

INDEX

1. What is Dart ?.....	3
2. Data Types.....	3 - 5
• Numbers	
• Strings	
• Boolean	
• List and Map	
3. Variables.....	5 - 7
• Dynamic,const,final variables	
4. Conditional Statements.....	7 - 12
• If	
• If.....else	
• Else if ladder	
• Nested if	
5. Looping Statements.....	12 - 14
• While	
• Do....while	
• for	
6. functions.....	15 - 21
• Passing arguments to function	
- Actual parameters	
- Formal parameters	
- Positional parameters	
• No arguments and return type	
• With arguments and return type	
• Recursion	
7. OOPs.....	21 - 22
• Class	
• Object	
• Inheritance	

- Polymorphism
- abstract

8. Dart Classes and Objects..... 23 - 25



DART

What is Dart Programming

Dart is an open-source, general-purpose, object-oriented programming language with C-style syntax developed by **Google in 2011**. The purpose of Dart programming is to create a frontend user interface for the web and mobile apps. It is under active development, compiled to native machine code for building mobile apps, inspired by other programming languages such as Java, JavaScript, C#, and is Strongly Typed. Since Dart is a compiled language so you cannot execute your code directly; instead, the compiler parses it and transfers it into machine code.

Data Types

Dart is a Strongly Typed programming language. It means, each value you use in your programming language has a type either string or number and must be known when the code is compiled. Here, we are going to discuss the most common basic data types used in the Dart programming language.

One of the most fundamental characteristics of a programming language is the set of data types it supports. These are the types of values that can be represented and manipulated in a programming language.

The Dart language supports the following types–

- Numbers
- Strings
- Booleans
- Lists
- Maps

Numbers

Numbers in Dart are used to represent numeric literals. The Number Dart come in two flavours –

- **Integer** – Integer values represent non-fractional values, i.e., numeric values without a decimal point. For example, the value "10" is an integer. Integer literals are represented using the **int** keyword.
- **Double** – Dart also supports fractional numeric values i.e., values with decimal points. The Double data type in Dart represents a 64-bit (double-precision) floating-point number. For example, the value "10.10". The keyword **double** is used to represent floating point literals.

Strings

Strings represent a sequence of characters. For instance, if you were to store some data like name, address etc. the string data type should be used. A Dart string is a sequence of UTF-16 code units. **Runes** are used to represent a sequence of UTF-32 code units.

The keyword **String** is used to represent string literals. String values are embedded in either single or double quotes.

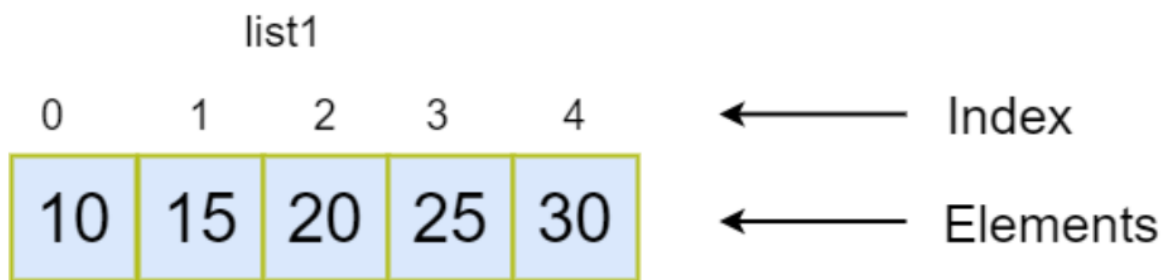
Boolean

The Boolean data type represents Boolean values true and false. Dart uses the **bool** keyword to represent a Boolean value.

