Introduction (ઈન્ટ્રોડક્શન)

- Dart એ એન્ડ્રોઈડ, iOS, વેબ, ડેસ્કટોપ વગેરે જેવા ઘણા પ્લેટફોર્મ પર ઝડપથી એપ બનાવવા માટે ક્લાયન્ટ–ઓપ્ટિમાઈઝ, ઓબ્જેક્ટ–ઓરિએન્ટેડ, લેટેસ્ટ પ્રોગ્રામિંગ લેંગ્વેજ છે.
- Dart પ્રોગ્રામિંગ લેંગ્વેજ ગુગલ દ્વારા ડેવલપ કરવામાં આવી છે.
- તે ફ્રી, ઓપન સોર્સ અને પ્લેટફોર્મ–ઈન્ડીપેન્ડન્ટ પ્રોગ્રામિંગ લેંગ્વેજ છે.
- તેમાં dart2js કમ્પાઈલર હોય છે જે dart કોડ ને જાવાસ્ક્રિપ્ટ કોડમાં કન્વર્ટ કરે છે જે તમામ લેટેસ્ટ બ્રાઉઝરમાં એક્ઝીક્યુટ થાય છે.
- તે **null safety** અને **asynchronous programming** જેવા લેટેસ્ટ પ્રોગ્રામિંગ ફીચર પ્રોવાઈડ કરે છે.
- તેનો ઉપયોગ flutter ફ્રેમવર્ક દ્વારા પ્રોગ્રામિંગ લેંગ્વેજ તરીકે થાય છે.

Variable and Data types (વેરીએબલ અને ડેટા ટાઈપ)

- વેરીએબલનો ઉપયોગ વેલ્યુને સ્ટોર કરવા અને ડેટા ટાઈપનો ઉપયોગ તમામ વિવિધ પ્રકારના ડેટાને કેટેગરાઈઝ કરવા માટે થાય છે.
- ડેટા ટાઈપ એ વેરીએબલ દ્વારા સ્ટોર કરવામાં આવતી વેલ્યુની ટાઈપ દર્શાવે છે.
- દરેક વેરીએબલને પોતાની ડેટા ટાઈપ હોય છે.
- Dart નીચે દર્શાવેલાં built-in ડેટા ટાઈપને સપોર્ટ કરે છે:

Data Type	Keyword	Description
Numbers	int, double, num	તે ન્યુમેરિક વેલ્યુ દર્શાવે છે.
Strings	String	તે કેરેક્ટર ની સિકવન્સ દર્શાવે છે.
Booleans	Bool	તે બુલિયન વેલ્યુ true કે false દર્શાવે છે.
Lists	List	તે ડેટા–આઈટમનું ઓર્ડર ગ્રુપ દર્શાવે છે.
Maps	Map	તે કી–વેલ્યુ પેર તરીકે વેલ્યુના ગ્રુપને દર્શાવે છે.
Dynamic	dynamic	જો વેરીએબલની ડેટા ટાઈપ ડિફાઈન કરેલ ન હોય, તો તેની ડીફોલ્ટ ટાઈપ ડાયનેમિક હોય છે.

1) Numbers (નંબર)

- int અથવા double નો ઉપયોગ Dart માં ન્યુમેરિક વેલ્યુને સ્ટોર કરવા માટે થાય છે. તે num નો સબ ટાઈપ છે.
- num નો ઉપયોગ int અથવા double ને સ્ટોર કરવા માટે થાય છે.

Example:

```
void main() {
        // Declaring Variables
        int num1 = 100; // without decimal point
        double num2 = 130.2; // with decimal point
        num num3 = 50;
        num  num4 = 50.4;

        num sum = num1 + num2 + num3 + num4;
```

```
        // Display output
        print("Num 1 is $num1");
        print("Num 2 is $num2");
        print("Num 3 is $num3");
        print("Num 4 is $num4");
        print("Sum is $sum");
        print("Sum is ${sum.toStringAsFixed(2)}"); // વેલ્યુને 2 ડેસીમલ પોઈન્ટ સુધી રાઉન્ડ–અપ કરવા માટે
    }
```

Output:

```
Num 1 is 100
Num 2 is 130.2
Num 3 is 50
Num 4 is 50.4
Sum is 330.59999999999997
Sum is 330.60
```

2) String (સ્ટ્રીંગ)

- સ્ટ્રીંગનો ઉપયોગ text ડેટાને સ્ટોર કરવા માટે થાય છે.
- આપણે single quotation અથવા double quotationનો ઉપયોગ કરીને સ્ટ્રિંગને પ્રિન્ટ કરી શકીએ છીએ.
- મલ્ટીલાઇનમાં આઉટપુટ પ્રિન્ટ કરવા માટે, triple, single અથવા double quotation નો ઉપયોગ થાય છે.
- પ્લસ (+) ઓપરેટરનો ઉપયોગ સ્ટ્રીંગને કોન્કેટ કરવા માટે થાય છે.
- ${} નો ઉપયોગ સ્ટ્રીંગમાં Dart એક્ષપ્રેશનની વેલ્યુને ઇન્ટરપોલેટ કરવા માટે થાય છે.

Example:

```
void main() {
        String schoolName = "Diamond School";
        String address = """New York 21
        welcome
        Dart"""; // to print in multiline
        String city = 'New York';

        String str = 'Darshan University';
        String str1 = ' is at hadala';
        print(str);
        print(str + str1);

        print("School name is $schoolName"); // Printing Values
        print("School address is $address");
        print("School city is $city and length is: ${city.length}");
}
```

Output:

```
Darshan University
Darshan University is at hadala
School name is Diamond School
School address is New York 21
```

```
welcome
Dart
School city is New York and length is: 8
```

3) Boolean (બુલિયન)

- Dart માં બુલિયન ડેટા ટાઈપને ડિફાઈન કરવા માટે bool કીવર્ડનો ઉપયોગ થાય છે.
- તે વેલ્યુને true કે false તરીકે પ્રિન્ટ કરે છે.

Example:

```
void main() {
        bool a = true;
        print("a is: $a");
}
```

Output:

```
a is: true
```

4) List (લિસ્ટ)

- ડેટા ટાઈપ લિસ્ટ અને મેપનો ઉપયોગ ઓબ્જેક્ટના કલેકશનને દર્શાવવા માટે થાય છે.
- **લિસ્ટ** એ ઓબ્જેક્ટનું ઓર્ડર–ગ્રુપ છે.
- Dart માં **લિસ્ટ** ડેટા ટાઈપનો કોન્સેપ્ટ એ અન્ય પ્રોગ્રામિંગ લેંગ્વેજમાં **એરે** જેવો જ છે.
- તેનો ઉપયોગ એક વેરીએબલમાં મલ્ટીપલ વેલ્યુને સ્ટોર કરવા માટે થાય છે.
- કોઈ પણ list-itemને એક્સેસ કરવા માટે ઈન્ડેક્ષનો ઉપયોગ થાય છે.
- લિસ્ટને નીચે દર્શાવેલ રીતે કલાસીફાઈડ કરી શકાય છે.
  1) ફિક્સ લેન્થ લિસ્ટ (Fixed Length List)
  2) ગ્રોએબલ લિસ્ટ (Growable List)

1. Fixed Length List

- ફિક્સ લેન્થ લિસ્ટની લેન્થ રનટાઈમે બદલી શકાતી નથી.

Syntax:

```
var list_name = new List.filled(initial_size, fill);
```

- **var** કીવર્ડ ઓટોમેટીકલી ડેટા ટાઈપ અલોકેટ કરે છે. તેનો અર્થ એ છે કે જ્યારે આપણે ડેટા ટાઈપને સ્પેસીફાય કરવા માંગતા નથી, ત્યારે તે ઓટોમેટીકલી ડેટા ટાઈપ અલોકેટ કરે છે.
- ઉપરોક્ત સિન્ટેક્ષ આપેલ સાઈઝનું લિસ્ટ બનાવે છે અને ડીફોલ્ટ વેલ્યુથી fill કરે છે.
- લિસ્ટને રનટાઈમે વધારી કે ઘટાડી શકાતું નથી.

