Dart Functions & Object-Oriented Programming

Created By:-

The EasyLearn Academy

Call us for training on +91 9662512857

Introduction to function

- A function is a group of statements used(called) to perform a specific task.
- A function normally has
  - Return value
  - Name
  - Body (group of statement)
  - Arguments(inputs)
- Functions are created when certain statements are repeatedly occurring in the program.
- Functions make it easy to divide the complex program into smaller sub-groups.
- Functions organize the program into logical blocks of code.
- Once defined, functions may be used/called to executed code.
- This makes the code reusable.
- Moreover, functions make it easy to read and maintain the program's code.
- **Each function should perform only single task.**

Syntax to create function

```
return_type function_name ( parameters ) {
   // Body of function
   return value;
}
```

Example

```
int getSquare(int number)
{
  int result = number * number;    // function body
  return result;  // returning value result
}

   void main()
{
  var output = getSquare(10); //  // Calling the function
  print(output);
}
```

Function without any return value

```
import 'dart:io';
  void PrintLine(String Decorater, int HowManyTimes) {
  for (int i = 0; i < HowManyTimes; i++) {
    stdout.write(Decorater);
  }
  print("");
}
  void PrintMessage(String Message) {
  PrintLine('*', 100);
  print(Message);
  PrintLine('^', 100);
}
void main() {
  print("What is your name");
  String Message = stdin.readLineSync().toString();
  PrintMessage(Message);
}
```

Arrow/Lambda function

- The arrow function is a function that can have only one line of code, arrow function has no braces , it has an arrow.
- It is sometimes called function returning expression.

Syntax

```
Return-type function-name(arguments) => expression
```

Example

```
void printName(String name) => print(name);
```

Anonymous function

- An anonymous function in Dart which do not have names associated with it.
- An anonymous function can have zero or more parameters with optional type annotations.
- This type of function is known as an anonymous function, lambda, or closure.
- An anonymous function consists of self-contained blocks of code and that can be passed around in our code as a function parameter.
- In Dart we can assign an anonymous function to constants or variables, later we can access or retrieve the value of closure based on our requirements:

```
void main() {
  var Fruits = ["Apple", "Banana", "Mango", "Pinapple"];
  int position = 0;
  Fruits.forEach((item) {
    print(item + " is at " + position.toString());
    position++;
  });
}
```

Example 2 (filter list using for each)

```
void main()
{
  var numbers = [1, 2, 3, 4, 5, 6];
  var odd = [];
  var even = [];
  numbers.forEach((CurrentNumber) {
    if (CurrentNumber % 2 == 0)
      even.add(CurrentNumber);
    else
      odd.add(CurrentNumber);
  });
  print(odd);
  print(even);
}
```

Optional Positional Parameter

- In Dart we can create functions which may have some argument optional.
- When calling function we may or we may not provide optional arguments.
- If we provide optional argument, then functional will use value provided by us otherwise it use default value.
- In Dart, there are two ways to specify optional parameters: they can be either positional or named.
- Optional positional parameters are passed in brackets []
- Optional named parameters are passed in braces {}

example

```
//optional positional argument
int getArea(int height, [int width = 0]) {
  if (width != 0)
    return height * width;
  else
    return height * height;
}
//Optional named parameters
int getVolume(int height, {int width = 1, int length = 1}) {
  return height * width * length;
}
void main() {
  // Calling the function with optional parameter
  print("Calling the function with optional parameter:");
  print("area = " + getArea(10).toString());
  print("area = " + getArea(10,20).toString());
  print("Calling the function with Optional Named parameter:");
  print("volume = " + getVolume(10,width: 10).toString());
  print("volume = " + getVolume(10,width: 10,length: 10).toString());
}
```

Object Oriented Programming (OOP)

- A dart is a programming language that support all the concepts of object oriented programming such as:
  - **Classes**
  - **Objects**
  - **Inheritance**
  - **Polymorphism**
  - **Interface**
  - **Abstract class**
- Let us understand each of the concept in detail.

class

- Class in one kind of named template/group/unit name which has variables, methods.
- One can create class for any concept he wants to implements in real world.
- Because class is created by user it is called user defined data type.
- Once class is created one can create class type variables to store detail in the variable defined inside class.

```
Syntax
class className {
    <fields>
    <getter/setter>
    <constructor>
    <functions>
}
```

