DART LANGUAGE

PART-1

WHY USE DART?

- Before you can start developing Flutter apps, you need to understand the programming language used, namely, Dart. Google created Dart and uses it internally with some of its big products such as Google AdWords. Made available publicly in 2011, Dart is used to build mobile, web, and server applications.

DEVELOPING THE DART HELLO WORLD PROGRAM

To develop a Dart Hello World program:

- ✓ First, create a simple file with the path ***D:\dart\hello_world.dart***. The Dart source code files have the **.dart** extension.
- ✓ Second, use your preferred editor to open the ***hello_world.dart*** file, write the following code into the file, and save it:

```
void main() {
  print("hello , world");
}
```

DEVELOPING THE DART HELLO WORLD PROGRAM

- Third, open a terminal (CMD) and execute the following command to run the Dart file:

```
dart run d:\dart\hello_world.dart
```

- It'll show a message like this:

```
Hello, World!
```

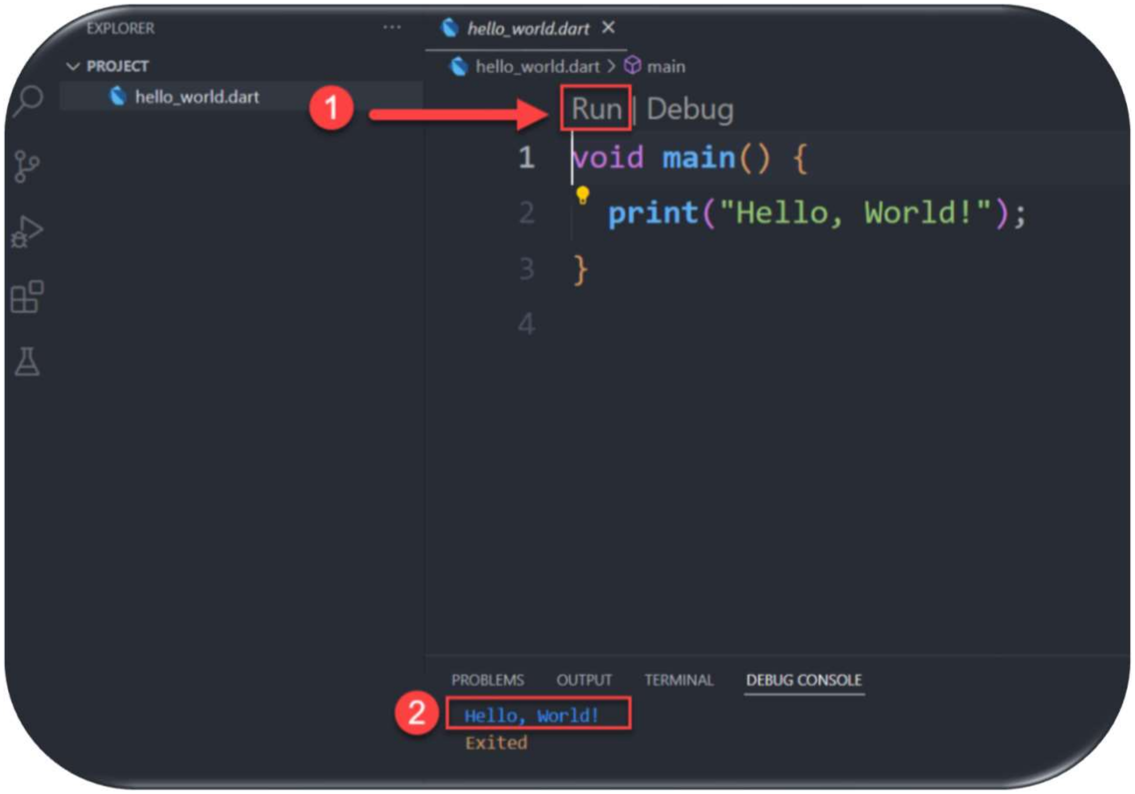
VISUAL STUDIO CODE

Visual Studio Code or VS Code is a popular tool for developing the Dart program.

- ✓ First, download and install VS Code.
- ✓ Second, install the Dart extension.

Once VS code is installed, you can open the project folder and click the Run button to run the **Hello, World!** program as shown in the following picture:

VISUAL STUDIO CODE



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

```
Hello, World!
Exited
```

- ➢ Go to **RUN** in the menu bar then Click **Run Without Debugging** or use Shortcut Keys **Ctrl+f5**

DART SYNTAX

- The Dart programming language follows the C-style syntax. If you're familiar with C/C++ or C#, you'll find similarities in the Dart language.

