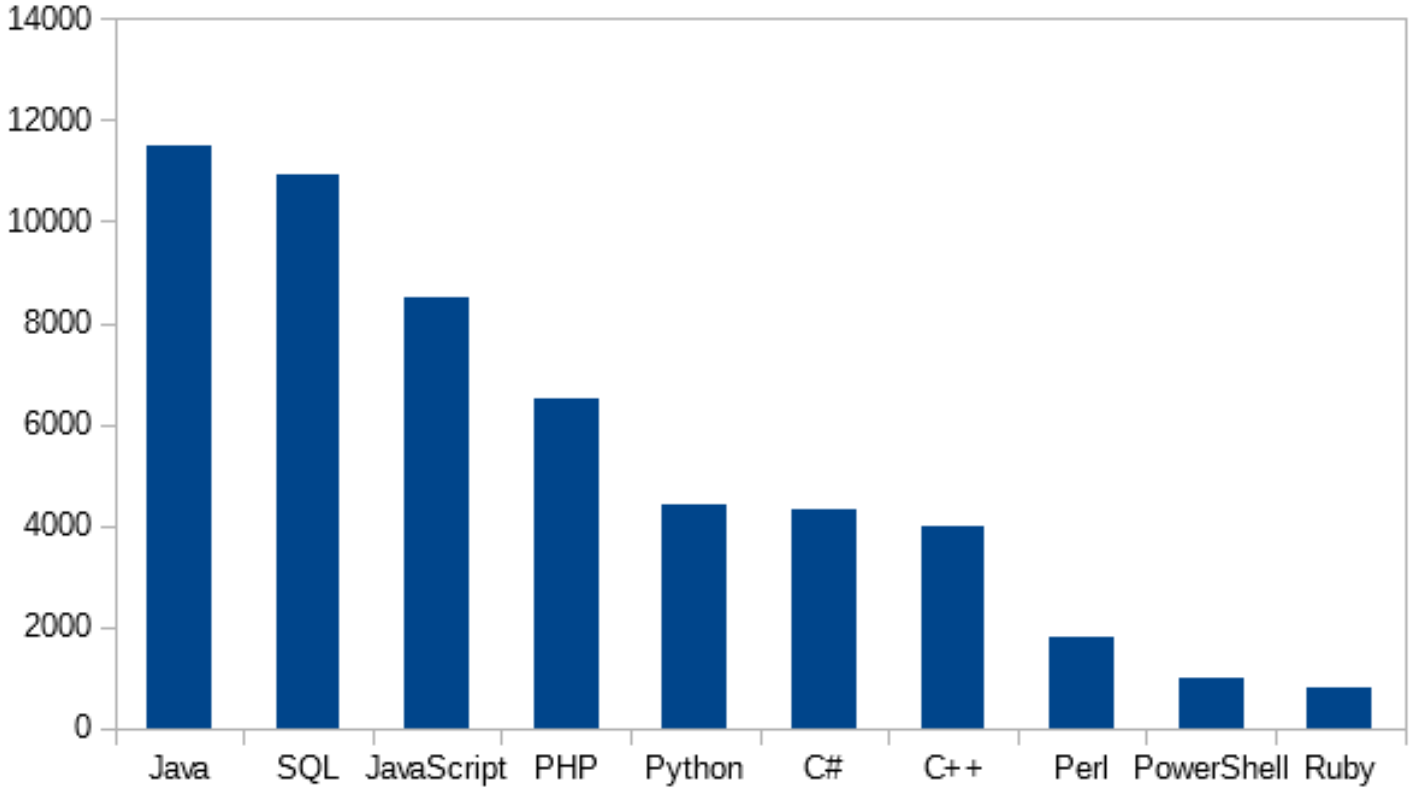
