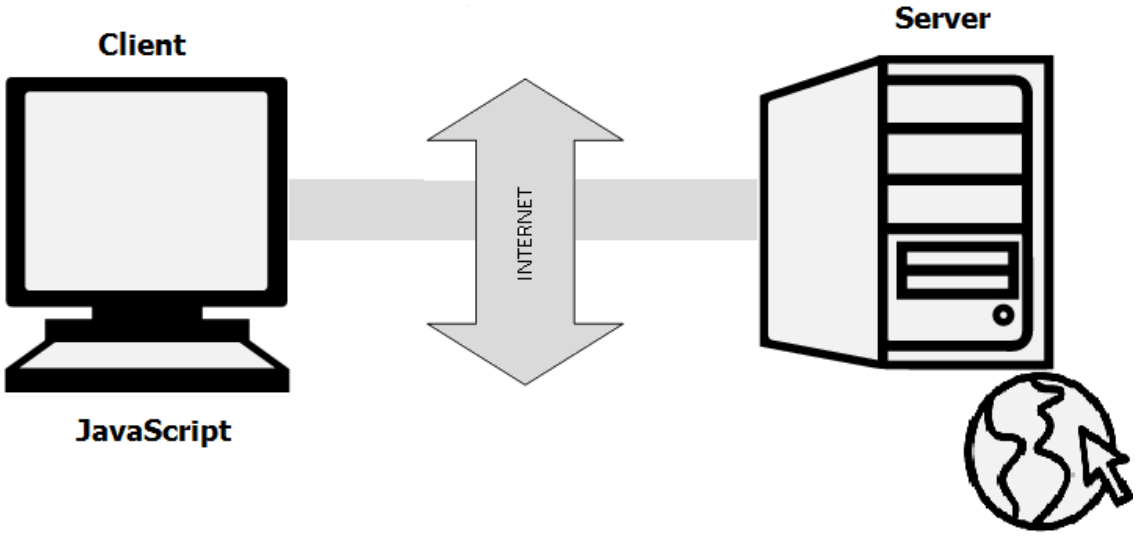
# What is JavaScript?

- JavaScript is a very powerful client-side scripting language.
- JavaScript is used mainly for enhancing the interaction of a user with the webpage.
- In other words, you can make your webpage more lively and interactive, with the help of JavaScript.
- JavaScript is also being used widely in game development and Mobile application development.



# What you can do with JavaScript?

- You can modify the content of a web page by adding or removing elements.
- You can change the style and position of the elements on a web page.
- You can monitor events like mouse click, hover, etc. and react to it.
- You can perform and control transitions and animations.
- You can create alert pop-ups to display info or warning messages to the user.
- You can perform operations based on user inputs and display the results.
- You can validate user inputs before submitting it to the server.

# Adding JavaScript to Your Web Pages

There are typically three ways to add JavaScript to a web page:

1. Embedding the JavaScript code between a pair of <script> and </script> tag.

2. Creating an external JavaScript file with the .js extension and then load it within the page through the src attribute of the <script> tag.

3. Placing the JavaScript code directly inside an HTML tag using the special tag attributes such as onclick, onmouseover, onkeypress, onload, etc.

# Adding JavaScript to Your Web Pages : 1

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Embedding JavaScript</title>
</head>
 <body>
    <script>
     var greet = "Hello World!";
    document.write(greet);                    // Prints: Hello World!
    </script>
</body>
 </html>
```

# Adding JavaScript to Your Web Pages : 2

- You can also place your JavaScript code into a separate file with .js extension, and then call that file in your document through the src attribute of the <script> tag, like this:

```
<script src="js/hello.js"></script>
```

Example:

# Adding JavaScript to Your Web Pages : 2

- .js File

```
// A function to display a message

 function sayHello()

{

        alert("Hello World!");

} // Call function on click of the button

document.getElementById("myBtn").onclick = sayHello;
```

- HTML File:

```
<body>

        <button type="button" id="myBtn">Click Me</button>

        <script src="js/hello.js">

       </script>

</body>
```

# Difference between Client-side and Server-side scripting

- Client-side scripting languages such as JavaScript, VBScript, etc. are interpreted and executed by the web browser.

- Server-side scripting languages such as PHP, ASP, Java, Python, Ruby, etc. runs on the web server and the output sent back to the web browser in HTML format.

- You can use JavaScript to check if the user has entered invalid data in form fields and show notifications for input errors.

- Response from a server-side script is slower as compared to a client-side script, because server-side scripts are processed on the remote computer not on the user's local computer.

# Understanding the JavaScripts syntax:

- The syntax of JavaScript is the set of rules that define a correctly structured JavaScript program.
- A JavaScript consists of JavaScript statements that are placed within the <script></script> HTML tags in a web page, or within the external JavaScript file having .js extension.

- Example:

```
var x = 5;
var y = 10;
 var sum = x + y;
 document.write(sum); // Prints variable value
```

# Case sensitivity in JavaScript

- JavaScript is case-sensitive.
- This means that variables, language keywords, function names, and other identifiers must always be typed with a consistent capitalization of letters.

- Example:

```
var myVar = "Hello World!";
console.log(myVar);
console.log(MyVar);
console.log(myvar);
```

# JavaScript Variables

- You can create a variable with the var keyword
- The assignment operator (=) is used to assign value to a variable

```
var varName = value;
```

Example:

```
var name = "Peter Parker";

var age = 21;

var isMarried = false;
```

# JavaScript Variables

- In JavaScript, variables can also be declared without having any initial values assigned to them.
- This is useful for variables which are supposed to hold values like user inputs.

Example:

```
// Declaring Variable
    var userName;
// Assigning value
    userName = “James";
```

# Naming Conventions for JavaScript Variables

These are the following rules for naming a JavaScript variable:

- A variable name must start with a letter, underscore (_), or dollar sign ($).
- A variable name cannot start with a number.
- A variable name can only contain alpha-numeric characters (A-z, 0-9) and underscores.
- A variable name cannot contain spaces.
- A variable name cannot be a JavaScript keyword or a JavaScript reserved word.

# JavaScript generating Output

In JavaScript there are several different ways of generating output including writing output to the browser window or browser console, displaying output in dialog boxes, writing output into an HTML element, etc.

- 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().

# JavaScript generating Output

## 1. 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:

Example:

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

<script>
document.getElementById("demo").innerHTML = 5 + 6;
</script>
```

# JavaScript generating Output

**2. Using document.write( ):**

You can use the document.write() method to write the content to the current document only while that document is being parsed.

Example:

```
// Printing a simple text message
    document.write("Hello World!");                    // Prints: Hello World!
// Printing a variable value
    var x = 10;
    var y = 20;
    var sum = x + y;
    document.write(sum);                               // Prints: 30
```

# JavaScript generating Output

**3. Using alert ( ):**

You can also use alert dialog boxes to display the message or output data to the user.

An alert dialog box is created using the alert( ) method.

Example:

```
// Displaying a simple text message
alert("Hello World!");                    // Outputs: Hello World!
// Displaying a variable value
var x = 10;
var y = 20;
var sum = x + y;
alert(sum);                               // Outputs: 30
```

# JavaScript generating Output

**3. Using console.log()**

For debugging purposes, you can call the console.log() method in the browser to display data.

Example:

```
// Printing a simple text message

console.log("Hello World!");                    // Prints: Hello World!

// Printing a variable value

var x = 10;

var y = 20;

var sum = x + y;

console.log(sum);                               // Prints: 30
```

# Data Types in JavaScript

**Data types basically specify what kind of data can be stored and manipulated within a program.**

1. String
2. Number
3. Boolean
4. Array
5. Undefined
6. Null
7. Object
8. Function

# Data Types in JavaScript

1. **String:**

- The string data type is used to represent textual data (i.e. sequences of characters).

- Strings are created using single or double quotes surrounding one or more characters

Example:

```
var a = 'Hi there!';        // using single quotes
var b = "Hi there!";        // using double quotes
```

# Data Types in JavaScript

1. **String:**

Example:

```
var a = "Let's have a cup of coffee.";          // single quote inside double  quotes
var b = 'He said "Hello" and left.';            // double quotes inside single quotes
var c = 'We\'ll never give up.';                // escaping single quote with backslash
```

# Data Types in JavaScript

**2. Number:**

The number data type is used to represent positive or negative numbers with or without decimal place, or numbers written using exponential notation.

Example:

```
var a = 25;          // integer
var b = 80.5;        // floating-point number
var c = 4.25e+6;     // exponential notation, same as 4.25e6 or 4250000
var d = 4.25e-6;     // exponential notation, same as 0.00000425
```

# Data Types in JavaScript

**3. Boolean:**

The Boolean data type can hold only two values: true or false. It is typically used to store values like yes (true) or no (false), on (true) or off (false)

Example:

```
var isReading = true;          // yes, I'm reading
var isSleeping = false;        // no, I'm not sleeping
```

OR

```
var a = 2, b = 5, c = 10;
alert(b > a)          // Output: true
alert(b > c)          // Output: false
```

# Data Types in JavaScript

**4. Undefined:**

The undefined data type can only have one value-the special value undefined.

