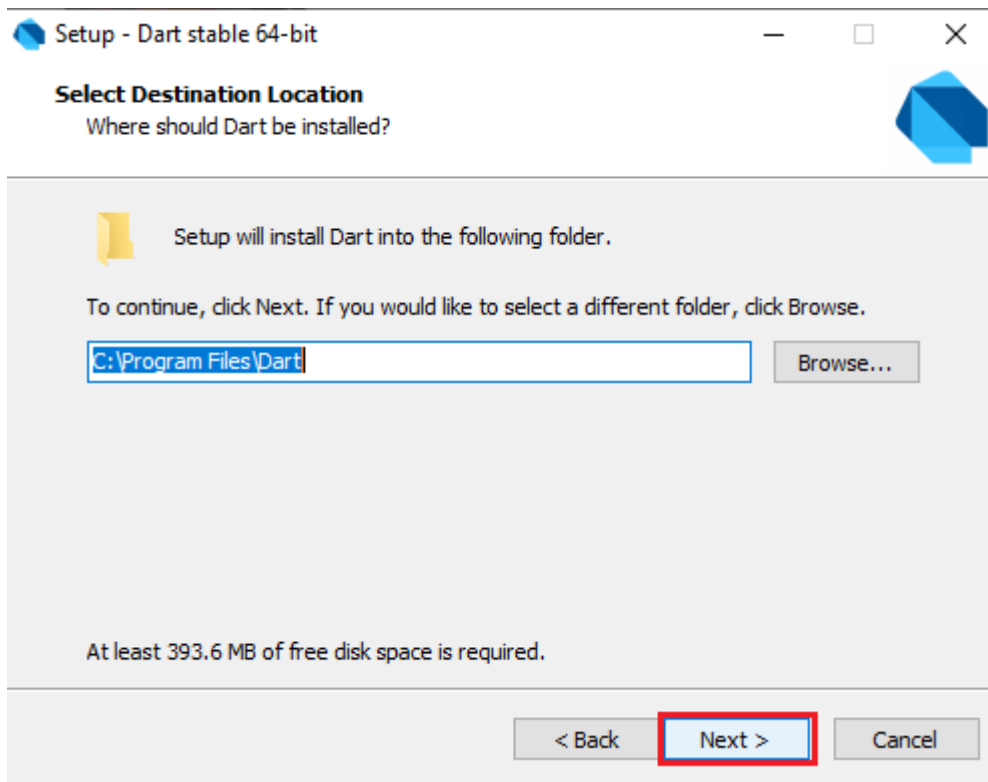
It will open the given page. Click on the following link.



**Step - 2:** Run the Dart installer(It is the .exe file that we downloaded in the previous step) and click on the **Next** button.

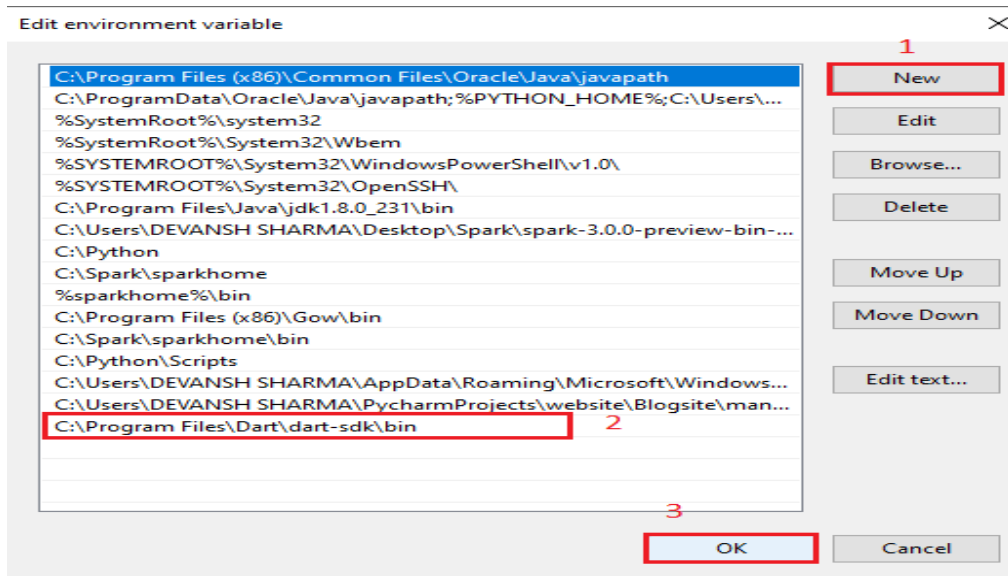


**Step - 3:** It provides the option to select the Dart installation path. After the path is selected, click on the **Next** button.



**Step - 4:** After the download is completed, set the PATH environment variable to "C:\Program Files\Dart\dart-sdk\bin" in advance system properties.





**Step - 5:** Now open the terminal and verify the Dart installation by typing dart.

```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows [Version 10.0.18362.778]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\DEVANSH SHARMA>dart
Usage: dart [<vm-flags>] <dart-script-file> [<script-arguments>]

Executes the Dart script <dart-script-file> with the given list of <script-arguments>.

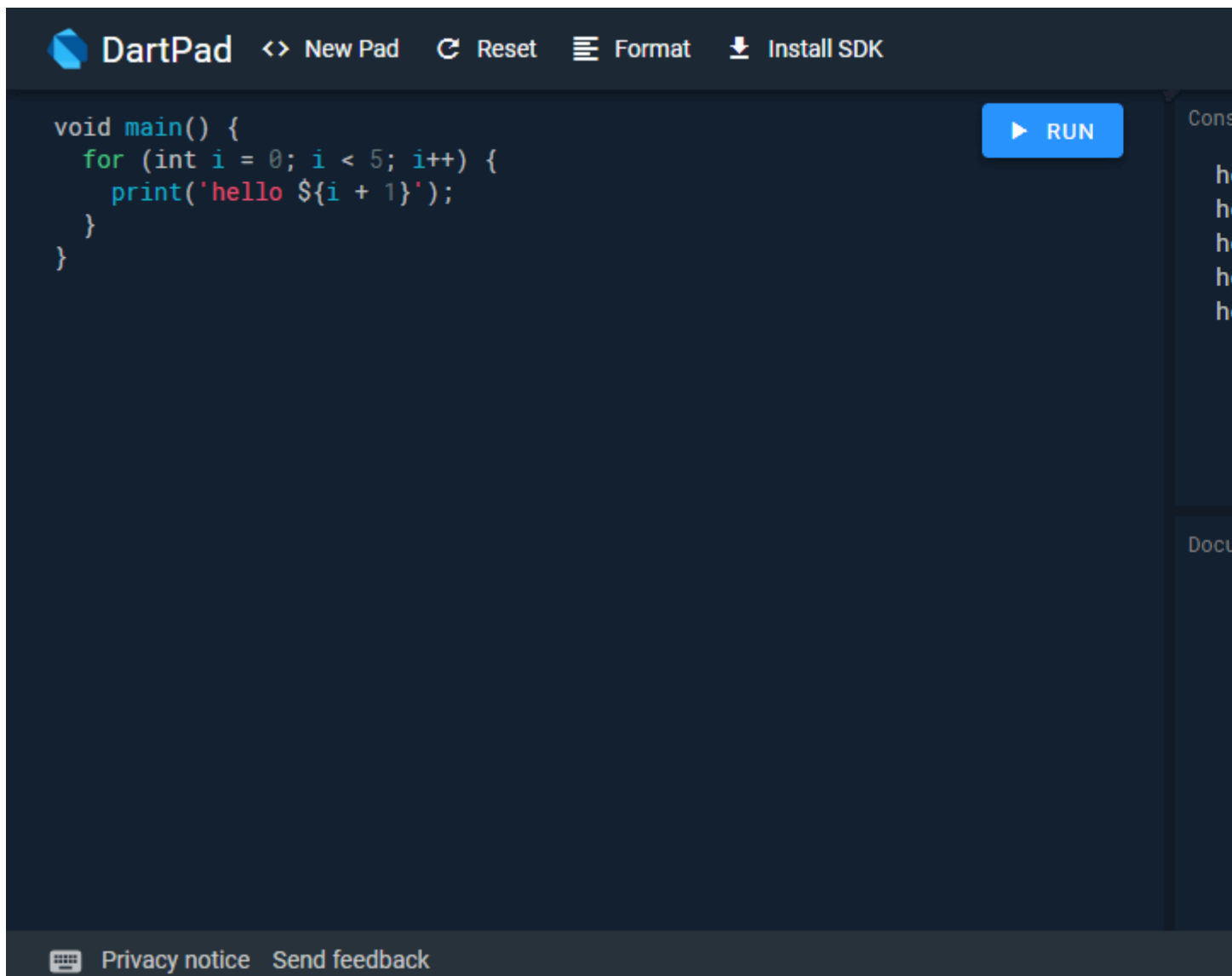
Common VM flags:
--enable-asserts
  Enable assert statements.
--help or -h
  Display this message (add -v or --verbose for information about
  all VM options).
--package-root=<path> or -p<path>
  Where to find packages, that is, "package:..." imports.
--packages=<path>
  Where to find a package spec file.
--observe[=<port>[/<bind-address>]]
  The observe flag is a convenience flag used to run a program with a
  set of options which are often useful for debugging under Observatory.
  These options are currently:
    --enable-vm-service[=<port>[/<bind-address>]]
    --pause-isolates-on-exit
    --pause-isolates-on-unhandled-exceptions
    --warn-on-pause-with-no-debugger
  This set is subject to change.
  Please see these options (--help --verbose) for further documentation.
--write-service-info=<file_name>
  Outputs information necessary to connect to the VM service to the
  specified file in JSON format. Useful for clients which are unable to
  listen to stdout for the Observatory listening message.
--snapshot-kind=<snapshot_kind>
--snapshot=<file_name>
  These snapshot options are used to generate a snapshot of the loaded
  Dart script:
    <snapshot-kind> controls the kind of snapshot, it could be
    kernel(default) or app-jit
    <file_name> specifies the file into which the snapshot is written
--version
  Print the VM version.

C:\Users\DEVANSH SHARMA>
```

If it is successfully installed then it looks like the above image.

## Online Dart Editor

We have discussed Dart installation on the various operating systems so far, but if we do not want to install Dart then there is an online Dart editor (Known as DartPad) is available to run the Dart programs. The online DartPad is provided at <https://dartpad.dev/>. The DartPad offers to execute the dart scripts and display [HTML](#) and also console output. The online DartPad looks like the below image.



## The dart2js Tool

The Dart SDK comes with the dart2js tool, which transmits the Dart code into runnable JavaScript code. It is necessary because few web browsers do not support the Dart VM.

Use the following command in the terminal to compile the Dart code into JavaScript code.

1. `dart2js - - out = <output_file>.js <dart_script>.dart`

The above command will create a file that contains the JavaScript code corresponding to the Dart code.

## Dart First Program

As we have discussed earlier, Dart is easy to learn if you know any of Java, C++, JavaScript, etc. The simplest "**Hello World**" program gives the idea of the basic syntax of the programming language. It is the way of testing the system and working environment. In this tutorial, we will give the basic idea of Dart's syntax. There are several ways to run the first program, which is given below:

- Using Command Line
- Running on Browser
- Using IDE

Before running the first program, it must ensure that we have installed the Dart SDK properly. We have discussed the complete installation guide in our previous tutorial. Let's run our first program.

