


# Introduction to Flutter

Module – 1

*Introduction to Flutter and Dart Programming Language*  
*The EasyLearn Academy*

Call us for training on +91 9662512857



# What is Flutter?

- Flutter is a set of tools that allows you to create beautiful apps that run on iOS, Android, the Web, and desktop (Desktop is coming soon. Flutter will work on Windows, macOS, Chromebooks, and Linux)
- It's latest version is 2.8
- Flutter is
  - Free
  - Open source
  - originated at Google
  - Being enhanced and maintained by a team of developers at Google and many non-Google contributors around the world.
  - is in use by developers in organizations in whole world for production apps
  - Fast because it compiles to truly native apps that don't use WebViews and JavaScript bridges.
  - Written one place and compiled to a web app, IOS app, android app.

# Why you should learn Flutter?

- Because google's mission about flutter is *To build a better way to develop for mobile.*
- Notice what is not in that mission. There's no mention of Android nor of iOS nor of the Web.
- Flutter's goal is to create a better way to develop for all devices.
- In other words, Flutter should be better to create iOS apps than Swift. It should be better to create Android apps than Kotlin.
- It should be better to create web apps than HTML/JavaScript. And if you get all of those things simultaneously with one codebase, all the better.

# Cross-platform development categories

	Technology	Pros	Cons
Progressive Web Apps (PWA)	HTML/CSS, react, angular, Vue	Easy to create	Not a real app. runs in a web browser. Not available in app stores. hard to create a desktop shortcut. Cannot access many of the device's resources like accelerometer, compass, proximity sensor, Bluetooth, NFC, and more
Hybrid	phoneGap, Cordova, Sencha, Ionic	Easy to learn by web developers	runs in a WebView so it can be slow. Nearly impossible to share code with the web app
native solutions	react Native, NativeScript, <b>Flutter</b> , Xamarin	run fast, Future of technology.	learning a framework may be difficult.



# Which is best for what use cases?

- If you have a very loyal audience, one where users value your app so much that they're willing to accept a poorer user experience, the cheapest solution is to create a PWA. So you can develop such app using html/CSS, react, angular, Vue.
- If your app is extremely basic and speed is not an issue, a **hybrid** solution might be good. So you can develop such app using phoneGap, Cordova, Sencha, Ionic.
- But if speed, smoothness, and sophisticated capability are important, you will need to go with a **native** solution. So you can develop such app using react Native, NativeScript, **Flutter**, Xamarin

# Native solutions

- As of today, there are four fairly popular compile-to-native solutions

## *Compile-to-native cross-platform frameworks*

				
	Xamarin	NativeScript	React Native	Flutter
<b>Year introduced</b>	2011	2014	2015	2018
<b>Backed by</b>	Microsoft	Telerik	Facebook	Google
<b>Presentation language</b>	XAML and/or xamarin.forms	Proprietary but looks like XML	Proprietary but looks like JSX	Dart
<b>Procedural language</b>	C#	JavaScript	JavaScript	Dart

# Why flutter is better choice?

- These are all decent options.
- All are free to develop in and are well- tested, having many production applications created. All have been used in large organizations.
- **But only one has an option to create a web application in addition to the iOS and Android apps that will be deployed to the app stores – Flutter.**
- Flutter is the latest of these frameworks to be released. As such it has a distinct advantage of observing those that had come before.
- The Flutter team took note of what worked well with other frameworks and what failed.
- In addition, Flutter added new innovations and ideas – all developed in from the scratch rather than being bolted on as improvements are made.

# Importance of flutter ...

## ➤ Same Codebase for all platforms

- here you have the same business logic and same UI in all platforms whether it is android, iOS, web, desktop it enables you to work in the concept of write once run everywhere.

## ➤ Hot-Reload

- it has always been a problem to observe sudden and small changes while writing code and especially during UI design, Flutter has overcome this problem with its hot-reload feature. It will not only help you to fix bugs easily but also add new features without effecting its speed.

## ➤ Architecture

- Building an app is easy but managing it according to business logic is quite a critical task so you need to follow an architecture to manage it and MVP(**Model-view-presenter**) is the best suitable architecture for it.
- its benefits in easy integration, maintaining speed and responsiveness of it.



