JavaScript , JSON,AJAX

Reference: W3Schools

JS Primitive

- A **primitive value** is a value that has no properties or methods.
- A **primitive data type** is data that has a primitive value.
- **Primitive Data Types**
  - **string**
  - **number**
  - **boolean**
  - **null**
  - **undefined**

Data Types

Value	Type	Comment
"Hello"	string	"Hello" is always "Hello"
3.14	number	3.14 is always 3.14
true	boolean	true is always true
false	boolean	false is always false
null	null (object)	null is always null
undefined	undefined	undefined is always undefined

In JavaScript, almost "everything" is an object.

- Boolean if defined with “new” keyword are objects
- Number if defined with “new” keyword are objects
- String if defined with “new” keyword are objects
- Date are always objects
- Math are always objects
- Regular Expression are always objects
- Arrays are always objects
- Function are always objects

JavaScript Object

- A JavaScript object is a collection of **named values**
- var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};
- The named values, in JavaScript objects, are called **properties**.

Property	Value
firstName	John
lastName	Doe
age	50
eyeColor	blue

Sample Code

```
<!DOCTYPE html>
<html>
<body>
<p>Creating a JavaScript Object.</p>
<p id="demo"></p>
<script>
var person = {
  firstName : "John",
  lastName  : "Doe",
  age : 50,
  eyeColor  : "blue"
};
document.getElementById("demo").innerHTML = person.firstName + " " + person.lastName;
</script>
</body>
</html>
```

Object Methods

- Methods are **actions** that can be performed on objects.
- Object properties can be both primitive values, other objects, and functions.
- An **object method** is an object property containing a **function definition**.

Property	Value
firstName	John
lastName	Doe
age	50
eyeColor	blue
fullName	function() {return this.firstName + " " + this.lastName;}

Creating JavaScript Objects

- Create Object using create object literal
- Create Object using new keyword
- Create Constructor using constructor create object

Using Object literal

- var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};
- var p = {
  firstName: "John",
  lastName: "Doe",
  age: 50,
  eyeColor: "blue"
  };

Object using JavaScript Keyword “new”

- var person = new Object();
  person.firstName = "John";
  person.lastName = "Doe";
  person.age = 50;
  person.eyeColor = "blue";

JavaScript Objects are Mutable

- Objects are mutable: They are addressed by reference, not by value.
- If person is an object, the following statement will not create a copy of person
- var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"}

  var x = person;
  x.age = 10; // This will change both x.age and person.age

Accessing Object 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]
```

Example

- var person = {fname:"John", lname:"Doe", age:25};
- person.firstname + " is " + person.age + " years old.";
- person["firstname"] + " is " + person["age"] + " years old.”

```
for (x in person) {
  txt += person[x];
}
```

Adding New Properties

- You can add new properties to an existing object by simply giving it a value.
- Assume that the person object already exists - you can then give it new properties:
- person.nationality = “Pakistan";

Deleting Properties

- 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.

Calling Object Methods

- *objectName.methodName()*
- name = person.fullName();

- Person.fullName

GetterMethod

- // Create an object:
  var person = {
    firstName: "John",
    lastName : "Doe",
    language : "en",
    get lang() {
      return this.language;
    }
  };

  // Display data from the object using a getter:
  document.getElementById("demo").innerHTML = person.lang;

-

Setter

- ```
  var person = {
    firstName: "John",
    lastName : "Doe",
    language : "",
    set lang(lang) {
      this.language = lang;
    }
  };

  // Set an object property using a setter:
  person.lang = "en";

  // Display data from the object:
  document.getElementById("demo").innerHTML = person.language;
  ```
  ```

Object Types / Classes

The examples from the previous chapters are limited. They only create single objects.

Sometimes we need a "**blueprint**" for creating many objects of the same "type".

The way to create an "object type", is to use an **object constructor function**.

In the example above, `function Person()` is an object constructor function.

Objects of the same type are created by calling the constructor function with the `new` keyword:

```
var myFather = new Person("John", "Doe", 50, "blue");
var myMother = new Person("Sally", "Rally", 48, "green");
```

Constructor

- function Person(first, last, age, eye) {
  this.firstName = first;
  this.lastName = last;
  this.age = age;
  this.eyeColor = eye;
  }

New Property or Method

- Adding a Property to an Object vs Constructor
- Adding a Method to an Object vs Constructor
- We also learned that you can **not** add a new property to an existing object constructor:
-

Parent class or Prototype

All JavaScript objects inherit properties and methods from a prototype:

- `Date` objects inherit from `Date.prototype`
- `Array` objects inherit from `Array.prototype`
- `Person` objects inherit from `Person.prototype`

The `Object.prototype` is on the top of the prototype inheritance chain:

`Date` objects, `Array` objects, and `Person` objects inherit from `Object.prototype`.

