**CONTROL FLOW**

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.

**CATEGORIES OF FLOW STATEMENT**

In Dart, Control flow statement can be categorized mainly in three following ways.

* Decision-making statements
* Looping statements
* Jump statements

**Decision-making Statements**

Dart provides following types of Decision-making statement.

1. If Statement: executes a block of code if a specified condition is true.

// if statement outputs depending on a certain conditional expression

void main() {

  var age = 30;

  if (age > 18) {

    print("Mariam is a voter in Kenya");

  }

}

/\*OUTPUT:

  Mariam is a voter in Kenya

\*/

1. If-else Statements: In Dart, the if-else statement is used to execute a block of code based on whether a condition is true or false.

Here's the syntax:

void main() {

  var age = 20;

  if (age > 18) {

    print("Mariam is a voter in Kenya");

  }

  else {

    print("Mariam is still young to vote");

  }

}

/\*OUTPUT:

  Mariam is a voter in Kenya

\*/

1. If-else if-else Statement: In Dart, the if-else if-else statement allows you to evaluate multiple conditions and execute different blocks of code based on these conditions.

Here's the syntax:

void main() {

  var age = 18;

  if (age > 18) {

    print("Mariam is a voter in Kenya");

  } else if (age == 18) {

    print("Mariam just became eligible to vote in Kenya");

  } else {

    print("Mariam is still young to vote");

  }

}

/\*

If age is greater than 18, the statement "Mariam is a voter in Kenya" will be printed.

If age is exactly 18, the statement "Mariam just became eligible to vote in Kenya" will be printed.

If neither of the above conditions is met, the statement "Mariam is still young to vote" will be printed.

\*/

1. Switch Case Statement: In Dart, the switch statement is used to evaluate an expression and execute different blocks of code based on matching cases.

The syntax of the switch statement is as follows:

void main() {

  int i = 11;

  switch (i) {

    case 1:

      print("The value is 1");

      break;

    case 2:

      print("The value is 2 ");

      break;

    case 3:

      print("the value is 3");

      break;

    default:

      print("The value is out of range ");

      break;

  }

}

//The code checks the value of variable i using a switch statement and prints a message based on its value, defaulting to an "out of range" message if none match.

// OUTPUT: The value is out of range

**LOOPING STATEMENTS**

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.

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

void main()

{

    int num = 1;

    for(num; num<=3; num++)           //for loop to print 1-10 numbers

    {

        print(num);     //to print the number

    }

}

// The code prints numbers from 1 to 3 using a for loop in Dart.

1. Dart for…in loop

The for..in loop is similar to for loop but different in its syntax. It iterates through an object's properties. The Dart for..in loop accepts an expression as iterator and iterates through the elements one at a time in sequence. The variable var holds the values of the iteration. The for…in will execute until elements remain in iterators.

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

    }

}  //The code iterates over elements in the list list1 using a for-in loop and prints each element.

1. Dart while loop

The while loop is used when the number of execution of a block of code is not known. It will execute as long as the condition is true. It initially checks the given condition then executes the statements that are inside the while loop. The while loop is mostly used to create an infinite loop.

void main()

{

  var a = 1;

  var maxnum = 11;

  while(a<maxnum){ // it will print until the expression return false

    print(a);

    a = a+1; // increase value 1 after each iteration

}

}

//The code prints numbers from 1 to 10 using a while loop in Dart.

1. Dart do-while loop

Dart do while loop executes a block of the statement first and then checks the condition. If the condition returns true, then the loop continues its iteration. It is similar to Dart while loop but the only difference is, in the do-while loop a block of statements inside the body of loop will execute at least once.

void main() {

  var a = 1;

  var maxnum = 10;

  do {

    print("The value is: ${a}");

    a = a + 1; // adding 1 to variable a after every sequence

  } while (a < maxnum);

}

//The code iteratively prints the value of variable a until it reaches 10, using a do-while loop in Dart.

**ACCEPTING USER’S INPUT**

The exclamation mark (`!`) in the `stdin.readLineSync()!` method in Dart serves a specific purpose related to **null safety**. Let me explain:

1. **Null Safety in Dart**:

- Dart introduced null safety to help developers write more robust and reliable code.

- With null safety, variables can be categorized into two types:

- **Nullable**: These variables can hold either a value or `null`.

- **Non-nullable**: These variables are guaranteed to always have a value (i.e., not `null`).

2. **Nullable and Non-nullable Types**:

- In Dart, function calls can return nullable types (e.g., `String?`, which means the result can be either a string or `null`).

- When you call a function that returns a nullable type, you can use the exclamation mark (`!`) to **cast away nullability** and assert that the value will never be `null`.

3. **Specifically for `stdin.readLineSync()`:**

- The `stdin.readLineSync()` method reads a line of input from the standard input stream (stdin).

- By default, it returns a nullable string (`String?`) because the input might be empty or the user might press Ctrl+D (EOF) to signal the end of input.

- When you use `stdin.readLineSync()!`, you are telling Dart that you are absolutely sure the result will never be `null`.

- If, for some reason, the method does return `null`, your app will crash with an exception.

4. **Use with Caution**:

- Be cautious when using the exclamation mark (`!`).

- Only use it if you are certain that the value will never be `null`.

- If there's any doubt, consider using the safe navigation operator (`?.`) instead, which allows you to call methods on nullable objects without risking a crash.

Remember, the exclamation mark is a powerful tool, but it comes with responsibility. Use it wisely! 🚀