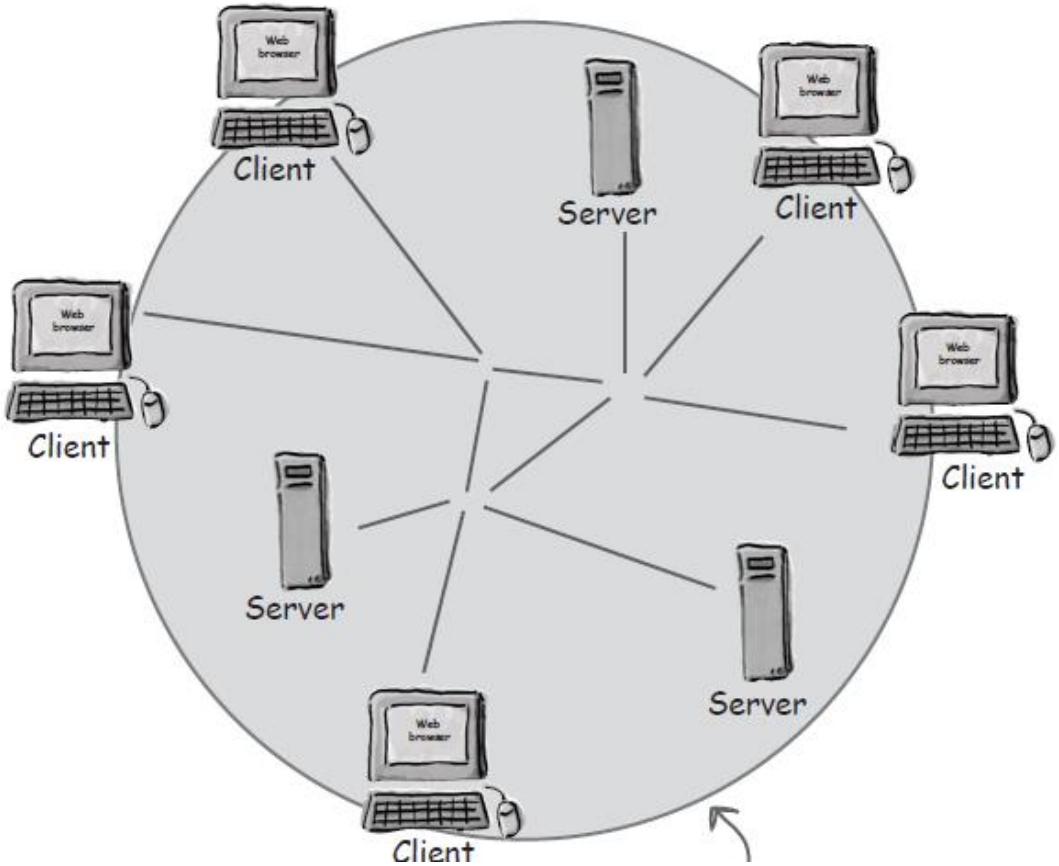
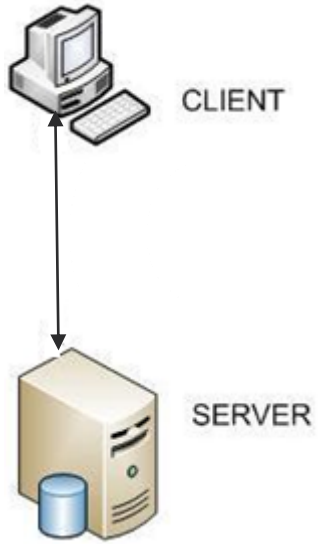
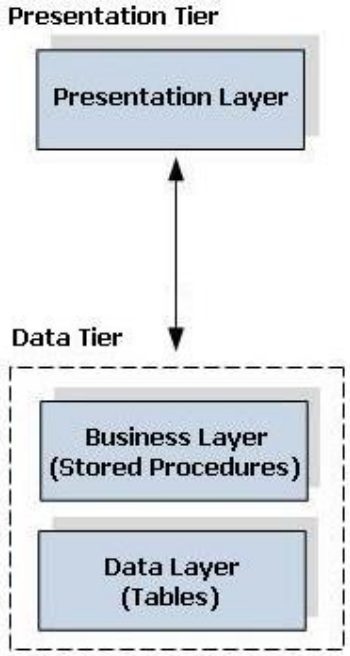

