**Flutter Basic Introduction and Setup**

**Introduction:**

Flutter is an open source framework to create high quality, high performance mobile applications across mobile operating systems - Android and iOS. It provides a simple, powerful, efficient and easy to understand SDK to write mobile application in Google’s own language, *Dart*.

Flutter – a simple and high performance framework based on Dart language, provides high performance by rendering the UI directly in the operating system’s canvas rather than through native framework. Flutter also offers many ready to use widgets (UI) to create a modern application. These widgets are optimized for mobile environment and designing the application using widgets is as simple as designing HTML.

Flutter application is itself a widget. Flutter widgets also supports animations and gestures. The application logic is based on reactive programming. Widget may optionally have a state. By changing the state of the widget, Flutter will automatically (reactive programming) compare the widget’s state (old and new) and render the widget with only the necessary changes instead of re-rendering the whole widget.

**2 Dart Programming Introduction Flutter with Dart Programming:**

Dart is an open-source general-purpose programming language. It is originally developed by Google. Dart is an object-oriented language with C-style syntax. 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.

The following code shows a simple Dart program:

void main()

{

  print("This is Dart programming for flutter");

}

For run this code write a command in terminal

d**art filename.dart**

**Dart Comments :**

Comments are statement , useful for user but not for compiler. It’s a message for understanding our code for us and other programmer.

Single line comment will be shown as //.

Multi-Line Comments in Flutter: A Deeper DiveMulti-line comments, or block comments, span across multiple lines. They start with /\* and end with \*/ . These comments are ideal for detailed explanations or temporarily disabling chunks of code.

**Dart Datatypes** Variables and Data types

*Variable* is named storage location and *Data types* simply refers to the type and size of data associated with variables and functions.

#### Dart uses *var* keyword to declare the variable. The syntax of *var* is defined below,

var name = 'Dart';

#### The *final* and *const* keyword are used to declare constants. They are defined as below:

Dart language supports the following data types:

* **Numbers**: It is used to represent numeric literals – Integer and Double.
* **Strings**: It represents a sequence of characters. String values are specified in either single or double quotes.
* **Booleans**: Dart uses the *bool* keyword to represent Boolean values – true and false.
* **Lists and Maps**: It is used to represent a collection of objects. A simple List can be defined as below:

* + **Dynamic**: If the variable type is not defined, then its default type is *dynamic*. The following example illustrates the dynamic type variable:

## Code for Datatypes:

## for run code:

## we need to open terminal in vscode.

## Make sure you are in right path.

## Write command **dart filename.dart**

## If there is no error code will run.

## Example 1 : all kinds of datatypes

//import 'dart:io';

void main() {

  print("Dart language is easy to learn");

  var a = 7;  // when we pass var as datatypes, it will directly consider it as define type data

  print("A: $a");

  int? a1;  // for null value

  a1 =null;

  print(a1);

  final no = 12;  // this is also considered as constant value

  print(no);

  const pi = 3.14;

  print(pi);

  var name = 'Jasmin';

  print("name: $name");

  int num = 4;

  print("num: $num");

  double num1 = 87.34;

  print("num1: $num1");

  var list = [1,2,3,4,5];

  print(list);

  String fname = "Rahi";

  print("fname: $fname");

  bool flag = true;

  print("flag: $flag");

var mapping = {'id': 1,'name':'Dart'};

  print("mapping: $mapping");

  dynamic lname = "Dart";  //variable type is not defined, then its default type is dynamic.

  print("lname: $lname");

  }

## Example 2: user side input in dart.

///method readLineSync() is one of the methods used to take input from the user.

// importing dart:io file

import 'dart:io';

void main()

{

  // integer value

  print("Enter your favourite number:");

  int? n = int.parse(stdin.readLineSync()!);

  print("Your favourite number is $n");

// String value

  print("Enter your name?");

  // Reading name

  String? name = stdin.readLineSync();

  print("Hello, $name ! \nWelcome to Dart Programming!!");

///The print() statement brings the cursor to next line while

///stdout.write() don’t bring the cursor to the next line, it remains in the same line.

//  If the print statements are switched in the above program then:

  stdout.write("Welcome to dart programming! // printing from stdout.write()   \t");

  print("Welcome to dart programming! // printing from print statement");

}

**basic of numbers in dart** Number function provided by dart library.

void main()

{

  //abs() : to find absolute value

  var no1 = -99;

  var p = no1.abs();

  print("abs(): $p");

  // ceil() : to round off value.. update to nearest big.

 var no = 34.45;

  p = no.ceil();

  print("ceil(): $p");

//floor() : to round off value.. update to nearest small.

  no = 34.45;

  p = no.floor();

  print("floor(): $p");

// remainder()  : find out remainder

  no1 = 30;

  p = no1.remainder(4);

  print("remainder(): $p");

  //round() : round off to nearest one as per maths

  no = 20.399;

  p = no.round();

  print("round(): $p");

  //truncate() : discard after point value

  no = 20.99;

  p = no.truncate();

  print("truncate(): $p");

// toString()

  no1 = 25;

  var str = no1.toString();

  print("toString(): $str");

print(str is String);

// toInt()

  no = 20.99;

  p = no.toInt();

  print("toInt(): $no");

  print(p is int);

//toDouble()

  no = 20;

  var p1 = no.toDouble();

  print("toDouble(): $p1");

  print(p1 is double);

  //compareTo()  ans in 0 = equal , -1 = first one is small,

  // +1 = first one is big

  var num=25;

  var ans = num.compareTo(25);

  print(ans);

  var n=23;

  ans = n.compareTo(12);

  print(ans);

  n=23;

  ans = n.compareTo(52);

  print(ans);

  }

## numbers properties.

void main()

{

  // parse()  : convert numeric string to number formate

  var no1 = num.parse("25");

  var no2 = num.parse("75");

  var c = no1+no2;

  print("Sum is : $c");

  // isEven  or  isOdd  : ans is in true/false

  var no3 = 25;

  var ans = no3.isEven;

  var ans1 = no3.isOdd;

  print("isEven: $ans");

  print("isOdd: $ans1");

  // isFinite  or isInfinite: ans is in true/false

  ans=no1.isFinite;

  ans1=no1.isInfinite;

  print("isFinite: $ans");

  print("isInfinite: $ans1");

  // isNegative

  no1=-78;

  no2=45;

  ans = no1.isNegative;

  ans1= no2.isNegative;

  print("isNegative: $ans");

  print("isNegative: $ans1");

}

## **Strings in dart**

## String: The String data type represents a sequence of characters.

## • Syntax:-

## String variable\_name = ‘value’ or

## String variable\_name = “value” or

## String variable\_name = ```line1 line2``` or

## String variable\_name = ``````line1 line2``````

## Some methods of string:

## isEmpty & isNotEmpty : Return true if this string is empty

## Length: Return the length of the String including space , tab and newline characters.

## toUpperCase(): Converts All characters in this string to upper case.

## toLowerCase() : Converts all characters in this string to lower case.

## trim(): Returns the string without any leading and trailing whitespace. However, this method doesn’t discard spaces between two strings.

## indexOf(), lastIndexof(): Returns the position of the first and last matches of the given pattern:

## split(): Splits the string at the matching pattern, returning a list of substrings

## replaceAll(): Replaces all substrings that match the specified pattern with the replacement string:

## replaceFirst(): Replace First Will Replace First Character Only.

## replaceRange(): Replace Range will Replace range Word with new One.

## endsWith(): Checks whether the string ends with the specified characters.

## startsWith(): Checks whether the string starts with the specified characters.

## indexOf(): indexOf() will return index of character.

## Substring(): Get SubString from String

## Contains():Contains will check String/Character is present or not.

## CompareTo(): Compares this object to another.

## • Returns an integer representing the relationship between two strings.

## • 0− when the strings are equal.

## • 1− when the first string is greater than the second

## •-1− when the first string is smaller than the second

## toString(): Returns a string representation of this object.

void main()

{

  String str1 = "This is  first method..";

  String str2 ="This is  second method..";

  String str3 ='''This is

                  second method..''';

  String str4 ="""This is

              fourth method..""";

  print(str1);

  print(str2);

  print(str3);

  print(str4);

  // isEmpty & inNotEmpty

  print(str1.isEmpty);

  print(str3.isNotEmpty);

  //length

  print(str1.length);

  // toUpperCase() & toLOwerCase()

  print(str1.toUpperCase());

  print(str1.toLowerCase());

  //trim()

  str1 = "    Hello, hi   ";

  print(str1);

print(str1.trim());

  print(str1.trimLeft());

  print(str1.trimRight());

  // indexof,   lastindexof

  print(str1.indexOf('el'));

  print(str1.lastIndexOf('or'));

  print(str1.indexOf('h'));

  print(str1.indexOf('world'));

  //split()

  str1= "hello Dart";

  print(str1.split("\n"));

  print(str1.split(" "));

  print(str1.split("o"));

//replaceAll(), replaceFirst(),replaceRange()

  print(str1.replaceAll('o','e'));

  print(str1.replaceFirst('o','e'));

  print(str1.replaceRange(0,3,'welcome'));

  //endsWith()

  print(str1.endsWith('d'));

  print(str1.endsWith('rl'));

  print(str1.endsWith('ld'));

  print(str1.endsWith('world'));

  // startWith

  print(str1.startsWith('h'));

  print(str1.startsWith('H'));

  print(str1.startsWith('e'));

  print(str1.startsWith('Hello'));

  // substring

  print(str1.substring(3));

  print(str1.substring(0,5));

  print(str1.substring(6,8));

  print(str1.substring(6));

  // contains

  print(str1.contains('o'));

  print(str1.contains('e'));

  print(str1.contains('he'));

  print(str1.contains('Wo'));

// Join / concatenate

String s1 = 'hello';

String s2 = 'World';

String s3 = '$s1$s2'; //interpolation

String s4 = '$s1' '$s2'; // separated in one statement

String s5 = s1+s2; // operator

String s6 =  s1 \* 3; // concatenate multiply

print(s3);

print(s4);

print(s5);

print(s6);

// compareTo()

var a = 'Hello';

var b = 'World';

var c = 'Hello';

print(a.compareTo(b));

print(a.compareTo(c));

print(b.compareTo(a));

//toString

int no = 101;

var no1 = no.toString();

print(no1);

print(b is String);

}

# Dart Collection

## Dart doesn't support the array to store the data, unlike the other programming language. We can use the Dart collection in place of array data structure. We can enable the other classes of the collection in our Dart script by using the dart::core library. Ex List, Set,Map.

## **List in dart** List: Dart represents arrays in the form of List objects.

## • A List is simply an ordered group of objects.

## • The dart:core library provides the List class that enables creation and manipulation of lists.

## some methods

## Single: Checks if the list has only one element and returns it.

## first : Returns the first element case.

## isEmpty: Returns true if the collection has no elements.

## isNotEmpty: Returns true if the collection has at least one element.

## length: Returns the size of the list.

## last: Returns the last element in the list.

## reversed: Returns an iterable object containing the lists values in the reverse order.

## Example 4: list example

void main()

{

  var mylist = [10];

  print(mylist.single);

  mylist = [10,20,30];

  print("List: $mylist");

  print(mylist.first);

  print(mylist.length);

  mylist.add(50);

  print(mylist.isEmpty);

  print(mylist.isNotEmpty);

  print(mylist.last);

  print(mylist.reversed);

}

## Adding Elements into a List

## There are different methods to add elements into a list.

## add():  The List.add() function appends the specified value to the end of the List and returns a  modified List object

## addAll() :  The List.addAll() function accepts  multiple values separated by a  comma and appends these to the  List.

## insert() :  The insert() function accepts a value and inserts it at the specified index.

## Syntax :- List.insert(index,value)

## insertAll() : The insertAll()function inserts the given list of values,   beginning from the index specified.

## Syntax :-

## • List.insertAll(index, iterable\_list\_of\_values)

## Updating the index : Dart allows modifying the value of an item in a List using index,

## List.replaceRange(): This function replaces the value of the elements within the specified range.

## Syntax:-List.replaceRange(intstart\_index,intend\_index,Iterable<items>)

## Example 5: list function

void main()

{

   // ADDING ELEMENT TO EMPTY LIST

  var fruits = [];

  fruits.add('Banana');

  fruits.add('Apple');

  print(fruits);

  fruits.addAll(['Guava, Grapes']);

  print(fruits);

  fruits.insert(2, 'Cabbage');

  print(fruits);

  fruits.insertAll(2,['papaya','orange']);

  print(fruits);

  fruits[0]= 'brinjal';

  print(fruits);

  print("List Before replacing:  $fruits");

  fruits.replaceRange(1,4,['potato','tomato','cucumber']);

  print("List After replacing:  $fruits");

}

## Removing List Items

## There are different methods to remove an element from the list.

## List.remove() :function removes the first occurrence of the specified item in the list. This function returns true if the specified value is removed from the list.

## Syntax :- List.remove(value)

## List.removeAt():The List.removeAt function removes the value at the specified index and returns it.

## Syntax :- List.removeAt(intindex)

## List.removeLast() :The List.removeLast() function pops and returns the last item in the List.

## Syntax :- List.removeLast()

## List.removeRange() : The List.removeRange() function removes the items within the specified range.

## Syntax :- List.removeRange(int start, intend)

void main()

{

  var list = [10,20,50,30,60,70];

  print("list before removing : $list");

  bool result = list.remove(20);

  print("list after removing : $list");

dynamic res = list.removeAt(1);

  print("removed elements : $res");

print("list after removing : $list");

  res = list.removeLast();

  print("removed elements : $res");

  print("list after removing : $list");

list.removeRange(1,3);

  print("list after removing : $list");

}

## **Map in dart and Set in Dart** Set : A set in Dart is an unordered collection of unique items. Dart support for sets is provided by set literals and the Set type.

