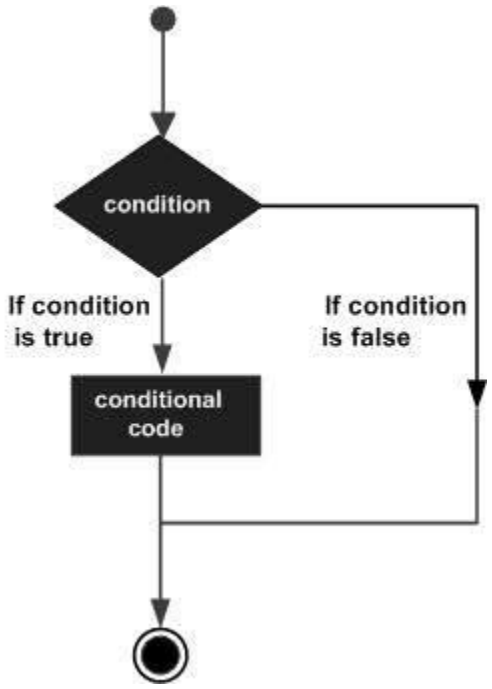
When should we use conditional statements?

While writing a program, there may be a situation when you need to adopt one out of a given set of paths. In such cases, you need to use conditional statements that allow your program to make correct decisions and perform right actions.

JavaScript supports *conditional statements which are used to perform different actions based on different conditions*. Here we will explain the **if..else** statement.

Flow Chart of if-else

The following flow chart shows how the if-else statement works.



JavaScript supports the following forms of **if..else** statement −

- if statement
  - The **if** statement is the fundamental control statement that allows JavaScript to make decisions and execute statements conditionally.

Syntax:

```
if (expression)
{
Statement(s) to be executed if expression is true
}
```

Example:

```
<html>
  <body>

    <script type="text/javascript">
      <!--
        var age = 20;

        if( age > 18 ){
          document.write("<b>Qualifies for driving</b>");
        }
      //-->
    </script>

    <p>Set the variable to different value and then try...</p>
  </body>
</html>
```

---

if...else statement

The **'if...else'** statement is the next form of control statement that allows JavaScript to execute statements in a more controlled way.

Syntax:

```
if (expression){
  Statement(s) to be executed if expression is true
}

else{
  Statement(s) to be executed if expression is false
}
```

Example:

```
<html>
  <body>

    <script type="text/javascript">
```

```
      <!--
        var age = 15;

        if( age > 18 ){
          document.write("<b>Qualifies for driving</b>");
        }

        else{
          document.write("<b>Does not qualify for driving</b>");
        }
      //-->
    </script>

    <p>Set the variable to different value and then try...</p>
  </body>
</html>
```

- **if...else if... statement.**
    - The **if...else if...** statement is an advanced form of **if…else** that allows JavaScript to make a correct decision out of several conditions. if none of the conditions is true, then the **else** block is executed.

Syntax:

```
if (expression 1){
  Statement(s) to be executed if expression 1 is true
}

else if (expression 2){
  Statement(s) to be executed if expression 2 is true
}

else if (expression 3){
  Statement(s) to be executed if expression 3 is true
}

else{
  Statement(s) to be executed if no expression is true
}
```

Example:

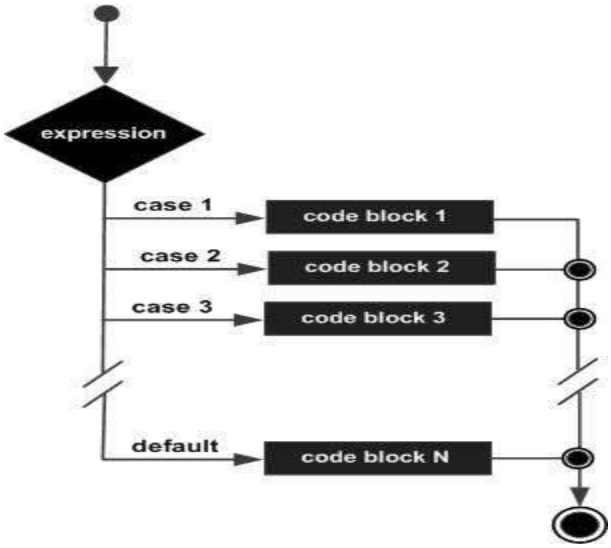
```
<html>
  <body>

    <script type="text/javascript">
      <!--
        var book = "maths";
        if( book == "history" ){
          document.write("<b>History Book</b>");
        }

        else if( book == "maths" ){
          document.write("<b>Maths Book</b>");
        }

        else if( book == "economics" ){
          document.write("<b>Economics Book</b>");
        }

        else{
          document.write("<b>Unknown Book</b>");
        }
      //-->
    </script>

    <p>Set the variable to different value and then try...</p>
  </body>
<html>
```