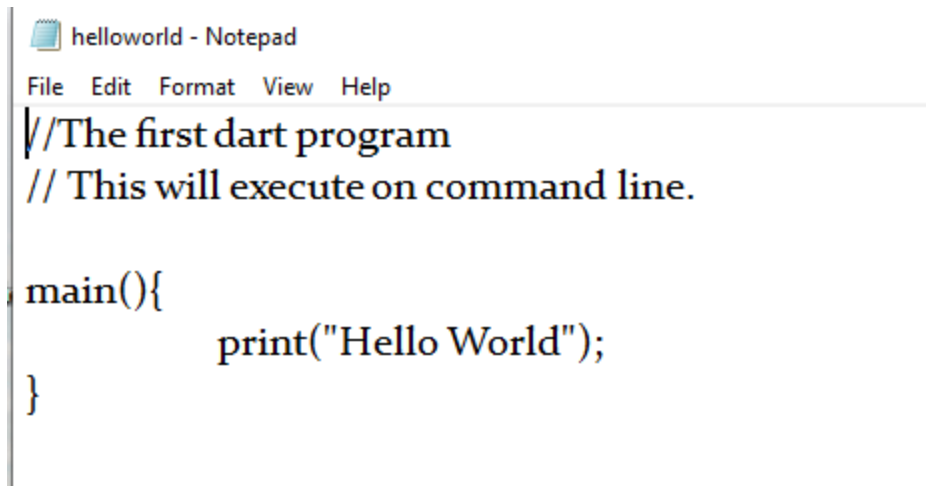
## Using Command Line

### Step - 1:

Type `dart` on the terminal if it is showing dart runtime then Dart is successfully installed.

### Step - 2:

Open a text editor and create a file called "helloworld.dart". The file extension should be **.dart** that specifies; it is a Dart program file.



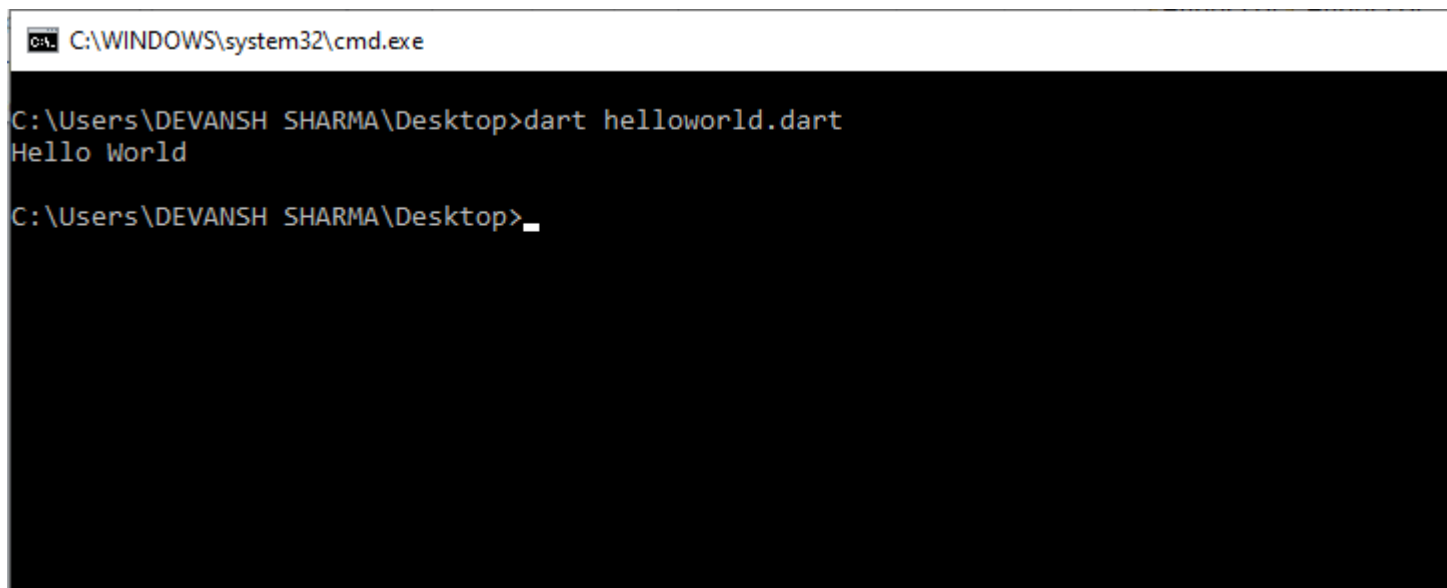
```
helloworld - Notepad
File Edit Format View Help
//The first dart program
// This will execute on command line.

main(){
    print("Hello World");
}
```

- **main()** - The main() function indicates that it is the beginning of our program. It is an essential function where the program starts its execution.
- **print()** - This function is used to display the output on the console. It is similar to C, JavaScript, or any other language. The curly brackets and semicolons are necessary to use appropriately.

### Step - 3:

Open the command line; compile the program run the Dart program by typing **dart helloworld.dart**. It will show **Hello World** on the screen.



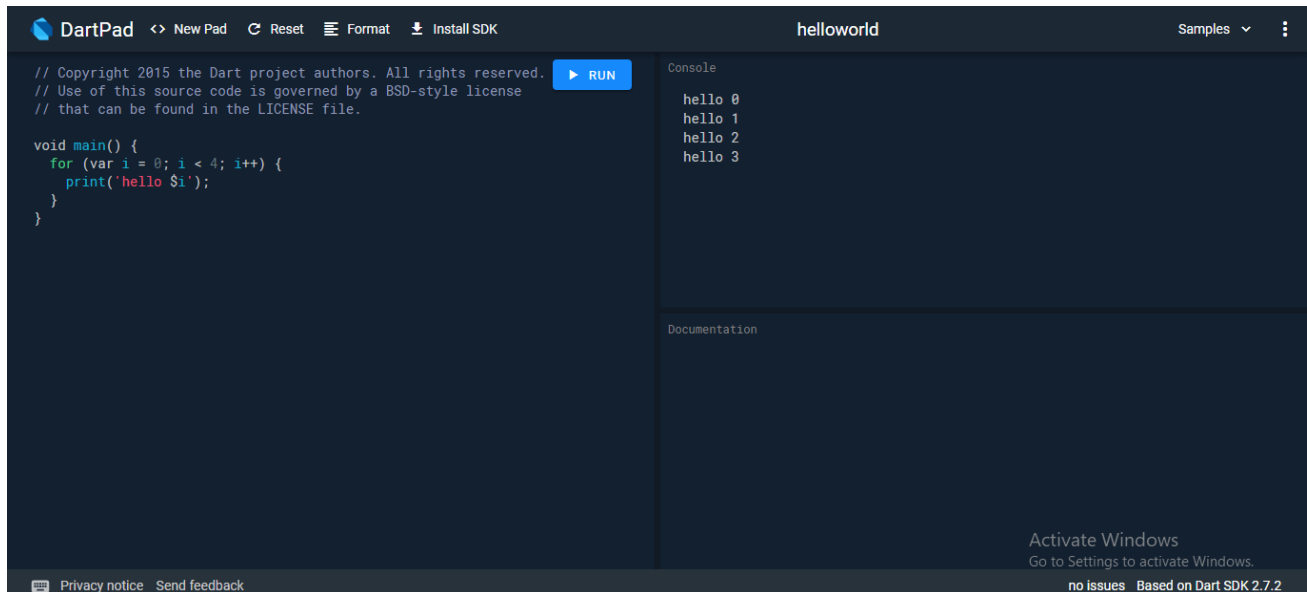
```
C:\WINDOWS\system32\cmd.exe

C:\Users\DEVANSH SHARMA\Desktop>dart helloworld.dart
Hello World

C:\Users\DEVANSH SHARMA\Desktop>_
```

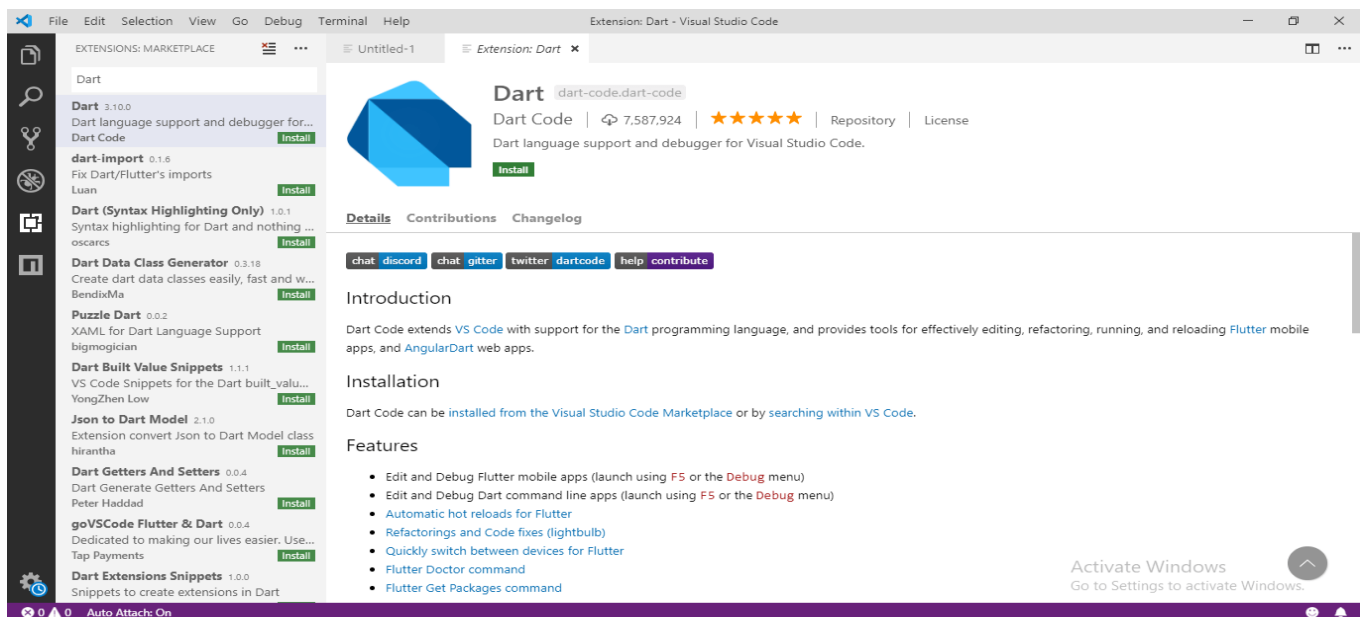
## Running on Browser

Dart provides an online editor which is known as DartPad available at <https://dartpad.dartlang.org/>. On the left side, we can write the code and the output displays on the right side of the screen. We can add **HTML** and **CSS** with the Dart Code. Dart also provides some sample programs to learn. Let's have a look at the following image.



## Using IDE

There are various IDEs that support Dart, such as Visual Studio Code, WebStorm, IntelliJ, etc. For the visual studio code, download the dart extension and run the code.



We recommend running the Dart code using the command line or IDE.

## Dart Basic Syntax

In this tutorial, we will learn the basic syntax of the Dart programming language.

### Dart Identifiers

Identifiers are the name which is used to define variables, methods, class, and function, etc. An Identifier is a sequence of the letters([A to Z],[a to z]), digits([0-9]) and underscore(\_), but remember that the first character should not be a numeric. There are a few rules to define identifiers which are given below.

- The first character should not be a digit.
- Special characters are not allowed except underscore (\_) or a dollar sign (\$).
- Two successive underscores (\_\_) are not allowed.
- The first character must be alphabet(uppercase or lowercase) or underscore.
- Identifiers must be unique and cannot contain whitespace.
- They are case sensitive. The variable name **Joseph** and **joseph** will be treated differently.

Below is the table of valid and invalid identifiers.

Valid Identifiers	Invalid Identifiers
firstname	__firstname
firstName	first name
var1	5Var
\$count	first-name
_firstname	1result

First_name	@var
------------	------

## Dart Printing and String Interpolation

The **print()** function is used to print output on the console, and **\$expression** is used for the string interpolation. Below is an example.

### Example -

```
1. void main()
2. {
3.   var name = "Peter";
4.   var roll_no = 24;
5.   print("My name is ${name} My roll number is ${roll_no}");
6. }
```

### Output:

My name is Peter My roll number is 24

## Semicolon in Dart

The semicolon is used to terminate the statement that means, it indicates the statement is ended here. It is mandatory that each statement should be terminated with a semicolon (;). We can write multiple statements in a single line by using a semicolon as a delimiter. The compiler will generate an error if it is not use properly.

