JavaScript Display Possibilities

JavaScript can "display" data in different ways:

* Writing into an HTML element, using **innerHTML**.
* Writing into the HTML output using **document.write()**.
* Writing into an alert box, using **window.alert()**.
* Writing into the browser console, using **console.log()**.

Using innerHTML

To access an HTML element, JavaScript can use the **document.getElementById(id)** method.

The **id** attribute defines the HTML element. The **innerHTML** property defines the HTML content:

# JavaScript Global Variable

A **JavaScript global variable** is declared outside the function or declared with window object. It can be accessed from any function.

#### Declaring JavaScript global variable within function

To declare JavaScript global variables inside function, you need to use **window object**.

function m(){

window.value=100;//declaring global variable by window object

}

function n(){

alert(window.value);//accessing global variable from other function

}

Internals of global variable in JavaScript

When you declare a variable outside the function, it is added in the window object internally. You can access it through window object also.

var value=50;

function a(){

alert(window.value);//accessing global variable

}

# Javascript Data Types

JavaScript provides different **data types** to hold different types of values. There are two types of data types in JavaScript.

1. Primitive data type
2. Non-primitive (reference) data type

JavaScript is a **dynamic type language**, means you don't need to specify type of the variable because it is dynamically used by JavaScript engine. You need to use **var** here to specify the data type. It can hold any type of values such as numbers, strings etc.

For example:

1. var a=40;//holding number
2. var b="Rahul";//holding string

JavaScript primitive data types

There are five types of primitive data types in JavaScript. They are as follows:

|  |  |
| --- | --- |
| **Data Type** | **Description** |
| String | represents sequence of characters e.g. "hello" |
| Number | represents numeric values e.g. 100 |
| Boolean | represents boolean value either false or true |
| Undefined | represents undefined value |
| Null | represents null i.e. no value at all |

# JavaScript Operators

JavaScript operators are symbols that are used to perform operations on operands. For example:

var sum=10+20;

Here, + is the arithmetic operator and = is the assignment operator.

There are following types of operators in JavaScript.

1. Arithmetic Operators
2. Comparison (Relational) Operators
3. Bitwise Operators
4. Logical Operators
5. Assignment Operators
6. Special Operators

# JavaScript If-else

The **JavaScript if-else statement** is used to execute the code whether condition is true or false. There are three forms of if statement in JavaScript.

1. If Statement
2. If else statement
3. if else if statement

### JavaScript If statement

It evaluates the content only if expression is true. The signature of JavaScript if statement is given below.

if(expression){

//content to be evaluated

}

**<script>**

var a=20;

if(a**>**10){

document.write("value of a is greater than 10");

}

**</script>**

### JavaScript If...else Statement

It evaluates the content whether condition is true of false. The syntax of JavaScript if-else statement is given below.

if(expression){

//content to be evaluated if condition is true

}

else{

//content to be evaluated if condition is false

}

# JavaScript Switch

The **JavaScript switch statement** is used to execute one code from multiple expressions. It is just like else if statement that we have learned in previous page. But it is convenient than if..else..if because it can be used with numbers, characters etc.

The signature of JavaScript switch statement is given below.

switch(expression){

case value1:

 code to be executed;

 break;

case value2:

 code to be executed;

 break;

......

default:

 code to be executed if above values are not matched;

}

Let’s see the simple example of switch statement in javascript.

**<script>**

var grade='B';

var result;

switch(grade){

case 'A':

result="A Grade";

break;

case 'B':

result="B Grade";

break;

case 'C':

result="C Grade";

break;

default:

result="No Grade";

}

document.write(result);

**</script>**

# JavaScript Loops

The **JavaScript loops** are used to iterate the piece of code using for, while, do while or for-in loops. It makes the code compact. It is mostly used in array.

There are four types of loops in JavaScript.

1. for loop
2. while loop
3. do-while loop
4. for-in loop

1) JavaScript For loop

The **JavaScript for loop** *iterates the elements for the fixed number of times*. It should be used if number of iteration is known. The syntax of for loop is given below.

for (initialization; condition; increment)

{

    code to be executed

}

Let’s see the simple example of for loop in javascript.