Object

- Object is the most important part of OOP.
- Class type variables are called objects.
- One can create any number of class type variable.
- Each object variable in program will occupy different memory location and has different value for each and every object of specific class.
- That is why variables inside class are called instance variables.
- Instance variables are created with object creation and gets destroyed with object destruction.
- Default scope of instance variable is public means one can directly change instance variable value outside the class using object
- To create private instance variable use _ underscore before variable name.
- However if one use objectname._variablename one can still change private variable value.
- In Dart, the privacy is at library level rather than class level. It means other classes and functions in the same library still have the access. So, a data member is either public (if not preceded by _) or private (if preceded by _)

constructor

- Constructor is special member function of class which is automatically called when one create object of class type.
- Constructor do not return any value not even void.
- Constructor name is always same as name of class.
- It is used to initialize instance variable of the class.
- There is no need to overload constructor, as one can use optional argument concept to create constructor with different number of arguments.
- However one can create multiple constructor using **named constructors.**

getter and setter methods

- **Getters** and **Setters**, are used to initialize and retrieve the values of class fields respectively.
- Getters or accessors are defined using the **get** keyword.
- Setters or mutators are defined using the **set** keyword.
- A default getter/setter is associated with every class.
- However, the default ones can be overridden by explicitly defining a setter/ getter.
- A getter has no parameters and returns a value, and the setter has one parameter and does not return a value.

```
Syntax: Defining a getter
Return_type  get identifier
{
}
Syntax: Defining a setter
set identifier
{
}
```

Example 1

```
class book {
  String BookName = "The Atomic Habit";
  int Price = 100;
  double Weight = 1.2;
  void ShowDetail() {
    print("Book Name " + BookName);
    print("Price " + Price);
    print("Weight " + Weight);
  }
}

   void main() {
//how to create object
//class object = new class();
  book b1 = new book();
//how to call method of class using object
  b1.ShowDetail();
}
```

Example 2

```
class Rectangle {
  int height = 0, width = 0; //constructor
  Rectangle(int height, [int width = 1]) {
    print("normal constructor is called...");
    this.height = height;   this.width = width;
  }
  int getArea() {
    return height * width;
  }
  void display() {
    print("Height = " + this.height.toString());
    print("Width = " + this.width.toString());
  }
}
void main() {
  Rectangle r1 = new Rectangle(10, 20);
  int area = r1.getArea();
  print("area = " + area.toString());
  r1.display();
  Rectangle r2 = new Rectangle(15);
  area = r2.getArea();
  print("area = " + area.toString());
  r2.display();
}
```

Example 3

```
class Circle {
  int radius = 0;
  Circle.namedConst(int radius) {
    print("Named constructor is called..");   this.radius = radius;
  }
  //constructor
  Circle() {
    print("normal constructor is called..."); this.radius = 1;
  }
  double getArea() {
    return radius * radius * 3.14;
  }
  void display() {
    print("Radius = " + this.radius.toString());
  }
}
void main() {
  Circle c1 = new Circle.namedConst(100); //calling named constructor
  double area = c1.getArea();
  print("area = " + area.toString());
  Circle c2 = new Circle();
  area = c2.getArea();
  print("area = " + area.toString());
}
```

Example 4

```
class Person {
  String? first; //public
  String? _last; //private
}
void main() {
  Person a = new Person();
  a.first = 'ankit';
  //a.last = 'patel'; it won't work
  a._last = 'patel';
  print('${a.first}  ${a._last}');
}
```

Example 5

```
class Triangle {
  int _base = 0, _height = 0;
  int get base { return _base;}
  int get height { return _height;}
  set Base(int base) { _base = base;}
  set Height(int height) { _height = height;}
  double getArea() {
    return _height * _base * 0.5;
  }
}
void main() {
  Triangle t1 = new Triangle();
  t1.Base = 10;  t1.Height = 10;
  print("Base = " + t1.base.toString());
  print("Height = " + t1.height.toString());
  print("area = " + t1.getArea().toString());
  t1.Height = 20;   t1.Base = 100;
  print("now it has");
  print("Base = " + t1.base.toString());
  print("Height = " + t1.height.toString());
  print("area = " + t1.getArea().toString());
}
```