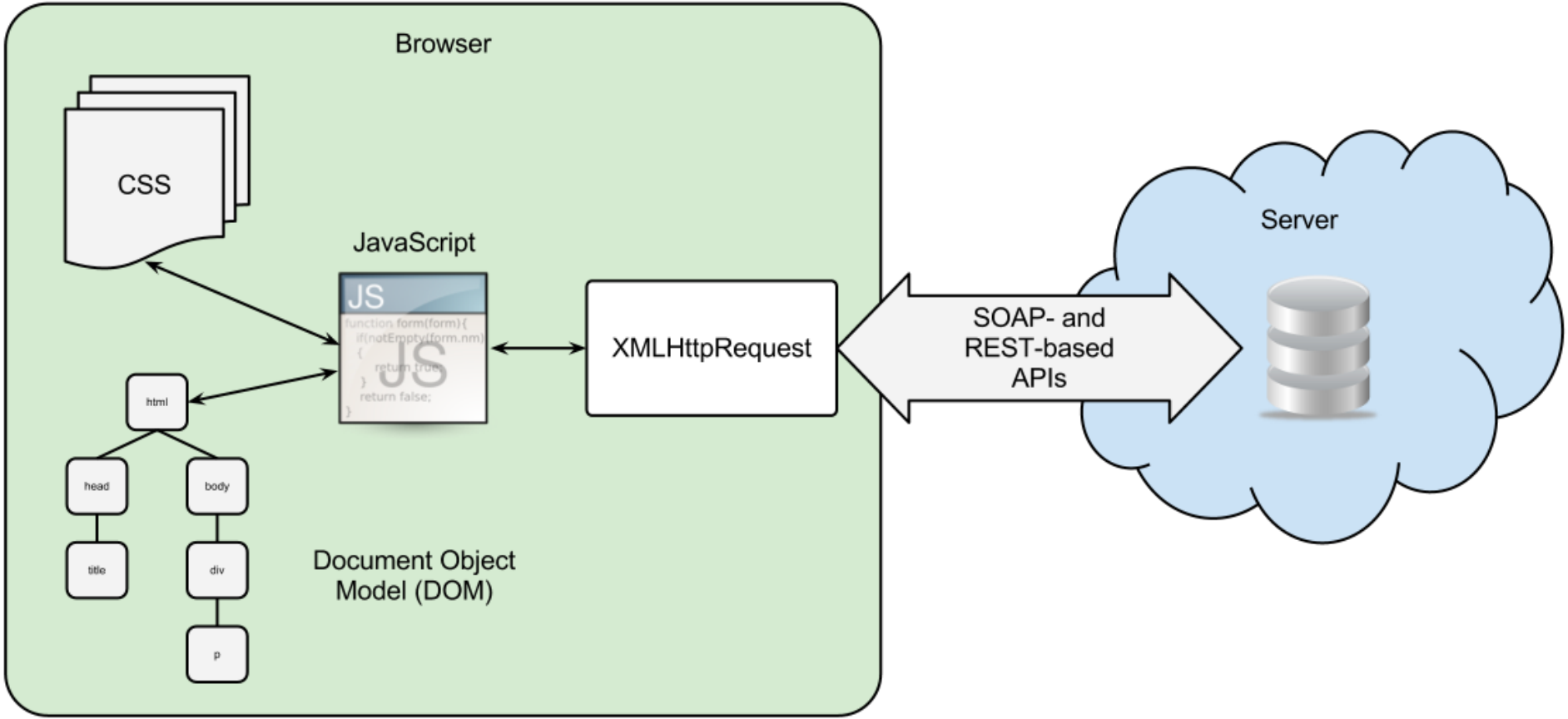
JavaScript


