

Java Script

Java Script

2

S
U
J
A
T
A

B
A
T
R
A



Brenden Eich

Introduction

Java Script

4

S
U
J
A
T
A

B
A
T
R
A

- ▶ A scripting language Created by **Brendan Eich** at **Netscape**
- ▶ Developed under the name **Mocha**, and officially called **LiveScript**
- ▶ Renamed to JavaScript
 - ▶ When Netscape added support for Java technology in its Netscape Navigator web browser.
- ▶ Its official name is **ECMAScript**,
- ▶ It's **interpreted** by the browser engine while the page is loaded.
 - ▶ **HTML**, gives **structure** to the web pages
 - ▶ **CSS**, controls the **appearance** of web pages
 - ▶ **JavaScript** gives the **behavior** for the page.

Scripts vs. programs

5

S
U
J
A
T
A

B
A
T
R
A

- ▶ A scripting language is a simple, interpreted programming language
 - ▶ scripts are embedded as plain text, interpreted by application
 - ▶ **simpler execution model:** don't need compiler or development environment
 - ▶ **saves bandwidth:** source code is downloaded, not compiled executable
 - ▶ **platform-independence:** code interpreted by any script-enabled browser

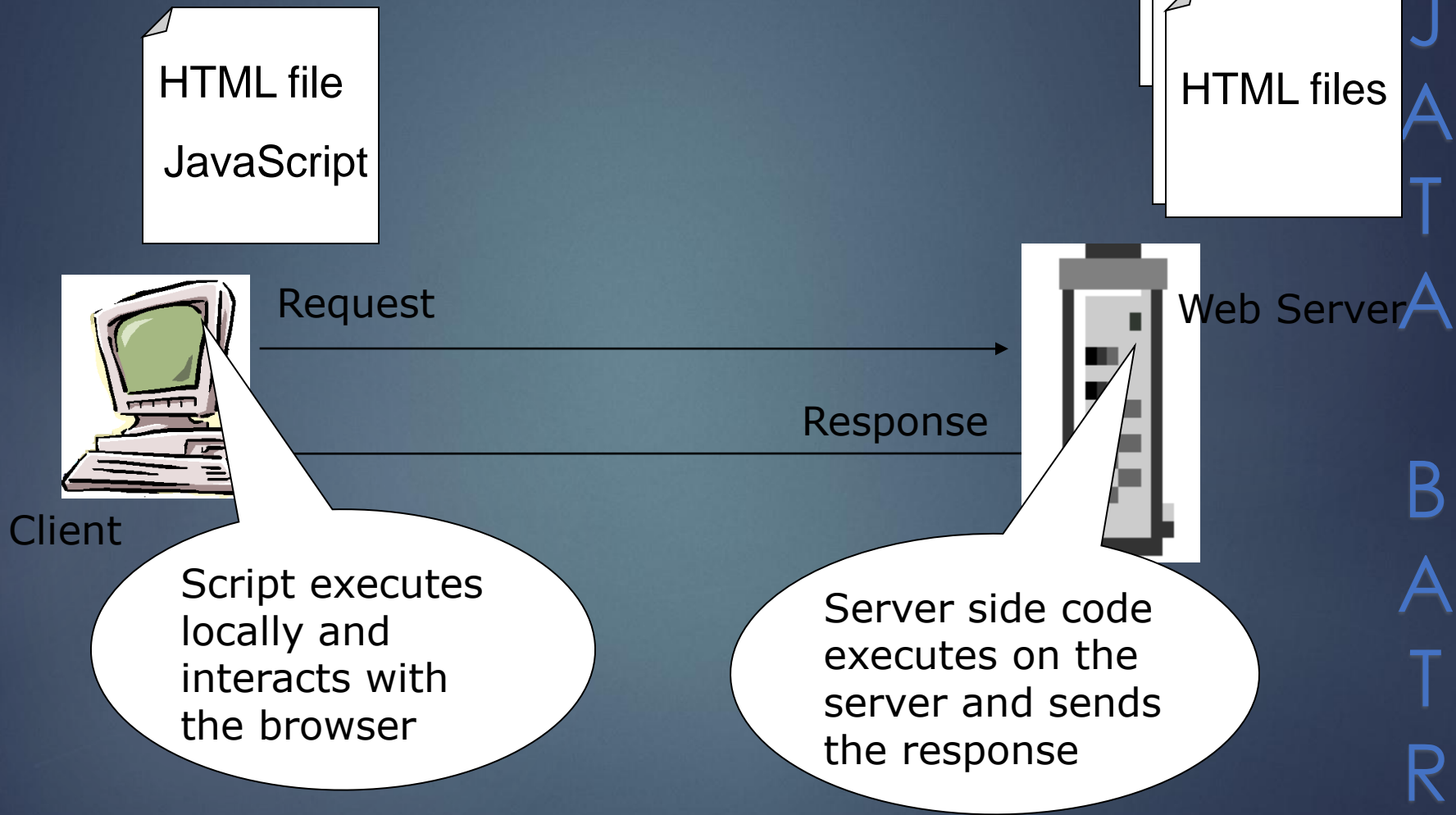
Interpreted Language

- ▶ Older implementations are purely *interpreted language*.
 - ▶ The user's browser *interprets* the script analyzes and immediately executes it.
 - ▶ interpreter in the browser reads over the JavaScript code, interprets each line, and runs it
- ▶ In modern implementations have Just In Time Compilers
 - ▶ Modern browsers now compile on the fly.
 - ▶ JIT compilers use their own strategies for optimization.
 - ▶ compiler are dedicated to optimizing a single task;
 - ▶ may optimize common operations such as loops or functions.
 - ▶ May employ some strategy to improve the performance of your code.

