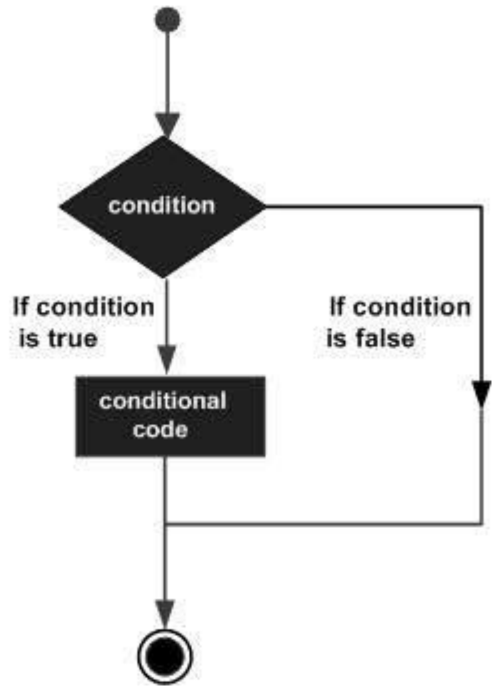
JavaScript - if...else Statement

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
- if...else statement
- if...else if... statement.

if statement

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

Syntax

The syntax for a basic if statement is as follows-

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

Here a JavaScript expression is evaluated. If the resulting value is true, the given statement(s) are executed. If the expression is false, then no statement would be not executed. Most of the times, you will use comparison operators while making decisions.

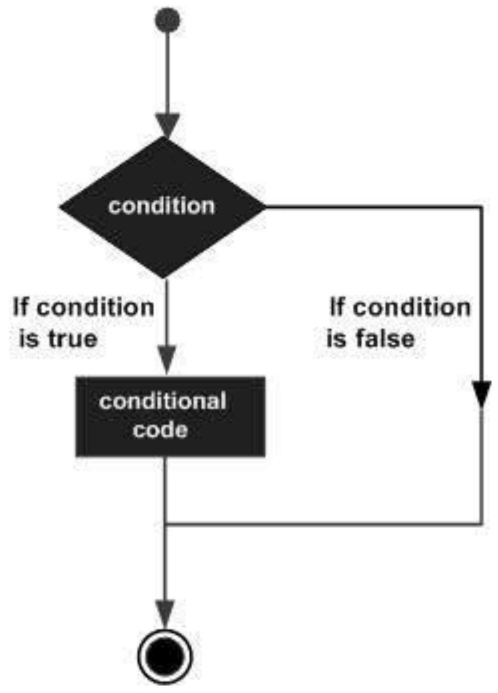
JavaScript - if...else Statement

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
- if...else statement
- if...else if... statement.

if statement

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

Syntax

The syntax for a basic if statement is as follows-

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

Here a JavaScript expression is evaluated. If the resulting value is true, the given statement(s) are executed. If the expression is false, then no statement would be not executed. Most of the times, you will use comparison operators while making decisions.

• 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.

Syntax

The syntax of an if-else-if statement is as follows −

```
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
}
```

There is nothing special about this code. It is just a series of **if** statements, where each **if** is a part of the **else** clause of the previous statement. Statement(s) are executed based on the true condition, if none of the conditions is true, then the **else** block is executed.

