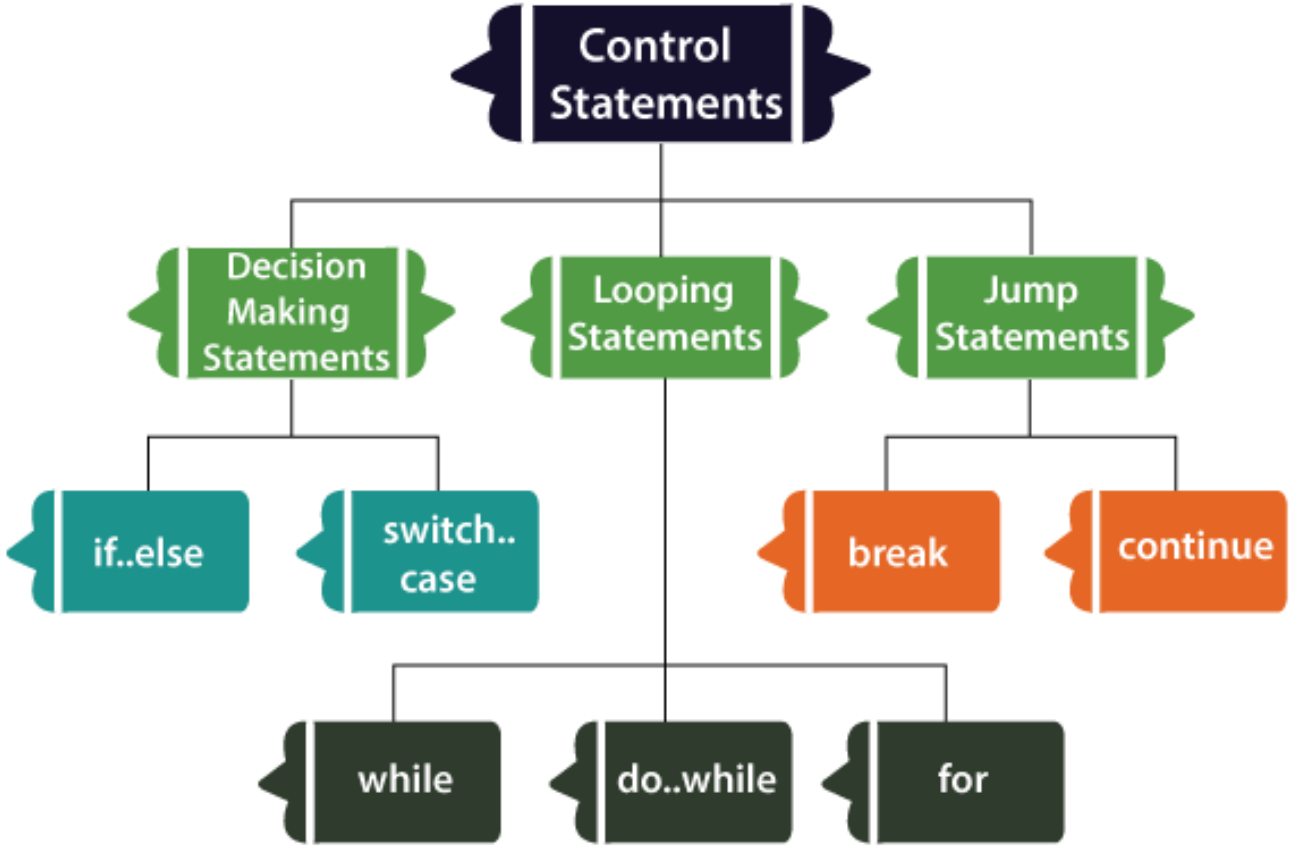
Dart Control Flow Statement

The control statements or flow of control statements are used to control the flow of Dart program. These statements are very important in any programming languages to decide whether other statement will be executed or not. The code statement generally runs in the sequential manner. We may require executing or skipping some group of statements based on the given condition, jumps to another statement, or repeat the execution of the statements.