Common scripting tasks

- ▶ **adding dynamic features to Web pages**
 - ▶ validation of form data
 - ▶ image rollovers
 - ▶ time-sensitive or random page elements
 - ▶ handling cookies
- ▶ **defining programs with Web interfaces**
 - ▶ utilize buttons, text boxes, clickable images, prompts, frames
- ▶ **limitations of client-side scripting**
 - ▶ since script code is embedded in the page, viewable to the world
 - ▶ for security reasons, scripts are limited in what they can do
 - ▶ since designed to run on any machine platform, scripts do not contain platform specific commands
 - ▶ user can always disable JavaScript

JavaScript Execution



So JavaScripts on client side executes faster than server-side code.

Embedding JavaScript into HTML page

- ▶ `<SCRIPT>.....</SCRIPT>` tag

```
<SCRIPT LANGUAGE="JavaScript">
```

```
-----
```

```
    (JavaScript code goes here)
```

```
-----
```

```
</SCRIPT>
```

- ▶ LANGUAGE - the scripting language used for writing scripts

Deferred and Immediate Script

- ▶ SCRIPT tag can be placed in HEAD or BODY tag
- ▶ Mode of execution is decided by where is the SCRIPT tag placed.
- ▶ Immediate mode
 - ▶ SCRIPT tag is placed in the BODY tag or the HEAD tag
 - ▶ Scripts gets executed as the page loads.

```
<body>
```

```
<h4> Immediate Demo</h4>
```

```
<script language="JavaScript">
```

```
document.write("<h5> Using JavaScript</h5>");
```

```
</script>
```

```
</body>
```

Deferred and Immediate Script

► Deferred mode

- Script is executed based on some user action
- SCRIPT tag is placed in the HEAD tag

```
<script language="JavaScript">
<!--
/*comments : calling function when user clicks on the button */
    function msg(){
        alert("Hi");
    }
// → </script> <form name="f1">
    <input type="button" value=" ok " onClick="msg()">
</form>
```

Adding Java Script

- ▶ Added By Using Script Tag
 - ▶ `<script></script>`
 - ▶ `<script type="text/javascript"></script>`
- ▶ `<noscript>`
 - ▶ An alternate content for users that have disabled scripts in their browser or have a browser that doesn't support script.

Placeholders for scripts within HTML

13

- **Inside head**
 - only declarations. Declarations could for variables or functions.
 - No standalone executable statements must appear
- **Inside the body**
 - any statements can appear
 - standalone executable statements inside the body are interpreted in place where it appears.
- **Along with the event handler**
 - Script expression can be written as a value when an event like button click happens.

Script Execution – Head Section

- Scripts, placed in the head section will be executed
 - When they are called or
 - When an event is triggered.

```
<html>
```

```
<head>
```

```
  <script type="text/javascript">
```

```
    some statements
```

```
  </script>
```

```
</head>
```

```
</html>
```


- **Advantage →**
- **The script will be loaded before anyone uses it.**

Scripts Execution - Body Section

15

- A script to be executed while the page is being loaded is placed in the body section.

```
<html>
<head> </head>
<body>
  <script type="javascript">
    some statements
  </script>
</body>
</html>
```

- **Advantage** →  The script generates the content of the page.

Script Execution

```
<head>
<script>
    var app={};
    app.show=function(){
        document.writeln("<h1> Hello From Java  Script</h1>");
    }
</script>
</head>
<body>
<script>
    app.show();
</script>
</body>
```

In lining of JavaScript

17

```
<html>
<head>
<title>Example</title>
</head>
<body>

<input type="button" value="Click"
      onclick="javascript:alert('Wrong Button Clicked')"/>
</body>

</html>
```

Calling Java Script From .js Files

18

```
var app={};

app.showMessage= function() {

console.log("Hello World");
};

app.init=function(){

    app.element = document.getElementById("btn1");

    app.element.onclick = app.showMessage;

};
```

HTML Page

19

S
U
J
A
T
A

B
A
T
R
A

```
<body>  
  <button id="btn1">Click</button>  
  
  <script src="scripts/Sample.js"></script>  
  
  <script>  
    app.init();  
  </script>  
  
</body>
```

Java Script Object

- ▶ Everything in JavaScript is an object
- ▶ **Document object**
 - ▶ Print JavaScript messages on the web page;
 - ▶ Access HTML Elements
- ▶ Objects have **properties and methods**.
- ▶ **Properties** are values associated with an object.
- ▶ **Methods** represent what an object can do, its behavior