CSS
CORP
JS
Java Script

World Wide Web

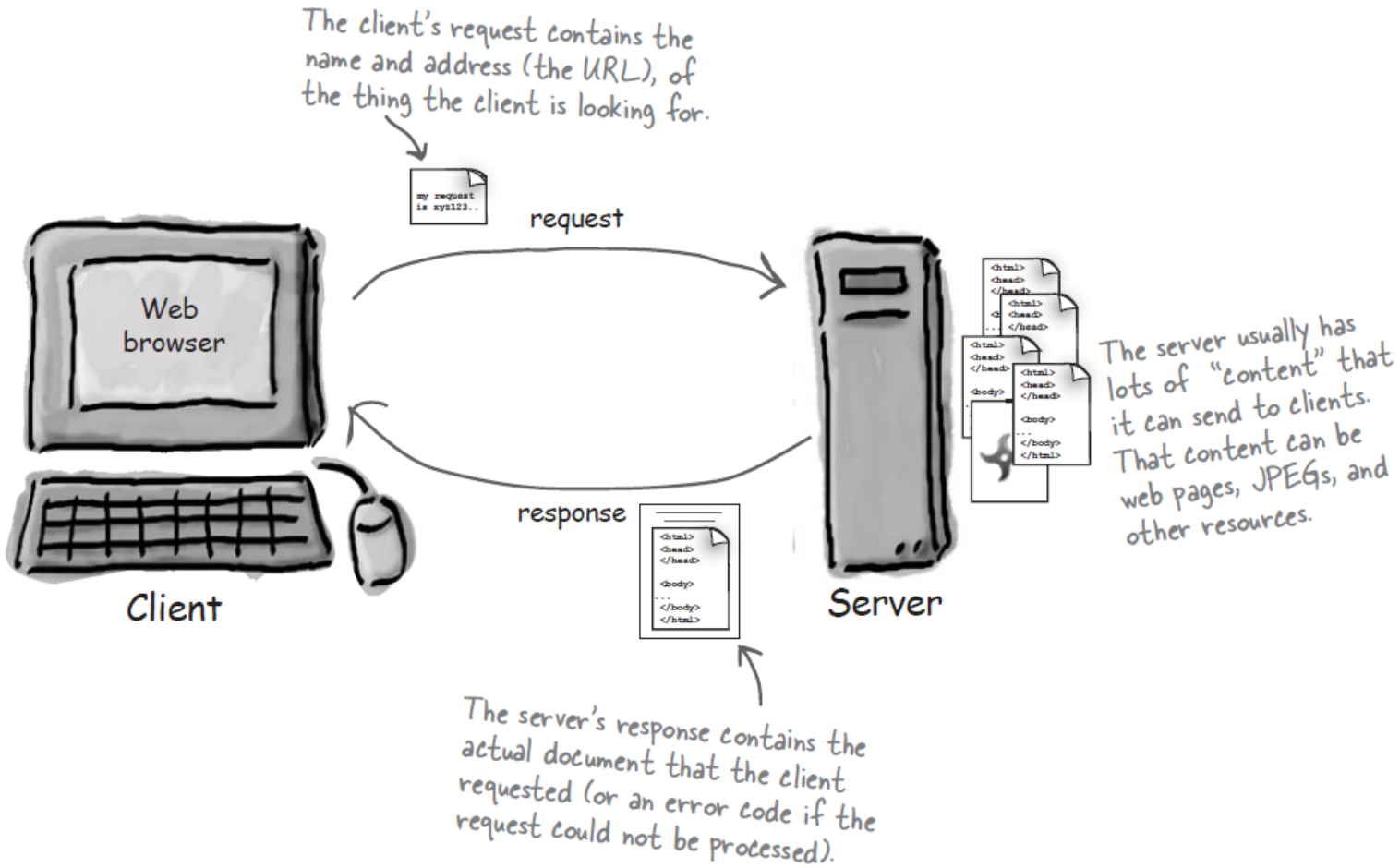


Client Server Architecture

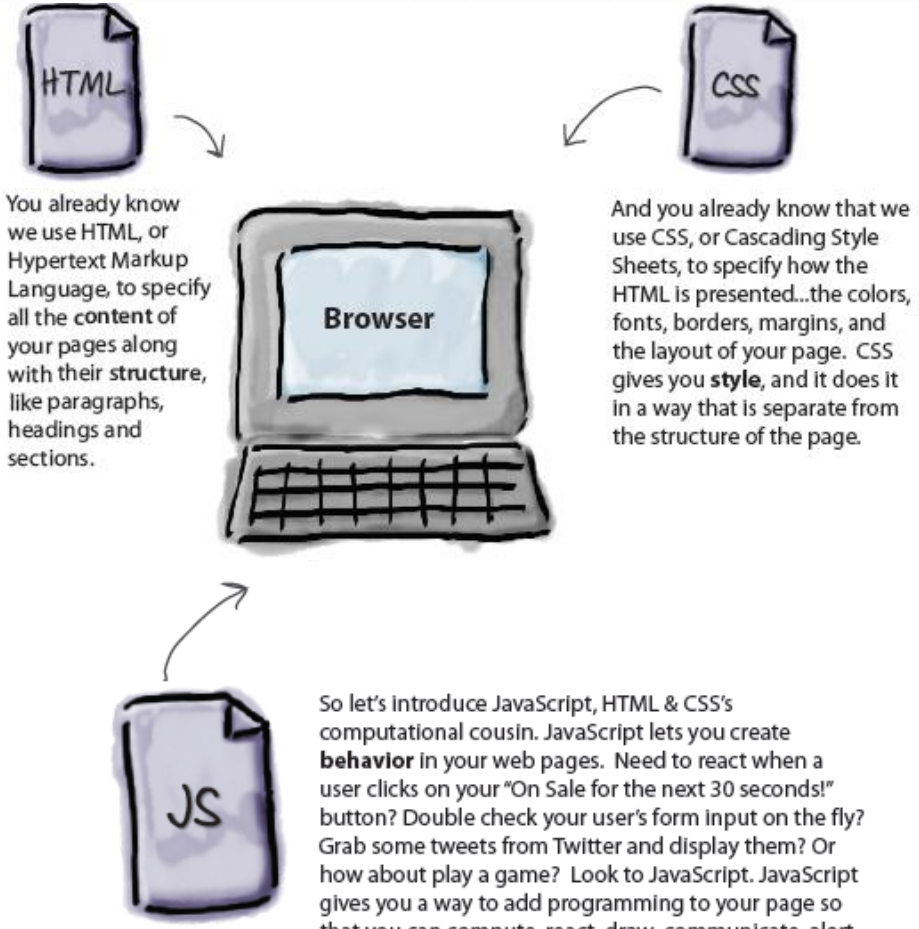




Web Server & Web Browser



Client Side Scripting using JavaScript

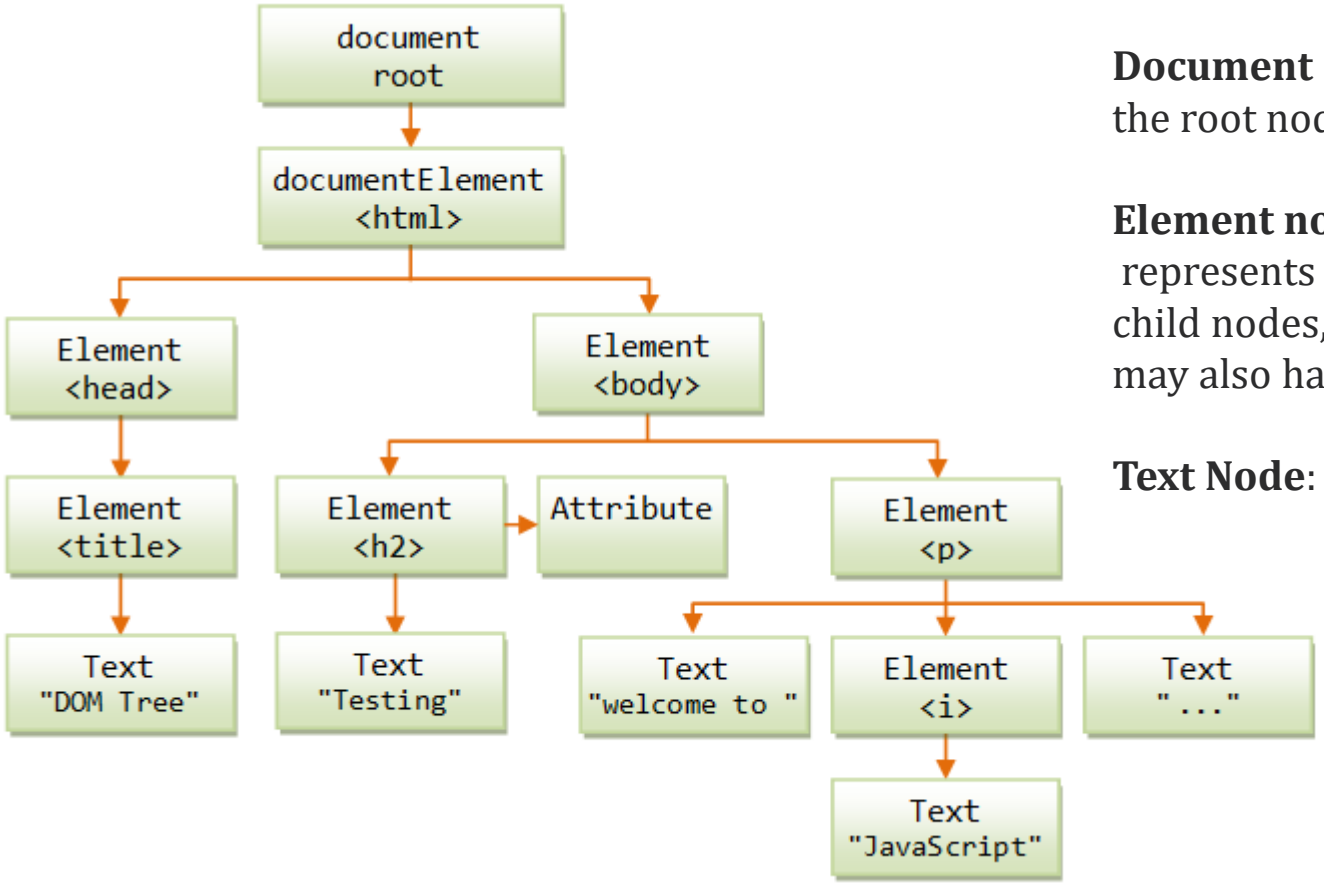


Version History*

Version	Release date	Equivalent to	Netscape Navigator	Mozilla Firefox	Internet Explorer	Opera	Safari	Google Chrome
1.0	March 1996		2.0		3.0			
1.1	August 1996		3.0					
1.2	June 1997		4.0-4.05			3[105]		
1.3	October 1998	ECMA-262 1st + 2nd edition	4.06-4.7x		4.0	5[106]		
1.4			Netscape Server			6		
1.5	November 2000	ECMA-262 3rd edition	6.0	1.0	5.5 (JScript 5.5), 6 (JScript 5.6), 7 (JScript 5.7), 8 (JScript 5.8)	7.0	3.0-5	1.0-10.0.666
1.6	November 2005	1.5 + array extras + array and string generics + E4X		1.5				
1.7	October 2006	1.6 + Pythonic generators + iterators + let		2.0				28.0.1500.95
1.8	June 2008	1.7 + generator expressions + expression closures		3.0		11.50		
1.8.1		1.8 + native JSON support + minor updates		3.5				
1.8.2	June 22, 2009	1.8.1 + minor updates		3.6				
1.8.5	July 27, 2010	1.8.2 + new features for ECMA-262 Edition 5 compliance.		4.0				

*(From Wiki Page)

DOM tree



A DOM-tree comprises

Document Node
the root node representing the entire HMTL document.

Element node
represents an HTML element (or tag). An element node may have child nodes, which can be either element or text node. Element node may also have attributes.

Text Node: contains the text content of an element.

Where to place JavaScript?

- Within HTML page
- In separate .js file

Within HTML page

```
<!DOCTYPE html>
<html>

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

<body>

<h1>My Web Page</h1>

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

<button type="button" onclick="myFunction()">Try it</button>

</body>
</html>
```

```
<!DOCTYPE html>
<html>
<body>

<h1>My Web Page</h1>

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

<button type="button" onclick="myFunction()">Try it</button>

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

</body>
</html>
```

External .js file

myScript.js

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

```
<!DOCTYPE html>
<html>
<body>

<h1>External JavaScript</h1>

<p id="demo">A Paragraph.</p>

<button type="button" onclick="myFunction()">Try it</button>

<p><strong>Note:</strong> myFunction is stored in an external file called
"myScript.js".</p>

<script src="myScript.js"></script>

</body>
</html>
```

JavaScript Programming

- Display methods
- Variables
- Control Structures
- Loops
- Operators
- Arrays
- Data Structures
- Functions
- Event Handling
- Objects
- Object Oriented js

Interacting with users

`alert(string)`: Pop-up a box to alert user for important information. The user will have to click "OK" to proceed. The `alert()` returns nothing (or `undefined`).