If a variable has been declared, but has not been assigned a value, has the value undefined.

Example:

```
var a;
var b = "Hello World!"
alert(a)                    // Output: undefined
alert(b)                    // Output: Hello World!
```

# Data Types in JavaScript

**5. NULL:**

A null value means that there is no value.

It is not equivalent to an empty string ("") or 0, it is simply nothing.

A variable can be explicitly emptied of its current contents by assigning it the null value.

Example:

```
var a = null;
alert(a);                    // Output: null
var b = "Hello World!"
alert(b);                    // Output: Hello World!
b = null;
alert(b)                     // Output: null
```

# Data Types in JavaScript

**6. Object:**

The object is a complex data type that allows you to store collections of data.

An object contains properties, defined as a key-value pair.

Example:

```
var emptyObject = {};
var person = {"name": "Clark", "surname": "Kent", "age": "36"};
// For better reading
var car = {
        "modal": "BMW X3",
        "color": "white",
        "doors": 5
       }
```

# Data Types in JavaScript

**7. Array:**

The array index starts from 0, so that the first array element is arr[0] not arr[1].

The simplest way to create an array is by specifying the array elements as a comma-separated list enclosed by square brackets

Example:

```
var colors = ["Red", "Yellow", "Green", "Orange"];
var cities = ["London", "Paris", "New York"];
alert(colors[0]);                    // Output: Red
alert(cities[2]);                    // Output: New York
```

# Data Types in JavaScript

**8. Function:**

The function is callable object that executes a block of code.

Since functions are objects, so it is possible to assign them to variables

Example:

```
var greeting = function()
{
return "Hello World!";
 }
 // Check the type of greeting variable
alert(typeof greeting)              // Output: function
alert(greeting());                  // Output: Hello World!
```

# Operators in JavaScript

| Operator | Description | Example | Result |
|---|---|---|---|
| + | Addition | $x + $y | Sum of $x and $y |
| - | Subtraction | $x - $y | Difference of $x and $y. |
| * | Multiplication | $x * $y | Product of $x and $y. |
| / | Division | $x / $y | Quotient of $x and $y |
| % | Modulus | $x % $y | Remainder of $x divided by $y |

```
var x = 10;
var y = 4;
alert(x + y);        // Outputs: 14
alert(x - y);        // Outputs: 6
alert(x * y);        // Outputs: 40
alert(x / y);        // Outputs: 2.5
alert(x % y);        // Outputs: 2
```

# JavaScript Assignment operators

| Operator | Description | Example | Is The Same As |
|---|---|---|---|
| = | Assign | x = y | x = y |
| += | Add and assign | x += $ | x = x + y |
| -= | Subtract and assign | x -= y | x = x - y |
| *= | Multiply and assign | x *= y | x = x * y |
| /= | Divide and assign quotient | x /= y | x = x / y |
| %= | Divide and assign modulus | x %= y | x = x % y |

# JavaScript Assignment operators

```
var x;                          // Declaring Variable
x = 10;
alert(x);                       // Outputs: 10

x = 20; x += 30;
 alert(x);                      // Outputs: 50

x = 50; x -= 20;
alert(x);                       // Outputs: 30

x = 5; x *= 25;
alert(x);                        // Outputs: 125

x = 50; x /= 10;
alert(x);                       // Outputs: 5

 x = 100; x %= 15;
alert(x);                       // Outputs: 10
```

# JavaScript String operators

| Operator | Description | Example | Result |
|---|---|---|---|
| + | Concatenation | str1 + str2 | Concatenation of str1 and str2 |
| += | Concatenation assignment | str1 += str2 | Appends the str2 to the str1 |

```
var str1 = "Hello";
 var str2 = " World!";

alert(str1 + str2);                    // Outputs: Hello World!

str1 += str2;
alert(str1);                           // Outputs: Hello World!
```

# JavaScript Increment & Decrement operators

| Operator | Name | Effect |
| --- | --- | --- |
| `++x` | Pre-increment | Increments x by one, then returns x |
| `x++` | Post-increment | Returns x, then increments x by one |
| `--x` | Pre-decrement | Decrements x by one, then returns x |
| `x--` | Post-decrement | Returns x, then decrements x by one |