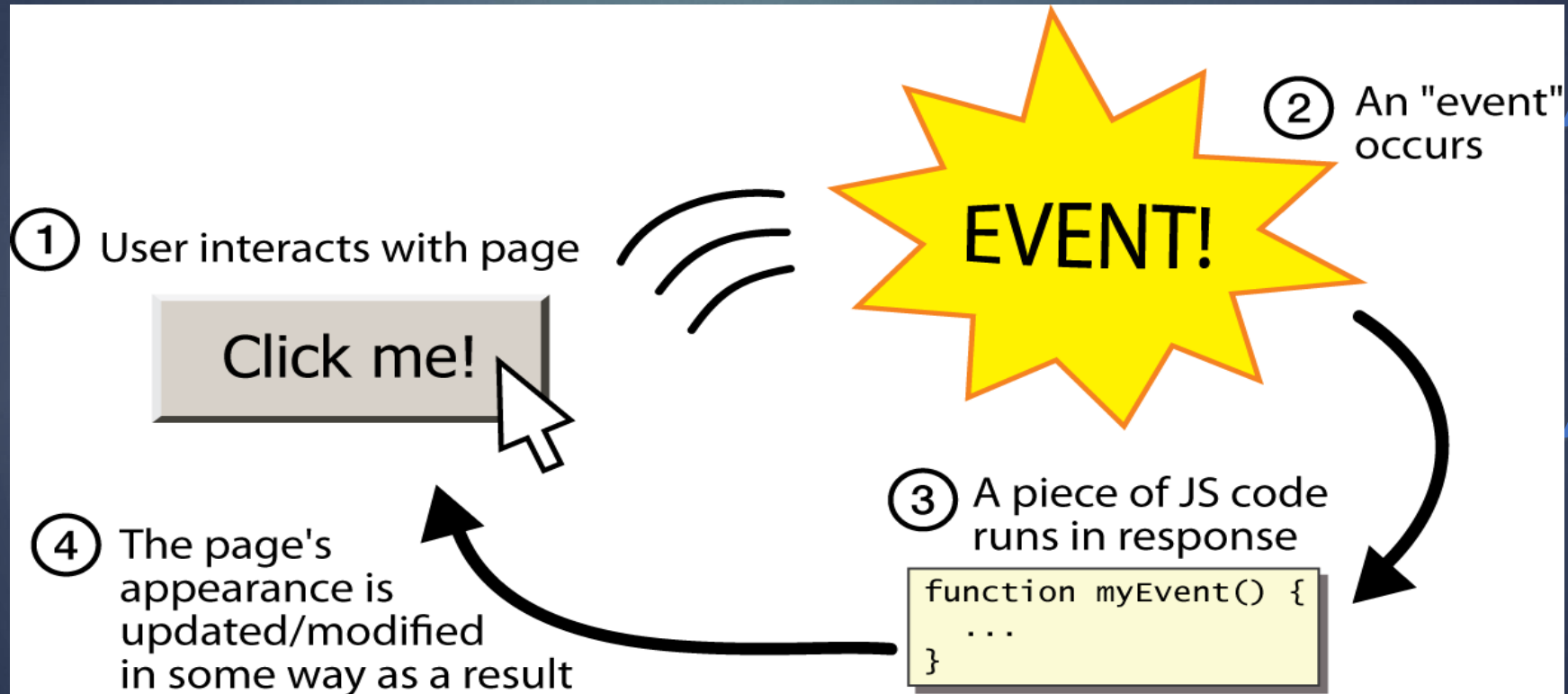
getElementById

- ▶ **HTML Elements** tags are associated with an **id**.
- ▶ **document.getElementById('id')**
 - ▶ **Method** to get the reference of the HTML Element
- ▶ **innerHTML**
 - ▶ **Property** to set Contents into a section of the HTML Tags

Java Script Events

22

- Occurrences in the context of the **interactions** between **web server**, **web browser**, and **web user**.
- Every element on a web page has certain events which can trigger invocation of event handlers



addEventListener

- ▶ Method is used to separate JavaScript from the HTML markup
- ▶ Method to attach an event handler to the specified element.
- ▶ Can add many event handlers to one element.
- ▶ Events are attached without overwriting existing handlers.
- ▶ Easier to control how the event reacts to bubbling.

```
<button id='btn2'>Click</button>
var clickButton= document.getElementById("btn2");

clickButton.addEventListener('click', function(){
    console.log("I am added");
});
```

Lexical Structure

- ▶ Character Set
- ▶ Case Sensitivity
- ▶ Optional Semicolons
- ▶ Valid Identifiers
- ▶ Reserved Word

Lexical Structure

▶ Character Set

- ▶ Scripts are written using the Unicode character set.
- ▶ Unicode supports virtually every written language currently used on the planet.

▶ Case Sensitivity

- ▶ The Following should typed with a consistent capitalization of letters.
 - ▶ **keywords,**
 - ▶ **variables,**
 - ▶ **function names**
 - ▶ *Identifiers*

Case sensitivity

26

S
U
J
A
T
A

```
app.attach = function(){
```

```
var element = document.getElementById("clickButton");
```



```
element.onClick = function (){
```

```
    alert("camel case");
```

```
}
```

```
element.onClick = function (){
```



```
    alert("lower case");
```

```
}
```

```
}
```

B
A
T
R
A

Optional Semi Colons

- ▶ Required When two statement are on the Same Line
 - ▶ `var i = 0; i++`
- ▶ Can be omitted if the statement is followed by a line break
 - ▶ `var i;`
 - ▶ `i = 5;`
 - ▶ `i = i + 1;`
 - ▶ `var x = 9;`
- ▶ Shouldn't put a semicolon after a closing curly bracket `}`.
- ▶ The only exceptions are *assignment statements*,
- ▶ `var obj = {};`

Valid Identifiers

- ▶ A JavaScript identifier can start
 - ▶ with a letter,
 - ▶ underscore (`_`),
 - ▶ dollar sign (`$`);
- ▶ Subsequent characters can also be digits (0-9).
- ▶ Examples of valid identifiers
 - ▶ logo
 - ▶ number_hits
 - ▶ \$credit
 - ▶ temp99

Reserved Word

- ▶ Abstract, super , export ,interface
- ▶ synchronized , extends , let , this
- ▶ catch , final , native , throws
- ▶ finally , transient , class , const
- ▶ Package, import, try , continue ,
- ▶ Private, protected default public
- ▶ void implements volatile static

Data Types

- ▶ Data Types
 - ▶ Primitive Data Types
 - ▶ Composite Data Types
- ▶ Variables
 - ▶ Valid Names
 - ▶ Declaring and Initializing Variables
 - ▶ Dynamically or Loosely Typed Language
 - ▶ Scope of Variables
 - ▶ Concatenation and Variables
- ▶ Constants

Data Types

- ▶ **Primitive data types** are the simplest building blocks of a program.
- ▶ JavaScript supports three core or basic data types:
 - ▶ numeric
 - ▶ string
 - ▶ Boolean
- ▶ Two other special types that consist of a single value:
 - ▶ null
 - ▶ Undefined
- ▶ **Composite Data Types**
 - ▶ A data type that can consist of many types grouped in some way.
 - ▶ Object & Array is the Composite data types.