# Importance of flutter ...

## ➤ Performance

- If you compare native apps and flutter on the basis of its performance in the flutter code is written in dart and which do not use JavaScript so it helps to boost its performance in terms of speed which is **60 FPS**.

## ➤ Dart

- Flutter use Dart & Dart is an object-oriented, garbage-collected, class defined, language using a C-style syntax that trans-compile optionally into JavaScript.
- Dart uses ahead of time compilation and , it helps to start fast & you can customize its widgets.

## ➤ Material Design

- Flutter use material design, Material Design works across different types of device and OS. **Your design will look same no matter what device or OS is used to use your app.**

# Importance of flutter

## ➤ Add to App

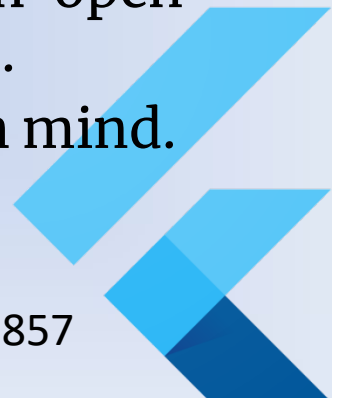
- Another Feature of Flutter is it makes developers enable to add flutter in the current Android and iOS app,
- Migration of an app in flutter is possible once, instead of creating it from the beginning.

## ➤ Animation

- In flutter, there are some inbuilt basic animations to make beautiful applications but if you want to add more animations in your app you can use Rive.

## ➤ Fuchsia

- Fuchsia is being developed by Google which is an open-source operating system works on any kind of device.
- Google flutter has been developed keeping Fuchsia in mind.
- Fuchsia has its own micro-kernel called **Zircon**.



# Disadvantages

- A developer needs to learn new language (however it is easy to learn).
- Modern framework tries to separate logic and UI as much as possible but, in Flutter, user interface and logic is written together.
- We can solve this problem using smart coding and using high level module to separate user interface and logic.



# Introduction to Dart

- Flutter apps are developed using Dart Programming Language.
- Dart is a compiled, statically typed, object-oriented, procedural programming language.
- Dart is an open-source general-purpose programming language developed by Google.
- It supports application development in both client and server-side. But it is widely used for the development of android apps, iOS apps, IoT(Internet of Things), and web applications using the Flutter Framework.
- It has a very mainstream structure much like other OO languages, making it very easy to learn for student/developers who have experience with Java, C#, C++, or other OO, C-like languages.
- It supports programming concepts like interfaces, classes, unlike other programming languages Dart doesn't support arrays.
- Dart collections can be used to replicate data structures such as arrays, generics, and optional typing.

# Dart program may have.....

- Variables and Operators
- Classes
- Functions
- Expressions and Programming Constructs
- Decision Making and Looping Constructs
- Comments
- Libraries and Packages
- Typedefs
- Data structures represented as Collections / Generic



# Introduction to Dartpad

- Dartpad is a browser-based text editor that can execute Dart code. Now it also has option to run Flutter code.
- DartPad is created by the Dart team.
- If you're just starting with Dart, you can use Dartpad to run your Dart code without installing the Dart SDK and IDE plugins.
- It's features include a linter, analyzer, and embedded web view, enabling you to play with HTML and CSS with your Dart code.

*Linting is the automated checking of your source code for programmatic and stylistic errors.*

- To use Dartpad open <https://dartpad.dev/>
- Currently it do not provide facility to store file so you have to create file on your disk with .dart extension to use it later.

# What is Dart SDK?

- In order to do a lot of interesting programming stuff using Dart programming language, we have to install the Dart SDK.
- The Dart SDK has the libraries and command-line tools that you need to develop Dart web, command-line, and server apps. If you're developing Flutter apps, then you don't need to separately download the Dart SDK; just install Flutter.
- Dart SDK is a pre-compiled version so we have to download and extract it only.
- Now let us see what it has



# What's in the Dart SDK?

- The Dart SDK includes a lib directory for the Dart libraries and a bin directory that has these command-line tools:
- **dart**
  - The Dart command-line tool
- **dart2native**
  - A tool that AOT compiles Dart code to native x64 machine code, producing either a standalone executable or a snapshot.
- **dartaotruntime**
  - A Dart runtime for AOT-compiled snapshots
- **dart2js**
  - The Dart-to-JavaScript compiler (used only for web development)
- **dartanalyzer**
  - The static analyzer





