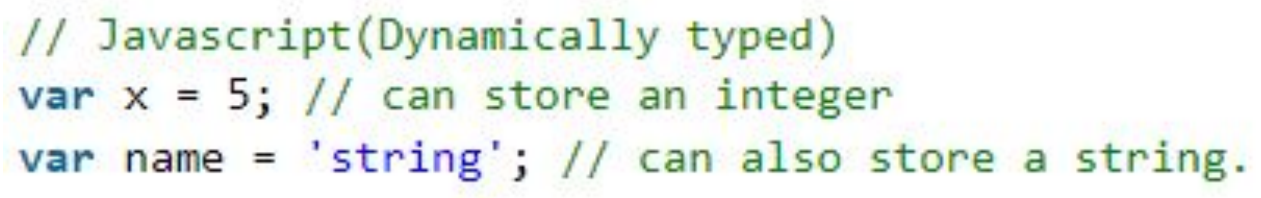
JavaScript

Data Types in JavaScript

- JavaScript is a Dynamically typed (loosely typed) scripting language.
- ie, in javascript variables can receive different data types over time.
- Datatypes are basically types of data that can be used and manipulated in a program.



```
// Javascript(Dynamically typed)
var x = 5; // can store an integer
var name = 'string'; // can also store a string.
```

JavaScript

Data Types in JavaScript

- A variable in JavaScript can contain any data; ie, a variable at one time can be a number and at another time be a string.
- Hence, JavaScript can receive different data types over time.
- Other examples for Dynamically typed languages - **Ruby, Python** etc.

JavaScript

Data Types in JavaScript

Six Primitive Data Types in Java Script

- **Numbers**: 5, 6.5, 7 etc.
- **String**: “Hello” etc.
- **Boolean**: Represent a logical entity and can have two values: true or false.
- **Null**: This type has only one value : null.
- **Undefined**: A variable that has not been assigned a value is undefined.
- **Object**: It is the most important data-type and forms the building blocks for modern JavaScript.

JavaScript

Data Types in JavaScript

number

- The number type in javascript contains both integer and floating point numbers.

  Besides these numbers, there are some ‘special-numbers’ in javascript that are:

  ‘Infinity’, ‘-Infinity’ and ‘NaN’.

- The Infinity basically represents the mathematical ‘?’.

  The ‘NaN’ denotes a computational error

JavaScript

Data Types in JavaScript

number

- let num = 2;   // integer
- let num2 = 1.3;   // floating point number
- let num3 = Infinity;   // Infinity
- let num4 = 'something here too'/2;   // NaN

JavaScript

Data Types in JavaScript

String

- A String in JavaScript is basically a series of characters that are surrounded by quotes.

Examples

- let str = "Hello There";
- let str2 = 'Single quotes works fine';
- There’s no difference between ‘single’ and “double” quotes in javascript.

JavaScript

Data Types in JavaScript

Boolean

- The boolean type has only two values: true and false.
- This data type is used to store yes/no values:
- true means “yes, correct”, and false means “no, incorrect”.

- let isCoding = true; // yes
- let isOld = false; // no

JavaScript

Data Types in JavaScript

null

- The special null value does not belong to any of the default data types.
- It forms a separate type of its own which contains only the null value:
- let age = null;
- The ‘null’ data type basically defines a special value which represents ‘nothing’, ’empty’ or ‘value unknown’.

JavaScript

Data Types in JavaScript

Undefined

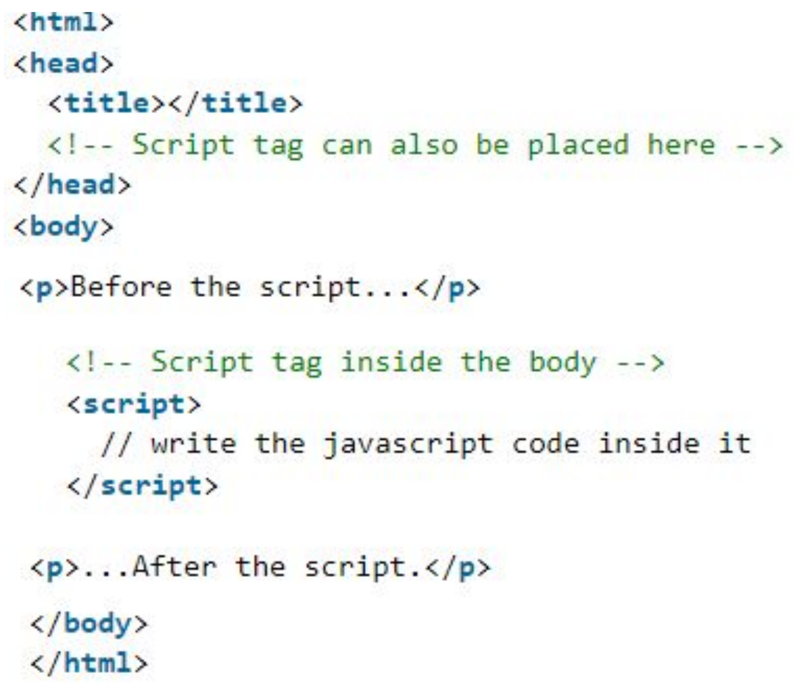
Just like null, Undefined makes its own type. The meaning of undefined is ‘value is not assigned’.

```
let x;
console.log(x);     // undefined
```

JavaScript

Printing Hello World in JavaScript

The ‘script’ tag is used to write javascript code and this ‘script’ tag is either placed inside the ‘head’ tag or inside the ‘body’ tag.

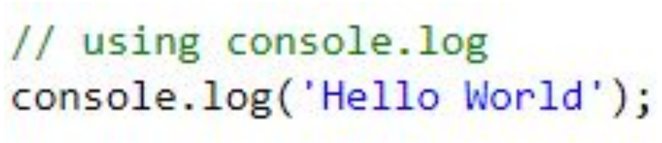


```
<html>
<head>
  <title></title>
  <!-- Script tag can also be placed here -->
</head>
<body>

<p>Before the script...</p>

   <!-- Script tag inside the body -->
   <script>
     // write the javascript code inside it
   </script>

 <p>...After the script.</p>

 </body>
 </html>
```

JavaScript

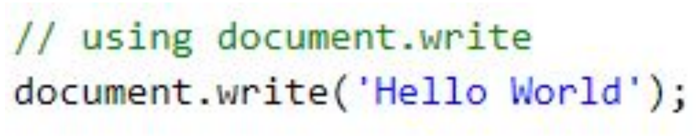
Printing Hello World in JavaScript

console.log()



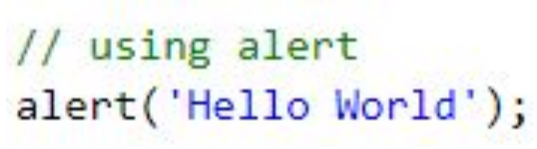
```
// using console.log
console.log('Hello World');
```

document.write()



```
// using document.write
document.write('Hello World');
```

alert()



```
// using alert
alert('Hello World');
```

JavaScript

Variables in JavaScript

In JavaScript, users can declare a variable using 3 keywords that are var, let, and const.

var

- The var is the oldest keyword to declare a variable in JavaScript.

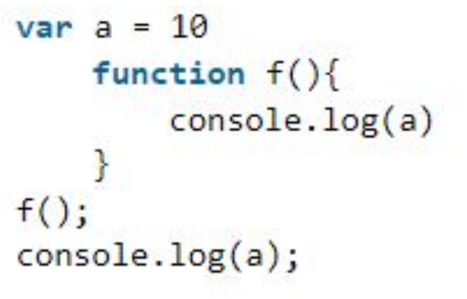
Scope: Global scoped or function scoped.