Strongly Typed

- ▶ A **strongly typed language** has variable which will always behave as a certain **type** until it is reassigned.
- ▶ **Java** and C#
- ▶ They are More likely to generate an error or refuse to compile if the argument passed to the function does not closely match the expected type.

Loosely Typed Language

- ▶ Can declare a variable, but it doesn't require to classify the type
- ▶ **Dynamic programming language are loosely typed**
 - ▶ Can add new code
 - ▶ Can extend objects and definitions
 - ▶ Can modify the type system during runtime
 - ▶ Values are checked during execution
 - ▶ A poorly typed operation might cause the program to halt or otherwise signal an error at run time.
- ▶ `var answer = 42;`
- ▶ `answer = 'Thanks all';`

Variable

34

- Information is stored in variables.
- It's value can be changed during the execution of script.
- It can be referenced by its name to see its value or to change its value.
- Data type supported – ***Variant***.

Variant Any Type of Data.

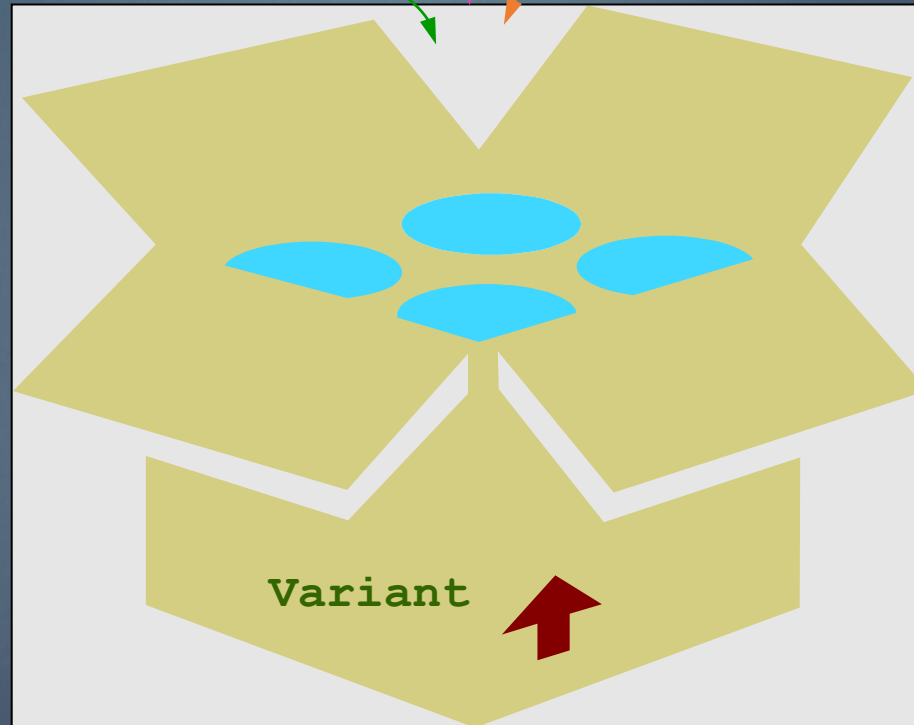
35

Float

Integer

Boolean

Char



Declaring Variables

- There are two ways of declaring a variable.
- **Implicit Declaration**
 - Declared by using its name directly in the script.
 - name="Tom"
- **Explicit Declaration**
 - Declared with the keyword 'var' keyword
 - var name = "Tom"

```
var a;
```

```
console.log('The value of a is ' + a);  
// The value of a is undefined
```

```
console.log('The value of c is ' + c);  
// Uncaught ReferenceError: c is not defined
```

Variable Declaration

- ▶ variable declaration
`var state;`
- ▶ variable definition (assignment)
`state = "ready";`
- ▶ declaration plus definition (**Initialization**)
`var state = "ready";`
- ▶ *Best Practice is to declare all variables at the beginning of every scope.*

undefined

```
var input;
```

```
if (input == undefined) {  
    doThis();  
} else {  
    doThat();  
}
```

- ▶ The undefined value behaves as false when used in a Boolean context

Nan and Null

```
var a;  
a + 2; // Evaluates to NaN
```

▶ null value

- ▶ Behaves as 0 in numeric contexts
- ▶ Behaves as false in boolean contexts.

```
var n = null;  
console.log(n * 32); // Will log 0 to the console
```

Scope of variables

- ▶ Can be **local** or **global scope**
- ▶ **Local Scope:**
 - ▶ Created using **var** inside the function
 - ▶ visible, or accessible, only within the function in which it lives.

```
function three() {  
    var a = 3;  
    alert(a); //prints 3  
}
```

```
three();
```

Scope of Variables

- ▶ **Global scope** means that it's accessible, anywhere in script..
 - ▶ Created **with or without “var” outside** the function
 - ▶ Created **without “var” inside** the function

```
var a = 1;  
  
function one() {  
    alert(a);    //prints 1  
}
```

Local and Global Variables

42

```
(function () {  
    glbl_total = 0;  
    var glbl_z = 50;  
    function sum() {  
        glbl_y = 20;  
        var local_x = 10;  
        glbl_total = local_x + glbl_y;  
        alert(glbl_z);    // print 50  
    }  
    sum();  
    alert("Total"+glbl_total); // prints 30  
    alert("Value of y" + glbl_y); // prints 20  
    alert(glbl_z);    //prints 50  
    alert("Reference Error -x" + local_x); // error  
})();
```

Variable Scope

- ▶ JavaScript's doesn't have a block scope.
 - ▶ As in C or Java, blocks ({...})
- ▶ **Variables in JavaScript are scoped to their nearest parent function or globally if there is no function present.**
- ▶ The JavaScript interpreter make two passes on a section of JavaScript code.
 - ▶ The first pass processes variables and function declarations and lifts them to the top (the 'hoisting').
 - ▶ The second pass processes the function expressions and undeclared variables.

