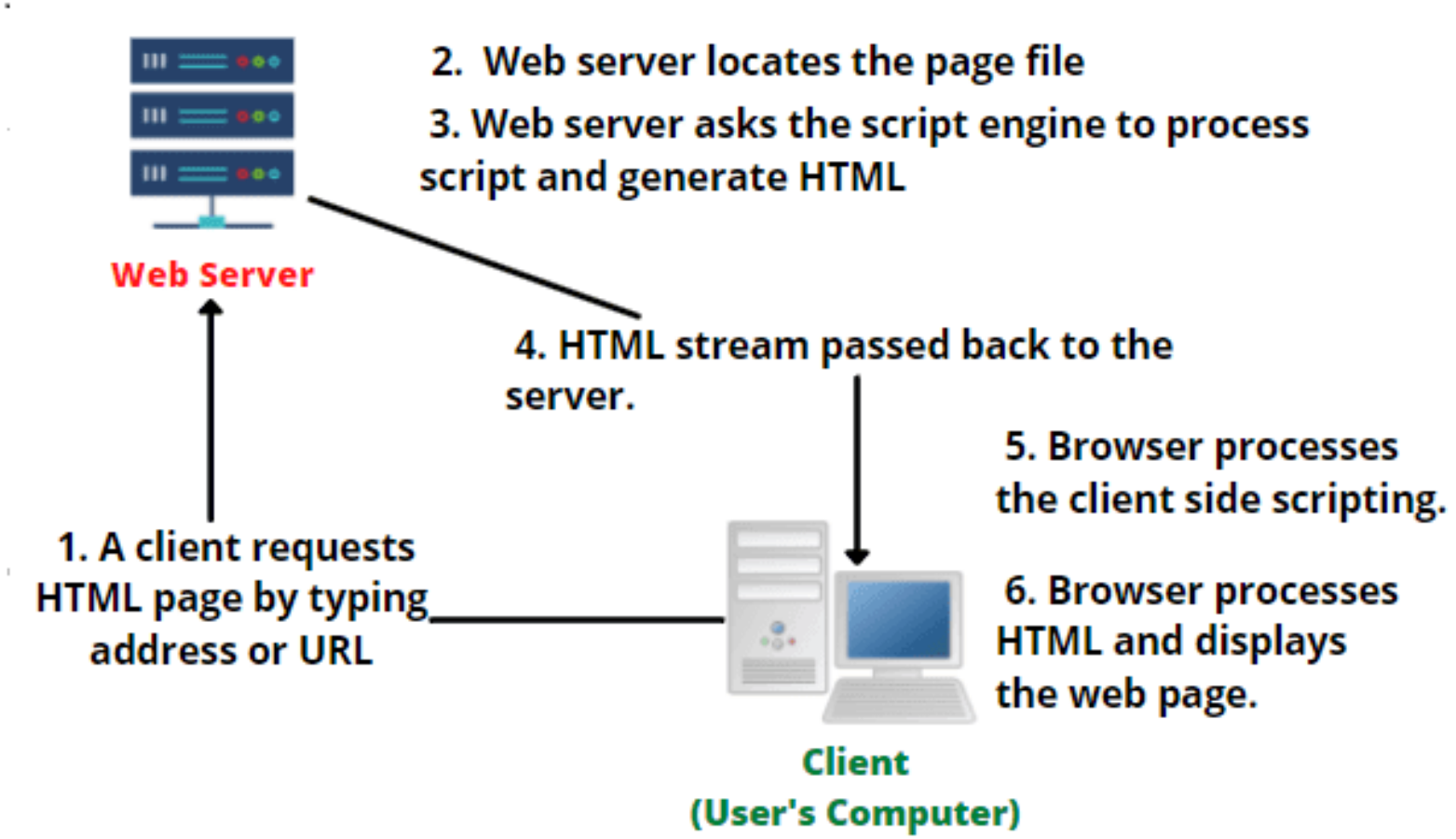
JS

VERY VERY BASIC

Client Side JavaScript



Java Script

CLIENT SIDE

Usability
- Can Modify the page without having to post back to server (Faster UI)

Efficency
- Can Make small quick changes to page without waiting for server

Event-Driven
- Can respond to user actions like clicks or key presses

SERVER SIDE

Security
- Has access to server's private data. Client cant see source code

Compatibility
- No subject to browser compatibility issues

Power
- Can write files, open connections to other servers, connect to database

JS

Primarily used for web development

Server-side development,

Mobile app development, and even

Desktop applications

JS

Interpreted and Dynamic:

- executed line by line by the browser's JavaScript engine.
- It is dynamically typed, allowing variables to hold values of any type without explicit type declarations.

Client-Side Scripting:

- used to enhance the interactivity and functionality of web pages.
- It can manipulate the Document Object Model (DOM) to update and change the content of web pages dynamically.

Multi-Paradigm:

- object-oriented programming (OOP) and
- functional programming (FP) styles. This flexibility allows developers to choose the paradigm that best fits their problem.

