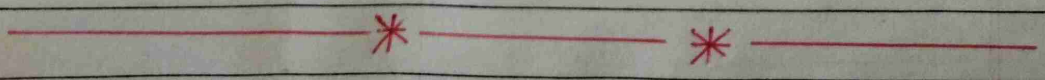
Date / /

Dart Language Course ...



26/11/21

Page No. 01
Date / /

DART Language Tutorial

What is Dart ?

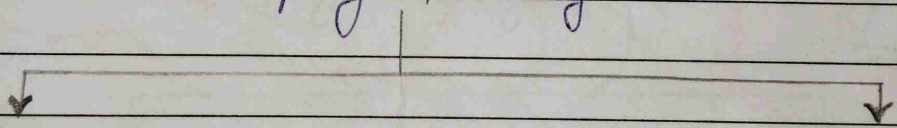
Dart is a skillable language that we can use to write simple scripts or full featured applications.

Dart helps us in creating mobile App, web App, Server App.

Dart is Created by Google itself.

Introduction & Setup :-

To program using Dart



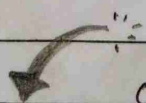
IntelliJ IDEA

- Install Dart SDK.
- Install IntelliJ IDEA
- Integrate Dart Plugin

DartPad.

- No Setup required.
- Online IDE by Dart.

Go to : dartpad.dart-lang.org



or Search : Dartpad.

dartpad.dartlang.org

Your Codes here ...	OUTPUT ...
	Errors (if any)

Version ---

Creating first Dart program :-

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

→ It is the entry point of our application.

; ✓ → To display any message.

Output

Hello World.

Other display statements :-

```
print("PS");
print(4);
print(4-2);
print(4*2);
print(true);
print(false);
```

} Boolean values. (No " " needed)

Comments in Dart :-

```
void main()
{
    // This is my comment
    /* Multiline
       Comments */
    print("Praveen");
}
```

// This is my comment : Single line comment (more used)

/* Multiline Comments */ : multi-line comment (less used)

Data-types & Variables :-

Dart has special support for these data types

- Numbers → int
  → double (float is not supported)
- Strings
- Booleans
- Lists (or Arrays)
- Maps
- Runes (for expressing unicode characters in String)
- Symbols.

NOTE: All data-types in Dart are Objects. Therefore, values are by default null.

Syntax for declaration :-

⇒ data type variable name = value ;

ie.

```
int age = 10;
```

OR

⇒ var age = 10; // It is inferred automatically.

More examples...

```
String name = "Henry";
```

OR

```
var name = "Henry";
```

bool isValid = true;

OR

Var isValid = true;

* Examples ...

```
void main() {
int score = 23;
var count = 23;          // Inferred as an integer automatically by compiler.
int hexValue = 0xEADEBAEE;          // 8 digits must be there in hexvalue)
                                    // It'll be converted to integer value)
String name;
double exponent = 1.42e5;           // converted to boolean)
print(count);
print(hexvalue);
print(exponent);
print(name);
}
```

(0x: 0 = Zero, x = X; Keyword)

23
394046390
142000.0

Output

null

∵ "name" was not initialized.
∴ It is by default null.

- Literals
- Various ways to write String literals.
- String Interpolation.

```
void main()
{
  // Literals
    var isCool = true;
    int x = 2;
    "John";
    4.5;

  // Various ways to write String literals.
    String s1 = 'Single quote string';
    String s2 = "Double quote string";
    String s3 = 'It\'s easy';
    String s4 = "It's easy";
    String s5 = 'This is going to be a very long'
                'String.'
                'This is just a sample String demo in'
                'Dart language.';

  // String Interpolation.
    String name = "Kevin";
    print("My name is $name");
    print("The number of characters in String Kevin
           is ${name.length}");
    int l = 20;
    int b = 10;
    print("Sum of $l & $b is ${l+b}");
    print("Area of length $l & breadth $b is
           ${l*b}");
}
```

To ignore any particular character just after \

\ is escape character

Prefix dollar $ added

curly braces used

: We use ["My name is $name"] instead of ["My name is " + name].

(c) Final & Const Keywords :-

If you never want to change a value then use final & const keywords.

- final name = "Peter";
- Const PI = 3.14;

Difference between final & Const

: final variable can only be set once & it is initialized when accessed.

: Const variable is implicitly final but it is a Compile-time constant.
i.e. it is initialized during compilation.

: Instance variable (in class) can be final but Cannot be Const.
If you want a constant at class level then make it static const.