List and Map

The data types list and map are used to represent a collection of objects. A **List** is an ordered group of objects. The List data type in Dart is synonymous to the concept of an array in other programming languages. The **Map** data type represents a set of values as key-value pairs. The **dart: core** library enables creation and manipulation of these collections through the predefined List and Map classes respectively.

```
var list1 = [10, 15, 20, 25, 25];
```



Syntax - Initialize the fixed size list element

1. `list_name[index] = value;`

Example - 1

```
void main() {
    var list1 = [10,11,12,13,14,15];
    print(list1);
}
```

Output:

[10, 11, 12, 13, 14, 15]

Adding elements

```
void main() {
    var list1 = [10,11,12,13,14,15];
    list1.add(20);
    list1.add(21);
    print(list1);
}
```

Output:

[10, 11, 12, 13, 14, 15, 20, 21]

List Properties

Property	Description
first	It returns the first element case.
isEmpty	It returns true if the list is empty.
isNotEmpty	It returns true if the list has at least one element.
length	It returns the length of the list.
last	It returns the last element of the list.
reversed	It returns a list in reverse order.
Single	It checks if the list has only one element and returns it.



Inserting Element into List

Dart provides four methods which are used to insert the elements into the lists. These methods are given below.

- `add()`
- `addAll()`
- `insert()`
- `insertAll()`

The add() Method

This method is used to insert the specified value at the end of the list. It can add one element at a time and returns the modified list object. Let's understand the following example -

Syntax -

```
list_name.add(element);
```

```
void main() {  
    var odd_list = [1,3,5,7,9];  
    print(odd_list);  
    odd_list.add(11);  
    print(odd_list);  
}
```

Output:

```
[1, 3, 5, 7, 9]  
[1, 3, 5, 7, 9, 11]
```

The addAll() Method

```
void main() {  
    var odd_list = [1,3,5,7,9]  
    print(odd_list);  
    odd_list.addAll([11,13,14]);  
    print(odd_list);  
}
```

Output:

```
[1, 3, 5, 7, 9]  
[1, 3, 5, 7, 9, 11, 13, 14]
```

The insert() Method

```
void main(){  
    List lst = [3,4,2,5];  
    print(lst);  
    lst.insert(2,10);  
    print(lst);  
}
```

output

[3, 4, 2, 5]

[3, 4, 10, 2, 5]

The insertAll() Method

```
void main(){  
    List lst = [3,4,2,5];  
    print(lst);  
    lst.insertAll(0,[6,7,10,9]);  
    print(lst);  
}
```

output

[3, 4, 2, 5]

[6, 7, 10, 9, 3, 4, 2, 5]

Removing List Elements

The Dart provides following functions to remove the list elements.

- remove()

- `removeAt()`
- `removeLast()`
- `removeRange()`

Variables

A variable is “a named space in the memory” that stores values. In other words, it acts as a container for values in a program. Variable names are called identifiers.

Following are the naming rules for an identifier –

- Identifiers cannot be keywords.
- Identifiers can contain alphabets and numbers.
- Identifiers cannot contain spaces and special characters, except the underscore (`_`) and the dollar (`$`) sign.
- Variable names cannot begin with a number.

Syntax

A variable must be declared before it is used. Dart uses the `var` keyword to achieve the same. The syntax for declaring a variable is as given below –

```
var name = 'Smith';
```

The dynamic keyword

Variables declared without a static type are implicitly declared as dynamic. Variables can be also declared using the dynamic keyword in place of the `var` keyword.

The following example illustrates the same.

```
void main () {  
    dynamic x = "tom";  
    print(x);  
}
```

Output

Tom

Final and Const

The **final** and **const** keywords are used to declare constants. Dart prevents modifying the values of a variable declared using the final or const keyword. These keywords can be used in conjunction with the variable's data type or instead of the **var** keyword.

The **const** keyword is used to represent a compile-time constant. Variables declared using the **const** keyword are implicitly final.

