JavaScript

CSCI 3000
Web Programming

Dr. Luis A. Cueva-Parra

JavaScript

- ***JavaScript*** is an interpreted (scripting) programming language.
- It was created by Netscape Communications in 1995. It was called ***Mocha***, then ***LiveScript*** (beta release of Netscape Navigator 2.0) and finally renamed ***JavaScript*****.**
- ***JavaScript*** extends the capabilities of HTML for creating interactive web pages.
- ***JavaScript*** is case sensitive. Beware of line breaks and spaces.

JavaScript – Changes HTML content

```
<!DOCTYPE html>
<html>
<body>
<h1>What Can JavaScript Do?</h1>
<p id="demo">JavaScript can change HTML content.</p>
<button type="button" onclick="document.getElementById('demo').
innerHTML = 'Hello JavaScript!'">Click Me!</button>
</body>
</html>
```

JavaScript – Changes HTML attributes

```
<p>JavaScript changes HTML attributes.</p>

<p>In this case JavaScript changes the src (source) attribute of an image.</p>

<button onclick="document.getElementById('myImage').src='pic_bulbon.gif'">Turn on the light</button>

<img id="myImage" src="pic_bulboff.gif" style="width:100px">

<button onclick="document.getElementById('myImage').src='pic_bulboff.gif'">Turn off the light</button>
```

JavaScript – Changes HTML styles

```
<!DOCTYPE html>
<html>
<body>
<h1>What Can JavaScript Do?</h1>
<p id="demo">JavaScript can change the
style of an HTML element.</p>
<button type="button"
onclick="document.getElementById('demo').
style.fontSize='35px'">Click Me!</button>
</body>
</html>
```

JavaScript – Hides HTML elements

```
<!DOCTYPE html>
<html>
<body>
<h1>What Can JavaScript Do?</h1>
<p id="demo">JavaScript can hide HTML
elements.</p>
<button type="button"
onclick="document.getElementById('demo').
style.display='none'">Click Me!</button>
</body>
</html>
```

JavaScript – Shows HTML elements

```
<!DOCTYPE html>
<html>
<body>
<h1>What Can JavaScript Do?</h1>
<p id="demo" style="display:none">
JavaScript can show HTML elements.</p>
<button type="button"
onclick="document.getElementById('demo').
style.display='block'">Click Me!</button>
</body>
</html>
```

JavaScript Location

- A ***JavaScript*** code can be embedded in an HTML tag, located in the `<head>`, located in the `<body>` or written in an external file (with `.js` extension).
- For locating a ***JavaScript*** code in the `<head>` or `<body>` we use the HTML tag `<script>`.

```
<script>
function myFunction() {
document.getElementById("demo").
innerHTML = "Paragraph changed."; }
</script>
```

JavaScript Location

- Then have the function call in the HTML file:

```
<p id="demo" style="display:none">
JavaScript can show HTML elements.</p>
<button type="button"
onclick="myFunction()">Click Me!</button>
```

- If an external file (myScript.js) contains the ***JavaScript*** code, it should be linked in the HTML file as:

```
<script src=”myScript.js”></script>
```

or

```
<script src=”https://www.w3schools.com/
js/myScript.js”></script>
```

JavaScript Output

- ***JavaScript*** displays data in different ways.
- It writes into
  - an HTML element, using **`innerHTML`**.
  - into the HTML output using **`document.write()`**.
  - into an alert box, using **`window.alert()`**.
  - into the browser console, using **`console.log()`**.

JavaScript Output

- To display data using an *alert box*:

```
<script>
window.alert(5 + 6);
</script>
```

- To *write* into HTML output:

```
<script>
document.write(5 + 6);
</script>
```

- After the HTML document is loaded `document.write` will override it. Use it for testing only.