Syntax :-

```
void main()
{
    // final keyword.
    final    CityName = 'Mumbai';

    final    String CountryName = 'India';

    // Const keyword.
    Const    PI = 3.14;
    Const    double gravity = 9.8;

}

// at class level.
class Circle {

    final Color = 'Red';
    Const    PI = 3.14;    ✗ error (Only static fields can be declared as const.)

    Static Const PI = 3.14; ✓

}
```

Control Flow Statements

→ If & Else
→ Conditional Expressions
→ Switch & Case statements.

If & Else

If the condition is true then (do something), else (do something else).

Syntax :-

```
if (condition) {
        // Execute if true.
} else {
        // Execute if condition is false.
}
```

* eg.

```
void main() {
// IF and ELSE
var Salary = 15000;
if (salary > 20000) {
        print ("You got promotion !");
} else {
        print ("You need to work hard");
}

// IF ELSE IF Ladder
var marks = 70;

if (marks >= 90 && marks < 100) {
    print ("A+ grade");
}
```

```
else if (marks >= 80 && marks < 90) {
    print ("A grade");
} else if (marks >= 70 && marks < 80) {
    print ("B grade");
} else if (marks >= 60 && marks < 70) {
    print ("C grade");
} else if (marks >= 30 && marks < 60) {
    print ("D grade");
} else if (marks >= 0 && marks < 30) {
    print ("You have failed");
} else {
    print ("Invalid marks. Try again!");
}
}
```

Conditional Expressions

* eg.

```
void main ()
{
    // 1. Condition ? exp1 : exp2
    // If Condition is true, evaluate exp1;
    // Otherwise, evaluate & return exp2:
    int a = 2;
    int b = 3;

    int smaller = a < b ? a : b;
    print ("$ smaller is smaller");
```

2 is smaller

P.T.O.

// 2. exp1 ?? exp2
// If exp1 is non-null, returns its value; Otherwise, evaluates & returns exp2.

```
String name = null;

String nameToPrint = name ?? "Guest User";
print(nameToPrint);
}
```

Guest User

Switch - Case Statements :-

★ : In dart, it is applicable only for int & String.
: bool, double are not allowed.

// It is similar to If - Else - If ladder.

Syntax :-

```
Switch (variable) {
    Case 1 :
        // Execute code here
        break;
    Case 2 :
        // Execute code here
        break;
    default :
        // Execute default
}
```

* eg.

```
void main () {
    String grade = 'A';
    switch (grade) {
        case 'A' :
            print ("Excellent, grade of A");
            break;
        case 'B' :
            print ("very Good !");
            break;
        case 'C' :
            print ("Good, but work hard");
            break;
        case 'F' :
            print ("You've failed");
            break;
        default :
            print ("Invalid grade");
    }
}
```

Loop Control Statements

→ FOR, WHILE, DO - WHILE
→ break & continue.
→ how to use 'Labels' in control flow statements ?

Iterators :-

// Syntax FOR Loop

```
for (var i=0 ; i<4 ; i++) {
        print ("Hello");
}
```

// Syntax WHILE Loop

```
var i=0;
while (i<4) {
        print ("Hello");
        i++;
}
```

// Syntax DO - WHILE Loop

```
var i = 0;
do {
        print ("Hello");
        i++;
} while (i<4) ;
```

Loops → Definite → FOR loop
Loops → Indefinite → WHILE → Condition checked first.
Indefinite → DO-WHILE → Condition checked at end.

* // for ... in loop

eg.

```
void main () {

    List planetList = ["Mercury", "Venus", "Earth", "Mars"];

    for ( String planet in planetList ) {
        print (planet);
    }
}
```

Output:
Mercury
Venus
Earth
Mars.

break keyword :- (To come out of loop after a condition is met)

*

```
void main () {

    for (int i = 1; i <= 10; i++) {
        print (i);
        if ( i == 5) {
            break;
        }
    }
}
```

Output:
1
2
3
4
5

: but 'break' works only for its parent loop & fails in case of nested loops (first breaks 1 ~~to~~ loop, & rest are continued).

: So, in case of nested loops, we use **Labels**.