- The scope of the var keyword is the global or function scope.
- It means variables defined outside the function can be accessed globally, and variables defined inside a particular function can be accessed within the function.

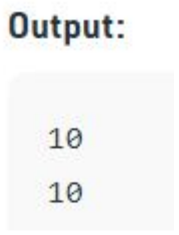
JavaScript

Variables in JavaScript

var



```
var a = 10
    function f(){
        console.log(a)
    }
f();
console.log(a);
```



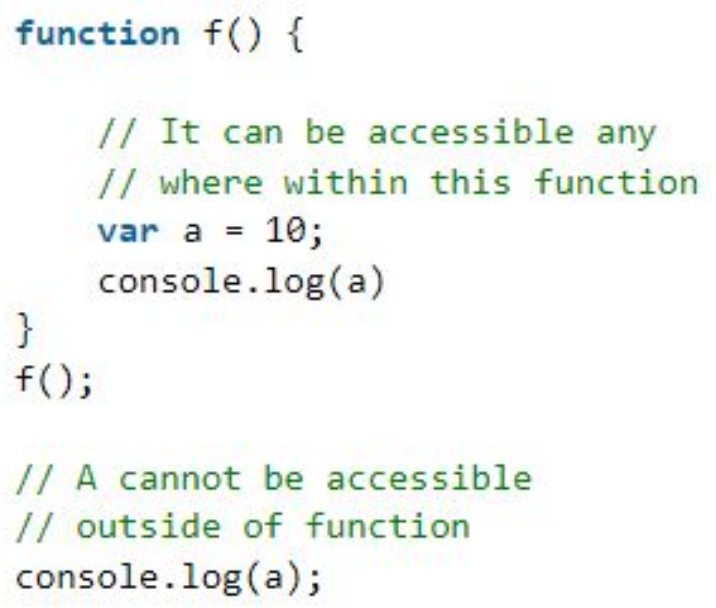
Output:

```
10
10
```

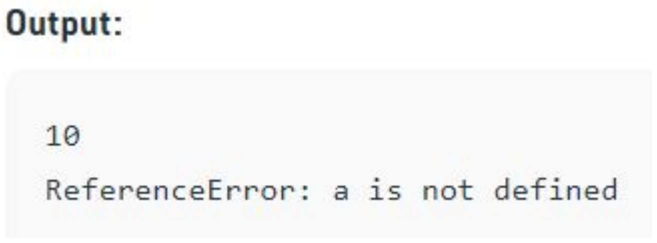
JavaScript

Variables in JavaScript

var



```
function f() {

    // It can be accessible any
    // where within this function
    var a = 10;
    console.log(a)
}
f();

// A cannot be accessible
// outside of function
console.log(a);
```



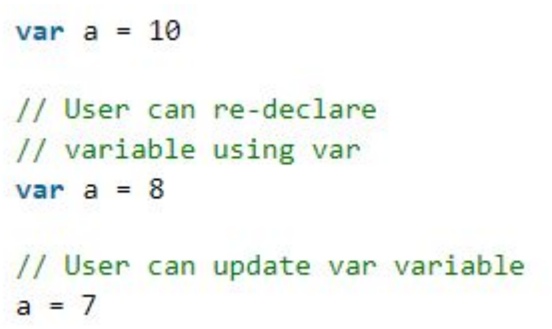
Output:

```
10
ReferenceError: a is not defined
```

JavaScript

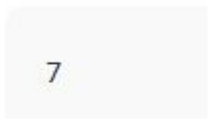
Variables in JavaScript

var



```
var a = 10

// User can re-declare
// variable using var
var a = 8

// User can update var variable
a = 7
```

Output:



```
7
```

JavaScript

Variables in JavaScript

let

- The let keyword is an improved version of the var keyword.

Scope: Block scoped

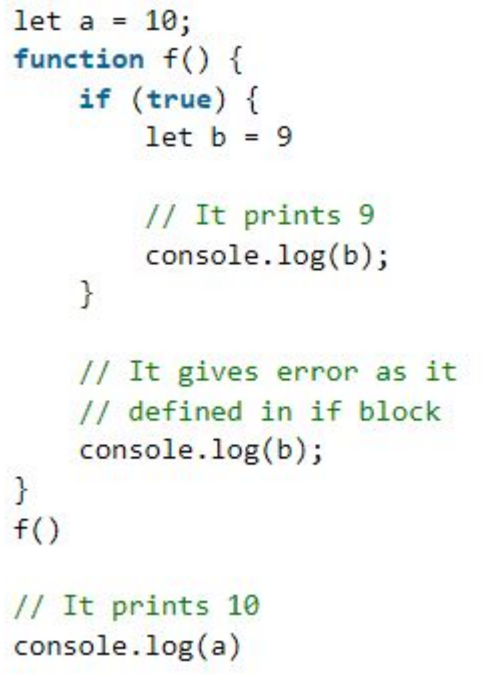
- The scope of a let variable is only block scoped. It can’t be accessible outside the particular block ({block}).

JavaScript

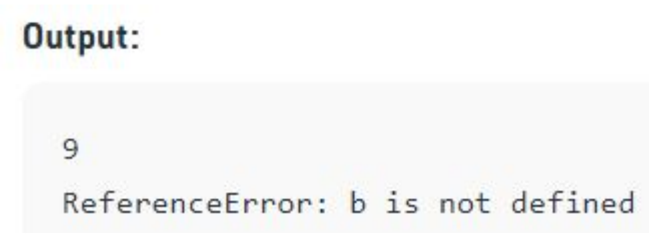
Variables in JavaScript

Let

The code returns an error because we are accessing the let variable outside the function block.



```
let a = 10;
function f() {
    if (true) {
        let b = 9

        // It prints 9
        console.log(b);
    }

    // It gives error as it
    // defined in if block
    console.log(b);
}
f()

// It prints 10
console.log(a)
```



Output:

```
9
ReferenceError: b is not defined
```

JavaScript

Variables in JavaScript

Let

Users cannot re-declare the variable defined with the let keyword but can update it.

```
let a = 10

// It is not allowed
let a = 10

// It is allowed
a = 10
```

Output:

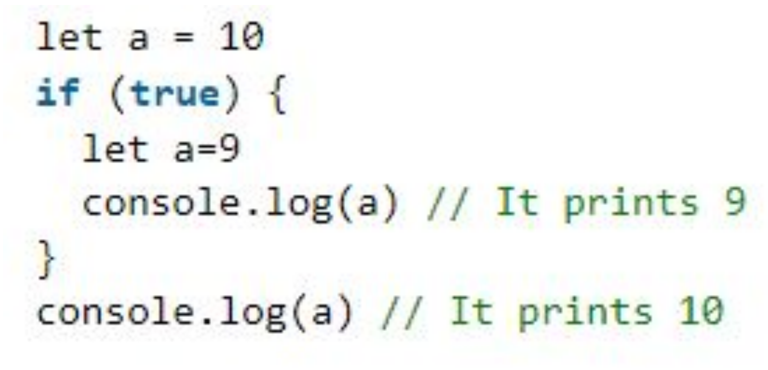
```
Uncaught SyntaxError: Identifier 'a' has already been declared
```

JavaScript

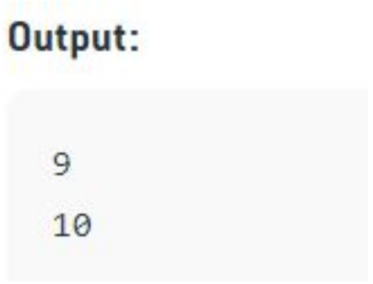
Variables in JavaScript

Let

Users can declare the variable with the same name in different blocks using the let keyword.



```
let a = 10
if (true) {
  let a=9
  console.log(a) // It prints 9
}
console.log(a) // It prints 10
```



Output:

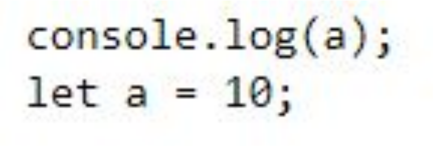
```
9
10
```

JavaScript

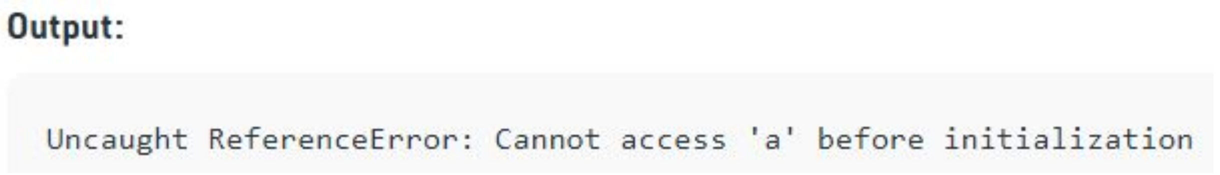
Variables in JavaScript

Let

If users use the let variable before the declaration, it does not initialize with undefined just like a var variable, and returns an error.



```
console.log(a);
let a = 10;
```



Output:

```
Uncaught ReferenceError: Cannot access 'a' before initialization
```

JavaScript

Variables in JavaScript

const

The const keyword has all the properties that are the same as the let keyword, except the user cannot update it.

Scope: block scoped

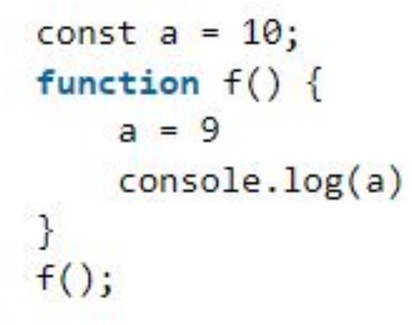
When users declare a const variable, they need to initialize it, otherwise, it returns an error. The user cannot update the const variable once it is declared.

JavaScript

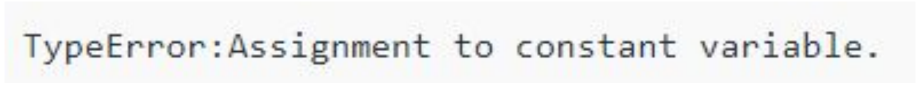
Variables in JavaScript

const

Changing the value of the const variable will return an error.



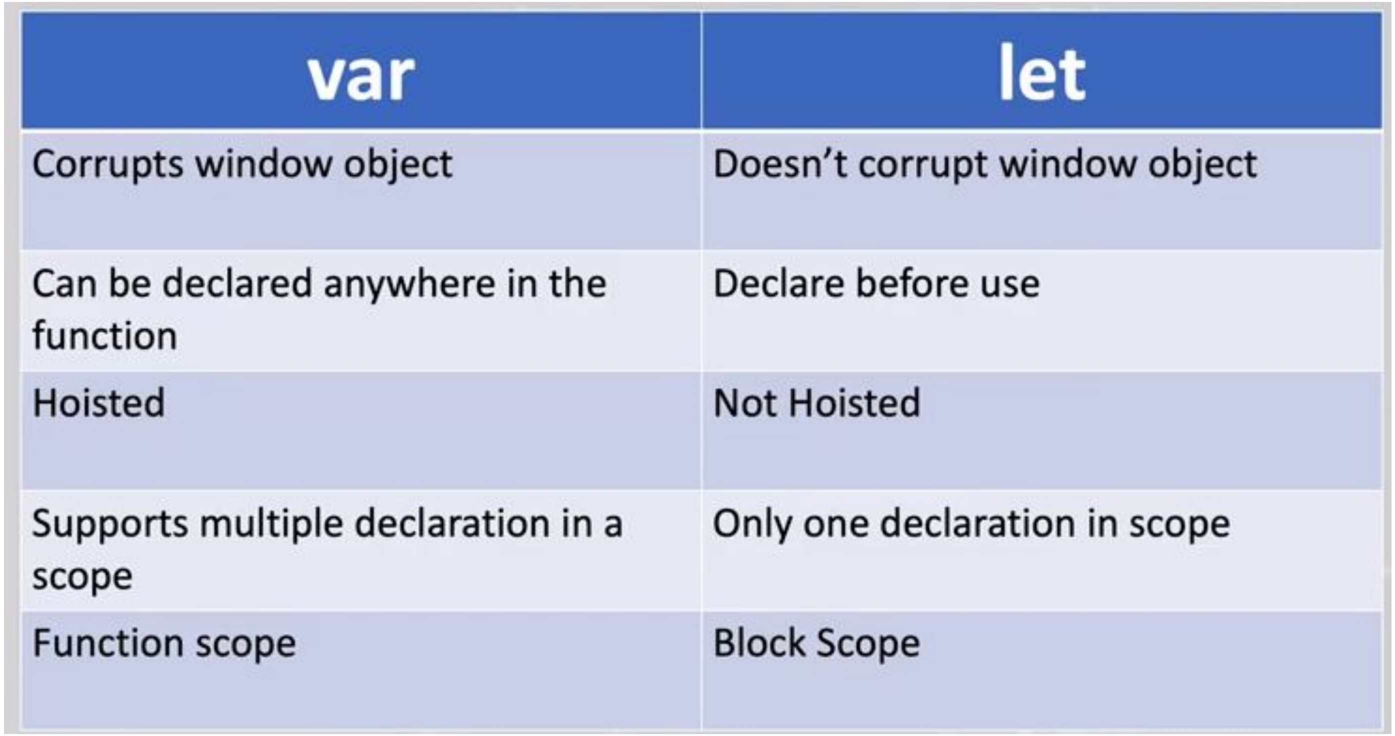
```
const a = 10;
function f() {
    a = 9
    console.log(a)
}
f();
```



```
TypeError:Assignment to constant variable.
```