Hoisting

- ▶ Hoisting is moving all declarations to the top of the current scope
 - ▶ To the top of the current script or the current function
- ▶ Only **variable declarations are hoisted to the top**,
- ▶ **Initialization and assignments are NOT hoisted** to the top.
- ▶ variable can be declared after it has been used.

```
console.log(x === undefined); // true
var x = 3;
```
- ▶ The above code is Interpreted as

```
var x;
console.log(x === undefined); // true
x = 3;
```


Hoisting

Foo is hoisted
up to the
nearest
parent
function.

```
var foo = 1;  
function bar() {  
  if (!foo) {  
    var foo = 10;  
  }  
  alert(foo);  
}
```

- ▶ `bar();`
- ▶ **Will display 10**

```
var foo = 1;  
function bar() {  
  if (!foo) {  
    foo = 10;  
  }  
  alert(foo);  
}
```

- ▶ `bar();`
- ▶ **Will Display 1**

The undefined value behaves as false when used in a Boolean context

Window Object –Dialog boxes

46

- **Alert Box:**
 - A small window that pops up with a yellow warning sign used important message.
- **Confirm box**
 - User will have to click either "OK" or "Cancel" to proceed
- **Prompt box**
 - User will have to click either "OK" or "Cancel" to proceed after entering an input value

Example

```
app.promptBox = function(){
    var user = prompt("Enter Your Name", "Guest");
    if(user.toUpperCase()=== "RAMESH") {
        app.update("info","welcome:="+user);
    }
    else {
        app.update("info","Dear:="+user);
    }
}

app.update = function(elementId,message){

    var showMsgElement = document.getElementById(elementId);
    showMsgElement.innerHTML= message;
}
```

Example

```
app.confirmBox = function(){  
  
    var result = confirm("Are You sure to submit");  
  
    if(!result)  
    {  
        app.update("info","Try Again");  
    }  
    else  
    {  
        window.location.assign("FormExample.html");  
    }  
}
```

Example

```
app.init = function(){  
  
  var promptElement = document.getElementById("promptButton");  
  
    promptElement.addEventListener('click',app.promptBox);  
  
  
  var confirmElement = document.getElementById("cfrmButton");  
  
    confirmElement.addEventListener('click',app.confirmBox);  
  
  }  
}
```

Example

```
<button id="promptButton">Prompt Box</button>
```

```
  <button id="cfrmButton">Confirm Box</button>
```

```
<div id="info"></div>
```

```
<script type="text/javascript"
```

```
  src="scripts/DialogBoxes.js"></script>
```

```
<script type="text/javascript">
```

```
  app.init();
```

```
</script>
```


Operators

- ▶ JavaScript Operators and Expressions
 - ▶ Assignment
 - ▶ Precedence and Associativity
- ▶ Types of Operators
 - ▶ Arithmetic Operators
 - ▶ Shortcut Assignment Operators
 - ▶ Autoincrement and Autodecrement Operators
 - ▶ Comparison Operators
 - ▶ Logical Operators
 - ▶ The Conditional Operator
- ▶ Data Type Conversion
 - ▶ The parseInt() Function
 - ▶ The parseFloat() Function
 - ▶ The eval() Function

Operators

- Arithmetic:

+ - * / % += -= *= /= %= ++ --

- Logical:

& | ! && ||

- Relational:

> >= < <= == != === !==

- String concatenation: +

- Ternary: ?:

=== and !==

53

- `alert("34"==34);` Returns true
- `alert("34"===34);` Returns false
- `alert(true===1);` returns false
- The (=) operator is used to **assign or give** a value to a variable. It is **not a sign for equality**.
- (==) compares only values, (===) compares both values and data type.

Ternary Operator

- ▶ Operator that takes three operands.
 - ▶ *used as a shortcut for the if statement*
- ▶ ***condition ? expr1 : expr2***
- ▶ If condition is true, returns the value of expr1; otherwise, value of expr2.
- ▶ **`console.log(mark>20 ? 'pass':'fail');`**

parseInt

- ▶ Function parses a string argument and returns an integer of the specified radix
- ▶ **`parseInt(string, radix);`**
 - ▶ **`console.log(parseInt('15.99', 10));`**
 - ▶ **`console.log(parseInt(' 0xF', 16));`**
 - ▶ **`console.log(parseInt(' F', 16));`**
 - ▶ **`console.log(parseInt('17', 8));`**

parseFloat

- ▶ Parses a string argument and returns a floating point number.
- ▶ **parseFloat(*string*)**
- ▶ If the first character cannot be converted to a number, NaN is returned.
 - ▶ `console.log(parseFloat('3.14'));`
 - ▶ `console.log(parseFloat('FF2'));`

eval

57

- ▶ Evaluates JavaScript code represented as a string.
- ▶ If the completion value is empty, undefined is returned.
- ▶ The argument of the eval() function is a string.
- ▶ If the string represents an expression, evaluates the expression.
- ▶ Should not be used to evaluate an arithmetic expression