Example:

```
void main() {
        var lst = new List.filled(3,0);

        lst[2] = 11;
        print("List is: $lst"); }
```

Output:

```
List is: [0, 0, 11]
```

2. Growable List

- ગ્રોએબલ લિસ્ટની સાઈઝ રનટાઈમે બદલી શકાય છે.

Example:

```
void main()
{
        var growList = [1,2,3];
        growList.add(4);
        growList.add(5);
        growList.add(6);
        print("Growable List is: $growList");
        print("Value at 3rd index is: ${growList[3]}");
        print("List size is ${growList.length}");
}
```

Output:

```
Growable List is: [1, 2, 3, 4, 5, 6]
Value at 3rd index is: 4
List size is 6
```

5) Map (મેપ)

- મેપ એ એક ઓબ્જેક્ટ છે જેમાં તમે **કી–વેલ્યુ** પેર તરીકે ડેટા સ્ટોર થાય છે.
- દરેક કી–વેલ્યુ પેરમાં key યુનીક જ હોવી જોઈએ જ્યારે વેલ્યુ સરખી હોય શકે.
- મેપમાં કી અને વેલ્યુની ડેટા ટાઈપ કોઈપણ હોય શકે છે.
- તે એક ડાયનેમીક કલેક્શન છે.

Syntax:

```
var mapName = Map();
mapName[key] = value;
```

Example:

```
void main()
{
        var mapname = Map();
        mapname['Hello'] = "Welcome";
        print(mapname);

        Map<String, String> mapName = Map();
        mapName ['First'] = 'Darshan';
        mapName ['Second'] = 'University';
        print(mapName);
}
```

Output:

```
{Hello: Welcome}
{First: Darshan, Second: University}
```

6) Dynamic (ડાયનેમીક)

- જ્યારે આપણે વેરીએબલની ડેટા ટાઈપને **dynamic** તરીકે ડીકલેર કરીએ છીએ, ત્યારે વેરીએબલની ડેટા ટાઈપ રન ટાઈમ પર નક્કી થાય છે.

Example:

```
void main() {
```

```
        dynamic myVariable = 50;
        myVariable = "Hello";
        print(myVariable);
}
```

Output:

```
Hello
```

Null Safety (Null સેફ્ટી)

- Null Safety નો અર્થ એ છે કે વેરીએબલમાં 'null' વેલ્યુ ન હોય શકે.
- જેથી અજાણતા null વેલ્યુ સ્ટોર થયેલ કોઈપણ વેરીએબલને એક્સેસ કરવાથી જે એરર જનરેટ થાય છે તેને અટકાવી શકાય છે.
- વેરીએબલના બે પ્રકાર છે:
  1) Non-nullable Types
  2) Nullable Types

1) Non-nullable Types

- જ્યારે આપણે null safety નો ઉપયોગ કરીએ છીએ, ત્યારે બધી ટાઈપ ડીફોલ્ટ રીતે **non-nullable** હોય છે.
- **ઉદાહરણ,** int વેરીએબલમાં integer વેલ્યુ હોવી જરુરી છે.

Example:

```
void main(){
        int number;
        number = 0;
}
```

2) Nullable Types

- **Nullable type (?)** ઓપરેટર દર્શાવે છે કે વેરીએબલ null હોઈ શકે છે.
- Nullable વેરીએબલનો ઉપયોગ કરતા પહેલા તેને ઈનીશીએલાઈઝ કરવાની જરૂર નથી.

Example:

```
void main() {
        String? houseLocationName;  // By default, it's set to null.
        int? number = 36;  // By default, it's set to non-null.
        number = null; // It's possible to reassign it to null.
        print(number);
        print(houseLocationName);
        }
```

Output:

```
null
null
```

Assertion Operator (!) (એસરટન ઓપરેટર (!))

- તેનો ઉપયોગ dart કમ્પાઈલરને જાણ કરવા માટે થાય છે કે વેરીએબલની વેલ્યુ null નહિ હોય.

Example:

```
void main(){
        int? anyNumber = 50;
```

```
        int data = anyNumber!;   // Because the value is not nullable, this is valid
    }
```

Read/User Input (રીડ/યુઝર ઈનપુટ)

- પ્રોગ્રામને ડાયનેમિક બનાવવા માટે, યુઝર પાસેથી વેલ્યુ લેવી જરૂરી છે તેના માટે આપણે **'dart:io'** પેકેજ ઈમ્પોર્ટ કરવું પડે છે.
- આપણે **readLineSync()** ફંક્શનનો ઉપયોગ કરીને કોન્સોલ દ્વારા યુઝર પાસેથી સ્ટાન્ડર્ડ ઈનપુટ લઈ શકીએ છીએ.
- આ મેથડ માત્ર સ્ટ્રિંગ ફોર્મેટમાં વેલ્યુ રીડ કરે છે. જો આપણે int અથવા double વેલ્યુ રીડ કરવી હોય, તો તેને parse() મેથડનો ઉપયોગ કરીને parse કરવી પડે છે.

Example:

```
import 'dart:io';
void main()
{
    print('Enter BookName');
    String? bookName  = stdin.readLineSync();
    print('Enter Total Pages');
    int? page = int.parse(stdin.readLineSync()!);
    print('Enter Price');
    double? price = double.parse(stdin.readLineSync()!);
    print('BookName is $bookName');
    print('Pages are $page');
    print('Price is $price');
}
```

Output:

```
Enter BookName
Flutter Beginner
Enter Total Pages
500
Enter Price
1050.5
BookName is Flutter Beginner
Pages are 500
Price is 1050.5
```

Operators (ઓપરેટર)

- ઓપરેટર એ સ્પેશિયલ સિમ્બોલ છે જેનો ઉપયોગ ઓપરેન્ડ પર ચોક્કસ ઓપરેશન કરવા માટે થાય છે.
- Dart માં નીચે દર્શાવેલ in-built ઓપરેટર છે:
  1) એરિથમેટિક ઓપરેટર
  2) રીલેશનલ ઓપરેટર
  3) અસાઈનમેન્ટ ઓપરેટર
  4) લોજિકલ ઓપરેટર
  5) બીટવાઈઝ ઓપરેટર

6) કંડીશનલ ઓપરેટર
7) ઈન્ક્રીમેન્ટ/ડીક્રીમેન્ટ ઓપરેટર
8) ટાઈપ ટેસ્ટ ઓપરેટર

1) એરિથમેટિક ઓપરેટર

- આ ઓપરેટરનો ઉપયોગ ઓપરેન્ડ પર એરિથમેટિક ઓપરેશન કરવા માટે થાય છે.

Operator	Name	Description
+	Addition	બે ઓપરેન્ડના એડીશન માટે ઉપયોગ થાય છે.
-	Subtraction	બે ઓપરેન્ડના સબટ્રેકશન માટે ઉપયોગ થાય છે.
-expr	Unary Minus	એક્સપ્રેશનની sign ને રિવર્સ કરવા માટે ઉપયોગ થાય છે.
*	Multiply	બે ઓપરેન્ડના મલ્ટીપ્લીકેશન માટે ઉપયોગ થાય છે.
/	Division	બે ઓપરેન્ડના ડીવીઝન માટે ઉપયોગ થાય છે.
~/	Division (Integer Division)	બે ઓપરેન્ડને ડીવાઈડ કરવા માટે ઉપયોગ થાય છે પરંતુ આઉટપુટ integerમાં આપે છે
%	Modulus	બે ઓપરેન્ડના રીમેઈન્ડર આપે છે.