Dart if Statements

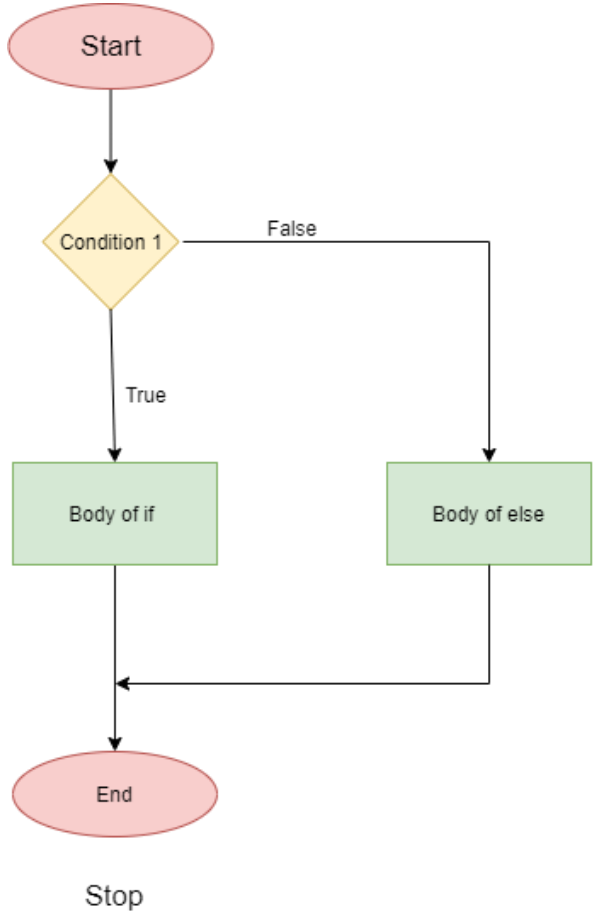
If statement allows us to a block of code execute when the given condition returns true. In Dart programming, we have a scenario where we want to execute a block of code when it satisfies the given condition. The condition evaluates Boolean values TRUE or FALSE and the decision is made based on these Boolean values.

```
void main() {
  // define a variable which hold numeric value
  var n = 35;

  // if statement check the given condition
  if (n < 40) {
    print("The number is smaller than 40");
  }
}
```

Dart if-else Statement

In Dart, if-block is executed when the given condition is true. If the given condition is false, else-block is executed. The else block is associated with the if-block.



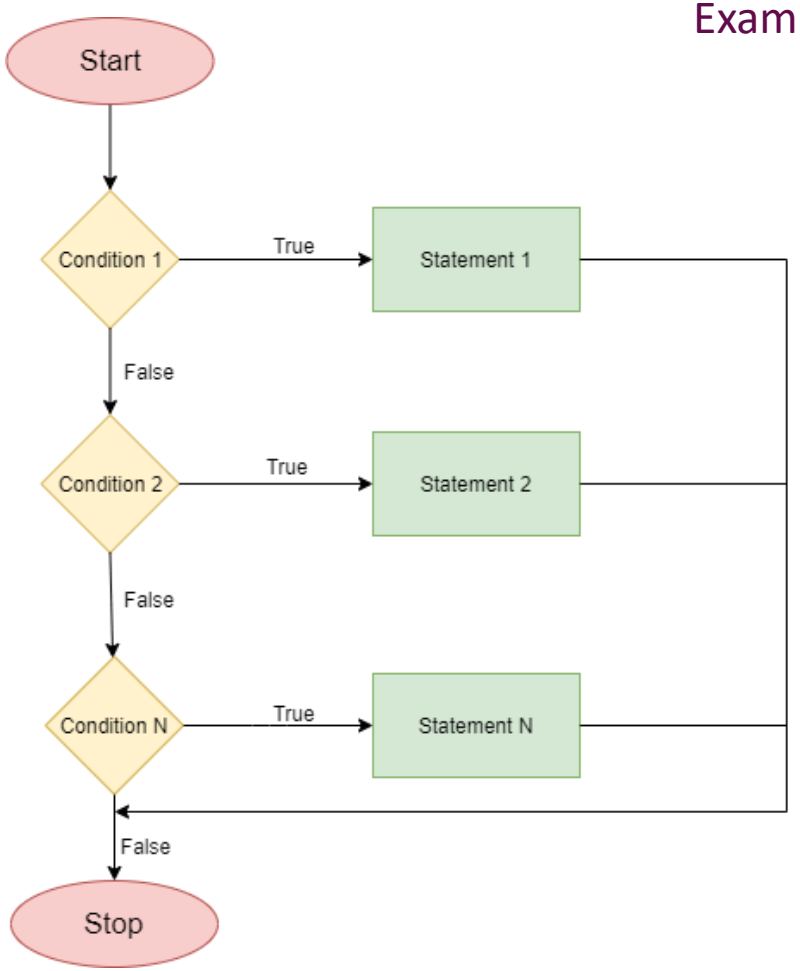

```dart
void main() {
  var x = 20;
  var y = 30;
  print("if-else statement example");

  if (x > y) {
    print("x is greater than y");
     }
  else {
    print("y is greater than x");
     }
}
```

