Chapter 1
Basics of JavaScript

CO(COURSE OUTCOME)

CO1 Build interactive web pages using program flow control structure.

CO2 Implement Arrays , functions and create event based web forms using Java Script.

CO3 Use JavaScript for browser data persistence.

CO4 Create menus, navigation in interactive webpages using regular expressions for

JavaScript

- JavaScript is an object-based scripting language which is lightweight and cross-platform.
- JavaScript is not a compiled language, but it is a translated language.
- The JavaScript Translator (embedded in the browser) is responsible for translating the JavaScript code for the web browser.
- JavaScript (js) is a light-weight object-oriented programming language which is used by several websites for scripting the webpages. It is an interpreted, full-fledged programming language that enables dynamic interactivity on websites when applied to an HTML document.

Features of Javascript

Scripting
Javascript executes the client-side script in the browser.

Interpreter
The browser interprets JavaScript code.

Event Handling
Events are actions. Javascript provides event-handling options.

Light Weight
As Javascript is not a compiled language, source code never changes to byte code before running time. Low-end devices can also run Javascript because of its lightweight feature.

Case Sensitive
In Javascript, names, variables, keywords, and functions are case-sensitive.

Control Statements
Javascript has control statements like if-else-if, switch case, and loop. Users can write complex code using these control statements.

Objects as first-class Citizens
Javascript arrays, functions, and symbols are objects which can inherit the Object prototype properties. Objects being first-class citizens means Objects can do all tasks.

Supports Functional Programming
Javascript functions can be an argument to another function, can call by reference, and can assign to a variable.

JavaScript Advantages

Following are the advantages of JavaScript –

•**Simple** − JavaScript is simple to comprehend and pick up. Both users and developers will find the structure to be straightforward. Additionally, it is very doable to implement, saving web developers a tonne of money when creating dynamic content.

•**Speed** − JavaScript is a "interpreted" language, it cuts down on the time needed for compilation in other programming languages like Java. Another client-side script is JavaScript, which accelerates programme execution by eliminating the wait time for server connections.
•No matter where JavaScript is hosted, it is always run in a client environment to reduce bandwidth usage and speed up execution.

•**Interoperability** − Because JavaScript seamlessly integrates with other programming languages, many developers favour using it to create a variety of applications. Any webpage or the script of another programming language can contain it.

•**Server Load** − Data validation can be done within the browser itself rather than being forwarded to the server because JavaScript is client-side. The entire website does not need to be reloaded in the event of any discrepancy. Only the chosen area of the page is updated by the browser.

JavaScript Disadvantages:

Following are the disadvantages of JavaScript –

•**Cannot Debug** – Although some HTML editors allow for debugging, they are not as effective as editors for C or C++. Additionally, the developer has a difficult time figuring out the issue because the browser doesn't display any errors.

•**Unexpected stop of rendering** – The website's entire JavaScript code can stop rendering due to a single error in the code. It appears to the user as though JavaScript is absent. The browsers, however, are very forgiving of these mistakes.

•**Client-side Security** – The user can see the JavaScript code; it could be misused by others. These actions might involve using the source code anonymously. Additionally, it is very simple to insert code into the website that impair the security of data transmitted via the website.

•**Inheritance** – JavaScript does not support multiple inheritance; only one inheritance is supported. This property of object-oriented languages might be necessary for some programmes.

•**Browser Support** – Depending on the browser, JavaScript is interpreted differently. Therefore, before publication, the code needs to run on various platforms. We also need to check the older browsers because some new functions are not supported by them.

JavaScript Objects

Real Life Objects, Properties, and Methods

Object	Properties	Methods

	car.name = Fiat
	car.model = 500
	car.weight = 850kg
	car.color = white
		car.start()
		car.drive()
		car.brake()
		car.stop()

In real life, a car is an **object**.

- A car has **properties** like weight and color, and **methods** like start and stop
- All cars have the same **properties**, but the property **values** differ from car to car.
- All cars have the same **methods**, but the methods are performed **at different times**.

JavaScript Variables

Variables are containers for storing data (storing data values)

4 Ways to Declare a JavaScript Variable:

- Using var
- **Using let**
- Using const
- Using nothing.

Examples:

```
In this example, x, y, and z, are variables, declared with the var keyword:
var x = 5;
var y = 6;
var z = x + y;

In this example, x, y, and z, are variables, declared with the let keyword:
let x = 5;
let y = 6;
let z = x + y;
```

In this example, x, y, and z, are undeclared variables:

```
x = 5;
y = 6;
z = x + y;
```

When to Use JavaScript var?

- Always declare JavaScript variables with var, let, or const.
- The var keyword is used in all JavaScript code from 1995 to 2015.
- **The let and const keywords were added to JavaScript in 2015.**
- If you want your code to run in older browsers, you must use var.

When to Use JavaScript const?

- If you want a general rule: always declare variables with const.
- If you think the value of the variable can change, use let.

```
const price1 = 5;
const price2 = 6;
let total = price1 + price2;
```

- The two variables price1 and price2 are declared with the const keyword.
- These are constant values and cannot be changed.
- The variable total is declared with the let keyword.
- This is a value that can be changed.

JavaScript Identifiers:

- All JavaScript **variables** must be **identified** with **unique names**.
- These unique names are called **identifiers**.
- Identifiers can be short names (like x and y) or more descriptive names (age, sum, totalVolume).

The general rules for constructing names for variables (unique identifiers) are:

- **Names can contain letters, digits, underscores, and dollar signs.**
- Names must begin with a letter.
- Names can also begin with $ and _.
- Names are case sensitive (y and Y are different variables).
- Reserved words (like JavaScript keywords) cannot be used as names.

Note: JavaScript identifiers are case-sensitive.

The Assignment Operator

- In JavaScript, the equal sign (=) is an "assignment" operator, not an "equal to" operator.
- This is different from algebra. The following does not make sense in algebra:

x = x + 5:

In JavaScript, however, it makes perfect sense: **it assigns the value of x + 5 to x.**

Note: The "equal to" operator is written like == in JavaScript.

JavaScript Data Types

- **Strings are written inside double or single quotes. Numbers are written without quotes.**
- **If you put a number in quotes, it will be treated as a text string.**

Example:

```
const pi = 3.14;
let person = "John Doe";
let answer = 'Yes I am!';
```

Declaring a JavaScript Variable:

You declare a JavaScript variable with the var or the let keyword:

```
var carName;     or        let carName;
```

After the declaration, the variable has no value (technically it is undefined).

To assign a value to the variable, use the equal sign:

carName = "Volvo";

You can also assign a value to the variable when you declare it:

```
let carName = "Volvo";
```

Note:

It's a good programming practice to declare all variables at the beginning of a script.

One Statement, Many Variables

You can declare many variables in one statement.