Example:

```
void main()
{
        int a = 2, b = 3;

        var c = a + b;  // Adding a and b
        print("Sum of a and b is $c");
        var d = a - b; // Subtracting a and b
        print("The difference between a and b is $d");

        var e = -d; // Using unary minus
        print("The negation between a and b is $e");

        var f = a * b; // Multiplication of a and b
        print("The product of a and b is $f");

        var g = b / a; // Division of a and b
        print("The quotient of a and b is $g");

        var h = b ~/ a; // Using ~/ to divide a and b
        print("The quotient of a and b is $h");

        var i = b % a; // Remainder of a and b
        print("The remainder of a and b is $i");
}
```

Output:

```
Sum of a and b is 5
The difference between a and b is -1
The negation between a and b is 1
The product of a and b is 6
```

The quotient of a and b is 1.5
The Integer quotient of a and b is 1
The remainder of a and b is 1

2) રિલેશનલ ઓપરેશન

- આ ઓપરેટરનો ઉપયોગ ઓપરેન્ડ પર રિલેશનલ ઓપરેશન કરવા માટે થાય છે.

Operator	Name	Description
>	Greater than	કયો ઓપરેન્ડ મોટો છે તે ચેક કરવા માટે ઉપયોગ થાય છે અને બુલિયન true કે false તરીકે રીઝલ્ટ આપે છે.
<	Less than	કયો ઓપરેન્ડ નાનો છે તે ચેક કરવા માટે ઉપયોગ થાય છે અને બુલિયન true કે false તરીકે રીઝલ્ટ આપે છે.
>=	Greater than or equal to	કયો ઓપરેન્ડ મોટો કે સમાન છે તે ચેક કરવા માટે ઉપયોગ થાય છે અને બુલિયન true કે false તરીકે રીઝલ્ટ આપે છે.
<=	less than equal to	કયો ઓપરેન્ડ નાનો કે સમાન છે તે ચેક કરવા માટે ઉપયોગ થાય છે અને બુલિયન true કે false તરીકે રીઝલ્ટ આપે છે.
==	Equal to	ઓપરેન્ડ સમાન છે કે નહિ તે ચેક કરવા માટે ઉપયોગ થાય છે અને જો સમાન હોય તો true નહિતર false રીઝલ્ટ આપે છે.
!=	Not Equal to	ઓપરેન્ડ અસમાન છે કે નહિ તે ચેક કરવા માટે ઉપયોગ થાય છે અને જો અસમાન હોય તો true નહિતર false રીઝલ્ટ આપે છે.

Example:

```
void main()
{
        int a=2;
        int b=3;
        var c = a > b; // Greater between a and b
        print("a is greater than b is $c");

        var d = a < b; // Smaller between a and b
        print("a is smaller than b is $d");

        var e = a >= b; // Greater than or equal to between a and b
        print("a is greater than b is $e");

        var f = a <= b; // Less than or equal to between a and b
        print("a is smaller than b is $f");
        var g = b == a; // Equality between a and b
        print("a and b are equal is $g");

        var h = b != a; // Inequality between a and b
        print("a and b are not equal is $h"); }
```

Output:

```
a is greater than b is false
a is smaller than b is true
a is greater than b is false
a is smaller than b is true
a and b are equal is false
a and b are not equal is true
```

3) અસાઈનમેન્ટ ઓપરેટર

- અસાયનમેન્ટ ઓપરેટરનો ઉપયોગ વેરીએબલને અમુક વેલ્યુ અસાઈન કરવા માટે થાય છે.

Operator	Name	Description
=	Equal to	એક્ષપ્રેશન અથવા વેરીએબલને વેલ્યુ અસાઈન કરવા માટે ઉપયોગ થાય છે.
+=	Addition & Equal to	તે ડાબી બાજુના ઓપરેન્ડમાં જમણી બાજુના ઓપરેન્ડને એડ કરે છે અને રીઝલ્ટ ડાબી બાજુના ઓપરેન્ડમાં અસાઈન કરે છે.
-=	Subtraction & Equal to	તે ડાબી બાજુના ઓપરેન્ડમાંથી જમણી બાજુના ઓપરેન્ડને સબટ્રેકટ કરે છે અને રીઝલ્ટ ડાબી બાજુના ઓપરેન્ડમાં અસાઈન કરે છે.
*=	Multiplication & Equal to	તે ડાબી બાજુના ઓપરેન્ડમાં જમણી બાજુના ઓપરેન્ડને મલ્ટીપ્લાય કરે છે અને રીઝલ્ટ ડાબી બાજુના ઓપરેન્ડમાં અસાઈન કરે છે.
/=	division & Equal to	તે ડાબી બાજુના ઓપરેન્ડને જમણી બાજુના ઓપરેન્ડને ડીવાઈડ કરે છે અને રીઝલ્ટ ડાબી બાજુના ઓપરેન્ડમાં અસાઈન કરે છે.
??=	Null-aware Assignment	જો વેરીએબલ null હોય તો જ વેલ્યુ અસાઈન કરે છે.

Example:

```
void main()
{
        dynamic a = 2;
        dynamic b = 3;

        b += a; // Perform b = b + a
        print("Addition of b & a and result in b is $b");

        b -= a; // Perform b = b - a
        print("Subtraction of b & a and result in b is $b");

b *= a; // Perform b = b * a
print("Multiplication of b & a and result in b is $b");
```

```
        b /= a; // Perform b = b / a
        print("Division of b & a and result in b is $b");

        var c;
        c ??= a + b; // Value is assign as it is null
        print("Value of c is $c");
    }
```

Output:

```
Addition of b & a and result in b is 5
Subtraction of b & a and result in b is 3
Multiplication of b & a and result in b is 6
Division of b & a and result in b is 3.0
Value of c is 5.0
```

4) લોજીકલ ઓપરેટર

- લોજીકલ ઓપરેટરનો ઉપયોગ લોજીકલ રીતે બે અથવા વધુ કંડીશનની કમ્પેરીઝન કરવા માટે થાય છે.

Operator	Name	Description
&&	And Operator	બે કંડીશન ઉમેરવા માટે ઉપયોગ થાય છે અને જો બંને true હોય તો તે true રીટર્ન કરશે.
||	Or Operator	બે કંડીશન ઉમેરવા માટે ઉપયોગ થાય છે અને જો તેમાંથી એક પણ true હોય તો તે true રીટર્ન કરશે.
!	Not Operator	તેનો ઉપયોગ રીઝલ્ટને રીવર્સ કરવા માટે થાય છે.

Example:

```
void main()
{
    int a = 5;
    int b = 7;

    bool c = a > 10 && b < 10;
    print(c);

    bool d = a > 10 || b < 10;
    print(d);

    bool e = !(a > 10);
    print(e);
}
```

Output:

```
false
true
true
```

5) બીટવાઈઝ ઓપરેટર

- બીટવાઈઝ ઓપરેટરનો ઉપયોગ ઓપરેન્ડ પર બીટવાઈઝ ઓપરેશન કરવા માટે થાય છે.

Operator	Name	Description
&	Bitwise AND	બે ઓપરેન્ડ પર બીટવાઇઝ AND ઓપરેશન પરફોર્મ કરે છે.
|	Bitwise OR	બે ઓપરેન્ડ પર બીટવાઇઝ OR ઓપરેશન પરફોર્મ કરે છે.
^	Bitwise XOR	બે ઓપરેન્ડ પર બીટવાઇઝ XOR ઓપરેશન પરફોર્મ કરે છે.
~	Bitwise NOT	બે ઓપરેન્ડ પર બીટવાઇઝ NOT ઓપરેશન પરફોર્મ કરે છે.
<<	Left Shift	બીટને ડાબી બાજુ શિફ્ટ કરે છે અને જમણી બાજુ એ 0 ઇન્સર્ટ કરે છે.
>>	Right Shift	બીટને જમણી બાજુ શિફ્ટ કરે છે અને ડાબી બાજુ એ 0 ઇન્સર્ટ કરે છે.