# How to install it?

- Download Dart SDK. Download Dart SDK from the Dart SDK from the below url.
- <https://docs.flutter.dev/get-started/install>
- Click on DART SDK to download SDK for Windows 64-Bit Architecture.
- The download will start and a zip file will be downloaded into download folder.
- Extract the downloaded zip file in any drive lets say G drive. Extract the contents of downloaded zip file.
- Now open bin folder of dart sdk folder in command prompt
- Type dart command to see is it working or not.
- If command run successfully it means you have installed DART-SDK properly.
- Now set path of dart.exe in environment variable to run dart file anywhere from your computer



**Now let us learn dart in detail.**

Call us for training on +91 9662512857

# More about dart

- Dart has C-style syntax so easy to learn & use.
- It can optionally trans compile into JavaScript, classes, collections, generics, and optional typing.
- Dart can be used to create single-page applications. Single-page applications apply only to websites and web applications.
- Single-page applications enable navigation between different screens of the website without loading a different webpage in the browser.



# First program in dart

```
void main()  
{  
    print("hello world....");  
    print("we are learning Dart Programming @ The  
    easylearn academy");  
}
```

- The main() function is a predefined method in Dart.
- This method acts as the entry point to the application.
- A Dart script needs the main() method for execution.
- print() is a predefined function that prints the specified string or value to the standard output i.e. the terminal.

# How to run dart file in visual code?

- Open folder which has dart file in visual studio code editor
- Run following command
  - `dart filename.dart`
- For example if you have `hello.dart` file then run it
  - `dart hello.dart`



# Comments

## ➤ Types of Dart Comments:

1. Dart Single line Comment.

`//` is single line comment

2. Dart Multiline Comment.

`/* */` is multi line comment

1. Dart Documentation Comment.

`///` is dart documentaion comment



# Variables

- A variable is named memory location where the user stores the data and that data can be fetched when required using variable name.
- There are various types of variable which are used to store the data.
- The type which will be used to store data depends upon the type of data we want to store in it.
- Variable Declaration:
  - To declare a variable:  
Syntax:  
`type variable_name;`
  - To declare multiple variables of same type:  
Syntax:  
`type variable1_name, variable2_name, variable3_name;`

# Variable's data-type

## ➤ data type of variable can be

1. Integer
2. Double
3. String
4. Booleans
5. Lists
6. Maps

## ➤ Rule for identifiers name

1. identifiers can't be the keyword.
2. identifiers can contain alphabets and numbers `_` and `$`.
3. identifiers can't contain spaces and special characters, except the underscore(`_`) and the dollar(`$`) sign.
4. identifiers can't begin with number.

Let us see example





```
void main() {  
    int age = 10;  
    double location = 21.424567;  
    bool gender = true;  
    String name = "The EasyLearn Academy";  
  
    print(age);  
    print(location);  
    print(gender);  
    print(name);  
  
    print("age $age");  
    print("location $location");  
    print("gender $gender");  
    print("name $name");  
  
    print("age " + age.toString());  
    print("location " + location.toString());  
    print("gender " + gender.toString());  
    print("name " + name);  
}
```



# Keywords in dart

- Keywords are the set of reserved words which can't be used as a variable name or identifier
- Keywords are used to perform very specific task. Such as variable declaration.
- There are currently 33 keywords in dart.

assert	default	finally	rethrow	try
break	do	for	return	var
case	else	if	super	void
catch	enum	in	switch	while
class	extends	is	this	with
const	false	new	throw	
continue	final	null	true	

# How to take input from user?

- In Dart programming language, you can take standard input from the user through the console via the use of `.readLineSync()` function.
- To take input from the console you need to import a library, named **dart:io** from libraries of Dart.

```
import 'dart:io';  
void main() {  
    print("Enter your name");  
    String? name = stdin.readLineSync();  
    print("Hello, $name");  
    print("what is your age");  
    int age = int.parse(stdin.readLineSync().toString());  
    print("The number is $age");  
    print("what is your weight");  
    double weight =  
double.parse(stdin.readLineSync().toString());  
    print("your weight = $weight");  
}
```

# Dynamic type variable in Dart:

- This is a special variable initialised with keyword dynamic.
- this type of variable can store any value.