What is the world's most popular language?
A: French
B: English
C: Spanish
D: JavaScript


14000
12000
10000
8000
6000
4000
2000
0
Java
SQL
JavaScript
PHP
Python
C#
C++
Perl
PowerShell
Ruby

Javascript

- Javascript is used by millions of applications in the universe for developing rich and interactive web-applications
- Javascript is supported in all the major Browsers like Chrome, IE, Mozilla, safari etc
- Javascript is client-side programming language


Browser
CSS
JavaScript
JS
XMLHttpRequest
SOAP- and
REST-based
APIs
Server
html
head
body
title
div
p
Document Object
Model (DOM)

What is Javascript?

- JavaScript was designed to add interactivity to HTML pages
- JavaScript is a scripting language (a scripting language is a lightweight programming language)
- A JavaScript consists of lines of executable computer code
- A JavaScript is usually embedded directly into HTML pages
- JavaScript is an interpreted language (means that scripts execute without preliminary compilation)
- Everyone can use JavaScript without purchasing a license

Is Java and JavaSript Same?

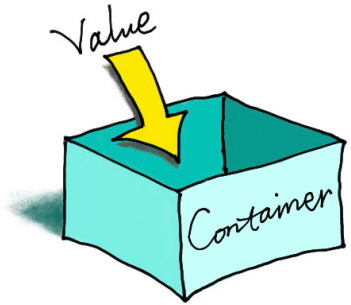
- No
- Java is a powerful programming language (Mostly used for Server Side)
- Javascript is a client-side scripting language
- Java was from Sun Microsystem (now its Oracle)
- Javascript was from Netscape

JavaScript in HTML

- How do we embed Javascript in HTML

```
<html>
<body>
<script type="text/javascript">
      document.write("Hello Students!!!")
</script>
</body>
</html>
```

Variables in Javascript



- Variables are used for storing the data in javascript
- All JavaScript **variables** must be **identified** with **unique names**.
- These unique names are called **identifiers**.
- Identifiers can be short names (like x and y) or more descriptive names (age, sum, totalVolume).

Identifiers Rules

- Names can contain letters, digits, underscores, and dollar signs.
- Names must begin with a letter
- Names can also begin with $ and _ (but we will not use it in this tutorial)
- Names are case sensitive (y and Y are different variables)
- Reserved words (like JavaScript keywords) cannot be used as names

JavaScript identifiers are case-sensitive.

Writing 1st Function in JS

```
function myFirstFunction(){
    var x = 10;
    var y = 20;
    var result = 10 * 20;
    document.write("Result is "  + result);
}
```

function keyword

Name of the function

Variables

Write the result to HTML

Operators

- Other than +, -, %, / operators

Operator	Description	Example
==	is equal to	5==8 returns false
===	is equal to (checks for both value and type)	x=5 y="5" x==y returns true x===y returns false
!=	is not equal	5!=8 returns true
>	is greater than	5>8 returns false
<	is less than	5<8 returns true
>=	is greater than or equal to	5>=8 returns false
<=	is less than or equal to	5<=8 returns true

Fun Fact

Anonymous function

```
function() {

}
```

-- A function without a name is called an anonymous function!

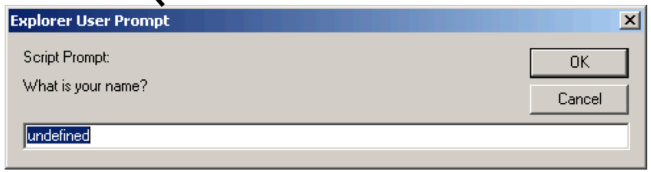
BUT the above code won't work... yet

JS Popups

alert(), confirm(), and prompt()

```
var answer = confirm("Are you OK?");
answer = true/false (if ok/cancel)
```

```
<script type="text/javascript">
alert("This is an Alert method");
confirm("Are you OK?");
prompt("What is your name?");
prompt("How old are you?","20");
</script>
```



Data Types

- Primitive data types
  - Number: integer & floating-point numbers
  - Boolean: true or false
  - String: a sequence of alphanumeric characters

- Composite data types (or Complex data types)
  - Object: a named collection of data
  - Array: a sequence of values (an array is actually a predefined object)

- Special data types
  - Null: the only value is "null" – to represent nothing.
  - Undefined: the only value is "undefined" – to represent the value of an unintialized variable

Arrays

- An array is represented by the **`Array`** object. To create an array of N elements, you can write

```
var myArray = new Array(N);
```

- Index of array runs from 0 to N-1.