Dart if else-if Statement

Dart if else-if statement provides the facility to check a set of test expressions and execute the different statements. It is used when we have to make a decision from more than two possibilities.

Example - Write a program to print the result based on the student's marks.




```dart
void main() {
  var marks = 74;
  if (marks > 85) {
    print("Excellent");
  } else if (marks > 75) {
    print("Very Good");
  } else if (marks > 65) {
    print("Good");
  } else {
    print("Average");
  }
}
```

Nested If else Statement

Dart nested if else statement means one if-else inside another one. It is beneficial when we need a series of decisions. Let's understand the following example.

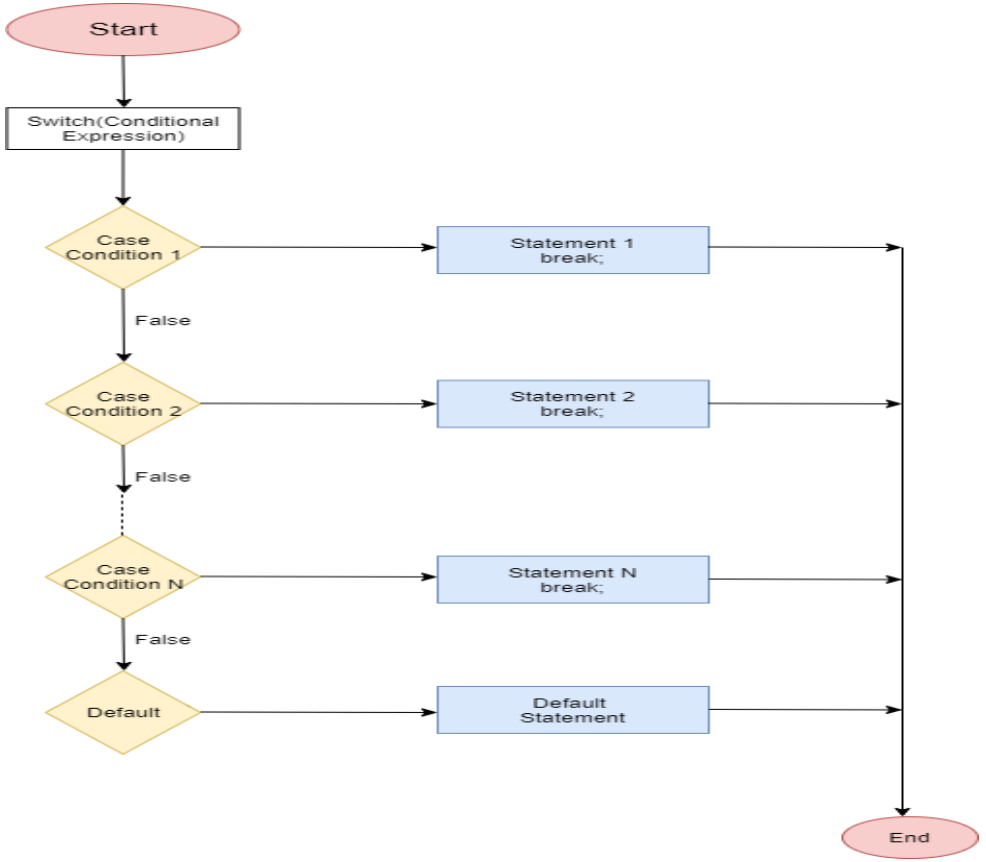
Example - Write a program to find the greatest number.

```
void main() {
  var a = 10;
  var b = 20;
  var c = 30;
  if (a > b) {
    if (a > c) {
      print("a is greater");
    } else {
      print("c is greater");
    }
  } else if (b > c) {
    print("b is greater");
  } else {
    print("c is greater");
  }}
```

Dart Switch Case Statement

Dart Switch case statement is used to avoid the long chain of the if-else statement. It is the simplified form of nested if-else statement. The value of the variable compares with the multiple cases, and if a match is found, then it executes a block of statement associated with that particular case.