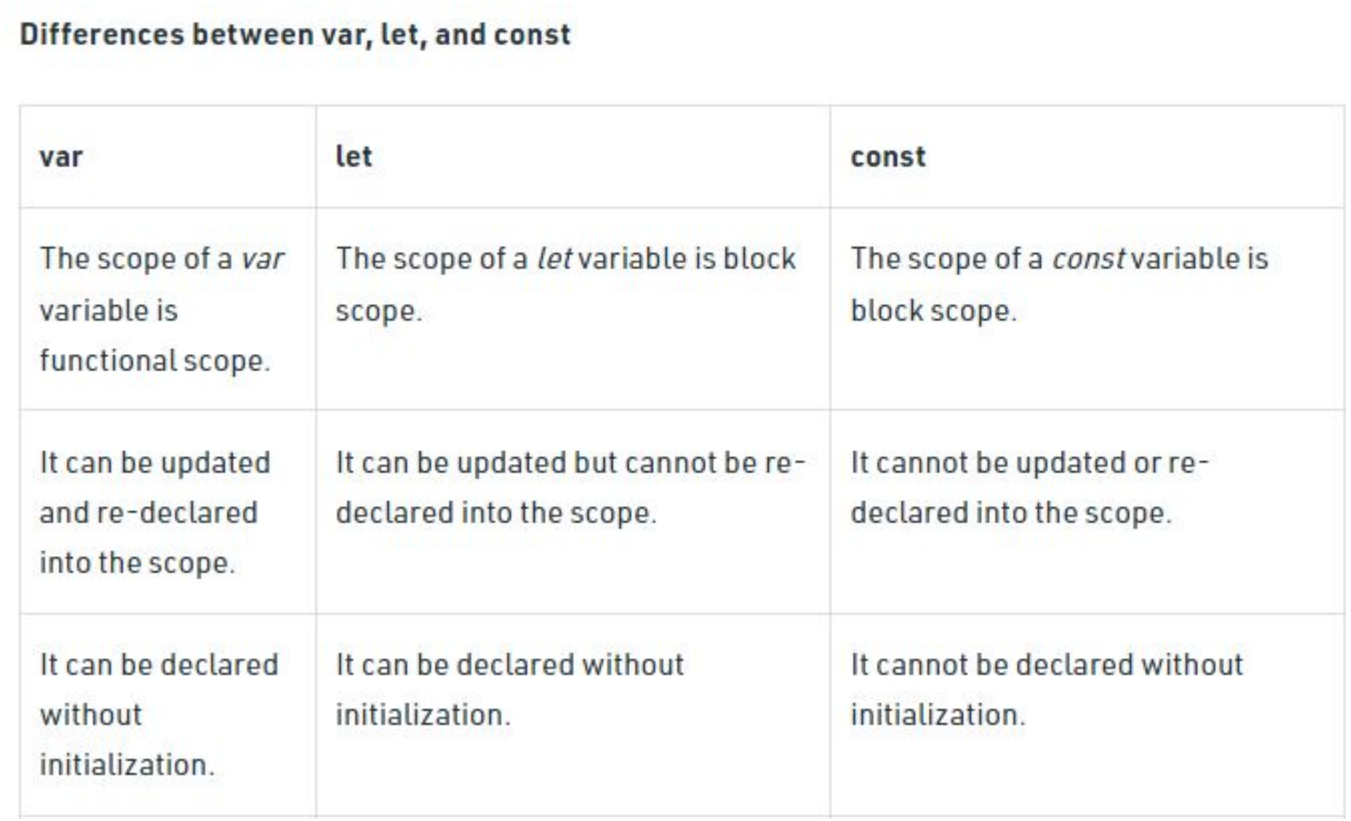
JavaScript

Variables in JavaScript



var	let
Corrupts window object	Doesn’t corrupt window object
Can be declared anywhere in the function	Declare before use
Hoisted	Not Hoisted
Supports multiple declaration in a scope	Only one declaration in scope
Function scope	Block Scope

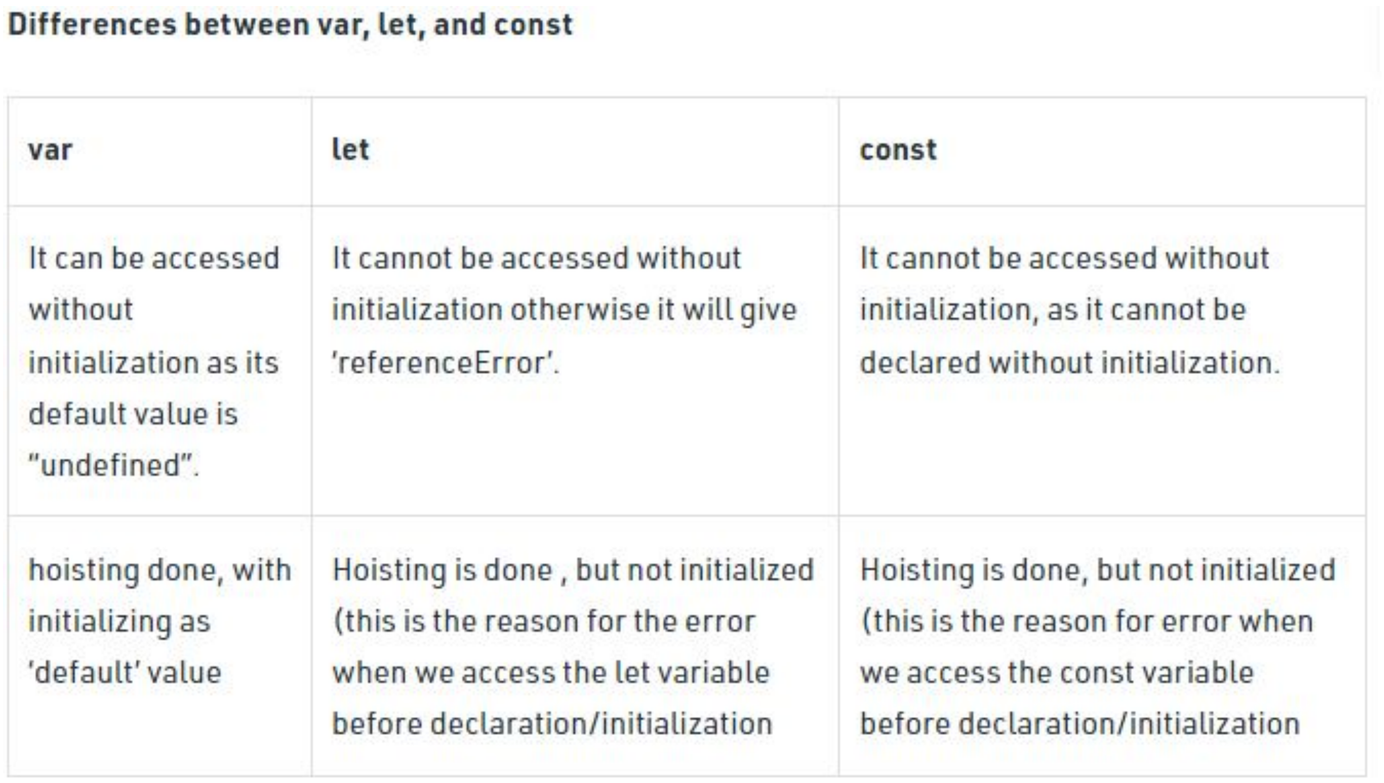
JavaScript



Differences between var, let, and const

var	let	const
The scope of a *var* variable is functional scope.	The scope of a *let* variable is block scope.	The scope of a *const* variable is block scope.
It can be updated and re-declared into the scope.	It can be updated but cannot be re-declared into the scope.	It cannot be updated or re-declared into the scope.
It can be declared without initialization.	It can be declared without initialization.	It cannot be declared without initialization.

JavaScript



Differences between var, let, and const

var	let	const
It can be accessed without initialization as its default value is "undefined".	It cannot be accessed without initialization otherwise it will give 'referenceError'.	It cannot be accessed without initialization, as it cannot be declared without initialization.
hoisting done, with initializing as 'default' value	Hoisting is done , but not initialized (this is the reason for the error when we access the let variable before declaration/initialization	Hoisting is done, but not initialized (this is the reason for error when we access the const variable before declaration/initialization

JavaScript

Control Statements in JavaScript

- if
- if-else
- nested-if
- if-else-if ladder

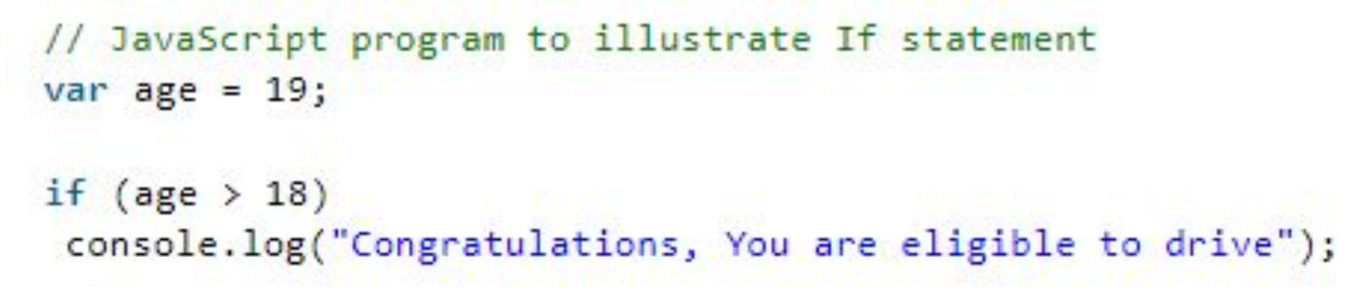
if-statement:

- **It is a conditional statement used to decide whether a certain statement or block of statements will be executed or not i.e if a certain condition is true then a block of statement is executed otherwise not.**

JavaScript

Control Statements in JavaScript

if-statement:



```
// JavaScript program to illustrate If statement
var age = 19;

if (age > 18)
 console.log("Congratulations, You are eligible to drive");
```

Syntax

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

JavaScript

Control Statements in JavaScript

if-statement:

- The if statement accepts boolean values – if the value is true then it will execute the block of statements under it.
- If we do not provide the curly braces '{' and '}' after if(condition) then by default if statement considers the immediate one statement to be inside its block.