JavaScript Output

- To write *into* an HTML element:

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

- To write to the browser *console* (F12), usually for debugging:

```
<script>
console.log(5 + 6);
</script>
```

JavaScript Syntax

- JavaScript statements are separated by semicolons.
- In HTML, JavaScript programs can be executed by the web browser.
- JavaScript statements are composed of: Values, Operators, Expressions, Keywords, and Comments.
- *JavaScript Values*:
  - Constants or literals (fixed): `10.34` or `234` or `“joe”`
  - Variables:

```
var x; //declaration
x = 6; //assignment
```

JavaScript Syntax

- *JavaScript Operators*:
  - Assignment operator (`=`)
  - Arithmetic operators (`+ - * / % ++ --`)
- *JavaScript Expressions*: Are combination of values, variables, and operators, which computes to a value.
- *JavaScript Keywords*: Are reserved words for the JavaScript language: `var`, `function`, `if`, `else`, `true`, `false`, `while`, `abstract`, `arguments`, `boolean`, `in`, etc.

JavaScript Syntax

- *JavaScript Comments*:

```
   var x = 5; // this is a comment
  /* var x = 6;  this will NOT be
  executed */
```

- *JavaScript Identifiers*: Are names used for variables, keywords, functions, and labels.
  - Their first character must be a letter, an underscore (_), or a dollar sign ($).
  - Subsequent characters may be letters, digits, underscores, or dollar signs (no hyphens).
- JavaScript is case sensitive.

JavaScript Statements

```
var x = 5; var y = 6; var z = x + y;
var pi = 3.14;
var person = "John Doe";
var answer = 'Yes I am!';
```

- JavaScript code:

```
<p id="demo"></p>
<script>
var carName = "Volvo";
document.getElementById("demo").
innerHTML = carName;
</script>
```

JavaScript Keywords

KEYWORD	DESCRIPTION
`break`	Terminates a switch or a loop
`continue`	Jumps out of a loop and starts next iteration
`debugger`	Stops JavaScript ans calls the debugging function
`do … while`	Executes and then repeats while the condition is true
`for`	Executes as long as a condition is true
`function`	Declares a function
`if … else`	Executes depending on a condition
`return`	Exit a function
`switch`	Executes depending on different cases
`try … catch`	Implements error handling
`var`	Declares a variable

JavaScript Arithmetic

```
var x = 5 + 2 + 3; // 10
var pi = “Tom” + “ “ + “Doe”; //Tom Doe
var x = “5” + 2 + 3; // 523
var x = 5 + 2 + “3”; // 73
```

- **Re-declaring JavaScript variables**

  If we re-declare a variable, it will not lose its value. After the following statements, `carName` will still have the value “`Volvo`”.

```
var carName = "Volvo";
var carName;
```

JavaScript Assignment Operators

OPERATOR	EXAMPLE	EQUIVALENT TO
=	`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`

The **+=** operator can be used to concatenate strings

```
txt1 = "What a very ";
txt1 += "wonderful day";
// What a very wonderful day
```

JavaScript 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

JavaScript Comparison Operators

```
var x = 5;
(x == 8);     // false
(x == 5);     // true
(x == “5”);  // true
(x === 5);   // true
(x === “5”); // false
(x != 8);     // true
(x !== 5);   // false
(x !== “5”); // true
(x !== 8);   // true
```

JavaScript Logical Operators

OPERATOR	DESCRIPTION
&&	and
||	or
!	not

```
var x = 6;
var y = 3;
(x < 10 && y > 1);  // true
(x == 5 || y == 5); // false
!(x == y);          // true
```

JavaScript Data Types (1/5)

- JavaScript variables can hold different data types.

```
var myNumber = 53;                // Number
var myCar = “Volvo”;              // String
var p = {firstName:“Tom”,
lastName:”White”};                // Object
```

- Data types will help to decide about the result of operations on different data types.

```
var myString = 25 + "Volvo";
```

- Adding a number and string, JavaScript will treat the number as string.

