Conditional Statements in JavaScript

- Conditional statements are used to perform different actions based on different conditions.
- A programming language uses control statements to control the flow of execution of the program based on certain conditions.
- The JavaScript if-else statement is used to execute the code whether condition is true or false.
- JavaScript includes if-else conditional statements to control the program flow, similar to other programming languages.
- Conditional statements are also known as Control statements / Decision Making statements / Conditional statements.

JavaScript has following statements that are used to check conditions: -

a) if statement
b) if-else statement
c) nested-if
d) if-else-if ladder
e) switch case

a) if Statement

- The if statement is used to execute a block of code only if the specified condition evaluates to true.
- In case the condition evaluates to a non-Boolean value, JavaScript implicitly converts its result into a Boolean value by calling the Boolean() function.

Syntax

```
if(condition) {
   // Code to be executed
}
```

Example

```
var a=20;
if(a>10){
console.log("A is greater than 20");
}
```

Example

```
var a=20;
if(a>10)
console.log("A is greater than 20");
```

Example

```
var password = prompt("Enter your password ... ");
if(password == "learntoearnlabs")
   document.write("Welcome LearntoEarn Labs")
if(password != "learntoearnlabs")
   document.write("Access Denied");
```

Example

```
var password = prompt("Enter your password ... ");
if(password == "learntoearnlabs")
{
   document.write("Welcome to LearntoEarn Labs")
}
if(password != "learntoearnlabs")
{
   document.write("Access Denied")
}
```

Example

```
var password = prompt("Enter your password ... ");
if(password == "learntoearnlabs")
document.write("Welcome LearntoEarn Labs")

if (true) // seperate block of code
document.write(" Hello ")

if(password != "learntoearnlabs")
document.write("Access Denied");
```

b) if-else Statement

- We can enhance the decision-making capabilities of your JavaScript program by providing an alternative choice through adding an else statement to the if statement.
- The if...else statement allows you to execute one block of code if the specified condition is evaluating to true and another block of code if it is evaluating to false.
- 'else' condition must be placed only once at the end. It must come after if or 'else if' statement.

Syntax

```
if(condition expression)
{
   //Execute this code..
}
else{
   //Execute this code..
}
```

Example

```
var a=20;
if(a%2==0)
{
console.log("a is even number");
}
else
{
console.log("a is odd number");
}
```

Example

```
var password = prompt("Enter your password ... ");
if(password == "learntoearnlabs")
{
   document.write("Welcome to LearntoEarn Labs")
}
else
{
   document.write("Access Denied")
}
```

Example

```
var password = prompt("Enter your password ... ");
if(password == "learntoearnlabs")
   document.write("Welcome to LearntoEarn Labs")
else
   document.write("Access Denied");
```

Example -- If ... else Shortcut

```
var pass = prompt("Enter your password ...");
var isLoggedIn = pass == "learntoearnlabs" ? document.write("Welcome")
: document.write("Access Denied");
```

Example -- If ... else Shortcut

```
var pass = prompt("Enter your password ...");
var isLoggedIn = pass == "learntoearnlabs" ?
(
   document.write("<h1>Welcome to our website</h1>"),
   document.write("<p>Hello User</p>")
)
:
(
   document.write("<h1>Access Denied</h1>"),
   document.write("<p>Bad Request</p>")
);
```

c) nested-if Statement

- We can have if statement inside another if statement. JavaScript allows us to nest, if statements within if statements
- A nested if is like the if statement that is the target of another if or else.

Syntax

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

Example

```
var i = 10;
 if (i == 10) {
    if (i < 15)
  {
     console.log("i is smaller than 15");
  }
   if (i < 12)
  {
     console.log("i is smaller than 12 too");
  }
  else
  {
     console.log("i is greater than 15");
  }
}
```

d) if-else-if ladder

- The if...else if...else a special statement that is used to combine multiple if...else statements.
- Use "else if" condition when you want to apply second level condition after if statement.