➢Statements

- A statement is an instruction that declares a type or instructs the program to perform a task. A statement is always terminated by a semicolon (;). For example, the following program has two statements:

DART SYNTAX

```
void main() {
  String message = 'Welcome to Dart!';
  print(message);
}
```

- ✓ The first statement declares a string variable and initializes its value to the string 'Welcome to Dart!':
- ✓ The second statement displays the value of the message variable to the console:

➢ BLOCKS

- ✓ In Dart, a block is a sequence of zero or more statements. A block is surrounded by curly braces **{ }** . For example, you can group statements into a block as follows:

```
 {
   String message = 'Welcome to Dart!';
   print(message);
}
```

- ✓ Unlike a statement, a block is not terminated by a semicolon (;). In practice, you'll use blocks with the control flow statements like if else, while, do while, and for.

➢ IDENTIFIERS

Identifiers are names that you assign to the variables, constants, functions, etc. In Dart, the names of identifiers follow these rules:

- ✓ The alphabetic ([a-z], [A-Z]) and underscore (_) characters can appear at any position.
- ✓ Digits (0-9) cannot be in the first position but everywhere else.
- ✓ Identifiers are case-sensitive. For example, message and Message identifiers are **different**.

➢ KEYWORDS

- ✓ Keywords are names that have a special meaning to the Dart compiler. All keywords are reserved identifiers. Therefore, you cannot use them as the names of identifiers.
- ✓ Example

else	**import**	**for**	**if**	**case**	**switch**	**do**	**null**
final	**try**	**get**	**set**	**function**	**continue**	**on**	**var**
default	**void**	**hide**	**with**	**class**	**async**	**static**	**Etc. …**

➢ COMMENTS

Comments help you to document your code. Dart has the following types of comments:

✓ **Single-line comments:** begin with //.

```
// The greeting message
String message = 'Welcome to Dart!'; // Single Comment
```

✓ **Block comments:** begin with /* and end with */.

```
/* you can write multi
line comment
*/
String message = 'Welcome to Dart!';
```

DART VARIABLES

- ✓ In programming, you **need to manage values** like numbers, strings, and Booleans. To store these values in programs, you use variables.
- ✓ A variable is an identifier that **stores a value** of a specific type. By definition, a variable is associated with a type and has a name.
- ✓ **The following syntax illustrates how to declare a variable:**

```
type   variableName;
```

DART VARIABLES

- ✓ **int** – represents whole numbers like -**1**, **0**, **1**, **2**.
- ✓ **double** – represents practical values like **-0.5**, and **9.98**.
- ✓ **String** – represents text such as **"Good Morning!".**
- ✓ **bool** – represents Boolean values including **true** and **false**.

```
void main() {
  int carSpeed = 240;
  print('Car Speed = : $carSpeed’); // print(carSpeed);
}
```

DART VARIABLES

In Dart, all variables are declared public (available to all) by default, but by starting the variable name with an underscore (_), you can declare it as private. By declaring a variable private,

Example

- ✓ String myname= ‘ abcd ‘ **//public**
- ✓ String _myname= ‘ abcd ‘ **//private**

DART VARIABLES

- ✓ What if the value of a variable doesn’t need to change? Begin the declaration of the variable with **final** or **const**. Use final when the value is assigned at runtime (can be changed by the user). Use const when the value is known at compile time (in code) and will not change at runtime.

- ✓ final String filter = 'company’;
- ✓ const String filter = 'company';

NUMBERS

- ✓ Declaring variables as numbers restricts the values to numbers only. Dart allows numbers to be int (integer) or (double).
- ✓ int counter = 0;
- ✓ double price = 0.0;

```
void main() {
    //declare a integer value
    int num1 = 1;
    // declare a double value
    double num2 = 1.5;
    print(num1);
    print(num2);
}
```

DART CONVERT -1

- ✓ To convert an integer into a double, you use the **int.toDouble()**

- ✓ To convert an double into an integer, you use the **double.toInt()**

```
void main() {
  int a = 10;
  double b = a.toDouble();
//convert int to Double
  print(a); //output 10
  print(b); //output 10.0

  double c = 4.99;
  int d = c.toInt();
//convert Double to int
  print(c); //output 4.99
  print(d); //output 4
}
```

DART STRING TYPE

- ✓ A string is a sequence of characters or more precisely a sequence of UTF-16 code units. Dart uses the **String** type to represent strings.
- ✓ To create a string literal, you can use either **single quotes or double quotes** like this:

```
void main() {
  String s1 = 'A single-quoted string';
  String s2 = "A double-quoted string";
}
```

DART STRING TYPE

✓ Dart allows you to place a variable in a string using the **$variableName** syntax.

```dart
void main() {
  String name = 'Ali';
  String message = 'Hello $name';
  print(message); // Output = Hello Ali
}
```

DART STRING TYPE

- To embed an expression in a string, you use the **${expression}** syntax. For example

```
void main() {
  var price = 10;
  var tax = 0.08;
  var message = 'The price with tax is ${price + price * tax}';
  print(message);
}
```