## var halogens = {'fluorine', 'chlorine', 'bromine', 'iodine', 'astatine'};

## Example 7: example on set

void main()

{

  // Declaring set in First Way

  var str = <String>{'Hello Dart!!!'};

    print("Output of first set:   $str");

  // Declaring set in Second Way

  Set<String> str1 = {'Do you like Dart?'};

 print("Output of second set:   $str1");

  var str3 = ['hello','Banana','Hello'];

  // list will aloowed repeating value

  print("Output of the list is: $str3");

  // Declaring set with repeated value

  var str4 = {'hello','Banana','hello'};

  print("Output of the set is: $str4");

}

## Example 8: function on set

void main()

{

  // Declaring set with value

  var str = <String>{'Hello Dart'};

  // Printing Set

  print("Value in the set is: $str");

  // Adding an element in the set

  str.add("DartProgramming");

  // Printing Set

  print("Values in the set is: $str");

  // Adding multiple values to the set

  var str\_var = {"Dart","Dart1","Dart2"};

  str.addAll(str\_var);

  // Printing Set

  print("Values in the set is: $str");

  // Getting element at Index 0

  var str1 = str.elementAt(0);

  // Printing the element

  print("Element at index 0 is: $str1");

  // Counting the length of the set

  int l = str.length;

  // Printing length

  print("Length of the set is: $l");

  // Finding the element in the set

  bool check = str.contains("DartProgramming");

  // Printing boolean value

  print("The value of check is: $check");

  // Removing an element from the set

  str.remove("Hello Dart");

  // Printing Set

  print("Values in the set is: $str");

  // Using forEach in set

  print(" ");

  print("Using forEach in set:");

  str.forEach((element) {

    if(element == "Hello")

    {

      print("Found");

    }

    else

    {

      print("Not Found");

    }

  });

  // Deleting elements from set

  str.clear();

  // Printing set

  print("Values in the set is: $str");

}

## Map : a map is an object that associates keys and values. Both keys and values can be any type of object. Each key occurs only once, but you can use the same value multiple times. Dart support for maps is provided by map literals and the [Map](https://api.dart.dev/stable/dart-core/Map-class.html) type.

## Example 9: example on map

//Syntax: Creating the Map using Map Literals

//var map\_name = { key1 : value1, key2 : value2, ..., key n : value n }

void main()

{

  // Creating Map using is literals

  var str = {'position1' : 'bike', 'position2' : 'car', 'position3' : 'cycle'};

  // Printing Its content

  print(str);

  print(str['position1']);

  print(str[0]);

//  multiple value in same key.

  str = {'position1' : 'bike' ' is' ' good.'};

  // Printing Its content

  print(str);

  print(str['position1']);

}

## Example 10: map using constructor:

void main() {

    // Creating Map using Constructors

    var str = new Map();

    // Inserting values into Map

    str [0] = 'banana';

    str [1] = 'apple';

    str [2] = 'orange';

    // Printing Its content

    print(str);

    // Printing Specific Content Key is defined

    print(str[0]);

    print("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

var str1=new Map();

    str1 [0] = 'banana';

    str1 [0] = 'apple';

    str1 [0] = 'orange';

    print(str1);

    print(str1[0]);

}

## Example 11: Queue in dart

// With type notation(E)

//Queue<E> variable\_name = new Queue<E>.from(list\_name);

// Without type notation

//var variable\_name = new Queue.from(list\_name);

import 'dart:collection';  // library must be added

void main()

{

    // Creating a queue through a constructor and

    // then inserting the elements in it.

    Queue<String> str = new Queue<String>();

    print(str);

    str.add("strs");

    str.add("For");

    str.add("strs");

    print(str);

  //Creating a queue through a list

  List<String> str\_list = ["strs","For","strs"];

  Queue<String> str\_queue = new Queue<String>.from(str\_list);

  print(str\_queue);

  // function in queue

  Queue<String> str1 = new Queue<String>();

  print(str1);

  str1.add("strs");

  print(str1);

  List<String> str\_data = ["For","strs"];

  //Adding a element and displaying it

  str1.addAll(str\_data);

  print(str1);

  //Deleting all the data from queue

  str1.clear();

  print(str1);

  // Checking if the queue is empty or not

  print(str1.isEmpty);

  // Adding first element

  str1.addFirst("strs");

  print(str1);

  //Adding Last element

  str1.addLast("For");

  str1.addLast("strs");

  print(str1);

  // Checking length of the queue

  print(str1.length);

  // Removing First Element from queue

  str1.removeFirst();

  print(str1);

  // Removing Last element from queue

  str1.removeLast();

  print(str1);

  // Displaying all the elements of the queue

  str1.forEach(print);

}

## **enumerations** or enums, are a special kind of class used to represent a fixed number of constant values.

## Example 12: enumeration

/\* Syntax:

enum variable\_name{

  // Insert the data members as shown

  member1, member2, member3, ...., memberN

} \*/

// dart program to print all the

// elements from the enum data class

enum Str {

      // Inserting data

      Welcome, to, strsandstrs

}

void main() {

  for (Str str in Str.values) {

    print(str);

  }

}

enum Str {

Welcome, to, strsForstrs

}

void main() {

/ Assign a value from

// enum to a variable str

var str = Str.strsForstrs;

// Switch-case

switch(str) {

  case Str.Welcome: print("This is not the correct case.");

  break;

  case Str.to: print("This is not the correct case.");

  break;

  case Str.strsForstrs: print("This is the correct case.");

  break;

}

}

**Operators and Control Flow in Dart Operators**

## Operator: Operator is a special character which can perform a specific task.

## There are various operators in Dart.

## Arithmetic operators: they can perform a mathematical operation.

// for mathematical operation

void main()

{

  double num1=45;

  double num2=60;

  dynamic c = num1+num2;

  print("Addition: $c");

  c = num1-num2;

  print("Subtraction: $c");

  c = -num1;

  print("unary minus: $c");

  c = num1\*num2;

  print("Multipilcation: $c");

  c = num1/num2;

  print("Division: $c");

  c = num1~/num2; //Use to divide two operands but give output in integer(returns quotient)

  print("Quotient with only int: $c");

  c = num1%num2;  //Use to give remainder of two operands(returns remainder)

   print("Modulus: $c");

}

## Relational operators:

/\*Relational Operators

This class of operators contain those operators which are used to perform relational operation on the operands. \*/

void main()

{

   int a = 4;

   int b = 5;

   bool c = a==b;

   print(" a == b: $c");

   c= a!=b;

   print(" a != b: $c");

   c= a<=b;

   print(" a <= b: $c");

   c= a>=b;

   print(" a >= b: $c");

   c= a<b;

   print(" a < b: $c");

   c= a>b;

   print(" a > b: $c");

}

## [Type Test Operators](https://www.geeksforgeeks.org/operators-in-dart/?ref=lbp" \l "3-type-test-operators):

//This class of operators contain those operators which are used to perform comparison on the operands.//

void main()

{

    String a = 'Jay';

    double b = 3.3;

//is  Gives boolean value true as output if the object has specific type

    print(a is String);

    // Using is! to compare

print(b is !int);//is!  is not  Gives boolean value false as output if the object has specific type

// as Operator

/// as Operator is used for Typecasting. It performs a cast at runtime if the cast is valid else, it    throws an error. It is of two types Downcasting and Type Assertion.

    dynamic value = "Hello";

  // TypeCast dynamic -> String

    String str= value as String;

    print("str : $str");

}

## [Bitwise Operators](https://www.geeksforgeeks.org/operators-in-dart/?ref=lbp" \l "4-bitwise-operators):

// Dart Program to Demonstrate Use of Dart Bitwise Operators which work on binary bits

void main()

{

    print("Demonstrate use of Dart Bitwise Operators");

    int a = 5;

    int b = 7;

    // Performing Bitwise AND on a and b

    var c = a & b;

    print("a & b : $c");

    // Performing Bitwise OR on a and b

    c = a | b;

    print("a | b : $c");

    // Performing Bitwise XOR on a and b

    c = a ^ b;

    print("a ^ b : $c");

    // Performing Bitwise NOT on a

    c = ~a;

    print("~a : $c");

    // Performing left shift on a

    c = a << b;

    print("a << b : $c");

    // Performing right shift on a

    c = a >> b;

    print("a >> b : $c");

    c = -a >>> b; // unsiged right shift Shifts a in binary representation to b bits to left (it ignores sign).

    print("-a >>> b : $c");

    /\* it works on 64 bit bits

      this one is shown as 32 bit compiler

      for 5 no. will be 0101

      but for negative (-5)

      1111 1111 1111 1111 1111 1111 1111 1111 0101

      for shifting 7 bits it shift 7 bits towards right.

      0000 0001 1111 1111 1111 1111 1111 1111 1111

      same for 64 bits

      and this will show you  144115188075855871

    \*/

}

## [Assignment Operators](https://www.geeksforgeeks.org/operators-in-dart/?ref=lbp" \l "5-assignment-operators):

// Assignment Operators

///   = Equal to  Use to assign values to the expression or variable

///   ??= Assignment operator Assign the value only if it is null.

void main()

{

    dynamic a = 9;

    dynamic b = 12;

    var c = a \* b;

    print("assignment  operator used c = a\*b so now c = $c\n");

    var d; // Assigning value to variable d

    c??=a;

    print("Assigning value only if c is null : $c");

    d ??= a + b;// Value is assign as it is null

    print("Assigning value only if d is null");

    print("d??= a+b so d = $d \n");

    // Again trying to assign value to d

    d ??= a - b;

    print("Assigning value only if d is null");

    print("d??= a-b so d = $d");

    print("As d was not null value was not updated");

    ///Apart from there is another way where we can use a operator that is compound assignment operator where we combine an operator with an assignment operatorso to shorten the steps and make code more effective.

    a+=b;

    print("a+=b:  $a");

    a-=b;

    print("a-=b:  $a");

    a\*=b;

    print("a\*=b:  $a");

    a/=b;

    print("a/=b:  $a");

    a~/=b;

    print("a!/=b:  $a");

    a%=b;

    print("a%=b:  $a");

}

## [Logical Operators](https://www.geeksforgeeks.org/operators-in-dart/?ref=lbp" \l "6-logical-operators):

## 

///Logical Operators

///This class of operators contain those operators which are used to logically combine two or more conditions of the operands.

/\*  && : And Operator : Use to add two conditions and if both are true than it will return true.

    || : Or Operator :  Use to add two conditions and if even one of them is true than it will return   true.

    ! : Not Operator :  It is use to reverse the result. \*/

  void main()

{

    int a = 5;

    int b = 7;

    // Using And Operator

    bool c = a > 10 && b < 10;

    print("a > 10 && b < 10: $c");

    // Using Or Operator

    c = a > 10 || b < 10;

    print("a > 10 || b < 10: $c");

    // Using Not Operator

    c = !(a > 10);

    print("!(a > 10): $c");

}

## [increment / Decrement Operators](https://www.geeksforgeeks.org/operators-in-dart/?ref=lbp" \l "7-conditional-operators):

void main()

{

  int x=3;

  var y=x++;

  print("x++ : $y");

  y=++x;

  print("++x : $y");

  y=x--;

  print("x-- : $y");

  y=--x;

  print("--x : $y");

}

## [Conditional Operators:](https://www.geeksforgeeks.org/operators-in-dart/?ref=lbp" \l "8-cascade-notation-operators)

//Conditional Operators

//This class of operators contain those operators which are used to perform comparison on the operands.

void main()

{

  dynamic a = 7;

  dynamic b = 45;

  dynamic max = (a>b)? a :b;

  print(" max value is : $max");

}

## [Cascade Notation Operators:](https://www.geeksforgeeks.org/operators-in-dart/?ref=lbp" \l "8-cascade-notation-operators)

// Dart Program to Demonstrate Use of Cascading Operator

///.. Cascading Method  It is used to perform multiple methods on the same object.

/// ..?   Null Shorting Cascade

/// It is used when we are sure that the none of the cascade operations are done on Null elements.