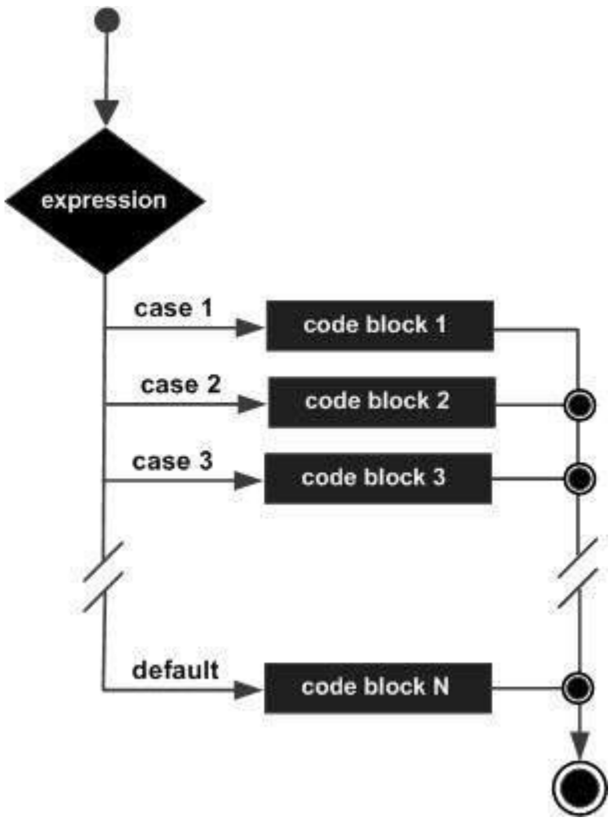
JavaScript - Switch Case

You can use multiple **if...else...if** statements, as in the previous chapter, to perform a multiway branch. However, this is not always the best solution, especially when all of the branches depend on the value of a single variable.

Starting with JavaScript 1.2, you can use a **switch** statement which handles exactly this situation, and it does so more efficiently than repeated **if...else if**statements.

Flow Chart

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



Syntax

The objective of a **switch** statement is to give an expression to evaluate and several different statements to execute based on the value of the expression. 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.

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

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

   case condition n: statement(s)
   break;

   default: statement(s)
}
```

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.

JavaScript - While Loops

While writing a program, 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.

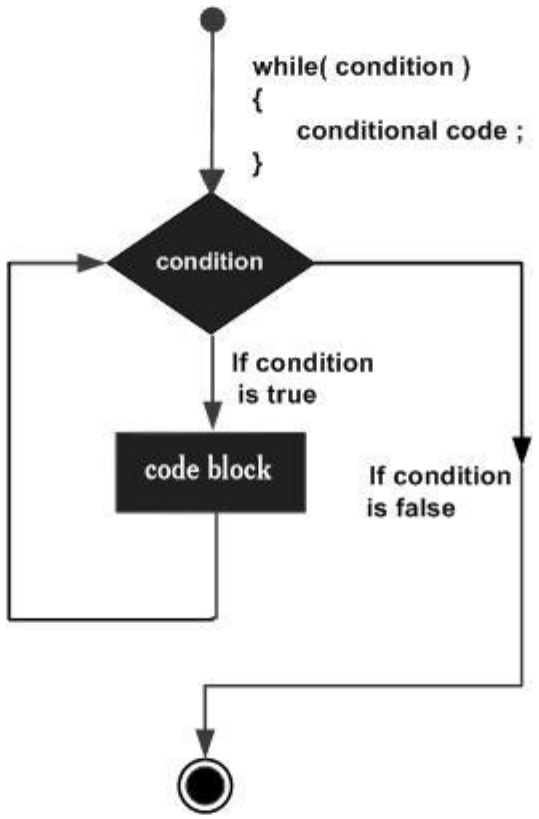
JavaScript supports all the necessary loops to ease down the pressure of programming.

The while Loop

The most basic loop in JavaScript is the **while** loop which would be discussed in this chapter. 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 −



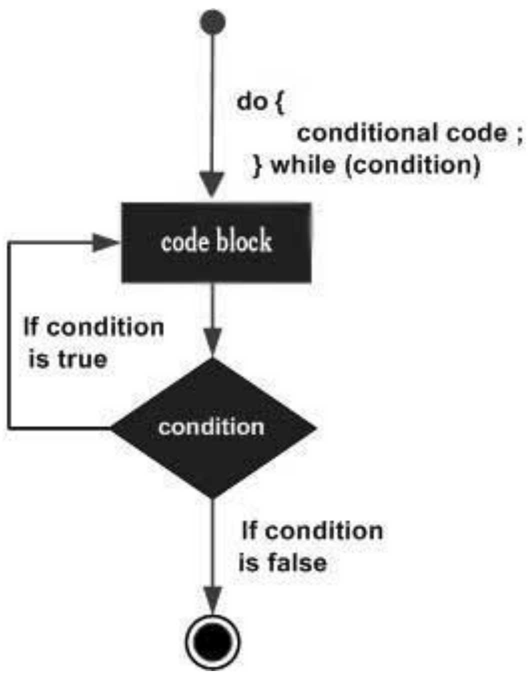
Syntax

The syntax of **while loop** in JavaScript is as follows −

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

The 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);
```