- ▶ `console.log(eval(200+'400'));`

- ▶ If the Argument is not String it returns the Object itself
 - ▶ `console.log(eval(new String('2 + 2'))); // returns String Object`

- ▶ `console.log(eval('2 + 2')); // returns 4`

Control Structures

- ▶ Conditionals
 - ▶ **if/else**
 - ▶ **if/else if**
 - ▶ **switch**
- ▶ Loops
 - ▶ **The while Loop**
 - ▶ **The do/while Loop**
 - ▶ **The for Loop**
 - ▶ **The for/in Loop**
 - ▶ **Loop Control with break and continue**
 - ▶ **Nested Loops and Labels**

If-else

- ▶ The **if statement** executes a statement if a specified condition is true.
- ▶ If the condition is false, another statement can be executed.
- ▶ **if (condition)**
- ▶ **statement1**
- ▶ **[else statement2]**
- ▶ condition An expression that evaluates to true or false.

Switch Statement

60

```
switch (expression) {  
    case value1:  
        [break;]  
    case value2:  
        [break;]  
    case valueN:  
        [break;]  
    default:  
}
```

For Loop

- ▶ All three expressions in the head of the for loop are optional.
- ▶ `for (var i = 0; i < 9; i++) { console.log(i); // more statements }`
- ▶ *initialization* block it is not required to initialize variables:

```
var i = 0;
  for (; i < 9; i++) {
    console.log(i);
    // more statements
  }
```

For Loop

- ▶ Like the *initialization* block, the *condition* block is also optional.
- ▶ Loop is broken condition infinite loop.

```
for (var i = 0;; i++) {  
    console.log(i);  
    if (i > 3)  
        break;  
    // more statements  
}
```


For Loop

- ▶ Can omit all three blocks.
- ▶ A `break` statement to end the loop and also modify (increase) a variable,

```
var i = 0;
for (;;) {
  if (i > 3)
    break;
  console.log(i);
  i++;
}
```

Do-while

- ▶ The **do...while statement** creates a loop that executes a specified statement until the test condition evaluates to false.
- ▶ The condition is evaluated after executing the statement, resulting in the specified statement executing at least once.

```
var i = 0;  
do {  
    i += 1;  
    console.log(i);  
}  
while (i < 5);
```

While Statement

- ▶ The **while statement** creates a loop that executes a specified statement as long as the test condition evaluates to true.
- ▶ The condition is evaluated before executing the statement.

```
var n = 0;  
var x = 0;  
while (n < 3) {  
    n++; x += n;  
}
```

Break

- ▶ The **break statement** terminates the current loop, switch, or label statement and transfers program control to the statement following the terminated statement.

```
var i = 0;
var x = 2;
while (i < 6) {
  if (i == 3) {
    break;
  }
  i += 1;
}
console.log(i * x);
```

- ▶ Returns value 6

Continue

- ▶ Terminates execution of the statements
- ▶ Continues execution of the loop with the next iteration.

```
var i = 0;
var n = 0;
while (i < 5) {
    i++;
    if (i === 3) {
        continue;
    }
    n += i;
    console.log(n);
}
```

→ 1, 3, 7, and 12

Java script functions

Functions

- ▶ A way of packaging JavaScript commands
 - ▶ defined inside `<head>` , `<body>` or in a **external file**
 - ▶ Used to implement some functionality for web pages
 - ▶ Makes it easy to reuse .
 - ▶ Its **also** a object.
- ▶ Functions are associated with Objects
 - ▶ **write(), alert(), getElementById(), random()**
- ▶ Can also be associated with global **window object**
 - ▶ **eval(),parseFloat(),parseInt()**

Function Declaration

- Declared (created) and then it is Invoked by its name (used).
 - ▶ function keyword appears first in the statement
 - ▶ followed by a function name

```
function functionName(var1,var2,...,varX)
{
  //some code
}
```

Function Expression

- ▶ Can also be created by **saving them to a variable**
- ▶ Creates a new function object and returns it.
- ▶ The **variable** can be **used to invoke** it.
- ▶ Variable can be used pass as an argument to another function

```
var greet=function(person,greeting){
```

```
    var text=greeting+"-"+person};
```

```
    Console.log(text);
```

```
}
```

```
Greet("ram","Hello')
```

Function that returns value

72

S
U
J
A
T
A

```
function greet(){  
    return "Hello";  
}
```

```
var result = greet();  
console.log(result);  => Hello;
```

```
function greet(){  
}
```

```
var result = greet();  
console.log(result);      => undefined
```

B
A
T
R
A

Arguments to Functions

- ▶ A function can be called with less or more number of arguments than required.
- ▶ **Missing arguments** are set to **undefined**.

```
function callMe(a, b, c) {  
    console.log("c is " + typeof c);  
}
```

- ▶ callMe("Good", "Morning");
 - ▶ **Output: "c is undefined"**
- ▶ callMe("Learn", "JavaScript", "Functions");
 - ▶ **Output: "c is string"**

Optional Arguments

```
function power(base, exponent) {  
    if (exponent == undefined) {  
        exponent = 2;  
    }  
    var result = 1;  
    for (var count = 0; count < exponent; count++) {  
        result *= base;  
    }  
    return result;  
}  
  
console.log(power(4));  
console.log(power(4, 3));
```


Arguments Object

- ▶ Variables created inside a function including their parameters, are **local** to the function.
- ▶ Arguments are array object called **arguments**
- ▶ Used to access **individual arguments**
- ▶ Can get the **total number of arguments** passed to the function.

```
function callMe() {  
    for (var i = 0; i < arguments.length; i++) {  
        console.log(arguments[i]);  
    }  
    console.log("Total arguments passed: " + arguments.length);  
}
```

Anonymous functions

- ▶ Functions that are dynamically declared at runtime.
- ▶ They aren't given a name in the same way as normal functions.
- ▶ Declared using the function operator instead of the function declaration.

```
function useAnnonFunction(funcRef){  
    var ref=funcRef();  
    console.log(ref);  
}
```

```
useAnnonFunction(function(){  
    return "I am from Annon function"  
});
```

Document Object Model

What is DOM?