OUTPUT: The price with tax is 10.8

DART STRING TYPE

- ✓ Getting the length of a string
- ✓ To get the length of a string, you use the **length** property:

```
void main() {
  var message = 'Hello';
  print(message.length); //output = 5
}
```

DART STRING TYPE

- ✓ Accessing individual characters in a string
- ✓ To access an individual character in a string, you can use the square brackets **[ ]** with an index:
- ✓ String indexing is **zero-based**. It means that the first character in the string has an index of 0, the next is 1, and so on. For example:

```
void main() {
  var message = 'Hello’;

  print(message[0]); // H
  print(message[1]); // e
  print(message[2]); // l
  print(message[3]); // l
  print(message[4]); // o
}
```

DART MULTILINE STRINGS

To form a multiline string, you use **triple quotes**, either single or double quotes. For example:

```
void main() {
  var sql = '''select phone
  from phone_books
  where name =?''';
  print(sql);
}
```

Output:

```
select phone
  from phone_books
  where name
```

DART CONVERT -2

- ✓ To convert a **string into an integer**, you use the **int.parse()** method.
- ✓ To convert a **string to a double**, you use the **double.parse()** method.

```
void main() {
  int qty = 5;
  String amount = "100";
  int total = qty * int.parse(amount);
  print('Total: $total'); //output : Total: 500

  String priceStr = "1.55";
  double price = double.parse(priceStr);

  print(price);//output : 1.55
}
```

DYNAMIC AND VAR

dynamic: can change TYPE of the variable, & can change VALUE of the variable later in code. var: can't change TYPE of the variable, but can change VALUE of the variable later in code.

Dynamic	var
Dynamic age=12	Var age= 12
Age='abcd' ☑	Age =23 ☑
Age =12 ☑	Age ='abcd' ☒

BOOLEANS

✓ Declaring variables as bool (Boolean) allows a value of **true** or **false** to be entered.

bool isDone = false;

isDone = true;

```
void main() {
  bool val = true;
  print(val);
}
```

OPERATORS

- Dart language supports all operators, as you are familiar with other programming languages such as C, Kotlin, and Swift. The operator's name is listed below:

- ✓ Arithmetic
- ✓ Equality
- ✓ Increment and Decrement
- ✓ Logical
- ✓ Comparison

ARITHMETIC OPERATORS

The following table shows the arithmetic operators supported by Dart.

Operators & Meaning
+ Add
– Subtract
***** Multiply
/ Divide
% Get the remainder of an integer division (modulo)
++ Increment
-- Decrement
~/ To get only the integer part

EQUALITY AND RELATIONAL OPERATORS

Relational Operators tests or defines the kind of relationship between two entities.

Relational operators return a Boolean value i.e. true/ false.

Operator	Description	Example (A=10 , B=20)
>	Greater than	(A > B) is False
<	Lesser than	(A < B) is True
>=	Greater than or equal to	(A >= B) is False
<=	Lesser than or equal to	(A <= B) is True
==	Equality	(A==B) is False
!=	Not equal	(A!=B) is True

LOGICAL OPERATORS

Logical operators are used to combine two or more conditions. Logical operators return a Boolean value.

Operator	Description	Example (A=10,B=20)
&&	**And** – The operator returns true only if all the expressions specified return true	(A > 10 && B > 10) is False.
||	**OR** – The operator returns true if at least one of the expressions specified return true	(A > 10 || B > 10) is True.
!	**NOT** – The operator returns the inverse of the expression's result. For E.g.: !(7>5) returns false	!(A > 10)

CONDITIONAL EXPRESSIONS

Dart has two operators that let you evaluate expressions that might otherwise require ifelse statements –

condition ? expr1 : expr2

If condition is true, then the expression evaluates **expr1** (and returns its value); otherwise, it evaluates and returns the value of **expr2**.