`prompt(string, defaultValue)`: Pop-up a box to prompt user for input, with an optional *`defaultValue`*. The `prompt()` returns the user's input as a string. For example,

```
var number1 = prompt('Enter the first integer:');
var number2 = prompt('Enter the second integer:');
alert('The sum is ' + number1 + number2);                  // Concatenate two strings
alert('The sum is ' + (Number(number1) + Number(number2))); // Add two numbers
```

`confirm(string)`: Pop-up a box and ask user to confirm some information. The user will have to click on "OK" or "Cancel" to proceed. The `confirm()` which returns a boolean value. For example,

```
var isFriday = confirm("Is it Friday?");  // Returns a boolean
if (isFriday) {
   alert("Thank God, it's Friday!");
} else {
   alert('Not a Friday');
}
```

`document.write(string)`, `document.writeln(string)`: Write the specified string to the current document. The `writeln()` (write-line) writes a newline after the string, while `write()` does not. Take note that browsers ignores extra white spaces (blanks, tabs, newlines) in an HTML document, and treat multiple white spaces as a single blank character. You need to write a `<br>` or `<p>...</p>` tag to ask the browser to display a line break.

`console.log(value)`: write to the JavaScript console, used mainly for debugging.

Display Methods

JavaScript does NOT have any built-in print or display function

- Writing into an alert box, using **window.alert()**
- Writing into the HTML output using **document.write()**
- Writing into an HTML element, using **innerHTML**
- Writing into the browser console, using **console.log()**

Display Method 1

- Writing into an alert box, using **window.alert()**
  - window.alert("Welcome");

```
<!doctype html>
    <html>
        <meta charset="utf-8">
         <body>
          <script>
            window.alert("Welcome");
          </script>
        </body>
    </html>
```

Finding & Selecting elements on DOM

Function	Description	Example

`document.getElementById(anId)`	Returns the element with the given unique `id`.	`<input type="text" id="foo">`
		`var elm = document.getElementById("foo");`
		`var input = elm.value;`

`document.getElementsByTagName(aTagName)`	Returns an array of elements with the given tag name.	`<input type="text">`
		`var elms = document.getElementByTagName("input");`
		`var input = elms[0].value;`

`document.getElementsByClassName(aClassName)`	Returns an array of elements with the given `class` attribute name.	`<input type="text" class="bar">`
		`var elms = document.getElementByClassName("bar");`
		`var input = elms[0].value;`

`document.getElementsByName(aName)`	Returns an array of elements with the given `name` attribute.	`<input type="checkbox" name="gender" value="m">Male`
		`<input type="checkbox" name="gender" value="f">Female`
		`var x = document.getElementsByName("gender");`
		`for (var i = 0; i < x.length; ++i) {`
		`  if (x[i].checked) {`
		`  value = x[i].value;`
		`  break;`
		`  }`
		`}`