Example:

```
void main()
{
    int a = 10;
    int b = 10;

    var c = a & b; // Performing Bitwise AND on a and b
    print("a & b is: $c");

    var d = a | b; // Performing Bitwise OR on a and b
    print("a | b is: $d");

    var e = a ^ b; // Performing Bitwise XOR on a and b
    print("a ^ b is: $e");

    var f = ~a; // Performing Bitwise NOT on a
    print("~a is: $f");

    var g = a << 2; // shift 2 bit left-side of a
    print("a << b is: $g");

    var h = a >> 2; // shift 2 bit right-side of a
    print("a >> b is: $h");
}
```

Output:

```
a & b is: 10
a | b is: 10
a ^ b is: 0
~a is: -11
a << b is: 40
a >> b is: 2
```

6) કંડીશનલ/ટર્નરી ઓપરેટર

- કંડીશનલ ઓપરેટર એ **if-else** કંડીશન જેવું જ છે.

- Dart માં બે પ્રકારના કન્ડીશનલ ઓપરેટરની સિન્ટેક્ષ છે.
- એક null safety ચેક અને અન્ય જે આપણે બીજી પ્રોગ્રામિંગ લેંગ્વેજમાં સામાન્ય રીતે ઉપયોગ કરીએ છીએ.

Syntax:

```
condition ? expressionOne : expressionTwo;
```

Example:

```
void main(){
        var ans = 10;
        ans == 10 ? print("Answer is 10") : print("Oh no!");
        }
```

Output:

```
Answer is 10
```

- **Null Aware Operator:**
  - તે એક કંડીશનલ સ્ટેટમેન્ટ દર્શાવે છે જે કંડીશનલ ઓપરેટર સ્ટેટમેન્ટ જેવું જ છે.
  - જો કોઈ એક્ષપ્રેશન null હોય અને તમે કોઈ એક્ષપ્રેશનને ઈવેલ્યુએટ કરી અને રીટર્ન કરવા માંગો છો ત્યારે null aware ઓપરેટરનો ઉપયોગ થાય છે.
  - તેને **if-null** ઓપરેટર પણ કહેવામાં આવે છે.
  - Null-aware ઓપરેટર **??** એ એક્ષપ્રેશનની વેલ્યુ null સિવાયની હોય તો ડાબી બાજુ એક્ષપ્રેશનમાં રીટર્ન કરે છે.

Example:

```
void main(){
        var b;
        String a = b ??  'Hello';
        print("Value of a is: $a");
    }
```

Output:

```
Value of a is: Hello
```

7) ઈન્ક્રીમેન્ટ/ડિક્રીમેન્ટ ઓપરેટર

- ઈન્ક્રીમેન્ટ/ડિક્રીમેન્ટ ઓપરેટરનો ઉપયોગ વેલ્યુમાં 1 નો વેલ્યુ વધારો અથવા ઘટાડો કરવા માટે થાય છે.
- તેનો prefix અથવા postfix તરીકે ઉપયોગ કરી શકાય છે.

Operator	Name	Description
++var	Pre-Increment	varની વેલ્યુમાં 1 નો વધારો કરે છે. var = var + 1 એક્ષપ્રેશનની વેલ્યુ var+1 થશે.
--var	Pre-Decrement	varની વેલ્યુમાં 1 નો ઘટાડો કરે છે. var = var - 1 એક્ષપ્રેશનની વેલ્યુ var-1 થશે.
var++	Post-Increment	varની વેલ્યુમાં 1 નો વધારો કરે છે. var = var + 1 એક્ષપ્રેશનની વેલ્યુ var થશે.
var--	Post-Decrement	varની વેલ્યુમાં 1 નો ઘટાડો કરે છે. var = var - 1 એક્ષપ્રેશનની વેલ્યુ var થશે.

Example:

```
void main() {
        int num1=0;
        int num2=0;

        num2 = ++num1;
        print("The value of num1 is $num1 & num2 is $num2");

        num2 =  num1++;
        print("The value of num1 is $num1 & num2 is $num2");
        }
```

Output:

```
The value of num1 is 1 & num2 is 1
The value of num1 is 2 & num2 is 1
```

8) ટાઈપ ટેસ્ટ ઓપરેટર

- ટાઈપ ટેસ્ટ ઓપરેટરનો ઉપયોગ રનટાઈમે ડેટા ટાઈપ ચેક કરવા માટે થાય છે.

Operator	Name	Description
is	is	જો ઓબ્જેક્ટની ડેટા ટાઈપ આપેલ ડેટા ટાઈપ જ હોય તો બુલિયન વેલ્યુ true આપશે.
is!	is not	જો ઓબ્જેક્ટની ડેટા ટાઈપ આપેલ ડેટા ટાઈપ જ હોય તો બુલિયન વેલ્યુ false આપશે.

Example:

```
void main() {
        String name = "Darshan University";
        int since = 2009;

        print(name is String);
        print(since is !int); }
```

Output:

```
true
false
```

Decision Making (ડીસીઝન મેકિંગ)

- ડીસીઝન મેકિંગ સ્ટેટમેન્ટ પ્રોગ્રામરને વિવિધ કંડીશન હેઠળ કયું સ્ટેટમેન્ટ એક્ઝીક્યુટ કરવું જોઈએ તે પસંદ કરવાની પરમીશન આપે છે.
- Dart માં ચાર પ્રકારના ડીસીઝન મેકિંગ સ્ટેટમેન્ટ આવેલ છે.
  1) if સ્ટેટમેન્ટ
  2) if-else સ્ટેટમેન્ટ
  3) if else if સ્ટેટમેન્ટ
  4) switch-case સ્ટેટમેન્ટ

1) if સ્ટેટમેન્ટ

- if સ્ટેટમેન્ટ આપેલ કંડીશનને ચેક કરશે અને જો તે true હોય, તો તે કોડ નો બ્લોક એક્ઝિક્યુટ થશે.

Syntax:

```
if(condition) {
   Statement 1;
   Statement 2;
     .
     .
   Statement n; }
```

Example:

```
void main(){
        int a = 8;

        if(a > 5) {
        print(" $a is greater than 5"); }
}
```

Output:

8 is greater than 5

2) if-else સ્ટેટમેન્ટ

- if-else સ્ટેટમેન્ટ આપેલ કંડીશનને ચેક કરશે અને જો તે true હોય, તો તે કોડના if-blockને એક્ઝિક્યુટ કરશે નહિતર કોડના else બ્લોકને એક્ઝિક્યુટ કરશે.

Syntax:

```
if(condition){
        statements;
}
else{
        statements;
}
```

Example:

```
void main(){
        int a = 2;
        if(a > 5){
                print(" $a is greater than 5");
        }
        else{
                print(" $a is smaller than 5");
        }
}
```

Output:

2 is smaller than 5

3) if-else-if સ્ટેટમેન્ટ

- જો આપણી પાસે બે કરતા વધુ ઓપ્શન હોય ત્યારે if-else-if સ્ટેટમેન્ટનો ઉપયોગ થાય છે

- if else if સ્ટેટમેન્ટ કન્ડીશનને ચેક કરશે અને જો તે true હોય, તો તેની અંદરના સ્ટેટમેન્ટ એક્ઝિક્યુટ થશે.
- જો તે false હોય તો, અન્ય કંડીશન ચેક કરવામાં આવે છે, અને જો કોઈ કંડીશન true હોય તો, તેની અંદરના સ્ટેટમેન્ટ ને એક્ઝિક્યુટ કરે છે.