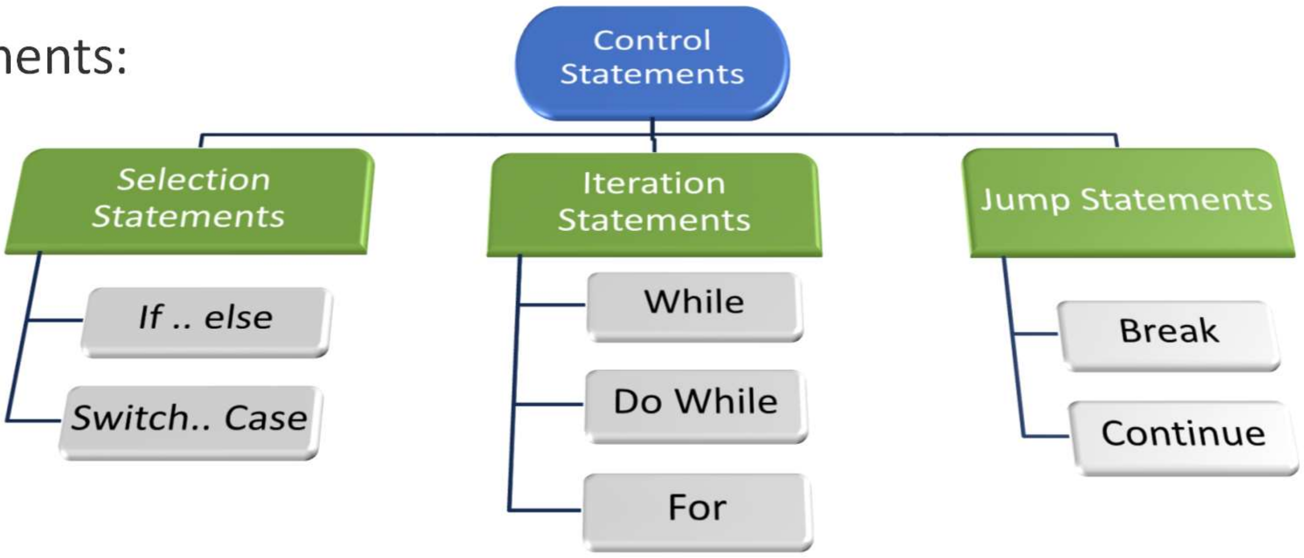
```
void main() {
   var a = 10;
   var res = a > 12 ? "greater than 10":"lesser than or equal to 10";
   print(res);
}
```

DART LANGUAGE

PART-2

DECISION MAKING AND LOOPS

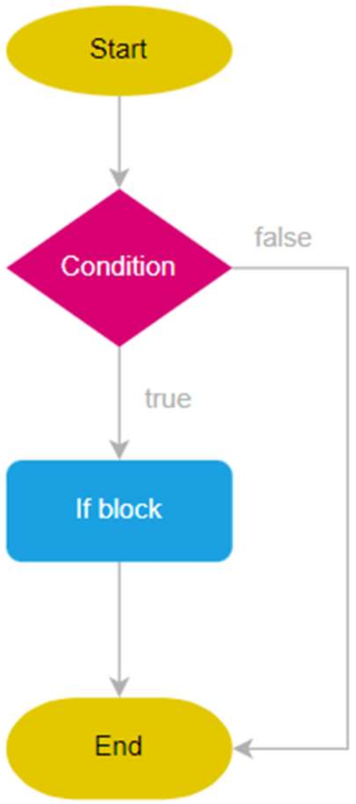
The decision-making is a feature that allows you to evaluate a condition before the instructions are executed. The Dart language supports the following types of decision-making statements:



STATEMENTS SELECTION

Dart **If** is a simple conditional statement where a block of statements get executed if the given Boolean expression evaluates to true.

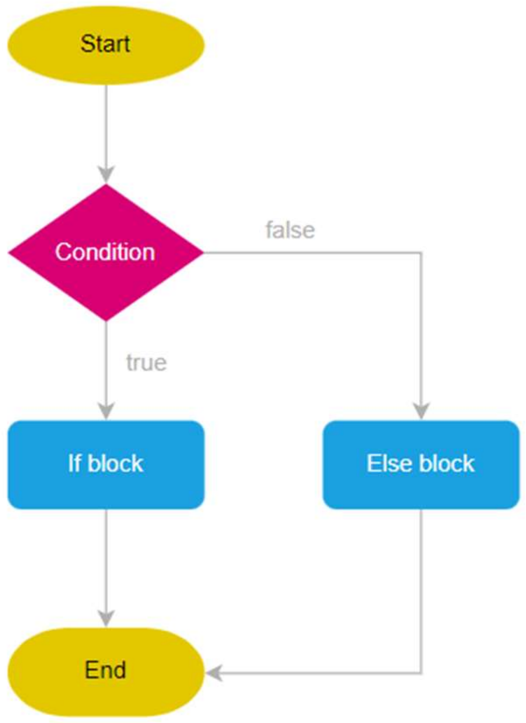
```
void main() {
  bool HaveAcar = true;
  if (HaveAcar) {
    print(" I Have a BMW ");
    print(“models 2011");
  }
}
```