Syntax

```
if(condition1) {
   // Code to be executed if condition1 is true
} else if(condition2) {
   // Code to be executed if the condition1 is false and condition2 is true
} else {
   // Code to be executed if both condition1 and condition2 are false
}
```

Example

```
var a = 500;
var b = 1000;
if( a > b)
{
   console.log("a is greater than b");
}
else if(a < b)
{
   console.log("a is less than");
}
else if(a == b)
{
   console.log("a is equals to b");
}
```

Example

```
var x=20;
if(x==0){
   alert("zero is neither positive, nor negative")
}
else if( x>0){
   alert("Number is positive")
}
else{
   alert("Number is negative")
}
```

Example

```
var marks = prompt("Enter your marks (out of 500)");
if(marks <= 500)
{
   var percentage = marks / 500 * 100;

   if (percentage > 90) {
      document.write("Your percentage is: A+");
   }
   else if (percentage > 80) {
      document.write("Your percentage is: A");
   }
   else if (percentage > 70) {
      document.write("Your percentage is: B");
   }else if (percentage > 60) {
      document.write("Your percentage is: C");
   }else if (percentage > 50) {
      document.write("Your percentage is: D");
   }else{
      document.write("Your percentage is: E");
   }
}
else{
   document.write("Marks Out Of Range")
}
```

e) Switch-case Statement

- The switch..case statement is an alternative to the if...else if...else statement, which does almost the same thing.
- The switch...case statement tests a variable or expression against a series of values until it finds a match, and then executes the block of code corresponding to that match.
- The JavaScript switch statement is used to execute one code from multiple expressions.
- A switch statement can replace multiple if checks.
- The 'switch' has one or more 'case' blocks and an optional 'default' statement.
- In switch... case if none of the cases match with switch expression value then the default case will be executed.
- We can also use break keyword to stop the execution of switch .. case and exit from the switch.
- In switch case we can write multiple statements in a case without using curly braces { }.

Syntax

```
switch(x){
    case value1:
        // Code to be executed if x === value1
        break;
    case value2:
        // Code to be executed if x === value2
        break;
    ...
    default:
        // Code to be executed if x is different from all values
}
```

Example

```
var a = 3;
switch (a) {
    case 1:
        alert('case 1 executed');
        break;
    case 2:
        alert("case 2 executed");
        break;
   case 3:
        alert("case 3 executed");
        break;
    case 4:
        alert("case 4 executed");
        break;
    default:
        alert("default case executed");
}
```

Example

```
var d = new Date();
switch(d.getDay()) {
	case 0:
		alert("Today is Sunday.");
		break;
	case 1:
		alert("Today is Monday.");
		break;
	case 2:
		alert("Today is Tuesday.");
		break;
	case 3:
		alert("Today is Wednesday.");
		break;
	case 4:
		alert("Today is Thursday.");
		break;
	case 5:
		alert("Today is Friday.");
		break;
	case 6:
		alert("Today is Saturday.");
		break;
	default:
		alert("Day Not Match");
		break;
}
```

Example

```
var alphabet = prompt("Enter an alphabet");
switch (alphabet){
        case 'a': document.write("Vowel a");
          break;
        case 'A': document.write("Vowel A");
          break;
        case 'e': document.write("Vowel e");
          break;
        case 'E': document.write("Vowel E");
          break;
        case 'i': document.write("Vowel i");
          break;
        case 'I': document.write("Vowel I");
          break;
        case 'o': document.write("Vowel o");
          break;
        case 'O': document.write("Vowel O");
          break;
        case 'u': document.write("Vowel u");
          break;
        case 'U': document.write("Vowel U");
          break;
        default: document.write("Not a vowel");
}
```

Example

```
let a = 2 + 1;
switch (a) {
  case 4:
    console.log('Case 1');
    break;
  case 3: // grouped two cases
  case 5:
    console.log('Case 3');
    console.log('Case 5');
    break;

  default:
    alert('No Case Matched');
}
```

Looping In JavaScript

- Loops are used to execute the same block of code again and again, until a certain condition satisfies.
- The basic idea behind a loop is to automate the repetitive tasks within a program to save the time and effort.
- Loop makes the code compact.