# JavaScript Logical operators

| Operator | Name | Example | Result |
|---|---|---|---|
| && | And | x && y | True if both x and y are true |
| \|\| | Or | x \|\| y | True if either x or y is true |
| ! | Not | !x | True if x is not true |

# JavaScript Comparison operators

| Operator | Name | Example | Result |
|---|---|---|---|
| == | Equal | x == y | True if x is equal to y |
| === | Identical | x === y | True if x is equal to y, and they are of the same type |
| != | Not equal | x != y | True if x is not equal to y |
| !== | Not identical | x !== y | True if x is not equal to y, or they are not of the same type |
| < | Less than | x < y | True if x is less than y |
| > | Greater than | x > y | True if x is greater than y |
| >= | Greater than or equal to | x >= y | True if x is greater than or equal to y |
| <= | Less than or equal to | x <= y | True if x is less than or equal to y |

# JavaScript Functions

- A function is a group of statements that perform specific tasks and can be kept and maintained separately from main program.

- **Functions reduces the repetition of code within a program —** Function allows you to extract commonly used block of code into a single component.

- **Functions makes the code much easier to maintain —** Since a function created once can be used many times, so any changes made inside a function automatically implemented at all the places without touching the several files.

- **Functions makes it easier to eliminate the errors —** When the program is subdivided into functions, if any error occur you know exactly what function causing the error and where to find it. Therefore, fixing errors becomes much easier.

# JavaScript Functions

**Defining and Calling a Function**

- The declaration of a function start with the function keyword, followed by the name of the function you want to create, followed by parentheses i.e. () and finally place your function's code between curly brackets {}.

```
function functionName( )
    {
      // Code to be executed
    }
```

- Once a function is defined it can be called (invoked) from anywhere in the document, by typing its name followed by a set of parentheses.

# JavaScript Functions

**Adding Parameters to Functions**

- You can specify parameters when you define your function to accept input values at run time.

- The parameters work like placeholder variables within a function; they're replaced at run time by the values (known as argument) provided to the function at the time of invocation.

```
function functionName(parameter1, parameter2, parameter3)
    {
       // Code to be executed
    }
```

- However for each parameter you specify, a corresponding argument needs to be passed to the function when it is called, otherwise its value becomes **undefined.**

# JavaScript Functions

**JavaScript Return Value**

- You can also create JS functions that return values.

- Inside the function, you need to use the keyword return followed by the value to be returned.

```
function functionname(arg1, arg2)
 {
lines of code to be executed return val1;
 }
```

# JavaScript Functions

| Method | Description |
|---|---|
| apply() | It is used to call a function contains this value and a single array of arguments. |
| bind() | It is used to create a new function. |
| call() | It is used to call a function contains this value and an argument list. |
| toString() | It returns the result in a form of a string. |

# JavaScript Functions

**Working with Function Expressions**

- Once function expression has been stored in a variable, the variable can be used as a function.

```
// Function Declaration
function getSum(num1, num2)
{
        var total = num1 + num2;
       return total;
}
 // Function Expression
var getSum = function(num1, num2)
{
 var total = num1 + num2;
 return total;
};
```

# JavaScript Functions

**Working with Functions:**

Reverse a String With Built-In Functions

- The **split( )** method splits a String object into an array of string by separating the string into sub strings.

- The **reverse( )** method reverses an array in place. The first array element becomes the last and the last becomes the first.

- The **join( )** method joins all elements of an array into a string.

# JavaScript Loops

- Loops are useful when you have to execute the same lines of code repeatedly, for a specific number of times or as long as a specific condition is true

- There are mainly four types of loops in JavaScript.
    1. for loop
    2. For - in a loop
    3. while loop
    4. do…while loop

# JavaScript Loops

**1. for loop**

- The for loop repeats a block of code as long as a certain condition is met.

- It is typically used to execute a block of code for certain number of times.

- Its syntax is:

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

# JavaScript Loops

## 1. for loop

The parameters of the for loop statement have following meanings:

- **initialization —** it is used to initialize the counter variables, and evaluated once unconditionally before the first execution of the body of the loop.

- **condition —** it is evaluated at the beginning of each iteration. If it evaluates to true, the loop statements execute. If it evaluates to false, the execution of the loop ends.

- **increment —** it updates the loop counter with a new value each time the loop runs.