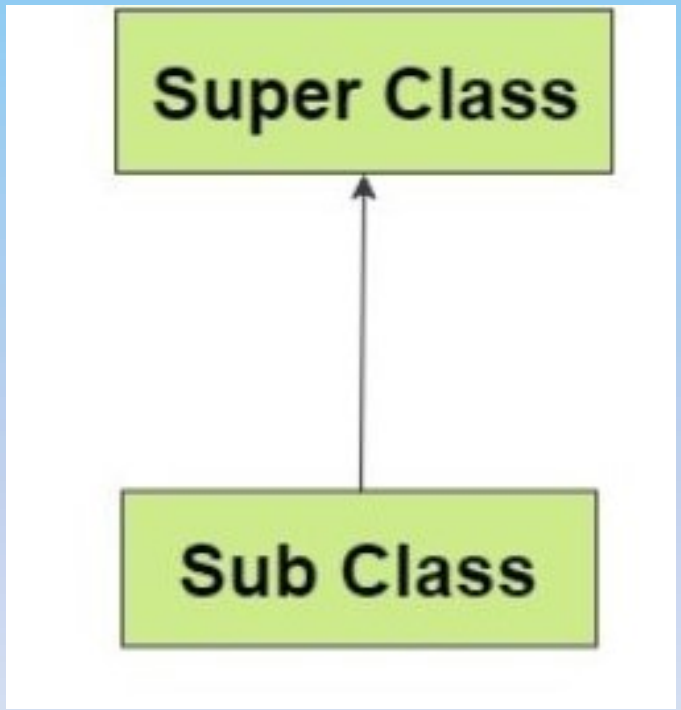
inheritance

- Inheritance is very important concept in OOP. It is used to create new class using existing class.
- Newly created class is called derived/child class/sub class.
- While existing class is called parent/super/base class.
- Because of inheritance derived class can directly access/change instance variable of parent class as well as can call parent class method.
- Inheritance increase reusability of code and decrease complexity of code.
- There are many type of inheritance

1. Single level inheritance
2. Multiple inheritance
3. Multi-level inheritance
4. Hierarchical inheritance
5. Hybrid inheritance

Single level inheritance

- When we create one new class using one existing class it is called single level inheritance.
- There are always only two class in single level inheritance.

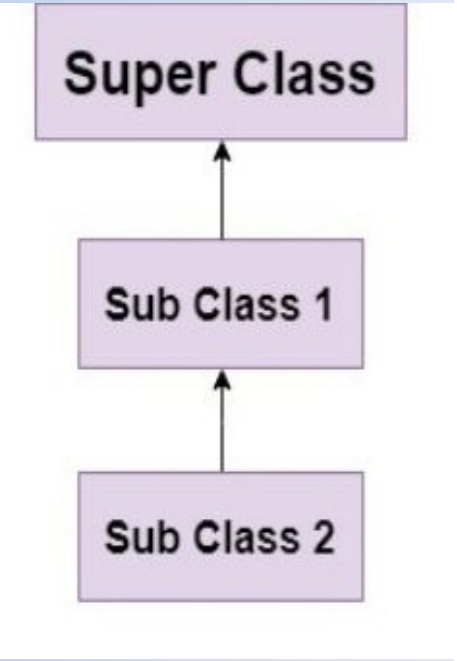


Single level inheritance

```
class person {
  void walk() {
    print("I can walk");
  }
  void talk() {
    print("I can talk");
  }
}
class student extends person {
  void read() {
    print("I can read");
  }
  void write() {
    print("I can write");
  }
  void WhatICanDo() {
    super.walk();
    super.talk();
    this.read();
    this.write();
  }
}
void main() {
  student s1 = new student();
  s1.WhatICanDo();
}
```

Multilevel inheritance

- When we create new class from already derived class it is called multilevel inheritance.
- In multilevel inheritance there are always at least 3 class involved. Each of the class belongs to different level.
- It is mostly used type of inheritance.
- There is no level restriction on multilevel inheritance means any number of class can be created using multilevel inheritance