First-Class Functions:

- Functions are first-class citizens in JavaScript,
- they can be assigned to variables, passed as arguments to other functions, and returned as values from functions.

JS

Event-Driven and Asynchronous:

- It handles events such as user interactions (clicks, keypresses) and responds to them asynchronously.

Single-Threaded Event Loop:

- It employs an event loop to handle asynchronous operations efficiently.
- Perform non-blocking I/O operations

Cross-Browser Compatibility:

- differences in browser implementations,
- leading to the use of libraries like jQuery or modern frameworks like React or Vue.

JavaScript Interpreters

Browser-based JavaScript Interpreters:

- V8 (Chrome,Node) by Google
- SpiderMonkey (Firefox)
- JavaScriptCore (Safari)
- Duktape (Embedded JavaScript Interpreters) for IOT devices
- JavaScriptCore (IOS)

Helping Material

https://www.w3schools.com/js/default.asp

Use it as a reference Guide

Where to write Js

Script Tag

```
<script>
document.getElementById("demo").innerHTML = "My First JavaScript";
</script>
```

Or Make a separate Js file and attach like below

```
<script src="/js/myScript1.js"></script>
```

JS Syntax

```
var age = 25; //global. Variable attached to window object

let name = 'John’; //local scope

const PI = 3.14;
```

JS Variables

```
var num = 42;          // Number
var text = 'Hello';    // String
var flag = true;       // Boolean
var person = {         // Object
    name: 'John',
    age: 30
};
```

JS Variables

Number

Boolean

String

Array

Object

Function

Null

Undefined

JS Types

Number

- var age = 25;
- var pi = 3.14;
- No int or double just numbers
- Auto-Convert Types 2 * ”6” = 12

JS Types String

var s = "Connie Client";

var fName = s.substring(0, .index0f(" ")); // "Connie"

var len = s.length; // 13

Js Methods:

- charAt, charCodeAt, fromCharCode, indexOf, lastIndex0f, replace, split, substring, toLowerCase, toUpperCase
  - charAt returns a one-letter String (there is no char type)

Length property (not a method as in Java)

Strings can be specified with "" or ‘’

Concatenation with + : **1 +1 is 2**, but **“l" +1** is"11"

JS Types String

Escape sequences behave as in Java: \I \" \& \n \t \\

Converting between numbers and Strings:

- var count = 10;
- var s1 = "" + count; // "10"
- var s2 = count + " bananas!"; // "10 bananas!"
- var n1 = parseInt("42 is the answer"); // 42
- var n2 = parseFloat("booyah"); // NaN

Accessing Letters of a String

- var firstLetter = s[0]; // fails in IE
- var firstLetter = s.charAt(0); // does work in IE
- var lastLetter = s.charAt(s.length - 1);

JS Boolean

```
var iLike190M = true;
var ieIsGood = "IE6" > 0; // false
if ("web dev is great") { /* true */ }
if  (0)  { /* false */  }
```

Any value can be used as a Boolean

- "Falsey" values: 0, 0.0, NaN, "", null, and undefined
- "Truthy" values: anything else

Converting a value into a Boolean explicitly:

- var boolValue = Boolean(otherValue);
- var boolValue = 1!(otherValue);

Special Values null, NaN, undefined

var ned = null;

var benson = 9;

- // at this point in the code,
- // ned is null // benson is 9
- // caroline is undefined

NaN: not a number (only returned by the isNaN() function)

undefined : has not been declared, does not exist

null : exists, but was specifically assigned an null value

Why does JavaScript have both of these?

Math Object

```
console.log(Math.PI); // Outputs: 3.141592653589793

console.log(Math.E); // Outputs: 2.718281828459045

console.log(Math.abs(-5)); // Outputs: 5

console.log(Math.ceil(4.2));   // Outputs: 5

console.log(Math.floor(4.9));  // Outputs: 4

console.log(Math.round(4.5));  // Outputs: 5

console.log(Math.max(10, 5, 8)); // Outputs: 10

console.log(Math.min(10, 5, 8)); // Outputs: 5
```

```
console.log(Math.pow(2, 3));    // Outputs: 8

console.log(Math.sqrt(25));      // Outputs: 5

console.log(Math.exp(2));        // Outputs: 7.389056098930649

console.log(Math.log(Math.E));   // Outputs: 1

console.log(Math.sin(Math.PI / 2)); // Outputs: 1 (sine of 90 degrees)

console.log(Math.cos(Math.PI));     // Outputs: -1 (cosine of 180 degrees)

console.log(Math.tan(0));           // Outputs: 0 (tangent of 0 radians)
```

Logical Operators

> < >= <= && || !== != === ==

Most Logical Operators automatically convert types

- 5 < “7” true
- 42==42.0 true
- “5.0” == 5 true

=== and !== are strict equality checks: tests types and values

- “5.0”===5 is false

Control Flow

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

for (var i = 0; i < 5; i++) {
   // code to be repeated in a loop
}
```

JS Arrays