// Class

class Add {

    dynamic a;

    dynamic b;

    void set(x, y)

    {

        this.a = x;

        this.b = y;

    }

    void add()

    {

        var z = this.a! + this.b;

        print("Addition is : $z");

    }

}

void main()

{

    // Creating objects of class GFG

    Add add = new Add();

    Add add1 = new Add();

    // Without using Cascade Notation

    add.set(1, 2);

    add.add();

    // Using Cascade Notation

    add1..set(3, 4)

         ..add();

}

**4.2 If Statements Decision Making :**

A decision making block evaluates a condition before the instructions are executed. Dart supports If, If..else and switch statements. Loops are used to repeat a block of code until a specific condition is met. Dart supports for, for..in , while and do..while loops. Let us understand a simple example about the usage of control statements and loops:

Example on control statement

1. If

import 'dart:io';

void main()

{

  print("Enter number:");

  int? n = int.parse(stdin.readLineSync()!);

  if(n>0)

  {

    print("$n is positive no");

  }

}

1. If else

import 'dart:io';

void main()

{

  print("Enter number:");

  int? n = int.parse(stdin.readLineSync()!);

  if(n>0)

  {

    print("$n is Positive no");

  }

  else

  {

    print("$n is Negative no");

  }

  print("\_\_\_\_\_\_\_\_\_\_Even Odd code\_\_\_\_\_\_\_\_\_\_");

  print("Enter number:");

  int? num = int.parse(stdin.readLineSync()!);

  if(num%2==0)

  {

    print("$num is Even no.");

  }

  else

  {

    print("$num is Odd no.");

  }

}

1. **If else-if**

import 'dart:io';

void main()

{

  print("Enter number:");

  int? n = int.parse(stdin.readLineSync()!);

  if(n>0)

  {

    print("$n is positive no");

  }

  else if(n==0)

  {

    print("$n is Zero");

  }

  else

  {

    print("$n is negative no");

  }

  print("\_\_\_\_\_\_\_\_\_\_Profit or Loss code\_\_\_\_\_\_\_\_\_\_");

  print("Enter sp:");

  int? sp = int.parse(stdin.readLineSync()!);

  print("Enter cp:");

  int? cp = int.parse(stdin.readLineSync()!);

  if(sp>cp)

  {

    var profit = sp-cp;

    print("Profit of Rupees $profit");

  }

  else if(cp>sp)

  {

    var loss = cp-sp;

    print("Profit of Rupees $loss");

  }

  else

  {

    print("No Profit No Loss");

  }

}

1. Ladder if

import 'dart:io';

void main()

{

  print("Enter 5 sub marks:");

  int? html = int.parse(stdin.readLineSync()!);

  int? c = int.parse(stdin.readLineSync()!);

  int? cpp = int.parse(stdin.readLineSync()!);

  int? python = int.parse(stdin.readLineSync()!);

  int? java = int.parse(stdin.readLineSync()!);

  int total = html+c+cpp+python+java;

  print(" HTML  : $html \n C     : $c \n CPP   : $cpp \n PYTHON: $python \n JAVA  : $java");

  print("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

  print("Total: $total");

  double per = (total\*100)/500;

  print("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");;

  print("Percentage: $per");

  print("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

  if(per<=100 && per>=90)

  {

    print("Grade A");

  }

  else if(per<90 && per>=80)

  {

    print("Grade B");

  }

  else if(per<80 && per>=60)

  {

    print("Grade C");

  }

  else if(per<60 && per>=50)

  {

    print("Grade D");

  }

  else if(per<50 && per>=35)

  {

    print("Grade E");

  }

  else

  {

    print("FAIL");

  }

}

1. Nested if

import 'dart:io';

void main()

{

    print("Enter 4 no.");

    int? num1 = int.parse(stdin.readLineSync()!);

    int? num2 = int.parse(stdin.readLineSync()!);

    int? num3 = int.parse(stdin.readLineSync()!);

    int? num4 = int.parse(stdin.readLineSync()!);

    if(num1>num2)

    {

      if(num1>num3)

      {

        if(num1>num4)

        {

          print("$num1 is biggest.....");

        }

        else

        {

          print("$num4 is biggest.....");

        }

      }

      else

      {

        if(num3>num4)

        {

          print("$num3 is biggest.....");

        }

        else

        {

          print("$num4 is biggest.....");

        }

      }

    }

  else

  {

    if(num2>num3)

      {

        if(num2>num4)

        {

          print("$num2 is biggest.....");

        }

        else

        {

          print("$num4 is biggest.....");

        }

      }

      else

      {

        if(num3>num4)

        {

          print("$num3 is biggest.....");

        }

        else

        {

          print("$num4 is biggest.....");

        }

      }

  }

}

1. Nested if1

import 'dart:io';

void main()

{

  int? year = int.parse(stdin.readLineSync()!);

  if(year%100 == 0)

  {

    if(year % 400 == 0)

    {

      print("$year is a leap year.");

    }

    else

    {

      print("$year is not a leap year.");

    }

  }

  else

  {

    if(year %4 == 0)

    {

      print("$year is a leap year.");

    }

    else

    {

      print("$year is not a leap year.");

  }

  }

}

**Switch case in dart**

import 'dart:io';

void main()

{

  int q,total;

  print("Enter your choice...\n1.Gujarati thali.\n 2. Panjabi thali");

  int? ch = int.parse(stdin.readLineSync()!);

  switch(ch)

  {

    case 1:

      print("Your choice is Gujarati Thali\n enter choice..");

      print("1.Fix Thali\n2. Unlimited Thali");

      int? ch1 = int.parse(stdin.readLineSync()!);

      switch(ch1)

      {

        case 1:

          print("your choice is Fix Thali.\n Enter Quantity..");

          q= int.parse(stdin.readLineSync()!);

          total=q\*180;

          print("your total payable bill is : $total");

          break;

        case 2:

          print("your choice is Unlimited Thali.\n Enter Quantity..");

          q= int.parse(stdin.readLineSync()!);

          total=q\*280;

          print("your total payable bill is : $total");

          break;

      }

    break;

    case 2:

      print("Your choice is Punjabi Thali\n enter choice..");

      print("1.Fix Thali\n2. Unlimited Thali");

      int? ch1 = int.parse(stdin.readLineSync()!);

      switch(ch1)

      {

        case 1:

          print("your choice is Fix Thali.\n Enter Quantity..");

          q= int.parse(stdin.readLineSync()!);

          total=q\*220;

          print("your total payable bill is : $total");

          break;

        case 2:

          print("your choice is Unlimited Thali.\n Enter Quantity..");

          q= int.parse(stdin.readLineSync()!);

          total=q\*320;

          print("your total payable bill is : $total");

          break;

      }

  }

}

**Loops in Dart** For loop

import 'dart:io';

//example 1:

void main()

{

  var i=0;

  for(i=0;i<=10;i++)

  {

    stdout.write("$i ");

  }

  print("Example 2:");

  print('\_'\*50);

  for(i=10;i>=0;i--)

  {

    stdout.write("$i ");

  }

  print("Example 3:");

  print('\_'\*50);

  print("Enter 2 no");

  int? num1 = int.parse(stdin.readLineSync()!);

  int? num2 = int.parse(stdin.readLineSync()!);

  int gcd=1;

  for(int i=1;i<=num1 && i<= num2;i++)

  {

    if(num1 % i == 0 && num2 % i == 0)

    {

      gcd=i;

    }

  }

  print("Gcd of $num1 and $num2 is : $gcd");

  print("Example 4:");

  print('\_'\*50);

  print("Enter no");

  int? num = int.parse(stdin.readLineSync()!);

  print("Factors are: ");

  for(int i=1;i<=num;i++)

  {

    if(num%i==0)

    {

      stdout.write("$i  ");

    }

  }

  print("Example 5:");

  print('\_'\*50);

  print("Enter no");

  num = int.parse(stdin.readLineSync()!);

  int count=0;

  for(int i=1;i<=num;i++)

  {

    if(num%i==0)

    {

      count++;

    }

  }

  if(count==2)

  {

    print("$num is prime no.");

  }

  else

  {

    print("$nun is non prime no.");

  }

  print("Example 6:");

  print('\_'\*50);

  int n1=0;

  int n2=1;

  int next=0;

  print("Enter no for Fibonacci series");

  var n = int.parse(stdin.readLineSync()!); // input

//  print('Fibonacci($i) = ${Fibonacci(i)}');

  stdout.write("$n1 $n2");

  for(int i=1; i<=n-2;i++)

  {

    next=n1+n2;

    n1=n2;

    n2=next;

    stdout.write(" $n2");

  }

}

**For in**

void main()

{

  var list=[1,4,76,2,34,89];

  for (int i in list)

  {

    print(i);

  }

  print("\_\_\_\_\_\_\_\_\_\_");

  var list1=['bike','car','cycle'];

  for( String i in list1)

  {

      print(i);

  }

}

For each

void main()

{

  var list=[1,4,76,2,34,89];

  list.forEach((var num) => print(num));

  print("\_\_\_\_\_\_\_\_\_\_");

  var list1=['bike','car','cycle'];

  list1.forEach((var l)=> print(l));

}

While

import 'dart:io';

import 'dart:math';

void main()

{

  int i=0;

  while(i<=10)

  {

    //print(i);

    stdout.write("$i ");

    i++;

  }

  print("Example 2:  LCM ");

  print('\_'\*50);

  print("enter 2 no");

  int? num1 = int.parse(stdin.readLineSync()!);

  int? num2 = int.parse(stdin.readLineSync()!);

int max = (num1>num2)?num1:num2;

  while(true)

  {

      if(max%num1==0 && max %num2==0)

      {

        print("Lcm of $num1 and $num2 is : $max");

        break;

      }

    max++;

  }

  print("Example 3: no is palingdrom or not:");

  print('\_'\*50);

  print("enter no for finding palingdrom number");

  dynamic num = int.parse(stdin.readLineSync()!);

  int org = num;

  dynamic rem=0,rev=0;

  while(num!=0)

  {

    rem= (num % 10);

    print("rem: $rem");

    rev = rev \* 10 + rem;

    print("rev: $rev");

    num~/=10;

  }

  print("reverse no is :$rev");

  num=org;

  if(org==rev)

  {

    print("$num is palingdrom");

  }

  else{

    print("$num is not palingdrom");

  }

  print("Example 4:");

  print('\_'\*50);

  print("enter no for finding number is armstrong or not");

  num = int.parse(stdin.readLineSync()!);

  org = num;

  rem=0;

  rev=0;

  dynamic arm=0;

  int count=0;

  while(num!=0)

  {

    count++;

    num~/=10;

  }

  num=org;

  while(num!=0)

  {

    rem= (num % 10);

    print("rem: $rem");

    arm = arm + pow(rem,count);

    print("arm: $arm");

    num~/=10;

  }

  if(org==arm)

  {

    print("$org is armstrong no");

  }

  else{

    print("$org is not armstrong no");

  }

 }

**Dowhile:**

import 'dart:io';

void main()

{

  int i=1;

  do{

    stdout.write("$i ");

    i++;

  }while(i<=10);

  print("-"\*20);

  print("example 2:");

  int a=1;

  int? n = int.parse(stdin.readLineSync()!);

  do{

    int r= n\*a;

    print("$n \* $a = $r");

    a++;

  }while(a<=10);

}

**Nested loop**

import 'dart:convert';

import 'dart:io';

void main()

{

  int i,j;

  for(i=0;i<=4;i++)

  {

    for(j=0;j<=4;j++)

    {

      stdout.write("$i ");

    }

    print("");

  }

  print("\_\_\_\_\_\_\_ example 2\_\_\_\_\_\_\_\_\_\_");

  for(i=0;i<=4;i++)

  {

    for(j=0;j<=i;j++)

    {

      stdout.write("$j ");

    }

    print("");

  }

  print("\_\_\_\_\_\_\_ example 3\_\_\_\_\_\_\_\_\_\_");

  for(i=0;i<=4;i++)

  {

    for(j=0;j<=4;j++)

    {

      if(i==0 || i==4 || j==0 || j ==4)

        stdout.write("$i ");

      else

        stdout.write("  ");

    }

    print("");

  }

  print("\_\_\_\_\_\_\_ example 4\_\_\_\_\_\_\_\_\_\_");

  for(i=4;i>=0;i--)

  {

    for(j=i;j>=0;j--)

    {

      stdout.write("$j ");

    }

    print("");

  }

  print("\_\_\_\_\_\_\_ example 5\_\_\_\_\_\_\_\_\_\_");

  for(i=4;i>=0;i--)

  {

    for(j=4;j>=i;j--)

    {

      stdout.write("$j ");

    }

    print("");

  }

  print("\_\_\_\_\_\_\_ example 6\_\_\_\_\_\_\_\_\_\_");

  int value = 'A'.codeUnitAt(0); //get unicode for A

  for(i=0;i<=4;i++)

  {

    for(j=0;j<=i;j++)

    {

      String char = String.fromCharCode(value); //convert ascci in char

      stdout.write("$char ");

      value++;

    }

    print("");

  }

  print("\_\_\_\_\_\_\_ example 7\_\_\_\_\_\_\_\_\_\_");

  int k=0;

  for(i=0;i<=4;i++)

  {

    for(j=0;j<=4-i;j++)

    {

      stdout.write("  ");

    }

    for(k=0;k!=2\*i+1;k++)

    {

      stdout.write("$k ");

    }

    print("");

  }

  for(i=3;i>=0;i--)

  {

    for(j=0;j<=4-i;j++)

    {

      stdout.write("  ");

    }

    for(k=0;k!=2\*i+1;k++)

    {

      stdout.write("$k ");

    }

    print("");

  }

  print("\_\_\_\_\_\_\_ example 8\_\_\_\_\_\_\_\_\_\_");

  k=0;

  for(i=0;i<=4;i++)

  {

    for(j=0;j<=4-i;j++)

    {

      stdout.write("  ");

    }

    for(k=0;k!=2\*i+1;k++)

    {

        if(k==0 || k==2\*i)

        {

          stdout.write("$k ");

        }

        else{

          stdout.write("  ");

        }

    }

    print("");

  }

  for(i=3;i>=0;i--)

  {

    for(j=0;j<=4-i;j++)

    {

      stdout.write("  ");

    }

    for(k=0;k!=2\*i+1;k++)

    {

      if(k==0 || k==2\*i )

        {

          stdout.write("$k ");

        }

        else{

          stdout.write("  ");

        }

    }

    print("");

  }

}

1. **Functions in Dart :**

A function is a group of statements that together performs 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. 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](https://www.javatpoint.com/dart-programming) 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.