STATEMENTS SELECTION

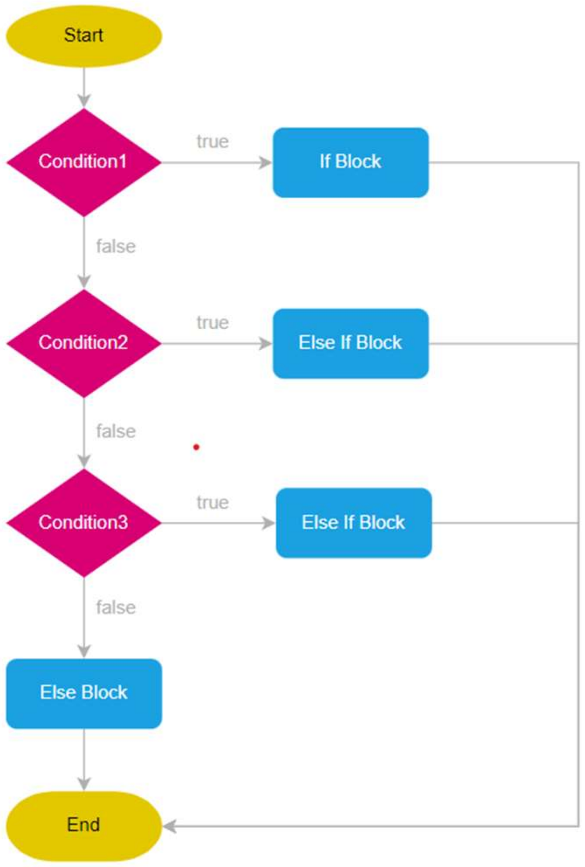
Dart **If-Else** statement contains two blocks. If block and Else block.

```
void main() {
  bool HaveAcar = false;
  if (HaveAcar) {
    print(" I Have a BMW ");
  } else {
    print("I don\'t have enough money");
  }
}
```



STATEMENTS SELECTION

Dart **If-Else-If** statement is and extension to if-else statement. If-Else-If contains more than one Boolean expression.