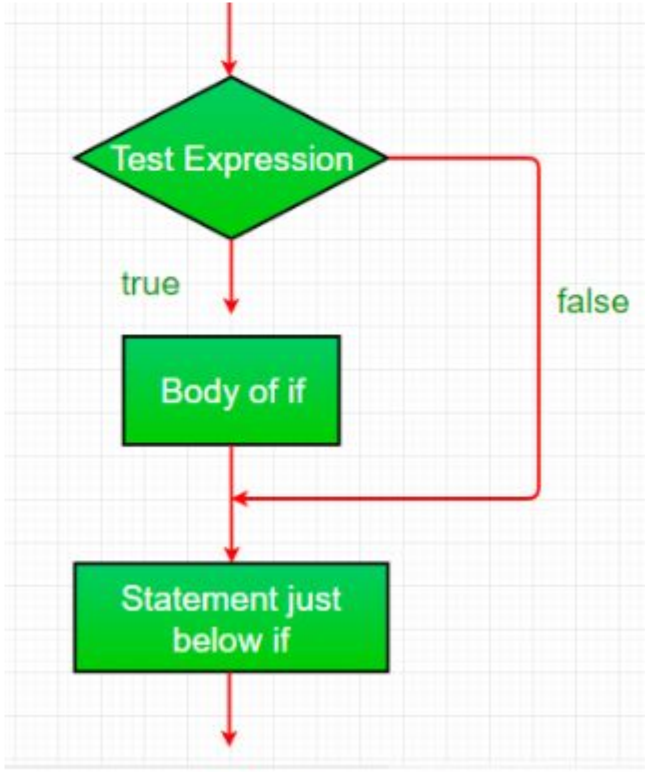
For example,

```
if(condition)
   statement1;
   statement2;
// Here if the condition is true, if block will consider only statement1 to be inside its block
```

JavaScript

Control Statements in JavaScript

Flow chart of if-statement :



if-condition statement

JavaScript

Control Statements in JavaScript

if-else

The if-else or conditional statement will perform some action for a specific condition.

If the condition meets then a particular block of action will be executed otherwise it will execute another block of action that satisfies that particular condition.

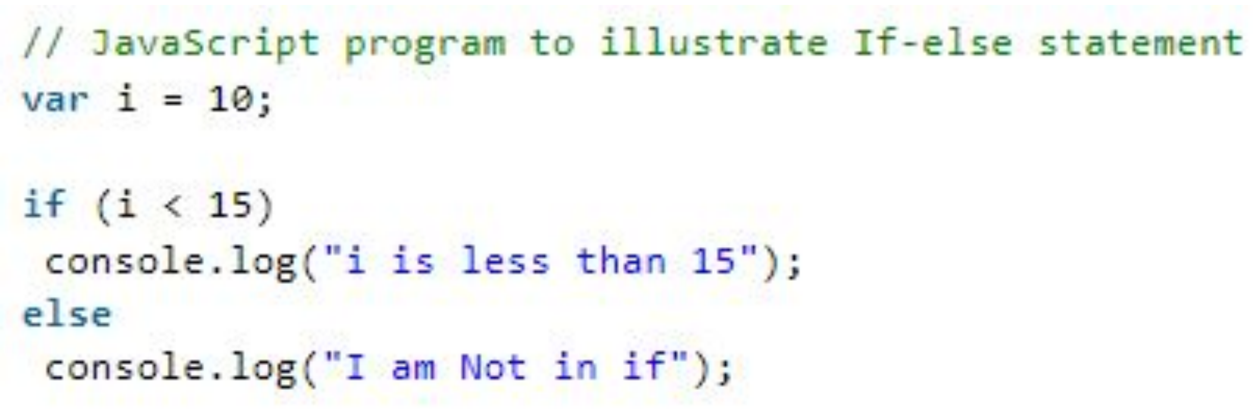
Such control statements are used to cause the flow of execution to advance and branch based on changes to the state of a program.

JavaScript

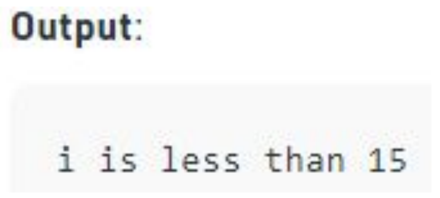
Control Statements in JavaScript

if-else Syntax:

```
if (condition)
{
   // Executes this block if
   // condition is true
}
else
{
   // Executes this block if
   // condition is false
}
```



```
// JavaScript program to illustrate If-else statement
var i = 10;

if (i < 15)
 console.log("i is less than 15");
else
 console.log("I am Not in if");
```



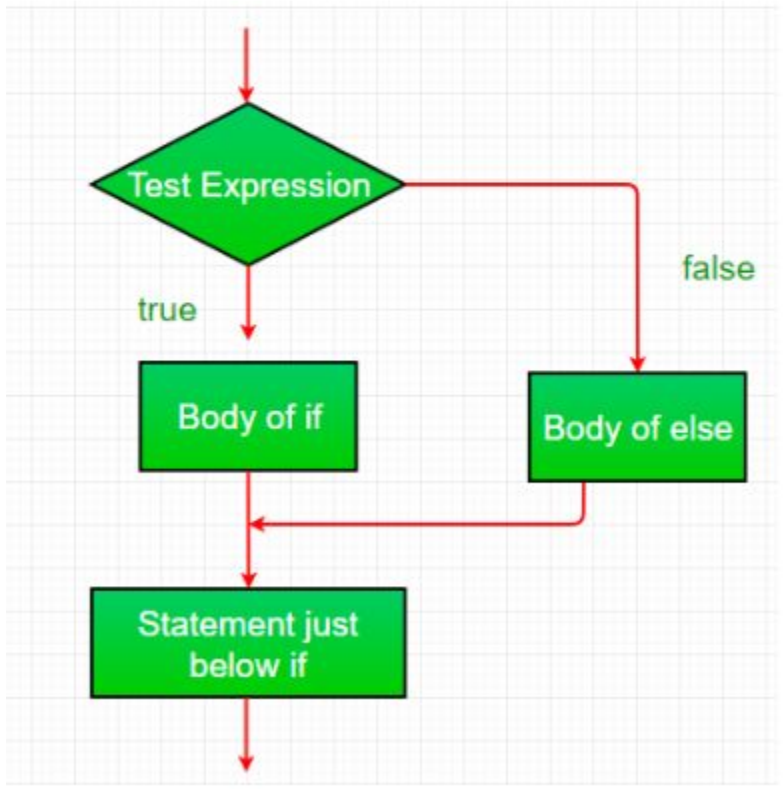
Output:

```
i is less than 15
```

JavaScript

Control Statements in JavaScript

Flow chart of if-else statement :



if-else statement

JavaScript

Control Statements in JavaScript

nested-if statement:

- JavaScript allows us to nest if statements within if statements.
- i.e, we can place an if statement inside another if statement.
- A nested if is an if statement that is the target of another if or else.