### Example -

```
1. var msg1 = "Hello World!";
2. var msg2 = "How are you?"
```

## Dart Whitespace and Line Breaks

The Dart compiler ignores whitespaces. It is used to specify space, tabs, and newline characters in our program. It separates one part of any statement from another part of the statement. We can also use space and tabs in our program to define indentation and provide the proper format for the program. It makes code easy to understand and readable.

## Block in Dart

The block is the collection of the statement enclosed in the curly braces. In Dart, we use curly braces to group all of the statements in the block. Consider the following syntax.

### Syntax:

1. `{//start of the block`
2. `//block of statement(s)`
3. `}// end of the block`

## Dart Command-Line Options

The Dart command-line options are used to modify Dart script execution. The standard command-line options are given below.

Sr.	Command-line Options	Descriptions
1.	-c or -c	It allows both assertion and type checks.
2.	--version	It shows VM version information.
3.	--package<path>	It indicates the path to the package resolution configuration file.
4.	-p <path>	It indicates where to find the libraries.
5.	-h or -help	It is used to seek help.

## Enable Checked Mode

Generally, the Dart program runs in two modes which are given below.

- Checked Mode
- Production Mode



**Checked Mode** - The checked mode enables various checks in the Dart code, such as type-checking. It warns or throws errors while developing processes. To start the checked mode, type -c or --checked option in the command prompt before the dart script-file name. By default, the Dart VM runs in the checked mode.

**Production Mode** - The Dart script runs in the production mode. It provides assurance of performance enhancement while running the script. Consider the following example.

Example -

1. **void** main() {
2.   **int** var = "hello";
3.   print(var);
4. }

Now activate the checked mode by typing dart -c or --checked mode

1. dart -c mode.dart

The Dart VM will through the following error.

1. Unhandled exception:
2. type 'String' is not a subtype of type 'int' of 'n' where
3.   String is from dart:core
4.   **int** is from dart:core

## Dart Comments

Comments are the set of statements that are ignored by the Dart compiler during the program execution. It is used to enhance the readability of the source code. Generally, comments give a brief idea of code that what is happening in the code. We can describe the working of variables, functions, classes, or any statement that exists in the code. Programmers should use the comment for better practice. There are three types of comments in the Dart.

## Types of Comments

Dart provides three kinds of comments

- Single-line Comments

- Multi-line Comments
- Documentation Comments

## Single-line Comment

We can apply comments on a single line by using the // (double-slash). The single-line comments can be applied until a line break.

### Example -

```
1. void main(){
2.     // This will print the given statement on screen
3.     print("Welcome to JavaTpoint");
4. }
```

### Output:

Welcome to JavaTpoint

The // (double-slash) statement is completely ignored by the Dart compiler and returned the output.

## Multi-line Comment

Sometimes we need to apply comments on multiple lines; then, it can be done by using /\*.....\*/. The compiler ignores anything that written inside the /\*...\*/, but it cannot be nested with the multi-line comments. Let's see the following example.

### Example -

```
1. void main(){
2.     /* This is the example of multi-line comment
3.     This will print the given statement on screen
4.     */
5.
6.     print("Welcome to JavaTpoint");
7. }
```

### Output:

## Dart Documentation Comment

The document comments are used to generate documentation or reference for a project/software package. It can be a single-line or multi-line comment that starts with `///` or `/*`. We can use `///` on consecutive lines, which is the same as the multiline comment. These lines ignore by the Dart compiler expect those which are written inside the curly brackets. We can define classes, functions, parameters, and variables. Consider the following example.

### Syntax

1. `///This`
2. `///is`
3. `///a example of`
4. `/// multiline comment`

### Example -

1. `void main(){`
2.  `///This is`
3.  `///the example of`
4.  `///multi-line comment`
5.  `///This will print the given statement on screen.`
6.  `print("Welcome to Java");`
7. `}`

### Output:

Welcome to Java

## Dart Keywords

Dart Keywords are the **reserve words** that have special meaning for the compiler. It cannot be used as the variable name, class name, or function name. Keywords are case sensitive; they must be written as they are defined. There are 61 keywords in the Dart. Some of them are common, you may be already familiar and few are different. Below is the list of the given Dart keywords.

abstract <sup>2</sup>	else	import <sup>2</sup>	super
as <sup>2</sup>	enum	in	switch
assert	export <sup>2</sup>	interface <sup>2</sup>	sync <sup>1</sup>
async <sup>1</sup>	extends	is	this
await <sup>3</sup>	extension <sup>2</sup>	library <sup>2</sup>	throw
break	external <sup>2</sup>	mixin <sup>2</sup>	true
case	factory	new	try
catch	false	null	typedef <sup>2</sup>
class	final	on1	var
const	finally	operator <sup>2</sup>	void
continue	for	part <sup>2</sup>	while
covariant <sup>2</sup>	Function <sup>2</sup>	rethrow	with
default	get <sup>2</sup>	return	yield <sup>3</sup>
deffered <sup>2</sup>	hide <sup>1</sup>	set <sup>2</sup>	
do	if	show <sup>1</sup>	
dynamic <sup>2</sup>	implements <sup>2</sup>	static <sup>2</sup>	

# Dart Data Types

The data types are the most important fundamental features of programming language. In Dart, the data type of the variable is defined by its value. The variables are used to store values and reserve the memory location. The data-type specifies what type of value will be stored by the variable. Each variable has its data-type. The Dart is a static type of language, which means that the variables cannot modify.

**Note - Dart is static typed and types annotations language. Dart can infer a type and types annotations are optional.**

Dart supports the following built-in Data types.

- Number
- Strings
- Boolean
- Lists
- Maps
- Runes
- Symbols

## Dart Number

The Dart's Number is used to store the numeric values. The number can be two types - integer and double.

- **Integer** - Integer values represent the whole number or non-fractional values. An integer data type represents the 64-bit non-decimal numbers between  $-2^{63}$  to  $2^{63}$ . A variable can store an unsigned or signed integer value. The example is given below -

1. `int marks = 80;`

- **Double** - Double value represents the 64-bit of information (double-precision) for floating number or number with the large decimal points. The double keyword is used to declare the double type variable.

1. `double pi = 3.14;`

## Dart Strings

A string is the sequence of the character. If we store the data like - name, address, special character, etc. It is signified by using either single quotes or double quotes. A Dart string is a sequence of UTF-16 code units.

1. `var msg = "Welcome to point";`

## Dart Boolean

The Boolean type represents the two values - true and false. The bool keyword uses to denote Boolean Type. The numeric values 1 and 0 cannot be used to represent the true or false value.

1. `bool isValid = true;`

## Dart Lists

In Dart, The list is a collection of the ordered objects (value). The concept of list is similar to an array. An array is defined as a collection of the multiple elements in a single variable. The elements in the list are separated by the comma enclosed in the square bracket[]. The sample list is given below.

1. `var list = [1,2,3];`

## Dart Maps

The maps type is used to store values in key-value pairs. Each key is associated with its value. The key and value can be any type. In Map, the key must be unique, but a value can occur multiple times. The Map is defined by using curly braces {}, and comma separates each pair.

1. `var student = {'name': 'Joseph', 'age':25, 'Branch': 'Computer Science'} ;`

## Dart Runes

As we know that, the strings are the sequence of Unicode UTF-16 code units. Unicode is a technique which is used to describe a unique numeric value for each digit, letter, and symbol. Since Dart Runes are the special string of Unicode UTF-32 units. It is used to represent the special syntax.

For example - The special heart character ♥ is equivalent to Unicode code \u2665, where \u means Unicode, and the numbers are hexadecimal integer. If the hex value is less or greater than 4 digits, it places in a curly bracket ({}). For example - An emoji 😊 is represented as \u{1f600}. The example is given below.

Example -

```
1. void main(){
2.   var heart_symbol = '\u2665';
3.   var laugh_symbol = '\u{1f600}';
4.   print(heart_symbol);
5.   print(laugh_symbol);
6. }
```

**Output:**

♥

😊

## Dart Symbols

The Dart Symbols are the objects which are used to refer an operator or identifier that declare in a Dart program. It is commonly used in APIs that refers to identifiers by name because an identifier name can changes but not identifier symbols.

## Dart Dynamic Type

Dart is an optionally typed language. If the variable type is not specified explicitly, then the variable type is dynamic. The dynamic keyword is used for type annotation explicitly.

## Dart Variable

Variable is used to store the value and refer the memory location in computer memory. When we create a variable, the Dart compiler allocates some space in memory. The size of the memory block of memory is depended upon the type of variable. To create a

variable, we should follow certain rules. Here is an example of a creating variable and assigning value to it.

1. `var name = 'Devansh';`

Here the variable called **name** that holds 'Devansh' string value. In **Dart**, the variables store references. The above variable stores reference to a String with a value of Devansh.

## Rule to Create Variable

Creating a variable with a proper name is an essential task in any programming language. The Dart has some rules to define a variable. These rules are given below.

- The variable cannot contain special characters such as whitespace, mathematical symbol, runes, Unicode character, and keywords.
- The first character of the variable should be an alphabet([A to Z],[a to z]). Digits are not allowed as the first character.
- Variables are case sensitive. For example, - variable age and AGE are treated differently.
- The special character such as #, @, ^, &, \* are not allowed except the underscore(\_) and the dollar sign(\$).
- The variable name should be readable to the program and readable.

## How to Declare Variable in Dart

We need to declare a variable before using it in a program. In Dart, The **var** keyword is used to declare a variable. The Dart compiler automatically knows the type of data based on the assigned to the variable because Dart is an infer type language. The syntax is given below.

Syntax -

1. `var <variable_name> = <value>;`  
or  
1. `var <variable_name>;`

**Example -**

1. `var name = 'Andrew'`

In the above example, the variable **name** has allocated some space in the memory. The semicolon(;) is necessary to use because it separates program statement to another.



## Type Annotations

As we had pointed out, The Dart is an infer language but it also provides a type annotation. While declaring the variable, it suggests the type of the value that variable can store. In the type annotation, we add the data type as a prefix before the variable's name that ensures that the variable can store specific data type. The syntax is given below.

### Syntax -

1. `<type> <variable_name>;`  
or
1. `<type> <name> = <expression>;`

### Example -

1. `int age;`
2. `String msg = "Welcome to College ";`  
In the above example, we have declared a variable named **age** which will store the integer data. The variable named **msg** stored the string type data.

## Declaring the variable with Multiple Values

Dart provides the facility to declare multiple values of the same type to the variables. We can do this in a single statement, and each value is separated by commas. The syntax is given below.

### Syntax -

1. `<type> <var1,var2....varN>;`

### Example -

1. `int i,j,k;`

## Default Value

While declaring the variable without initializing the value then the Dart compiler provides default value (Null) to the variable. Even the numeric type variables are initially assigned with the null value. Let's consider the following example.

1. `int count;`
2. `assert(count == null);`

## Final and const

When we do not want to change a variable in the future then we use final and const. It can be used in place of **var** or in addition to a type. A final variable can be set only one time where the variable is a compile-time constant. The example of creating a final variable is given below.

### Example -

1. `final name = 'Ricky';` // final variable without type annotation.
2. `final String msg = 'How are you?';` // final variable with type annotation.  
If we try to change these values then it will throw an error.
1. `name = 'Roger';` // Error: Final variable can't be changed.

The **const** is used to create compile-time constants. We can declare a value to compile-time constant such as number, string literal, a const variable, etc.

1. **const** a = 1000;

The const keyword is also used to create a constant value that cannot be changed after its creation.

1. var f = **const**[];

If we try to change it, then it will throw an error.