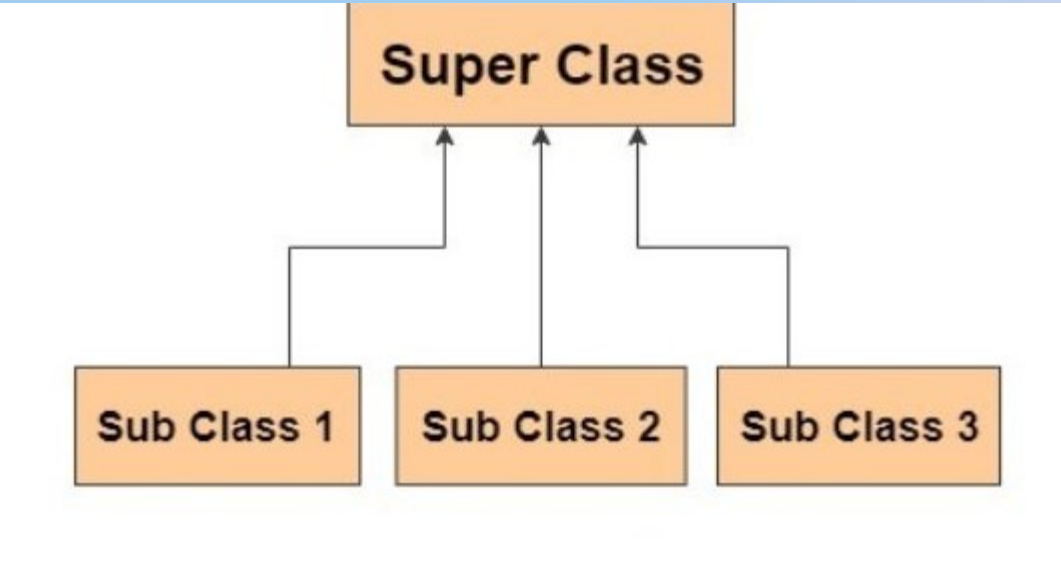


multilevel inheritance

```
class person {
  void walk() {
    print("I can can walk");
  }
  void talk() {
    print("I can talk");
  }
}
class student extends person {
  void read() {
    print("I can read");
  }
  void write() {
    print("I can write");
  }
  void WhatICanDo() {
    super.walk();
    super.talk();
    this.read();
    this.write();
  }
}
class developer extends student {
  void code() { print("I can write code.");}
  void debug() { print("I can debug code."); }
  void WhatICanDo() {
    super.WhatICanDo();
    this.code();
    this.debug();
  }
}
void main() {
  developer d1 = new developer();
  d1.WhatICanDo();
}
```

Hierarchical inheritance

- When we create two or more class from one existing class then it is called Hierarchical inheritance.
- In Hierarchical inheritance multiple derived class has same parent.
- Parent class has most general methods while derived class has more specific methods.

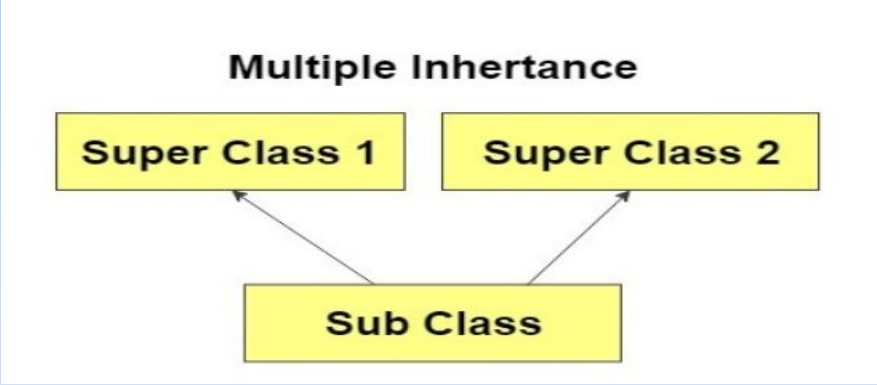


Hierarchical inheritance

```
class person {
  void walk() {
    print("I can can walk");
  }
  void talk() {
    print("I can talk");
  }
}
class student extends person {
  void read() {print("I can read"); }
  void write() { print("I can write");}
  void WhatICanDo() {
    super.walk();
    super.talk();
    this.read();
    this.write();
  }
}
class teacher extends person {
  void teach() {  print("I can teach specific subject.");}
  void help() {print("I can help you in solving problems.");}
  void WhatICanDo() {
    super.walk();
    super.talk();
    this.teach();
    this.help();
  }
}
void main() {
  student s1 = new student();
  s1.WhatICanDo();
  print("-----------------------------------------------");
  teacher t1 = new teacher();
  t1.WhatICanDo();
}
```

Multiple inheritance

- When we create one new class using two or more existing class then it is called multiple inheritance.
- It means there are always two or more parent class of newly created class in multiple inheritance.
- **Dart doesn't support multiple inheritance because it creates complexity in the program.**
- However we can do it using the concept of interface



interface

- The interface in the dart provides class which has methods that child class must overrides means redefined.
- If we don't override method then dart will generate compile time error.
- Dart doesn't have any direct way to create interface class, we have to make use of **implements** keyword to do so.
- It is also possible to create object of inertface class.