Syntax:

```
if(condition1){
        statements-1;
}
else if(condition2){
statements-2;
}
else if(condition3){
        statements-3;
}
.
.
else{
        statements-N;
}
```

Example:

```
void main()
{
        int a = 15;
        int b = 5;
        if (a < b) {
                print("Condition 1 is true");
        }
        else if (a > b) {
                print("Condition 2 is true");
        }
        else if (a == b) {
        print("Condition 3 is true"); }
        else {
                print("All the conditions are false");
        }
                }
```

Output:

```
Condition 2 is true
```

4) switch-case સ્ટેટમેન્ટ

- તેને multi-way decision-making statement (મલ્ટી–વે–ડીસીઝન–મેકિંગ સ્ટેટમેન્ટ) તરીકે પણ ઓળખવામાં આવે છે.
- switch-case સ્ટેટમેન્ટ એ if-else-if સ્ટેટમેન્ટ જેવું જ છે.

Syntax:

```
switch(expression)
{
        case value1:
                // statements
                break;
        case value2:
                // statements
                break;
        .
        default:
                //statements;
}
```

- એક્ષપ્રેશનને એકવાર ઈવેલ્યુએટ કરવામાં આવે છે અને ત્યારબાદ દરેક **case** લેબલની વેલ્યુ સાથે કમ્પેરીઝન કરવામાં આવે છે.
- જો કોઈ case ના લેબલ સાથે **match** થશે તો તે case ના સ્ટેટમેન્ટ એક્ઝીક્યુટ થશે.
- **ઉદાહરણ,** જો એક્ષપ્રેશનની વેલ્યુ **value2** ની બરાબર હોય, તો **case value2:** પછીના સ્ટેટમેન્ટ **break** ન આવે ત્યાં સુધી એક્ઝીક્યુટ થશે.
- જો એકપણ case ના લેબલ સાથે match ન થાય હોય, તો ડિફોલ્ટ સ્ટેટમેન્ટ એક્ઝીક્યુટ થશે.

Example:

```
void main() {
        var grade = "B";
        switch(grade) {
                case "A":
                        print("Excellent");
                        break;
                case "B":
                        print("Good");
                        break;
                case "C":
                        print("Fair");
                        break;
                case "D":
                        print("Poor");
                        break;
                default:
                        print("Invalid choice");
        }
}
```

Output:

Good

Loops (લૂપ)

- લૂપનો ઉપયોગ જ્યાં સુધી કંડીશન ખોટી ન થાય ત્યાં સુધી કોડના બ્લોકને રીપીટ કરવા માટે થાય છે.
- Dart માં વિવિધ પ્રકારની લૂપ આવેલ છે.
  1) for loop

2) for… in loop
3) for each loop
4) while loop
5) do-while loop

1) for લૂપ

- Dart માં for લૂપ એ અન્ય પ્રોગ્રામિંગ લેંગ્વેજની જેમ જ છે અને એક્ઝીક્યુશનનો ફ્લો અન્ય પ્રોગ્રામિંગ લેંગ્વેજ જેવો જ છે.

  Syntax:

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

- for લૂપના કંટ્રોલનો ફ્લો નીચે આપેલ છે:
  1) Initialization (ઈનીશીએલાઈઝેશન)
  2) Condition (કંડીશન)
  3) Body of loop (લૂપની Body)
  4) Test expression (ટેસ્ટ એક્ષપ્રેશન)
- પ્રથમ સ્ટેપ માત્ર એક જ વાર લુપની શરૂઆતમાં એક્ઝીક્યુટ થશે, જ્યારે અન્ય ત્રણ સ્ટેપ કંડીશન ખોટી ન થાય ત્યાં એક્ઝીક્યુટ થશે.

  Example:

```
void main()
{
    for (int i = 0; i < 3; i++) {
        print('Darshan University');
    }
}
```

  Output:

```
Darshan University
Darshan University
Darshan University
```

2) for…in લૂપ

- for...in લૂપનો ઉપયોગ ઓબ્જેક્ટની પ્રોપર્ટીને લૂપ કરવા માટે થાય છે.

  Syntax:

```
for (variablename in object){
    //statement or block to execute
}
```

- દરેક ઈટરેશનમાં, ઓબ્જેક્ટમાંથી એક પ્રોપર્ટી વેરીએબલને અસાઇન કરવામાં આવે છે.
- જ્યાં સુધી ઓબ્જેક્ટની તમામ પ્રોપર્ટી એક્સેસ ન થાય ત્યાં સુધી લૂપ ચાલુ રહે છે.

Example:

```
void main() {
        var obj = [12,13,14];
        for (var prop in obj) {
                print(prop);
        }
}
```

Output:

```
12
13
14
```

3) for Each લૂપ

- **for Each** લૂપ લિસ્ટના બધા એલીમેન્ટ અથવા વેરીએબલને એક્સેસ કરે છે. તે લિસ્ટ/કલેક્શનને લૂપ કરવા માટે ઉપયોગી છે.

  Syntax:

  collectionName.**forEach(void f(value));**

- **f(value):** તેનો ઉપયોગ કલેક્શનના દરેક એલીમેન્ટ માટે function **f** ને કોલ કરવા માટે થાય છે.

  Example:

```
void main() {
        var obj = [12,13,14];
        obj.forEach((num) => print(num));
}
```

  Output:

```
12
13
14
```

4) while લૂપ

- Dart માં while loop એ અન્ય પ્રોગ્રામિંગ લેંગ્વેજ જેવી જ છે.
- **while loop** માં કંડીશન ખોટી ન થાય ત્યાં સુધી લૂપની body એકઝીક્યુટ થશે.
- તમારે સ્ટેટમેન્ટની પહેલાં કંડીશન લખવી જરુરી છે.
- આ લૂપમાં દરેક ઈટરેશનમાં કંડીશન ચેક થશે.
- જો કંડીશન સાચી હોય, તો કોડ નો બ્લોક એક્ઝિક્યુટ થશે અને જો કંડીશન ખોટી હોય તો લૂપ સ્ટોપ થઈ જશે.

  Example:

```
void main() {
        int i = 1;
        while (i <= 3) {
                print(i);
                i++; }
 }
```

Output:

```
1
2
3
```

5) do while લૂપ

- do while લૂપમાં લૂપની body સૌપ્રથમ એક્ઝિક્યુટ થશે, અને પછી કંડીશન ચેક કરવામાં આવશે.
- કંડીશન ખોટી ન થાય ત્યાં સુધી લૂપની body એક્ઝિક્યુટ થશે.
- લૂપની body ઓછામાં ઓછી એકવખત તો એક્ઝીક્યુટ થશે જ.

Syntax:

```
do
{
        statement1;
        statement2;
          .
          .
          .
          .
          .
        statementN;
  }
 while(condition);
```

Example:

```
void main()
{
        int i = 1;
                do {
                print(i);
                i++;
        }
        while (i <= 3);
}
```

Output:

```
1
2
3
```

Break and Continue Statement (બ્રેક અને કન્ટીન્યુ સ્ટેટમેન્ટ)

- આપણે કંડીશન ચેક કર્યા વિના અમુક ઈટરેશન સ્કીપ કરવા અથવા તરત જ લૂપને ટર્મિનેટ કરવા માટે લૂપમાં break અને continue સ્ટેટમેન્ટનો ઉપયોગ થાય છે.

Break Statement (બ્રેક સ્ટેટમેન્ટ)

- break સ્ટેટમેન્ટનો ઉપયોગ લૂપમાંથી exit કરવા માટે થાય છે. તે તરત જ લૂપને સ્ટોપ કરે છે, અને પ્રોગ્રામનો કંટ્રોલ લૂપની બહાર નીકળી જાય છે.

Example:

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

Output:

1
2

Continue Statement (કન્ટીન્યુ સ્ટેટમેન્ટ)

- continue સ્ટેટમેન્ટ લૂપના ઈટરેશનને સ્કીપ કરે છે.
- તે લૂપના સ્ટેટમેન્ટ ને બાયપાસ કરશે.
- તે લૂપને ટર્મીનેટ કરતું નથી પરંતુ નેક્સ્ટ ઈટરેશન સાથે લૂપ ચાલુ રહે છે.