1. f = [12]; //Error, The const variable cannot be change

## Dart Operators

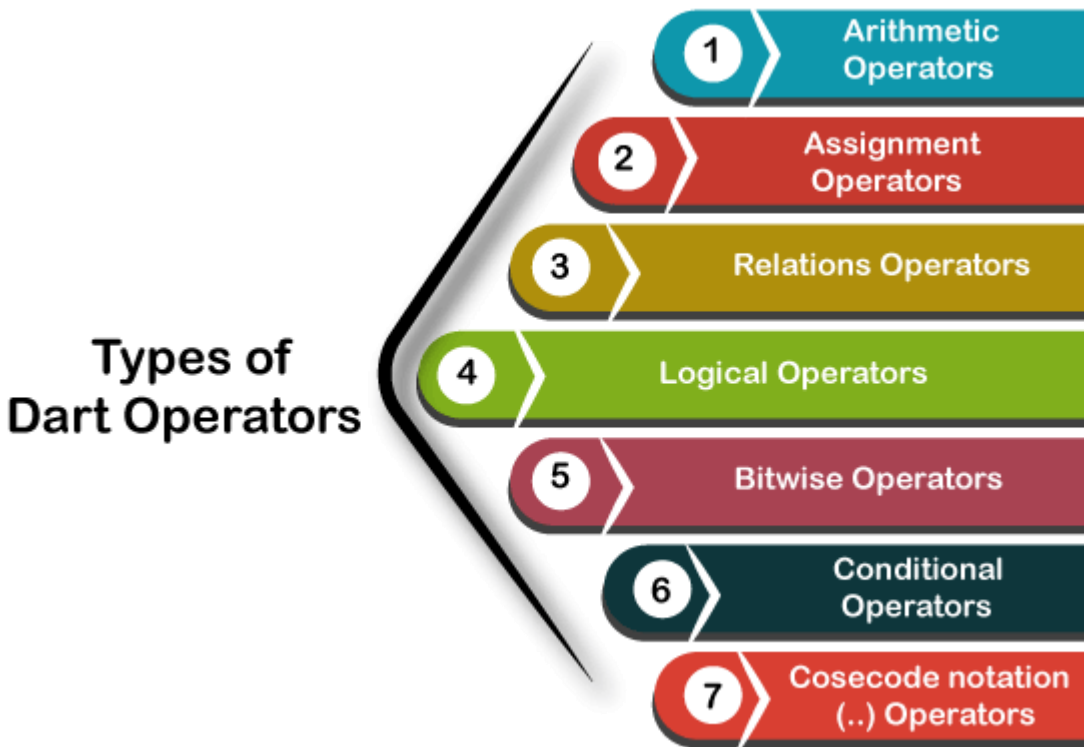
An operator is a symbol that is used to manipulating the values or performs operations on its operand. The given expression: 5+4, in this expression, 5 and 4 are operands and "+" is the operator.

Dart provides an extensive set of built-in operators to accomplish various types of operations. Operators can be unary or binary, which means unary take only on operand and binary take two operands with operators. There are several types of operators. Following is the list of Dart Operators.

## Types of Operators

Dart supports the following types of operators.

- Arithmetic Operators
- Assignment Operators
- Relational Operators
- Type test Operators
- Logical Operators
- Bitwise Operator
- Conditional Operators
- Casecade notation(..) Operators



## Dart Arithmetic Operators

Arithmetic Operators are the most common operators that are used to perform addition, subtraction, multiplication, divide, etc. Let's take variable a holds 20 and variable b hold 10, then -

Sr.	Operator Name	Description	Example
1.	Addition(+)	It adds the left operand to the right operand.	a+b will return 30
2.	Subtraction(-)	It subtracts the right operand from the left operand.	a-b will return 10
3	Divide(/)	It divides the first operand by the second operand and returns quotient.	a/b will return 2.0

4.	Multiplication(*)	It multiplies the one operand to another operand.	a*b will return 200
5.	Modulus(%)	It returns a remainder after dividing one operand to another.	a%b will return 0
6.	Division(~/)	It divides the first operand by the second operand and returns integer quotient.	a/b will return 2
7.	Unary Minus(-expr)	It is used with a single operand changes the sign of it.	-(a-b) will return -10

### Example -

```

1. void main(){
2.   print("Example of Assignment operators");
3.   var n1 = 10;
4.   var n2 = 5;
5.
6.   print("n1+n2 = ${n1+n2}");
7.   print("n1-n2 = ${n1-n2}");
8.   print("n1*n2 = ${n1*n2}");
9.   print("n1/n2 = ${n1/n2}");
10.  print("n1%n2 = ${n1%n2}");
11. }
```

### Output:

Example of Arithmetic operators

n1+n2 = 15

n1-n2 = 5

n1\*n2 = 50

n1/=n2 = 2

n1%n2 = 0

#### o Dart Unary Operators (post and pre)

In [Java](#), there are ++ and -- operators are known as increment and decrement operators and also known as unary operators, respectively. Unary operators, operate on single operand where ++ adds 1 to operands and -- subtract 1 to operand respectively.

The unary operators can be used in two ways - postfix and prefix. If ++ is used as a postfix (like x++), it returns the value of operand first then increments the value of x. If -- is used as a prefix (like ++x), it increases the value of x.

Sr.	Operator Name	Description
1.	++(Prefix)	It increment the value of operand.
2.	++(Postfix)	It returns the actual value of operand before increment.
3.	--(Prefix)	It decrement the value of the operand.
4.	--(Postfix)	It returns the actual value of operand before decrement.

Let's understand the following example -

#### Example -

```
1. void main() {  
2.   var x = 30;  
3.   print(x++);           //The postfix value  
4.   var y = 25;  
5.   print(++y);           //The prefix value,  
6.   var z = 10;  
7.   print(--z);           //The prefix value  
8.   var u = 12;  
9.   print(u--); }        //The postfix value
```

#### Output:

```
30  
26  
9  
12
```

## Assignment Operator

Assignment operators are used to assigning value to the variables. We can also use it combined with the arithmetic operators. The list of assignment operators is given below. Suppose a holds value 20 and b holds 10.

Operators	Name	Description
= (Assignment Operator)	It assigns the right expression to the left operand.	
+=(Add and Assign)	It adds right operand value to the left operand and resultant assign back to the left operand. For example - $a+=b \rightarrow a = a+b \rightarrow 30$	
-= (Subtract and Assign)	It subtracts right operand value from left operand and resultant assign back to the left operand. For example - $a-=b \rightarrow a = a-b \rightarrow 10$	
*=(Multiply and Assign)	It multiplies the operands and resultant assign back to the left operand. For example - $a*=b \rightarrow a = a*b \rightarrow 200$	
/=(Divide and Assign)	It divides the left operand value by the right operand and resultant assign back to the left operand. For example - $a%=b \rightarrow a = a\%b \rightarrow 2.0$	
~/=(Divide and Assign)	It divides the left operand value by the right operand and integer remainder quotient back to the left operand. For example - $a\%=b \rightarrow a = a\%b \rightarrow 2$	
%=(Mod and Assign)	It divides the left operand value by the right operand and remainder assign back to the left operand. For example - $a\%=b \rightarrow a = a\%b \rightarrow 0$	
<<=(Left shift AND assign)	The expression $a<<=3$ is equal to $a = a<<3$	
>>=(Right shift AND assign)	The expression $a>>=3$ is equal to $a = a>>3$	
&=(Bitwise AND assign)	The expression $a\&=3$ is equal to $a = a\&3$	

$\wedge$ =(Bitwise exclusive OR and assign)	The expression $a\wedge=3$ is equal to $a = a\wedge 3$	
$ $ =(Bitwise inclusive OR and assign)	The expression $a =3$ is equal to $a = a 3$	

Let's understand the following example -

### Example -

```

1. void main(){
2.   print("Example of Assignment operators");
3.   var n1 = 10;
4.   var n2 = 5;
5.   n1+=n2;
6.   print("n1+=n2 = ${n1}");
7.   n1-=n2;
8.   print("n1-=n2 = ${n1}");
9.   n1*=n2;
10.  print("n1*=n2 = ${n1}");
11.  n1/=n2;
12.  print("n1/=n2 = ${n1}");
13.  n1%=n2;
14.  print("n1%=n2 = ${n1}");
15. }
```

### Output:

```

Example of Assignment operators
n1+=n2 = 15
n1-=n2 = 10
n1*=n2 = 50
n1~/n2 = 10
n1%=n2 = 0
```

## Relational Operator

Relational operators or Comparison operators are used to making a comparison between two expressions and operands. The comparison of two expressions returns the Boolean true and false. Suppose a holds 20 and b hold 10 then consider the following table.

Sr.	Operator	Description
1.	>(greater than)	a>b will return TRUE.
2.	<(less than)	a<b will return FALSE.
3.	>=(greater than or equal to)	a>=b will return TRUE.
4.	<=(less than or equal to)	a<=b will return FALSE.
5.	==(is equal to)	a==b will return FALSE.
6.	!=(not equal to)	a!=b will return TRUE.

Let's understand the following example -

#### Example -

```
1. void main() {
2.   var a = 30;
3.   var b = 20;
4.   print("The example of Relational Operator");
5.   var res = a>b;
6.   print("a is greater than b: "+res.toString());
7.   var res0= a<b;
8.   print("a is less than b: "+res0.toString());
9.   var res1 = a>=b;
10.  print("a is greater than or equal to b: "+res1.toString());
11.  var res2 = a<=b;
12.  print("a is less than and equal to b: "+res2.toString());
13.  var res3 = a!= b;
14.  print("a is not equal to b: "+res3.toString());
```



```

15. var res4 = a==b;
16. print("a is equal to b: "+res4.toString());
17. }

```

### Output:

The example of Relational Operator

a is greater than b: true

a is less than b: false

a is greater than or equal to b: true

a is less than and equal to b: false

a is not equal to b: true

a is equal to b: false

## Dart Type Test Operators

The Type Test Operators are used to testing the types of expressions at runtime. Consider the following table.

Sr.	Operator	Description
1.	as	It is used for typecast.
2.	is	It returns TRUE if the object has specified type.
3.	is!	It returns TRUE if the object has not specified type.

Let's understand the following example.

```

1. void main()
2. {
3.   var num = 10;
4.   var name = "JavaTpoint";
5.   print(num is int);
6.   print(name is! String );
7. }

```

### Output:

true  
false

## Dart Logical Operators

The Logical Operators are used to evaluate the expressions and make the decision. Dart supports the following logical operators.

Sr.	Operator	Description
1.	&&(Logical AND)	It returns if all expressions are true.
2.	(Logical OR)	It returns TRUE if any expression is true.
3.	!(Logical NOT)	It returns the complement of expression.

Let's understand the following example.

```
1. void main(){  
2.   bool bool_val1 = true, bool_val2 = false;  
3.   print("Example of the logical operators");  
4.     var val1 = bool_val1 && bool_val2;  
5.   print(val1);  
6.     var val2 = bool_val1 || bool_val2;  
7.   print(val2);  
8.     var val3 = !(bool_val1 || bool_val2);  
9.   print(val3);  
10. }
```

### Output:

```
Example of the logical operators  
false  
true  
false
```

## Dart Bitwise Operators

The Bitwise operators perform operation bit by bit on the value of the two operands. Following is the table of bitwise operators.

Let's understand the following example.

1. If  $a = 7$
2.  $b = 6$
3. then  $\text{binary}(a) = 0111$
4.  $\text{binary}(b) = 0011$
5. Hence  $a \& b = 0011$ ,  $a|b = 0111$  and  $a^b = 0100$

Sr.	Operators	Description
1.	&(Binary AND)	It returns 1 if both bits are 1.
2.	(Binary OR)	It returns 1 if any of bit is 1.
3.	^(Binary XOR)	It returns 1 if both bits are different.
4.	~(Ones Compliment)	It returns the reverse of the bit. If bit is 0 then the compliment will be 1.
5.	<<(Shift left)	The value of left operand moves left by the number of bits present in the right operand.
6.	>>(Shift right)	The value of right operand moves left by the number of bits present in the left operand.