Syntax: final Keyword

final variable_name

OR

final data_type variable_name

Syntax: const Keyword

const variable_name

OR

const data_type variable_name

Example – final Keyword

```
void main () {  
  final val1 = 12;  
  print(val1);  
}
```

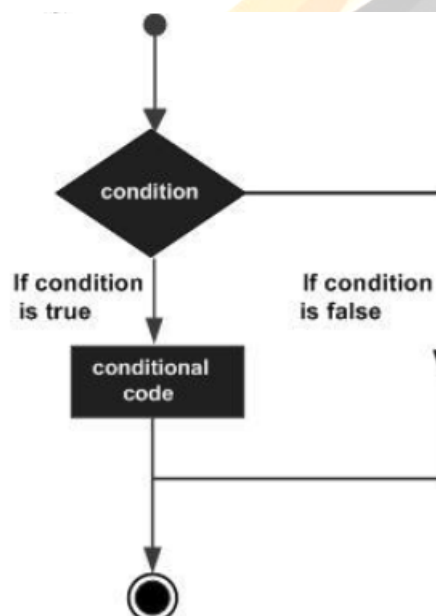
Output

12

Conditional Statements and Loop

Conditional/Decision-making statements:

A conditional/decision-making construct evaluates a condition before the instructions are executed.



Following are the conditional statements in Dart:

1. If statement:-

This type of statements simply checks the condition and if it is true the statements within it is executed but if it is not then the statements are simply ignored in the code.

Syntax:

```
if ( condition ){  
    // body of if  
}
```

Example:

```
void main()  
{  
  
    int gfg = 10;  
  
    // Condition is true  
  
    if (gfg > 3) {  
  
        // This will be printed  
        print("Condition is true");  
    }  
}
```

Output:

Condition is true

2.Else-if statement:-

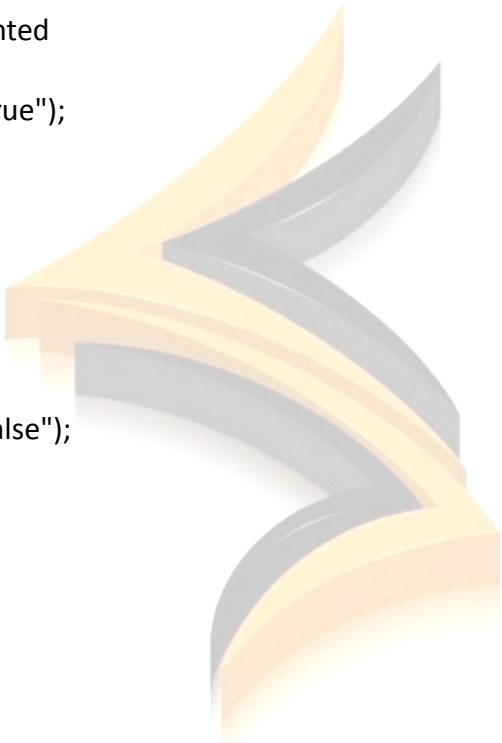
This type of statement simply checks the condition and if it is true, the statements within is executed but if not then else statements are executed.

Syntax:

```
if ( condition ){  
    // body of if  
}  
  
else {  
    // body of else  
}
```

Example:

```
void main()
{
    int gfg = 10;
    // Condition is false
    if (gfg > 30) {
        // This will not be printed
        print("Condition is true");
    }
    else {
        // This will be printed
        print("Condition is false");
    }
}
```



Output:

Condition is false

3.Else if ladder:-

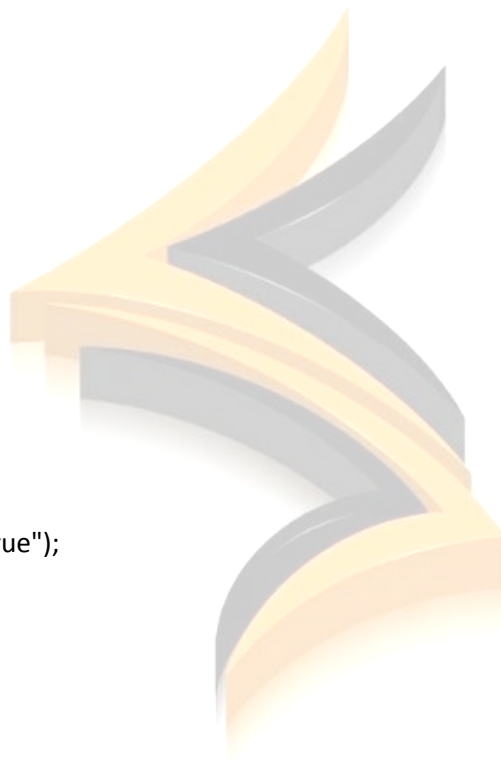
This type of statement simply checks the condition and if it is true the statements within it are executed but if it is not then other if conditions are checked, if they are true then they are executed and if not then the other if conditions are checked. This process is continued until the ladder is completed.