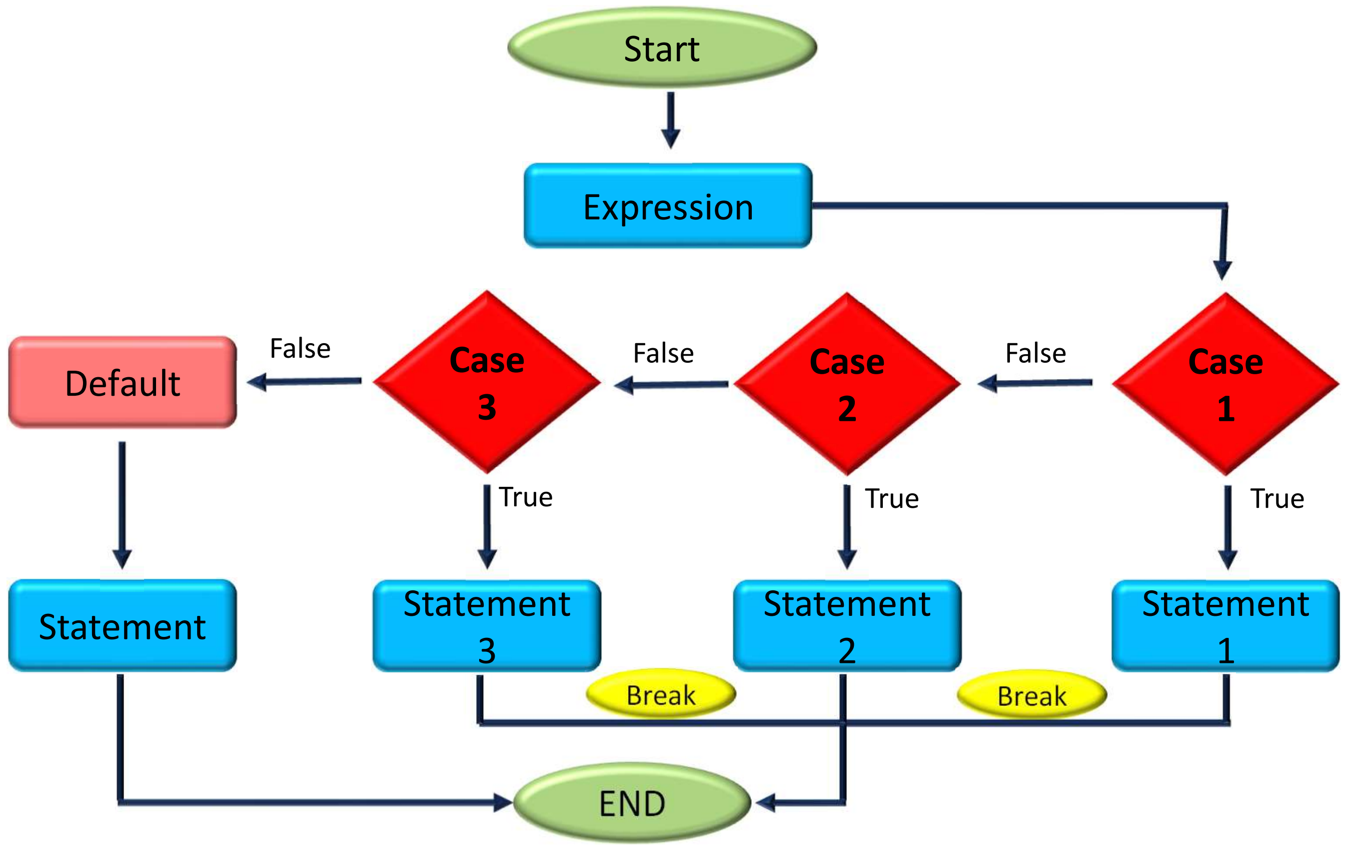


STATEMENTS SELECTION

```
void main() {
  String season = "";
  String month = "Sep";

  if (month == "Jan" || month == "Feb" || month == "March") {
    season = "Spring";
  } else if (month == "Apr" || month == "Jun" || month == "July") {
    season = "Summer";
  } else if (month == "Aug" || month == "Sep" || month == "Oct") {
    season = "Autumn";
  } else if (month == "Nov" || month == "Dec" || month == "Jan") {
    season = "Winter";
  } else {
    season = "this month doesn\'t exist";
  }
  print(season);
}
```

STATEMENTS SELECTION

The **switch** statement evaluates an expression, matches the expression's value to a **case** clause and executes the statements associated with that **case**.



STATEMENTS SELECTION

```
void main() {
  const weather = "cloudy";
  switch (weather) {
    case "sunny":
      print("Its a sunny day. Put sunscreen.");
      break;
    case "snowy":
      print("Stay at Home");
      break;
    case "cloudy":
    case "rainy":
      print("Please bring umbrella.");
      break;
    default:
      print("Sorry I am not familiar with such weather.");
  }}
```

EXAMPLE

```
void main() {
  int a = 13;

  if (a % 2 == 0) {
    print('$a is even number.');
  } else {
    print('$a is odd number.');
  }
}
```

```
void main() {
  int a = -12;
  if (a < 0) {
    print('$a is negative number.');
  } else if (a == 0) {
    print('$a is zero. Neither negative nor
positive');
  } else {
    print('$a is positive number.');
  }}
```

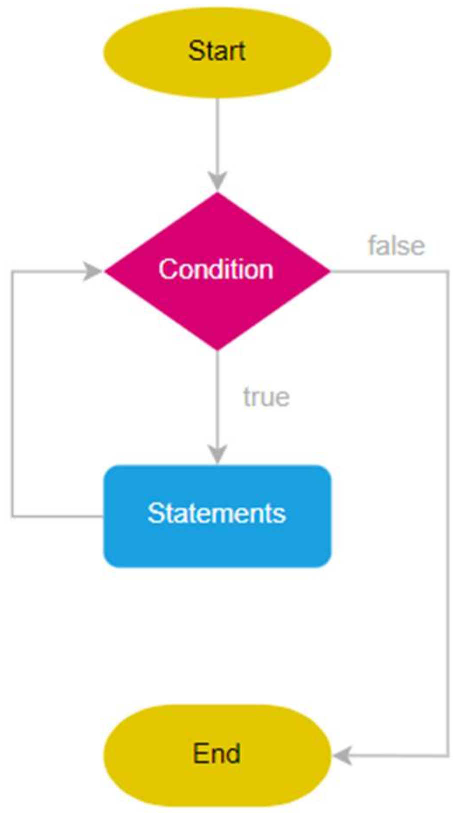
EXAMPLE

```
void main() {
   var grade = "A";
   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"); }
   } }
```

ITERATION STATEMENTS (LOOPS)

The **while** statement evaluates a Boolean expression and executes statements repeatedly as long as the result of the expression is **true**.

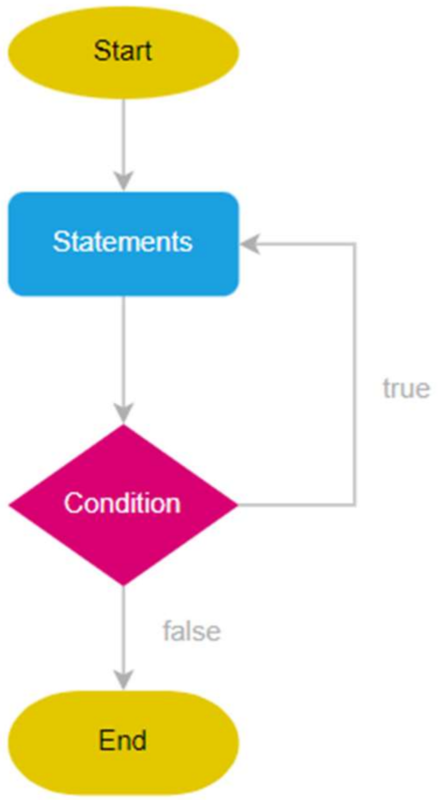
```
void main() {
  int current = 0;

  while (current < 5) {
    current++;
    print(current);
  }
}
```



ITERATION STATEMENTS (LOOPS)

The Dart **do while** statement executes statements as long as a condition is **true**. Here’s the example of the do while statement:

```
void main() {
  int number = 0; //change number to 6

  do {
    number++;
    print(number);
  } while (number < 5);
}
```