**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.

return\_type func\_name (parameter\_list):

{

    //statement(s)

**return** value;

}

// function with no argument with no return

import 'dart:io';

void add()

{

  int? a= int.parse(stdin.readLineSync()!);

  int? b= int.parse(stdin.readLineSync()!);

  int result = a + b;

  print("Result: $result");

}

// function with argument but no return

void add1(int a, int b)

{

  int result = a + b;

  print("Result: $result");

}

// function with no argument but return

int add2()

{

  int? a= int.parse(stdin.readLineSync()!);

  int? b= int.parse(stdin.readLineSync()!);

  int result = a + b;

  return result;

}

// function with argument with return

int add3(int a, int b)

{

  int result = a + b;

  return result;

}

void main()

{

  add();

  print("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

  int? a= int.parse(stdin.readLineSync()!);

  int? b= int.parse(stdin.readLineSync()!);

  add1(a,b);

  print("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

  print("outout3: ");

  print(add2());

  print("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

  print("outout4: ");

  print(add3(a,b));

}

**Example 2:**

void fun1(int g1, [ var g2 ])

{

  // Creating function 1

  print("g1 is $g1");

  print("g2 is $g2");

}

void fun2(int g1, { var g2, var g3 })

{

  // Creating function 1

  print("g1 is $g1");

  print("g2 is $g2");

  print("g3 is $g3");

}

void fun3(int g1, { int g2 = 12 })

{

  // Creating function 1

  print("g1 is $g1");

  print("g2 is $g2");

}

void main()

{

  // Calling the function with optional parameter

  print("Calling the function with optional parameter:");

  fun1(56); // vale for g1 is 56 , g2 = null

  // Calling the function with Optional Named parameter

  print("Calling the function with Optional Named parameter:");

  fun2(01, g3 : 12); // g1 = 01 and g3 is 12 nut g2 is null

  // Calling function with default valued parameter

  print("Calling function with default valued parameter");

  fun3(01);  //g1 =1 and g2 has already value assign

}

Example 3: Recursion function

import 'dart:io';

int fibonacci(int n)

{

  if(n<=1)

  {

    return n;

  }

  else

  {

    return fibonacci(n-1)+fibonacci(n-2);

  }

}

void fiboprint(int n)

{

  for(int i=0;i<n;i++)

  {

    int m=fibonacci(i);

    stdout.write("$m ");

  }

}

int factorial(int n)

{

  if(n<=1)

    return 1;

  else

    return n\* factorial(n-1);

}

void factor(int n,int i)

{

  if(i<=n)

  {

    if(n%i==0)

    {

      stdout.write("$i ");

    }

    factor(n, i+1);

  }

}

void main()

{

  print("Enter no");

  int? n= int.parse(stdin.readLineSync()!);

  print("febonacci Series:");

  fiboprint(n);

  print(" ");

  print("Enter no");

  n= int.parse(stdin.readLineSync()!);

  var fact=factorial(n);

  print("factorial of $n is: $fact");

  print("Enter no");

  n= int.parse(stdin.readLineSync()!);

  int i=1;

  factor(n, i);

}

import 'dart:io';

import 'dart:math';

// Function to calculate the area of a circle

double CircleArea(double radius) {

  return pi \* radius \* radius;

}

// Function to calculate the circumference of a circle

double CircleCircumference(double radius) {

  return 2 \* pi \* radius;

}

double SphereVolume(double radius, double height) {

  return pi \* radius \* radius \* height;

}

// Function to calculate the area of a rectangle

double RectangleArea(double length, double width) {

  return length \* width;

}

// Function to calculate the perimeter of a rectangle

double RectanglePerimeter(double length, double width) {

  return 2 \* (length + width);

}

// Function to calculate the volume of a rectangle

double RectangleVolume(double length, double width, double height) {

  return length \* height \* width;

}

// Function to calculate the area of a triangle

double TriangleArea(double base, double height) {

  return 0.5 \* base \* height;

}

// Function to calculate the perimeter of a triangle

double TrianglePerimeter(double side1, double side2, double side3) {

  return side1 + side2 + side3;

}

void main() {/ Circleprint("Enter Circle radius and height: ");

  double? circleRadius = double.parse(stdin.readLineSync()!);

  double? volume = double.parse(stdin.readLineSync()!);

  print('Circle with radius $circleRadius');

  print('Area: ${CircleArea(circleRadius)}');

  print('Circumference: ${CircleCircumference(circleRadius)}\n');

  print('Volume: ${SphereVolume(circleRadius,volume)}');

  // Rectangle

  print("Enter Rectangle's length, width and height: ");

  double? rectangleLength = double.parse(stdin.readLineSync()!);

  double? rectangleWidth = double.parse(stdin.readLineSync()!);

  double? rectangleheight = double.parse(stdin.readLineSync()!);

  print('Rectangle with length $rectangleLength and width $rectangleWidth');

  print('Area: ${RectangleArea(rectangleLength, rectangleWidth)}');

  print('Perimeter: ${RectanglePerimeter(rectangleLength, rectangleWidth)}\n');

  print('Volume: ${RectangleVolume(rectangleLength, rectangleWidth,rectangleheight)}');

  // Triangle

  print("Enter Triangle's base and height ");

  double? triangleBase = double.parse(stdin.readLineSync()!);

  double? triangleHeight = double.parse(stdin.readLineSync()!);

   print('Triangle with base $triangleBase and height $triangleHeight');

  print('Area: ${TriangleArea(triangleBase, triangleHeight)}');

print("Enter Triangle's side1,side2,side3 ");

double? triangleSide1 = double.parse(stdin.readLineSync()!);

  double? triangleSide2 = double.parse(stdin.readLineSync()!);

  double? triangleSide3 = double.parse(stdin.readLineSync()!);

   print('Perimeter: ${TrianglePerimeter(triangleSide1, triangleSide2, triangleSide3)}');

}

**Dart Anonymous Function**

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:

(parameter\_list) {     statement(s)

}

/\*For functions that contain just one expression, you can use a shorthand syntax. The => expr syntax is a shorthand for { return expr; }.

The => notation is sometimes referred to as arrow syntax.

Note: Only an expression—not a statement—can appear between the arrow (=>) and the semicolon (;). For example, you can’t put an if statement there, but you can use a conditional expression.

\*/

int sum(int x, int y) {

  return x + y;

}

//can be rewritten using the arrow function syntax as follows:

int sum1(int x, int y) => x + y;

String checkNumber(int x) => x > 10 ? "Number is greater than 10" : "Number is less than or equal to 10";

//is equivalent to:

String checkNumber1(int x) {

  return x > 10 ? "Number is greater than 10" : "Number is less than or equal to 10";

}

void main() {

  var list = ["James","Patrick","Mathew","Tom"];

  print("Example of anonymous function");

  list.forEach((item) {

      print('${list.indexOf(item)}: $item');

});

  print(" ");

// another way...

      list.forEach(

        (item) => print("${list.indexOf(item)}: $item"));

  print(sum(5, 7));

  print(sum1(2, 3));

  print(checkNumber(3));

  print(checkNumber1(50));

}

**Lexical function:**

/\*

Dart is a lexically scoped language. Lexically scoped means that as we move downwards to the latest variable declaration, the variable value will depend on the innermost scope in which the variable is present.

Example 1:

 \*/

 void main(){

 var language = 'Dart';

   void printLanguage(){

      language = 'DartLang';

      print("Language is ${language}");

   }

   printLanguage();

}

//example 2:

main() {

  void inner ()

  {

        int level = 1; //not visible in main()

        String example = "scope"; //not visible in main()

        print('example: $example, level: $level');

  }

  inner();// call function

  // without calling function we cant go in inner fun

 // print('level: $level and example: $example'); //results in an Error

}

//Example 3:

main() { //a new scope

  String language = "Dart";

  void outer()  {

    //curly bracket opens a child scope

    String level = 'one'; // var for outer

    String example = "scope";

    void inner() { //another child scope

      //the next 'level' variable has priority over previous

      //named variable in the outer scope with the same named identifier

      Map level = {'count': "Two"};

      //prints example: scope, level:two

    print(" ");

      print("Inner fun");

      print('example: $example, level: $level');

      //inherited from the outermost scope: main

      print('What Language: $language');

    } //end inner scope

    inner();

    print(" ");

    print(" outer fun value");

    //prints example: scope, level:one

    print('example: $example, level: $level');

  } //end outer scope

  outer();

} //end main scope

**oops concepts of Dart Class and Object**

Dart is an object-oriented language. It supports object-oriented programming features like classes, interfaces, etc.A class is a blueprint for creating objects. It is a collection of data member and member functions. Class is a programmer-defined data type, which includes local methods and local variables. A class encapsulates data for the object.

Syntax :-

class class\_name{

<fields>

<getters/setters>

<constructors>

<functions>

}

**How to create objects from a class?**

• We can create several objects from the same class, with each object having its own set of properties.

• In order to work with a class, we need to create an object from it.

• In order to create an object, we use the new keyword.

• Example :

obj = new Class ();

Obj1 = new Class ();

**Object in Dart**

There are two ways to define object of a class.

• Using period operator(.)

• Using Cascade operator (..)

// Creating Class named Gfg

class Greet {

  String str = '';

  void greeting()

  {

    print("Welcome to $str");

  }

}

void main()

{

  Greet greet = new Greet();

greet.str = 'Hello Dart!!!!';

  greet.greeting();

}

**Constructor and types of Constructor** A constructor is a special function of the class that is responsible for initializing the variables of the class.

Dart defines a constructor with the same name as that of the class.

• A constructor is a function and hence can be parameterized.

• However, unlike a function, constructors cannot have a return type.

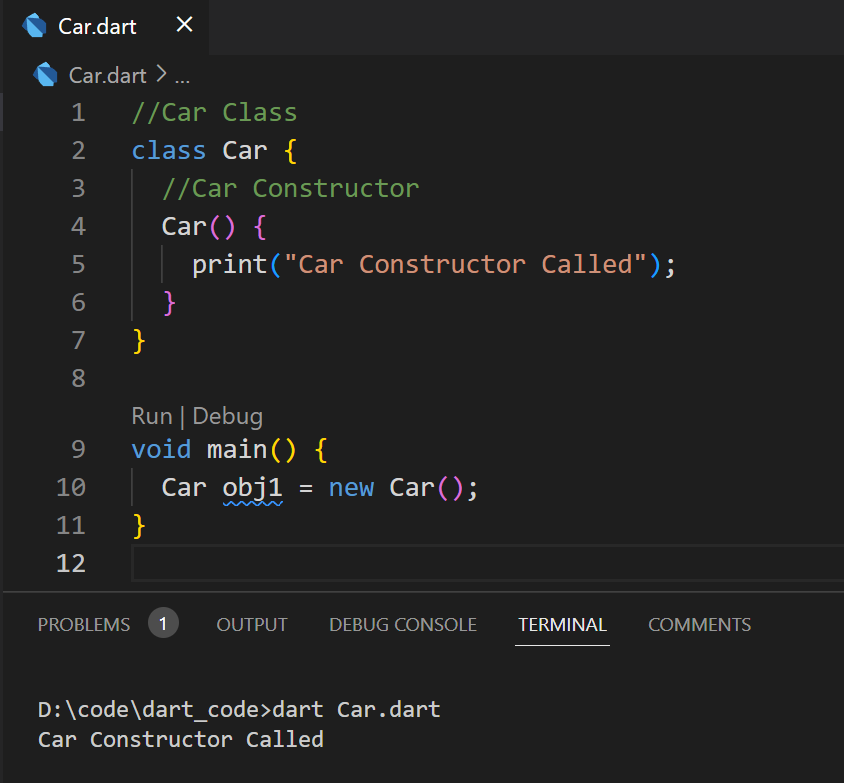
• Constructor will automatically called when you call object of class.

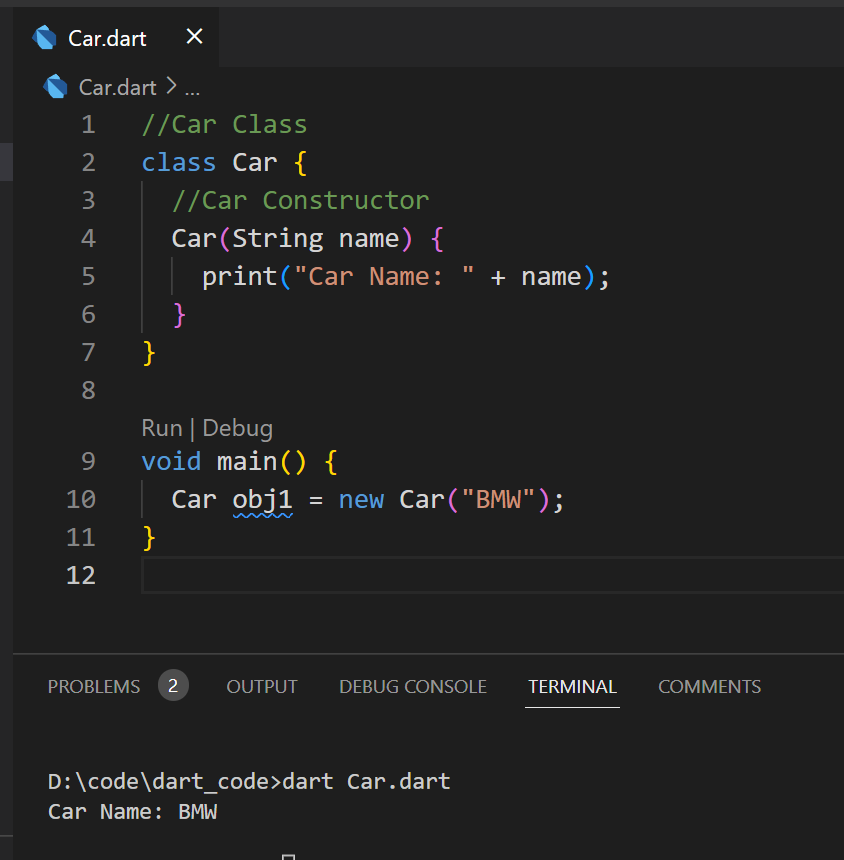
Syntax :-

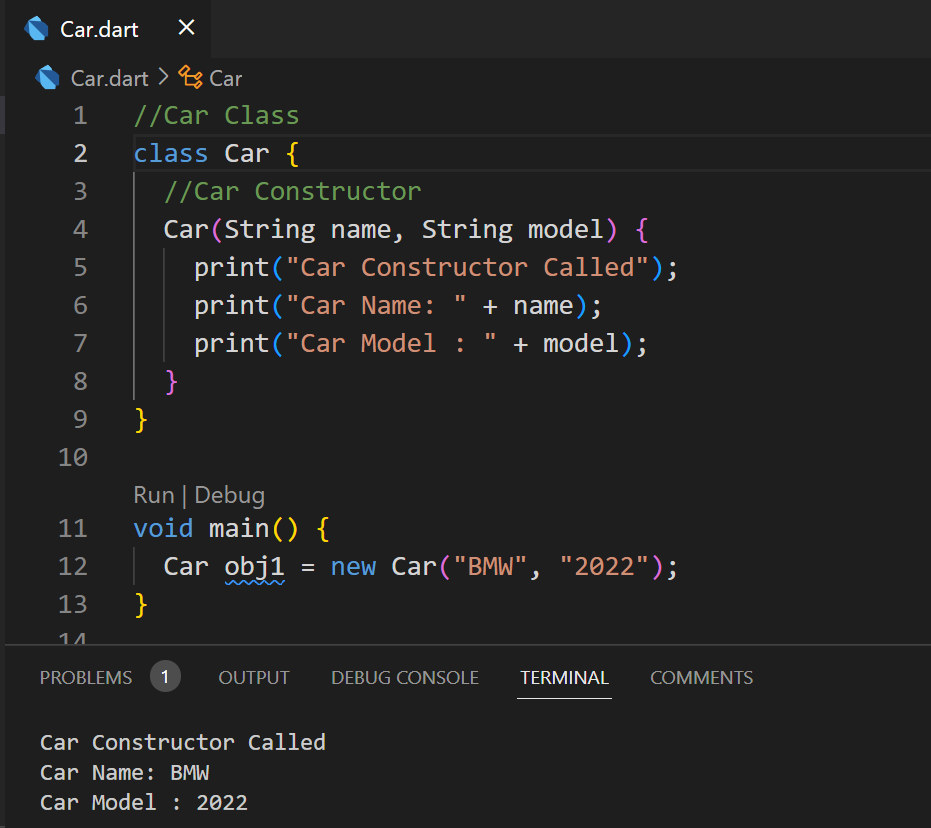
Class\_name(parameter\_list) {

//constructor body

}







import 'dart:io';

class Room

{

  var l,b,h;