We give name (or label) to any particular (parent) loop & use break for that label name.

```
void main() {

myOuterLoop : for (int i = 1; i <= 3; i++) {
// Label
    innerLoop : for (int j = 1; j <= 3; j++) {
                    print ("$i $j ");

                    if (i == 2 && j == 2) {
                        break myOuterLoop;
                    }
                }
            }
        }
```

```
1 1
1 2
1 3
2 1
2 2
- - - - -
```

continue keyword : (To skip the iteration when condition is met.)

```
void main() {
    for (int i = 1; i <= 5; i++) {
        if (i == 3) {
            continue;
        }
        print(i);
    }
}
```

1	
2	(skipped 3)
4	
5	

: We can use labels, in continue also like before.

★ Function :-

: Collection of statements grouped together to perform an operation.

```
int Area(int length, int breadth).
{
// func. body
return length * breadth;
}
```

(int → return-type (→ It is optional in dart); length, breadth → Required parameters.)

: Functions in dart are actually 'objects'.
i.e. it can be assigned to a variable or passed as parameter to other functions.

: All functions in Dart returns a value.
If no return value is specified, the funcn. will return null.

: Specifying return type is optional but is recommended as per code convention.

* (function)

```
void main() {

    findPerimeter(4, 2);

    int rectArea = getArea(10, 5);
    print("The area is $rectArea");
}

void findPerimeter(int length, int breadth) {
    int perimeter = 2 * (length + breadth);
    print("The perimeter is $perimeter");
}

int getArea(int length, int breadth) {
    int area = length * breadth;
    return area;
}
```

Expression in function.

: We can convert function into a single line of code using expression.

* (c.f. prev. eg.)

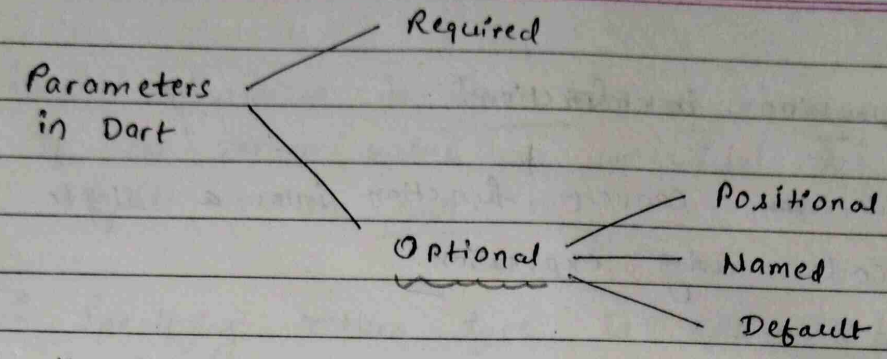
```
void main()
{
  ~~find pe~~ findPerimeter (4,2);

      int rectArea = getArea (10, 5);
      print ("The area is $rectArea");
}
```

(fat arrow)

✓ `void findPerimeter (int length, int breadth) => print ("The perimeter is ${2 * (length + breadth)}");`

✓ `int getArea (int length, int breadth) => length * breadth;`

(: Also called Short-hand Syntax)



Positional

* // 1. Required Parameters
// 2. Optional positional parameter

```
void main () {

    printCities ("New York", "New Delhi", "Sydney");
    print (" ");
    printCountries (" USA " );
}


// Required parameters
void printCities (String name1, String name2, String name3)
   {
      print ("Name 1 is $name1");
      print ("Name 2 is $name2");
      print ("Name 3 is $name3");
   }


// Optional positional parameter
void printCountries (String name1, [String name2, String name3]
                                                    ) {

    print ("Name 1 is $name1);
    Print ("Name 2 is $name2);
    Print ("Name 3 is $name3);
}
```

OUTPUT ⇒	Name 1 is New York // 1 Name 2 is New Delhi Name 3 is Sydney
	Name 1 is USA // 2 Name 2 is null Name 3 is null

Optional named parameters

: Prevent errors if there are large no. of parameters.

Syntax :-

```
int findVolume ( int length, { int breadth, int height } ) {
        return length * breadth * height;
    }
// calling the above function.
var result = findVolume ( 2, breadth : 3, height : 10 );
print (result);                    // 2 * 3 * 10 = 60

// Sequence of arguments does not matter
var result = findvolume ( 2, height : 10, breadth : 3);
print (result);
                                   // 2 * 3 * 10 = 60
```

Optional default parameter

: You can assign default values to parameters.