Syntax:

```
if ( condition1 ){  
    // body of if  
}  
else if ( condition2 ){  
    // body of if  
}  
.  
.  
.  
else {  
    // statement  
}
```

Example:

```
void main(){  
  
    int gfg = 10;  
  
    if (gfg < 9) {  
        print("Condition 1 is true");  
        gfg++;  
    }  
  
    else if (gfg < 10) {  
        print("Condition 2 is true");  
    }  
  
    else if (gfg >= 10) {  
        print("Condition 3 is true");  
    }  
  
    else {  
        print("All the conditions are false");  
    }  
}
```



```
}  
  
}
```

Output:

Condition 3 is true

3.Nested if statement: -

This type of statements checks the condition and if it is true then the if statement inside it checks its condition and if it is true then the statements are executed otherwise else statement is executed.

Syntax:

```
if ( condition1 ){  
    if ( condition2 ){  
        // Body of if  
    }  
    else {  
        // Body of else  
    }  
}
```

Example:

```
void main()  
{  
  
    int a = 10;  
  
    if (a > 9) {  
  
        a++;  
  
        if (a < 10) {
```

```
    print("Condition 2 is true");  
  }  
  
  else {  
    print("All the conditions are false");  
  }  
}  
}
```

Output:

All the conditions are false

Looping statements

A looping statement in Dart or any other programming language is used to repeat a particular set of commands until certain conditions are not completed. There are different ways to do so. They are:

- for loop
- while loop
- do-while loop

For loop:-

For loop in Dart is similar to that in Java and also the flow of execution is the same as that in Java.

Syntax:

```
for(initialization; condition; text expression){  
  // Body of the loop  
}
```

Example:


```
// Printing Flutter 5 times
```

```
void main()
{
for (int i = 0; i < 5; i++) {
    print('Flutter');
}
}
```

Output

Flutter

Flutter

Flutter

Flutter

Flutter

While loop:-

The body of the loop will run until and unless the condition is true.

Syntax:

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

```
void main()
{
    Var a = 4;
    int i = 1;
    while(i <= a) {
        print('Hello Flutter');
        i++;
    }
}
```

```
}
```

Output:

Hello Flutter

Hello Flutter

Hello Flutter

Hello Flutter

Do-while:-

The body of the loop will be executed first and then the condition is tested.

Syntax:

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

```
void main()
```

```
{
```

```
    Var a = 4;
```

```
    int i = 1;
```

```
    do{
```

```
        print('Hello Flutter');
```

```
        i++;
```

```
    }while(i <= a)
```

```
}
```

Output:

Hello Flutter

Hello Flutter

Hello Flutter

Hello Flutter

FUNCTIONS

The function is a set of statements that take inputs, do some specific computation, and produce output. Functions are created when certain statements are repeatedly occurring in the program and a function is created to replace them. Functions make it easy to divide the complex program into smaller sub-groups and increase the code reusability of the program.

Defining a Function:

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

In the above syntax:

- `function_name` defines the name of the function.
- `return_type` defines the datatype in which output is going to come.
- `return value` defines the value to be returned from the function.

The function is called as:

Syntax:

```
function_name (argument_list);
```

In the above syntax:

- `function_name` defines the name of the function.
- `argument list` is the list of the parameters that the function requires.