Start the statement with let and separate the variables by comma:

```
let person = "John Doe", carName = "Volvo", price = 200;
```

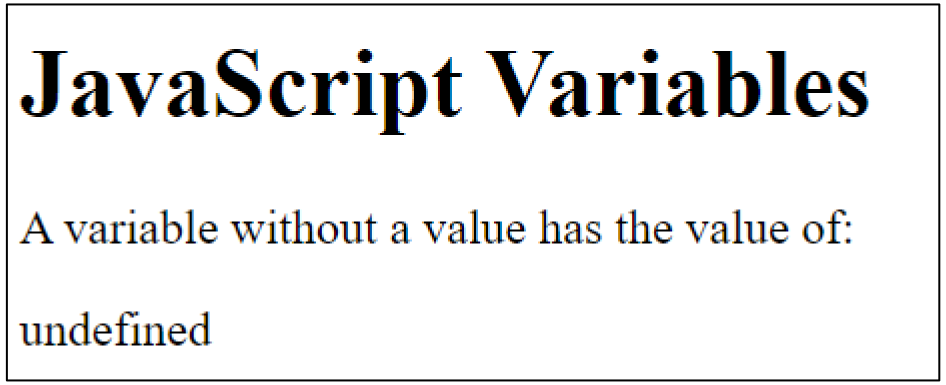
Value = undefine

- **A variable declared without a value will have the value undefined.**

- The variable carName will have the value undefined after the execution of this statement:

```
let carName;
```

```
<html>
<body>
<h1>JavaScript Variables</h1>
<p>A variable without a value has the value of:</p>
<p id="demo"></p>

<script>
let carName;
document.getElementById("demo").innerHTML = carName;
</script>

</body>
</html>
```



JavaScript Variables

A variable without a value has the value of:

undefined

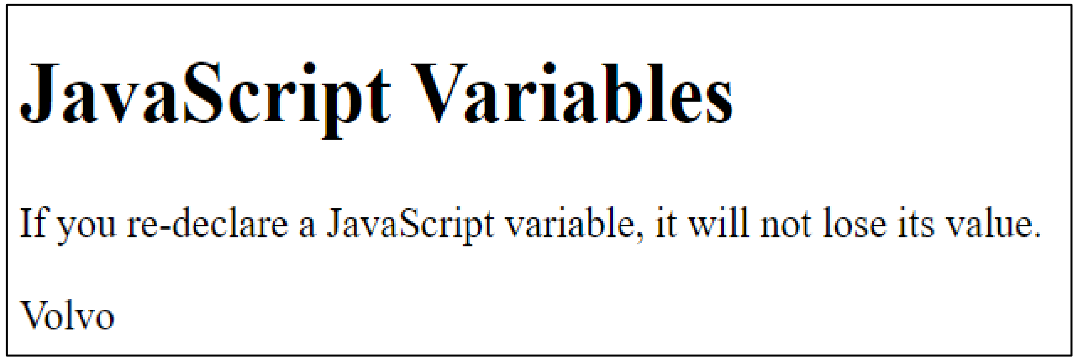
Note: The innerHTML property sets or returns the HTML content (inner HTML) of an element.

Re-Declaring JavaScript Variables:

If you re-declare a JavaScript variable declared with var, it will not lose its value.

The variable carName will still have the value "Volvo" after the execution of these statements:

```
<html>
<body>

<h1>JavaScript Variables</h1>

<p>If you re-declare a JavaScript variable, it will not lose its value.</p>

<p id="demo"></p>

<script>
var carName = "Volvo";
var carName;
document.getElementById("demo").innerHTML = carName;
</script>

</body>
</html>
```



```
JavaScript Variables

If you re-declare a JavaScript variable, it will not lose its value.

Volvo
```

Difference between var and let

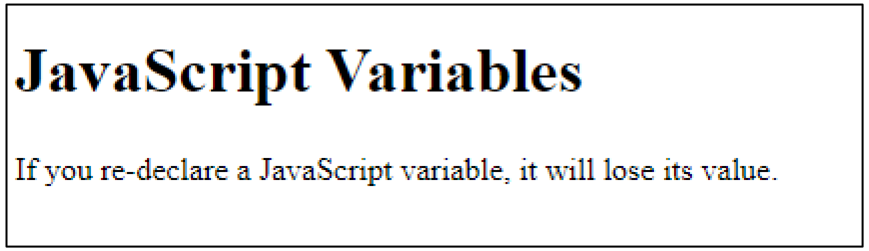
Note

You cannot re-declare a variable declared with let or const.

This will not work:

```
let carName = "Volvo";
let carName;
```

```
<h1>JavaScript Variables</h1>

<p>If you re-declare a JavaScript variable, it will lose its value.</p>

<script>
let carName = "Volvo";
let carName;
document.getElementById("demo").innerHTML = carName;
</script>
```

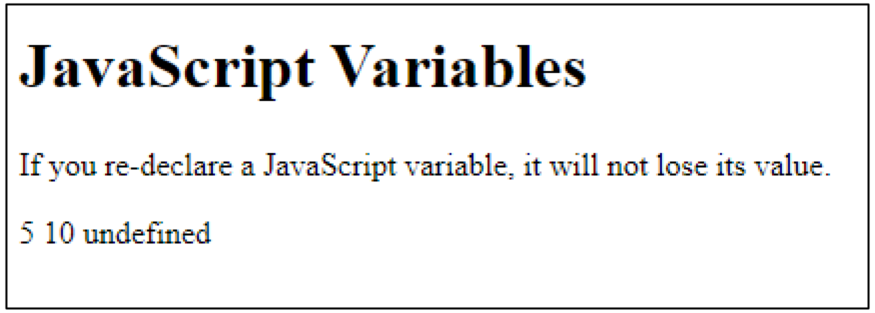


JavaScript Variables

If you re-declare a JavaScript variable, it will lose its value.

Difference between var and let

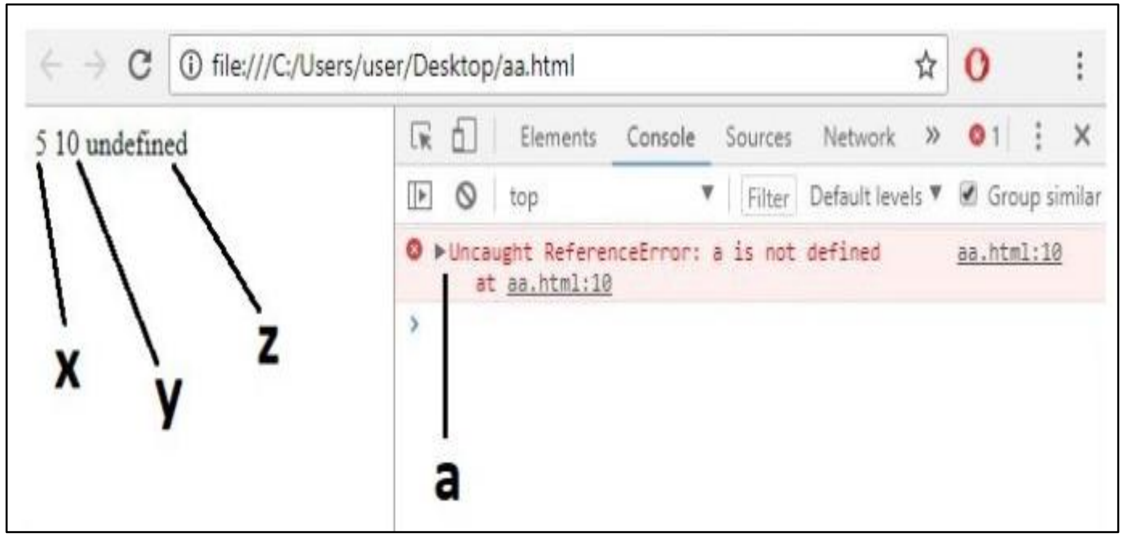
```
<script>
// calling x after definition
    var x = 5;
    document.write(x, "\n");

    // calling y after definition
    let y = 10;
    document.write(y, "\n");

    // calling var z before definition will return undefined
    document.write(z, "\n");
    var z = 2;

    // calling let a before definition will give error
    document.write(a);
    let a = 3;
</script>
```



JavaScript Variables

If you re-declare a JavaScript variable, it will not lose its value.

```
5 10 undefined
```



file:///C:/Users/user/Desktop/aa.html

```
5 10 undefined
```

```
Uncaught ReferenceError: a is not defined    aa.html:10
    at aa.html:10
```

JavaScript Arithmetic:

As with algebra, you can do arithmetic with JavaScript variables, using operators like = and +

```
let x = 5 + 2 + 3;.........................10
```

You can also add strings, but strings will be concatenated:

```
let x = "John" + " " + "Doe"; ................................................... John Doe
```

```
let x = "5" + 2 + 3; ............................................. 523
```

Note
If you put a number in quotes, the rest of the numbers will be treated as strings, and concatenated.