Types of loops

There are primarily two types of loops in any programming language, and JavaScript is no exception. These are:

a) Entry Controlled Loops

- Any loop where we check the test condition before entering the loop is an entry-controlled loop.
- In these loops, the test condition determines whether the program will enter the loop or not.
- These include for, while etc.

b) Exit Controlled Loops

- Any loop where we check the test condition after the statements are executed once is an exit-controlled loop.
- In these loops, the test condition determines whether or not the program will exit the loop.
- This category includes do…while loop.

Types Of Loop Statements

JavaScript supports six different types of loops :-

a) while loop.
b) do..while loop.
c) for loop
d) for..in loop
e) for..of loop
f) foreach loop

a) while loop

A while statement in JavaScript executes until the boolean condition evaluates to true. This loop is an entry-controlled loop

Syntax

```
while(condition expression)
{
   /* code to be executed
   till the specified condition is true */
}
```

Example

```
var i =0;
while(i < 5){
   console.log(i);
   i++;
}
```

Example

```
let i = 3;
while (i) console.log(i--);
```

Example

```
var i=11;
while (i<=15)  {
console.log(i + "<br/>");
i++;
}
```

Example

```
var iterator1 = 0;
while (iterator1 < 5) {
   document.write(iterator1 + 1 + ". " + "Hello JavaScript</br>");
   iterator1++;
}
document.write("</br>Loop end");
```

b) do..while loop

- The JavaScript do while loop iterates the elements for the infinite number of times like while loop.
- The do-while loop is similar to while loop the only difference is, it evaluates condition expression after the execution of code block. So do-while loop will execute the code block at least once whether the condition is true or false.
- This is an exit controlled loop.

Syntax

```
do{
   //code to be executed
}while(condition expression);
```

Example

```
let i = 0;
do {
  console.log( i );
  i++;
} while (i < 3);
```

Example

```
var i = 1;
do {
   console.log("<p>The number is " + i + "</p>");
   i++;
}
while(i <= 5);
```

Example

```
var i =0;
do{
  console.log(i);
   i++;
} while(i > 1);
```

c) for loop

- The for loop repeats a block of code until a certain condition is met.
- The for loop is more complex, but it's also the most commonly used loop.
- The JavaScript for in loop is used to iterate the properties of an object.
- It's an entry controlled loop.

Syntax

```
for(initialization; condition; iteration) {
   // Code to be executed
}
```

Example

```
for(var i=1; i<=5; i++)
{
   console.log("<p>The number is " + i + "</p>");
}
```

Example

```
var language = ["HTML", "CSS", "JavaScript", "JQuery", "PHP"];
for(var i=0; i<language.length; i++) {
   console.log(language[i]);
}
```

Example

```
for (var iterator1 = 0; iterator1 < 5; iterator1++)
{
   document.write(iterator1 + 1 + ". " + "Hello JavaScript</br>");
}
document.write("</br>Loop ends");
```

d) for .. in loop

- The for-in loop is a special type of a loop that iterates over the properties of an object, or the elements of an array.
- This loop can iterate an Object as Well as Array.
- The reason why the developers created the for…in loop is to work with user-defined properties in an object. It is better to use a traditional for loop over Array elements with numeric indexes.

Syntax

```
for (keys in objProperties)
{
  statements
}
```

Example

```
var language = ["HTML", "CSS", "JavaScript", "JQuery", "PHP"];
for(var i in language) {
    console.log(language[i]);
}
```

Example

```
var person = {"name": "Rohit", "surname": "Singh", "age": 24};
for(var prop in person) {
    console.log(person[prop]);
}
```

Example

```
var courses = ["JavaScript", "IOT", "Python", "R", "Data Mining", "AI"];
document.write("<b>Below there is a list of IT languages</b><br/><br/>");
for(course in courses)
{
   document.write(courses[course] + " Programs</br>");
}
document.write("<br/>Loop Ends");
```

e) for .. of loop

- for-of loop allows us to iterate over arrays.