Syntax:

```
dynamic variable_name;
```

example

```
void main() {  
    dynamic value = "the easylearn academy";  
    print(value);  
  
    value = 91;  
    print(value);  
  
    value = true ;  
    print(value);  
  
    value = 1.23456;  
    print(value);  
}
```



# final & const Keyword

- final and const keywords are used to define constant variable in **Dart**
- constant variable must be initialized at the time of creating variable and its value can not be changed during execution of program if you try to do so you will get error
- **These** keyword can be used with or without data type name.

## **Syntax for Final:**

*// Without datatype*

final variable\_\_name

*// With datatype*

final data\_\_type variable\_\_name

## **Syntax for Const:**

*// Without datatype*

const variable\_\_name

*// With datatype*

const data\_\_type variable\_\_name


# example

```
void main() {  
    final name = "The Easylearn Academy";  
    print(name);  
    final String address = "Hill Drive, Bhavnagar";  
    print(address);  
    const value = 12.7;  
    const double value2 = 10.2;  
    print(value.toString() + " " + value2.toString());  
    // value = 11.5; wont work  
}
```



# List in dart

Call us for training on +91 9662512857



# List

- List in Dart is special type of variable which can hold more than one value.
- it is like array in c language.
- each value in list has unique integer position which start with zero known as index
- value can be accessed or changed using index
- list can hold either mix value of fixed type value such as integer
- list can be of either fixed size or it can hold as many value as we need (growable/dynamic list).
- List is an ordered collection which maintains the insertion order of the items.
- List allows duplicates and null values.
- While an operation on the list is being performed, modifying the list's length (adding or removing items) will break the operation.
- list has built in methods which we can use to work upon it



## Example 1/6

```
void main(){
    //Fixed type Fixed List
    int InitialSize = 3;
    String Defaultvalue = '';
    List<String> myList =
    List.filled(InitialSize,Defaultvalue,growable:false);
    myList[0] = 'the';
    myList[1] = 'easy';
    myList[2] = 'learn';
    print(myList);
    //myList.add('Bhavnagar'); won't work because size is
    fixed
}
```

## Example 2/6

```
void main()
{
    //fixed type Dynamic list
    int InitialSize = 2;
    int Defaultvalue = 0;
    List<int> DynamicList = new List<int>.filled(InitialSize, Defaultvalue,growable:
true);
    DynamicList[0] = 0;
    DynamicList[1] = 10;
    DynamicList.add(100);
    DynamicList.add(200);
    print('Elements in the list are as follows: $DynamicList');
    var myList = ['India', 2,true,10.2]; //mixed growable list
    myList.add(false);
    myList.add(123);
    print(myList);
    var dish= []; //empty list
    print(dish);
}
```

## Example 3/6

```
void main()
{
    //fixed type Dynamic list
    int InitialSize = 2;
    int Defaultvalue = 0;
    List<int> DynamicList = new List<int>.filled(InitialSize,
Defaultvalue,growable: true);
    DynamicList[0] = 0;
    DynamicList[1] = 10;
    DynamicList.add(100);
    DynamicList.add(200);
    print('Elements in the list are as follows: $DynamicList');
    var myList = ['India', 2,true,10.2]; //mixed growable list
    myList.add(false);
    myList.add(123);
    print(myList);
    var dish= []; //empty list
    print(dish);
}
```

## Example 4/6

```
void main(){
    var list1 = [1,2,3];
    var list2 = [4, 5];
    var list3 = [6, 7, 8];
    var list4 = null;
    // from() and addAll() method
    var combinedList1= List.from(list1)..addAll(list2)..addAll(list3);
    print(combinedList1);
    // expand() method
    var combinedList2 = [list1, list2, list3].expand((x) => x).toList();
    print(combinedList2);
    // operator +
    var combinedList3 = list1 + list2 + list3;
    print(combinedList3);
    // spread operator (list1,list2,list3 must not be null or we get error)
    var combinedList4 = [...list1, ...list2, ...list3];
    print(combinedList4);
    var combinedList5 = [...?list1, ...?list2, ...?list3,...?list4];
    // null-aware spread operator ...?check & remove null list automatically
    print(combinedList5);
}
```