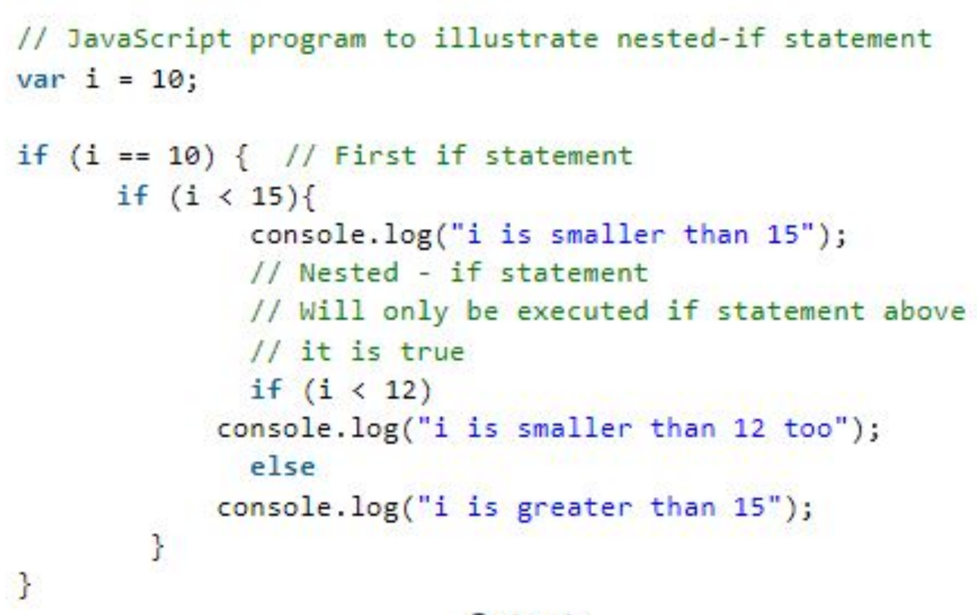
JavaScript

Control Statements in JavaScript

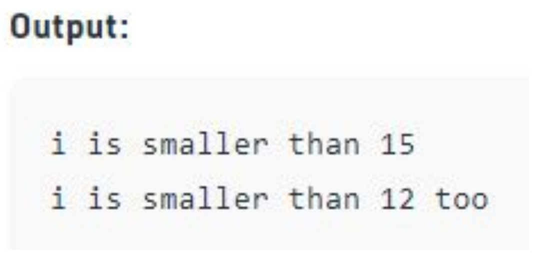
nested-if statement:

Syntax:

```
if (condition1)
{
   // Executes when condition1 is true
   if (condition2)
   {
      // Executes when condition2 is true
   }
}
```



```
// JavaScript program to illustrate nested-if statement
var i = 10;

if (i == 10) {  // First if statement
      if (i < 15){
              console.log("i is smaller than 15");
              // Nested - if statement
              // Will only be executed if statement above
              // it is true
              if (i < 12)
            console.log("i is smaller than 12 too");
              else
            console.log("i is greater than 15");
        }
}
```



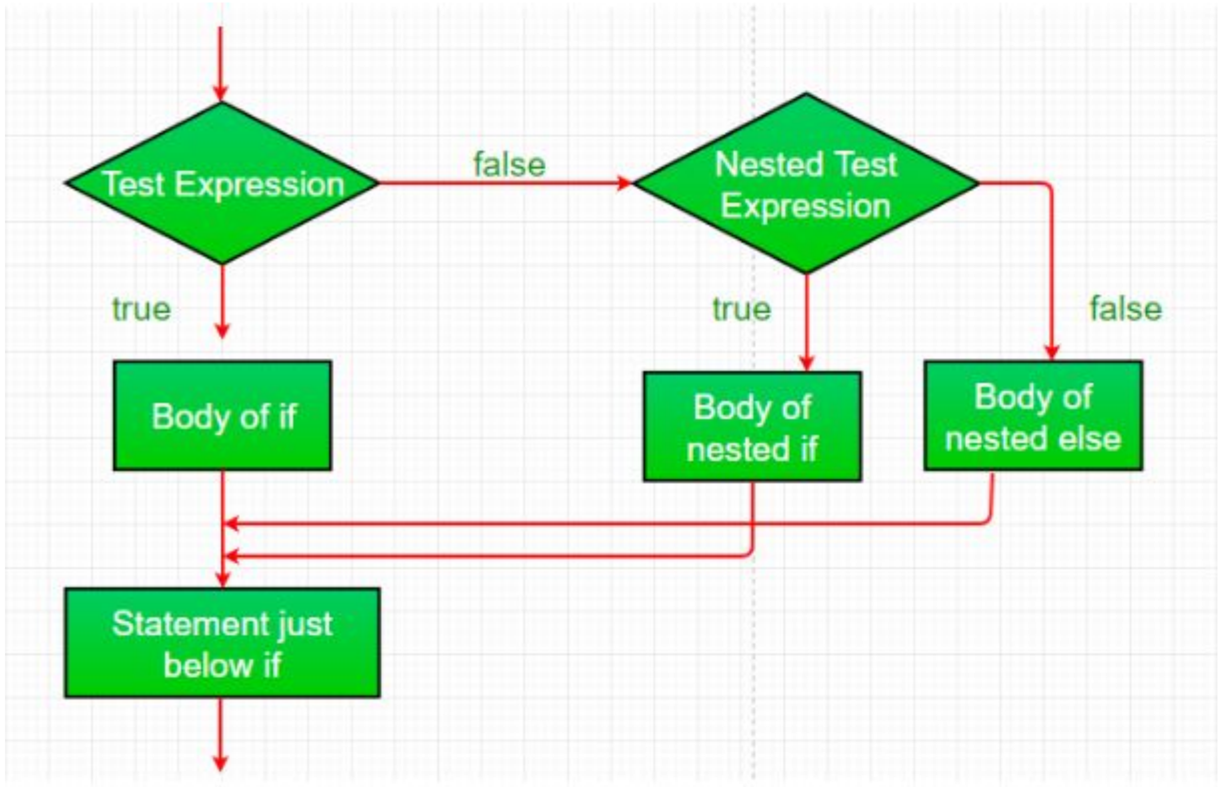
Output:

```
i is smaller than 15
i is smaller than 12 too
```

JavaScript

Control Statements in JavaScript

nested-if statement:



nested-if statement

JavaScript

Control Statements in JavaScript

if-else-if ladder statement:

- Here, a user can decide among multiple options. The if statements are executed from the top down.
- As soon as one of the conditions controlling the if is true, the statement associated with that if is executed, and the rest of the ladder is bypassed.
- If none of the conditions is true, then the final else statement will be executed.

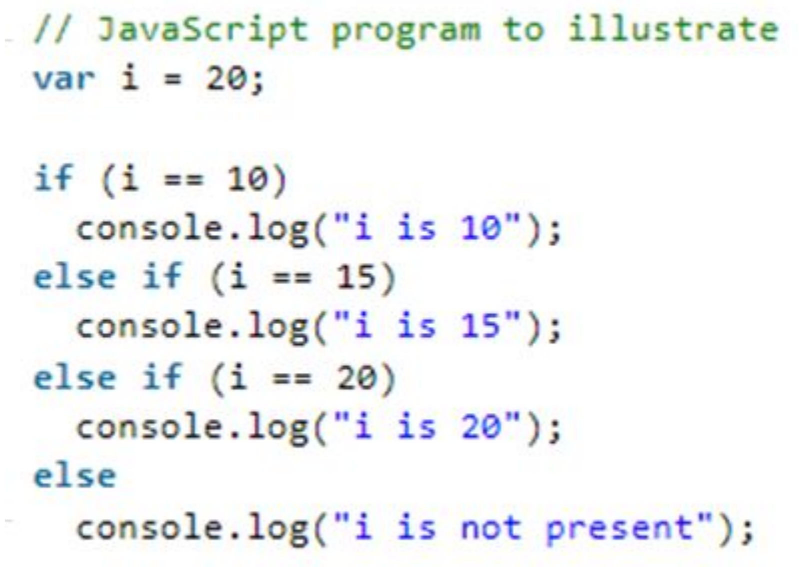
JavaScript

Control Statements in JavaScript

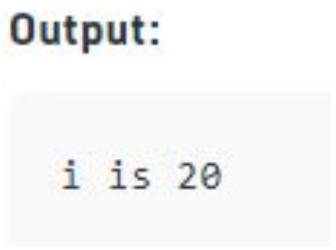
if-else-if ladder statement

Syntax:

```
if (condition)
    statement;
else if (condition)
    statement;
.
.
else
    statement;
```



```
// JavaScript program to illustrate
var i = 20;

if (i == 10)
  console.log("i is 10");
else if (i == 15)
  console.log("i is 15");
else if (i == 20)
  console.log("i is 20");
else
  console.log("i is not present");
```