# JavaScript Loops

**2. 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.

- The generic syntax of the for-in loop is:

```
for(variable in object)
    {
      // Code to be executed
    }
```

- The loop counter i.e. variable in the for-in loop is a string, not a number.

- It contains the name of current property or the index of the current array element.

# JavaScript Loops

**3. While loop**

- This is the simplest looping statement provided by JavaScript.
- The while loop loops through a block of code as long as the specified condition evaluates to true.
- As soon as the condition fails, the loop is stopped.
- The generic syntax of the while loop is:

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

# JavaScript Loops

**4. Do - While loop**

- The do-while loop is a variant of the while loop, which evaluates the condition at the end of each loop iteration.

- With a do-while loop the block of code executed once, and then the condition is evaluated, if the condition is true, the statement is repeated as long as the specified condition evaluated to is true.

- The generic syntax of the do-while loop is:

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

# JavaScript Conditional Statements

- Conditional statements are used to decide the flow of execution based on different conditions.

- If a condition is true, you can perform one action and if the condition is false, you can perform another action.

  - The if statement
  - The if...else statement
  - The if...else if....else statement
  - The switch...case statement

# JavaScript Conditional Statements

**The if statement**

- The if statement is used to execute a block of code only if the specified condition evaluates to true.
- This is the simplest JavaScript's conditional statements and can be written like:

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

# JavaScript Conditional Statements

**The if statement**

- Example:

```
var now = new Date();
var dayOfWeek = now.getDay();          // Sunday - Saturday : 0 - 6
if(dayOfWeek == 5)
{
alert("Have a nice weekend!");
 }
```

# JavaScript Conditional Statements

- **The if...else statement**
- The if...else statement allows you to execute one block of code if the specified condition is evaluates to true and another block of code if it is evaluates to false.

•It can be written, like this:

```
if(condition)
    {
      // Code to be executed if condition is true
    }
else
    {
      // Code to be executed if condition is false
    }
```

# JavaScript Conditional Statements

- **The if...else statement**
- Example:

```
var now = new Date();
var dayOfWeek = now.getDay();          // Sunday - Saturday : 0 - 6
if(dayOfWeek == 5)
{
alert("Have a nice weekend!");
}
else
 {
alert("Have a nice day!");
 }
```

# JavaScript Conditional Statements

**The if...else if....else statement**

- The if...else if...else a special statement that is used to combine multiple if...else statements.

```
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
        }
```

# JavaScript Conditional Statements

**The if...else if....else statement**

- Example:

```
var now = new Date();
var dayOfWeek = now.getDay();            // Sunday - Saturday : 0 - 6
if(dayOfWeek == 5)
{
alert("Have a nice weekend!");
 }
else if(dayOfWeek == 0)
{
alert("Have a nice Sunday!");
 }
Else
 {
alert("Have a nice day!");
 }
```

# JavaScript Conditional Statements

**The switch...case statement**

- 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.
- It's syntax is:

```
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
    }
```

# JavaScript Conditional Statements

**The switch...case statement**

- Example:

```
var d = new Date();
 switch(d.getDay())
{
case 0: alert("Today is Sunday.");
break;
case 1: alert("Today is Monday.");
break;
        .
        .
        .
        .
        .
default: alert("No information available for that day.");
break;
}
```

# JavaScript Objects

- Many times, variables or arrays are not sufficient to simulate real-life situations.
- JavaScript allows you to create objects that act like real life objects.
- You can create properties and methods to your objects to make programming easier.
- If your object is a student, it will have properties like first name, last name, id etc and methods like calculateRank, changeAddress etc.
- If your object is a home, it will have properties like a number of rooms, paint color, location etc and methods like

# JavaScript Objects

## Creating Objects:

- An object can be created with curly brackets {} with an optional list of properties.

- A property is a "key: value" pair, where the key (or property name) is always a string, and value (or property value) can be any data type, like strings, numbers, Booleans or complex data type like arrays, functions, and other objects.

```
var objName = new Object();
objName.property1 = value1;
objName.property2 = value2;
objName.method1 = function()
{
 line of code
 }
```

OR

```
var objName= {property1:value1, property2:value2, method1: function()
                    { lines of code}
              };
```

# JavaScript Objects

## Access Object Properties and Methods

- You can access properties of an object like this:

  objectname.propertyname;

- You can access methods of an object like this:

  objectname.methodname();