## Example 5/6

```
void main() {  
    var myList = [0, 'one', 'two', 'three', 'four', 'five'];  
    print(myList.isEmpty); // false  
    print(myList.isNotEmpty); // true  
    print(myList.length); // 6  
    print(myList.elementAt(2)); // two  
    print(myList[2]); // two  
    myList[3] = 3; // myList: [0, one, two, 3, four, five]  
    print(myList.getRange(1, 3).toList()); // [one, two]  
    print(myList.take(3).toList()); // [0, one, two]  
    myList.removeAt(0);  
    print(myList);  
    myList.removeRange(1, 4); //remove items from index 1 to 3  
    print(myList);  
    myList.clear(); //makes list empty  
    print(myList); //print nothing except [] empty bracket  
}
```

## Example 6/6

```
void main() {  
    var myList = [0, 2, 4, 6, 8, 2, 7];  
    print(myList.contains(2)); // return true  
    print(myList.contains(5)); // return false  
    print(myList.indexOf(2)); // return first position of value 1  
    print(myList.lastIndexOf(2)); // return last position of 5  
    //filter all items in List that match the condition | where()  
    print(myList.where((item) => item > 5).toList()); // [6, 8, 7]  
    //get the first item in List that matches the condition |  
    firstWhere()  
    print(myList.firstWhere((item) => item > 5)); // 6  
    // get the last item in List that matches the condition |  
    lastWhere()  
    print(myList.lastWhere((item) => item > 5)); // 7  
    var intList = [0, 5, 2, 3, 8, 17, 11];  
    intList.sort(); //sorting list in ascending order  
    print(intList);  
}
```

# Map

- Map in Dart is special type of variable which can also hold more than one value like list.
- in map each value has unique **key** and it is normally **string**.
- value can be accessed or changed using key.
- map can hold either mix value of fixed type value such as integer.
- map has built in methods which we can use to work upon it.
- There are three types of map, depending in the order of iteration:

1. HashMap is unordered. The key-value pair coming later could be ordered first.
2. LinkedHashMap has predictable iteration order by the insertion order. The key-value pair coming later will be ordered later.

By default, creating an instance using Map constructor (Map(), Map.from(), Map.of()) will return a LinkedHashMap.

1. SplayTreeMap iterates the keys in sorted order. It is a self-balancing binary tree, frequently accessed elements will be moved more closely to the root of the tree.

## Example 1/3

```
import 'dart:collection';

void main() {
    HashMap map1 = new HashMap<String, dynamic>();
    LinkedHashMap map2 = new LinkedHashMap<String, dynamic>();
    SplayTreeMap map3 = new SplayTreeMap <String, dynamic>();
    map1['name'] = "Ankit"; map1['age'] = 35;
    map1['weight'] = 75;      map1['gender'] = true;
    print(map1);
    map2['name'] = "Ankit"; map2['age'] = 35;
    map2['weight'] = 75;      map2['gender'] = true;
    print(map2);
    map3['name'] = "Ankit"; map3['age'] = 35;
    map3['weight'] = 75;      map3['gender'] = true;
    print(map3);
}
```



## Example 2/3

```
import 'dart:collection';  
void main() {  
    //2nd way to create map  
    Map<String, int> map1 = {'zero': 0, 'one': 1, 'two': 2};  
    print(map1);  
  
    // 3rd way to create map  
    Map map2 = {'zero': 0, 'I': 'one', 10: 'X'};  
    print(map2);  
  
    //4th way to create map  
    var map3 = {'zero': 0, 'I': 'one', 10: 'X'};  
    print(map3);  
  
    Map readOnly = Map.unmodifiable({1: 'one', 2: 'two'});  
    print(readOnly);  
}
```