JavaScript Data Types (2/5)

- JavaScript evaluates expressions from left to right.

```
var q = 3 + 8 + “Joe”;      // 11Joe
var r = “Joe” + 2 + 5;      // Joe25
```

- JavaScript data types are dynamic.

```
var x;              // x is undefined
x = 20;             // now x is a Number
x = “John”;         // now x is a String
```

- Nested quotes can be used for Strings.

```
x = ‘We call him “Pete”’;
y = “He is called ‘Tim’”;
```

JavaScript Data Types (3/5)

- JavaScript Numbers:

```
var x = 13;        // without decimals
var y = 25.45;     // with decimals
var z = 789e3;     // 789000
var w = 789e-4;    // 0.0789
```

- JavaScript Booleans

```
var a = 2;
var b =5;
(a == b)          // returns false
(a < b)           // returns true
```

JavaScript Data Types (4/5)

- JavaScript Arrays:

```
var colors = [“blue”,”red”,”green”];
color[0]     // has value “blue”
color[2]     // has value “green”
```

- JavaScript Objects

```
var person = {firstName:”Mark”,
lastName:”Johnson”, age:26,
hairColor:”brown”};
person.firstName  // has value “Mark”
person.age        // has value 26
```

JavaScript Data Types (5/5)

- The **typeof** operator returns the data type of a variable or expression.

```
typeof 53.3        // returns “number”
typeof “Volvo”     // returns “string”
var x; //value and type are “undefined”
typeof x           // returns “undefined”
```

- **null** is an object that has no value.

```
var p = {name:“Tom”, lastName:”Rod”};
p = null         //value is null, type is still
                    an object
p = undefined  //value and type are
                    “undefined”
```

JavaScript Conditional (Ternary) Operator

- Syntax

```
varName = (condition) ? value1 : value2
```

- Example

```
function myFunction() {
var age, voteable;
age = document.getElementById("age").value;
voteable = (age < 18) ? "Too young":"Old
enough";
document.getElementById("demo").innerHTML =
voteable + " to vote.";
}
```

JavaScript Functions

- Syntax

```
function funcName (par1, par2, par3){
   Code to be executed
}
```

- Example

```
function myFunction(n1, n2) {
       return n1 * n2;
}
document.getElementById("demo").innerHTML =
 myFunction(6, 9);
```

-

JavaScript Objects

- Objects are variables that hold multiple information.
- Objects ***properties:*** Pairs `name:value`

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

- Object ***methods:*** Actions performed on objects. Methods are stored in properties as *functions* definitions.

```
var person = {firstName:"John",
lastName:"Doe", age:50, eyeCol:"blue",
fullName:function(){return this.firstName
+ " " + this.lastName;}};
```

JavaScript Objects

- Accessing objects properties:

  Dot operator: `objectName.propertyName`

```
document.getElementById("demo").innerHTML
= “Hello” + person.firstName;
```

  Array notation: `objectName["propertyName"]`

```
document.getElementById("demo").innerHTML
= person["firstName"] + " " +
person["lastName"];
```

JavaScript Objects

- Accessing objects methods:

```
objectName.methodName()
```

```
document.getElementById("demo").innerHTML
= person.fullName();
```

- Accessing the `fullName` method, without `()`, it will return the function definition.

```
document.getElementById("demo").innerHTML
= person.fullName;
```

Will return

```
function() { return this.firstName + "
" + this.lastName;}
```

JavaScript Events

- HTML events are actions that happen to HTML elements. JavaScript can react on HTML events.
- Examples of HTML events:
  - An HTML web page has finished loading
  - An HTML input field was changed
  - An HTML button was clicked
- Syntax for embedded JavaScript code into HTML elements:

  <element event='JavaScript code'> or

  <element event="JavaScript code">

JavaScript Events

- Example:

```
<button
onclick="document.getElementById('demo'
).innerHTML=Date()">The time
is?</button>


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


<button
onclick="this.innerHTML=Date()">Click
here!</button>
```

JavaScript Events

HTML 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

JavaScript Strings (1/3)

- JavaScript Strings:

```
var x = “It’s alright”; //quotes inside
var sln = x.length; //string length
var z = “The \”King\” of rock”;
//double quotes inside double quotes
```

- To display special characters such as: ‘ (single quote), “ (double quote) or \ (backslash) the **backslash escape character** is used (\).

```
var a = ‘It\’s alright.’;
var b = “the character \\ is called
         Backslash”;
```

JavaScript Strings (2/3)

- Other escape sequences valid in JavaScript:

Code	Result
\b	Backspace
\f	Form Feed
\n	New line
\r	Carriage Return
\t	Horizontal Tabulator
\v	Vertical Tabulator

JavaScript Strings (3/3)

- Breaking Long Code Lines:

```
document.getElementById(“demo”).innerHTML =
“Hi there!”; //after an operator (=)
document.getElementById(“demo”).innerHTML = “hi \
there!”;//within a text string use backslash: \
document.getElementById(“demo”).innerHTML = “hi” +
“there!”;//within a text string better use: +
```

- Strings can be objects:

```
var myName = “Ben”;
var myName = new String (“Ben”);
```

- Objects cannot be compared

JavaScript Strings Methods (1/6)

- Finding a String in a String:

  The **indexOf()** method returns the index of (the position of) the first occurrence of a specified text in a string:

```
var str = "Please locate where 'locate'
occurs!";

var pos1 = str.indexOf("locate");
```

- The **lastIndexOf()** method returns the index of the last occurrence of a specified text in a string:

```
var pos2 = str.lastIndexOf("locate");
```

JavaScript Strings Methods (2/6)

- If the string is not found then both **indexOf()**, and **lastIndexOf()** return -1.
- Both methods accept a second parameter as the starting position for the search:

```
var pos3 = str.indexOf("locate", 13);
```

- The **search()** method returns the index of the first occurrence of a specified text in a string:

```
var pos4 = str.search("locate");
```

- The methods **indexOf()** and **search()** are not the same.
- The **search()** method cannot take a second argument.

JavaScript Strings Methods (3/6)

- The **indexOf()** method cannot take regular expressions as search values.
- The **slice()** method extracts a part of a string and returns the extracted part in a new string.
- Its two parameters are the starting and ending position;

```
var str = "Apple, Banana, Kiwi";
var res = str.slice(7, 13);  //returns
Banana
```

JavaScript Strings Methods (4/6)

- Negative parameters are allowed being the last character in the string at position 0.

```
var res = str.slice(-12, -6);  // also
returns Banana
```

- Omitting the second parameter, the method will slice out the rest of the string.

```
var res = str.slice(7); or
var res = str.slice(-12);
```

- The **substring()** method is similar to the **slice()** method, but it cannot accept negative indexes.

-
  ```
  var res = str.substring(7, 13);
  ```

JavaScript Strings Methods (5/6)

- The **substr()** method is similar to the **slice()** method, but its second parameter specifies the length of the part to be extracted.

```
var str = "Apple, Banana, Kiwi";
var res = str.substr(7, 6); // returns
Banana
```

- Omitting the second parameter, the method will slice out the rest of the string.

```
var res = str.substr(7);
```

- If the first parameter is negative the position counts from the end of the string.

```
var res = str.substr(-1, 1);
```

JavaScript Strings Methods (6/6)

- The **replace()** method replaces a specified value with another value in a string.

```
var str = "Please use Microsoft OS";
var p = str.replace(“Microsoft”,
“Linux”);
```

- The **replace()** method replaces only the first match.
- The **replace()** method can use a *regular expression* with flags. e.g. /i flag (case insensitive) or /g (global replacement).

```
var p=str.replace(/MICROSOFT/i, “Tux”);
var p=str.replace(/Microsoft/g, “Tux”);
```

Conditional Statements (1/4)

- The **if** statement:

```
if (condition) {
    Code to be executed if the
condition is true
}
```

- Example:

```
if (grade >= 90) {
      Message = “You got an A!”;
}
```

Conditional Statements (2/4)

- The **if-else** statement:

```
if (condition) {
    Code if condition is true
} else {
    Code if the condition is false
}
```

- Example:

```
if (grade >= 90) {
      Message = “You got an A!”;
} else{
      Message = “You didn’t get an A!;
}
```

Conditional Statements (3/4)

- The **else if** statement:

```
if (condition1) {
    Code if condition1 is true
} else if (condition2) {
    Code if the condition1 is false and
condition2 is true
} else {
    Code if the condition1 is false and
condition2 is false
}
```

Conditional Statements (4/4)