Types of JavaScript Operators

There are different types of JavaScript operators:

- Arithmetic Operators
- Assignment Operators
- Comparison Operators
- Logical Operators
- Conditional Operators
- Type Operators

JavaScript Arithmetic Operators:

Arithmetic Operators are used to perform arithmetic on numbers:

Arithmetic Operators Example

```
let a = 3;
let x = (100 + 50) * a;
```

OUTPUT:

```
450
```

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
**	Exponentiation
/	Division
%	Modulus (Division Remainder)
++	Increment
--	Decrement

JavaScript Assignment Operators

Assignment operators assign values to JavaScript variables.

The Addition Assignment Operator (+=) adds a value to a variable.

```
let x = 10;
x += 5;
```

OUTPUT

```
15
```

Operator	Example	Same As
=	x = y	x = y
+=	x += y	x = x + y
-=	x -= y	x = x - y
*=	x *= y	x = x * y
/=	x /= y	x = x / y
%=	x %= y	x = x % y
**=	x **= y	x = x ** y

Adding JavaScript Strings

The + operator can also be used to add (concatenate) strings.

```
let text1 = "John";
let text2 = "Doe";
let text3 = text1 + " " + text2;
```

OUTPUT:

```
John Doe
```

The += assignment operator can also be used to add (concatenate) strings:

```
let text1 = "What a very ";
text1 += "nice day";
```

OUTPUT:

```
What a very nice day
```

Note: When used on strings, the + operator is called the concatenation operator.

Adding Strings and Numbers

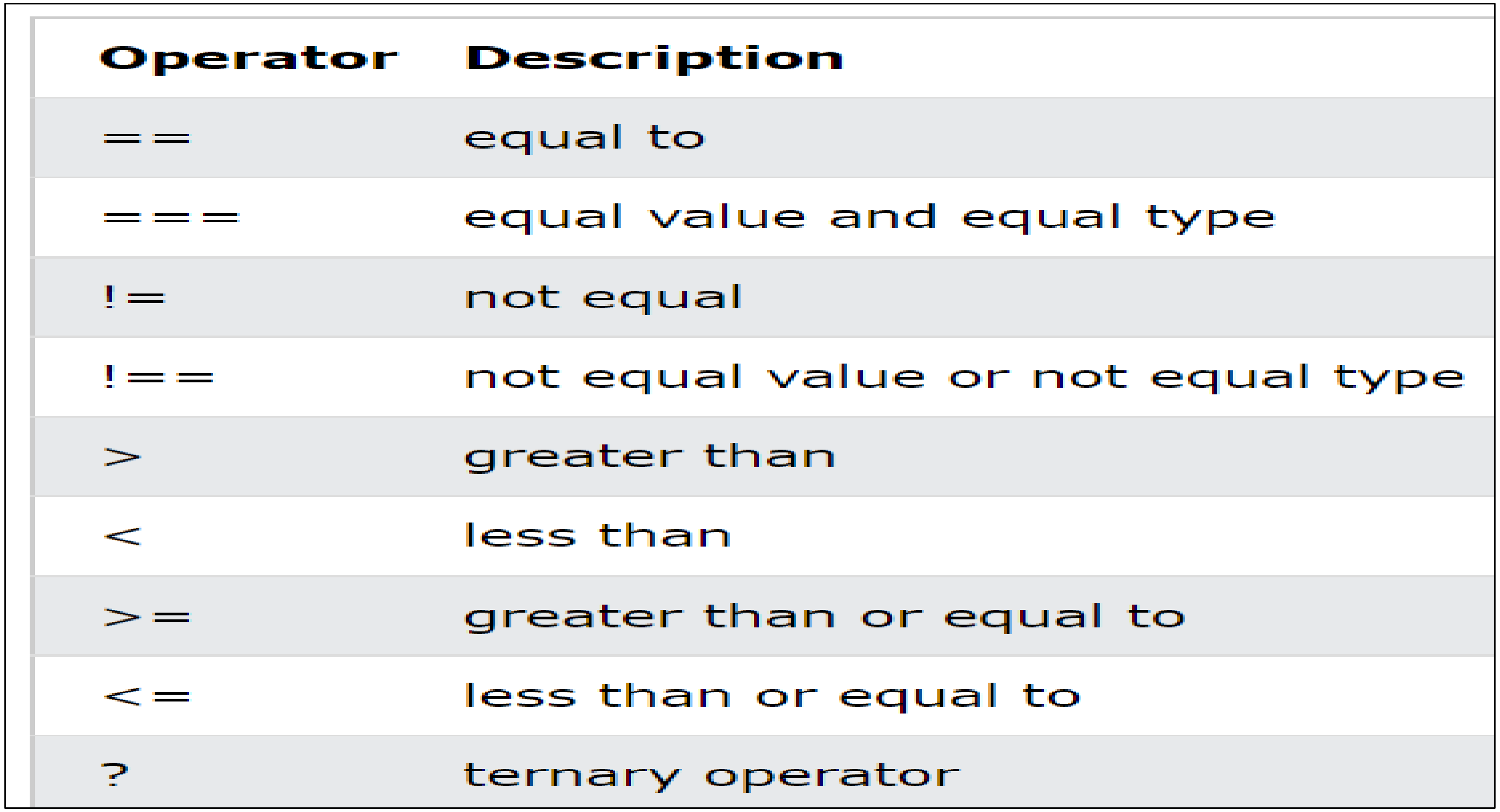
```
let x = 5 + 5;
let y = "5" + 5;
let z = "Hello" + 5;
```

The result of *x*, *y*, and *z* will be:

```
10
55
Hello5
```

If you add a number and a string, the result will be a string!

JavaScript Comparison Operators



Operator	Description
==	equal to
===	equal value and equal type
!=	not equal
!==	not equal value or not equal type
>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to
?	ternary operator

Example 1:

```
<p>Assign 5 to x, and display the value of the comparison (x === 5):</p>

<p id="demo"></p>

<script>
let x = 5;
document.getElementById("demo").innerHTML = (x === 5);
</script>
```

OUTPUT

```
Assign 5 to x, and display the value of the comparison (x === 5):

true
```

Example 2:

```
<script>
let x = 5;
document.getElementById("demo").innerHTML = (x === "5");
</script>
```

OUTPUT

```
false
```

Example 3:

```
<html>
<body>
<h1>JavaScript Comparison</h1>
<h2>The !== Operator</h2>
<p>Assign 5 to x, and display the value of the comparison (x !== 5):</p>
<p id="demo"></p>
<script>
let x = 5;
document.getElementById("demo").innerHTML = (x !== 5);
</script>
</body>
</html>
```

<u>OUTPUT</u>

JavaScript Comparison
The !== Operator
Assign 5 to x, and display the value of the comparison (x !== 5):
false

Example 4:

```
<html>
<body>
<h1>JavaScript Comparison</h1>
<h2>The !== Operator</h2>
<p>Assign 5 to x, and display the value of the comparison (x !== "5"):</p>
<p id="demo"></p>
<script>
let x = 5;
document.getElementById("demo").innerHTML = (x !== "5");
</script>
</body>
</html>
```

<u>OUTPUT</u>

```
JavaScript Comparison
The !== Operator
Assign 5 to x, and display the value of the comparison (x !== "5"):
true
```