# JavaScript Objects

## Access Object Properties and Methods

- To access or get the value of a property, you can use the dot (.), or square bracket ([ ]) notation.

```
var book = { "name": “C Programming", "author": “Kanetkar", "year": 2000 };

// Dot notation
document.write(book.author);                    // Prints: Kanetkar

// Bracket notation
document.write(book["year"]);                   // Prints: 2000
```

# JavaScript Objects

## Looping Through Object's Properties

- You can iterate through the key-value pairs of an object using the for...in loop.
- This loop is specially optimized for iterating over object's properties.
- Example:

```
var person = { name: "Peter", age: 28, gender: "Male" };

 // Iterating over object properties
for(var i in person)
{
      document.write(person[i] + "<br>");            // Prints: name, age and gender
}
```

# JavaScript Objects

## Setting Object's Properties

```
var person = { name: "Peter", age: 28, gender: "Male" };
 // Setting a new property
person.country = "United States";
      document.write(person.country);                    // Prints: United States
person["email"] = "peterparker@mail.com";
      document.write(person.email);                      // Prints: peterparker@mail.com

// Updating existing property
 person.age = 30;
      document.write(person.age);                              // Prints: 30
person["name"] = "Peter Parker";
      document.write(person.name);                             // Prints: Peter Parker
```

# JavaScript Objects

## Deleting Object's Properties

- The delete operator can be used to completely remove properties from an object.
- Deleting is the only way to actually remove a property from an object.
- Setting the property to undefined or null only changes the value of the property.
- It does not remove property from the object.

```
var person = { name: "Peter",
               age: 28,
              gender: "Male",
              displayName: function()
              {
               alert(this.name);
              }
            };
// Deleting property
delete person.age;
alert(person.age);                    // Outputs: undefined
```

# JavaScript Objects

## Calling Object's Methods

- You can access an object's method the same way as you would access properties—using the dot notation or using the square bracket notation.

- ```
  var person = { name: "Peter",
                 age: 28,
                 gender: "Male",
                 displayName: function()
                 {
                 alert(this.name);
                 }
               };
  person.displayName( );                    // Outputs: Peter
  person["displayName"]( );                 // Outputs: Peter
  ```
  ```

# JavaScript Properties:

- Properties are the values associated with a JavaScript object.
- A JavaScript object is a collection of unordered properties.
- Properties can usually be changed, added, and deleted, but some are read only.

## Accessing JavaScript Properties

- The syntax for accessing the property of an object is:

```
objectName.property                 // person.age
    or
objectName["property"]      // person["age"]
    or
objectName[expression]      // x = "age"; person[x]
```

# Deleting Properties

The delete keyword deletes a property from an object:

Example

```
var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};
delete person.age;              // or delete person["age"];
```

- The delete keyword deletes both the value of the property and the property itself.
- After deletion, the property cannot be used before it is added back again.
- The delete operator is designed to be used on object properties. It has no effect on variables or functions.
- The delete operator should not be used on predefined JavaScript object properties. It can crash your application.

# JavaScript Date Object

- The JavaScript date object can be used to get year, month and day.
- You can display a timer on the webpage by the help of JavaScript date object.
- You can use different Date constructors to create date object.
- It provides methods to get and set day, month, year, hour, minute and seconds.

## Constructor

You can use 4 variant of Date constructor to create date object.

- Date()
- Date(milliseconds)
- Date(dateString)
- Date(year, month, day, hours, minutes, seconds, milliseconds)

# JavaScript Date Object

These methods can be used for getting information from a date object:

| Method | Description |
|---|---|
| getFullYear() | Get the **year** as a four digit number (yyyy) |
| getMonth() | Get the **month** as a number (0-11) |
| getDate() | Get the **day** as a number (1-31) |
| getHours() | Get the **hour** (0-23) |
| getMinutes() | Get the **minute** (0-59) |
| getSeconds() | Get the **second** (0-59) |
| getMilliseconds() | Get the **millisecond** (0-999) |
| getTime() | Get the time (milliseconds since January 1, 1970) |
| getDay() | Get the weekday as a number (0-6) |
| Date.now() | Get the time. ECMAScript 5. |

# JavaScript Date Object

Set Date methods are used for setting a part of a date:

| Method | Description |
|---|---|
| setDate() | Set the day as a number (1-31) |
| setFullYear() | Set the year (optionally month and day) |
| setHours() | Set the hour (0-23) |
| setMilliseconds() | Set the milliseconds (0-999) |
| setMinutes() | Set the minutes (0-59) |
| setMonth() | Set the month (0-11) |
| setSeconds() | Set the seconds (0-59) |
| setTime() | Set the time (milliseconds since January 1, 1970) |

# JavaScript Date Object