Syntax

```
int findVolume (int length, int breadth,
                    { int height = 10 }) {
    return length * breadth * height;
}
// Calling
var result = findVolume (2, 3);
print (result);          // 2*3*10 = 60
```

→ by default value of height will be assigned, if not entered.

```
// Overriding.
var result = findVolume (2, 3, height : 20);
print (result);
                    // Overrides the default value.
                    // 2*3*20 = 120
```

Exception Handling in Dart :-

→ What is exception handling ?
→ Hande exceptions using : TRY
: CATCH
: ON
: FINALLY
→ What is Stack Trace ?
→ How to create own custom exception handling class ?

Exception handling

When normal flow of program is disrupted & Application crashes.

: So, we need to handle these type of exception for convenience of our user.

Common exceptions in dart :-

1.) Format Exception
2.) IO Exception.
3.) Integer Division By Zero Exception
4.) Timeout
5.) Deferred Load Exception
6.) Isolate Spawn Exception , etc.

* eg.

```
// OBJECTIVES
// 1. ON clause
// 2. Catch clause with Exception object.
// 3. Catch clause with Exception object & StackTrace object.
// 4. Finally clause
// 5. Create our own custom Exception.
```

```
void main() {
```

1

```
print("Case 1");
// When we know the exception to be thrown, we use ON clause.
try {
    int result = 12 ~/ 0;
    print("The result is $result");
}
on IntegerDivisionByZeroException {
    print("Cannot divide by zero");
}
```

* This together is division for integers in dart.
~ → tilde

2

```
print(""); print("Case 2");
// When we don't know the exception, use CATCH clause.
try {
    int result = 12 ~/ 0;
    print("The result is $result");
}
catch(e) {
    print("The exception thrown is $e");
}
```

3 ●
```
print (" "); print (" Case 3");
// using STACK TRACE to know the events occured
    before Exception was thrown
try {
    int result = 12 ~/ 0;
    print (" The result is $result ");
  }                    ↑ Stack Trace object in catch
catch (e, [s]) {
    print (" The exception thrown is $e ");
    print (" STACK TRACE \n $s ");
  }
```

4 ●
```
print (" "); print (" Case 4");
// Whether there is an Exception or not, FINALLY
clause is always executed
try {
    int result = 12 ~/ 3;
    print (" The result is $result ");
  } catch (e) {
    print (" The exception thrown is $e ");
  } finally {
    print (" This is FINALLY clause & is always
            ~~thrown~~ executed. ");
  }
```

5 ●
```
print (" "); print (" Case 5 ");
// Custom Exception
try {
    depositMoney (-200);
  } catch (e) {
    print (e.errorMessage());
  }
}
```

P.T.O.

// using class & objects.

```
Class DepositException implements Exception {

    String errorMessage() {
        return "You cannot enter amount less than
                0";
    }
}

void depositMoney (int amount) {
    if (amount < 0) {

        throw new DepositException();
    }
}
```

* OUTPUTS :-

Case 1

1. Cannot divide by zero

Case 2

2. The exception thrown is IntegerDivisionByZeroException.

Case 3

3. The exception ---- " ----.

STACK TRACE

#0 int. ~/ (dart:core-patch/integers.dart:33)
#1 main (file://Users/---- :32:19)
#2 ------
#3 ------

4. Case 4

The result is 4.

This is FINALLY clause and is always executed.

5. Case 5

You cannot enter amt. less than zero.

OOP in Dart ...

→ Classes & Objects.
→ Instance variables ; Functions in class.
→ Constructors

* eg. void main () {

var student1 = student () ; // One object, student1 is
student1. id = 23 ; reference variable
student 1. name = "Peter" ;
print (" ${ student1.id } and ${ student 1. name} ") ;

Student 1. study () ;
Student 1. sleep () ;

var student 2 = student() ; // One object, student 2
Student 2. id = 45 ; is refereuce variable.
Student 2. name = "Sam" ;
print (" ${ student 2. id } and ${ student 2. name} ") ;

student 2. study ( ) ;
Student 2. sleep ( ) ;

}

// Now, we create class to define states (properties)
& behaviour of a student.

class Student {
int id = -1 ; —→ // Instance or field variable.
String name ; (default value is : - 1) (P.T.O.)
└→ // Inst. or field var., def. value is null

```
// Functions in class.
void Study () {
    print ("${this.name} is now studying");
}
                    ↳ Pointer to object
void Sleep () {
    print ("${this.name} is now sleeping");
}

}
```

OUTPUT ...

```
23 and Peter
Peter is now studying
Peter is now sleeping
45 and Sam
Sam is now studying
Sam is now sleeping
```

Constructors :-

```
// default constructor ...

Student () {
    print ("This is my default constructor");
}

// defined in class
```

∴ The code within default constructor will be first executed before any code execution, for each object.