## Example 3/3

```
import 'dart:collection';

void main() {
  Map map = {1: 'one', 2: 'two'};
  print(map.containsKey(1)); // true
  print(map.containsKey(3)); // false
  print(map.containsValue('two')); // true
  print(map.containsKey('three')); // false
  print(map.length); // return size of map
  print(map.isEmpty); // false
  print(map.isNotEmpty); // true
  print(map.keys); // (1, 2)
  print(map.values); // (one, two)
  print(map[2]); // two
  print(map[3]); // null
  map.remove(2);      print(map);
  map.clear();        print(map);
}
```

# Set

- Sets in Dart is a special type of unordered List where all the values are unique means it doesn't contain any repeated value at all.
- If we try to add duplicate value in set, it will ignore it.
- Set has most the methods we used with list.
- The sets are declared by the use of a **set** keyword.  
`var variable_name = <variable_type>{};`  
`set <variable_type> variable_name = {};`



# Example

```
// Dart program to show the Sets concept
void main()
{
    // Declaring set using 1st method
    var fruits = <String>{'apple', 'banana', 'pinapple', 'mango',
    'graps','apple'};
    print(fruits);
    // Declaring set using 2nd method
    Set<String> colors = {'red','green','blue','white','blue'};
    print(colors);
    var list1 = ['pink', 'yellow'];
    var list2 = ['brown', 'cyan', 'yellow', 'pink'];

    colors
    ..addAll(list1)
    ..addAll(list2); //will ignore duplicate value if any
    print(colors);
}
```

# Operators

- Operators are special symbols used to perform specific operation on variable or value.
- dart has many built-in operators which can be used to carry out different operations.
- There are two type of operators depending upon variable on which it works
- Unary operators
  - Works on single variable
- Binary operators
  - Works on two variable
- Following are 7 important type of operators
  1. Arithmetic Operators
  2. Relational Operators
  3. Type Test Operators
  4. Assignment Operators
  5. Logical Operators
  6. Conditional Operator
  7. Cascade Notation Operator



# 1. Arithmetic Operators:

Operator Symbol	Operator Name	Operator Description
+	Addition	Use to add two operands
–	Subtraction	Use to subtract two operands
-expr	Unary Minus	It is Use to reverse the sign of the expression
*	Multiply	Use to multiply two operands
/	Division	Use to divide two operands
~/	Division	Use two divide two operands but give output in integer
%	Modulus	Use to give remainder of two operands

```
void main(){  
    int a = 2,b = 3;  
    var c = a + b;  
    print("Sum of a and b is $c");  
    var d = a - b;  
    print("The difference between a and b is $d");  
    var e = -d;    // Using unary minus  
    print("The negation of difference between a and b is $e");  
    var f = a * b;  
    print("The product of a and b is $f");  
    var g = b / a;  
    print("The quotient of a and b is $g");  
    var h = b ~/ a;    // Using ~/ to divide a and b in nteger  
    print("The quotient of a and b is $h");  
    var i = b % a;  
    print("The remainder of a and b is $i");  
}
```

# Relational operators

operator Symbol	Operator Name	Operator Description
>	Greater than	Check which operand is bigger and give result as boolean expression.
<	Less than	Check which operand is smaller and give result as boolean expression.
>=	Greater than or equal to	Check which operand is greater or equal to each other and give result as boolean expression.
<=	less than equal to	Check which operand is less than or equal to each other and give result as boolean expression.
==	Equal to	Check whether the operand are equal to each other or not and give result as boolean expression.
!=	Not Equal to	Check whether the operand are not equal to each other or not and give result as boolean expression.



```
void main() {  
    int a = 20, b = 30;  
    var result = a > b;  
    print("a is greater than b is $result");  
    result = a < b;  
    print("a is smaller than b is $result");  
    result = a >= b;  
    print("a is greater than b is $result");  
    result = a <= b;  
    print("a is smaller than b is $result");  
    result = b == a;  
    print("a and b are equal is $result");  
    result = b != a;  
    print("a and b are not equal is result");  
}
```

# Type Test Operators

- Type Test operator is used to check data type of variable.

Operator Symbol	Operator Name	Operator Description
is	is	Return true as output if the object has specific type
is!	is not	Return false as output if the object has specific type

```
void main()
{
    var value = '100';
    print(value is int);
    print(value is String);
    print(value is! Double);
    print(value is! Bool);
}
```

## 5. Assignment Operators

- Assignment operators are used to assign value to variable.