Display Method 2

- Writing into the HTML output using **document.write()**
  - document.write("Welcome in HTML page");

```
<!doctype html>
    <html>
        <meta charset="utf-8">
         <body>
          <script>
            document.write("Welcome displayed in HTML page");
          </script>
        </body>
    </html>
```

Inner HTML property

- You can access and modify the content of an element via the "innerHTML" property, which contains all the texts (includes nested tags) within this element

Display Method 3

- Writing into an HTML element, using **innerHTML**
- You can access and modify the content of an element via the "innerHTML" property, which contains all the texts (includes nested tags) within this element
    - document.getElementById("msg").innerHTML = "Hello "+ yourName;

Display Method 3 – cont…

```
<!doctype html>
    <html>
        <meta charset="utf-8">
         <body>
            <p id="msg"></p>
          <script>
            var yourName =prompt("what is your name?");

            if (yourName != null) {
                document.getElementById("msg").innerHTML = "Hello "+ yourName;
                 }
            else
            {
                alert("Please enter a name next time");
            }
          </script>
        </body>
    </html>
```

Display Method 4

- Writing into the browser console, using **console.log()**
  - console.log("Welcome message in console");

```
<!doctype html>
    <html>
        <meta charset="utf-8">
         <body>
            <p id="msg"></p>
          <script>
            console.log("Welcome displayed in console log");
          </script>
        </body>
    </html>
```

Activate the browser console with F12, and select "Console" in the menu to see the result

Comments

- **// : single line comment**
- **/* */ : multiline comment**

JavaScript Values

- The JavaScript syntax defines two types of values
  - Fixed values - literals
  - Variable values - Variables

Literals

- Numbers with or without decimals

  Example : document.getElementById("demo").innerHTML = 10.50;

- **Strings** are text, written within double or single quotes

  Example : document.getElementById("demo").innerHTML = ‘javascript';

Identifiers

- All JavaScript **variables** must be **identified** with **unique names** called **identifiers**.
- Identifiers can be
    - Short descriptive names (age, sum, Volume).

The general rules for constructing names for variables (unique identifiers) are:

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

Variables

- **Variables** are used to **store** data values

  var is the keyword used to declare variables
  = is the symbol to assign values

```
<!doctype html>
    <html>
        <meta charset="utf-8">
         <body>
            <p id="msg"></p>
          <script>
            var yourName =prompt("what is your name?");

            if (yourName != null) {
                document.getElementById("msg").innerHTML = "Hello "+ yourName;
                }
            else
            {
                alert("Please enter a name next time");
            }
          </script>
        </body>
    </html>
```

Variables – cont…

- Variables can hold data belonging to any data type
- Many variables can be declared in one statement
- A variable declared without a value will have the value undefined

Example : 1

```
var x = 5;
var y = 6;
var z = x + y;
document.getElementById("demo").innerHTML = z;
```

Example : 2

```
var person = “APJ", carName = 'swift', price = 200;
document.getElementById("demo").innerHTML = carName;
```

Basic Data Types

There are 7 basic types in JavaScript.

- `number` for numbers of any kind: integer or floating-point.
- `string` for strings. A string may have one or more characters, there's no separate single-character type.
- `boolean` for `true` / `false`.
- `null` for unknown values – a standalone type that has a single value `null`.
- `undefined` for unassigned values – a standalone type that has a single value `undefined`.
- `object` for more complex data structures.
- `symbol` for unique identifiers.

Data Types - Examples

```
var length = 16;                                    // Number
var lastName = “Kapoor";                  // String
var cars = [“Merc", "Volvo", "BMW"];   // Array
var x = {firstName:“Java", lastName:“Script"};    // Object
var flag = true;                                    //Boolean
```

Dynamic Types

```
var x;                    // Now x is undefined
var x = 5;                // Now x is a Number
var x = “JavaScript";     // Now x is a String
```

Number

Numbers

- Can be with or without decimals
- Extra large or extra small numbers can be written with scientific (exponent) notation
  - var x = 123e5; // 12300000
  - var y = 123e-5; // 0.00123
- Stores numbers in 64 bits
  - number (the fraction) is stored in bits 0 to 51
  - exponent in bits 52 to 62
  - sign in bit 63
- Integers are accurate up to 15 digits
- Numeric constants are considered as hexadecimal if they are preceded by 0x

HexaDecimal

```
<script>
function myFunction() {
   document.getElementById("demo").innerHTML = "0xFF = " + 0xFF; //255
}
</script>
```

- By default numbers are considered as base 10 decimals
- toString() method can be used to output numbers as base 16 (hex), base 8 (octal), or base 2 (binary)
- Example

```
var myNumber = 128;
myNumber.toString(16);    // returns 80
myNumber.toString(8);     // returns 200
myNumber.toString(2);     // returns 10000000
```

Infinity

Infinity / -Infinity is the value JavaScript will return when a number outside the largest possible number is calculated

Example : document.write(10/0);

NaN - Not a Number

- NaN is a JavaScript reserved word indicating that a value is not a number
- Trying to do arithmetic with a non-numeric string will result in NaN (Not a Number)

Example : **var x = 100 / "Apple";**

- If the string contains a numeric value , the result will be a number

Example : document.getElementById("demo").innerHTML = 100 / "10";

- Function **isNaN()** to find out if a value is a number

Example: var x = 100 / "Apple";
document.getElementById("demo").innerHTML = isNaN(x);

String

String

- JavaScript string simply stores a series of characters
- Can be any text inside quotes
- The length of a string is found in the built in property **length**
    - Example :

```
var txt = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
document.getElementById("demo").innerHTML = txt.length;
```

Escape Sequences

Code	Outputs
\'	single quote
\"	double quote
\\	backslash
\n	new line
\r	carriage return
\t	tab
\b	backspace
\f	form feed

String Methods

```
var msg = "JavaScript is a high-level, dynamic programming language";
 document.write("type :" + typeof msg);
 document.write(msg.length,"<br/>");
 document.write(msg.indexOf("dynamic"),"<br/>");
 document.write("sliced data : "+ msg.slice(28,35),"<br/>");
 document.write("substring data : "+ msg.substr(28,7),"<br/>");
 document.write(msg.fontcolor("red"));
// HTML wrapper methods
 document.write(msg.fontsize(".25em").sub().strike().italics());
document.write(msg.toLowerCase(msg.replace("dynamic","best"),"<br/>"));
```

typeOf Operator

Example:
var msg = "JavaScript is a high-level, dynamic”;
document.write("type :" + **typeof** msg);

- typeof NaN returns number

String Methods – cont….