**<script>**

for (i=1; i**<**=5; i++)

{

document.write(i + "**<br/>**")

}

**</script>**

2) JavaScript while loop

The **JavaScript while loop** *iterates the elements for the infinite number of times*. It should be used if number of iteration is not known. The syntax of while loop is given below.

while (condition)

{

    code to be executed

}

Let’s see the simple example of while loop in javascript.

**<script>**

var i=11;

while (i**<**=15)

{

document.write(i + "**<br/>**");

i++;

}

**</script>**

3) JavaScript do while loop

The **JavaScript do while loop** *iterates the elements for the infinite number of times* like while loop. But, code is *executed at least*once whether condition is true or false. The syntax of do while loop is given below.

do{

    code to be executed

}while (condition);

Let’s see the simple example of do while loop in javascript.

**<script>**

var i=21;

do{

document.write(i + "**<br/>**");

i++;

}while (i**<**=25);

**</script>**

# JavaScript Functions

**JavaScript functions** are used to perform operations. We can call JavaScript function many times to reuse the code.

#### Advantage of JavaScript function

There are mainly two advantages of JavaScript functions.

1. **Code reusability**: We can call a function several times so it save coding.
2. **Less coding**: It makes our program compact. We don’t need to write many lines of code each time to perform a common task.

## JavaScript Function Syntax

The syntax of declaring function is given below.

function functionName([arg1, arg2, ...argN]){

 //code to be executed

}

JavaScript Functions can have 0 or more arguments.

## JavaScript Function Example

Let’s see the simple example of function in JavaScript that does not has arguments.

**<script>**

function msg(){

alert("hello! this is message");

}

**</script>**

**<input** type="button" onclick="msg()" value="call function"**/>**

JavaScript Function Arguments

We can call function by passing arguments. Let’s see the example of function that has one argument.

**<script>**

function getcube(number){

alert(number\*number\*number);

}

**</script>**

**<form>**  **<input** type="button" value="click" onclick="getcube(4)"**/>**

**</form>**

Function with Return Value

We can call function that returns a value and use it in our program. Let’s see the example of function that returns value.

**<script>**

function getInfo(){

return "hello javatpoint! How r u?";

}

**</script>**

**<script>**

document.write(getInfo());

**</script>**

Creating Objects in JavaScript

There are 3 ways to create objects.

1. By object literal
2. By creating instance of Object directly (using new keyword)
3. By using an object constructor (using new keyword)

1) JavaScript Object by object literal

The syntax of creating object using object literal is given below:

1. object={property1:value1,property2:value2.....propertyN:valueN}

As you can see, property and value is separated by : (colon).

Let’s see the simple example of creating object in JavaScript.

1. **<script>**
2. emp={id:102,name:"Shyam Kumar",salary:40000}
3. document.write(emp.id+" "+emp.name+" "+emp.salary);
4. **</script>**

# JavaScript Errors - Throw and Try to Catch

The **try** statement lets you test a block of code for errors.

The **catch** statement lets you handle the error.

The **throw** statement lets you create custom errors.

The **finally** statement lets you execute code, after try and catch, regardless of the result.

## Errors Will Happen!

When executing JavaScript code, different errors can occur.

Errors can be coding errors made by the programmer, errors due to wrong input, and other unforeseeable things.

### Example

In this example we have written alert as adddlert to deliberately produce an error:

<p id="demo"></p>  
  
<script>  
try {  
    adddlert("Welcome guest!");  
}  
catch(err) {  
    document.getElementById("demo").innerHTML = err.message;  
}  
</script>

JavaScript try and catch

The **try** statement allows you to define a block of code to be tested for errors while it is being executed.

The **catch** statement allows you to define a block of code to be executed, if an error occurs in the try block.

The JavaScript statements **try** and **catch** come in pairs:

try {  
    *Block of code to try*}  
catch(err) {  
    *Block of code to handle errors*}

JavaScript Throws Errors

When an error occurs, JavaScript will normally stop and generate an error message.