Example – 1

```
int mul(int a, int b){
```

```
int c;  
    c = a+b;  
  
print("The sum is:${c}");  
  
}
```

Example-2

```
int add(int a, int b)  
  
{  
    // Creating function  
    int result = a + b;  
    // returning value result  
    return result;  
}
```

```
void main()  
{  
    // Calling the function  
    var output = add(10, 20);  
  
    // Printing output  
    print(output);  
}
```

Output:

30

Passing Arguments to Function:

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

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

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

- Named and positional parameters:

The **named parameters** are referenced by name and declared within curly braces{ }.

The named parameters can be utilized in a different order from their declaration.

The **positional parameters** are declared within a square bracket [].

- No arguments and No return type:

Basically in this function, we do not give any argument and expect no return type.

Syntax

```
void function_name(){  
    // Statements  
}
```

Example:

```
void Name(){  
    print("Dart");  
}  
main(){  
    Name();  
}
```

Output:

Dart

- With arguments and No return type:

In this function, we do not give any argument but expect a return type.

Syntax

```
function_name(args1, args2, ...argsN){  
  // Statements  
}
```

Example:

```
void sum(int a, int b)  
{  
  
  Print("sum=${a+b}");  
}  
void main()  
{  
  sum(20,30);  
}
```

Output:

Sum=50



No arguments and return type:

The function has no arguments but rather it has a return type.

Syntax

```
return_type function_name() {  
  // Statements  
  return value;  
}
```

Example:

```
int MyAge(){  
  int age = 20;
```

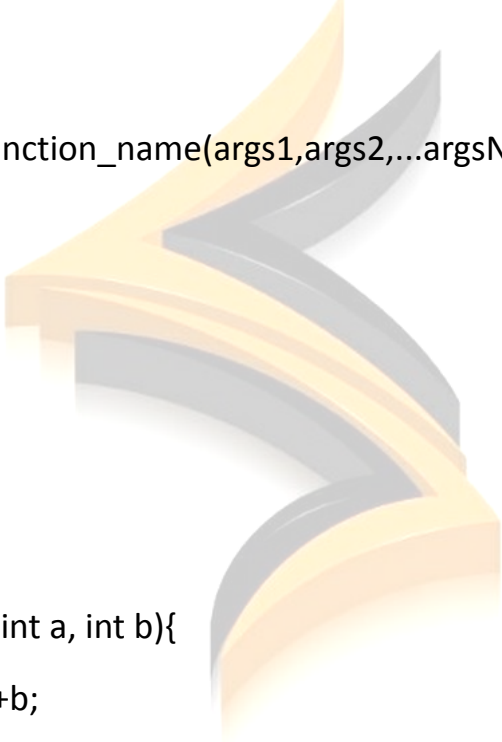
```
    return age;
}
main(){
    int Age =MyAge();
    print(Age);
}
```

Output
20

With arguments and a return type:

Syntax:

```
return_type function_name(args1,args2,...argsN) {
//statements
return value;
}
```



Example:

```
int sum( int a, int b){
    int ab=a+b;
    return ab;
}
main()
{
    int sum1=sum(10,20);
    print(sum1);
}
```

Output


Recursion

Recursion in any programming language means a function making a call to itself. It is used to solve large complex problems by breaking them into smaller subproblems. Dart also implements recursion similarly.

In a recursive function, the function calls itself repeatedly until a base condition is reached. The function basically has two parts.

- **Recursive:** The recursive part is called again and again with a smaller subproblem.
- **Base:** The base condition is checked every time a function call is made. If the function is in a base condition it is used to provide a solution.

Example - 1



```
int factorial(int num){  
  
    //base case of recursion.  
  
    if(num<=1) { // base case  
  
        return 1;  
  
    }  
  
    else{  
  
        return num*factorial(num-1); //function call itself.  
  
    }  
}
```



```
}  
  
void main() {  
  
    var num = 5;  
  
    // Storing function call result in fact variable.  
  
    var fact = factorial(num);  
  
    print("Factorial Of 5 is: ${fact}");  
  
}
```



Dart Object-Oriented Concepts

Dart is an object-oriented programming language, and it supports all the concepts of object-oriented programming such as classes, object, inheritance, mixin, and abstract classes. As the name suggests, it focuses on the object and objects are the real-life entities. The Object-oriented programming approach is used to implement concepts like polymorphism, data-hiding, etc. The main goal of oops is to reduce programming complexity and do several tasks simultaneously. The oops concepts are given below.