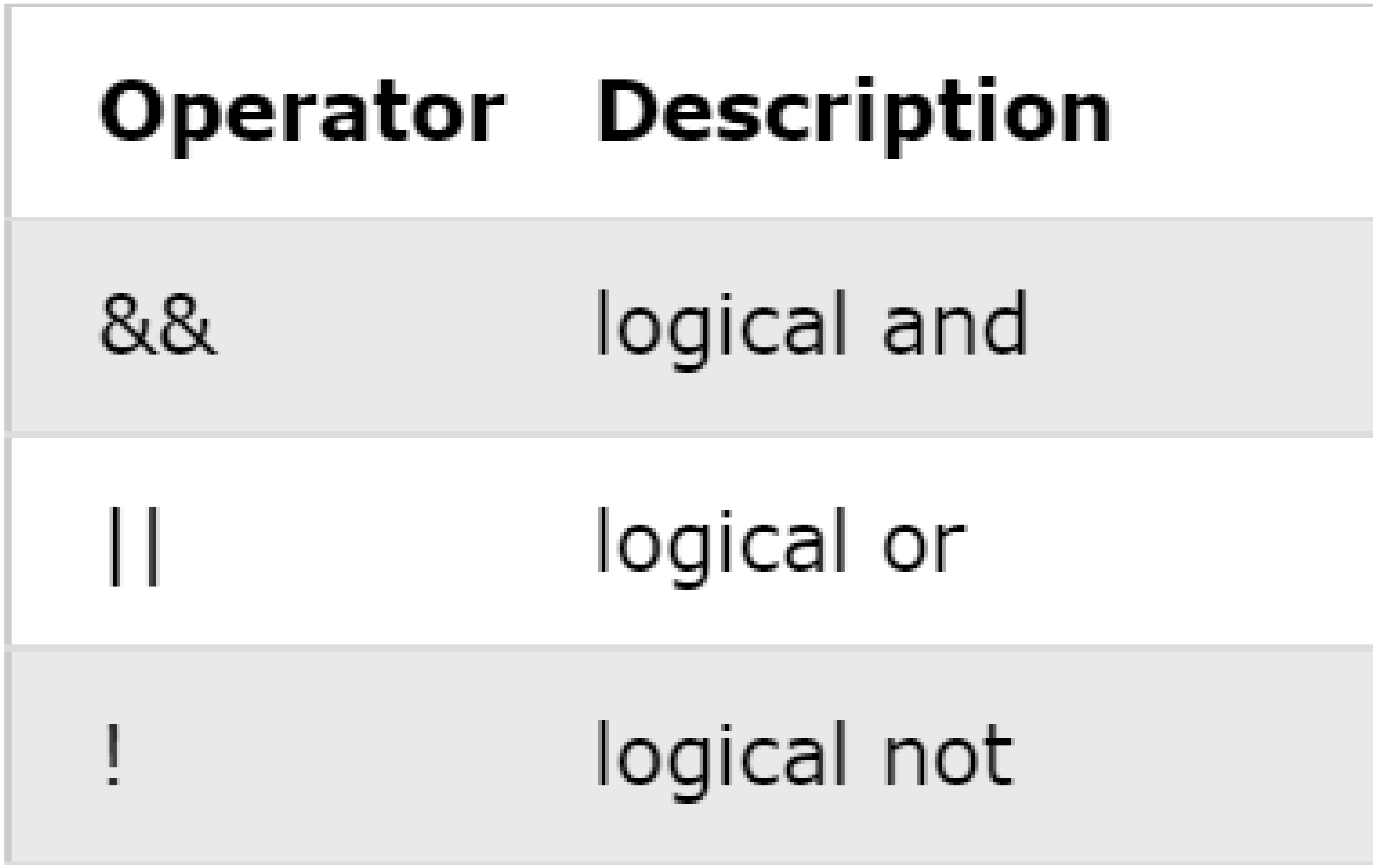
Example 5:

```
<html>
<body>
<h1>JavaScript Comparison</h1>
<h2>The !== Operator</h2>
<p>Assign 5 to x, and display the value of the comparison (x !== 8):</p>
<p id="demo"></p>
<script>
let x = 5;
document.getElementById("demo").innerHTML = (x !== 8);
</script>
</body>
</html>
```

OUTPUT

```
JavaScript Comparison
The !== Operator
Assign 5 to x, and display the value of the comparison (x !== 8):
true
```

JavaScript Logical Operators



Operator	Description
&&	logical and
||	logical or
!	logical not

The && Operator (Logical AND)

```
<html>
<body>
<h1>JavaScript Comparison</h1>
<h2>The && Operator (Logical AND)</h2>
<p>The && operator returns true if both expressions are true, otherwise it returns false.</p><p id="demo"></p>
<p id="demo"></p>
<script>
let x = 6;
let y = 3;
document.getElementById("demo").innerHTML =
(x < 10 && y > 1) + "<br>" +
(x < 10 && y < 1);
</script>
</body>
</html>
```

OUTPUT

JavaScript Comparison

The && Operator (Logical AND)

```
The && operator returns true if both expressions are true, otherwise it returns false.
true
false
```

The || Operator (Logical OR)

```
<h1>JavaScript Comparison</h1>
<h2>The || Operator (Logical OR)</h2>
<p>The || returns true if one or both expressions are true, otherwise it returns false.</p>
<p id="demo"></p>
<script>
let x = 6;
let y = 3;
document.getElementById("demo").innerHTML =
(x == 5 || y == 5) + "<br>" +
(x == 6 || y == 0) + "<br>" +
(x == 0 || y == 3) + "<br>" +
(x == 6 || y == 3);
</script>
```

OUTPUT

```
JavaScript Comparison
The || Operator (Logical OR)
The || returns true if one or both expressions are true, otherwise it returns false.
false
true
true
true
```

The NOT operator (!)

```
<h2>JavaScript Comparison</h2>

<p>The NOT operator (!) returns true for false statements and false for true statements.</p>

<p id="demo"></p>

<script>
let x = 6;
let y = 3;

document.getElementById("demo").innerHTML =
!(x === y) + "<br>" +
!(x > y);
</script>
```

OUTPUT:

```
JavaScript Comparison
The NOT operator (!) returns true for false statements and false for true statements.
true
false
```

Ternary operator?

A ternary operator evaluates a condition and executes a block of code based on the condition.

Syntax:

condition ? expression1 : expression2

The ternary operator evaluates the test condition.

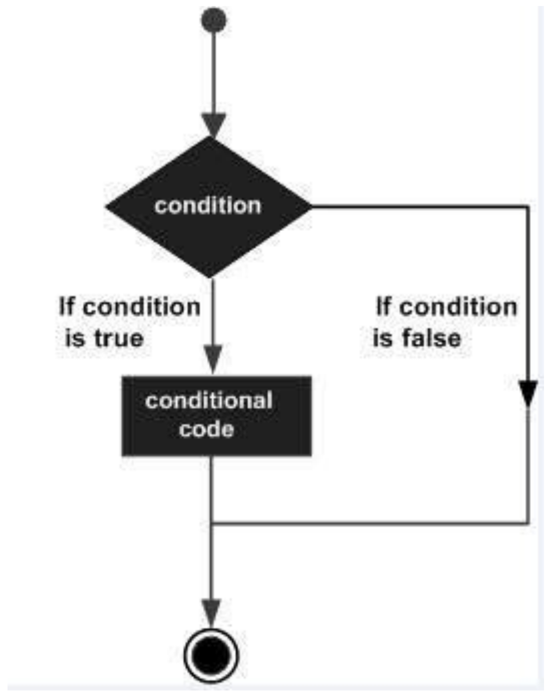
If the condition is true, expression1 is executed.
If the condition is false, expression2 is executed.

The ternary operator takes three operands, hence, the name ternary operator. It is also known as a conditional operator.

<u>Control structures and looping:</u>

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.

1) **if..else** statement.