  Room(var l,var b,var h)

  {

      this.l = l;

      this.b = b;

      this.h = h;

  }

  int cal\_area()

  {

    return l\*b;

  }

  int cal\_volume()

  {

    return l\*b\*h;

  }

  void cal\_cost()

  {

    int area = cal\_area();

    print("Area of Room is : $area");

    var cost = 25; // per sqm

    var totalcost = cost\*area;

    print("total color cost of Room is : $totalcost");

 }

}

void main()

{

  print("enter length, breath, height");

  int? l =  int.parse(stdin.readLineSync()!);

  int? b =  int.parse(stdin.readLineSync()!);

  int? h =  int.parse(stdin.readLineSync()!);

  Room r = new Room(l,b,h);

  int area = r.cal\_area();

  print("Area of Room is : $area");

  int volume = r.cal\_volume();

  print("Area of Room is : $volume");

  r.cal\_cost();

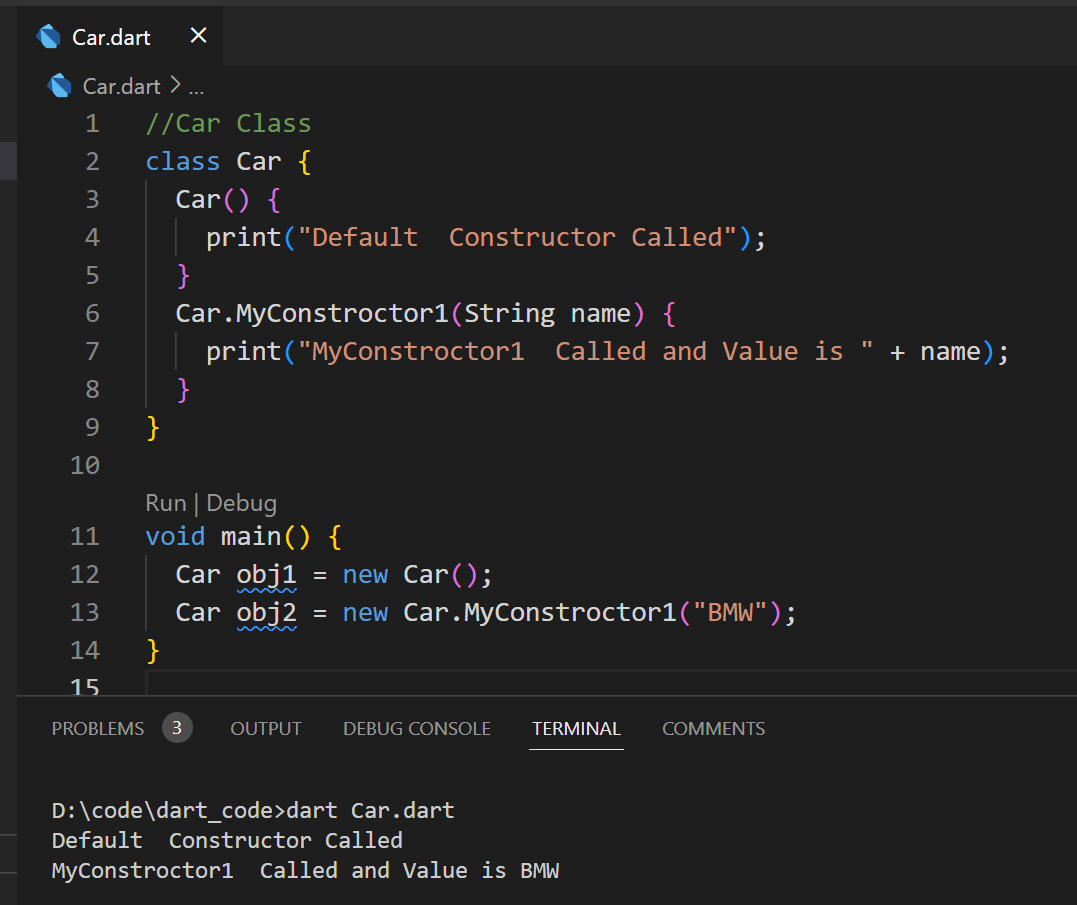
}

**Named Constructors**

Dart provides named constructors to enable a class define multiple constructors.

Syntax : Defining the constructor

Class\_name.constructor\_name(param\_list)

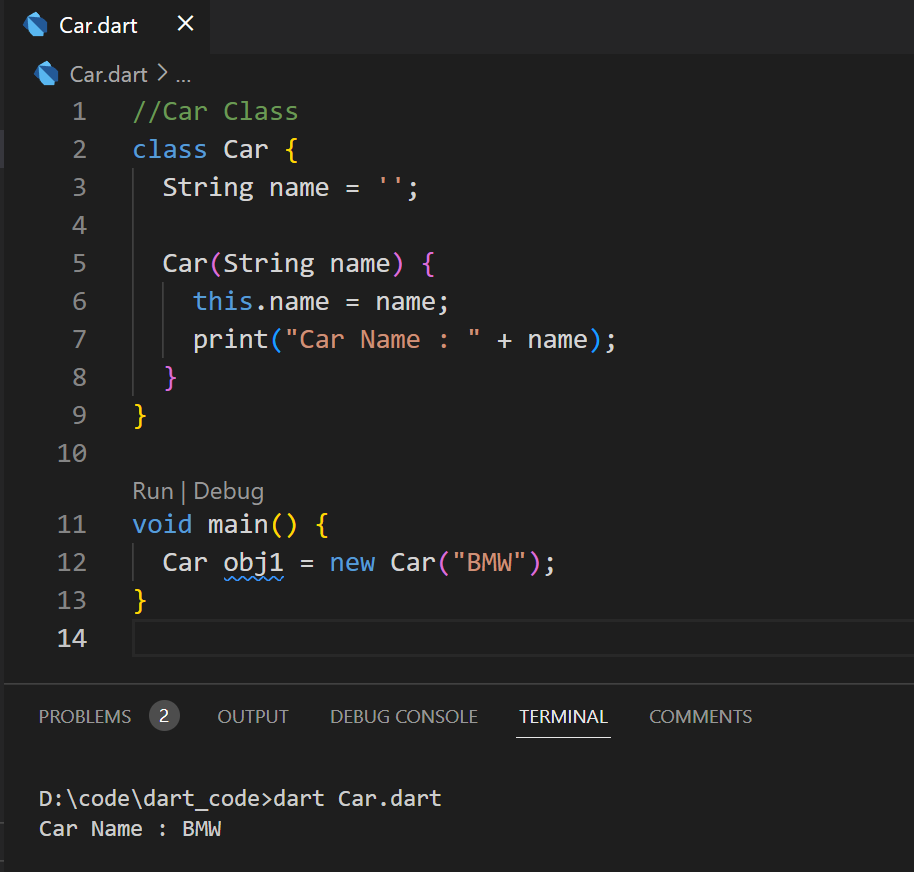


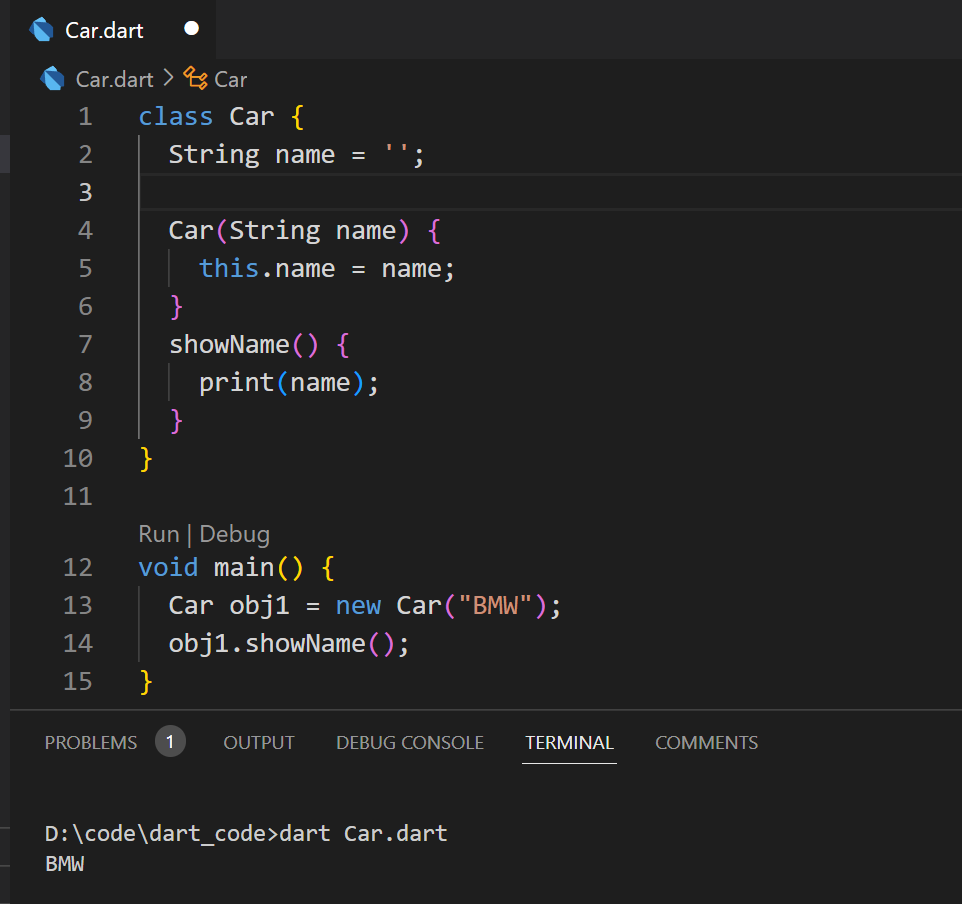
**The this Keyword**

The this keyword refers to the current instance of the class.

Here, the parameter name and the name of the class’s field are the same.

Hence to avoid ambiguity, the class’s field is prefixed with the this keyword.





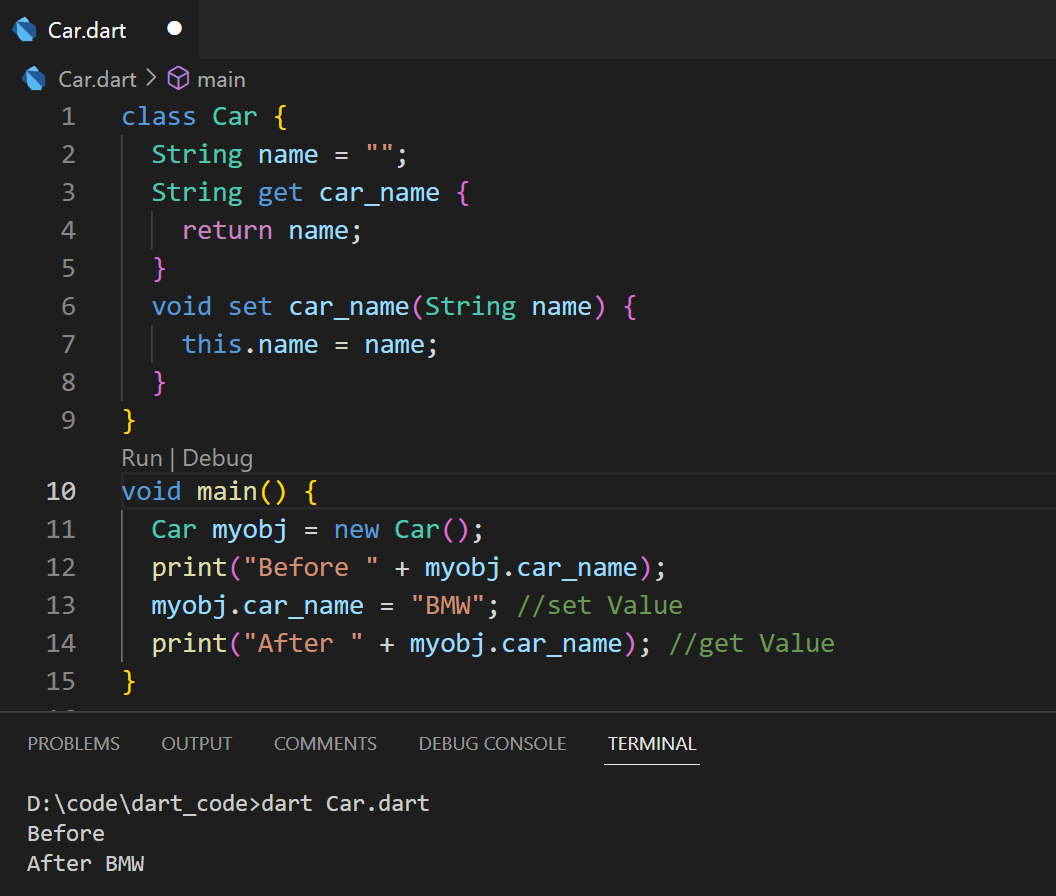
**Class ─ Getters and Setters**

• Syntax: Defining a getter

Return\_type get identifier { }

• Syntax: Defining a setter

set identifier { }



class Employee {

  String name = " ";