JavaScript - For Loop

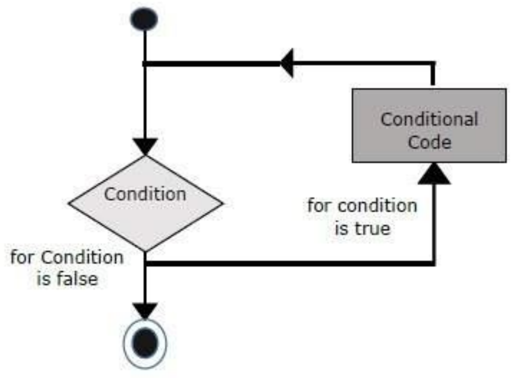
The '**for**' loop is the most compact form of looping. It includes the following three important parts –

- 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

The syntax of **for** loop is JavaScript is as follows –

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

JavaScript *for…in* loop

The **for...in** loop is used to loop through an object's properties. As we have not discussed Objects yet, you may not feel comfortable with this loop. But once you understand how objects behave in JavaScript, you will find this loop very useful.

Syntax

The syntax of 'for..in' loop is −

```
for (variablename in object) {
   statement or block to execute
}
```

JavaScript - Loop Control

JavaScript provides full control to handle loops and switch statements. 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.

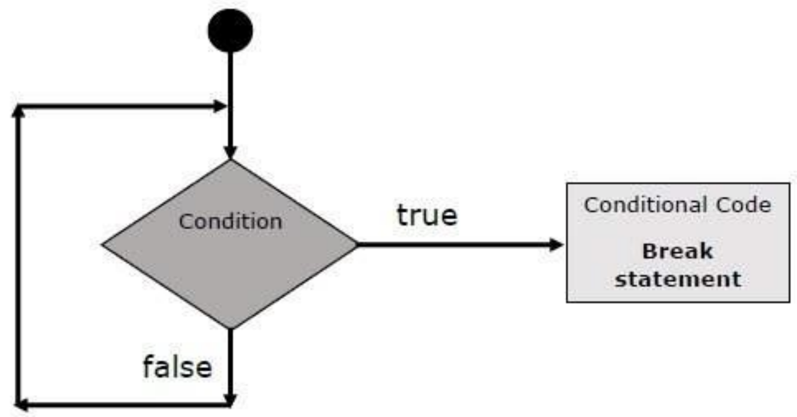
To handle all such situations, JavaScript provides **break** and **continue**statements. These statements are used to immediately come out of any loop or to start the next iteration of any loop respectively.

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 −



The continue Statement

The **continue** statement tells the interpreter to immediately start the next iteration of the loop and skip the remaining code block. When a **continue**statement is encountered, the program flow moves to the loop check expression immediately and if the condition remains true, then it starts the next iteration, otherwise the control comes out of the loop.

JavaScript - Errors & Exceptions Handling

There are three types of errors in programming: (a) Syntax Errors, (b) Runtime Errors, and (c) Logical Errors.

Syntax Errors

Syntax errors, also called **parsing errors,** occur at compile time in traditional programming languages and at interpret time in JavaScript.

For example, the following line causes a syntax error because it is missing a closing parenthesis.

```
<script type = "text/javascript">
   <!--
      window.print(;
   //-->
</script>
```

When a syntax error occurs in JavaScript, only the code contained within the same thread as the syntax error is affected and the rest of the code in other threads gets executed assuming nothing in them depends on the code containing the error.

Runtime Errors

Runtime errors, also called **exceptions,** occur during execution (after compilation/interpretation).

For example, the following line causes a runtime error because here the syntax is correct, but at runtime, it is trying to call a method that does not exist.

```
<script type = "text/javascript">
   <!--
      window.printme();
   //-->
</script>
```

Exceptions also affect the thread in which they occur, allowing other JavaScript threads to continue normal execution.

Logical Errors

Logic errors can be the most difficult type of errors to track down. These errors are not the result of a syntax or runtime error. Instead, they occur when you make a mistake in the logic that drives your script and you do not get the result you expected.

You cannot catch those errors, because it depends on your business requirement what type of logic you want to put in your program.

The try...catch...finally Statement

The latest versions of JavaScript added exception handling capabilities. JavaScript implements the **try...catch...finally** construct as well as the **throw** operator to handle exceptions.

You can **catch** programmer-generated and **runtime** exceptions, but you cannot **catch** JavaScript syntax errors.

The throw Statement

You can use **throw** statement to raise your built-in exceptions or your customized exceptions. Later these exceptions can be captured and you can take an appropriate action.