Let's understand the following example -

#### Example -

1. `void main(){`
2. `print("Example of Bitwise operators");`
3.
4. `var a = 25;`
5. `var b = 20;`
6. `var c = 0;`
7. `}`

```

8. // Bitwise AND Operator
9. print("a & b = ${a&b}");
10.
11. // Bitwise OR Operator
12. print("a | b = ${a|b}");
13.
14. // Bitwise XOR
15. print("a ^ b = ${a^b}");
16.
17. // Complement Operator
18. print("~a = ${(~a)}");
19.
20. // Binary left shift Operator
21. c = a <<2;
22. print("c<<1= ${c}");
23.
24. // Binary right shift Operator
25. c = a >>2;
26. print("c>>1= ${c}");
27. }

```

### Output:

```

Example of Bitwise operators
a & b = 16
a | b = 29
a ^ b = 13
~a = 4294967270
c<<1= 100
c>>1= 6

```

## Dart Conditional Operators (?:)

The Conditional Operator is same as if-else statement and provides similar functionality as conditional statement. It is the second form of **if-else statement**. It is also identified as "**Ternary Operator**". The syntax is given below.

### Syntax 1 -

1. condition ? exp1 : exp2

If the given condition is TRUE then it returns exp1 otherwise exp2.

### Syntax 2 -

1. exp1 ?? expr2

If the exp1 is not-null, returns its value, otherwise returns the exp2's value.

Let's understand the following example.

### Example - 1

1. **void** main() {
2.   var x = **null**;
3.   var y = **20**;
4.   var val = x ?? y;
5.   print(val);
6. }

### Output:

20

Let's have a look at another scenario.

### Example -2

1. **void** main() {
2.   var a = **30**;
3.   var output = a > **42** ? "value greater than 10": "value lesser than equal to 30";
4.   print(output);
5. }

### Output:

value lesser than or equal to 30

## Dart Cascade notation Operators

The Cascade notation Operators (..) is used to evaluate a series of operation on the same object. It is an identical as the method chaining that avoids several of steps, and we don't need store results in temporary variables.

## Dart Control Flow Statement

The **control statements** or **flow of control statements** are used to control the flow of Dart program. These statements are very important in any programming languages to decide whether other statement will be executed or not. The code statement generally runs in the sequential manner. We may require executing or skipping some group of statements based on the given condition, jumps to another statement, or repeat the execution of the statements.

In Dart, control statement allows to smooth flow of the program. By using the control flow statements, a Dart program can be altered, redirected, or repeated based on the application logic.

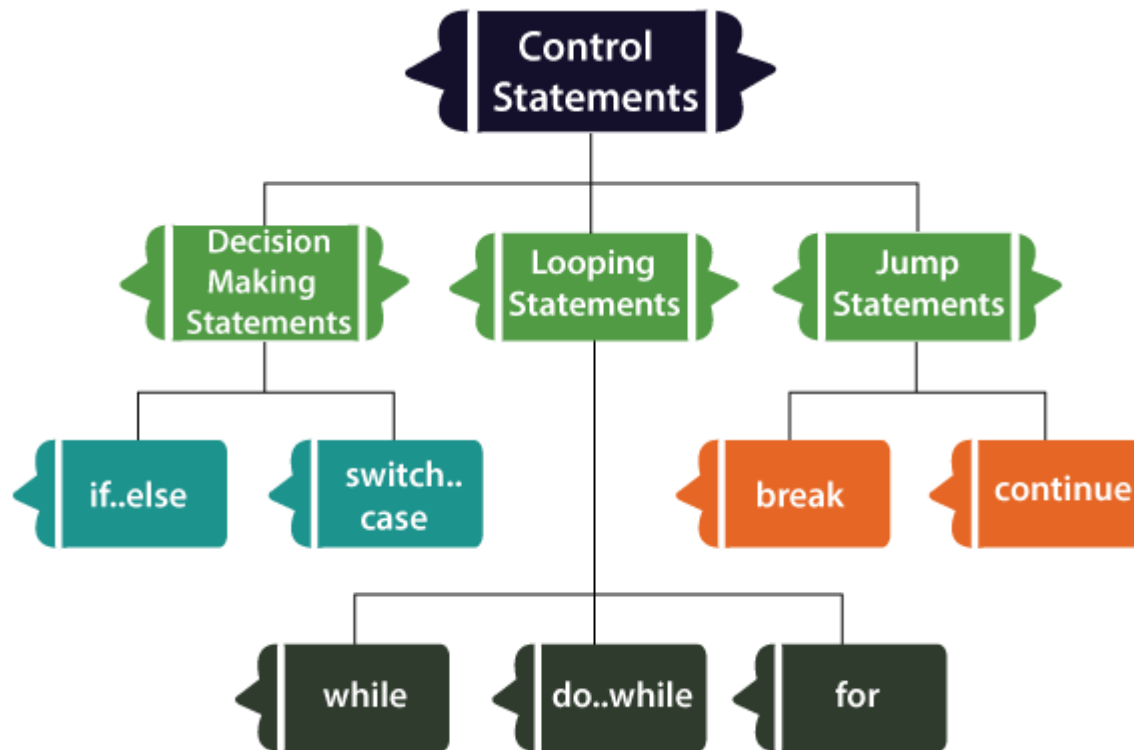
## Categories of Flow Statement

In Dart, Control flow statement can be categorized mainly in three following ways.

- Decision-making statements
- Looping statements
- Jump statements

## Dart Decision-Making Statements

The Decision-making statements allow us to determine which statement to execute based on the test expression at runtime. Decision-making statements are also known as the Selection statements. In Dart program, single or multiple test expression (or condition) can be existed, which evaluates Boolean TRUE and FALSE. These results of the expression/condition helps to decide which block of statement (s) will execute if the given condition is TRUE or FALSE.



Dart provides following types of Decision-making statement.

- If Statement
- If-else Statements
- If else if Statement
- Switch Case Statement

## Dart Looping Statements

Dart looping statements are used to execute the block of code multiple-times for the given number of time until it matches the given condition. These statements are also called Iteration statement.

Dart provides following types of the looping statements.

- Dart for loop
- Dart for....in loop
- Dart while loop
- Dart do while loop

## Dart Jump Statements

Jump statements are used to jump from another statement, or we can say that it transfers the execution to another statement from the current statement.

Dart provides following types of jump statements -

- Dart Break Statement
- Dart Continue Statement

The above jump statements behave differently.

## Dart if Statements

If statement allows us to a block of code execute when the given condition returns true. In Dart programming, we have a scenario where we want to execute a block of code when it satisfies the given condition. The condition evaluates Boolean values TRUE or FALSE and the decision is made based on these Boolean values.

## Dart If Statement Flow Diagram

The syntax of if statement is given below.

### Syntax -

1. If (condition) {
2.     //statement(s)
3. }

The given condition is if statement will evaluate either TRUE or FALSE, if it evaluates true then statement inside if body is executed, if it evaluates false then statement outside if block is executed.

Let's understand the following example.

### Example - 1

1. **void** main () {
2.     // define a variable which hold numeric value



```

3. var n = 35;
4.
5. // if statement check the given condition
6. if (n<40){
7.     print("The number is smaller than 40")
8. };
9. }

```

### Output:

The number is smaller than 40

### Explanation -

In the above program, we declared an integer variable n. We specified the condition in if statement. Is the given number is smaller than 40 or not? The if statement evaluated the true, it executed the if body and printed the result.

### Example - 2

```

1. void main () {
2.     // define a variable which holds a numeric value
3.     var age = 16;
4.     // if statement check the given condition
5.     if (age>18){
6.         print("You are eligible for voting");
7.     }
8.     else{
9.         print("You are not eligible for voting"); }
10. }

```

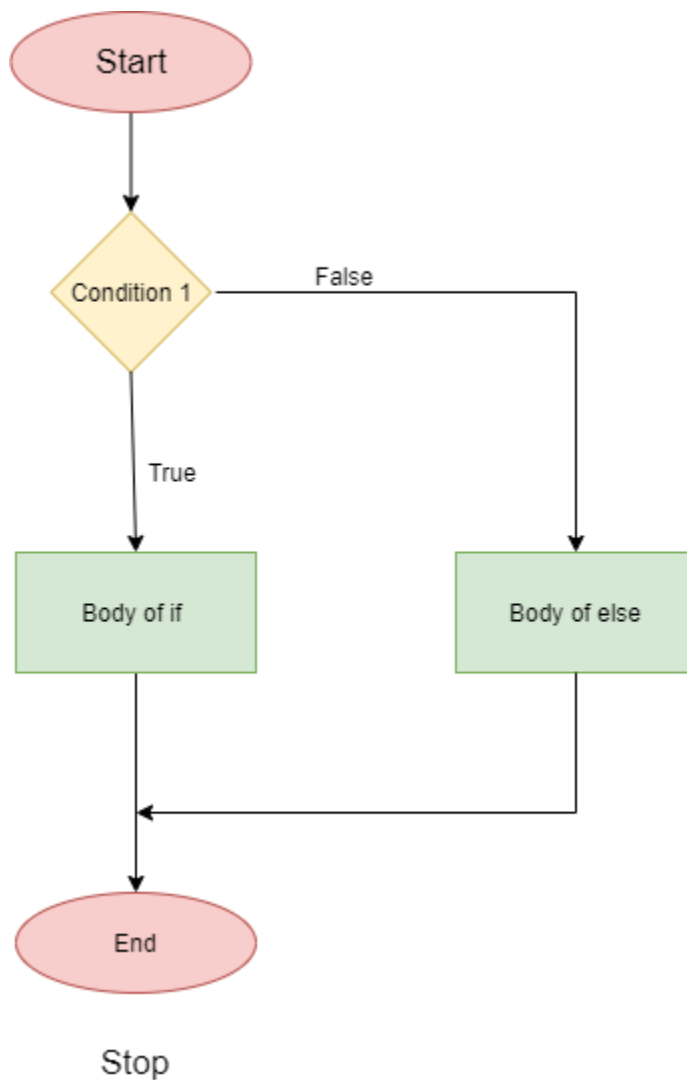
### Output:

You are not eligible for voting

In the above program, we can see that the if condition evaluated the false then execution skipped the if body and executed the outside statement of if block.

In Dart, if-block is executed when the given condition is true. If the given condition is false, else-block is executed. The else block is associated with the if-block.

## Dart if...else Statement Flow Diagram



### Syntax:

1. **if**(condition) {
2.     // statement(s);
3. } **else** {
4.     // statement(s);
5. }

Here, if -else statement is used for either types of result TRUE or False. If the given condition evaluates true, then if body is executed and if the given condition evaluates false; then, the else body is executed.

Let's understand the following example.

### Example -

```
1. void main() {
2.     var x = 20;
3.     var y = 30;
4.     print("if-else statement example");
5.
6.     if(x > y){
7.         print("x is greater than y");
8.     } else {
9.         print("y is greater than x");
10.
11. };
12.
13. }
```

### Output:

```
if-else statement example
y is greater than x
```

### Explanation -

In the above code, we have two variables which stored integer value. The given condition evaluated false then it printed the else-block.