Method	Description
charAt()	Returns the character at the specified index (position)
charCodeAt()	Returns the Unicode of the character at the specified index
concat()	Joins two or more strings, and returns a new joined strings
endsWith()	Checks whether a string ends with specified string/characters
fromCharCode()	Converts Unicode values to characters
includes()	Checks whether a string contains the specified string/characters
indexOf()	Returns the position of the first found occurrence of a specified value in a string
lastIndexOf()	Returns the position of the last found occurrence of a specified value in a string
localeCompare()	Compares two strings in the current locale
match()	Searches a string for a match against a regular expression, and returns the matches
repeat()	Returns a new string with a specified number of copies of an existing string
replace()	Searches a string for a specified value, or a regular expression, and returns a new string where the specified values are replaced
search()	Searches a string for a specified value, or regular expression, and returns the position of the match

String Methods – cont….

slice()	Extracts a part of a string and returns a new string
split()	Splits a string into an array of substrings
startsWith()	Checks whether a string begins with specified characters
substr()	Extracts the characters from a string, beginning at a specified start position, and through the specified number of character
substring()	Extracts the characters from a string, between two specified indices
toLocaleLowerCase()	Converts a string to lowercase letters, according to the host's locale
toLocaleUpperCase()	Converts a string to uppercase letters, according to the host's locale
toLowerCase()	Converts a string to lowercase letters
toString()	Returns the value of a String object
toUpperCase()	Converts a string to uppercase letters
trim()	Removes whitespace from both ends of a string
valueOf()	Returns the primitive value of a String object

Explicit Type Conversion

- Converting a number to a string

  Simply concatenate the number with an empty string, e.g., "" + 5 gives "5“

- Converting a string to a number
  - Use built-in functions parseInt(string), parseFloat(string) or Number(string) to convert a string which contains a valid number
  - For example, parseInt("55") gives 55
  - parseInt(55.66) gives 55
  - parseInt("55.66") gives 55
  - parseFloat("55.66") gives 55.66
  - parseInt("55px") gives 55
  - but parseInt("Hello") gives NaN.

- Converting a float to an integer
  - Use parseInt() (e.g., parseInt(55.66) gives 55)
  - built-in mathematical functions such as Math.round(), Math.ceil() or Math.floor().

Array

- An array is a special variable, which can hold more than one value at a time
- It is a special type of object

```
<script>
var cars = [“Merc","Volvo","BMW"];
document.getElementById("demo").innerHTML = cars[0];
</script>
```

Array Methods

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];

document.getElementById("demo").innerHTML = fruits.toString();
document.getElementById("demo").innerHTML = fruits.join(" * ");
fruits.pop();                              // Removes the last element ("Mango") from fruits
var x = fruits.pop();              // the value of x is "Mango“
fruits.push("Kiwi");                       //  Adds a new element ("Kiwi") to fruits at the end
var x = fruits.push("Kiwi");   //  returns the new array length
fruits.shift();                    // Removes the first element "Banana" from fruits
fruits.unshift("Lemon");       // Adds a new element "Lemon" to fruits
fruits[0] = "Kiwi";                  // Changes the first element of fruits to "Kiwi“
fruits[fruits.length] = "Kiwi";          // Appends "Kiwi" to fruit
delete fruits[0];                        // Changes the first element in fruits to undefined
```

Array Methods

Purpose	Example

Add one item to the *end* using `array.length`	`var a = [0, 'a', 'b'];`
	`a[a.length] = 3;`
	`console.log(a.length); // 4`
	`console.log(a); // [0, "a", "b", 3]`

Add one or items to the *end* using `push()`.
`push()` returns the resultant length of the array.
	`var a = [0, 'a', 'b'];`
	`console.log(a.push(1, 'c')); // 5`
	`console.log(a.length); // 5`
	`console.log(a); // [0, "a", "b", 1, "c"]`

Add one or items to the *beginning* using `unshift()`.
`unshift()` returns the resultant length of the array.
	`var a = [0, 'a', 'b'];`
	`console.log(a.unshift(-2, -1, 'c')); // 6`
	`console.log(a.length); // 6`
	`console.log(a); // [-2, -1, "c", 0, "a", "b"]`

Remove and return the *last* item using `pop()`	`var a = [0, 'a', 'b'];`
	`console.log(a.pop()); // b`
	`console.log(a.length); // 2`
	`console.log(a); // [0, "a"]`

Remove and return the *first* item using `shift()`	`var a = [0, 'a', 'b'];`
	`console.log(a.shift()); // 0`
	`console.log(a.length); // 2`
	`console.log(a); // ["a", "b"]`

undefined

Variable without a value, has the value undefined

Example:

```
<script>
var person;
document.getElementById("demo").innerHTML =person + "<br>" + typeof person;
</script>
```

*In Javascript , null is nothing

Variables – cont…

- When re-declared a JavaScript variable will not lose its value

Example :

```
<script>
var carName = “Punto";
var carName;
document.getElementById("demo").innerHTML = carName;
</script>
```

Concatenation

```
document.write("5+4=", 5+4);
```

output : 5 +4 = 9

```
document.write("5+4=“+5+4);
```

output : 5 + 4 = 54

Arithmetic Operators

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus
++	Increment
--	Decrement

Logical 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

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

Type Operators

Operator	Description
typeof	Returns the type of a variable
instanceof	Returns true if an object is an instance of an object type

Conditional Control
&
Loops

Conditional Control : If .. Else & Switch

- Use **if** to specify a block of code to be executed, if a specified condition is true
- Use **else** to specify a block of code to be executed, if the same condition is false
- Use **else if** to specify a new condition to test, if the first condition is false

Syntax	

`if (condition) {`
`  trueBlock;`
`}`
	`if (day === 'sat' || day === 'sun') {`
	`  alert('Super weekend!');`
	`}`

`if (condition) {`
`  trueBlock;`
`} else {`
`  falseBlock;`
`}`
	`if (day === 'sat' || day === 'sun') {`
	`  alert('Super weekend!');`
	`} else {`
	`  alert('It is a weekday...');`
	`}`

`variable = (condition) ? trueValue : falseValue;`
Same as
`if (condition) { variable = trueValue; }`
`else { variable = falseValue; }`
	`var max = (a > b) ? a : b;`
	`var abs = (a >= 0) ? a : -a;`

```
if (condition1) {
  block1;
} elseif (condition2) {
  block2;
} elseif (...) {
......
} else {
  elseBlock;
}
```

```
if (day === 'sat' || day === 'sun') {
   alert('Super weekend!');
} else if (day === 'fri') {
   alert("Thank God, it's Friday!");
} else {
   alert('It is a weekday...');
}
```

```
switch (expression) {
  case value1:
    statements; break;
  case value2:
    statements; break;
  ......
  ......
  default:
    statements;
}
```

```
switch (day) {
   case 'sat': case 'sun':
      alert('Super weekend!'); break;
   case 'mon': case 'tue': case 'wed': case 'thu':
      alert('It is a weekday...'); break;
   case 'fri':
      alert("Thank God, it's Friday"); break;
   default:
      alert("You are on earth?! Aren't you?");
}
```

If .. Else