: We don't need to declare default constructors, unless we want to write any code in it.

: Default constructors are already declared by the compiler.

// Parameterized Constructor ...

: If a constructor contains any parameter.

NOTE

Constructors don't return any value.
(Property of Constructors.)

eg.
```
Student ( int id, String name ) {
    this.id = id;
    this.name = name;
}
```
→ we used 'this' to refer object name & avoid name conflict with variable name (both are same).

// If var name is diff. (say _id & _name) then we don't need to write 'this'. We write:-
```
---- {
    _id = id;
    _name = name;
}
```

★ In same class (in Dart), we can't have both, default & parameterized constructor at the same time. It will give error.

: Another way of declaring parameterized constructor (in dart) :- (Short)

```
Student (this.id, this.name); // Parametr. Constr.
```

// Named constructor ...

```
Student.myCustomConstructor() {
    Print("This is my Custom Constructor");
}                                    // named Constr.
```

Calling →

```
var Student3 = Student.myCustomConstructor();
```

: We can also add parameters to the named constructor.

: Within same class, we can have multiple named constructors.

// Constant constructor ...

: It doesn't find use in our scope, unless we make a GUI App.

Properties of Constructors :-

: Used to create objects.
: You can initialize your instance or field variables within constructors.
: You can't have both default & parameter constr. at the same time.
: You can have as many Named constr. as you want.

: Function have return type, but constr. don't have any return type.

⇒ Classes & Objects :-

```
class Ex {
    int a;
    String b;

    Ex() {
        // code here
    }

    void fun() {
        int c;
    }
}
```

Instance variables ← int a; String b;

Local variables. ← int c;

< More on Object Oriented Dart >

→ Getters & Setters : Default & Custom.
→ Inheritance
→ Method overriding
→ Using Constructors during inheritance
→ Abstract classes & methods
→ Using Interface in Dart
→ Static methods & variables.

* Objectives : // 1. Default Getter & Setter.
// 2. Custom Getter & Setter.
// 3. Private instance variable.

```
void main() {
    var student = Student();
    student.name = "Peter"; // Calling default setter to set value
    print(student.name); // Calling default getter to get value.

    student.percentage = 438.0; // Custom setter
    print(student.percentage); // Custom getter
}

class Student {
    String name; // Instance var. with default getter & setter.
    double percent;
    void set percentage(double markSecured) {
        percent = (markSecured / 500) * 100;
    } // Instance var. with custom setter.
```

```
double get percentage {
    return percent;
}                    // instance var. with custom
}                       getter.
```

Output :-

Peter
87.6

// 3. Private instance var.

: Unlike other OOP languages, we can't use visibility modes in dart.

* - - - - - < main() code here > - - - -

```
class Student {
    String name;          // Inst var with def. get & set.
    double _percent;      // Private instance var (for its own library.)
// Inst var with custom setter
    Void Set percentage (double marksecured) =>
        _percent = (marksecured / 500) * 100;
// Inst var with custom getter.
    double get percentage => _percent;
}
```

Inheritance :-

Animals [SUPER class / Base / Parent- Class]

```
class Animal {
    String color;
    void eat() {}
}
```

Dog

```
class Dog extends Animal {
    String breed;
    void() {}
}
```

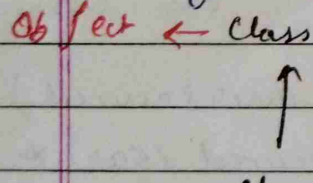
Cat

```
class Cat extends Animal {
    int age;
    void meow() {}
}
```

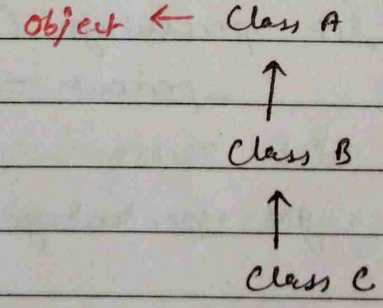
[Sub class / Child class]

Types (in Dart) :-

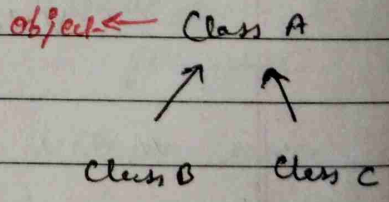
1. Single Inheritance



2. Multi-Level Inheritance



3. Hierarchial Inheritance



: Inheritance is a mechanism in which one object acquires properties of its parent class object.