Syntax

```
for (values of objProperties)
{
  //statements
}
```

Example

```
let alphabets = ["h", "e", "l", "l", "o"];
for(let character of alphabets) {
    console.log(character);
}
```

Example

```
let word = "Hello Students";
for(let character of word) {
    console.log(character);
}
```

Example

```
var courses = ["JavaScript", "IOT", "Python", "R", "Data Mining", "AI"];
document.write("<b>Below there is a list of IT languages</b><br/><br/>");
for(course of courses)
{
   document.write(course + " Programs</br>");
}
document.write("<br/>Loop Ends");
```

f) foreach loop

- The forEach() method executes a provided function once for each array element.
- The forEach() method calls a function once for each element in an array, in order.

Syntax

```
array.forEach(function(currentValue, index, arr){}, thisValue)
```

Example

```
var array1 = ['a', 'b', 'c'];
array1.forEach(function(element) {
  console.log(element);
});
```

Example

```
const items = [1, 29, 47];
const copy = [];
items.forEach(function(item){
  copy.push(item);
});
console.log(copy);
```

Example

```
var num = [65, 44, 12, 4];
num.forEach(test)
function test(item, index, arr) {
  arr[index] = item * 10;
console.log(arr[index]);
}
```

Example

```
var num = [65, 44, 12, 4];
num.forEach(test)
function test(num, index, arr) {
  console.log(arr[index] = num)
}
```

Transfer statements / jump statements / loop control statements

- Loop control statements are certain JavaScript statements that interrupt the normal flow of the program.
- They direct the program control to a specific location in the code. Therefore, sometimes, we also call them “Jump Statements”.

JavaScript provides us with three loop control statements :-

1) Break Statement
2) Continue Statement
3) Label Statement

a) break statement in JavaScript

- The break statement terminates the current loop, switch, or label statement and transfers program control to the statement following the terminated statement.
- The break statement includes an optional label that allows the program to break out of a labeled statement.
- The break statement can also be used to jump out of a loop.

Example

```
var i = 0;
while (i < 6) {
  if (i === 3) {
    break;
  }
  i = i + 1;
}
console.log(i);
```

Example

```
for (var num = 10; num >=0 ; num--){
    if (num == 4){
          break;
    }else{
          document.write(num + "</br>");
    }
}
```

Example

```
for (var i = 1; i < 10; i++) {
    if (i % 8 == 0) {
        break;
    }
    else{
        document.write(i)
    }
}
```

b) continue in JavaScript

- The continue statement terminates execution of the current iteration in a loop.
- The continue statement breaks one iteration (in the loop), if a specified condition occurs, and continues with the next iteration in the loop.
- Continue statement skips the current matched iteration in the loop.

Example

```
var text = "";
var i = 0;
while (i < 5) {
  i++;
  if (i === 3) {
    continue;
  }
text += "<br>The number is " + i;
}
```

Example

```
var numbers = [65, 99, 34, 12, 10, 77];
for(number of numbers){
   if(number % 5 == 0){
        document.write(number + " is divisible by 5.</br>")
   }else{
      continue;
      document.write(number + " is not divisible by 5.</br>")
   }
}
```

c) label statement

- If we need to break out from multiple nested loops at once then we use label along with break & continue statements.
- A label offers an identifier to a statement that allows you to refer to it elsewhere in your program.
- The labeled statement can be used with the break or continue statements.
- A label is an identifier with a colon(:) before a loop.

Example 1

```
loop1:for (var i = 0; i < 5; i++) {
  if (i === 1) {
    console.log(i)
    continue loop1;
  }
}
```

Example 2

```
var wdays = 0;
weekday: for(var day = 1; day <= 31; day++)
{
    checkDay = day % 7;
    if (checkDay == 0)
    {
      continue weekday;
    }
    else
    {
      wdays++;
    }
}
document.write(wdays + " weekdays this month.");
```

Example 3

```
abc:for(i = 0;i < 5;i++)
{
   console.log("I Executed");
   for(j = 0;j < 5;j++)
   {
      console.log("J Executed");
      break abc;
   }
}
```

Example 4

```
var students = ["Aman", "Khush", "Reena", "Ram", "Ankita", "Anamika"];
checkName:for(var std = 0; std < students.length; std++)
{
   if (students[std] == "Ram")
   {
      document.write(students[std] + " is a good boy");
      break checkName;
   }
   else{
      document.write(students[std] + "</br>");
   }
}
```