```
<! doctype html>
<html>
    <body>
    <script>
        var age= prompt("enter your age");

        if (age >= 18) {
            alert("Eligible to vote");
        }
        else if((age != null) && (age <= 17)){
            alert("You are not eligible to vote");
        }
        else
        {
        alert("enter a valid age next time");
        }

    </script>
    </body>
 </html>
```

If .. Else

```
<label for="weather">Select the weather type today: </label>
<select id="weather">
  <option value="">--Make a choice--</option>
  <option value="sunny">Sunny</option>
  <option value="rainy">Rainy</option>
  <option value="snowing">Snowing</option>
  <option value="overcast">Overcast</option>
</select>

<p></p>
```

If .. Else

```
var select = document.querySelector('select');
var para = document.querySelector('p');

select.addEventListener('change', setWeather);

function setWeather() {
  var choice = select.value;

  if (choice === 'sunny') {
    para.textContent = 'It is nice and sunny outside today. Wear shorts! Go to the beach, or the park, and get
  } else if (choice === 'rainy') {
    para.textContent = 'Rain is falling outside; take a rain coat and a brolly, and don\'t stay out for too lon
  } else if (choice === 'snowing') {
    para.textContent = 'The snow is coming down — it is freezing! Best to stay in with a cup of hot chocolate,
  } else if (choice === 'overcast') {
    para.textContent = 'It isn\'t raining, but the sky is grey and gloomy; it could turn any minute, so take a
  } else {
    para.textContent = '';
  }
```

Switch

- Use **switch** to specify many alternative blocks of code to be executed

```
<!doctype html>
    <html>
        <body>
            <p id="demo"></p>
            <script>

switch (new Date().getDay()) {
    case 0:
        day = "Sunday";
        break;
    case 1:
        day = "Monday";
        break;
    case 2:
        day = "Tuesday";
        break;
    case 3:
        day = "Wednesday";
        break;
    case 4:
        day = "Thursday";
        break;
    case 5:
        day = "Friday";
        break;
    case 6:
        day = "Saturday";
}
document.getElementById("demo").innerHTML = "Today is " + day;
</script>
</body>
</html>
```

Looping Structures

- **for** - loops through a block of code a number of times
- **for/in** - loops through the properties of an object
- **while** - loops through a block of code while a specified condition is true
- **do/while** - also loops through a block of code while a specified condition is true

Syntax	

`while (test) {`
`  trueBlock;`
`}`
	`// Sum from 1 to 100`
	`var sum = 0, number = 1;`
	`while (number <= 100) {`
	`  sum += number;`
	`}`

`do {`
`  trueBlock;`
`} while (test);`
	`// Sum from 1 to 100`
	`var sum = 0; number = 1;`
	`do {`
	`  sum += number;`
	`}`

`for (initialization; test; post-processing) {`
`  trueBlock;`
`}`
	`// Sum from 1 to 100`
	`var sum = 0;`
	`for (var number = 1; number <= 100; number++) {`
	`  sum += number;`
	`}`

For

for (**statement 1**; **statement 2**; **statement 3**)
{
  code block to be executed
}
Statement 1 is executed before the loop (the code block) starts
Statement 2 defines the condition for running the loop (the code block)
Statement 3 is executed each time after the loop (the code block) has been executed

Example:

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

For – cont…

```
var cars = ["BMW", "Volvo", "Saab", "Ford"];
var i = 2;
var len = cars.length;
var text = "";
for (; i < len; i++)
 {
   text += cars[i] + "<br>";
}
```

```
var i = 0;
var len = cars.length;
for (; i < len; )
 {
   text += cars[i] + "<br>";
   i++;
}
```

For/In Loop

```
var person = {fname:"John", lname:"Doe", age:25};

var text = "";
var x;
for (x in person) {
   text += person[x];
}
```

While Loop

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

Example :

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

Do..While Loop

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

Example :

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

Break Statement

- Break statement can be used to jump out of a loop
- Break statement breaks the loop and continues executing the code after the loop

```
for (i = 0; i < 10; i++)
{
   if (i === 3) { break; }
   text += "The number is " + i + "<br>";
}
```

Continue Statement

- **Continue statement** breaks one iteration and continues with the next iteration in the loop

Example:

```
var text = "";
var i;
for (i = 0; i < 10; i++) {
   if (i === 3) { continue; }
   text += "The number is " + i + "<br>";
}
document.getElementById("demo").innerHTML = text;
```

Labels

- label: statements

- Example

break *labelname*;

continue *labelname*;

```
var cars = ["BMW", "Volvo", "Saab", "Ford"];
list:
 {
   text += cars[0] + "<br>";
   text += cars[1] + "<br>";
   text += cars[2] + "<br>";
   break list;
   text += cars[3] + "<br>";
   text += cars[4] + "<br>";
   text += cars[5] + "<br>";
}
```

Functions

Function

```
1  function sayHi() {
2    alert( "Hello" );
3  }
```

Function Declaration

```
1  let sayHi = function() {
2    alert( "Hello" );
3  };
```

Function Expression

Function

```
function functionName(parameters)
{
  code to be executed
}
```

Example:

```
function myFunction(a, b)
 {
    return a * b;
}
```

```
let r = myFunction(10,20);
```

Function

```
function sayHi() {
  alert( "Hello" );
}

alert( sayHi ); // shows the function code
```

```
function sayHi() {   // (1) create
  alert( "Hello" );
}

let func = sayHi;    // (2) copy

func(); // Hello     // (3) run the copy (it works)!
sayHi(); // Hello    //     this still works too (why wouldn't it)
```

Anonymous Functions

- A JavaScript function can also be defined using an **expression**.
- A function expression can be stored in a variable

- Example

```
var x = function (a, b) {return a * b};
var z = x(4, 3);
```

Self-Invoking Functions

```
<script>
(function () {
   document.getElementById("demo").innerHTML = "Hello! I called myself";
})();
</script>
```

Function Parameters and Arguments

- Function **parameters** are the **names** listed in the function definition.
- Function **arguments** are the real **values** passed to (and received by) the function.

Rules

- JavaScript function definitions do not specify data types for parameters
- JavaScript functions do not perform type checking on the passed arguments
- JavaScript functions do not check the number of arguments received
- If a function is called with **missing arguments** (less than declared), the missing values are set to: **undefined**

Arguments Object

- Have built-in object called the arguments object
- Contains an array of the arguments used when the function was called

- Example

```
x = sumAll(1, 123, 500, 115, 44, 88);

function sumAll() {
   var i, sum = 0;
   for (i = 0; i < arguments.length; i++) {
      sum += arguments[i];
   }
   return sum;
}
```

Function as a Object Method

- Functions can be defined as object methods

```
<script>
var myObject = {
   firstName:“Java",
   lastName: “Script",
   fullName: function() {
      return this.firstName + " " + this.lastName;
   }
}
document.getElementById("demo").innerHTML = myObject.fullName();
</script>
```

Invoking a Function with a Function Constructor

- A function invocation is preceded with the **new** keyword, it is a constructor invocation.

