Conditional Statements in Dart

Conditional statements are used to make decisions in a program.
They allow a program to execute a block of code **only if a specific condition is true**.

1. if Statement

The **if statement** executes a block of code only when the given condition is true.

Syntax:

```
if (condition) {
  // code to execute if condition is true
}
```

Example:

```
void main() {
  int age = 20;

  if (age >= 18) {
    print("You are eligible to vote.");
  }
}
```

Output:

```
You are eligible to vote.
```

Explanation:
The code inside the `if` block runs only if the condition `age >= 18` is true.

2. if-else Statement

The **if-else statement** allows two different code blocks —
one runs if the condition is true, and the other if it's false.

Syntax:

```
if (condition) {
  // runs if condition is true
} else {
  // runs if condition is false
}
```

Example:

```
void main() {
  int marks = 45;

  if (marks >= 50) {
    print("You passed the exam.");
  } else {
    print("You failed the exam.");
  }
}
```

Output:

```
You failed the exam.
```

3. else-if Ladder

The **else-if ladder** is used when you want to test **multiple conditions**.

Syntax:

```
if (condition1) {
  // block 1
} else if (condition2) {
  // block 2
} else if (condition3) {
  // block 3
} else {
  // block 4 (if none of the above are true)
}
```

Example:

```
void main() {
  int marks = 85;

  if (marks >= 90) {
    print("Grade: A+");
  } else if (marks >= 75) {
    print("Grade: A");
  } else if (marks >= 60) {
    print("Grade: B");
  } else {
    print("Grade: C");
  }
}
```

Output:

```
Grade: A
```

Explanation:
Dart checks each condition from top to bottom.
As soon as one condition is true, the rest are skipped.

4. Nested if Statement

A **nested if** means placing one `if` statement inside another.
It's used when you need to test a condition only if another condition is already true.

Syntax:

```
if (condition1) {
  if (condition2) {
    // code runs only if both conditions are true
  }
}
```

Example:

```
void main() {
  int age = 25;
  bool hasVoterID = true;

  if (age >= 18) {
    if (hasVoterID) {
      print("You can vote.");
    } else {
      print("You need a voter ID to vote.");
    }
  } else {
    print("You are not eligible to vote.");
  }
}
```

Output:

```
You can vote.
```

5. switch Statement

The **switch statement** is used when you have many possible values for a single variable. It's a cleaner alternative to multiple `if-else` conditions.

Syntax:

```
switch (expression) {
  case value1:
    // code block
    break;

  case value2:
    // code block
    break;

  default:
    // default code block
}
```

Example:

```
void main() {
  String day = "Monday";

  switch (day) {
    case "Monday":
      print("Start of the week!");
      break;
    case "Friday":
      print("Weekend is near!");
      break;

    case "Sunday":
      print("It's holiday!");
      break;

    default:
      print("Mid-week day!");
  }
}
```

Output:

```
Start of the week!
```

Explanation:
The `switch` compares the value of `day` with each case.
When a match is found, that block executes.
The `break` keyword stops the execution from falling into the next case.
The `default` block runs if no case matches.

Loops in Dart

Loops are used to execute a block of code **multiple times** until a certain condition is met. They help avoid writing repetitive code manually.

Dart supports the following types of loops:

1. `for` loop
2. `while` loop
3. `do-while` loop
4. `for-in` loop
5. `forEach` loop

Also, we'll learn about **break** and **continue** statements.

1. for Loop

The `for` loop is used when you know **exactly how many times** you want to repeat a block of code.

Syntax:

```
for (initialization; condition; increment/decrement) {
  // code to execute
}
```

Example:

```
void main() {
  for (int i = 1; i <= 5; i++) {
    print("Number: $i");
  }
}
```

Output:

```
Number: 1
Number: 2
Number: 3
Number: 4
Number: 5
```

Explanation:

- `int i = 1` → initialization (loop starts at 1)
- `i <= 5` → condition (loop runs until 5)
- `i++` → increment (increase by 1 after each iteration)

2. while Loop

The `while` loop is used when the number of iterations is **not known** in advance — it continues as long as the condition is true.

Syntax:

```
while (condition) {
  // code block
}
```

Example:

```
void main() {
  int i = 1;

  while (i <= 5) {
    print("Count: $i");
    i++;
  }
}
```

Output:

```
Count: 1
Count: 2
Count: 3
Count: 4
Count: 5
```

Explanation:
The loop checks the condition **before** each iteration.

3. do-while Loop

The `do-while` loop executes the code **at least once**, even if the condition is false, because the condition is checked **after** running the loop body.

Syntax:

```
do {
  // code block
} while (condition);
```

Example:

```
void main() {
  int i = 1;

  do {
    print("Hello $i");
    i++;
  } while (i <= 3);
}
```

Output:

```
Hello 1
Hello 2
Hello 3
```

4. for-in Loop

Used to iterate through elements of a **collection** like a List or Set.

Syntax:

```
for (var item in collection) {
  // code block
}
```

Example:

```
void main() {
  var fruits = ["Apple", "Banana", "Mango"];

  for (var fruit in fruits) {
    print(fruit);
  }
}
```

Output:

```
Apple
Banana
Mango
```

5. forEach Loop

A modern way to iterate through a **List or Map** in Dart.

Syntax (List):

```
listName.forEach((item) {
  // code block
});
```

Example (List):

```
void main() {
  var numbers = [2, 4, 6, 8];

  numbers.forEach((num) {
    print(num * num);
  });
}
```

Output:

```
4
16
36
64
```

Example (Map):

```
void main() {
  var person = {"name": "Neeraj", "age": 23, "city": "Lucknow"};

  person.forEach((key, value) {
    print("$key: $value");
  });
}
```

Output:

```
name: Neeraj
age: 23
city: Lucknow
```

Control Statements in Loops

break Statement

The `break` statement is used to **exit** a loop immediately.

Example:

```
void main() {
  for (int i = 1; i <= 10; i++) {
    if (i == 6) {
      break;
    }
    print(i);
  }
}
```

Output:

```
1
2
3
4
5
```

Explanation:
The loop stops completely when `i == 6`.

continue Statement

The `continue` statement **skips the current iteration** and jumps to the next one.

Example:

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

Output:

```
1
2
4
5
```

Explanation:
When `i == 3`, the loop skips that iteration but continues for the remaining numbers.

Quick Summary Table

Loop Type	When to Use	Condition Checked
`for`	Known number of iterations	Before loop starts
`while`	Unknown number of iterations	Before each iteration
`do-while`	At least one run needed	After loop body
`for-in`	Iterate through collections	Automatically handled
`forEach`	Clean iteration on lists/maps	Automatically handled
`break`	Exit loop early	Stops loop
`continue`	Skip current iteration	Moves to next