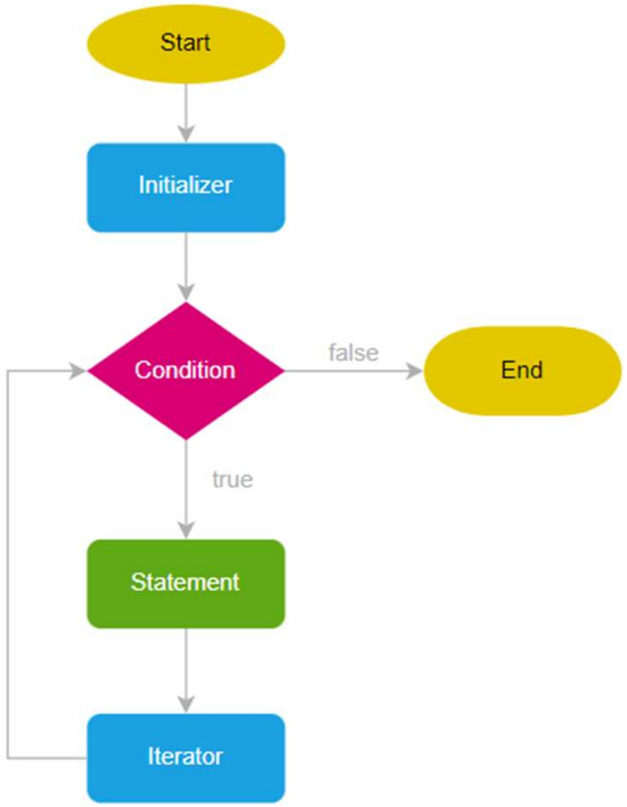


ITERATION STATEMENTS (LOOPS)

Dart **for** statement executes statements a fixed number of times. Here's the syntax of the for statement:

```
for(initializer; condition; iterator)
{
    // statement
}
```

```
void main() {
  int total = 0;
  for (var i = 1; i <= 10; i++) {
    total += i;
  } print("result is : $total");
}
```



COLLECTIONS

Dart doesn't support the array to store the data, unlike the other programming language. We can use the Dart collection in place of array data structure.

Dart collection can be classified as follows:

- ✓ List
- ✓ Set
- ✓ Maps

LIST

A **list** is an indexable collection of objects with a length. A list may contain **duplicate** elements and **null**. Dart uses the **List<E>** class to manage lists.

- The following **creates** an empty list that will store integers:

```
List scores = [];
```

```
List<DataType> scores = [];
```

you can move the type to the right-hand side:

```
List scores = <DataType> [];
```

LIST

```
void main() {
 bool check = true;
  List score = [1, 3.14, "text", check];
  print(score); //Displaying a list
}
```

```
Output: [1,3.14, text, true]
```

LIST

Accessing elements

Lists are zero-based indexing. It means that the first element has an index of 0, the second element has an index of 1, and so on.

```
void main() {

  List<String> G_Blood = ["A+", "B+", "O+", "A-", "B-", "O-"];

  print(G_Blood[3]);
}
```

INSERTING ELEMENTS INTO A LIST

- ✓ The **List.add()** function appends the specified value to the end of the List and returns a modified List object.
- ✓ The **List.addAll()** function accepts multiple values separated by a comma and appends these to the List.
- ✓ The **insert()** function accepts a value and inserts it at the specified index. Similarly, the **insertAll()** function inserts the given list of values, beginning from the index specified.

```
List.insert(index,value)
List.insertAll(index, iterable_list_of _values)
```

INSERTING ELEMENTS INTO A LIST

```
void main() {
  List Add_Element = [1, 2, 3];
  Add_Element.add(4); //output [1,2,3,4]
  Add_Element.addAll([5, 6, 7]); //output [1,2,3,4,5,6,7]
  Add_Element.insert(0, 999); //output [999,1,2,3,4,5,6,7]
  Add_Element.insertAll(0, [-2, -1]);
  print(Add_Element); //output [-2,-1,999,1,2,3,4,5,6,7]
}
```

UPDATING INDEX IN LIST

- ✓ The List class from the dart:core library provides the replaceRange() function to modify List items. This function replaces the value of the elements within the specified range.

```
List.replaceRange(int start_index,int end_index,Iterable <items>)
```

UPDATING INDEX IN LIST

```
void main() {
  List Update_Element = [1, 2, 3, 4, 5, 6, 7, 8, 9];
  print('The value of list before replacing ${Update_Element}');

  Update_Element.replaceRange(0, 3, [21, 23, 24]);
  print('replacing the items between the range [0-3] is ${Update_Element}');

  Update_Element[5] = -1;
  print("another ways to update the index ${Update_Element}");
}
```

REMOVING LIST ITEMS

✓ List.remove()

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

List.remove(value)

```
void main() {
  List remove_Element = [5, 6, 7, 8, 9];
  print("Display the list before removing $remove_Element");
  remove_Element.remove(7);
  print("Display the list After removing $remove_Element");
}
```

REMOVING LIST ITEMS

- ✓ List.removeAt()

The List.removeAt function removes the value at the specified **index** and returns it.