- Can store values of different types

- Property "**`length`**" tells the # of elements in the array.

- Consists of various methods to manipulate its elements. e.g., **`reverse()`**, **`push()`**, **`concat()`**, etc

Arrays Example

```
var Car = new Array(3);
Car[0] = "Ford";
Car[1] = "Toyota";
Car[2] = "Honda";

// Create an array of three elements with initial
// values
var Car2 = new Array("Ford", "Toyota", "Honda");

// Create an array of three elements with initial
// values
var Car3 = ["Ford", "Toyota", "Honda"];
```

Logical Operators

- `!` – Logical NOT

- `&&` – Logical AND
  - `OP1 && OP2`
  - If OP1 is true, expression evaluates to the value of OP2.
    Otherwise the expression evaluates to the value of OP1.
  - Results may not be a boolean value.

- `||` – Logical OR
  - `OP1 || OP2`
  - If OP1 is true, expression evaluates to the value of OP1. Otherwise the expression evaluates to the value of OP2.

Conditional Statements

- “if” statement
- “if … else” statement
- "? :" ternary conditional statement
- “switch” statement

- The syntax of these statements are similar to those found in C and Java.

Looping Statement

- “for” Loops
- “for/in” Loops [for(variable in object)]
- “while” Loops
- “do … while” Loops
- “break” statement
- “continue” statement

- All except "for/in" loop statements have the same syntax as those found in C and Java.
- Example for for/in for(var in arr) {}// var will have the index of arr

Let’s try out Examples

- Function examples
- Variable arguments example (using arguments reserved keyword)

Built-In Functions

- **`eval(expr)`**
  - evaluates an expression or statement
    - eval("3 + 4"); // Returns 7 (Number)
    - eval("alert('Hello')"); // Calls the function alert('Hello')

- **`isFinite(x)`**
  - Determines if a number is finite

- **`isNaN(x)`**
  - Determines whether a value is “Not a Number”

Built-In Functions

- **parseInt(s)**
- **parseInt(s, radix) (radix says binary, octal, hexdecimals – 2,8,16)**
  - Converts string literals to integers
  - Parses up to any character that is not part of a valid integer

```
parseInt("3 chances")                    // returns 3
parseInt("   5 alive")                   // returns 5
parseInt("How are you")          // returns NaN
parseInt("17", 8)                        // returns 15
```

- **parseFloat(s)**
  - Finds a floating-point value at the beginning of a string.

```
parseFloat("3e-1 xyz")                   // returns 0.3
parseFloat("13.5 abc")                   // returns 13.5
```

Creating objects using new Object()

```
var person = new Object();

// Assign fields to object "person"
person.firstName = "John";
person.lastName = "Doe";

// Assign a method to object "person"
person.sayHi = function() {
  alert("Hi! " + this.firstName + " " + this.lastName);
}

person.sayHi();  // Call the method in "person"
```

Creating objects using Literal Notation

```
var person = {
  // Declare fields
  // (Note: Use comma to separate fields)
  firstName : "John",
  lastName : "Doe",

  // Assign a method to object "person"
  sayHi : function() {
    alert("Hi! " + this.firstName + " " +
          this.lastName);
  }
}
person.sayHi();  // Call the method in "person"
```

Object Constructor and prototyping

```
function Person(fname, lname) {
  // Define and initialize fields
  this.firstName = fname;
  this.lastName = lname;

  // Define a method
  this.sayHi = function() {
    alert("Hi! " + this.firstName + " " +
          this.lastName);
  }
}

var p1 = new Person("John", "Doe");
var p2 = new Person("Jane", "Dow");

p1.sayHi();  // Show "Hi! John Doe"
p2.sayHi();  // Show "Hi! Jane Dow"
```