### Example -2 Write a program to find the given number is even or odd.

```
1. void main() {
2.     var num = 20;
3.
4.     print("if-else statement example");
5.
6.     if(num%2 == 0){
7.         print("The given number is even");
```

```
8. } else {  
9.     print("The given number is odd");  
10.};  
11.}
```

### Output:

```
If-else statement example  
The given number is even
```

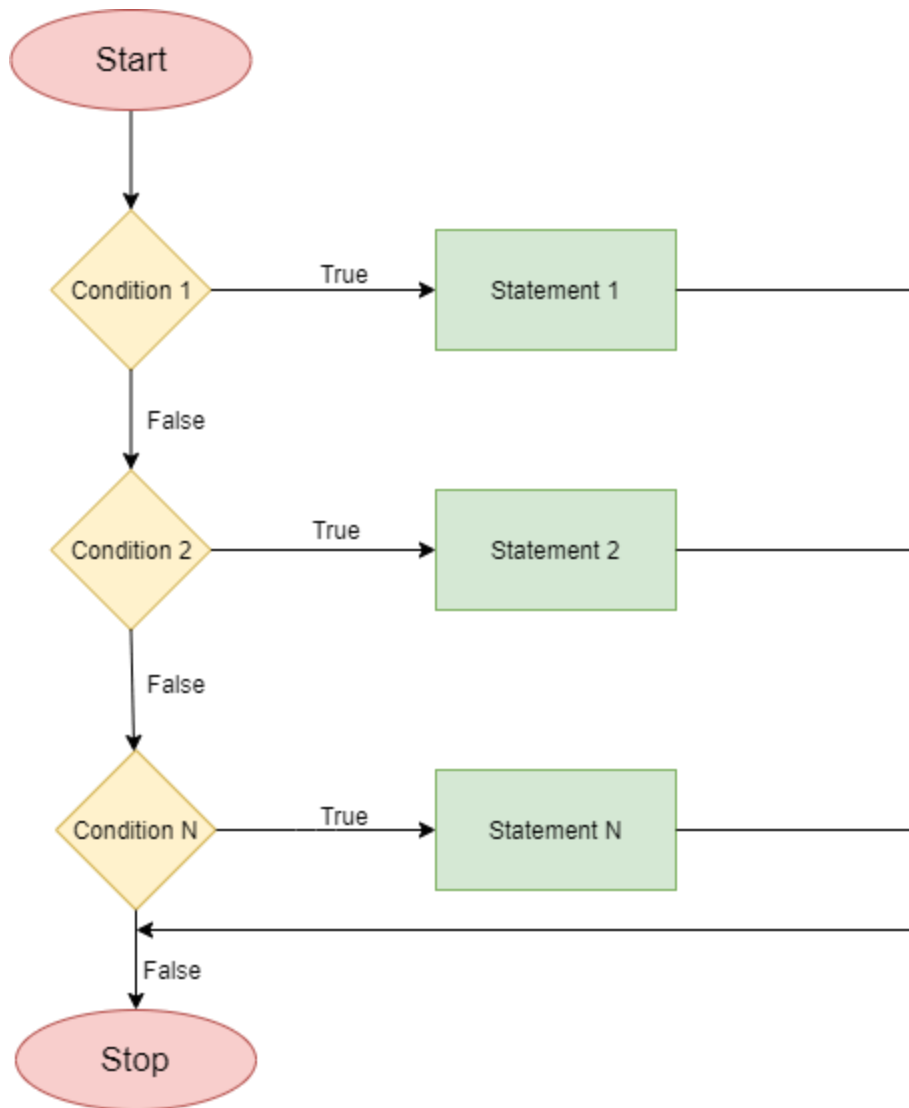
### Explanation -

In the above example, we have an integer variable **num** which stored 20 and we used the if-else statement to check whether a given number is even or odd. The given condition evaluated true because modulus of 20 is equal to 0 then it printed the given number is even on the screen.

## Dart if else-if Statement

Dart if else-if statement provides the facility to check a set of test expressions and execute the different statements. It is used when we have to make a decision from more than two possibilities.

### Dart if else if Statement Flow Diagram



## Syntax

1. **if** (condition1) {
2.   // statement(s)
3. }
4. **else if**(condition2) {
5.   // statement(s)
6. }
7. **else if** (conditionN) {
8.   // statement(s)
9. }

```
10. .  
11. .  
12. else {  
13. // statement(s)  
14. }
```

Here, this type of structure is also known as the else....if ladder. The condition is evaluated from top to bottom. Whenever it found the true condition, statement associated with that condition is executed. When all the given condition evaluates false, then the else block is executed.

Let's understand the following example.

**Example - Write a program to print the result based on the student's marks.**

```
1. void main() {  
2. var marks = 74;  
3. if(marks > 85)  
4. {  
5.     print("Excellent");  
6. }  
7. else if(marks>75)  
8. {  
9.     print("Very Good");  
10. }  
11. else if(marks>65)  
12. {  
13.     print("Good");  
14. }  
15. else  
16. {  
17.     print("Average");  
18. }  
19. }
```

**Output:**

Average

### Explanation -

The above program prints the result based on the marks scored in the test. We have used if else if to print the result. We have initialized the marks variable with the integer value 74. We have checked the multiple conditions in the program.

The marks will be checked with the first condition since it is false, and then it moved to check the second condition.

It compared with the second condition and found true, then it printed the output on the screen.

This process will continue until all expression is evaluated; otherwise the control will transfer out of the else if ladder and default statement is printed.

You should modify the above value and notice the result.

## Nested If else Statement

Dart nested if else statement means one if-else inside another one. It is beneficial when we need a series of decisions. Let's understand the following example.

**Example - Write a program to find the greatest number.**

```
1. void main() {
2.   var a = 10;
3.   var b = 20;
4.   var c = 30;
5.
6.   if (a>b){
7.     if(a>c){
8.       print("a is greater");
9.     }
10.  else{
11.    print("c is greater");
12.  }
13. }
14. else if (b>c){
```

```
15.  print("b is greater");
16. }
17. else {
18.  print("c is greater");
19. }
20. }
```

### Output:

C is greater

In the above program, we have declared three variables a, b, and c with the values 10, 20, and 30. In the outer if-else we provided the condition it checks if a is greater than b. If the condition is true then it will execute the inner block otherwise the outer block will be executed.

In the inner block we have another condition that checks if variable a is greater than c. If the condition is evaluated true then the inner block will be executed.

Our program returned the false in first condition, and then it skipped the inner block check another condition. If satisfied the condition and printed the output on the screen.

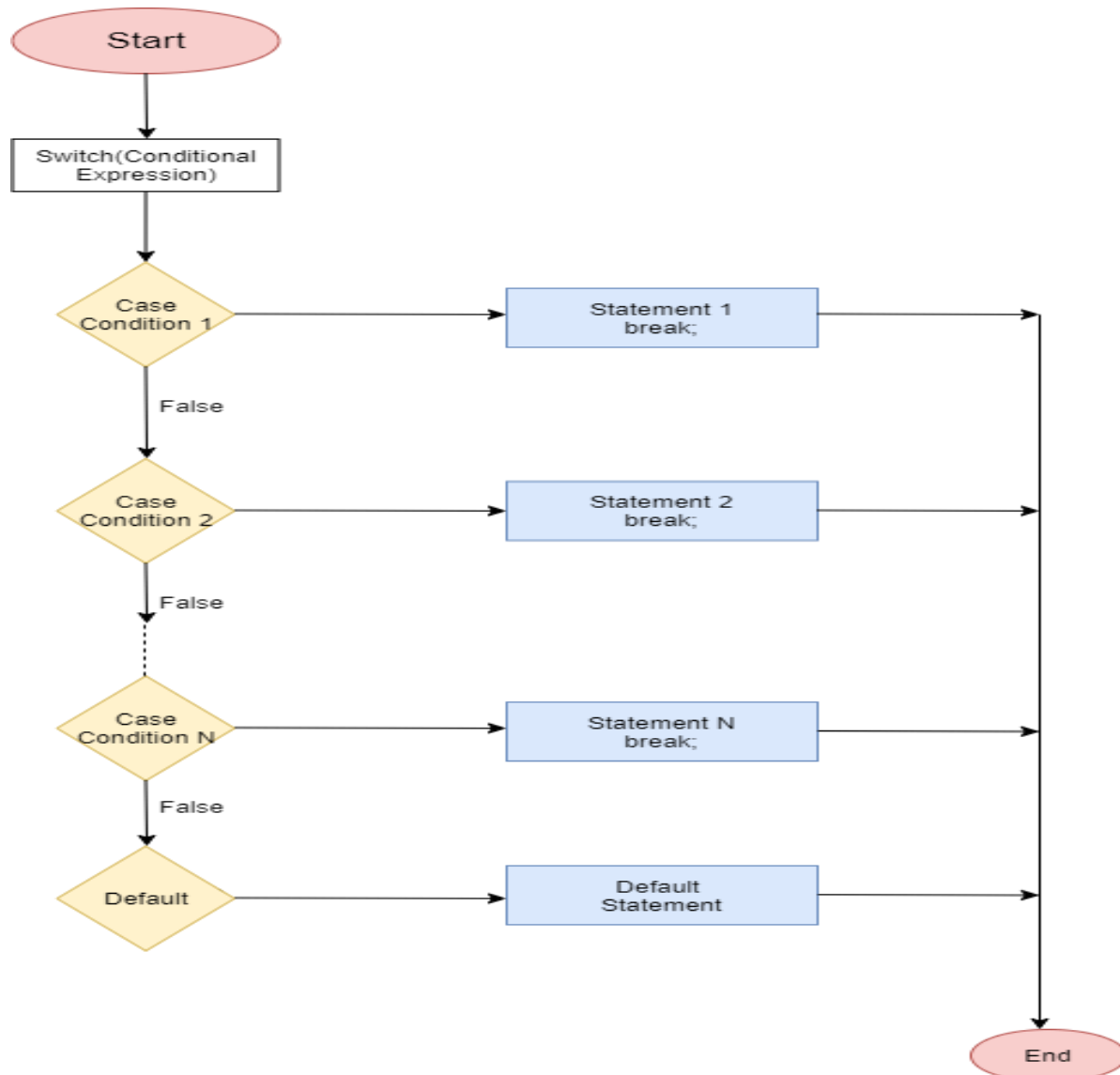
## Dart Switch Case Statement

Dart Switch case statement is used to avoid the long chain of the if-else statement. It is the simplified form of nested if-else statement. The value of the variable compares with the multiple cases, and if a match is found, then it executes a block of statement associated with that particular case.

The assigned value is compared with each case until the match is found. Once the match found, it identifies the block of code to be executed.

### Dart Switch Case Statement Flowchart





The syntax is given below.

### Syntax:

1. **switch**( expression )
2. {
3.     **case** value-1:{
4.         //statement(s)
5.         Block-1;
6.     }
7.     **break**;

```

8.  case value-2:{
9.  //statement(s)
10.      Block-2;
11.      }
12.      break;
13.  case value-N:{
14.      //statement(s)
15.      Block-N;
16.      }
17.      break;
18.  default: {
19.      //statement(s);
20.      }
21. }

```

Here, the expression can be integer expression or character expression. The value 1, 2, n represents the case labels and they are used to identify each case particularly. Each label must be ended with the colon(:).

The labels must be unique because same name label will create the problem while running the program.

A block is associated with the case label. Block is nothing but a group of multiple statements for a particular case.

Once the switch expression is evaluated, the expression value is compared with all cases which we have defined inside the switch case. Suppose the value of the expression is 2, then compared with each case until it found the label 2 in the program.

The **break statement** is essential to use at the end of each case. If we do not put the break statement, then even the specific case is found, it will execute all the cases until the program end is reached. The **break** keyword is used to declare the break statement.

Sometimes the value of the expression is not matched with any of the cases; then the default case will be executed. It is optional to write in the program.

Let's understand the following example.

- 1.
- 2.

```

3.
4. void main() {
5.     int n = 3;
6.     switch (n) {
7.         case 1:
8.             print("Value is 1");
9.             break;
10.        case 2:
11.            print("Value is 2");
12.            break;
13.        case 3:
14.            print("Value is 3");
15.            break;
16.        case 4:
17.            print("Value is 4");
18.            break;
19.        default:
20.            print("Out of range");
21.            break;
22.    }
23. }

```

### Output:

Value is 3

### Explanation -

In the above program, we have initialized the variable **n** with value 3. We constructed the switch case with the expression, which is used to compare the each case with the variable n. Since the value is 3 then it will execute the case-label 3. If found successfully case-label 3, and printed the result on the screen.

Let's have a look at another scenario.

### Example -

```

1. void main()

```

```

2. {
3.  // declaring a interger variable
4.  int Roll_num = 90014;
5.
6.  // Evalaute the test-expression to find the match
7.  switch (Roll_num) {
8.      case 90009:
9.          print("My name is Joseph");
10.         break;
11.         case 90010:
12.             print("My name is Peter");
13.             break;
14.             case 090011:
15.                 print("My name is Devansh");
16.                 break;
17.
18. // default block
19. default:
20.     print("Roll number is not found");
21. }
22. }

```

### Output:

Roll number is not found

### Explanation -

In the above program, we have initialized the variable **Roll\_num** with the value of 90014. The switch test-expression checked all cases which are declared inside the switch statement. The test-expression did not found the match in cases; then it printed the default case statement.

## Benefit of Switch case

As we discussed above, the switch case is a simplified form of if nested if-else statement. The problem with the nested if-else, it creates complexity in the program when the multiple

paths increase. The switch case reduces the complexity of the program. It enhances the readability of the program.