```
<html>
<body>
Current Time: <span id="txt"></span>
<script>
var today=new Date();
var h=today.getHours();
var m=today.getMinutes();
var s=today.getSeconds();
document.getElementById('txt').innerHTML=h+":"+m+":"+s;
</script>
</body>
</html>
```

# What is an Enum?

- An enum is an object which has a collection of related values stored together.

  Javascript does not support enums.

- Syntax:

  enum NameofEnum {

  value1,

  value2, ..

  }

Example:

```
enum Directions
    { North,
    South,
    East,
    West }
```

# JavaScript Callbacks

- A callback function can be defined as a function passed into another function as a parameter.

- Callbacks are mainly used to handle the asynchronous operations such as the registering event listeners, fetching or inserting some data into/from the file, and many more.

- If we want to execute a function right after the return of some other function, then callbacks can be used.

# JSON

- JSON: JavaScript Object Notation.
- JSON is a syntax for storing and exchanging data.
- JSON is text, written with JavaScript object notation.

## Exchanging Data

- When exchanging data between a browser and a server, the data can only be text.
- JSON is text, and we can convert any JavaScript object into JSON, and send JSON to the server.
- We can also convert any JSON received from the server into JavaScript objects.
- This way we can work with the data as JavaScript objects, with no complicated parsing and translations.

# Why use JSON?

- Since the JSON format is text only, it can easily be sent to and from a server, and used as a data format by any programming language.

- JavaScript has a built in function to convert a string, written in JSON format, into native JavaScript objects:

  JSON.parse( )

- So, if you receive data from a server, in JSON format, you can use it like any other JavaScript object.

# JSON Objects

- JSON objects are written inside curly braces.

- Just like in JavaScript, objects can contain multiple name/value pairs:
  {"firstName":"John", "lastName":"Doe"}

## JSON Arrays

- JSON arrays are written inside square brackets.

•Just like in JavaScript, an array can contain objects:

```
"employees":
    [
      {"firstName":"John", "lastName":"Doe"},
      {"firstName":"Anna", "lastName":"Smith"},
      {"firstName":"Peter", "lastName":"Jones"}
    ]
```

- The object "employees" is an array. It contains three objects.

- Each object is a record of a person (with a first name and a last name)

# JSON Objects

- Just like in JavaScript, objects can contain multiple name/value pairs:

```
{"firstName":"John", "lastName":"Doe"}
```

## JSON Arrays

```
"employees":
    [
      {"firstName":"John", "lastName":"Doe"},
      {"firstName":"Anna", "lastName":"Smith"},
      {"firstName":"Peter", "lastName":"Jones"}
    ]
```

# Converting a JSON Text to a JavaScript Object

- A common use of JSON is to read data from a web server, and display the data in a web page.

•First, create a JavaScript string containing JSON syntax:

```
var text = '{ "employees" : [' +
                '{ "firstName":"John" , "lastName":"Doe" },' +
                '{ "firstName":"Anna" , "lastName":"Smith" },' +
                '{ "firstName":"Peter" , "lastName":"Jones" } ]}';
```

- Then, use the JavaScript built-in function JSON.parse( ) to convert the string into a JavaScript object:

```
var obj = JSON.parse(text);
```

- Finally, use the new JavaScript object in your page.

# Converting a JSON Text to a JavaScript Object

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

<script>
var text = '{"employees":[' +
'{"firstName":"John","lastName":"Doe" },' +
'{"firstName":"Anna","lastName":"Smith" },' +
'{"firstName":"Peter","lastName":"Jones" }]}';


obj = JSON.parse(text);
document.getElementById("demo").innerHTML =
obj.employees[1].firstName + " " + obj.employees[1].lastName;
</script>
```

# JSON.parse ( )

## Parsing JSON Data in JavaScript

- When receiving data from a web server, the data is always a string.

- Parse the data with JSON.parse(), and the data becomes a JavaScript object.

Example - Parsing JSON
Imagine we received this text from a web server:.

'{ "name":"John", "age":30, "city":"New York"}'

- Use the JavaScript function JSON.parse() to convert text into a JavaScript object:

```
var obj = JSON.parse('{ "name":"John", "age":30, "city":"New York"}');
```

# JSON

## Parsing JSON Data in JavaScript

```
<script>

var txt = '{"name":"John", "age":30, "city":"New York"}‘

var obj = JSON.parse(txt);

document.getElementById("demo").innerHTML = obj.name + ", " + obj.age;

</script>
```

# Parsing Functions

- Functions are not allowed in JSON.
- If you need to include a function, write it as a string.
- You can convert it back into a function later.