- ▶ DOM stands for the Document Object Model.
- ▶ DOM is a document content model for HTML documents.
- ▶ It is a hierarchy of pre-existing Objects
- ▶ A programmer can reference the object and their contents.
- ▶ The World Wide Web Consortium (W3C) is the body which sets standards for the web.
- ▶ W3C has defined a standard Document Object Model, known as the W3C DOM.
- ▶ DOM stands apart from JavaScript because other scripting languages could access it.

Hierarchy of Objects

79



DOM Top Level objects

Object	Has	Description
<i>window</i>	methods properties	The window object is the top level object in the DOM. It contains information about the window and the frames
<i>document</i>	methods properties collections	The document object represents the HTML document, and is used to access the HTML elements inside the document
<i>event</i>	properties collections	The event object contains information about events that occurs
<i>navigator</i>	methods properties	Contains information about the version of Navigator in use.

Navigator object

- ▶ The navigator object refers to the browser itself.
- ▶ Has properties only. You cannot modify properties .
- ▶ All of the properties of the navigator object are read-only.

<i>appCodeName</i>	The code name of the browser
<i>appName</i>	The name of the browser.
<i>appVersion</i>	A string that represents version information about the browser.
<i>userAgent</i>	The value of the user-agent header sent in the HTTP protocol from client to server.
<i>platform</i>	indicates the platform (Windows, UNIX, and so on)

Window Object

- ▶ The window object is the "parent" object for all other objects
- ▶ Enables to access the browser window or the frame within it.
- ▶ Some methods are
 - ▶ `alert(message)`
 - ▶ `prompt(message[, inputDefault])`
 - ▶ `confirm(message)`
 - ▶ `moveTo(x-coordinate, y-coordinate)`
 - ▶ `open(URL, windowName[, windowFeatures])`
 - ▶ `close()` `setTimeout(function name, milliseconds)`
 - ▶ `clearTimeout(milliseconds)`
 - ▶ `status` property

Window Methods

```
<HTML>
```

```
<BODY>
```

```
<FORM name="Winform">
```

```
<INPUT TYPE="button" value="Open new window"
  onClick="NewWin=window.open(' ','NewWin','toolbar=no,status=no,wid
th=700,height=500');"><BR>
```

```
<!-- HTML Comments : NewWin is the reference of the new window -->
```

```
<INPUT TYPE="button" value="Close new window"
  onClick="NewWin.close();"><BR>
```

```
<INPUT type=button value="Close main window"
  onClick="window.close();">
```

```
</form>
```

```
<!-- HTML Comments : window is the reference of the current window
-->
```

```
</BODY>
```

```
</HTML>
```

Window Object – Method

- ▶ **setTimeout** – calls a function or evaluates an expression after a specified interval
- ▶ Syntax: **setTimeout("Update();",2000);**
- ▶ **setTimeout** returns a timer ID that can be used in a call to **clearTimeout()**.
- ▶ **clearTimeout**- cancels the timeout specified by timeout id.
- ▶ Syntax: **clearTimeout(id);**

History Object

- ▶ Contains information on URLs that the user has visited in a window
- ▶ Provides methods to revisit those URLs
- ▶ Properties :
 - ▶ `length` : Reflects the number of entries in the history list.
 - ▶ `next` : Specifies the URL of the next history entry.
 - ▶ `current` : Specifies the URL of the current history entry.
 - ▶ `previous` : Specifies the URL of the previous history entry.
- ▶ Methods :
 - ▶ `back()` : Loads the previous URL in the history list.
 - ▶ `forward()` : Loads the next URL in the history list.
 - ▶ `go(int)` : Loads a URL from the history list.

Eg: `history.go(-2)` // goes back to the URL the user visited three clicks ago in the current window.

Location Object

- ▶ Contains information on the current URL.
- ▶ properties

host	A string specifying the server name, subdomain, and domain name.
hostname	A string containing the full hostname of the server
href	A string specifying the entire URL.
protocol	A string specifying the beginning of the URL, up to and including the first colon.

🔗 Methods

reload()	Forces a reload of the window's current document
replace(URL)	Loads the specified URL over the current history entry.

Document Object

- ▶ Each page has only one document object.
- ▶ Contains information about the current document, and provides methods for displaying HTML output to the user.
- ▶ Methods

<code>write(string)</code>	Writes one or more HTML expressions to a document in the specified window.
<code>writeln(string)</code>	Writes one or more HTML expressions to a document in the specified window and follows them with a new line character.
<code>open()</code>	Opens a stream to collect the output of write method.
<code>close()</code>	Closes an output stream and forces data to display.