Example

Try the following example to understand how the **if** statement works.

```
<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>
```

Example

Try the following code to learn how to implement an if-else statement in JavaScript.

```
<html>
  <body>

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

          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>
```

Example

Try the following code to learn how to implement an if-else-if statement in JavaScript.

```
<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>
```

The JavaScript Switch Statement

- The switch statement is used to perform different actions based on different conditions.
- Use switch to specify many alternative blocks of code to be executed

Syntax:

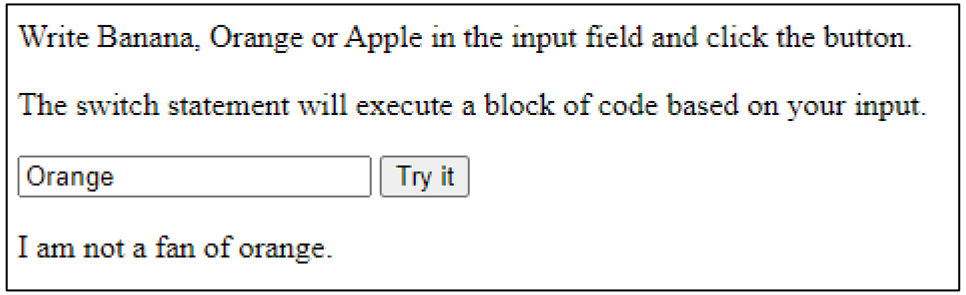
```
switch(expression) {
 case x:
  // code block
  break;
 case y:
  // code block
  break;
 default:
  // code block
}
```

This is how it works:
- The switch expression is evaluated once.
- The value of the expression is compared with the values of each case.
- If there is a match, the associated block of code is executed.
- If there is no match, the default code block is executed.

SWITCH CASE:

```
<html>
<body>
<input id="myInput" type="text">
<button onclick="myFunction()">Try it</button>
<p id="demo"></p>
<script>
function myFunction() {
 var text;
 var fruits = document.getElementById("myInput").value;
 switch(fruits) {
  case "Banana":
   text = "Banana is good!";
  break;
  case "Orange":
  text = "I am not a fan of orange.";
  break;
  case "Apple":
  text = "How you like them apples?";
  break;
  default:
  text = "I have never heard of that fruit...";
 }
 document.getElementById("demo").innerHTML = text;
}
</script>
</body>
</html>
```

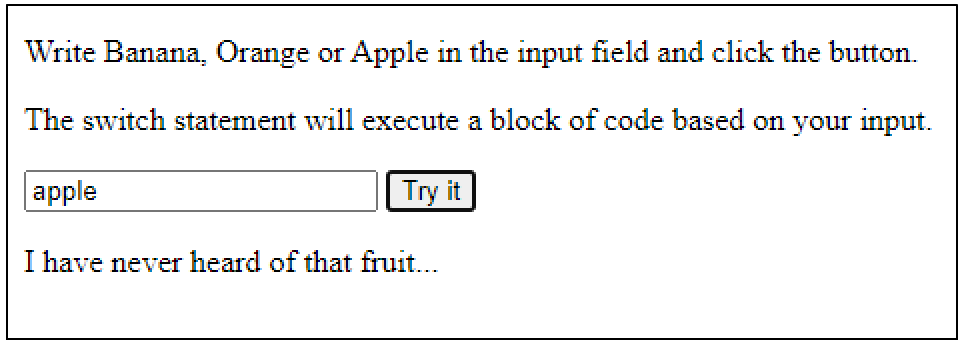
OUTPUT



Write Banana, Orange or Apple in the input field and click the button.

The switch statement will execute a block of code based on your input.

I am not a fan of orange.



Write Banana, Orange or Apple in the input field and click the button.

The switch statement will execute a block of code based on your input.

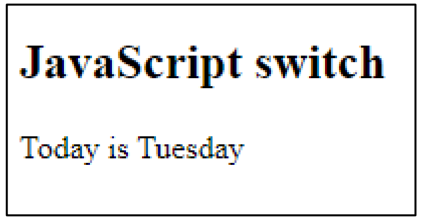
I have never heard of that fruit...

- The switch statement executes a block of code depending on different cases.
- The switch statement is a part of JavaScript's "Conditional" Statements, which are used to perform different actions based on different conditions. Use switch to select one of many blocks of code to be executed. This is the perfect solution for long, nested if/else statements.
- The switch statement evaluates an expression. The value of the expression is then compared with the values of each case in the structure. If there is a match, the associated block of code is executed.
- The switch statement is often used together with a break or a default keyword (or both). These are both optional:
- The **break keyword** breaks out of the switch block. This will stop the execution of more execution of code and/or case testing inside the block. If break is omitted, the next code block in the switch statement is executed.
- The **default keyword** specifies some code to run if there is no case match. There can only be one default keyword in a switch. Although this is optional, it is recommended that you use it, as it takes care of unexpected cases.

Example

- The getDay() method returns the weekday as a number between 0 and 6.
- (Sunday=0, Monday=1, Tuesday=2 ..)
- This example uses the weekday number to calculate the weekday name:

```
<html>
<body>
<h2>JavaScript switch</h2>
<p id="demo"></p>
<script>
let day;
switch (new Date().getDay()) {
  case 0:
    day = "Sunday";
    break;
  case 1:
    day = "Monday";
    break;
  case 2:
    day = "Tuesday";
    break;
  case 3:
    day = "Wednesday";
    break;
  case 4:
    day = "Thursday";
    break;
  case 5:
    day = "Friday";
    break;
  case  6:
    day = "Saturday";
}
document.getElementById("demo").innerHTML = "Today is " + day;
</script>
</body>
</html>
```



JavaScript switch

Today is Tuesday

The break Keyword

- When JavaScript reaches a break keyword, it breaks out of the switch block.
- This will stop the execution inside the switch block.
- It is not necessary to break the last case in a switch block. The block breaks (ends) there anyway.

The default Keyword

- The default keyword specifies the code to run if there is no case match:

- The getDay() method returns the weekday as a number between 0 and 6.

- If today is neither Saturday (6) nor Sunday (0), write a default message:

```
switch (new Date().getDay()) {
  case 6:
    text = "Today is Saturday";
    break;
  case 0:
    text = "Today is Sunday";
    break;
  default:
    text = "Looking forward to the Weekend";
}
```

Note:If default is not the last case in the switch block, remember to end the default case with a break.

```
<html>
<body>

<h2>JavaScript switch</h2>

<p id="demo"></p>

<script>
let text;
switch (new Date().getDay()) {
  default:
    text = "Looking forward to the Weekend";
    break;
  case 6:
    text = "Today is Saturday";
    break;
  case 0:
    text = "Today is Sunday";
}
document.getElementById("demo").innerHTML = text;
</script>

</body>
</html>
```

```
OUTPUT:

JavaScript switch
Looking forward to the Weekend
```

Common Code Blocks:

- Sometimes you will want different switch cases to use the same code.
- In this example case 4 and 5 share the same code block, and 0 and 6 share another code block:

```
<script>
let text;
switch (new Date().getDay()) {
  case 4:
  case 5:
    text = "Soon it is Weekend";
    break;
  case 0:
  case 6:
    text = "It is Weekend";
    break;
  default:
    text = "Looking forward to the Weekend";
}
document.getElementById("demo").innerHTML = text;
</script>
```