```
<script>
function myFunction(arg1, arg2) {
   this.firstName = arg1;
   this.lastName  = arg2;
}

var x = new myFunction("Java",“Script")
document.getElementById("demo").innerHTML = x.firstName;
</script>
```

Variable Lifetime

- Global variables live as long as your application lives
- Local variables have short lives

```
<!DOCTYPE html>
<html>
<body>

<p>Counting with a global variable.</p>

<button type="button" onclick="myFunction()">Count!</button>

<p id="demo">0</p>

<script>
var counter = 0;

function add() {
    return counter += 1;
}

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

</body>
</html>
```

JavaScript Nested Functions

- Nested functions have access to the scope "above" them

```
<!DOCTYPE html>
<html>
<body>

<p>Counting with a local variable.</p>

<p id="demo">0</p>

<script>
document.getElementById("demo").innerHTML = add();
function add() {
    var counter = 0;
    function plus() {counter += 1;}
    plus();
    return counter;
}
</script>

</body>
</html>
```

Callback Functions

```
function ask(question, yes, no) {
  if (confirm(question)) yes()
  else no();
}

function showOk() {
  alert( "You agreed." );
}

function showCancel() {
  alert( "You canceled the execution." );
}

// usage: functions showOk, showCancel are passed as arguments to ask
ask("Do you agree?", showOk, showCancel);
```

Event Handling

Handling Events

- JavaScript are often event-driven

- Events:
  - click: generated when the user clicks on an HTML element.
  - mouseover, mouseout : generated when the user positions the mouse pointer inside/away from the HTML element.
  - load, unload: generated after the browser loaded a document, and before the next document is loaded, respectively.
- Event Handler:
  - A piece of program that fires in response to a certain user's or browser's action
  - The event handler is typically attached to the target HTML tag, e.g.,

Events

Event Name	Event Handler	Description	HTML Element
click	onclick	User clicks on the component.	
submit	onsubmit	User clicks the "submit" button.	<form>, <input type="submit">
reset	onreset	User clicks the "reset" button.	<form>, <input type="reset">
select	onselect	User selects text in a text box or text area.	<textarea>, <input type="text">
keypress	onkeypress	User holds down a key.	document, image, link, textarea
keydown keyup	onkeydown onkeyup	User presses/releases a key.	
mousedown mouseup	onmousedown onmouseup	User presses/releases a mouse button.	button, document, link
mouseover mouseout	onmouseover onmouseout	User moves the mouse pointer at/away from a link or hot spot.	
mousemove	onmousemove	User moves the mouse pointer	
load	onload	When the page is loaded into the window.	<body>, <frameset>, <img>
unload	onunload	When another page is about to be loaded.	<body>, <frameset>
blur	onblur	When a particular form element losses focus. E.g., after the element is selected, and the user clicks somewhere or hit the tag key.	
change	onchange	Same as onblur, but the elements must be changed.	
focus	onfocus	Same as onblur, but the element gains focus.	
drapdrop	ondrapdrop	User drags and drops something (e.g., a file) onto the navigator window.	window
move resize	onmove onresize	User moves/resizes the window	window, frame
abort	onabort	Users stops or aborts an image from loading.	<img>
error	onerror	When a JavaScript or image error occurs while loading a document or an image.	<img>

Form Validation

Validation of form controls

```
<!DOCTYPE html>
<html>
<head>
<script>
function validateForm() {
    var x = document.forms["myForm"]["fname"].value;
    if (x == null || x == "") {
        alert("Name must be filled out");
        return false;
    }
}
</script>
</head>
<body>

<form name="myForm" action="demo_form.asp"
onsubmit="return validateForm()" method="post">
Name: <input type="text" name="fname">
<input type="submit" value="Submit">
</form>

</body>
</html>
```

Validation of form controls – cont…

```
<!DOCTYPE html>
<html>
<body>

<h1>JavaScript Can Validate Input</h1>

<p>Please input a number between 1 and 10:</p>

<input id="numb">

<button type="button" onclick="myFunction()">Submit</button>

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

Validation of form controls – cont…

```
<script>
function myFunction() {
    var x, text;

    // Get the value of the input field with id="numb"
    x = document.getElementById("numb").value;

    // If x is Not a Number or less than one or greater than 10
    if (isNaN(x) || x < 1 || x > 10) {
        text = "Input not valid";
    } else {
        text = "Input OK";
    }
    document.getElementById("demo").innerHTML = text;
}
</script>

</body>
</html>
```

Object Oriented Programming

Object Basics

- An object is a collection of related data and/or functionality
  - variables - properties
  - functions - methods

Object

```
<script>

var person = {
   firstName : “MK",
   lastName  : “Gandhi",
   age: 78,
   eyeColor : “Hazel"
};

document.getElementById("demo").innerHTML = person.firstName + " is " + person.age + " years old.";
</script>
```

Object – sample 2

```
var person = {
  name: ['Bob', 'Smith'],
  age: 32,
  gender: 'male',
  interests: ['music', 'skiing'],
  bio: function() {
    alert(this.name[0] + ' ' + this.name[1] + ' is ' + this.age + ' years old. He likes ' + this.interests[0]
  },
  greeting: function() {
    alert('Hi! I\'m ' + this.name[0] + '.');
  }
};
```

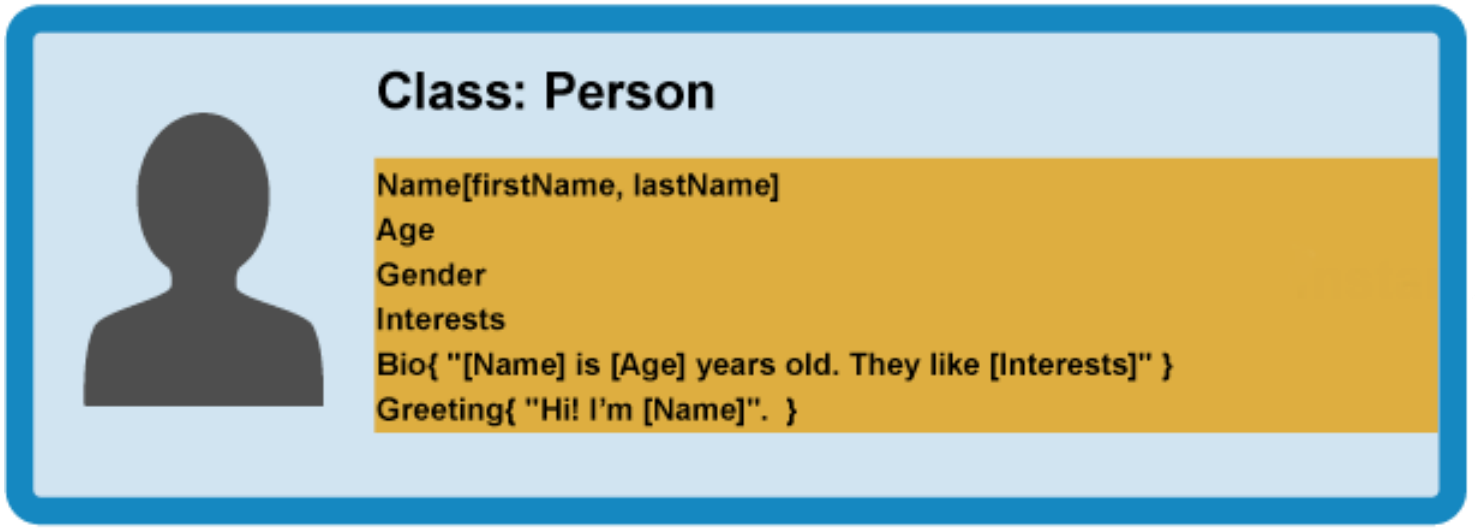
```
person.name[0]
person.age
person.interests[1]
person.bio()
person.greeting()
```