Example:

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

Output:

1
2
4
5

Function (ફંક્શન)

- ફંક્શન એ સ્ટેટમેન્ટનો સેટ છે, જે કોઈ સ્પેસીફિક ટાસ્ક પરફોર્મ કરે છે.
- ફંક્શન્સ એ પ્રોગ્રામને **લોજિકલ કોડના બ્લોક**માં ગોઠવે છે.
- ફંક્શન એ **રીડેબલ, મેન્ટેનેબલ** અને **રીયુઝેબલ** કોડના બિલ્ડીંગ બ્લોક છે.
- એકવાર ફંક્શન ડિફાઈન થઈ ગયા પછી, કોડને એક્સેસ કરવા માટે ફંક્શનને કોલ કરી શકાય છે.

Syntax:

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

- ફંક્શન ડિક્લેરેશન કમ્પાઈલરને **ફંક્શન ના નામ, રીટર્ન ટાઈપ** અને **પેરામીટર** વિશે જણાવે છે.
  - **function_name** ફંક્શનનું નામ ડિફાઈન કરે છે.

- **return_type** આઉટપુટ ટાઈપ ડિફાઈન કરે છે. તે void, String, int, double વગેરે હોય શકે છે. જો ફંક્શન કંઈ પણ રીટર્ન કરતું નથી, તો રીટર્ન ટાઈપ void હોવો જોઈએ.
- **parameters** એ ફંક્શન માટે ઈનપુટ છે, જે બ્રેકેટ () ની અંદર લખવામાં આવે છે.
- **return** વેલ્યુ ફંક્શનમાંથી જે વેલ્યુ રીટર્ન થશે તે દર્શાવે છે. જો ફંક્શનની રીટર્ન ટાઈપ **void** હોય, તો રીટર્ન સ્ટેટમેન્ટ લખવાની જરૂર નથી.

- **Function call (ફંક્શન કોલ)**

Syntax:

function_name **(argument_list);**

- **function_name** ફંક્શનનું નામ ડિફાઈન કરે છે.
- **argument_list** એ પેરામીટરનું લિસ્ટ છે જે ફંક્શન માટે જરૂરી છે.

Example:

```
int add(int a, int b) // Function declaration
{
        int result = a + b;
        return result; // Returning result
}
void main()
{
        var output = add(10, 20); // Calling the function
        print(output); // Display output
}
```

Output:

30

Types of Functions (ફંક્શનના પ્રકાર)

- ફંક્શનના ચાર પ્રકાર છે.
  1) પેરામીટર વગર અને રીટર્ન ટાઈપ વગર (No Parameter and No Return Type)
  2) પેરામીટર સાથે અને રીટર્ન ટાઈપ વગર (With Parameter and No Return Type)
  3) પેરામીટર વગર અને રીટર્ન ટાઈપ સાથે (No Parameter and With Return Type)
  4) પેરામીટર સાથે અને રીટર્ન ટાઈપ સાથે (With Parameter and With Return Type)

1) પેરામીટર વગર અને રીટર્ન ટાઈપ વગર (No Parameter and No Return Type)

Example:

```
void add()
{
        int a=10, b=20;
        int result = a + b;
        print(result);
}
void main()
```

```
{
        add();
}
```

Output:

30

2) પેરામીટર સાથે અને રીટર્ન ટાઈપ વગર (With Parameter and No Return Type)

Example:

```
void add(int a, int b)
{
        int result = a + b;
        print(result);
}
void main()
{
        add(10, 20);
}
```

Output:

30

3) પેરામીટર વગર અને રીટર્ન ટાઈપ સાથે (No Parameter and With Return Type)

Example:

```
int add()
{
        int a=10, b=20;
        int result = a + b;
        return result;
}
void main()
{
        var output = add();
        print(output);
}
```

Output:

30

4) પેરામીટર સાથે અને રીટર્ન ટાઈપ સાથે (With Parameter and With Return Type)

Example:

```
int add(int a, int b){
        int result = a + b;
        return result;
}
void main(){
        var output = add(10, 20);
        print(output); // Display output
}
```

Output:

30

Named Parameter

- Named parameter ઓપ્શનલ હોય છે જો તેને એક્સપ્લીસીટલી **required** તરીકે ન દર્શાવેલ હોય.
- ફંક્શનને ડિફાઈન કરતી વખતે, named parameters ને દર્શાવવા માટે **{param1, param2, ...}** નો ઉપયોગ થાય છે.
- જો આપણે ડિફોલ્ટ વેલ્યુ પ્રોવાઈડ કરતા નથી અથવા named parameter ને **required** તરીકે દર્શાવતા નથી, તો તેની ટાઈપ **nullable** હોવી જોઈએ કારણ કે તેની ડિફોલ્ટ વેલ્યુ **null** હશે.
- ફંક્શનને કોલ કરતી વખતે, **paramName: value** નો ઉપયોગ કરીને named arguments ને દર્શાવી શકાય અને **named parameters**માં ઓર્ડર મહત્વનો હોતો નથી.

Example:

```
void printInfo( {String? name, String? gender, required String city} ) {
        print("Hello $name your gender is $gender and city is $city.");
}
void main() {
        // you can pass values in any order in named parameters.
        printInfo(gender: "Male", city:"New York");
        printInfo(name: "Emily", gender: "Female", city:"Boston");
        }
```

Output:

Hello null your gender is Male and city is New York.
Hello Emily your gender is Female and city is Boston.

Optional Parameter (ઓપ્શનલ પેરામીટર)

- **ફંક્શન ઓવરલોડિંગ Dart માં સપોર્ટ કરતું નથી.**
- ઓપ્શનલ પેરામીટર એ એવા પેરામીટર છે જે ફંક્શનને કોલ કરતી દર્શાવવાની જરૂર રહેતી નથી.
- ઓપ્શનલ પેરામીટર એ આપણે ડિફાઈન કરેલા પેરામીટરમાં ડિફોલ્ટ વેલ્યુ પાસ કરવાની મંજૂરી આપે છે.
- ઓપ્શનલ પેરામીટરના ચાર પ્રકાર છે.
  1) ઓર્ડરડ (પોઝીશનલ) ઓપ્શનલ પેરામીટર (Ordered (positional) optional parameters)
  2) ઓપ્શનલ નેમ્ડ પેરામીટર (Optional Named parameters)
  3) ઓપ્શનલ પેરામીટર ડીફોલ્ટ વેલ્યુ સાથે (Optional parameters with Default Values)

1) ઓર્ડરડ (પોઝીશનલ) ઓપ્શનલ પેરામીટર (Ordered (positional) optional parameters)

- square brackets [] નો ઉપયોગ ઓપ્શનલ પોઝીશીનલ પેરામીટરને દર્શાવવા માટે થાય છે.
- ફંક્શનના એક્ઝીક્યુશન દરમિયાન આર્ગ્યુમેન્ટ ફરજીયાત પાસ કરવાની જરૂર ન હોય ત્યારે ઓપ્શનલ પેરામીટરનો ઉપયોગ થાય છે.

Example:

```
void  main() {
        print(pow(2, 5));
        print(pow(2, 3));
        print(pow(3));
}
int pow( int x, [int y = 6] )
```

```
{
        int r = x + y;
        return r;
}
```

Output:

```
7
5
9
```

2) ઓપ્શનલ નેમ્ડ પેરામીટર (Optional Named parameters)

- Optional Named parameters કર્લી (curly) **{}** બ્રેકેટમાં દર્શાવવામાં આવે છે.
- Optional Named parameter ને પાસ કરતી વખતે, આપણે પેરામીટરનું નામ અને વેલ્યુ બંને ને કોલોન (:)થી અલગ કરીને દર્શાવવામાં આવે છે.
- જો આર્ગ્યુમેન્ટ પાસ કરવી હોય તો પેરામીટરના નામનો ઉપયોગ કરવો જરૂરી છે.