<u>OUTPUT:</u>

```
JavaScript switch
Looking forward to the Weekend
```

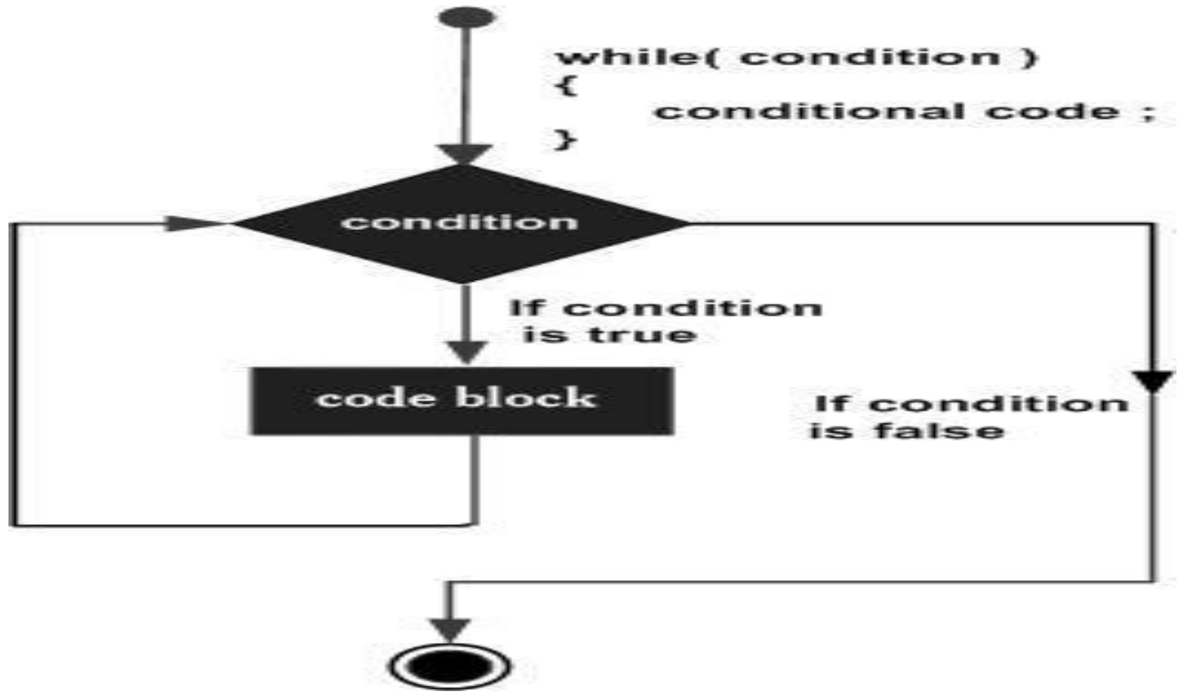
Switching Details

- If multiple cases matches a case value, the first case is selected.
- If no matching cases are found, the program continues to the default label.
- If no default label is found, the program continues to the statement(s) after the switch.

The 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.

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



The while Loop

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>
```

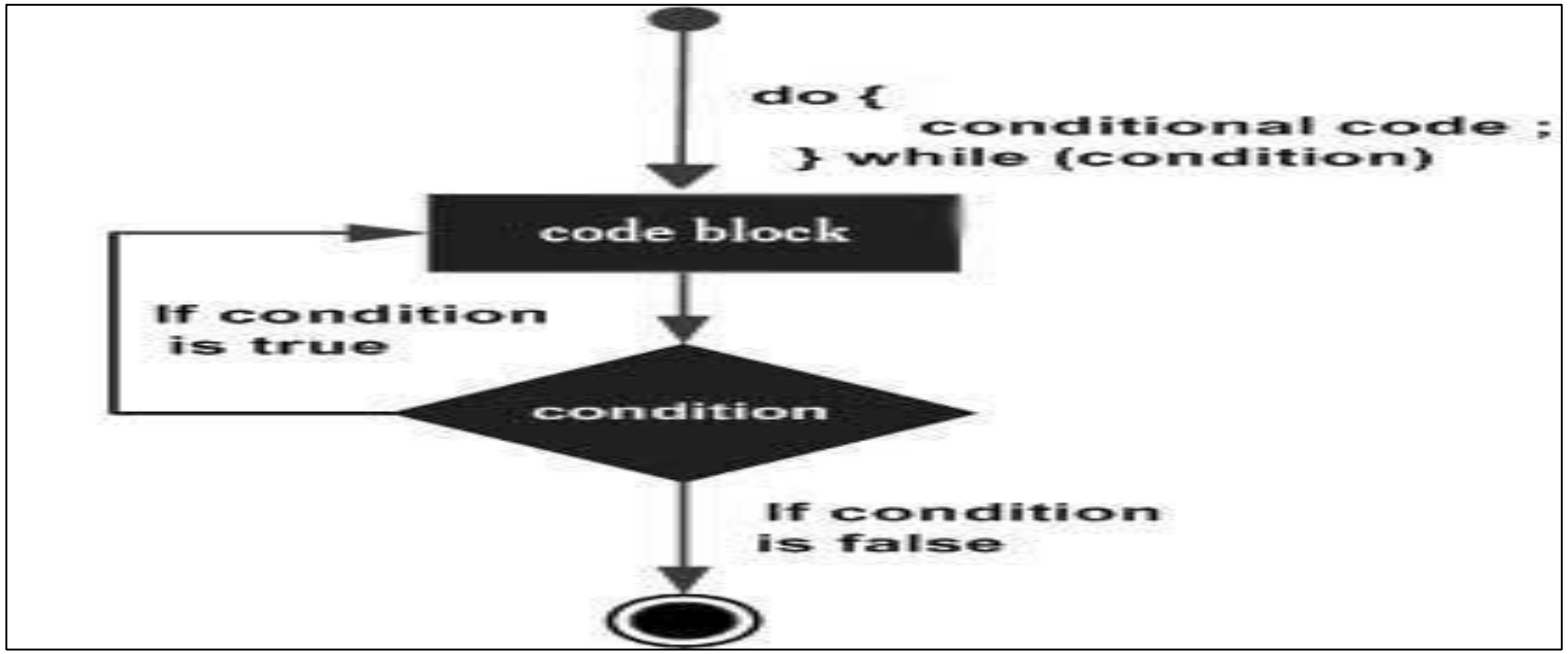
OUTPUT:

```
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...
```

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**.

```
do{
  Statement(s) to be executed;
}
while (expression);
```



The do...while Loop:

```
<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>
```

```
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.

FOR LOOP

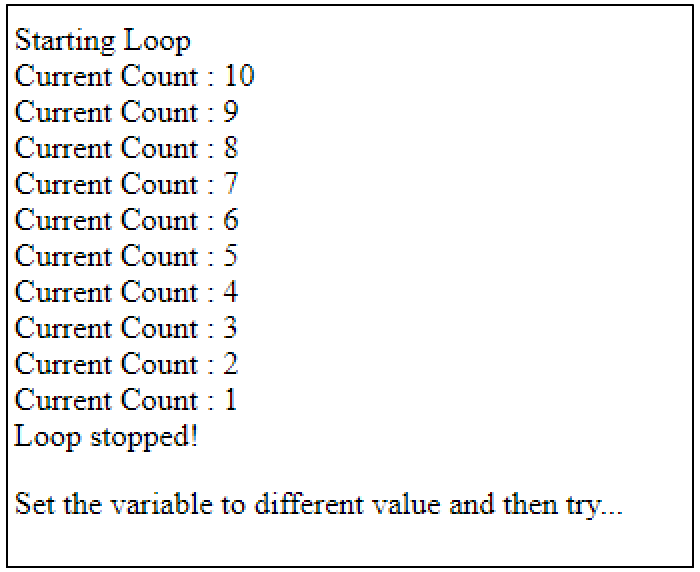
```
<html>
<body>
 <script type="text/javascript">
var count;
document.write("Starting Loop" + "<br />");
for(count = 0; count < 10; count++)
{
document.write("Current Count : " + count );
document.write("<br />");
}
 document.write("Loop stopped!");
</script>
<p>Set the variable to different value and 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...
```