: Super class of any class is Object :-

: Object class provides default implementation of :-

→ toString(), returns String representation of object.
→ hashCode Getter, returns hash code of an object.
→ Operator ==, to compare two objects.

Advantages of Inheritance :-

- Code reusability.
- Method overriding.
- Cleaner code, no repetition.

29/11/21

Method Overriding :-

```
void main() {
    var dog = Dog();
    dog.eat();
}

class Animal {
    String color;

    void eat() {
        print("Animal is eating !");
    }
}

class Dog extends Animal {
    String breed;

    void bark() {
        print("Bark !");
    }

    void eat() {                    // Func^n Overriding (child class is
        print("Dog is eating !");    given priority)

        * supr.eat();               // To also print Super class
    }                                  function
}
```

OUTPUT

```
Dog is eating
Animal is eating
```

: We can also override class variables (members) in dart.

: More priority is given to the members of child class.

eg.

```
class Parent {
    String color = "Blue";
}
class Child {
    String color = "Red";
}
```

Red

: Method overriding is a mechanism by which the child class redefines a method in its parent class.

Inheritance with Constructors :-

```
void main () {
    var dog1 = Dog ("Labrador", "Black");
    print ("");
    var dog2 = Dog ("Pug", "Brown");
    print ("");
    var dog3 = Dog.myNamedConstructor ("German Shepherd",
                                       "Black - Brown");
}

class Animal {
    String color;
    Animal (String color) {
        this.color = color;
        print ("Animal class constructor");
    }

    Animal.myAnimalNamedConstructor (String color) {
        print ("Animal class named constructor");
    }
}

class Dog extends Animal {
    String breed;
    Dog (String breed, String color) : super (color) {
        print ("Dog class constructor");
    }
}
```

NOTE

: By default, a constructor in a sub class calls the super class's no-argument constructor.

: Parent class constructor is always called before child class constructor.

: If default constructor is missing in Parent class, then you must manually call one of the constructors in super class.

Abstract Classes & Methods

// 1. Abstract Method
// 2. Abstract class

* eg.

```
void main() {

  // var shape = Shape();   // Error : Can't
                               instantiate Abstr. class

  var rectangle = Rectangle();
      rectangle.draw();

  var circle = Circle();
      circle.draw();
}
```

(P.T.O.)

```
abstract class Shape {
    // Define your Instance var if needed
    int x;
    int y;

    void draw();    // Abstract method.
    void myNormalFunction() {
        // Some code
    }
}

class Rectangle extends Shape {
    void draw() {
        print("Drawing Rectangle... ");
    }
}

class Circle extends Shape {
    void draw() {
        print("Drawing Circle... ");
    }
}
```

OUTPUT :-

Drawing Rectangle...
Drawing Circle...

Abstract method :-

: To make a method abstract, use Semicolon (;) instead of method body.

: Abstract method can only exist with Abstract class.

: You need to override Abstract methods in Sub-class.

Abstract Class :-

: Use abstract keyword to declare Abstract class.

: Abstract class can have Abstract methods, normal methods & Instance variables as well.

: The abstract class cannot be instantiated, you cannot create objects.

Interface in Dart :- (C.f. Java)

→ Dart doesn't have any special syntax to declare INTERFACE (like in Java, etc.)

→ An INTERFACE in dart is a Normal class.

→ "An INTERFACE is used when you need to concrete implementation of all its functions within its sub class."

: It is mandatory to override all methods in the implementing class.

→ You can implement multiple classes but

: You can't extend multiple classes during inheritance.

* eg. of. Interface

```
void main () {
        var tv = Television ();
        tv.volumeUp ();
        tv.volumeDown ();
    }

Class Remote {
        void volumeUp() {
                print (" Volume up from Remote");
        }
```

```
void volumeDown () {
        print (" volume down from Remote");
    }
}

Class Another_class {
    void justAnotherMethod () {
        // Code
    }
}
// Here remote acts as an Interface
   (& Another_class)
Class Television implements Remote , AnotherClass {
    void volume Up () {
        print ( " volume up in tv");
    }
    void volumeDown () {
        print (" volume down in tv");
    }

    Void justAnotherMethod () {
        print (" Some code ");
    }
}
```

OUTPUT :-

volume up in tv
volume down in tv
Some code.

Static methods & variables. :-

: Static variables & methods are also known as class variables & methods.

: Static variables are lazily initialized
→ ie. they are not initialized until they are used in program.
→ So, they consume memory only when they are used.