Example:

```
var text = '{ "name":"John", "age":"function ( ) {return 30;}", "city":"New York"}';

var obj = JSON.parse(text);

obj.age = eval("(" + obj.age + ")");

document.getElementById("demo").innerHTML = obj.name + ", " + obj.age( );
```

# Parsing Functions

- In JSON, functions are not allowed as object values.
- The JSON.stringify() function will remove any functions from a JavaScript object, both the key and the value:
- Example

```
var obj = { name: "John", age: function ( ) {return 30;}, city: "New York"};

var myJSON = JSON.stringify(obj);

document.getElementById("demo").innerHTML = myJSON;
```

# Parsing Functions

•This can be omitted if you convert your functions into strings before running the JSON.stringify() function.

Example

```
var obj = { name: "John", age: function () {return 30;}, city: "New York"};
obj.age = obj.age.toString( );
var myJSON = JSON.stringify(obj);
document.getElementById("demo").innerHTML = myJSON;
```

# JSON.stringify( )

## Parsing JSON Data in JavaScript

- When sending data to a web server, the data has to be a string.
- Convert a JavaScript object into a string with JSON.stringify().

Imagine we have this object in JavaScript:

```
var obj = { name: "John", age: 30, city: "New York" };
```

Use the JavaScript function JSON.stringify() to convert it into a string.

```
var myJSON = JSON.stringify(obj);
```

# JSON.stringify( )

## Parsing JSON Data in JavaScript

myJSON is now a string, and ready to be sent to a server:

```
var obj = { name: "John", age: 30, city: "New York" };

var myJSON = JSON.stringify(obj);

document.getElementById("demo").innerHTML = myJSON;
```

# JSON

## Sending Data

- If you have data stored in a JavaScript object, you can convert the object into JSON, and send it to a server:

- Example

```
var myObj = {name: "John", age: 31, city: "New York"};

var myJSON = JSON.stringify(myObj);

window.location = "demo_json.php?x=" + myJSON;
```

# JSON

## Receiving Data

- If you receive data in JSON format, you can convert it into a JavaScript object:

- Example

```
var myJSON = '{"name":"John", "age":31, "city":"New York"}';
var myObj = JSON.parse(myJSON);
document.getElementById("demo").innerHTML = myObj.name;
```

# JSON

## Storing Data

- When storing data, the data has to be a certain format, and regardless of where you choose to store it, text is always one of the legal formats.

- JSON makes it possible to store JavaScript objects as text.

## Example

Storing data in local storage

```
// Storing data:
myObj = {name: "John", age: 31, city: "New York"};
myJSON = JSON.stringify(myObj);
localStorage.setItem("testJSON", myJSON);

// Retrieving data:
text = localStorage.getItem("testJSON");
obj = JSON.parse(text);
document.getElementById("demo").innerHTML = obj.name;
```

# JSON Values

- In JSON, values must be one of the following data types:

  - a string
  - a number
  - an object (JSON object)
  - an array
  - a boolean
  - Null

In JavaScript values can be all of the above, plus any other valid JavaScript expression, including:

- a function
- a date
- undefined

# JSON vs XML

- Both JSON and XML can be used to receive data from a web server.

- The following JSON and XML examples both define an employees object, with an array of 3 employees:

- JSON Example

```
{"employees":
    [
      { "firstName":"BB", "lastName":"DD" },
      { "firstName":"Aa", "lastName":"SS" },
      { "firstName":"CC", "lastName":"JJ" }
    ]
}
```

# JSON vs XML

- XML Example

```
<employees>
    <employee>
        <firstName>BB</firstName> <lastName>DD</lastName>
    </employee>
    <employee>
        <firstName>Aa</firstName> <lastName>SS</lastName>
    </employee>
    <employee>
        <firstName>CC</firstName> <lastName>JJ</lastName>
    </employee>
</employees>
```

# JSON vs XML

- XML is much more difficult to parse than JSON.
- JSON is parsed into a ready-to-use JavaScript object.
- Both JSON and XML are "self describing" (human readable)
- Both JSON and XML are hierarchical (values within values)
- Both JSON and XML can be parsed and used by lots of programming languages
- Both JSON and XML can be fetched with an XMLHttpRequest