For Loop with -- operator

```
<html>
<body>
 <script type="text/javascript">
var count;
document.write("Starting Loop" + "<br />");
for(count = 10; count > 0; 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>
```



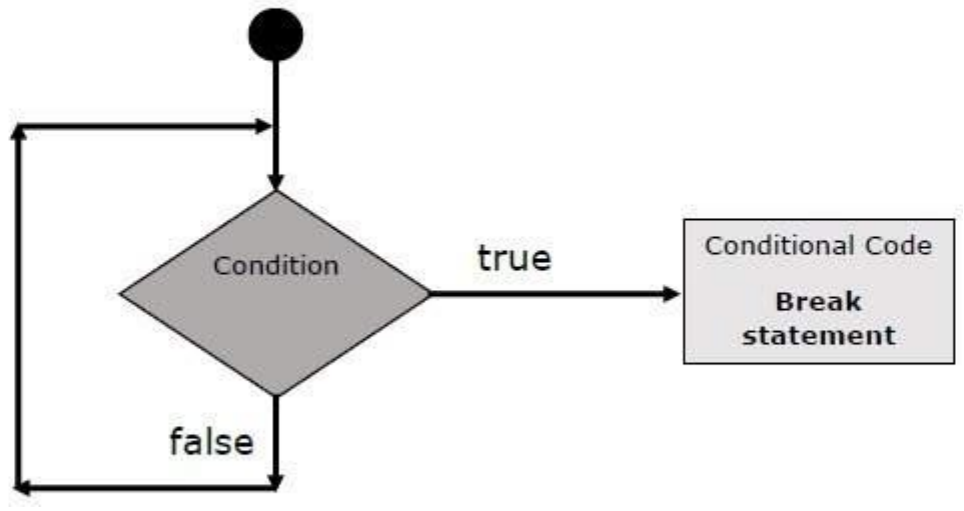
```
Starting Loop
Current Count : 10
Current Count : 9
Current Count : 8
Current Count : 7
Current Count : 6
Current Count : 5
Current Count : 4
Current Count : 3
Current Count : 2
Current Count : 1
Loop stopped!

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

The break Statement:

JavaScript provides full control to handle 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. These statements are used to immediately come out of any loop or to start the next iteration of any loop respectively.

The break Statement



Example:

```
<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 breaks one iteration (in the loop) if a specified condition occurs, and continues with the next iteration in the loop.
The difference between continue and the break statement, is instead of "jumping out" of a loop, the continue statement "jumps over" one iteration in the loop.

```
<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!
Set the variable to different value and then try...
```

Write javascript to print following pattern

```
*****
*****
*****
*****
*****
```

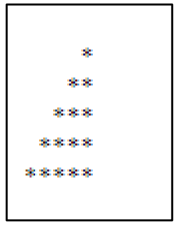
```
<html>
<body>
<script>
let n = 5; // row or column count
// defining an empty string
let string = "";

for(let i = 0; i < n; i++) { // external loop
  for(let j = 0; j < n; j++) { // internal loop
    string += "*";
  }
  // newline after each row
  string += "<br>";
}
// printing the string
document.write(string);
</script>
</body>
</html>
```

Write javascript to print following pattern

```
*
**
***
****
*****
```

```
<html>
<head>
   <title>right triangle star pattern - javascript</title>
</head>
<body>
  <h2>Right triangle star pattern in javascript</h2>
  <script>
    let n = 5; // you can take input from prompt or
change the value
    let string = "";
    for (let i = 0; i < n; i++) {
      for (let j = 0; j <= i; j++) {
        string += "*";
      }
      string += "<br>";
    }
    document.write(string);
  </script>
</body>
</html>
```

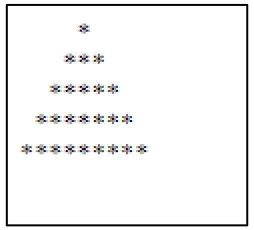
Write javascript to print following pattern



```
    *
   **
  ***
 ****
*****
```

```
<html>
<body>
  <h2>Left triangle star pattern in javascript</h2>
  <script>
    let n = 5; // you can take input from prompt or change the value
    let string = "";
    for (let i = 1; i <= n; i++) {
      // printing spaces
      for (let j = 0; j < n - i; j++) {
        string += " ";
      }
      // printing star
      for (let k = 0; k < i; k++) {
        string += "*";
      }
      string += "<br>";
    }
    document.write(`<pre>${string}</pre>`);
    //document.write(string);
  </script>
</body>
</html>
```

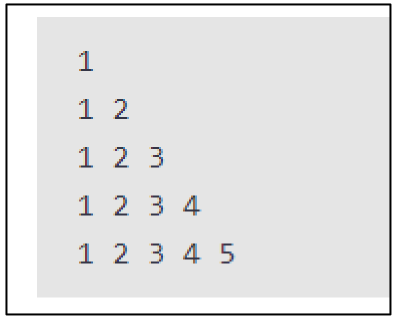
Write javascript to print following pattern



```
    *
   ***
  *****
 *******
*********
```

```
<html>
<body>
<script>
   let n = 5; // you can take input from prompt or change the value
   let string = "";
   // External loop
   for (let i = 1; i <= n; i++) {
     // printing spaces
     for (let j = n; j > i; j--) {
       string += " ";
     }
     // printing star
     for (let k = 0; k < i * 2 - 1; k++) {
       string += "*";
     }
     string += "<br>";
   }
   document.write(`<pre>${string}</pre>`);
   //document.write(string);
  </script>
</body>
</html>
```

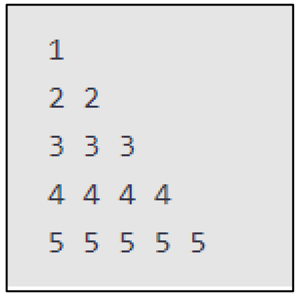
Write javascript to print following pattern



```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

```
<html>
<body>
  <h2>Number pattern using javascript</h2>
    <script>
    let n = 5; // height of pattern
    let string = "";
    // External loop
    for (let i = 1; i <= n; i++) {
      // Internal loop
      for (let j = 1; j <= i; j++) {
        string += j;
      }
      string += "<br>";
    }
    document.write(string);
  </script>
</body>
</html>
```

Write javascript to print following pattern



```
1
2 2
3 3 3
4 4 4 4
5 5 5 5 5
```

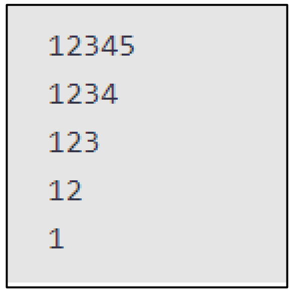
```
<html>
<body>
 <h2>Number pattern using javascript</h2>
  <script>
  let n = 5; // height of pattern
  let string = "";
  // External loop
  for (let i = 1; i <= n; i++) {
    // Internal loop
    for (let j = 1; j <= i; j++) {
      string += i;
    }
    string += "<br>";
  }
  document.write(string);
 </script>
</body>
</html>
```

Write javascript to print following pattern

```
1
2 3
4 5 6
7 8 9 10
```

```
<html>
<body>
 <h2>Number pattern using javascript</h2>
  <script>
  let n = 4; // height of pattern
  let string = "";
  let count = 1;
  // External loop
  for (let i = 1; i <= n; i++) {
   // Internal loop
   for (let j = 1; j <= i; j++) {
    string += count;
    count++;
   }
   string += "<br>";
  }
  document.write(string);
 </script>
</body>
</html>
```

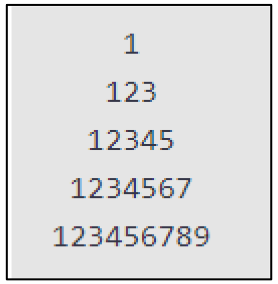
Write javascript to print following pattern