List.removeAt(int index)

```
void main() {
  List remove_Element = [5, 6, 7, 8, 9];
  print("Display the list before removing $remove_Element");
  remove_Element.removeAt(2); //error if Index out of Range
  print("Display the list After removing $remove_Element");
}
```

REMOVING LIST ITEMS

✓ List.removeLast()

The **List.removeLast()** function **pops** and returns the last item in the List. The syntax for the same is as given below

List.removeLast()

```
void main() {
  List remove_Element = [5, 6, 7, 8, 9];
  print("Display the list before removing $remove_Element");
  remove_Element.removeLast( );
  print("Display the list After removing $remove_Element");
}
```

REMOVING LIST ITEMS

✓ List.removeRange()

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

```
List.removeRange( int Start , int End)
```

Start – represents the starting position for removing the items.

End – represents the position in the list to stop removing the items.

REMOVING LIST ITEMS

```
void main() {
  List remove_Element = [1, 2, 3, 4, 5, 6, 7, 8, 9];
  print('before removing the list element ${remove_Element}');
  remove_Element.removeRange(2, 5);
  print('after removing the list element between the range 2-5
${remove_Element}');
}
```

LIST PROPERTIES

- ✓ To get the number of elements of a list, you use the **List.length** property.
- ✓ To access the **first** and **last** elements of a list, you use the **List.first** and **List.last** properties.
- ✓ To check if a list contains any elements, you can use the **List.isEmpty** or **List.isNotEmpty** property. That is a Boolean property return **true** or **false**.

LIST PROPERTIES

```
void main() {
  List prop = [2, 4, 6, 8, 10];
  print(prop.length); // 5
  print(prop.first); // 2
  print(prop.last); // 10
  print(prop.isEmpty); // false
  print(prop.isNotEmpty); // true
}
```

ITERATING OVER LIST ELEMENTS

```dart
void main() {
  var scores = [1, 3, 4, 2, 5];
  for (var i = 0; i < scores.length; i++) {
    print(scores[i]);
  }
}
```

```dart
void main() {
  var scores = [1, 3, 4, 2, 5];

  scores.forEach((i) => print(i));
}
```

```dart
void main() {
  var scores = [1, 3, 4, 2, 5];
  for (var i in scores) {
    print(i);
  }
}
```

```dart
void main() {
  var scores = [1, 3, 4, 2, 5];

  scores.forEach(print);

}
```

SET

A **set** is a collection of **unique** elements. Unlike a list, a set **doesn’t allow duplicates**.

Typically, a set is faster than a list, especially when working with large elements.

```
Set scores = { };
```

```
Set<DataType> scores = { };
```

```
Set scores = <DataType> { };
```

```
void main() {

Set ratings = {1, 2, 3,4,5,3,6};
print(ratings);
}
//Output:
//1 , 2 , 3 , 4 , 5 , 6
```

SET

- **Getting the number of elements**

To find the number of elements of a set, you use the **length property**.

- **Accessing an element by index**

Unlike a list, you cannot access an element at an index using square brackets []. Instead, you can use the **elementAt()** method

- Also, you can use the first and last property to access the **first** and **last elements** respectively.
- **Adding an element to a set**

To add an element to a set, you use the **add()** method. For example, the following uses the add()

- **Adding multiple elements**

To add multiple elements from a list to a set, you use the **addAll()** method.

SET

- **Checking the existence of elements**

To check if an element is in a set, you use the **contains()** method. The contains() method returns **true** if a set contains an element. Otherwise, it returns **false**.

- **Finding the intersection of two sets**

The **intersection** of two sets returns a set that contains the elements that are in both sets.

- **Finding the union of two sets**

The **union** of two sets returns unique elements that are in both sets.

EXAMPLE

```
void main() {
  Set A = {10, 20, 30,40};

  print(A.length); // length of set =4
  print(A.elementAt(1));// output = 20
  print(A.first);// output = 10
  print(A.last);// output = 40
  A.addAll([50,60,20]);
  print(A);// output = 10,20,30,40,50,60
  print(A.contains(30));// true

  Set B ={10,20,70,80};
  print(A.intersection(B));// output = 10 ,20
  print(A.union(B));// output = 10, 20, 30, 40, 50, 60, 70, 80
 }
```

ITERATING OVER ELEMENTS OF A SET

```
void main() {
  var ratings = {1, 2, 3, 4, 5};
  for (var i in ratings) {
    print(i);
  }
}
```

```
void main() {
  var ratings = {1, 2, 3, 4, 5};
  for (var i = 0; i < ratings.length; i++)
{
    print(ratings.elementAt(i));
}}
```

Any Questions