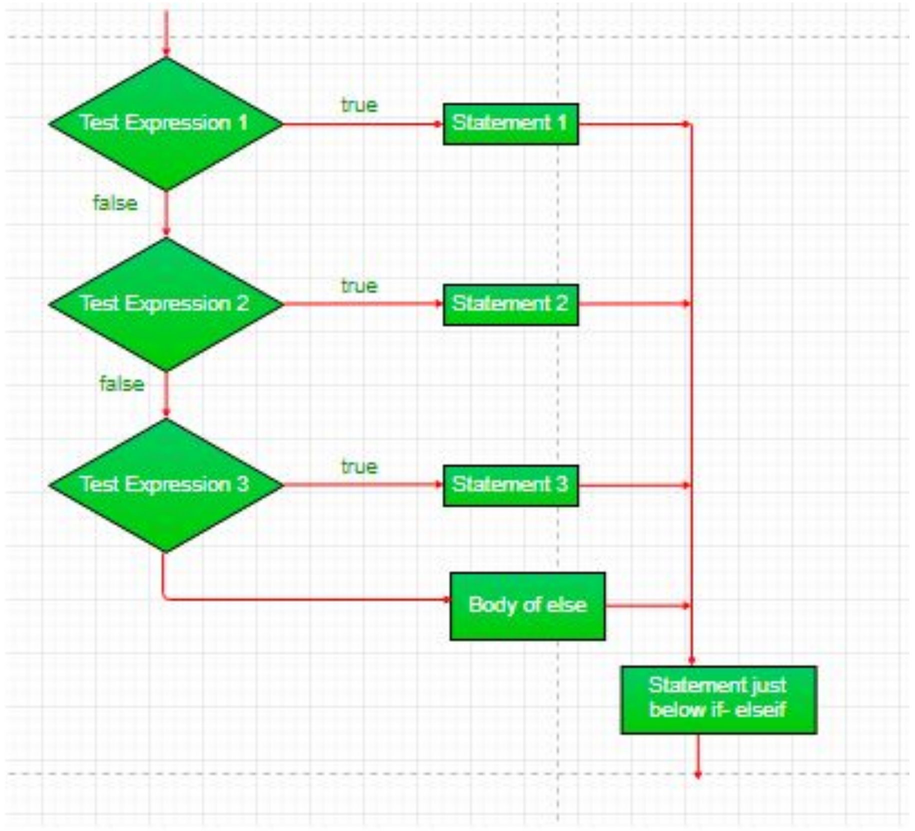
Output:

```
i is 20
```

JavaScript

Control Statements in JavaScript

if-else-if ladder statement



if-else-if ladder statement

JavaScript

Control Statements in JavaScript

switch - case:

- The switch case statement in JavaScript is also used for decision-making purposes.
- In some cases, using the switch case statement is seen to be more convenient than if-else statements.
- Consider a situation when we want to test a variable for hundred different values and based on the test we want to execute some task.
- Using if-else statements for this purpose will be less efficient than switch-case statements
- The switch case statement is a multiway branch statement.
- It provides an easy way to dispatch execution to different parts of code based on the value of

JavaScript

Control Statements in JavaScript

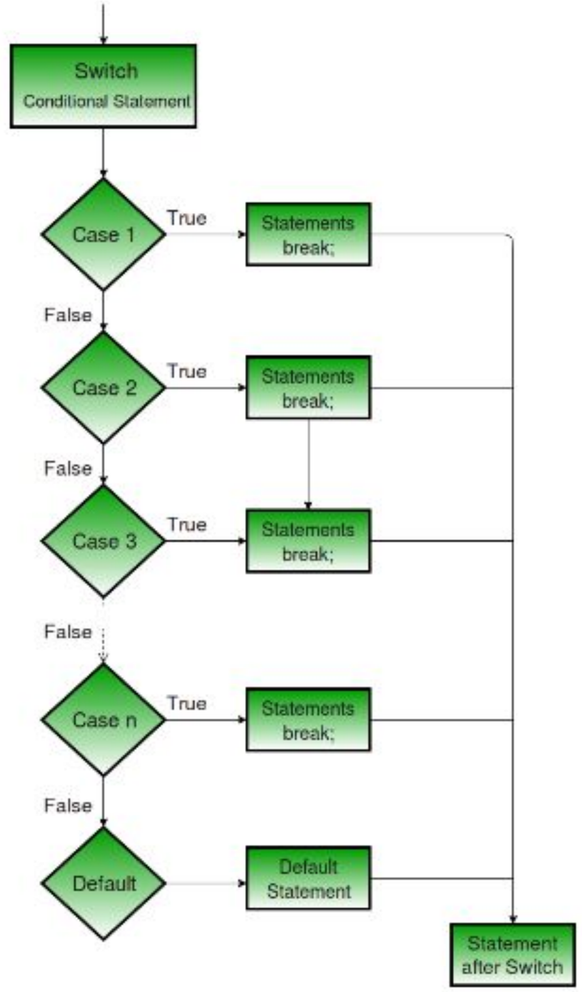
switch case - Syntax

```
switch (expression)
{
    case value1:
        statement1;
        break;
    case value2:
        statement2;
        break;
    .
    .
    case valueN:
        statementN;
        break;
    default:
        statementDefault;
}
```

Syntax Explanation

- **The expression can be of type numbers or strings.**
- **Duplicate case values are not allowed.**
- **The default statement is optional. If the expression passed to the switch does not match the value in any case then the statement under default will be executed.**
- **The break statement is used inside the switch to terminate a statement sequence.**
- **The break statement is optional. If omitted, execution will continue on into the next case.**
- **Cases are compared strictly.**