## Dart Loops

Dart Loop is used to run a block of code repetitively for a given number of times or until matches the specified condition. Loops are essential tools for any programming language. It is used to iterate the Dart iterable such as list, map, etc. and perform operations for multiple times. A loop can have two parts - a body of the loop and control statements. The main objective of the loop is to run the code multiple times. Dart supports the following type of loops.

- Dart for loop
- Dart for...in loop
- Dart while loop
- Dart do-while loop

We describe a brief introduction to the dart loops as follows.

### Dart for loop

The for loop is used when we know how many times a block of code will execute. It is quite same as the C for loop. The syntax is given below.

#### Syntax -

1. **for**(Initialization; condition; incr/decr) {
2. // loop body
3. }

The loop iteration starts from the initial value. It executes only once.

The condition is a test-expression and it is checked after each iteration. The for loop will execute until false returned by the given condition.

The incr/decr is the counter to increase or decrease the value.

Let's understand the following example.

### Example -

```
1. void main()
2. {
3.   int num = 1;
4.   for(num; num<=10; num++)    //for loop to print 1-10 numbers
5.   {
6.     print(num);  //to print the number
7.   }
8. }
```

### Output:

```
1
2
3
4
5
6
7
8
9
10
```

## Dart for... in Loop

The for...in loop is slightly different from the for loop. It only takes dart object or expression as an iterator and iterates the element one at a time. The value of the element is bound to var, which is and valid and available for the loop body. The loop will execute until no element left in the iterator. The syntax is given below.

### Syntax -

```
1. for (var in expression) {
2.   //statement(s)
3. }
```

### Example :

```

void main()
{
    var list1 = [10,20,30,40,50];
    for(var i in list1)    //for..in loop to print list element
    {
        print(i);    //to print the number
    }
}

```

### Output:

```

10
20
30
40
50

```

We need to declare the iterator variable to get the element from the iterator.

## Dart while loop

The while loop executes a block of code until the given expression is false. It is more beneficial when we don't know the number of execution. The syntax is given below.

### Syntax:

1. **while**(condition) {
2.     // loop body
3. }

Let's understand the following example.

### Example –

```
void main()
```

```
1. {  
2.     var a = 1;  
3.     var maxnum = 10;  
4.     while(a<maxnum)  
5. {     // it will print until the expression return false  
6.         print(a);  
7.         a = a+1;           // increase value 1 after each iteration  
8. } }
```

**Output:**

```
1  
2  
3  
4  
5  
6  
7  
8  
9
```

## Dart do...while Loop

The do...while loop is similar to the while loop but only difference is that, it executes the loop statement and then check the given condition. The syntax is given below.

**Syntax -**

```
1. do {  
2.     // loop body  
3. } while(condition);
```

**Example -**

```
1. void main()  
2. {  
3.     var a = 1;
```



```
4. var maxnum = 10;
5. do
6. {
7.     print("The value is: ${a}");
8.     a = a+1;
9. }while(a<maxnum);
10. }
```

### Output:

```
The value is: 1
The value is: 2
The value is: 3
The value is: 4
The value is: 5
The value is: 6
The value is: 7
The value is: 8
The value is: 9
```

## Selection of the loop

The selection of a loop is a little bit difficult task to the programmer. It is hard to decide which loop will be more suitable to perform a specific task. We can determine the loop based on the following points.

- Analyze the problem and observe that whether you need a pre-test loop or post-test loop. A pre-test loop is that, the condition is tested before entering the loop. In the post-test loop, the condition is tested after entering the loop.
- If we require a pre-test loop, then select the while or for loop.
- If we require a post-test loop, then select the do-while loop.

## Nested for Loop

The nested for loop means, "the for loop inside another for loop". A for inside another loop is called an inner loop and outside loop is called the outer loop. In each iteration of

the outer loop, the inner loop will iterate to entire its cycle. Let's understand the following example of nested for loop.

### Example -

```
1. void main()
2. {
3.     int i, j;
4.     int table_no = 2;
5.     int max_no = 10;
6.     for (i = 1; i <= table_no; i++) { // outer loop
7.         for (j = 0; j <= max_no; j++) { // inner loop
8.             print("${i} * ${j} = ${i*j}");
9.             //print("\n"); /* blank line between tables */
10.        }
11.    }
12. }
```

### Output:

```
1 * 0 = 0
1 * 1 = 1
1 * 2 = 2
1 * 3 = 3
1 * 4 = 4
1 * 5 = 5
2 * 0 = 0
2 * 1 = 2
2 * 2 = 4
2 * 3 = 6
2 * 4 = 8
2 * 5 = 10
```

Let's understand the working of nested for loop.

### Example - 2 Understand the inner loop cycle

```
1. void main(){
2.     for(int i = 1; i <= 5; i++) {
```

```
3.  
4.     print("Outer loop iteration : ${i}");  
5.  
6.     for (int j = 1; j <= i; ++j) {  
7.         print("i = ${i} j = ${j}");  
8.     }  
9.  
10. }  
11. }
```

### Output:

```
Outer loop iteration : 1  
i = 1 j = 1  
Outer loop iteration : 2  
i = 2 j = 1  
i = 2 j = 2  
Outer loop iteration : 3  
i = 3 j = 1  
i = 3 j = 2  
i = 3 j = 3  
Outer loop iteration : 4  
i = 4 j = 1  
i = 4 j = 2  
i = 4 j = 3  
i = 4 j = 4  
Outer loop iteration: 5  
i = 5 j = 1  
i = 5 j = 2  
i = 5 j = 3  
i = 5 j = 4  
i = 5 j = 5
```

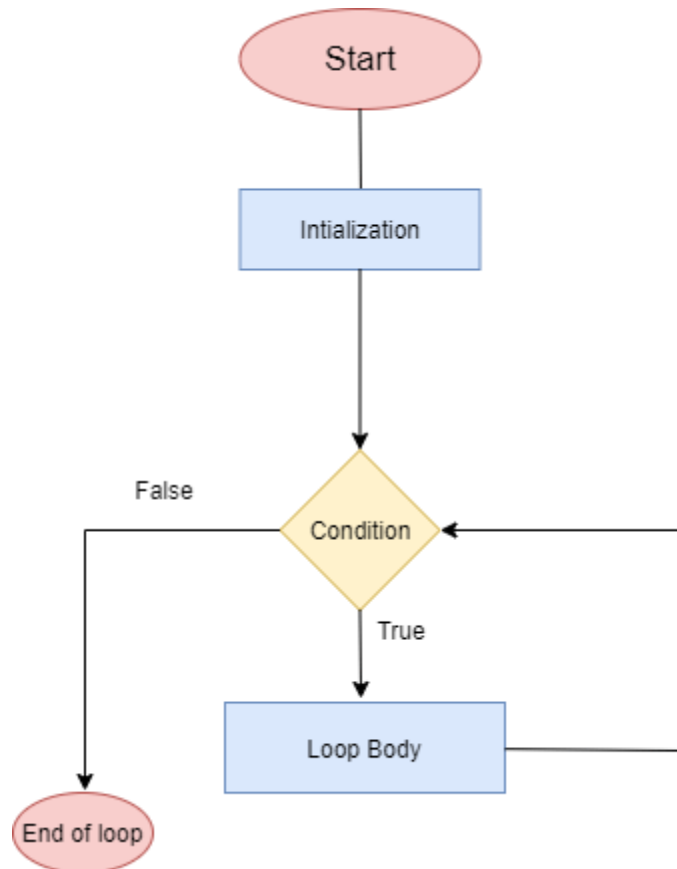
Observe the above code, we have defined the working of the inner loop. The inner loop will be repeated for each iteration of the outer loop.

## Dart While Loop

The while loop is used when the number of execution of a block of code is not known. It will execute as long as the condition is true. It initially checks the given condition then

executes the statements that are inside the while loop. The while loop is mostly used to create an infinite loop.

## Dart While Loop Flowchart



The syntax is given below.

### Syntax:

1. **while**(condition){
2.     //statement(s);
3.     // Increment (++) or Decrement (--) Operation;
4. }

Here, if the condition returns the true, then the loop body is executed and condition is evaluated again. If the condition returns false then the loop is terminated and the control transfer to out of the loop.

Let's understand the following example.

### Example - 1

```
1. void main()
2. {
3.     int i = 1;
4.     while (i <= 5)
5.     {
6.         print( i);
7.         ++i;
8.     }
9. }
```

### Output:

```
1
2
3
4
5
```

### Explanation:

Above example, we initialized the integer variable *i* with value 1 respectively, in the next statement we have defined while loop, that check the condition that, the value of *i* is smaller or greater than 5 in each iteration.

If the condition returns true then while loop body is executed and the condition is rechecked. It will be continued until the condition is false.

After that, the value of *i* is 6 that violated the condition; then, the loop is terminated. It printed the sequence of 1 to 5 on the console.

## Infinite While Loop

When the while loop executes an endless time is called infinite while loop. Let's have a look at the infinite loop example.

### Example -

```

1. void main()
2. {
3.     int i = 1;
4.
5.     while (i <= 5)
6.     {
7.         print( i);
8.         --i;
9.     }
10. }

```

We have made only one change in above code. We reduced the value of i for each iteration of while loop. So it will never match with the specified condition and became an infinite loop.

### Example - 2

```

1. void main()
2. {
3.     while (true)
4.     {
5.         print("Welcome");
6.     }
7. }

```

It will print the given statement for infinite time. When we declare Boolean true in while loop, then it automatically became an infinite loop.

## Logical Operator while loop

Sometimes we need to check the multiple conditions in a while loop. We can do this by using logical operators such as (||, &&, and !). Let's see the following concepts.

- **while (n1<5 && n2>10)** - It will execute if both conditions are true.
- **while (n1<5 || n2>10)** - It will execute if one of the condition is true.
- **while(!n1 = 10)** - It will execute if n1 is not equal to 10.

Consider the following example.

### Example -

```
1. void main() {
2.   int n1=1;
3.   int n2=1;
4.   // We are checking multiple condition by using logical operators.
5.   while (n1 <= 4 && n2 <= 3)
6.   {
7.     print("n1 : ${n1}, n2: ${n2}");
8.     n1++;
9.     n2++;
10.  }
11. }
```

### Output:

```
n1 : 1, n2: 1
n1 : 2, n2: 2
n1 : 3, n2: 3
```

### Explanation:

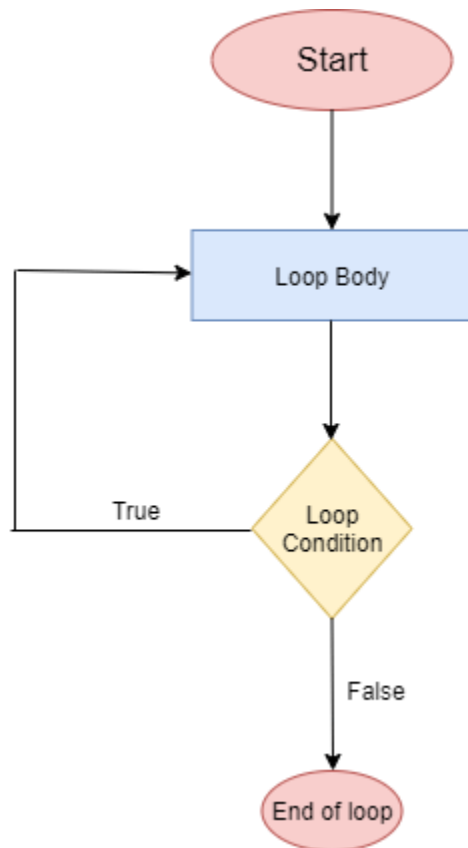
In the above code, we have assigned two variables **n1** and **n2** with the value 1 in both. Now we checked multiple conditions in the while loop where **n1** is less than or equal to 4 and **n2** is less than or equal to 3.

In the first iteration, it checked both values and printed the result. At one point, when the value of **n1** and **n2** is equal to 4. The **n1** satisfied the condition one, but **n2** did not meet the second condition, so the loop is terminated and printed the result to the screen.