  String get Employee1 {

    return name;

  }

  void set Employee1(String name) {

    this.name = name;

  }

  void result(){

    print(name);

  }

}

void main() {

Employee Emp = new Employee();

Emp.name=" employee : avinash ,aakanksha";

Emp.result();

}

import 'dart:io';

class Book {

  String title;

  String author;

  bool isAvailable;

  Book(this.title, this.author, this.isAvailable);

}

class User {

  String name;

  List<Book> borrowedBooks;

  User(this.name) : borrowedBooks = [];

  void borrowBook(Book book) {

    if (book.isAvailable) {

      book.isAvailable = false;

      borrowedBooks.add(book);

      print('$name has borrowed "${book.title}" by ${book.author}');

    } else {

      print('Sorry, "${book.title}" is not available.');

    }

  }

  void returnBook(Book book) {

    if (borrowedBooks.contains(book)) {

      book.isAvailable = true;

      borrowedBooks.remove(book);

      print('$name has returned "${book.title}" by ${book.author}');

    } else {

      print('You have not borrowed "${book.title}".');

    }

  }

}

List<Book> books = [

  Book('The Great Gatsby', 'F. Scott Fitzgerald', true),

  Book('To Kill a Mockingbird', 'Harper Lee', true),

  Book('1984', 'George Orwell', false)

];

List<User> users = [];

void borrowBook() {

  print('Enter your name:');

  var name = stdin.readLineSync()!;

  var user = users.firstWhere((user) => user.name == name,

      orElse: () {

        var newUser = User(name);

        users.add(newUser);

        return newUser;

      });

  print('Available Books:');

  for (var i = 0; i < books.length; i++) {

    if (books[i].isAvailable) {

      print('${i + 1}. ${books[i].title} by ${books[i].author}');

    }

  }

  print('Enter the number of the book you want to borrow:');

  var choice = int.parse(stdin.readLineSync()!);

  if (choice > 0 && choice <= books.length) {

    var selectedBook = books[choice - 1];

    user.borrowBook(selectedBook);

  } else {

    print('Invalid choice');

  }

}

void returnBook() {

  print('Enter your name:');

  var name = stdin.readLineSync()!;

  var user = users.firstWhere((user) => user.name == name,

      orElse: () {

        print('User not found');

        return User('');

      });

  if (user.name.isNotEmpty) {

    if (user.borrowedBooks.isEmpty) {

      print('You have not borrowed any books.');

    } else {

      print('Your borrowed books:');

      for (var i = 0; i < user.borrowedBooks.length; i++) {

        print('${i + 1}. ${user.borrowedBooks[i].title} by ${user.borrowedBooks[i].author}');

      }

      print('Enter the number of the book you want to return:');

      var choice = int.parse(stdin.readLineSync()!);

      if (choice > 0 && choice <= user.borrowedBooks.length) {

        var returnedBook = user.borrowedBooks[choice - 1];

        user.returnBook(returnedBook);

      } else {

        print('Invalid choice');

      }

    }

  }

}

void displayAvailableBooks() {

  print('Available Books:');

  for (var i = 0; i < books.length; i++) {

    if (books[i].isAvailable) {

      print('${i + 1}. ${books[i].title} by ${books[i].author}');

    }

  }

}

void main() {

  while (true) {

    print('\nMenu:');

    print('1. Borrow a book');

    print('2. Return a book');

    print('3. Display available books');

    print('4. Exit');

    print('Enter your choice:');

    var choice = int.parse(stdin.readLineSync()!);

    switch (choice) {

      case 1:

        borrowBook();

        break;

      case 2:

        returnBook();

        break;

      case 3:

        displayAvailableBooks();

        break;

      case 4:

        return;

      default:

        print('Invalid choice');

    }

  }

}

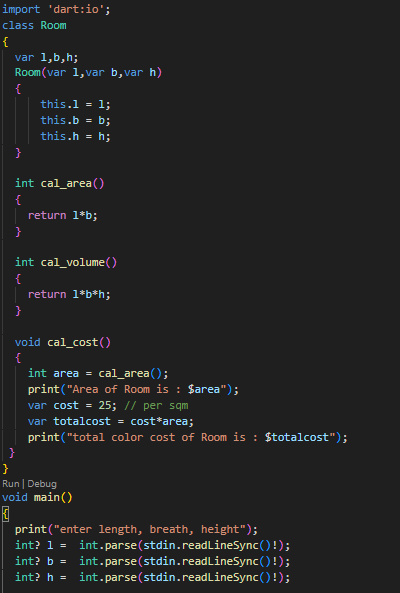
**6.3 This, static and super keyword The static Keyword**

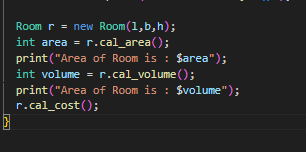
• The static keyword can be applied to the data members of a class,

i.e., fields and methods.

• A static variable retains its values till the program finishes execution.

• Static members are referenced by the class name.





Static Keyword Examples:

Example 1: Static Variables

// Dart Program to show

// Static methods in Dart

class Employee {

static var emp\_dept;

var emp\_name;

int? emp\_salary;

// Function to show details

// of the Employee

showDetails() {

  print("Name of the Employee is: ${emp\_name}");

  print("Salary of the Employee is: ${emp\_salary}");

  print("Dept. of the Employee is: ${emp\_dept}");

}

}

// Main function

void main() {

Employee e1 = new Employee();

Employee e2 = new Employee();

Employee.emp\_dept = "MIS";

print("Dart static Keyword Example");

e1.emp\_name = 'Rahul';

e1.emp\_salary = 50000;

e1.showDetails();

e2.emp\_name = 'Tina';

e2.emp\_salary = 55000;

e2.showDetails();

}

Example 2: Static Funtion

// Dart program in dart to

// illustrate static method

class StaticMem

{

static int? num;

static disp()

{

  print("The value of num is ${StaticMem.num}") ;

}

}

void main() {

// initialize the static variable

StaticMem.num = 75;

// invoke the static method

StaticMem.disp();

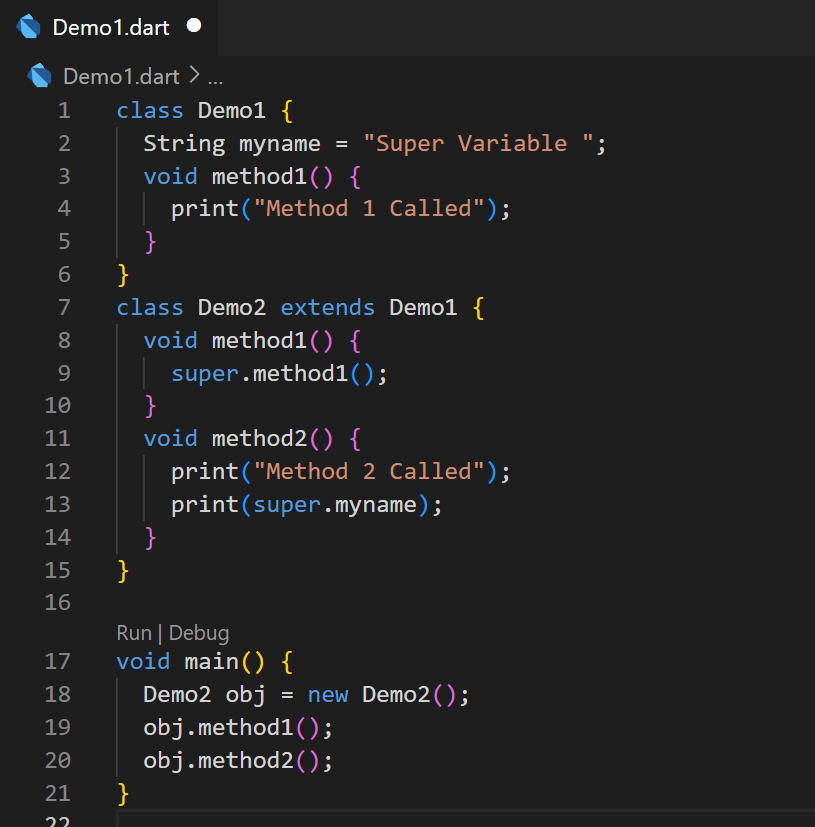
}

**Super Method:**

The super keyword is used to refer to the immediate parent of a class.

• The keyword can be used to refer to the super class version of a variable, property, or method.

Super KeyWord Example



**6.4 Concept of Inheritance**  Inheritance is a mechanism in which one object acquires all the properties and behaviours of parent object. A class inherits from another class using the ‘extends’ keyword. Child classes inherit all properties and methods except constructors from the parent class. Multiple inheritance is not supported by the Dart, but we can apply the multiple interfaces. We can say that, using multiple interfaces, we can achieve multiple inheritance in Dart

Syntax :-

class child\_class\_name extends parent\_class\_name

class Demo{

}

class Demo1 extends Demo

{

//methods and fields goes here

}

• Single inheritance

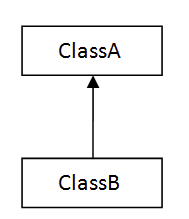
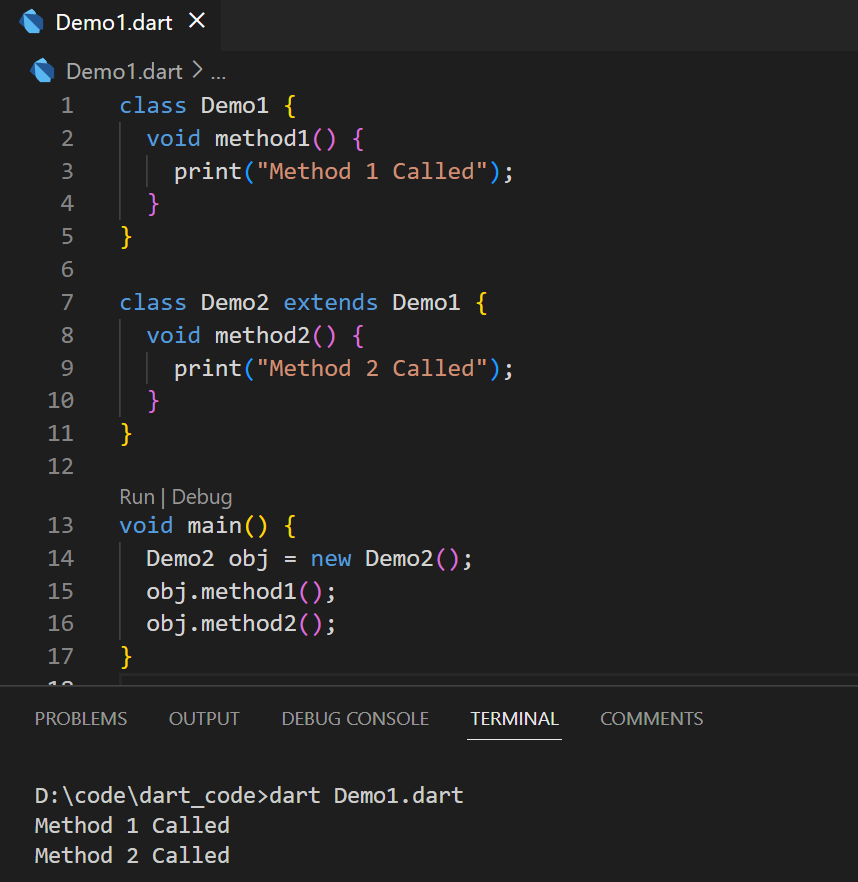
• Multilevel inheritance

• Hierarchical inheritance

• Multiple inheritance

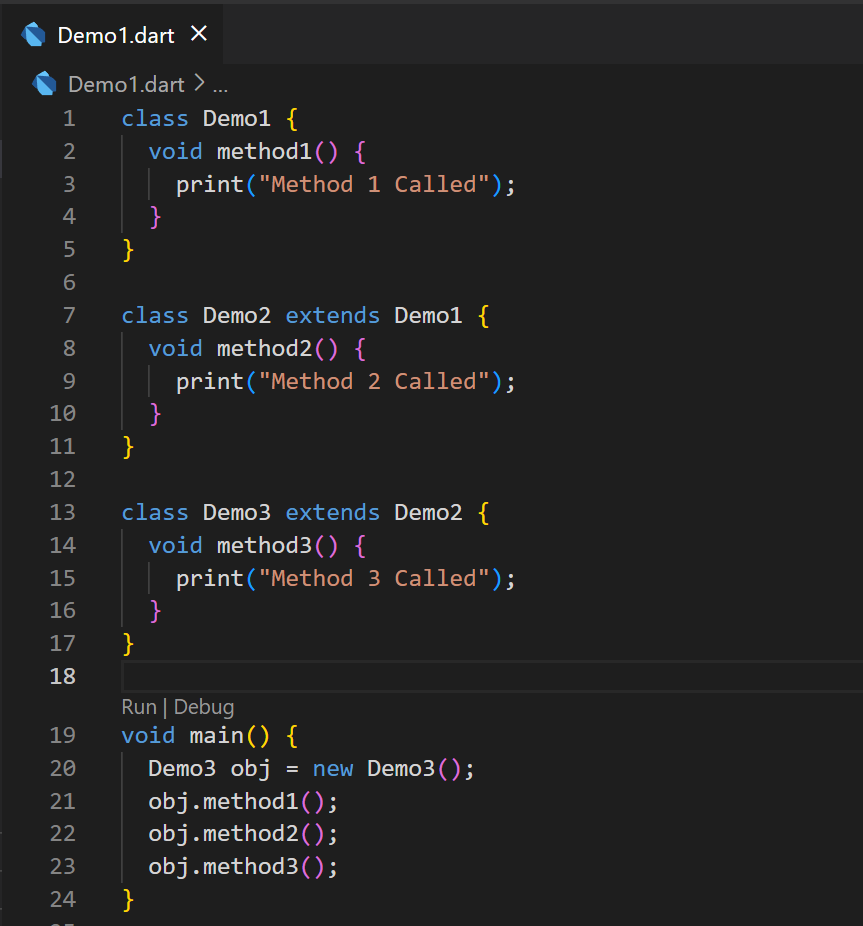
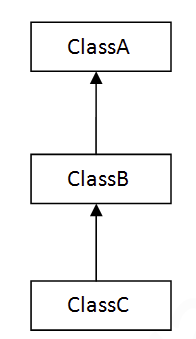
**Single Level Inheritance :**

• When one class inherits another class, it is known as single level inheritance.