The technical term for this is: JavaScript will  **throw an exception (throw an error)**.

JavaScript will actually create an **Error object** with two properties: **name** and **message**.

The throw Statement

The **throw** statement allows you to create a custom error.

Technically you can **throw an exception (throw an error)**.

The exception can be a JavaScript String, a Number, a Boolean or an Object:

throw "Too big";    // throw a text  
throw 500;          // throw a number

If you use **throw** together with **try** and **catch**, you can control program flow and generate custom error messages.

Input Validation Example

This example examines input. If the value is wrong, an exception (err) is thrown.

The exception (err) is caught by the catch statement and a custom error message is displayed:

<!DOCTYPE html>  
<html>  
<body>  
  
<p>Please input a number between 5 and 10:</p>  
  
<input id="demo" type="text">  
<button type="button" onclick="myFunction()">Test Input</button>  
<p id="message"></p>  
  
<script>  
function myFunction() {  
    var message, x;  
    message = document.getElementById("message");  
    message.innerHTML = "";  
    x = document.getElementById("demo").value;  
    try {   
        if(x == "") throw "empty";  
        if(isNaN(x)) throw "not a number";  
        x = Number(x);  
        if(x < 5) throw "too low";  
        if(x > 10) throw "too high";  
    }  
    catch(err) {  
        message.innerHTML = "Input is " + err;  
    }  
}  
</script>  
  
</body>  
</html>

## HTML Validation

The code above is just an example.

Modern browsers will often use a combination of JavaScript and built-in HTML validation, using predefined validation rules defined in HTML attributes:

<input id="demo" type="number" min="5" max="10" step="1"

You can read more about forms validation in a later chapter of this tutorial.

## The finally Statement

The **finally** statement lets you execute code, after try and catch, regardless of the result:

try {  
    Block of code to try}  
catch(err) {  
    Block of code to handle errors}   
finally {  
    Block of code to be executed regardless of the try / catch result}

### Example

function myFunction() {  
    var message, x;  
    message = document.getElementById("message");  
    message.innerHTML = "";  
    x = document.getElementById("demo").value;  
    try {   
        if(x == "") throw "is empty";  
        if(isNaN(x)) throw "is not a number";  
        x = Number(x);  
        if(x > 10) throw "is too high";  
        if(x < 5) throw "is too low";  
    }  
    catch(err) {  
        message.innerHTML = "Error: " + err + ".";  
    }  
    finally {  
        document.getElementById("demo").value = "";  
    }  
}

## The Error Object

JavaScript has a built in error object that provides error information when an error occurs.

The error object provides two useful properties: name and message.

## Error Object Properties

|  |  |
| --- | --- |
| **Property** | **Description** |
| name | Sets or returns an error name |
| message | Sets or returns an error message (a string) |

## Error Name Values

Six different values can be returned by the error name property:

|  |  |
| --- | --- |
| **Error Name** | **Description** |
| EvalError | An error has occurred in the eval() function |
| RangeError | A number "out of range" has occurred |
| ReferenceError | An illegal reference has occurred |
| SyntaxError | A syntax error has occurred |
| TypeError | A type error has occurred |
| URIError | An error in encodeURI() has occurred |

The six different values are described below.

## Eval Error

An **EvalError** indicates an error in the eval() function.

Newer versions of JavaScript does not throw any EvalError. Use SyntaxError instead.

## Range Error

A **RangeError** is thrown if you use a number that is outside the range of legal values.

For example: You cannot set the number of significant digits of a number to 500.

### Example

var num = 1;  
try {  
    num.toPrecision(500);   // A number cannot have 500 significant digits  
}  
catch(err) {  
    document.getElementById("demo").innerHTML = err.name;  
}

## Reference Error

A **ReferenceError** is thrown if you use (reference) a variable that has not been declared:

### Example

var x;  
try {  
    x = y + 1;   // y cannot be referenced (used)  
}  
catch(err) {  
    document.getElementById("demo").innerHTML = err.name;  
}