What is Switch Case and why should we use it?

if...else…if statements are not always the best solution, especially when all of the branches depend on the value of a single variable. The interpreter checks each **case** against the value of the expression until a match is found. If nothing matches, a **default** condition will be used. The **break** statements indicate the end of a particular case. If they were omitted, the interpreter would continue executing each statement in each of the following cases.

Flow Chart

The following flow chart explains a switch-case statement works.



Syntax:

```
switch (expression)
{
  case condition 1: statement(s)
  break;

  case condition 2: statement(s)
  break;
  ...

  case condition n: statement(s)
  break;

  default: statement(s)
}
```

Example:

```
<html>
  <body>
```

```
    <script type="text/javascript">
      <!--
        var grade='A';
        document.write("Entering switch block<br />");
        switch (grade)
        {
          case 'A': document.write("Good job<br />");
          break;

          case 'B': document.write("Pretty good<br />");
          break;

          case 'C': document.write("Passed<br />");
          break;

          case 'D': document.write("Not so good<br />");
          break;

          case 'F': document.write("Failed<br />");
          break;

          default:  document.write("Unknown grade<br />")
        }
        document.write("Exiting switch block");
      //-->
    </script>

    <p>Set the variable to different value and then try...</p>
  </body>
</html>
```

Loops in JavaScripts

When should we use Loops?

You may encounter a situation where you need to perform an action over and over again. In such situations, you would need to write loop statements to reduce the number of lines.

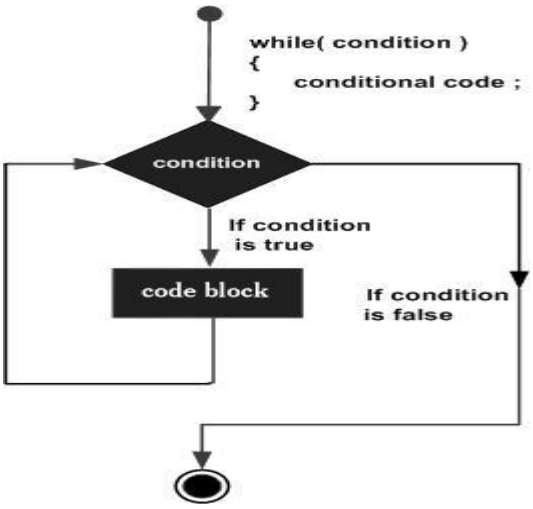
What are the types of Loops??

- **While Loop:**

  The purpose of a **while** loop is to execute a statement or code block repeatedly as long as an **expression** is true. Once the expression becomes **false,** the loop terminates.

  Flow Chart

  The flow chart of **while loop** looks as follows –



Syntax:

```
while (expression)
{
        Statement(s) to be executed if expression is true
}
```

Example:

```
    <html>
<body>

  <script type="text/javascript">
    <!--
     var count = 0;
```

```
            document.write("Starting Loop ");

            while (count < 10){
              document.write("Current Count : " + count + "<br />");
              count++;
            }

            document.write("Loop stopped!");
          //-->
        </script>

        <p>Set the variable to different value and then try...</p>
      </body>
</html>
```

Out put:

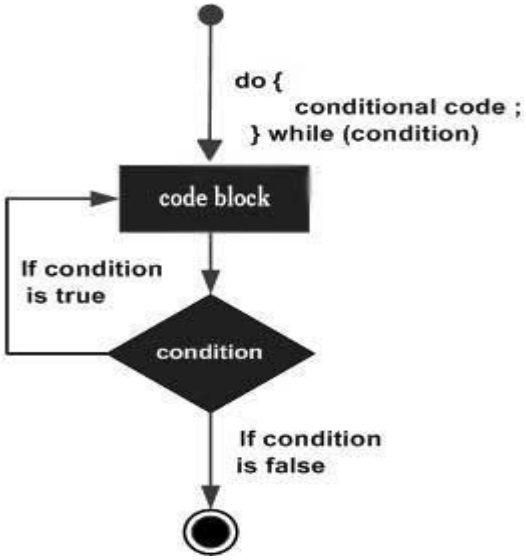
```
Starting Loop
Current Count : 0
Current Count : 1
Current Count : 2
Current Count : 3
Current Count : 4
Current Count : 5
Current Count : 6
Current Count : 7
Current Count : 8
Current Count : 9
Loop stopped!
Set the variable to different value and then try...
```