: Static methods has nothing to do with class object or instance.
→ You cannot use 'this' keyword within a static method.

: From a static method
→ You can only access static Method & variables.
→ But you can't access normal instance variables & methods of the class.

* eg

```
void main () {
    var Circle1 = Circle();
    // Circle1.pi ;          // 4 bytes

    var Circle2 = Circle();
    // Circle2.pi ;          // 4 bytes
                             Total 8 bytes consumed
```

// If we make static

// Circle.pi; // 4bytes

// Circle.pi; // No more memory will be allocated.

// Circle.calculateArea();
// print(Circle.pi);
// Circle.calculateArea();

print(Circle.pi);
Circle.pi = 6.14;
print(Circle.pi);

}

class Circle {

static const double pi = 3.14;
static int maxRadius = 5;

String color;

static void calculateArea() {
print("Code to calculate area of circle");

// my normal func^n. // Not allowed to call instance func^n.
// this.color // Can't use 'this' keyword.
}

void myNormalFunction() {
calculateArea();
this.color = "Red";
print(pi);
print(maxRadius);
}

}

Functional Programming in Dart (c.f. CotLan)

→ Introduction to Lambdas & Higher-Order funcn.

→ Implementing Lambdas Expressions & Higher-Order funcn.

→ Closures

What are Lambda ?

: It is a function without a name.
Also known as anonymous function or lambda.

★ [NOTE] A function in Dart is an object. ✓
→ int sum = 2;
→ String message = "Hello";
→ Function addNumbers = // some value.

```
* // Lambda functions.
void main () {
        // 1st way
    Function addTwoNumbers = (int a, int b) {
            var sum = a + b;
            print (sum);
        };

    var multiplyByFour = (int number) {
            return number * 4;
        };
```

```
// 2nd way (using short hand Syntax or
                    Fat arrow ⇒)
Function addNumbers = (int a, int b) ⇒
                    print (a+b);
var multiplyFour = (int number) ⇒ number * 4;

// Calling Lambda Functions

addTwoNumbers (2,5);
print (multiplyByFour (5));        // with return type.

addNumbers (3,7);
print (multiplyFour (10));         // with return type.

}

// Normal function
void addMyNumbers (int a, int b) {
        var sum = a + b;
        print (sum);
    }
```

Output :-

7
20
10
40

: Lambda funch are also known as anonymous function or just a Lambda.

Higher Order Function :-

: A funcn. that accepts another funcn. as a parameter.

: It can also return a function.
: Or can do both.

```
// Higher-Order function.
// How to pass funcn as parameter ?
// How to return funcn. from another funcn ?

void main(){

    // Example 1 : Passing funcn to higher-order funcn
    Function addNumbers = (a,b) => print(a+b);
        SomeOtherFunction("Hello", addNumbers);

    // Example 2 : Receiving funcn from higher-order funcn.
    var myFunc = taskToPerform();
    print(myFunc(10));  // multiplyByFour(10)
}                       // 40.

// Example 1 : Accepts function as parameter
void SomeOtherFunction(String message, Function myFunction){
                                // Higher-Order funcn.
    print(message);
    myFunction(2,4);  // addNumbers(2,4)
                      // print(a+b)   // 6
}
```

// Example 2 : Returns a function.

```
Function taskToPerform () {   // Higher-order funcn

    Function multiplyByFour = (int number) =>
                                    number * 4;

    return multiplyByFour;
}
```

Closures in Dart :-

: closure is a special function

: Within a closure, you can mutate (modify) the values of variables present in the parent scope.

: In Java 8, you are not allowed to modify parent scope variables.

*

```
// Closures
void main () {
    // Type-1
    String message = "Dart is good";

    Function showMessage = () {
        message = "Dart is awesome";
        print (message);
    };
    showMessage ();
```

```
// Type 2

Function talk = () {
    String msg = "Hi";

    Function say = () {
        msg = "Hello";
        print (msg);
    };
    return say;
};

Function speak = talk();

}
```

Output :-

→ Dart is awesome
→ Hello

Collections in Dart

→ List : Fixed - length list
: Growable list

→ Set : HashSet

→ maps : Hash map

List in dart :-

Types — Fixed - length list
(length once defined cannot be changed)

— Growable list
(length is dynamic)