Object Constructor and prototyping

```
// Suppose we have defined the constructor "Person"
// (as in the previous slide).

var p1 = new Person("John", "Doe");
var p2 = new Person("Jane", "Dow");

// Aattaching a new method to all instances of Person
Person.prototype.sayHello = function() {
  alert("Hello! " + this.firstName + " " +
                    this.lastName);
}

// We can also introduce new fields via "prototype"

p1.sayHello();  // Show "Hello! John Doe"
p2.sayHello();  // Show "Hello! Jane Dow"
```

Events

- An event occurs as a result of some activity
  - e.g.:
    - A user clicks on a link in a page
    - Page finished loaded
    - Mouse cursor enter an area
    - A preset amount of time elapses
    - A form is being submitted

Event Handlers

- Event Handler – a segment of codes (usually a function) to be executed when an event occurs

- We can specify event handlers as attributes in the HTML tags.

- The attribute names typically take the form "**onXXX**" where **XXX** is the event name.
  - e.g.:

```
<a href="…"
onClick="alert('Bye')">Other Website</a>
```

Event Handlers

Event Handlers	Triggered when
onChange	The value of the text field, textarea, or a drop down list is modified
onClick	A link, an image or a form element is clicked once
onDblClick	The element is double-clicked
onMouseDown	The user presses the mouse button
onLoad	A document or an image is loaded
onSubmit	A user submits a form
onReset	The form is reset
onUnLoad	The user closes a document or a frame
onResize	A form is resized by the user

Built-In JavaScript Objects

Objectx	Description
Array	Creates new array objects
Boolean	Creates new Boolean objects
Date	Retrieves and manipulates dates and times
Error	Returns run-time error information
Function	Creates new function objects
Math	Contains methods and properties for performing mathematical calculations
Number	Contains methods and properties for manipulating numbers.
String	Contains methods and properties for manipulating text strings

Javascript Exception Handling

- Javascript uses try/catch/finally similarly to Java

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

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

Try..Catch..Finally

```
try {
  // Contains normal codes that might throw an exception.

  // If an exception is thrown, immediately go to
  //   catch block.

} catch ( errorVariable ) {
  // Codes here get executed if an exception is thrown
  //   in the try block.

  // The errorVariable is an Error object.

} finally {
  // Executed after the catch or try block finish

  // Codes in finally block are always executed
}
// One or both of catch and finally blocks must accompany the try block.
```

AJAX

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

Understanding AJAX

```
<script type="text/javascript">
function loadXMLDoc() {
    var xmlhttp = new XMLHttpRequest();

    xmlhttp.onreadystatechange = function() {
        if (xmlhttp.readyState == XMLHttpRequest.DONE) {   // XMLHttpRequest.DONE == 4
           if (xmlhttp.status == 200) {
               document.getElementById("myDiv").innerHTML = xmlhttp.responseText;
           }
           else if (xmlhttp.status == 400) {
              alert('There was an error 400');
           }
           else {
               alert('something else other than 200 was returned');
           }
        }
    };

    xmlhttp.open("GET", "ajax_info.txt", true);
    xmlhttp.send();
}
</script>
```

Function with name

Create XMLHttpRequest (for Chrome/Mozilla) / ActiveXObject for IE

Callback when the call is sucess

Status – 200 means the response is success

Status – 400 means that there some error

Open the connection and fetch the file

Quiz

- What is radix used for in parseInt?
- Is onClick a valid event?
- Prototyping is used for extending the classes and creating new one? True/False?
- Is Java and Javascript same?
- What's the best practice in including the Javascript? Inline/External file?
- Abbreviation of AJAX?
- What object is used for Ajax in Chrome/Mozilla and IE?


Finally
done