**Multi Level Inheritance :**

* When one class inherits another class which is further inherited by another class, it is known as multi level inheritance.



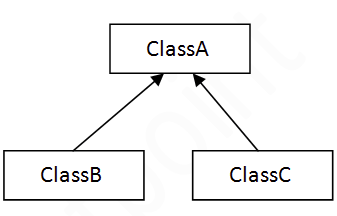
**Multiple Inheritance**

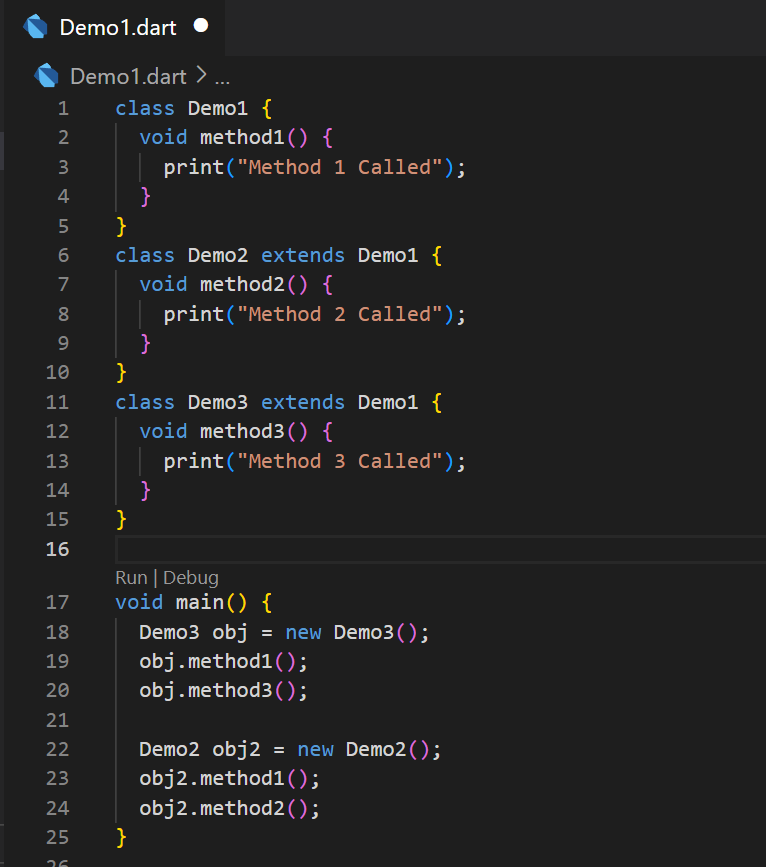
• A class can inherit from multiple classes. Dart doesn’t support multiple inheritance.

**Hyrarchical Inheritance**

• If more than one class can inherit members from one class is known as Hierarchical Inheritance.

**Hyrarchical Inheritance Example**





**Method Overriding**

Method overriding occurs in dart when a child class tries to override the parent class’s method. When a child class extends a parent class, it gets full access to the methods of the parent class and thus it overrides the methods of the parent class.

Example:

// Dart Program to illustrate the

// method overriding concept

class A {

// Creating a method

void show(){

  print("This is class A.");

}

}

class B extends A {

// Overriding show method

void show(){

  print("This is class B child of A.");

}

}

void main() {

// Creating objects

//of both the classes

A geek1 = new A();

B geek2 = new B();

// Calling same function

// from both the classes

// object to show method overriding

geek1.show();

geek2.show();

}

**Abstract Classes:**

An abstract class in Dart is defined as a class that contains one or more abstract methods (methods without implementation). To declare an abstract class, we use the abstract keyword. It’s important to note that a class declared as abstract may or may not include abstract methods. However, if a class includes an abstract method, it must be declared as abstract.

**Features of Abstract Class:**

**Abstract Methods**: A class containing an abstract method must be declared abstract. An abstract class may contain both abstract and concrete methods.

**Declaration**: A class can be declared abstract by using the abstract keyword.

Initialization: An abstract class cannot be instantiated.

**Inheritance**: An abstract class can be extended, but any subclass must implement all abstract methods.

Example:

// Creating Abstract Class

abstract class A {

    // Creating Abstract Methods

    void say();

    void write();

}

class B extends A {

    @override

    void say() {

        print("Hello!!");

}

    @override

    void write() {

        print("I am a Human");

    }

}

void main() {

    B b = B();

    b.say();

    b.write();

}

**Concept Of Callable Classes**

Dart allows the user to create a callable class which allows the instance of the class to be called as a function. To allow an instance of your Dart class to be called like a function, implement the **call()** method.

**Syntax:**

class class\_name {  
 ... // class content  
   
 return\_type **call** ( parameters ) {  
 ... // call function content  
 }  
   
}

Example:

// Creating Class A

class A {

  // Defining call method which create the class callable

  String call(String a, String b, String c) => 'Welcome to $a$b$c!';

}

// Main Function

void main() {

  // Creating instance of class

  var ex\_input = A();

  // Calling the class through its instance

  var ex\_output = ex\_input('Geeks', 'For', 'Geeks');

  // Printing the output

  print(ex\_output);

}

**Example 2:**

class Adder {

  int call(int a, int b) {

    return a + b;

  }

}

void main() {

  var adder = Adder();

  var sum = adder(1, 2); // Now you can call the instance directly

  print(sum); // prints 3

}

**6.5 Interface in Dart**

**Interfaces**

• Interfaces define a set of methods available on an object. Dart does not have a syntax for declaring interfaces. Class declarations are themselves interfaces in Dart. Classes should use the implements keyword to be able to use an interface. A class must redefine every function in the interface it wishes to implement.

Syntax: Implementing an Interface

class identifier implements interface\_name

Example



Example of Multiple Inheritance



class Student

{

   String name="";

   int age=0;

   void displayName( String name) {

         print("I am ${name}");

                            }

   void displayAge(int age) {

            print("My age is ${age}");

                               }

}

class Faculty

{

   String dep\_name="";

   int salary=0;

   void displayDepartment(String dep\_name) {

         print("I am a professor of ${dep\_name}");

                            }

   void displaySalary(int salary) {

            print("My salary is ${salary}");

                               }

}

// Defining interface by implenting another class

class College implements Student,Faculty

{

   // Overriding the Student class members

   String name="";

   int age=0;

   void displayName( String name) {

         print("I am ${name}");

                            }

   void displayAge(int age) {

            print("My age is ${age}");

                               }

//Overriding each data member of Faculty class

   String dep\_name="";

   int salary=0;

   void displayDepartment(String dep\_name) {

         print("I am a proffesor of ${dep\_name}");

                            }

   void displaySalary(int salary) {

            print("My salary is ${salary}");

}

}

void main()

{

College cg = new College();

cg.name = "Rohit Shah";

cg.age = 25;

cg.dep\_name = "Dart programming";

cg.salary = 50000;

cg.displayName(cg.name);

cg.displayAge(cg.age);

cg.displayDepartment(cg.dep\_name);

cg.displaySalary(cg.salary);

}

**Flutter Metadata:**

Dart supports metadata which is used to attach user defined annotations to program structures. Metadata consists of a series of annotations, each of which begin with the character @, followed a constant expression that starts with an identifier.

class Human{

    void run()

   {

      print("Human is running");

   }

}

class Man extends Human{

   @override   // this is metadata (data's data)

    void run(){

      print("Boy is running");

   }

}

void main(){

      Man m = new Man();

      //This will call the child class version of run()

      m.run();

}

# 

# Create own metadata

import 'dart:io';

class StudentMarks

{

  // Information about the student's marks.

  final double upperLimit;

  final double lowerLimit;

  // Here we are making the const constructor with upper and lower limit.

  const StudentMarks(this.lowerLimit, this.upperLimit);

}

// Using this meta data is for remarks for students.

// MetaData for the function topRemark().

@StudentMarks(80, 100)

void topRemark()

{

  print("Good Student.");

}

// MetaData for the function mediumRemark().

@StudentMarks(50, 79)

void mediumRemark()

{

  print("Medium Student.");

}

// MetaData for the function lowRemark().

@StudentMarks(0, 49)

void lowRemark()

{

  print("Need to focus on study.");

}

void main()

{

  print("enter marks");

  double? marks = double.parse(stdin.readLineSync()!);

  if(marks >= 80){

    topRemark();

  }

  else if(marks >= 50 && marks <= 79){

    mediumRemark();

  }

  else{

    lowRemark();

  }

}

# Dart Collection

# Dart doesn't support the array to store the data, unlike the other programming language. We can use the Dart collection in place of array data structure. We can enable the other classes of the collection in our Dart script by using the dart::core library. Ex List, Set,Map.

# **Dart Getters and Setters**

# 

# Getters and setters are the special class method that is used to read and write access to an object's properties. The getter method is used to reads the value of the variable or retrieve the value and setter method is used to set or initialize respective class fields. By default, all classes are associated with getter and setter method. However, we can override the default methods by defining getter and setter method explicitly.

# **Defining a getter**

# We can define the getters method by using the get keyword with no parameter a valid return type.

# Syntax:

# return\_type get field\_name{

# }

# **Defining a setter**

# We can declare the setter method using the set keyword with one 1 parameter and without return type.

# Syntax:

# set field\_name {

# }

class Car {

  String makedate;

  String modelname;

  int manufactureYear;

  int carAge;

  String color;

// Getter method

  int get age {

    return carAge;

  }

// Setter Method

  void set age(int currentYear) {

    carAge = currentYear - manufactureYear;

  }

// defining parameterized constructor

  Car({required this.makedate,required this.modelname,required this.manufactureYear,required this.color, this.carAge=0});

}

//Age here is both a getter and a setter. Let's see how we can use it.

void main() {

 Car c =

 Car(makedate:"Renault 20/03/2010",modelname:"Duster",manufactureYear:2010,color:"White");

  print("The car company is: ${c.makedate}");

  print("The modelname is: ${c.modelname}");

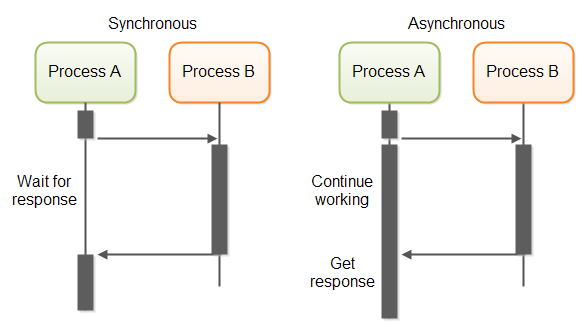
  print("The color is:${c.color}");

  c.age = 2024;

  print(c.age);

}

**6.9.2 Await Async in Dart**



**When to use?**

File I/O (Downloading File).

- Some sort of computation

- API call to a RESTFUL Service

**Async :**

Any Functions you want to run asynchronously(not existing or occurring at the same time.), needs to have the async modifier added to it. This modifier added to it. This modifier comes right after the function signature,

void hello() async {

print('something exciting is going to happen here...');

}

**Await :**

The await part basically says —go ahead and run this function asynchronously, and when it is done, continue on to the next line of code.

void main() async {

await hello();

print('all done');

}

Write these codes in dart\_code folder to see simple application of async await and future.

**Example : Async\_ex.dart**

String printMsg(){

  return ("Print Msg Called");

}

void main() async {

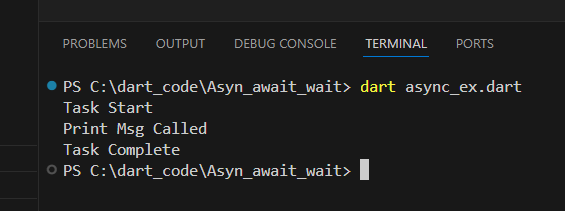
  print("Task Start");

  var msg = await printMsg();

  print(msg);

  print("Task Complete");

}



**Future**

Futures are a way to wrap something that will be resolved later on, so that we don’t need to wait for it to follow along with our program.