(Fixed length list)

```
// Fixed length list

void main () {

List <int> numberList = List (5);

numberList [0] = 73;      // Insert operation
numberList [1] = 64;
~~numberList [2]~~
numberList [3] = 21;
numberList [4] = 12;
```

```
numberList[0] = 99;      // update opn.
numberList[1] = null;    // delete opn.

print(numberList[0]);

print("\n");

// for (int element in numberList) {
//     print(element);      // using Individual
//                             Element (objects)
// }

numberList.forEach((element) => print(element));
                         // using Lambda

// for (int i = 0; i < numberList.length; i++) {
//     print(numberList[i]);   // using Index
// }

}
```

Output :-

99
null
null
21
12

(Growable - list)

* Growable - list

```
void main () {

    List <int> numberList = List (); // Growable list

    numberList.add (73);
    numberList.add (64);
    numberList.add (21);
    numberList.add (12);

    numberList [0] = 99 ;   // update operation
    numberList [1] = null ; // Delete operation.

// Other than these, we can also perform...

    numberList.remove (99);
    numbeList.    add (24);
    numbeList . removeAt (3);
    numberList.clear ();

    ---------- <rest all are same
                as in fixed list ←>

// Another method to define growable list

    List <String> Countries = ["USA", "INDIA", "JAPAN"];
```

Set :-

: Unordered collection of unique items
→ It doesn't contain duplicate elements.

: You can't get elements by INDEX, since the items are unordered.

HashSet :-

: Implementation of unordered set

: It is based on hash-table based set-implementation.

* // Set → Unordered collection [c.f. prev. eg.]
→ All elements are unique

```
void main () {
    Set<String> Countries = Set.from (["USA", "INDIA",
                                       "JAPAN"]);
                        // Method 1: From a List
    Countries.add ("Nepal");
    Countries.add ("Russia");

    Set<int> numberSet = Set();   // Method 2: using Constructor
    numberSet.add(73);   // Insert operations
    numberSet.add(64);
    numberSet.add(21);
    numberSet.add(12);
```

Ctrl + T (to select each particular word)


```
numberSet.add(73);  // Dublicate entries are ignored
numberSet.add(73);  // ignored

numberSet.contains(73);  // returns 'true' if element found
numberset.remove(64);  // returns 'true' " " & deleted
numberSet.isEmpty;  // returns 'true' is Set is empty
numberSet.length;  // returns no. of elements in Set
// numberSet.clear();  // clears all elements
```

(rest all codes are same as List.)

... Map & Hashmap in Dart ...

//MAP

: It is unordered collection of key-value pair

: Key-value can be of any object type.
→ Each KEY in a map should be unique.
→ The value can be repeated.

: Commonly called as hash or dictionary.

: Size of map is not fixed ; it can increase or decrease as per the nos. of elements.

// HASHMAP

→ Implementation of map.
→ Based on hash-table.

```
* // 1. maps
  // → KEY has to be unique.
  // → VALUE can be duplicate

Void main() {

                          // Method : Using Constructor
    Map<String, String> fruits = Map();
          Key     value

    fruits["apple"] = "red";
           Key       Value.
```

```
fruits ["banana"] = "yellow";
fruits ["guava"] = "green";

                    // method : using Literal.
Map <String, int> countryDialingCode = {
                    "USA" : 1,
                     Key    value
                    "INDIA" : 91.
                  };

//Methods in maps
// fruits.containKey("apple"); → returns true if KEY
                                  is present in map.
// fruits.update("apple", (value) => "green");
     ↳ update the value for the given key
// fruits.remove("apple");  → removes key & its
                              value & returns the value
// fruits.isEmpty();  → returns true if map
                        is empty.
// fruits.length();  → returns no. of elements
                       in map
// fruits.clear();  → Deletes all the elements.

for (String value in fruits.values) {
        print(value);          // Prints all values.
}
for (String key in fruits.keys) {
        print(keys);           // Prints all keys
}
```

(P.T.O)

```
print("\n");

fruits.forEach((key, value) => print("key : $key and
                value : $value"));
                // using Lambda.
}
```

OUTPUT :-

```
red
Yellow
Green

Apple
banana
guava

key: apple and value : red
---  .  --  ---
---  .  ---  ---
```

Callable Class :-

: When dart class is called like a function.

.

: We use call() function for this purpose.

* // Callable class.

```
void main () {
    var personOne = Person ();
    var msg = personOne (25, "Peter");
    print (msg);
}
class Person {
    String call (int age, String name) {
        return "Name is $name and age is $age";
    }
}
```

OUTPUT :-

[Name is Peter and age is 25]

... End of Course on Dart ...