Example:

```
void main() {
        var name = "John Doe";
        var occupation = "carpenter";
        info(name, occupation: occupation);
        info(name);
}
void info(String name, {String? occupation}) {
        print("$name is a $occupation.");
}
```

Output:

```
John Doe is a carpenter.
John Doe is a null.
```

3) ઓપ્શનલ પેરામીટર ડીફોલ્ટ વેલ્યુ સાથે (Optional Parameter with Default Value)

- ફંક્શન પેરામીટરને ડિફોલ્ટ વેલ્યુ પણ અસાઈન કરી શકાય છે.
- જો ફંક્શનને કોલ કરતી વખતે વેલ્યુ દર્શાવવામાં ન આવે તો તે ડિફોલ્ટ વેલ્યુ લેશે, પરંતુ જો ફંક્શનને કોલ કરતી વખતે વેલ્યુને દર્શાવવામાં આવે તો તે નવી વેલ્યુને ઓવરરાઈડ કરશે. જેથી કહી શકાય કે ફંક્શન **NULL safe** છે.
- ડિફોલ્ટ વેલ્યુ સાથે ઓપ્શનલ આર્ગ્યુમેન્ટને દર્શાવવા માટે {} કર્લી બ્રેકેટનો ઉપયોગ કરવામાં આવે છે.

Example:

```
void main() {
        var name = "John Doe";
        var occupation = "carpenter";
        var age = 22;
        info(name, occupation : occupation);
        info(name, age : age);
}
```

```
void info(String name, {String? occupation, int age = 20}) {
        print("$name is a $occupation and age is $age.");
}
```

Output:

```
John Doe is a carpenter and age is 20.
John Doe is a null and age is 22.
```

Anonymous Function (એનોનીમસ ફંક્શન)

- દરેક ફંક્શનને નામ અને રીટર્ન ટાઈપ હોય છે. જો ફંક્શનને નામ અને રીટર્ન ટાઈપ ન આપવામાં આવે તો ફંક્શન ને એનોનીમસ ફંક્શન **(anonymous function)** કહે છે.

Syntax:

```
(parameterList)
{
        // statements
}
```

Example:

```
void main(){
        var list = ["James","Patrick","Mathew","Tom"];
        print("Example of anonymous function");
        list.forEach((item){
                print("${list.indexOf(item)}: $item");
        });
}
```

Output:

```
Example of anonymous function
0: James
1: Patrick
2: Mathew
3: Tom
```

Collections (કલેક્શન)

- કલેક્શન એ ઓબ્જેક્ટનું ગ્રુપ છે જે ચોક્કસ એલીમેન્ટને દર્શાવે છે.
- Dart માં કલેક્શનને ઈમ્પલીમેન્ટ કરવા માટે **dart:collection** લાઈબ્રેરીનો ઉપયોગ થાય છે.
- કલેક્શન લાઈબ્રેરી નીચે મુજબ છે.
  1) List<E>
  2) Map<K, V>
  3) HashMap<K, V>

1) List<E>

- **લિસ્ટ** એ ઓબ્જેક્ટનું ઓર્ડર્ડ ગ્રુપ છે.
- Dart માં લિસ્ટ ડેટા ટાઈપ એ અન્ય પ્રોગ્રામિંગ લેંગ્વેજમાં **એરે** જેવો જ કોન્સેપ્ટ છે.
- **તેનો ઉપયોગ એક વેરીએબલમાં મલ્ટીપલ વેલ્યુને સ્ટોર કરવા માટે થાય છે.**
- એંગલ બ્રેકેટ (<>)ની અંદર ઓબ્જેક્ટ ટાઈપ હોય છે.
- કોઈપણ લિસ્ટ–આઈટમને એક્સેસ કરવા માટે ઈન્ડેક્ષનો ઉપયોગ થાય છે.
- લિસ્ટને નીચે મુજબ ક્લાસીફાય કરી શકાય છે.

1) ફિક્સ લેન્થ લિસ્ટ (Fixed Length List)
2) ગ્રોએબલ લિસ્ટ (Growable List)

1. Fixed Length List

- ફિક્સ લેન્થ લિસ્ટની લેન્થ રન ટાઈમે બદલી શકાતી નથી.

Syntax:

var list_name = new List.filled(initial_size, fill);

- **var** કીવર્ડ ઓટોમેટીકલી ડેટા ટાઈપ અલોકેટ કરે છે. તેનો અર્થ એ છે કે જ્યારે આપણે ડેટા ટાઈપને સ્પેસીફાય કરવા માંગતા નથી, ત્યારે તે ઓટોમેટીકલી ડેટા ટાઈપ અલોકેટ કરે છે.
- ઉપરોક્ત સિન્ટેક્ષ આપેલ સાઈઝનું લિસ્ટ બનાવે છે અને ડીફોલ્ટ વેલ્યુથી fill કરે છે.
- લિસ્ટને રનટાઈમે પર વધારી કે ઘટાડી શકાતું નથી.

Example:

```
void main() {
    var lst = new List.filled(3,0);
    lst[2] = 11;
    print("List is: $lst");
}
```

Output:

```
List is: [0, 0, 11]
```

2. Growable List

- ગ્રોએબલ લિસ્ટની સાઈઝ રનટાઈમે બદલી શકાય છે.
- લિસ્ટની પ્રોપર્ટી અને મેથડ નીચે મુજબ છે.

Properties and Methods	Description
indexOf(**value**)	લિસ્ટમાંથી આપેલ વેલ્યુની ઈન્ડેક્ષ રીટર્ન કરે છે.
Length	લિસ્ટની લેન્થ રીટર્ન કરે છે.
first	લિસ્ટનો first એલીમેન્ટ રીટર્ન કરે છે.
last	લિસ્ટનો last એલીમેન્ટ રીટર્ન કરે છે.
isEmpty	લિસ્ટ empty હોય તો **true** રીટર્ન કરે છે અને લિસ્ટ empty ન હોય તો **false** રીટર્ન કરે છે.
isNotEmpty	લિસ્ટ empty ન હોય તો **true** રીટર્ન કરે છે અને લિસ્ટ empty હોય તો **false** રીટર્ન કરે છે.
reversed	લિસ્ટ રિવર્સ ઓર્ડરમાં રીટર્ન કરે છે.
add(**value**)	લિસ્ટમાં એક એલીમેન્ટ ઉમેરે છે.
addAll(**[e1, e2,..]**)	આપેલ લિસ્ટમાં મલ્ટીપલ એલીમેન્ટ ઈન્સર્ટ કરે છે.
Insert(**index, value**)	આપેલ ઈન્ડેક્ષમાં એક એલીમેન્ટ ઈન્સર્ટ કરે છે.
insertAll(**index, [e1, e2,..]**)	આપેલ ઈન્ડેક્ષમાં મલ્ટીપલ એલીમેન્ટ ઈન્સર્ટ કરે છે.
replaceRange(**start, end, [e1, e2,..]**)	આપેલ લિસ્ટમાં મલ્ટીપલ એલીમેન્ટને **start થી end-1** સુધીની ઈન્ડેક્ષ વેલ્યુ રિપ્લેસ કરે છે.
remove()	આપેલ લિસ્ટમાંથી એક એલીમેન્ટ રીમુવ કરે છે.

removeAt(**index**)	આપેલ ઈન્ડેક્ષ પરના એક એલીમેન્ટને રીમુવ કરે છે.
removeLast()	આપેલ લિસ્ટમાંથી last એલીમેન્ટ રીમુવ કરે છે.
removeRange(**start, end**)	આપેલ લિસ્ટમાંથી મલ્ટીપલ એલીમેન્ટને **start થી end-1** સુધીની ઈન્ડેક્ષ વેલ્યુ રીમુવ કરે છે.