```
12345
1234
123
12
1
```

```
<html>
<body>
  <h2>Number pattern using javascript</h2>
    <script>
    let n = 5; // height of pattern
    let string = "";
    // External loop
    for (let i = 1; i <= n; i++) {
      for (let j = 1; j <= n - i + 1; j++) {
        string += j;
      }
      string += "<br>";
    }
    document.write(string);
  </script>
</body>
</html>
```

Write javascript to print following pattern

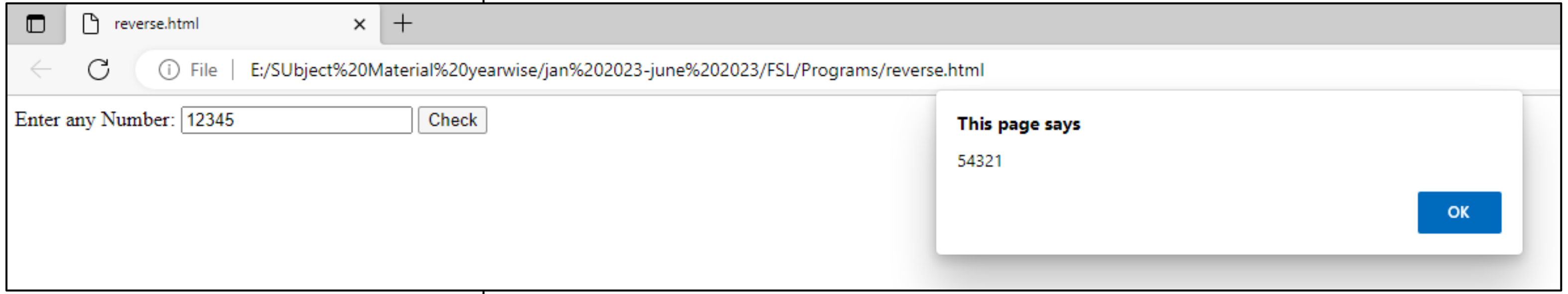


```
    1
   123
  12345
 1234567
123456789
```

```
<html>
<body>
  <script>
  let n = 5;
  let string = "";
  // External loop
  for (let i = 1; i <= n; i++) {
   // creating spaces
   for (let j = 1; j <= n - i; j++) {
    string += " ";
   }
   // creating alphabets
   for (let k = 1; k <= 2 * i - 1; k++) {
    string += k;
   }
   string += "<br>";
  }
  document.write(string);
 </script>
</body>
</html>
```

Write a javascript to reverse a given nos.

```
<html>
<head>
<script>
function palin()
{
var a,no,b,temp=0;
no=parseInt (document.getElementById("no_input").value);
b=no;
while(no>0)
{
a=no%10;
no=parseInt(no/10);
temp=temp*10+a;
}
alert(temp);
}
</script>
</head>
<body>
Enter any Number: <input id="no_input">
<button onclick="palin()">Check</button></br></br>
</body>
</html>
```


reverse.html
File
E:/SUbject%20Material%20yearwise/jan%202023-june%202023/FSL/Programs/reverse.html
Check
OK

Enter any Number: 12345

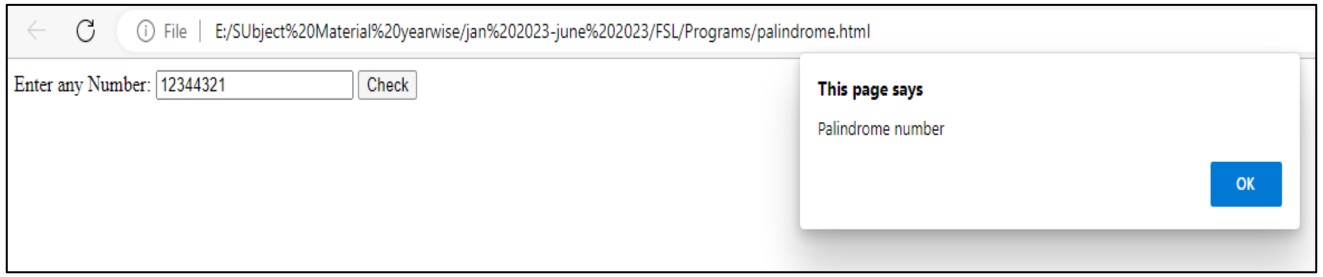
This page says

54321

Write a javascript to check given no is palindrome or not

```
<html>
<head>
<script>
function palin()
{
var a,no,b,temp=0;
no=parseInt(document.getElementById("no_input").value);
b=no;
while(no>0)
{
a=no%10;
no=parseInt(no/10);
temp=temp*10+a;
}
if(temp==b)
{
alert("Palindrome number");
}
else
{
alert("Not Palindrome number");
}
}
</script>
</head>
```

```
<body>
Enter any Number: <input id="no_input">
<button onclick="palin()">Check</button></br></br>
</body>
</html>
```


E:/SUbject%20Material%20yearwise/jan%202023-june%202023/FSL/Programs/palindrome.html

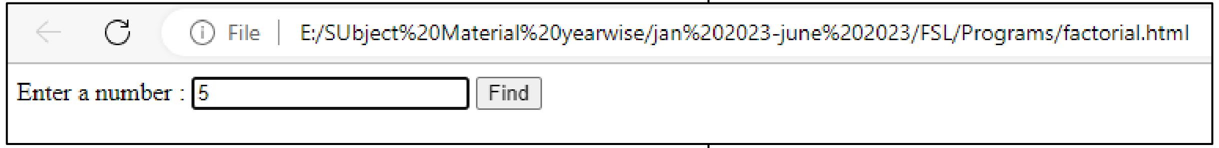
Enter any Number: 12344321

This page says

Palindrome number

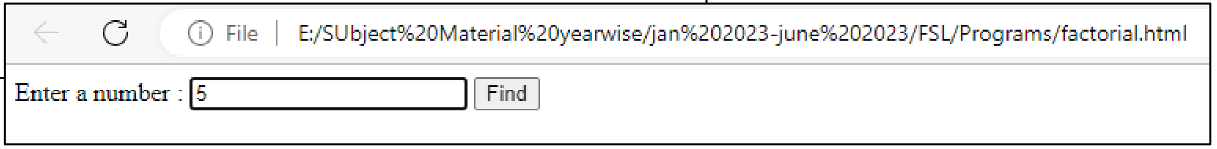
Factorial of a given no

```
<html>
<body>
<script>
function factor()
{
var f = 1;
var n = parseInt(document.getElementById("num").value);
for(i = 1;i<=n;i++)
{
   f = f*i ;
}
document.write (f);
}
</script>
<form>
Enter a number : <input type = "text" id = "num">
<input type = "button" value = "Find" onclick = "factor()" >
</form>
</body>
</html>
```



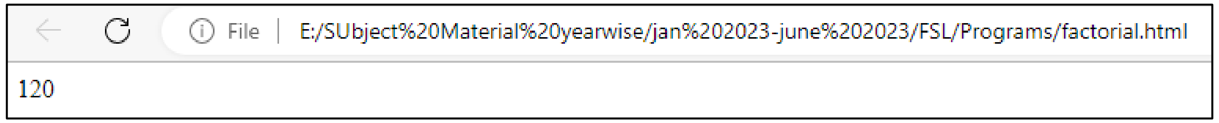
File | E:/SUbject%20Material%20yearwise/jan%202023-june%202023/FSL/Programs/factorial.html

Enter a number : 5 Find



File | E:/SUbject%20Material%20yearwise/jan%202023-june%202023/FSL/Programs/factorial.html

Enter a number : 5 Find



File | E:/SUbject%20Material%20yearwise/jan%202023-june%202023/FSL/Programs/factorial.html

```
120
```