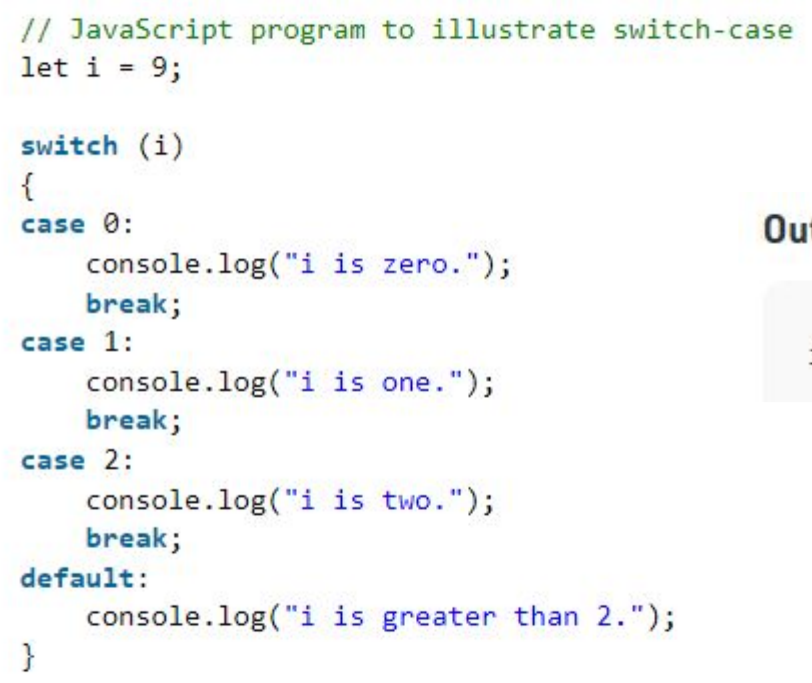
JavaScript



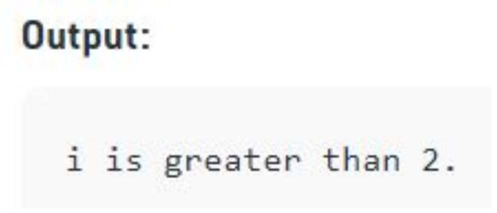
switch case Flowchart

JavaScript

Switch case



```
// JavaScript program to illustrate switch-case
let i = 9;

switch (i)
{
case 0:
    console.log("i is zero.");
    break;
case 1:
    console.log("i is one.");
    break;
case 2:
    console.log("i is two.");
    break;
default:
    console.log("i is greater than 2.");
}
```



Output:

```
i is greater than 2.
```

JavaScript

Loops in JavaScript

- Looping is a feature that facilitates the execution of a set of instructions/functions repeatedly while some condition evaluates to true.

- For example, suppose we want to print “Hello World” 5 times, then The iterative method to do this is to write the document.write() statement 5 times as shown below.

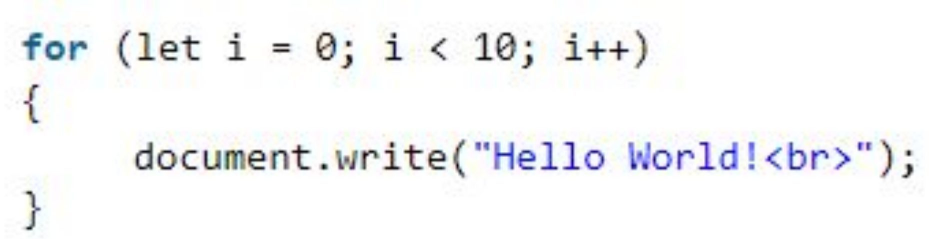
```
document.write("Hello World<br>");
document.write("Hello World<br>");
document.write("Hello World<br>");
document.write("Hello World<br>");
document.write("Hello World<br>");
```

JavaScript

Loops in JavaScript

Using Loops:

- **In Loop, the statement needs to be written only once and here the loop will be executed 10 times as shown below:**



```
for (let i = 0; i < 10; i++)
{
      document.write("Hello World!<br>");
}
```

JavaScript

Loops in JavaScript

There are mainly two types of loops:

- Entry Controlled loops: In these types of loops, the test condition is tested before entering the loop body. For Loops and While Loops are entry-controlled loops.

- Exit Controlled loops: In these types of loops the test condition is tested or evaluated at the end of the loop body. Therefore, the loop body will execute at least once, irrespective of whether the test condition is true or false. The do-while loop is exit controlled loop.

JavaScript

Loops in JavaScript

while loop: A while loop is a control flow statement that allows code to be executed repeatedly based on a given Boolean condition. The while loop can be thought of as a repeating if statement.

Syntax :

```
while (boolean condition)
{
   loop statements...
}
```

JavaScript

Loops in JavaScript

- While loop starts with checking the condition. If it is evaluated to be true, then the loop body statements are executed otherwise first statement following the loop is executed. For this reason, it is also called the Entry control loop.
- Once the condition is evaluated to be true, the statements in the loop body are executed. Normally the statements contain an update value for the variable being processed for the next iteration.
- When the condition becomes false, the loop terminates which marks the end of its life cycle.

JavaScript

Loops in JavaScript

- for loop: for loop provides a concise way of writing the loop structure. Unlike a while loop, a for statement consumes the initialization, condition, and increment/decrement in one line thereby providing a shorter, easy-to-debug structure of looping. Syntax:

- for (initialization condition; testing condition; increment/decrement)
- {
-     statement(s)
- }

JavaScript

Loops in JavaScript

- Initialization condition: Here, we initialize the variable in use. It marks the start of a for loop. An already declared variable can be used or a variable can be declared, local to loop only.
- Testing Condition: It is used for testing the exit condition for a loop. It must return a boolean value. It is also an Entry Control Loop as the condition is checked prior to the execution of the loop statements.
- Statement execution: Once the condition is evaluated to be true, the statements in the loop body are executed.
- Increment/ Decrement: It is used for updating the variable for the next iteration.
- Loop termination: When the condition becomes false, the loop terminates marking the end of

JavaScript

Loops in JavaScript

- **do-while: The do-while loop is similar to the while loop with the only difference that it checks for the condition after executing the statements, and therefore is an example of an Exit Control Loop.**

Syntax:

- **do**
- **{**
- **statements..**
- **}**

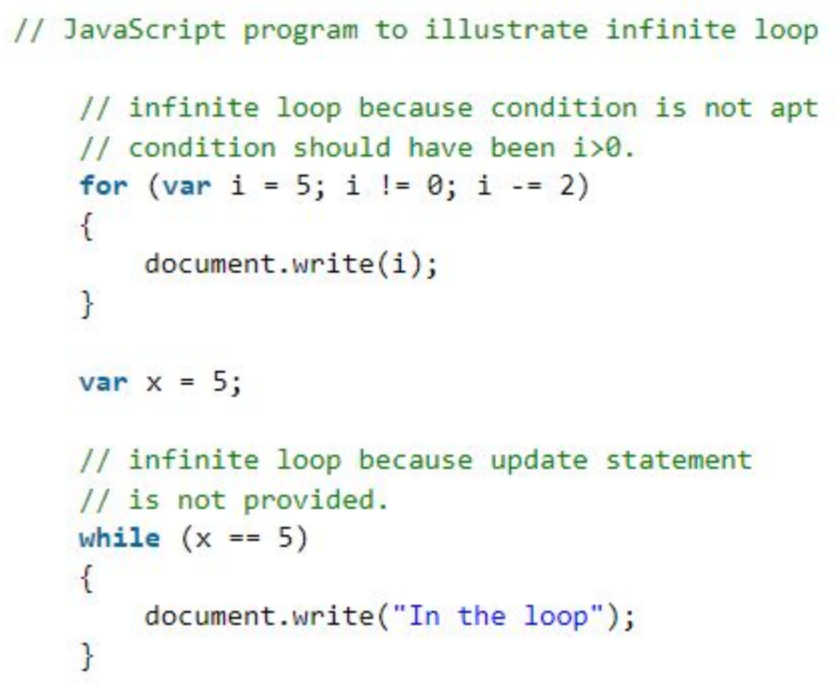
JavaScript

Loops in JavaScript

- The do-while loop starts with the execution of the statement(s). There is no checking of any condition for the first time.
- After the execution of the statements, and update of the variable value, the condition is checked for a true or false value. If it is evaluated to be true, the next iteration of the loop starts.
- When the condition becomes false, the loop terminates which marks the end of its life cycle.
- It is important to note that the do-while loop will execute its statements at least once before any condition is checked, and therefore is an example of the exit control loop.

JavaScript

Infinite Loop



```
// JavaScript program to illustrate infinite loop

    // infinite loop because condition is not apt
    // condition should have been i>0.
    for (var i = 5; i != 0; i -= 2)
    {
        document.write(i);
    }

    var x = 5;

    // infinite loop because update statement
    // is not provided.
    while (x == 5)
    {
        document.write("In the loop");
    }
```