```
var fruits = ['apple', 'orange', 'banana'];
console.log(fruits[0]); // Outputs: apple
console.log(fruits[1]); // Outputs: orange
fruits[1] = 'grape';
var numbers = new Array(1, 2, 3, 4, 5);
fruits.push('kiwi');
fruits.unshift('pineapple');
fruits.pop();
fruits.splice(1, 2); // Removes 2 elements starting from index 1
var moreFruits = ['grapefruit', 'mango'];
var combined = fruits.concat(moreFruits);
```

JS Arrays

```
var slicedFruits = fruits.slice(1, 3); // Extracts elements from index 1 to 2
var index = fruits.indexOf('banana');
var fruitsString = fruits.join(', '); // Joins with commas
fruits.forEach(function(fruit) {
    console.log(fruit);
});
var uppercasedFruits = fruits.map(function(fruit) {
    return fruit.toUpperCase();
}); // make a new array
```

JS Types

Function

```
function greet(name) {
  return 'Hello, ' + name + '!';
}
```

RegExp

```
var pattern = /ab+c/;
```

Functions as Variables

```
// Assigning the function to a variable
var myGreetFunction = function greet(name) {
  return 'Hello, ' + name + '!';
};
// Using the variable to call the function
var greeting = myGreetFunction('John');
// Output the result
console.log(greeting); // Outputs: Hello, John!
```

Js Output

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().

First use console.log() and alert for beginners

Statements

x stores the value 5

y stores the value 6

z stores the value 11

```
var x, y, z;        // Statement 1
x = 5;              // Statement 2
y = 6;              // Statement 3
z = x + y;          // Statement 4
```

Comments

```
var x = 5;    // I will be executed

// var x = 6;    I will NOT be executed
```

Case Sensitive

```
var lastname, lastName;
lastName = "Doe";
lastname = "Peterson";
```

Much Like Algebra

price1, price2, and total, are variables:

```
var price1 = 5;
var price2 = 6;
var total = price1 + price2;
```

Operators

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
**	Exponentiation (ES2016)
/	Division
%	Modulus (Division Remainder)
++	Increment
--	Decrement

Assignment Operators

Operator	Example	Same As
=	x = y	x = y
+=	x += y	x = x + y
-=	x -= y	x = x - y
*=	x *= y	x = x * y
/=	x /= y	x = x / y
%=	x %= y	x = x % y
**=	x **= y	x = x ** y

String Operators

```
var txt1 = "John";
var txt2 = "Doe";
var txt3 = txt1 + " " + txt2;
```

String And Numbers

10
55
Hello5

```
var x = 5 + 5;
var y = "5" + 5;
var z = "Hello" + 5;
```

Comparison Operators

Operator	Description
==	equal to
===	equal value and equal type
!=	not equal
!==	not equal value or not equal type
>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to
?	ternary operator

Logical Operators

Operator	Description
&&	logical and
||	logical or
!	logical not

Undefined

Undefined

In JavaScript, a variable without a value, has the value undefined. The type is also undefined.

Null

In JavaScript null is "nothing". It is supposed to be something that doesn't exist.

```
typeof undefined              // undefined
typeof null                   // object

null === undefined            // false
null == undefined             // true
```

Premitive Data

```
typeof "John"      // Returns "string"
typeof 3.14        // Returns "number"
typeof true        // Returns "boolean"
typeof false       // Returns "boolean"
typeof x           // Returns "undefined" (if
                   //x has no value)
```

Objects Properties and Objects

	car.name = Fiat car.model = 500 car.weight = 850kg car.color = white	car.start() car.drive() car.brake() car.stop()

Defining an Object

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

Property	Property Value
firstName	John
lastName	Doe
age	50
eyeColor	blue

More Advance Usage

```
var person = {
  firstName: "John",
  lastName : "Doe",
  id       : 5566,
  fullName : function() {
    return this.firstName + " " + this.lastName;
  }
};
```

The **this** Keyword

In a function definition, this refers to the "owner" of the function.

In the example above, this is the person object that "owns" the fullName function.

In other words, this.firstName means the firstName property of this object.

JavaScript Events

HTML events are **"things"** that happen to HTML elements.

When JavaScript is used in HTML pages, JavaScript can **"react"** on these events.

Examples

- An HTML web page has finished loading
- An HTML input field was changed
- An HTML button was clicked

Binding Events

```
<button onclick=“alert(‘you clicked me’)">
   Click Me
</button>
```

Common HTML Events

Event	Description
onchange	An HTML element has been changed
onclick	The user clicks an HTML element
onmouseover	The user moves the mouse over an HTML element
onmouseout	The user moves the mouse away from an HTML element
onkeydown	The user pushes a keyboard key
onload	The browser has finished loading the page

Array.map()

This example multiplies each array value by 2 and create a new value

```
var numbers1 = [45, 4, 9, 16, 25];
var numbers2 = numbers1.map(myFunction);
function myFunction(value) {
  return value * 2;
}
```

Array.filter()

The filter() method creates a new array with array elements that passes a test.

```
var numbers = [45, 4, 9, 16, 25];
var over18 = numbers.filter(myFunction);
function myFunction(value, index, array) {
  return value > 18;
}
```