- **do while Loop:**

  The **do...while** loop is similar to the **while** loop except that the condition check happens at the end of the loop. This means that the loop will always be executed at least once, even if the condition is **false**.

Flow Chart

The flow chart of a **do-while** loop would be as follows –



Syntax

The syntax for **do-while** loop in JavaScript is as follows −

```
do{

   Statement(s) to be executed;

} while (expression)
```

Example:

```
<html>
    <body>

        <script type="text/javascript">
          <!--
            var count = 0;

            document.write("Starting Loop" + "<br />");
            do{
```

```
            document.write("Current Count : " + count + "<br />");
            count++;
          }

          while (count < 5);
          document.write ("Loop stopped!");
        //-->
      </script>

      <p>Set the variable to different value and then try...</p>
    </body>
</html>
```

Output:

```
Starting Loop
Current Count : 0
Current Count : 1
Current Count : 2
Current Count : 3
Current Count : 4
Loop Stopped!
Set the variable to different value and then try...
```

- **For Loop:**

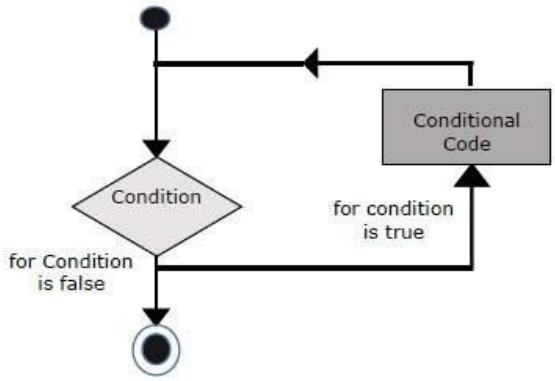
The '**for**' loop is the most compact form of looping. It includes the following three important parts −

- The **loop initialization** where we initialize our counter to a starting value. The initialization statement is executed before the loop begins.
- The **test statement** which will test if a given condition is true or not. If the condition is true, then the code given inside the loop will be executed, otherwise the control will come out of the loop.
- The **iteration statement** where you can increase or decrease your counter.

You can put all the three parts in a single line separated by semicolons.

Flow Chart

The flow chart of a for loop in JavaScript would be as follows −



Syntax

```
for (initialization; test condition; iteration statement){
   Statement(s) to be executed if test condition is true
}
```

Example

```
<html>
  <body>

    <script type="text/javascript">
      <!--
        var count;
        document.write("Starting Loop" + "<br />");

        for(count = 0; count < 10; count++){
          document.write("Current Count : " + count );
          document.write("<br />");
        }
```

```
            document.write("Loop stopped!");
          //-->
        </script>

        <p>Set the variable to different value and then try...</p>
      </body>
</html>
```

Output

Starting Loop

Current Count : 0

Current Count : 1

Current Count : 2

Current Count : 3

Current Count : 4

Current Count : 5

Current Count : 6

Current Count : 7

Current Count : 8

Current Count : 9

Loop stopped!

Set the variable to different value and then try...

- How to have control over the loops?

  There may be a situation when you need to come out of a loop without reaching its bottom. There may also be a situation when you want to skip a part of your code block and start the next iteration of the loop.

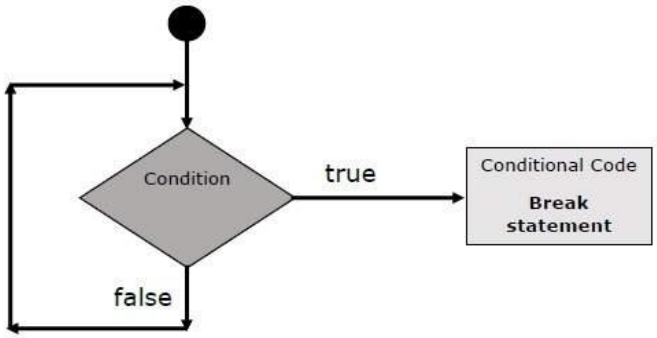
  To handle all such situations, JavaScript provides **break** and **continue** statements.