Document Object

► Properties

URL	A string that specifies the complete URL of a document.	SUBJECT AREA
title	A string that specifies the contents of the TITLE tag.	
lastModified	A string that specifies the date the document was last modified.	
alinkColor linkColor vlinkColor	A string that specifies the ALINK attribute. A string that specifies the LINK attribute. A string that specifies the VLINK attribute.	
fgColor bgColor	A string that specifies the TEXT attribute. A string that specifies the BGCOLOR attribute.	BROWSER AREA
forms	An array containing an entry for each form in the document.	
height, width	Height and width of the browser	

Form Object

- ▶ Each form in a document creates a form object.
- ▶ A document can have more than one form
- ▶ Form objects in a document are stored in a `forms[]` collection.
- ▶ The elements on a form are stored in an `elements[]` array.
- ▶ Each form element has a form property that is a reference to the element's parent form.

```
<form name="form1">  
  <input type="radio" name="rad1" value="1">Yes<br>  
  <input type="radio" name="rad1" value="0">No<br>  
</form>
```

To access the first radio button value you can use

```
document.forms[0].elements[0].value
```

(or)

```
document.form1.rad1[0].value
```

Form Object

► Properties

<code>action</code>	Reflects the ACTION attribute.
<code>elements</code>	An array reflecting all the elements in a form.
<code>length</code>	Reflects the number of elements on a form.
<code>method</code>	Reflects the METHOD attribute.
<code>target</code>	Reflects the TARGET attribute.

● Methods

<code>reset()</code>	Resets a form.
<code>submit()</code>	Submits a form.

● Event Handlers

- `onReset`, `onSubmit`

Text, Textarea, Password, hidden Objects

► Properties

- ▶ `defaultValue` : Reflects the VALUE attribute.
- ▶ `name` : NAME attribute.
- ▶ `type` : Reflects the TYPE attribute.
- ▶ `value` : Reflects the current value of the Text object's field.

► Methods

- ▶ `focus()` : Gives focus to the object.
- ▶ `blur()` : Removes focus from the object.
- ▶ `select()` : Selects the input area of the object.

► Event Handler

- ▶ `onBlur` : when a form element loses focus
- ▶ `onChange` : field loses focus and its value has been modified.
- ▶ `onFocus` : when a form element receives input focus.
- ▶ `onSelect` : when a user selects some of the text within a text field.

Radio object

► Properties

- name, type, value, defaultChecked, defaultvalue, checked
- checked property will have a Boolean value specifying the selection state of a radio button. (true/false)

► Methods

- click(), focus(), blur()

► Event Handler

- onClick, onBlur, onFocus()

```
if(document.forms[0].rad1[0].checked == true)
{
    alert("button is checked")
}
```


Checkbox object

▶ Properties

- ▶ checked, defaultChecked, name, type, value

▶ Methods

- ▶ click()

▶ Event Handler

- ▶ onClick, onBlur, onFocus()

```
if (document.form1.chk1.checked==true)
    red=255;
else
    red=0;
```

Button, reset, submit Objects

- ▶ Properties
 - ▶ form, name, type, value
 - ▶ Methods
 - ▶ click(), blur(), focus()
 - ▶ Event Handler
 - ▶ onClick, onBlur, onFocus, onMouseDown, onMouseUp
 - ▶ The disabled property
 - ▶ If this attribute is set to true, the button is disabled
 - ▶ This attribute can use with all other form elements to disable the element
- ```
<input type="button" name="b1" value="ok" disabled>
document.forms[0].b1.disabled=true;
```

# Select Object

- ▶ The user can choose one or more items from a selection list, depending on how the list was created.

- ▶ Properties

length	Reflects the number of options in the selection list.
options	Reflects the OPTION tags.
selectedIndex	Reflects the index of the selected option (or the first Selected option, if multiple options are selected).

- ▶ Methods

- ▶ blur(), focus()

- ▶ Event Handler

- ▶ onBlur, onChange, onFocus



# Option Object

- ▶ An option in a selection list.
- ▶ Properties

index	The zero-based index of an element in the <code>Select.options</code> array.
selected	Specifies the current selection state of the option
text	Specifies the text for the option
value	Specifies the value for the option
length	The number of elements in the <code>Select.options</code>

# Image Object

- ▶ An image on an HTML form.  
``
- ▶ Properties
  - ▶ src
  - ▶ width
  - ▶ Height
- ▶ You can change the image using the src property.

```
document.img1.src="s2.jpg"
```

# Form Validation

- ▶ Client-side form validation is an important part of a website where data needs to be collected from the user.
- ▶ It is the job of the web programmer, to make sure his pages which use forms include client-side form validation using JavaScript.
- ▶ What to Validate?
  - ▶ required fields are entered    correct data type is entered
  - ▶ user's input conforms to a certain standard (such as telephone numbers, email, etc.).
  - ▶ Data is in a certain range. (age between 1 and 100)
- ▶ Tips    Put your common JavaScript validation functions into a common file (.js).
  - ▶ validate data on all the fields in your forms ( if storing inside database)
  - ▶ JavaScript string object has a lot of functionality that many people spend time programming themselves
  - ▶ Limit the number of characters in a field and disable the fields which user need not enter the data. (use MAXLENGTH of form elements)



# Advantages of Client Side Validation

99

- ▶ Client side validation is faster than server side validation
- ▶ Validating forms on the client side lightens the load on the server and allows for you to format data before it is sent for processing.
- ▶ Using server-side validation, there is noticeable lag time when data the user submits contains an error. Form data must travel over the Internet and be examined by the server.
- ▶ Developers will find it easier and more intuitive to do form validation with JavaScript than with server side techniques since the code has already been written and is easy to implement.

# Form Validation

- ▶ Let's look at an example.
- ▶ You have a form with the following fields:
  - ▶ Name Age Email
- ▶ What to validate
  - ▶ All the fields are entered.
  - ▶ Name should be a string with minimum 3 Character.
  - ▶ Age should be a number between 1 and 100
  - ▶ Email should be a string with an "@" character.
- ▶ Make use of String functions and top level functions to validate the fields