**Example : future\_ex.dart**

Future<void> printMsg() async {

  await Future.delayed(Duration(seconds: 3));

  // output will be displayed after 3 seconds

  print("Print Msg Called");

}

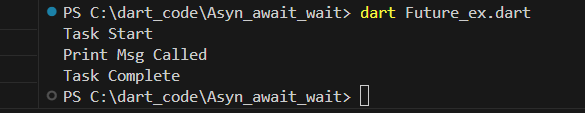
void main() async {

  print("Task Start");

  await printMsg();

  print("Task Complete");

}



**Example on without await**

Future<void> printMsg() async {

  await Future.delayed(Duration(seconds: 3));

  print("Print Msg Called");

}

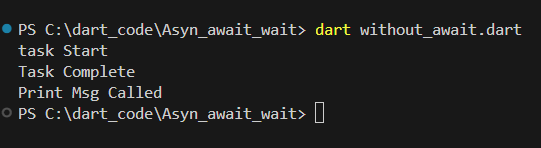
void main() async {

  print("task Start");

  printMsg(); // run after 3 second

  print("Task Complete"); // can't wait to run

}



Example: synchronous functions

String createOrderMessage() {

  var order = fetchUserOrder();

  return 'Your order is: $order';

}

Future<String> fetchUserOrder() =>

    // Imagine that this function is

    // more complex and slow.

    Future.delayed(

      const Duration(seconds: 2),

      () => 'Large Latte',

    );

void main() {

  print('Fetching user order...');

  print(createOrderMessage());

}

**6.9.3 Date Time:** DateTime Class is used to show the current date time or some specified date and time and use it in our programs for example:

Making an order at a particular date and time

Receipt

Bank Statements etc.

Example:

void main(){

// Get the current date and time.

var now = DateTime.now();

print(now);

// Create a new DateTime with the local time zone.

var y2k = DateTime(2000); // January 1, 2000

print(y2k);

// Specify the month and day.

y2k = DateTime(2000, 1, 2); // January 2, 2000

print(y2k);

// Specify the date as a UTC time.

y2k = DateTime.utc(2000); // 1/1/2000, UTC

print(y2k);

}

**Flutter**

**1 Flutter Introduction**

**1.1 Features of Flutter**

Flutter framework offers the following features to developers:

* Modern and reactive framework.
* Uses Dart programming language and it is very easy to learn.
* Fast development.
* Beautiful and fluid user interfaces.
* Huge widget catalog.
* Runs same UI for multiple platforms.
* High performance application.

**1.2 Advantages and disadvantages of flutter**

Flutter comes with beautiful and customizable widgets for high performance and outstanding mobile application. It fulfills all the custom needs and requirements. Besides these, Flutter offers many more advantages as mentioned below:

* Dart has a large repository of software packages which lets you to extend the capabilities of your application.
* Developers need to write just a single code base for both applications (both Android and iOS platforms). *Flutter* may to be extended to other platform as well in the future.
* Flutter needs lesser testing. Because of its single code base, it is sufficient if we write automated tests once for both the platforms.
* Flutter’s simplicity makes it a good candidate for fast development. Its customization capability and extendibility makes it even more powerful.
* With Flutter, developers has full control over the widgets and its layout.
* Flutter offers great developer tools, with amazing hot reload.

## 

## **1.3 Disadvantages of Flutter**

Despite its many advantages, flutter has the following drawbacks in it:

* Since it is coded in Dart language, a developer needs to learn new language (though 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 intermixed. We can overcome this using smart coding and using high level module to separate user interface and logic.
* Flutter is yet another framework to create mobile application. Developers are having a hard time in choosing the right development tools in hugely populated segment.

**Flutter Installation**

## Installation in MacOS

To install Flutter on MacOS, you will have to follow the following steps:

**Step 1**: Go to URL, https://flutter.dev/docs/get-started/install/macos and download latest Flutter SDK. As of April 2019, the version is 1.2.1 and the file is flutter\_macos\_v1.2.1- stable.zip.

**Step 2**: Unzip the zip archive in a folder, say /path/to/flutter

**Step 3**: Update the system path to include flutter bin directory (in ~/.bashrc file).

> export PATH="$PATH:/path/to/flutter/bin"

**Step 4**: Enable the updated path in the current session using below command and then verify it as well.

source ~/.bashrc

source $HOME/.bash\_profile echo $PATH

Flutter provides a tool, flutter doctor to check that all the requirement of flutter development is met. It is similar to the Windows counterpart.

**Step 5**: Install latest XCode, if reported by flutter doctor

**Step 6**: Install latest Android SDK, if reported by flutter doctor

**Step 7**: Install latest Android Studio, if reported by flutter doctor

**Step 8**: Start an android emulator or connect a real android device to the system to develop android application.

**Step 9**: Open iOS simulator or connect a real iPhone device to the system to develop iOS application.

**Step 10**: Install Flutter and Dart plugin for Android Studio. It provides the startup template to create a new Flutter application, option to run and debug Flutter application in the Android studio itself, etc.,

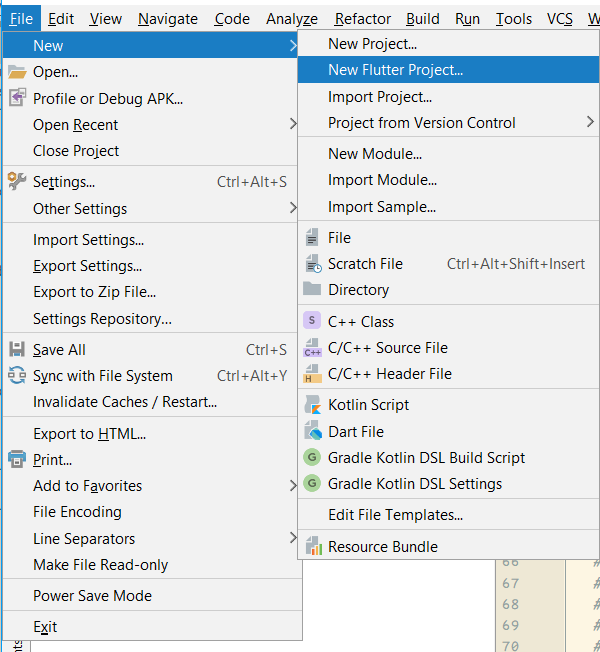
* Open Android Studio.
* Click **Preferences > Plugins**.
* Select the Flutter plugin and click Install.
* Click Yes when prompted to install the Dart plugin.
* Restart Android studio.

Create a simple *Flutter* application to understand the basics of creating a flutter application in the Android Studio.

**Step 1**: Open Android Studio

**Step 2**: Create Flutter Project. For this, click **File -> New -> New Flutter Project**

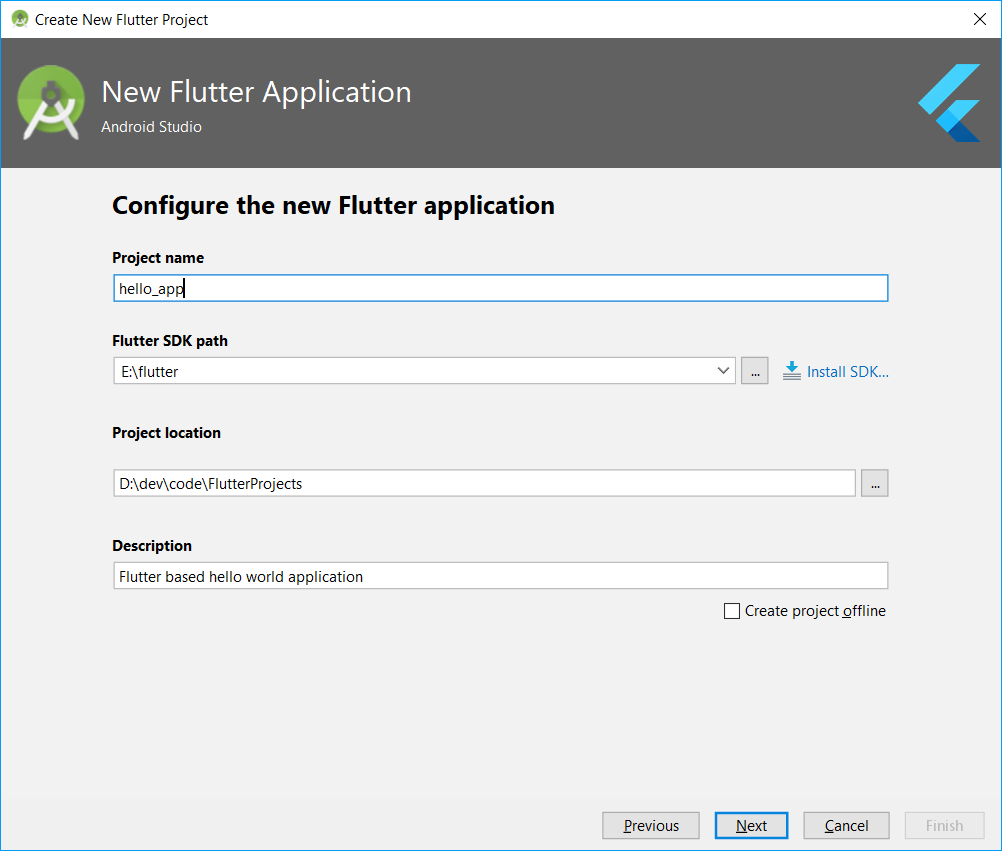
#### 



**Step 3**: Select Flutter Application. For this, select **Flutter Application** and click **Next**.

**Step 4**: Configure the application as below and click **Next**.

* + Project name: **hello\_app**
  + Flutter SDK Path: **<path\_to\_flutter\_sdk>**
  + Project Location: **<path\_to\_project\_folder>**
  + Description: **Flutter based hello world application**



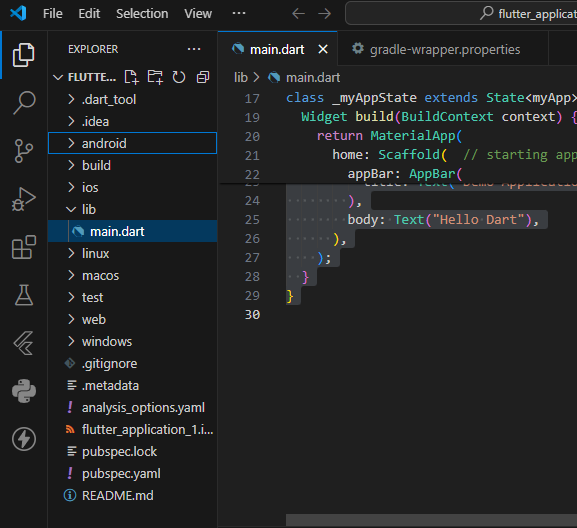
**Step 5:** Configure Project.

Set the company domain as **flutterapp.tutorialspoint.com** and click **Finish Step 6:** Enter Company domain.

2.1 Android Studio creates a fully working flutter application with minimal functionality.

Let us check the structure of the application and then, change the code to do our task.

The structure of the application and its purpose is as follows:



Various components of the structure of the application are explained here:

* + - **android** - Auto generated source code to create android application
    - **ios** - Auto generated source code to create ios application
    - **lib** - Main folder containing Dart code written using flutter framework
    - **lib/main.dart** - Entry point of the Flutter application
    - **test** - Folder containing Dart code to test the flutter application
    - **test/widget\_test.dart** - Sample code
    - **.gitignore** - Git version control file
    - **.metadata** - auto generated by the flutter tools
    - **.packages** - auto generated to track the flutter packages
    - **.iml** - project file used by Android studio
    - **pubspec.yaml** - Used by **Pub**, Flutter package manager
    - **pubspec.lock** - Auto generated by the Flutter package manager, **Pub**
    - **README.md** - Project description file written in Markdown format

**Step 7:** Replace the dart code in the *lib/main.dart* file with the below code:

import 'package:flutter/material.dart'; void main() => runApp(MyApp());

class MyApp extends StatelessWidget {

// This widget is the root of your application. @override

Widget build(BuildContext context) { return MaterialApp(

title: 'Hello World Demo Application', theme: ThemeData(

primarySwatch: Colors.blue,

),

home: MyHomePage(title: 'Home page'),

);

}

}

class MyHomePage extends StatelessWidget { MyHomePage({Key key, this.title}) : super(key: key);

final String title; @override

Widget build(BuildContext context) {

return Scaffold( appBar: AppBar(

title: Text(this.title),

),

body: Center( child:

Text(

'Hello World',

)

),

);

}

}

Let us understand the dart code line by line.

* + - **Line 1:** imports the flutter package, *material*. The *material* is a flutter package to create user interface according to the Material design guidelines specified by Android.
    - **Line 3:** This is the entry point of the Flutter application. Calls *runApp* function and pass it an object of *MyApp* class. The purpose of the *runApp* function is to attach the given widget to the screen.
    - **Line 5 - 17:** *Widget* is used to create UI in flutter framework. *StatelessWidget* is a widget, which does not maintain any state of the widget. *MyApp* extends *StatelessWidget* and overrides its *build* method. The purpose of the *build* method is to create a part of the UI of the application. Here, *build* method uses *MaterialApp*, a widget to create the root level UI of the application. It has three properties - *title*, *theme* and *home*.
      * *title* is the title of the application.
      * *theme* is the theme of the widget. Here, we set *blue* as the overall color of the application using *ThemeData* class and its property, *primarySwatch*.
      * *home* is the inner UI of the application, which we set another widget,

*MyHomePage*

* + - **Line 19 - 38**: *MyHomePage* is same as *MyApp* except it returns *Scaffold* Widget. *Scaffold* is a top level widget next to *MaterialApp* widget used to create UI conforming material design. It has two important properties, *appBar* to show the header of the application and *body* to show the actual content of the application. *AppBar* is another widget to render the header of the application and we have used it in *appBar* property. In *body* property, we have used *Center* widget, which centers it child widget. *Text* is the final and inner most widget to show the text and it is displayed in the center of the screen.

**Step 8:** Now, run the application using, **Run -> Run main.dart**

#### 