Operator Symbol	Operator Name	Operator Description
=	equal to	Use to assign values to the expression or variable
??=	assignment operator	if this object is null, then assign it to this value. If it's not, just return the object as is":

```
void main() {  
    int a = 50,b = 70;  
    var result = null;  
    result = a + b; // Value is assign as result is null  
    print(result);  
    var result2;  
    result2 ??= a + b; // Value is assign as result2 is null  
    print(result2);  
    result2 ??= a - b; // Value is not assign as it is not null  
    print(result2);  
}
```

# Logical operators

- Logical operators are used to combine two or more conditions.
- It is compulsory to use logical operators when there are more than one condition in decision making statement or in loop.

&&	And Operator	Return true than it both condition is true.
	Or Operator	Return true than it both any one or both condition is true.
!	Not Operator	It is use to reverse the result.

```
void main(){
    int a = 50;
    int b = 70;
    bool c = a > 10 && b < 100;
    print(c);
    bool d = a > 10 || b < 10;
    print(d);
    bool e = !(a > 10);
    print(e);
}
```

# Ternary Conditional operator

Operator Symbol	Operator Name	Operator Description
condition ? expersion1 : expersion2	Conditional Operator	If the condition is true than expersion1 will execute else expersion2 will execute. It is a simple version of if-else statement.
expersion1 ?? expersion2	Conditional Operator	If expersion1 is non-null returns its value else returns expersion2 value.

```
void main() {  
    bool isRegistered = false;  
    String message = (isRegistered == true)  
        ? 'Welcome back to our site!'  
        : 'Welcome, please sign up.';  
    print(message);  
    var ICanBeNull=null;  
    var IAlwaysHaveValue = "india"  
    var WhatDoIHave = ICanBeNull ?? IAlwaysHaveValue  
    print(WhatDoIHave);  
}
```

# Cascade Notation Operators

- Cascades (..) allow you to perform a sequence of operations on the same object.
- The Cascades notation(..) is similar to method chaining that saves you number of steps and need of temporary variable.

```
class MyMath {  
    var a,b;  
    void set(x, y){  
        this.a = x;  
        this.b = y;  
    }  
    void add(){  
        var result = this.a + this.b;  
        print("result of addition = $result");  
    }  
}  
  
void main(){  
    MyMath m1 = new MyMath();  
    MyMath m2 = new MyMath();  
    m1.set(10, 20);  
    m1.add();  
    m2..set(30, 40)..add();  
}
```

## Enumeration in dart

- An enumeration is a set of predefined named values, called as members.
- Enumerations are useful when we want deal with limited set of values for variable.
- For example you can think of the colors of a traffic light can only be one of three colors– red, yellow or green.

```
enum signal { red, yellow, green }  
void main() {  
    print(signal.red.index); //0  
    print(signal.yellow.index); //1  
    print(signal.green.index); //2  
    signal color;  
    color = signal.red;  
    print(color);  
    //color = 5; //won't work  
}
```

# Decision making statement in dart

- Dart provides following types of Decision-making statement.
  1. If Statement
  2. If-else Statements
  3. If else if Statement
  4. Switch Case Statement





# If Statement

- The **if** construct evaluates a condition if condition is true then it will execute block of code given in next **{}** pair.
- Following is the syntax.

```
if(condition){  
    statement....  
    statement....  
    statement....  
}
```

```
import 'dart:io';  
  
void main() {  
    print("enter number");  
    int num = int.parse(stdin.readLineSync().toString());  
    if (num > 0) {  
        print("number is positive");  
    }  
}
```

# If else Statement

- If else block evaluates conditions, if condition true, then the code block given in {} pair next to if will be executed, otherwise code block given {} pair after else will be executed.