Example:

```
void main()
{
        List<String> names = ["Maxwell", "John", "Emily"]; // Mutable List
        List<int>  ages = [];

        print(names);
        print(names[0]);
        print("name at index 3 is: ${names[2]}");
        print("names list length is: ${names.length}");
        print("John index is: ${names.indexOf("John")}");
        print("First element of the List is: ${names.first}");
        print("Last element of the List is: ${names.last}");
        print("Is names Empty: "+names.isEmpty.toString());
        print("Is names not Empty: "+names.isNotEmpty.toString());
        print("List in reverse: ${names.reversed}");
        names.add("Bill");
        names.add("Elon");
        names.add("Sundar");
        print("Updated names list is: ${names}");

        ages.addAll([12, 14, 16, 18]);
        print("Updated ages list is: ${ages}");

        ages.insert(2, 35);
        print("Updated ages list is: ${ages}");

        ages.insertAll(1, [40, 37]);
        print("Updated ages list is: ${ages}");

        ages.replaceRange(0, 2, [55, 56, 57]);
        print("ages list after replace is: ${ages}");

        names.remove("Maxwell");
        print("Names list after removing element : ${names}");

        names.removeAt(1);
        print("Names list after removing element : ${names}");

        names.removeLast();
```

```
print("List after removing last element: ${names}");

ages.removeRange(0, 4);
print("List after removing range element: ${ages}");

ages.forEach((n) => print(n));}
```

Output:

```
[Maxwell, John, Emily]
Maxwell
name at index 3 is: Emily
names list length is: 3
John index is: 1
First element of the List is: Maxwell
Last element of the List is: Emily
Is names Empty: false
Is names not Empty: true
List in reverse: (Emily, John, Maxwell)
Updated names list is: [Maxwell, John, Emily, Bill, Elon, Sundar]
Updated ages list is: [12, 14, 16, 18]
Updated ages list is: [12, 14, 35, 16, 18]
Updated ages list is: [12, 40, 37, 14, 35, 16, 18]
ages list after replace is: [55, 56, 57, 37, 14, 35, 16, 18]
Names list after removing element : [John, Emily, Bill, Elon, Sundar]
Names list after removing element : [John, Bill, Elon, Sundar]
List after removing last element: [John, Bill, Elon]
List after removing range element: [14, 35, 16, 18]
14
35
16
18
```

2) Map<K, V>

- મેપ એ એક ઓબ્જેક્ટ છે જેમાં **કી–વેલ્યુ** પેર તરીકે ડેટા સ્ટોર થાય છે.
- દરેક કી–વેલ્યુ પેરમાં key યુનીક જ હોવી જોઈએ જ્યારે વેલ્યુ સરખી હોય શકે.
- મેપમાં કી અને વેલ્યુની ડેટા ટાઈપ કોઈપણ હોઈ શકે છે.
- તે એક ડાયનેમિક કલેક્શન છે.
- મેપની પ્રોપર્ટી અને મેથડ નીચે મુજબ છે.

Properties and Methods	Description
Keys	મેપની બધી કી રીટર્ન કરે છે.
Values	મેપની બધી વેલ્યુ રીટર્ન કરે છે.
IsEmpty	જો મેપ empty હોય તો **true** અને જો મેપ empty ન હોય તો **false** રીટર્ન કરે છે.

isNotEmpty	જો મેપ empty ન હોય તો **true** અને જો મેપ empty હોય તો **false** રીટર્ન કરે છે.
Length	મેપની લેન્થ રીટર્ન કરે છે.
keys.toList()	બધી મેપની કી ને લિસ્ટ માં કન્વર્ટ કરે છે.
values.toList()	બધી મેપની વેલ્યુ ને લિસ્ટ માં કન્વર્ટ કરે છે
containsKey(**'key'**)	જો મેપમાં આપેલ કી આવેલ હોય તો **true** અને જો મેપમાં આપેલ કી આવેલ ન હોય તો **false** રીટર્ન કરે છે.
containsValue(**'value'**)	જો મેપમાં આપેલ વેલ્યુ આવેલ હોય તો **true** અને જો મેપમાં આપેલ વેલ્યુ આવેલ ન હોય તો **false** રીટર્ન કરે છે.
clear()	મેપમાંથી બધા એલિમેન્ટ રીમુવ કરે છે.
removeWhere()	જો કંડીશન valid હોય તો મેપમાંથી બધા એલિમેન્ટ રીમુવ કરે છે.

Example:

```
void main(){
        Map<int, String> names = {1: 'Maxwell', 2: 'John', 3: 'Emily'};
        print(names);
        print(names[2]);
        print("All keys of Map: ${names.keys}");
        print("All values of Map: ${names.values}");
        print("Is Map empty: ${names.isEmpty}");
        print("Is Map not empty: ${names.isNotEmpty}");
        print("Length of map is: ${names.length}");
        names[4] = 'Bill';
        print(names);

        names[2] = 'Elon';
        print(names);

        print("All keys of Map with List: ${names.keys.toList()}");
        print("All values of Map with List: ${names.values.toList()}");

        print("Does Map contain key 1: ${names.containsKey(1)}");

        print("Does Map contain value John: ${names.containsValue("John")}");

        names.remove(1);
        print(names);

        print("Display using for..in loop");
        for(MapEntry names in names.entries){
                print('Key is ${names.key}, value ${names.value}');
```

```
}

print("Display using forEach loop");
names.forEach((key,value) => print('Key is $key and value is $value'));

names.removeWhere((key, value) => value == "Emily");
print(names);

names.clear();
print(names);}
```

Output:

```
{1: Maxwell, 2: John, 3: Emily}
John
All keys of Map: (1, 2, 3)
All values of Map: (Maxwell, John, Emily)

Is Map empty: false
Is Map not empty: true
Length of map is: 3
{1: Maxwell, 2: John, 3: Emily, 4: Bill}
{1: Maxwell, 2: Elon, 3: Emily, 4: Bill}
All keys of Map with List: [1, 2, 3, 4]
All values of Map with List: [Maxwell, Elon, Emily, Bill]
Does Map contain key 1: true
Does Map contain value John: false
{2: Elon, 3: Emily, 4: Bill}

Display using for..in loop
Key is 2, value Elon
Key is 3, value Emily
Key is 4, value Bill
Display using forEach loop
Key is 2 and value is Elon
Key is 3 and value is Emily
Key is 4 and value is Bill
{2: Elon, 4: Bill}
{}
```

3) HashMap<K, V>

- HashMap એ કી–વેલ્યુ પેરનો અન–ઓર્ડર્ડ (Un-ordered) કલેક્શન છે.
- Map અને HashMap સમાન છે.
- પરંતુ, HashMap એ ઈન્સર્શન ઓર્ડર (insertion order) ની ગેરંટી આપતું નથી.
- જો તમે કી A સાથે એન્ટ્રી ઈન્સર્ટ કરો છો અને પછી કી B સાથે બીજી એન્ટ્રી ઈન્સર્ટ કરો છો, જ્યારે તમે HashMapમાંથી ડેટા એક્સેસ કરો છો ત્યારે તમને પહેલા B એન્ટ્રી પણ મળી શકે છે.
- HashMap દ્વારા મેપની બધી પ્રોપર્ટી અને મેથડનો ઉપયોગ કરી શકાય છે.

Example:

```
import 'dart:collection';
void main()
{
        var hash = HashMap<int,String>();
        hash[5] = 'Sundar';
        hash[10] = 'Emily';
        hash[15] = 'Maxwell';
        print(hash);
        hash.forEach((key, value){
            print(key);
        });
}
```

Output:

```
{0: John, 10: Emily, 5: Sundar, 15: Maxwell}
0
10
5
15
```