- Class
- Object
- Inheritance
- Polymorphism
- Interfaces
- Abstract class

Class

Dart classes are defined as the blueprint of the associated objects. A Class is a user-defined data type that describes the characteristics and behavior of it. To get all properties of the class, we must create an object of that class. The syntax of the class is given below.

Object

An object is a real-life entity such as a table, human, car, etc. The object has two characteristics - state and behaviour. Let's take an example of a car which has a name, model name, price and behaviour moving, stopping, etc. The object-oriented programming offers to identify the state and behaviour of the object.

We can access the class properties by creating an object of that class. In Dart, The object can be created by using a new keyword followed by class name. The syntax is given below.

Inheritance

Dart supports inheritance, which is used to create new classes from an existing class. The class that is to be extended is called parent /superclass, and the newly created class is called child/subclass. Dart provides **extended** keywords to inherit the properties of parent class in child class. The syntax is given below.

Polymorphism

Polymorphism is an object-oriented programming concept where one thing has many forms. There can be two types - Runtime polymorphism and Compile time polymorphism. For example - A function has the same name but with a different behaviour or class. Another example is the **shape()** class, and all the class inherited from the Rectangle, Triangle, and circle.

Abstract Class

A class that contains one or more abstract methods is called an abstract class. We can declare the abstract class using the **abstract** keyword followed by class declaration. The syntax is given below.

Dart Classes and Object

Dart classes are the blueprint of the object, or it can be called object constructors. A class can contain fields, functions, constructors, etc. It is a wrapper that binds/encapsulates the data and functions together; that can be accessed by creating an object. A class can refer to a user-defined data type which defines characteristics by its all objects.

We can assume a class as a sketch (prototype) or a car. It contains all the details about model name, year, features, price, etc.

Based on these properties of the car, we can build the car. Here the car is an object. There can be many cars so we can create many objects of cars to access all the properties.

Defining a Class in Dart

Dart provides a class keyword followed by a class name that is used to define a class; all fields and functions are enclosed by the pair of curly braces (`{}`). The syntax is given below.

Let's understand the following example.

Example -

```
void main() {  
  
    // Defining class  
  
    class Student {  
  
        var stdName;  
  
        var stdAge;  
  
        var stdRoll_nu;  
  
        // Class Function  
  
        showStdInfo() {
```



```

print("Student Name is : ${stdName}");

print("Student Age is : ${stdAge}");

print("Student Roll Number is : ${stdRoll_nu}")

}

```

In the above example of class, we declared a class called **Student**. This class has three fields **stdName**, **stdAge**, and **stdRoll_nu**. The **showStdInfo()** is a class function which prints the fields of class. To access the properties of the class, we need to create its object.

Dart Object

Dart is object-oriented programming, and everything is treated as an object in Dart. An object is a variable or instance of the class used to access the class's properties. Objects have two features - state and behaviour. Suppose a man is an object with a state (name, age, health) and behaviour (walking, running, and sleeping). Programming objects are theoretically similar to real-life objects; they also have state and behaviour. An object is created from a template which is known as class.

The fields of the classes are stored as object states, whereas the method represents an object's behaviour.

Creating Class Objects in Dart

After creating the class, we can create an instance or object of that class which we want to access its fields and functions. The **new** keyword is used to declare class followed by the class name. The general syntax of creating an object of a class is given below.

Syntax:

```

class Student {

    var stdName;

    var stdAge;

```

```
var stdRoll_nu;

// Class Function

showStdInfo() {

    print("Student Name is : ${stdName}");

    print("Student Age is : ${stdAge}");

    print("Student Roll Number is : ${stdRoll_nu}")

}

}

void main () {

    // Creating Object called std

    var std = new Student();

}
```

We have created the object called **std** of the class **Student** but only creating an object is not enough. We have to access the properties by using the newly created object.