The assigned value is compared with each case until the match is found. Once the match found, it identifies the block of code to be executed.



```
switch( expression )  {
 case value-1: {
       //statement(s)
      Block-1;
         }
  break;
 case value-2: {
       //statement(s)
       Block-2;
     }
 break;
 case value-N: {
        //statement(s)
        Block-N;
      }
   break;
 default:    {
       //statement(s);
  } }
```

The expression can be integer expression or character expression. The value 1, 2, n represents the case labels and they are used to identify each case particularly. Each label must be ended with the colon(:).

The labels must be unique because same name label will create the problem while running the program.

A block is associated with the case label. Block is nothing but a group of multiple statements for a particular case.

Once the switch expression is evaluated, the expression value is compared with all cases which we have defined inside the switch case. Suppose the value of the expression is 2, then compared with each case until it found the label 2 in the program.

The **break statement** is essential to use at the end of each case. If we do not put the break statement, then even the specific case is found, it will execute all the cases until the program end is reached. The **break** keyword is used to declare the break statement.

Sometimes the value of the expression is not matched with any of the cases; then the default case will be executed. It is optional to write in the program.

```
void main() {
   int std_num = 22012385;
  switch (std_num) {
    case 21990009:
      print("My name is Ali");
      break;
    case 22190010:
      print("My name is Omer");
      break;
    case 21890011:
      print("My name is Sami");
      break;
// default block
    default:
      print("Student number is not found");
  }
}
```

Dart Loops

Dart Loop is used to run a block of code repetitively for a given number of times or until matches the specified condition. Loops are essential tools for any programming language. It is used to iterate the Dart iterable such as list, map, etc. and perform operations for multiple times. A loop can have two parts - a body of the loop and control statements. The main objective of the loop is to run the code multiple times. Dart supports the following type of loops.

1. **Dart for loop**

The for loop is used when we know how many times a block of code will execute.

```
void main() {
    for (int i=0; i <= 10; i++) //for loop to print 1-10 numbers
  {
    print(i); //to print the number
  }
}
```

Nested for Loop

The nested for loop means, "the for loop inside another for loop". A for inside another loop is called an inner loop and outside loop is called the outer loop. In each iteration of the outer loop, the inner loop will iterate to entire its cycle.

```
void main() {
  int i, j;
  int table_no = 2;
  int max_no = 10;
  for (i = 1; i <= table_no; i++) {
    // outer loop
    for (j = 0; j <= max_no; j++) {
      // inner loop
      print("${i} * ${j} = ${i * j}");
        }
      print("\n");
  }
}
```

```
1 * 0 = 0
1 * 1 = 1
1 * 2 = 2
1 * 3 = 3
1 * 4 = 4
1 * 5 = 5
1 * 6 = 6
1 * 7 = 7
1 * 8 = 8
1 * 9 = 9
1 * 10 = 10

2 * 0 = 0
2 * 1 = 2
2 * 2 = 4
2 * 3 = 6
2 * 4 = 8
2 * 5 = 10
2 * 6 = 12
2 * 7 = 14
2 * 8 = 16
2 * 9 = 18
2 * 10 = 20
```

Dart for… in Loop

The for…in loop is slightly different from the for loop. It only takes dart object or expression as an iterator and iterates the element one at a time. The value of the element is bound to var, which is and valid and available for the loop body. The loop will execute until no element left in the iterator.

```
void main() {
  var list1 = [10, 20, 30, 40, 50];
  for (var i in list1) //for..in loop to print list element
  {
    print(i); //to print the number
  }
}
```

Dart while loop

The while loop executes a block of code until the given expression is false. It is more beneficial when we don't know the number of execution.

```
void main() {
  var a = 1;
  var maxnum = 10;
  while (a < maxnum) {
    // it will print until the expression return false
    print(a);
    a++; // increase value 1 after each iteration
  }
}
```

Dart do…while Loop

The do…while loop is similar to the while loop but only difference is that, it executes the loop statement and then check the given condition.

```
void main() {
  var a = 1;
  var maxnum = 10;
  do {
    print("The value is: ${a}");
    a = a + 1;
  } while (a < maxnum);
}
```