Multiple inheritance

```
class person {
  void walk() {
    print("I can can walk");
  }
  void talk() {
    print("I can talk");
  }
}
class animal {
  void eat() {}
  void sleep() {}
}
class student extends person implements animal {
  void read() {
    print("I can read");
  }
  void write() {
    print("I can write");
  }
  void eat() {
    print("I can eat");
  }
  void sleep() {
    print("I can sleep");
  }
```

```
void WhatICanDo() {
    super.walk();
    super.talk();
    this.read();
    this.write();
    this.eat();
    this.sleep();
  }
}
void main() {
  student s1 = new student();
  s1.WhatICanDo();
  print("----------------------------
---------------------");
}
```

Calling parent class constructor.

- When we inherit class which has constructor, we have to take understand how parent class constructor is called.
- Constructor in parent class is known as super constructor which can be
  - Implicit super constructor (automatically called)
  - Explicit super constructor (must be called manually)
- When parent class has constructor without any argument then it will be called automatically from child class constructor which is called implicit super constructor.
- When parent class has constructor with argument, then child class constructor must call parent class constructor using super keyword which is called explicit constructor.

Implicit super constructor.

```
class person {
  String name = '';   String surname = '';
  person() {
    name = 'Ankit';
    surname = 'Patel';
    print("person class constructor is called....");
  }
  void DisplayPerson() {
    print("Name = " + name + " Surname = " + surname);
  }
}
class student extends person {
  int rollno = 0;
  int standard = 0;
  student() {
    rollno = 1;
    standard = 12;
    print("student class constructor is called....");
  }
  void read() {
    print("I can read");
  }
  void DisplayStudent() {
    super.DisplayPerson();
    print(
        "rollno = " + rollno.toString() + " standard = " + standard.toString());
  }
}
void main() {
  student s1 = new student();
  s1.DisplayStudent();
}
```

Explicit super constructor.

```
class person {
  String name = '';   String surname = '';
  person() {
    name = 'Ankit';
    surname = 'Patel';
    print("person class constructor is called....");
  }
  void DisplayPerson() {
    print("Name = " + name + " Surname = " + surname);
  }
}
class student extends person {
  int rollno = 0;
  int standard = 0;
  student() {
    rollno = 1;
    standard = 12;
    print("student class constructor is called....");
  }
  void read() {
    print("I can read");
  }
  void DisplayStudent() {
    super.DisplayPerson();
    print(
        "rollno = " + rollno.toString() + " standard = " + standard.toString());
  }
}
void main() {
  student s1 = new student();
  s1.DisplayStudent();
}
```

What is abstract class?

- Abstract class is class whose object can not be created. It is mainly used create another class using inheritance.
- If we try to created of object of abstract class, we will get error.
- Abstract class can contain one or more abstract method.
- When abstract class is inherited, derived class must override all the abstract method otherwise we will get error.

Example of abstract class

```
abstract class shape {
  double getArea();
  void display();
}
class Square extends shape {
  double length = 0.0, width = 0.0;
  Square(double length, double width) {
    this.length = length;
    this.width = width;
  }
  @override
  double getArea() {
    return length * width;
  }
  @override
  void display() {
    print("length = " +
        this.length.toString() +
        " width = " +
        this.width.toString());
  }
}
main() {
  Square s1 = new Square(10.0,20.0);
  print('area = ' + s1.getArea().toString());
  s1.display();
}
```

How to create List of Object?

```
class Book {
  String name = '';
  int price = 0;
  Book(name, price) {
    this.name = name;            this.price = price;
  }
  void display() {    print("Name = " + this.name + " Price = " + price.toString());}
}
void main() {
  //List<class> list = new List<Class>.filled(initialsize,DefaultValue,isGrowable)
  List<Book> BookList =
      new List<Book>.filled(0,Book('any',1), growable: true);
  Book b1 = new Book('Learn Dart', 100);
  BookList.add(b1);
  Book b2 = new Book('Learn code', 300);
  BookList.add(b2);
  Book b3 = new Book('Learn Flutter', 200);
  BookList.add(b3);
  for (int i = 0; i < BookList.length; i++) {
    BookList[i].display();
  }
}
 //using for in loop
  for (var CurrentBook in BookList) {
    CurrentBook.display();
  }
```