- The **else if** statement (continued):
- Example:

```
if (time < 10) {
    greeting = "Good morning";
} else if (time < 20) {
    greeting = "Good day";
} else {
    greeting = "Good evening";
}
```

The switch Statement (1/3)

- The **switch** statement syntax:

```
switch(expression) {
    case x:
        code block
        break;
    case y:
        code block
        break;
    default:
        code block
}
```

The switch Statement (2/3)

- Example 1:

```
switch (new Date().getDay()) {
    case 6:
        text = "Today is Saturday";
        break;
    case 0:
        text = "Today is Sunday";
        break;
    default:
        text = "More days to go!”;
}
```

The switch Statement (3/3)

- Example 2:

```
switch (new Date().getDay()) {
    case 4:
    case 5:
        text = "Soon it is Weekend";
        break;
    case 0:
    case 6:
        text = "It is Weekend";
        Break;
    default:
        text = "More days to go!”;
}
```

The **for** Statement (1/2)

- The **for** statement syntax:

```
for (statement 1; statement 2; statement 3)
{
    code block to be executed
}
```

- Example 1:

```
var text = "";
var i;
for (i = 0; i < 5; i++) {
    text += "The number is " + i + "<br>";
}
```

The for Statement (2/2)

- Example 2:

```
<p id="demo"></p>
<script>
var fruits, text, fLen, i;
fruits = ["Banana", "Orange", "Apple", "Mango"];
fLen = fruits.length;
text = "<ul>";
for (i = 0; i < fLen; i++) {
    text += "<li>" + fruits[i] + "</li>";
}
text += "</ul>";
document.getElementById("demo").innerHTML = text;
</script>
```

The Array.forEach Function

- Example:

```
<p id="demo2"></p>
<script>
var fruits, text;
fruits = ["Banana", "Orange", "Apple", "Mango"];
text = "<ul>";
fruits.forEach(myFunction);
text += "</ul>";
document.getElementById("demo2").innerHTML = text;
function myFunction(value) {
    text += "<li>" + value + "</li>";
}
</script>
```

Adding Array Elements (1/2)

- Using the **push()** method:

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.push("Melon");     // adds a new element (Melon) to fruits
```

- Using the **length** property:

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits[fruits.length] = "Lemon";     // adds a new element (Lemon) to fruits
```

Adding Array Elements (2/2)

- Declaring new elements with high index values:

```
var fruits = ["Banana", "Orange", "Apple", 
"Mango"];
fruits[6] = "Melon";      // adds a new 
element (Melon) to fruits, but elements at 
indexes 4 and 5 are undefined.
```

Removing Array Elements (1/3)

- Using the **pop()** method:

```
var fruits = ["Banana", "Orange", "Apple",
"Mango"];
fruits.pop();   // removes the last element
(Mango) from fruits
```

- Using the **shift()** method:

```
var fruits = ["Banana", "Orange", "Apple",
"Mango"];
fruits.shift();   // removes the first
element (Banana) from fruits
```

Removing Array Elements (2/3)

- Using the **splice()** method:

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.splice(2,1);   // Starting at index position 2, removes one element. It removes the element: Apple from fruits
```

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.splice(0,2);   // Starting at index position 0, removes two elements. It removes elements: Banana and Orange from fruits
```

Removing Array Elements (3/3)

- Using the **indexOf()** method:

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.splice(fruits.indexOf('Orange'),1);
 // Find the index position of Orange then it removes one element from fruits (Orange is removed)
```

The for/in Statement

- The **for/in** statement loops through the properties of an object :

```
var person = {fname:"John", lname:"Doe", age:25};
var text = "";
var x;
for (x in person) {
    text += person[x];
}
```

The while Statement

- The **while** statement syntax:

```
while (condition) {
    code block to be executed
}
```

- Example:

```
var text = "";
var i = 0;
while (i < 10) {
    text += "<br>The number is " + i;
    i++;
}
```

The do/while Statement

- The **do/while** statement syntax:

```
do {
    code block to be executed
}
while (condition);
```

- Example:

```
var text = "";
var i = 0;
do {  text += "<br>The number is " + i;
      i++;
}
while (i < 10)
```