## Dart do while Loop

Dart do while loop executes a block of the statement first and then checks the condition. If the condition returns true, then the loop continues its iteration. It is similar to Dart while loop but the only difference is, in the do-while loop a block of statements inside the body of loop will execute at least once.

## Dart do-while loop Flowchart



The syntax is given below.

### Syntax:

1. **do** {
2. *// loop body*
3. } **while**(condition);

Here, the block of statement which is inside the do while body will executes first then evaluates the given condition.

A condition is evaluated either Boolean true or false. If it returns true, then the statements are executed again, and the condition is rechecked. If it returns false, the loop is ended, and control transfers to out of the loop.

Let's understand the following example.



### Example:

```
1. void main() {  
2.   int i = 10;  
3.   print("Dart do-while loop example");  
4.  
5.   do{  
6.  
7.  
8.     print(i);  
9.     i++;  
10.  
11. }while(i<=20);  
12. print("The loop is terminated");  
13. }
```

### Output:

```
Dart do-while loop example  
The value of i: 10  
The value of i: 11  
The value of i: 12  
The value of i: 13  
The value of i: 14  
The value of i: 15  
The value of i: 16  
The value of i: 17  
The value of i: 18  
The value of i: 19  
The value of i: 20  
The loop is terminated
```

### Explanation -

In the above code, we have initialized the variable `i` with value 10. In do-while loop body, we defined two statements.

In the first iteration, the statement printed the initial value of `i` and increased by 1. Now the value of `i` becomes 11 and then we checked the condition.

The condition is, the value of *i* must be less than or greater than the 20. It matched with the condition and loop moved to the next iteration. It printed the series of numbers 10 to 20 until the condition returned false.

## Dart Function

Dart function is a set of codes that together perform a specific task. It is used to break the large code into smaller modules and reuse it when needed. Functions make the program more readable and easy to debug. It improves the modular approach and enhances the code reusability.

Suppose, we write a simple calculator program where we need to perform operations number of times when the user enters the values. We can create different functions for each calculator operator. By using the functions, we don't need to write code for adding, subtracting, multiplying, and divide again and again. We can use the functions multiple times by calling.

The function provides the flexibility to run a code several times with different values. A function can be called anytime as its parameter and returns some value to where it called.

## Advantages of Functions

The few benefits of the Dart function is given below.

- It increases the module approach to solve the problems.
- It enhances the re-usability of the program.
- We can do the coupling of the programs.
- It optimizes the code.
- It makes debugging easier.
- It makes development easy and creates less complexity.

Let's understand the basic concept of functions.

# Defining a Function

A function can be defined by providing the name of the function with the appropriate parameter and return type. A function contains a set of statements which are called function body. The syntax is given below.

## Syntax:

```
return_type func_name (parameter_list):
{
    //statement(s)
    return value;
}
```

Let's understand the general syntax of the defining function.

- **return\_type** - It can be any data type such as void, integer, float, etc. The return type must be matched with the returned value of the function.
- **func\_name** - It should be an appropriate and valid identifier.
- **parameter\_list** - It denotes the list of the parameters, which is necessary when we called a function.
- **return value** - A function returns a value after complete its execution.

Let's understand the following example.

## Example - 1

```
1. int summ (int a, int b){
2.     int c;
3.     c = a+b;
4.     print("The sum is:${c}");
5. }
```

## Calling a Function

After creating a function, we can call or invoke the defined function inside the `main()` [function](#) body. A function is invoked simply by its name with a parameter list, if any. The syntax is given below.

### Syntax:

1. `fun_name(<argument_list>);`
2. or
3. `variable = function_name(argument);`

**Note - Calling function must be ended with semicolon (;).**

When we call a function, the control is transferred to the called function. Then the called function executes all defined statements and returns the result to the calling function. The control returns to the `main()` function..

### Example :

1. `summ (10,20);`

## Passing Arguments to Function

When a function is called, it may have some information as per the function prototype is known as a parameter (argument). The number of parameters passed and data type while the function call must be matched with the number of parameters during function declaration. Otherwise, it will throw an error. Parameter passing is also optional, which means it is not compulsory to pass during function declaration. The parameter can be two types.

**Actual Parameter** - A parameter which is passed during a function definition is called the actual parameter.


**Formal Parameter** - A parameter which is passed during a function call is called the formal parameter.

We will learn more about the parameter in the next tutorial.

## Return a Value from Function

A function always returns some value as a result to the point where it is called. The **return** keyword is used to return a value. The return statement is optional. A function can have only one return statement. The syntax is given below.

```
int sum (int a, int b){  
    .....  
    .....  
    return result;  
}  
  
var c = sum(30,20) ←
```

A diagram showing a line from the 'return result;' statement in the function definition, extending to the right and then turning downwards to point at the 'sum(30,20)' argument in the function call 'var c = sum(30,20)'. This illustrates how the value returned by the function is passed to the caller.

### Syntax:

1. **return** <expression/values>

### Example -

1. **return** result;

## Function Examples

Let's understand the functions by using a program of adding two numbers using functions.

### Dart Function with parameter and return value

In the following example, we are creating a sum() function to add two number.

#### Example - 1

1. **void** main() {
2.   print("Example of add two number using the function");
3.   // Creating a Function

```

4.
5.  int sum(int a, int b){
6.      // function Body
7.      int result;
8.      result = a+b;
9.      return result;
10. }
11. // We are calling a function and storing a result in variable c
12. var c = sum(30,20);
13. print("The sum of two numbers is: ${c}");
14. }

```

## Output

```

Example of add two number using the function
The sum of two numbers is: 50

```

## Explanation:

In the above example, we declared a function named **sum()** and passed two integer variables as actual parameters. In the function body, we declared a **result** variable to store the sum of two numbers and returned the result.

In order to add two values, we called a function with the same name, passed formal parameters 30 and 20. The **sum()** returned a value which we stored in the variable c and printed the sum on the console.

## Dart Function with No Parameter and Return Value

As we discussed earlier, the parameters are optional to pass while defining a function. We can create a function without parameter return value. The syntax is given below.

## Syntax:

```

1. return_type func_name()
2. {
3.     //Statement(s);
4.     return value;
5. }

```

Let's understand the following example.

### Example - 2

```
1. void main(){
2. // Creating a function without argument
3. String greetings(){
4.     return "Welcome to JavaTpoint";
5. }
6. // Calling function inside print statement
7. print(greetings());
8. }
```

### Output

```
Welcome to JavaTpoint
```

### Explanation:

In the above example, we created a function named **greetings()** without argument and returned the string value to the calling function. Then, we called the `greeting()` function inside the print statement and printed the result to the console.

## Dart Function with No Parameter and without a Return Value

We can declare a function without parameter and no return value. The syntax is given below.

### Syntax:

```
1. func_name() {
2. //statement
3. }
4. Or
5. void func_name() {
6. //statement(s)
7. }
```

In the above general syntax-

**void** - It represents the function has no return type.

**fun\_name** - It represents the function name.

Let's understand the following example.

### Example - 3

```
1. // Creating a function without argument
2. void greetings()
3. {
4.   print("Welcome to JavaTpoint");
5. }
6. void main() {
7.   print("The example of Dart Function");
8.   // function calling
9.   greetings();
10. }
```

### Output

```
The example of Dart Function
Welcome to JavaTpoint
```

### Explanation:

In the above example, we created a function called **greeting()** outside the **main()** function and writing the print statement. Inside the **main()** function, we called the defined function and printed the output to console.

## Dart Function with Parameter and without a Return Value

We are creating a function to find the given number is even or odd. Let's understand the following example.

### Example - 4

```
1. void main()
2. {
3.   void number(int n){
```



```
4.      // Check the given number is even or odd
5.      if (n%2 == 0){
6.          print("The given number is even");
7.      }
8.      else {
9.          print("The given number is odd");
10.     }
11. }
12. number(20);
13. }
```

### Output

```
The given number is even
```

## Dart Anonymous Function

We have learned the Dart Function, which is defined by using a user-defined name. Dart also provides the facility to specify a nameless function or function without a name. This type of function is known as an **anonymous function, lambda, or closure**. An anonymous function behaves the same as a regular function, but it does not have a name with it. It can have zero or any number of arguments with an optional type annotation.

We can assign the anonymous function to a variable, and then we can retrieve or access the value of the closure based on our requirement.

An Anonymous function contains an independent block of the code, and that can be passed around in our code as function parameters. The syntax is as follows.

### Syntax:

```
1. (parameter_list) {
2.     statement(s)
3. }
```

Let's consider the following example.

## Example -

```
void main() {  
  var list = ["James","Patrick","Mathew","Tom"];  
  print("Example of anonymous function");  
  list.forEach((item) {  
    print('${list.indexOf(item)}: $item');  
  });  
}
```

### Output:

```
Example of anonymous function  
0: James  
1: Patrick  
2: Mathew  
3: Tom
```

## Explanation:

In the above example, we defined an anonymous function with an untyped argument `item`. The function is called for each item in the list and prints the strings with its specified index value.

If the function consists of one statement, then we can also write the above code in the following way.

1. `list.forEach(`
2. `(item) => print('${list.indexOf(item)}: $item');`

It is equivalent to the previous code. You can verify it by pasting it in your Dart pad and running it.

## Lexical Scope

As we have discussed in the Dart introduction, it is a lexical scope language which means the variable's scope is decided at compile-time. The scope of the variable is determined when code is compiled. The variable behaves differently if they are defined in different curly braces. Let's understand the following example.

## Example -

```
1. bool topVariable = true;  
2.  
3. void main() {  
4.   var inside_Main = true;  
5.   // Defining Nested Function  
6.  
7.   void myFunction() {  
8.     var inside_Function = true;  
9.  
10.  void nestedFunction() {  
11.    var inside_NestedFunction = true;  
12.    // This function is using all variable of the previous functions.  
13.    assert(topVariable);  
14.    assert(inside_Main);  
15.    assert(inside_Function);  
16.    assert(inside_NestedFunction);  
17.  }  
18. }  
19. }
```

Observe the above code, the **nestedFunction()** used the variables of the previous function.

## Lexical Closure

A lexical closure is referred to as a closure, is a function object that has access to variables in its lexical scope even when the function is used of its original scope. In other words, it provides access to an outer function's scope from inner function. Let's understand the following example.

## Example -

```
1. void main() {  
2.   String initial() {
```

```

3.   var name = 'Will Smith'; // name is a local variable created by init
4.
5.   void disp_Name() { // displayName() is the inner function, a closure
6.       print(name); // use variable declared in the parent function
7.   }
8.   disp_Name();
9. }
10. init();

```

### Output

```
Will Smith
```

### Explanation:

In the above code, the **initial()** function created a local variable called **name** and function called **disp\_Name()**. The **disp\_Name()** function defined inside the **initial()** function and hence **disp\_Name()** function has no local variable its own.

The inner function can access the variable of the outer functions. The function **disp\_Name()** can access the **name** variable which is declared in the outer function, **initial()**.

## Dart Return Value

Sometimes the function returns a value after evaluating the function statements to the point where it is called from. The return statement holds the result of the function, and it is transferred to the function call. The **return** keyword is used to represent the return statement. If the return statement not specified, then the function returns null. The return statement is optional to specify in function, but there can be only one return statement in a function.

### Syntax:

```
1. return <expression/value>;
```

## Dart value with Return Value

Below is given syntax of return value.

## Syntax:

1. return\_type function\_name()
2. {
3.   //statement(s);
4.   **return** value;
5. }

Here is the description of the above syntax.

**function\_name** - It represents the function name, which can be any valid identifier.

return type - It denotes the return type of the function. It can be any valid data type. The return must be matched with the return type of the function.

Let's understand the following example -

## Example -

1. **void** main() {
2.   **int** mul(**int** a, **int** b){
3.     **int** c = a\*b;
4.     **return** c;
5. }
6. print("The multiplication of two numbers: \${mul(10,20)}");
7. }

## Output

```
The multiplication of two numbers: 200
```