The break Statement

The **break** statement, which was briefly introduced with the *switch* statement, is used to exit a loop early, breaking out of the enclosing curly braces.

Flow Chart

The flow chart of a break statement would look as follows −



Example

The following example illustrates the use of a break statement with a while loop. Notice how the loop breaks out early once x reaches 5 and reaches to document.write (..) statement just below to the closing curly brace −

```
<html>
  <body>

    <script type="text/javascript">
      <!--
      var x = 1;
      document.write("Entering the loop<br /> ");

      while (x < 20)
      {
        if (x == 5){
          break; // breaks out of loop completely
        }
        x = x + 1;
        document.write( x + "<br />");
      }
```

```
      document.write("Exiting the loop!<br /> ");
      //-->
    </script>

    <p>Set the variable to different value and then try...</p>
  </body>
</html>
```

Output

Entering the loop

2

3

4

5

Exiting the loop!

Set the variable to different value and then try...

The continue Statement

The **continue** statement tells the interpreter to immediately start the next iteration of the loop and skip the remaining code block.

Example

This example illustrates the use of a continue statement with a while loop. Notice how the continue statement is used to skip printing when the index held in variable x reaches 5 −

```
<html>
  <body>
```

```
    <script type="text/javascript">
      <!--
         var x = 1;
         document.write("Entering the loop<br /> ");

         while (x < 10)
         {
           x = x + 1;

           if (x == 5){
             continue; // skip rest of the loop body
           }
           document.write( x + "<br />");
         }

         document.write("Exiting the loop!<br /> ");
      //-->
    </script>

    <p>Set the variable to different value and then try...</p>
  </body>
</html>
```

Output:

```
Entering the loop
2
3
4
6
7
8
9
10
Exiting the loop!
```

Using Labels to Control the Flow:

Starting from JavaScript 1.2, a label can be used with **break** and **continue** to control the flow more precisely.

A **label** is simply an identifier followed by a colon (:) that is applied to a statement or a block of code. We will see two different examples to understand how to use labels with break and continue.

Note − Line breaks are not allowed between the **'continue'** or **'break'** statement and its label name. Also, there should not be any other statement in between a label name and associated loop.

Example:

```
<html>
  <body>

    <script type="text/javascript">
      <!--
        document.write("Entering the loop!<br /> ");
        outerloop: // This is the label name

        for (var i = 0; i < 5; i++)
        {
          document.write("Outerloop: " + i + "<br />");
          innerloop:
          for (var j = 0; j < 5; j++)
          {
            if (j > 3 ) break ; // Quit the innermost loop
            if (i == 2) break innerloop; // Do the same thing
            if (i == 4) break outerloop; // Quit the outer loop
            document.write("Innerloop: " + j + " <br />");
          }
        }

        document.write("Exiting the loop!<br /> ");
      //-->
    </script>

  </body>
```

```
</html>
```

Output

```
Entering the loop!
Outerloop: 0
Innerloop: 0
Innerloop: 1
Innerloop: 2
Innerloop: 3
Outerloop: 1
Innerloop: 0
Innerloop: 1
Innerloop: 2
Innerloop: 3
Outerloop: 2
Outerloop: 3
Innerloop: 0
Innerloop: 1
Innerloop: 2
Innerloop: 3
Outerloop: 4
Exiting the loop!
```

Example 2

```
<html>
  <body>

    <script type="text/javascript">
      <!--
      document.write("Entering the loop!<br /> ");
      outerloop: // This is the label name

      for (var i = 0; i < 3; i++)
      {
        document.write("Outerloop: " + i + "<br />");
        for (var j = 0; j < 5; j++)
        {
          if (j == 3){
            continue outerloop;
          }
```

```
            document.write("Innerloop: " + j + "<br />");
          }
        }

        document.write("Exiting the loop!<br /> ");
        //-->
      </script>

   </body>
</html>
```

Output

```
Entering the loop!
Outerloop: 0
Innerloop: 0
Innerloop: 1
Innerloop: 2
Outerloop: 1
Innerloop: 0
Innerloop: 1
Innerloop: 2
Outerloop: 2
Innerloop: 0
Innerloop: 1
Innerloop: 2
Exiting the loop!
```