Using prototype

- Prototype allows to add new properties and functions to constuctors
- function Person(first, last, age, eyecolor) {
  this.firstName = first;
  this.lastName = last;
  this.age = age;
  this.eyeColor = eyecolor;
  // this.nationalty=”English”
  }

  Person.prototype.nationality = "English";

Adding function using protoype

- function Person(first, last, age, eyecolor) {
  this.firstName = first;
  this.lastName = last;
  this.age = age;
  this.eyeColor = eyecolor;
  }

  Person.prototype.name = function() {
  return this.firstName + " " + this.lastName;
  };

Property Attributes

- Writable: If false, the value of the property can not be changed.
- Configurable: If false, any attempts to delete the property or change its attributes (Writable, Configurable, or Enumerable) will fail.
- Enumerable: If true, the property will be iterated over when a user does for (var prop in obj){} (or similar).
- Value.
- `Object.defineProperty(person, "language", {writable:true,enumerable:false});`
- `person.language=”English”`

Write Constructor function for Car

- Model
- Year
- Milage
- engineNum
- Color
- Brand
- getAllInfo()

JSON

- JSON: **J**ava**S**cript **O**bject **N**otation.
- JSON is a syntax for storing and exchanging data.
- JSON is text, written with JavaScript object notation.
-

JSON Syntax Rules

- JSON syntax is derived from JavaScript object notation syntax:
- Data is in name/value pairs
- Data is separated by commas
- Curly braces hold objects
- Square brackets hold arrays

JSON - Evaluates to JavaScript Objects

- The JSON format is almost identical to JavaScript objects.
- In JSON, *keys* must be strings, written with double quotes:
- JSON
  - { "name":"John" }

- Javacript
  - { name:"John" }

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

JSON Files

- The file type for JSON files is ".json"
- The MIME type for JSON text is "application/json"

JSON Example

- var jsonstring=’{"employees":[
  { "firstName":"John", "lastName":"Doe" },
  { "firstName":"Anna", "lastName":"Smith" },
  { "firstName":"Peter", "lastName":"Jones" }
]}’

- employeesobj=JSON.parse(jsonstring)
- arr=employeesobj.employees
- for obj in arr:
    console.log(obj.firstName)

var car={model:”audi” , year:2020}
jsonstring=JSON.stringify(car)

XML Example

```
<employees>
  <employee>
    <firstName>John</firstName> <lastName>Doe</lastName>
  </employee>
  <employee>
    <firstName>Anna</firstName> <lastName>Smith</lastName>
  </employee>
  <employee>
    <firstName>Peter</firstName> <lastName>Jones</lastName>
  </employee>
</employees>
```

JSON VS XML

- 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

JSON vs XML

- JSON doesn't use end tag
- JSON is shorter
- JSON is quicker to read and write
- JSON can use arrays
- The biggest difference is:
- XML has to be parsed with an XML parser. JSON can be parsed by a standard JavaScript function.

Why JSON

- For AJAX applications, JSON is faster and easier than XML:
  - Using XML
  - Fetch an XML document
  - Use the XML DOM to loop through the document
  - Extract values and store in variables
- Using JSON
  - Fetch a JSON string
  - JSON.Parse the JSON string

JSON.parse()

A common use of JSON is to exchange data to/from a web serve

When receiving data from a web server, the data is always a stri

Parse the data with `JSON.parse()`, and the data becomes a Java

Example - Parsing JSON

Imagine we received this text from a web server:

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

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

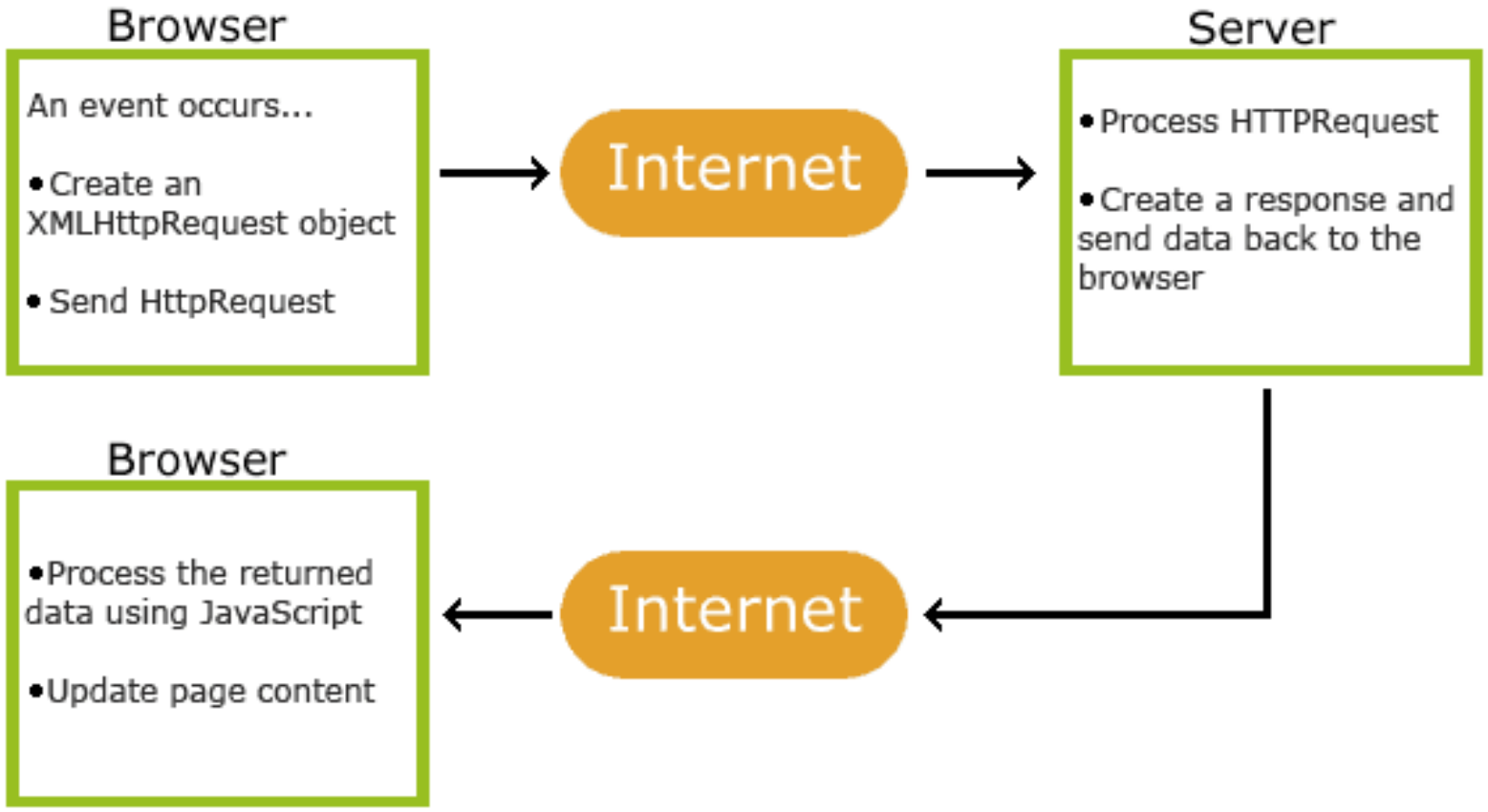
Stringify a JavaScript Object

- var obj = { name: "John", age: 30, city: "New York" };
- var myJSON = JSON.stringify(obj);
-

AJAX

- AJAX is a developer's dream, because you can:
- Read data from a web server - after the page has loaded
- Update a web page without reloading the page
- Send data to a web server - in the background
-

How AJAX Works



How AJAX Works

1. An event occurs in a web page (the page is loaded, a button is clicked)
2. An XMLHttpRequest object is created by JavaScript
3. The XMLHttpRequest object sends a request to a web server
4. The server processes the request
5. The server sends a response back to the web page
6. The response is read by JavaScript
7. Proper action (like page update) is performed by JavaScript

Create an XMLHttpRequest Object

- var xhttp = new XMLHttpRequest();
- For security reasons, modern browsers do not allow access across domains.
- This means that both the web page and the XML file it tries to load, must be located on the same server.
- The examples on W3Schools all open XML files located on the W3Schools domain.
- If you want to use the example above on one of your own web pages, the XML files you load must be located on your own server.
-

Older browser version

- if (window.XMLHttpRequest) {
  // code for modern browsers
  xmlhttp = new XMLHttpRequest();
  } else {
  // code for old IE browsers
  xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");
  }

xmlHttpRequest Methods

Method	Description
new XMLHttpRequest()	Creates a new XMLHttpRequest object
abort()	Cancels the current request
getAllResponseHeaders()	Returns header information
getResponseHeader()	Returns specific header information
open(*method, url, async, user, psw*)	Specifies the request *method*: the request type GET or POST *url*: the file location *async*: true (asynchronous) or false (synchronous) *user*: optional user name *psw*: optional password
send()	Sends the request to the server Used for GET requests
send(*string*)	Sends the request to the server. Used for POST requests
setRequestHeader()	Adds a label/value pair to the header to be sent

XMLHttpRequest Object Properties

Property	Description
onreadystatechange	Defines a function to be called when the readyState property changes
readyState	Holds the status of the XMLHttpRequest. 0: request not initialized 1: server connection established 2: request received 3: processing request 4: request finished and response is ready
responseText	Returns the response data as a string
responseXML	Returns the response data as XML data
status	Returns the status-number of a request 200: "OK" 403: "Forbidden" 404: "Not Found" For a complete list go to the Http Messages Reference
statusText	Returns the status-text (e.g. "OK" or "Not Found")