Following is the syntax.

```
if(condition){  
    statement....  
} else {  
    statement....  
}
```

```
import 'dart:io';  
void main() {  
    print("enter number");  
    int num = int.parse(stdin.readLineSync().toString());  
    if (num > 0) {  
        print("number is positive");  
    } else {  
        print("number is negative");  
    }  
}
```

# If else if ladder Statement

- if else if ladder is used where we need to check multiple conditions one after another in way that if first conditions is true then code block given in {} braces pair will execute and if condition is false then only second conditions will be checked and so on. if all conditions are false then else block will run
- Following is the syntax.

```
if (conditions) {  
    //statements  
}  
else if (conditions) {  
    //statements  
}  
else if(conditions)  
{  
    //statements  
}  
else {  
    //statements  
}
```



# Example

```
import 'dart:io';  
void main() {  
    print("enter number");  
    int num = int.parse(stdin.readLineSync().toString());  
    if(num > 0) {  
        print("${num} is positive");  
    }  
    else if(num < 0) {  
        print("${num} is negative");  
    } else {  
        print("${num} is neither positive nor negative");  
    }  
}
```

# Switch statement

- In case, where there are more conditions to be checked, it is better and more convenient to use the switch statements, for example, the operations on a calculator.
- 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.
- The 'break' keyword is used to stop the checks. When the program execution reaches the 'break', it jumps out of the curly braces of the switch statement to the next line after the curly braces
- switch is the simplified form of nested if-else statement.

```
switch(conditions) {  
    case value-1:  
        // statements;  
        break;  
    case value-2:  
        //statements;  
        break;  
    default:  
        //statements;  
        break;  
}
```



# Example

```
void main() {  
    print("enter day");  
    int day = int.parse(stdin.readLineSync().toString());  
    switch (day) {  
        case 1:  
            print("monday");  
            break;  
        case 2:  
            print("Tuesday");  
            break;  
        case 3:  
            print("Wednesday");  
            break;  
        case 4:  
            print("Thursday");  
            break;  
        case 5:  
            print("Friday");  
            break;  
        case 6:  
            print("Saturday");  
            break;  
        case 7:  
            print("Sunday");  
            break;  
        default:  
            print("invalid input");  
            break;  
    }  
}
```

# loop

- A loop in a computer program is an code block that repeats till a specified condition is true.
- A computer programmer who needs to use the same lines of code many times in a program can use a loop to save time & reduce code.
- Dart supports the following type of loops.
  - Dart for loop
  - Dart for...in loop
  - Dart while loop
  - Dart do-while loop



# While loop

- In while loop first we check some condition, if it is true then we execute block of code given in {} braces pair next to while loop.
- Again condition is checked if it is again true then code given in {} braces pair execute again.
- This will continue until condition is evaluated as false.
- While loop is one type of entry control loop.
- It has following syntax.

```
while(condition){  
    // Body of loop  
}
```





# example

```
import "dart:io";  
void main()  
{  
  int i = 1;  
  while (i <= 10) {  
    stdout.write(' $i '); //stdout.write printout on same line  
    i++;  
  }  
}
```

# Do While loop

- In Do while loop body of loop {} will execute first and then condition is checked.
- if condition is true then then code given in {} braces pair execute again.
- This will continue until condition is evaluated as false.
- Do While loop is one type of exit control loop. It always execute at least once.
- It has following syntax.

```
do {  
    // Body of loop  
}while(condition);
```



# example

```
import "dart:io";  
void main() {  
    int i = 1;  
    do {  
        stdout.write(' $i '); //stdout.write printout on same line  
        i++;  
    } while (i <= 10);  
}
```

# For loop

- For loop has three part initialization, condition, increment/decrement.
- When for loop run first time, initialization is done and then condition is checked, if it is true then loop body will execute
- In subsequent execution of loop, first initialization is done and then conditions is checked and so on.
- For loop is entry control loop. It has following syntax.

```
for(initialization; condition; increment/decrement)
{
    // Body of the loop
}
```



# example

```
import "dart:io";  
void main()  
{  
  for (int i = 1; i <= 10; i=i+1) {  
    stdout.write(' $i ');  
  }  
}
```

# For in loop

- The for...in loop is used to loop through an object's properties/list/map values.
- In each execution, one property/value from the object/list is assigned to the variable name and this loop continues till all the properties/value of the object/list are exhausted.
- It has following syntax.

```
for (var in expression) {  
    // Body of loop  
}
```



# example

```
import "dart:io";

void main() {
  var list = ['one', 2, 3.14, true];
  for (var value in list) {
    stdout.write(" $value ");
  }

  print("reversed for loop");
  for (var value in list.reversed) {
    stdout.write(" $value ");
  }
}
```