this

```
var person = {
  name: ['Bob', 'Smith'],
  age: 32,
  gender: 'male',
  interests: ['music', 'skiing'],
  bio: function() {
    alert(this.name[0] + ' ' + this.name[1] + ' is ' + this.age + ' years old. He likes ' + this.interests[0]
  },
  greeting: function() {
    alert('Hi! I\'m ' + this.name[0] + '.');
  }
};
```

Object Oriented Programming

- Encapsulation
  - By Defining an Object template
  - Most of OOP languages call it a CLASS



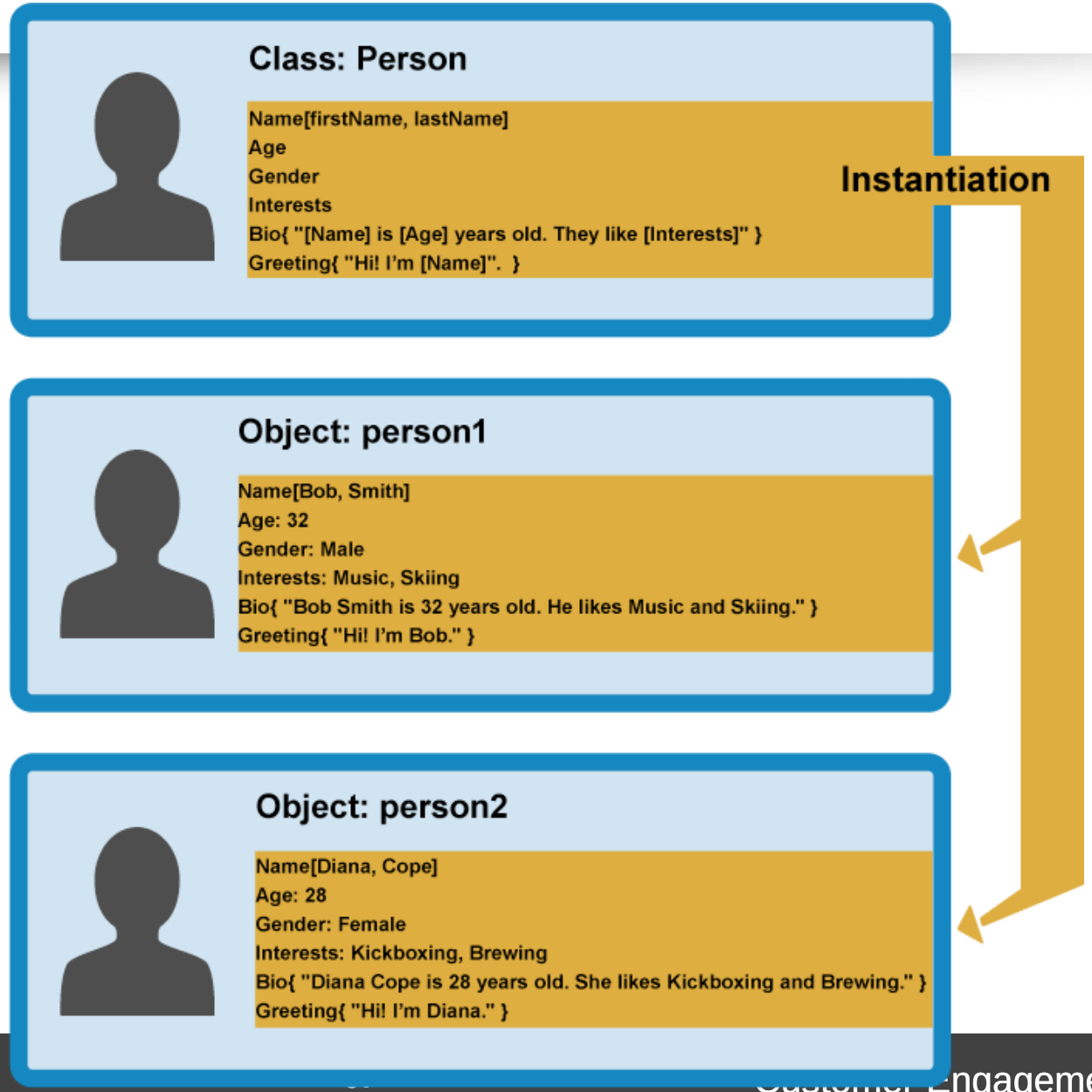
Implementation using Java Script

- The constructor function is JavaScript's version of a class

```
function Person(first, last, age, gender, interests) {
  this.name = {
    first,
    last
  };
  this.age = age;
  this.gender = gender;
  this.interests = interests;
  this.bio = function() {
    alert(this.name.first + ' ' + this.name.last + ' is ' + this.age + ' years old. He likes ' + this.
    interests[0] + ' and ' + this.interests[1] + '.');
  };
  this.greeting = function() {
    alert('Hi! I\'m ' + this.name.first + '.');
  };
}
```

Instantiation

- Process of creating actual Objects or instances



Implementation using Java Script

- New keyword is used for instantiation

```
var person1 = new Person('Tharun', 'Krishna', 8, 'male', ['Drawing', 'Playing']);
```

```
alert(person1['age']);
alert(person1.interests[1]);
person1.bio();
```

Implementation using Java Script

- Technique 2 : To create an instance using create()

```
var person = {
  name: ['Bob', 'Smith'],
  age: 32,
  gender: 'male',
  interests: ['music', 'skiing'],
  bio: function() {
    alert(this.name[0] + ' ' + this.name[1] + ' is ' + this.age + ' years old. He likes ' + this.interests[0]
  },
  greeting: function() {
    alert('Hi! I\'m ' + this.name[0] + '.');
  }
};
```

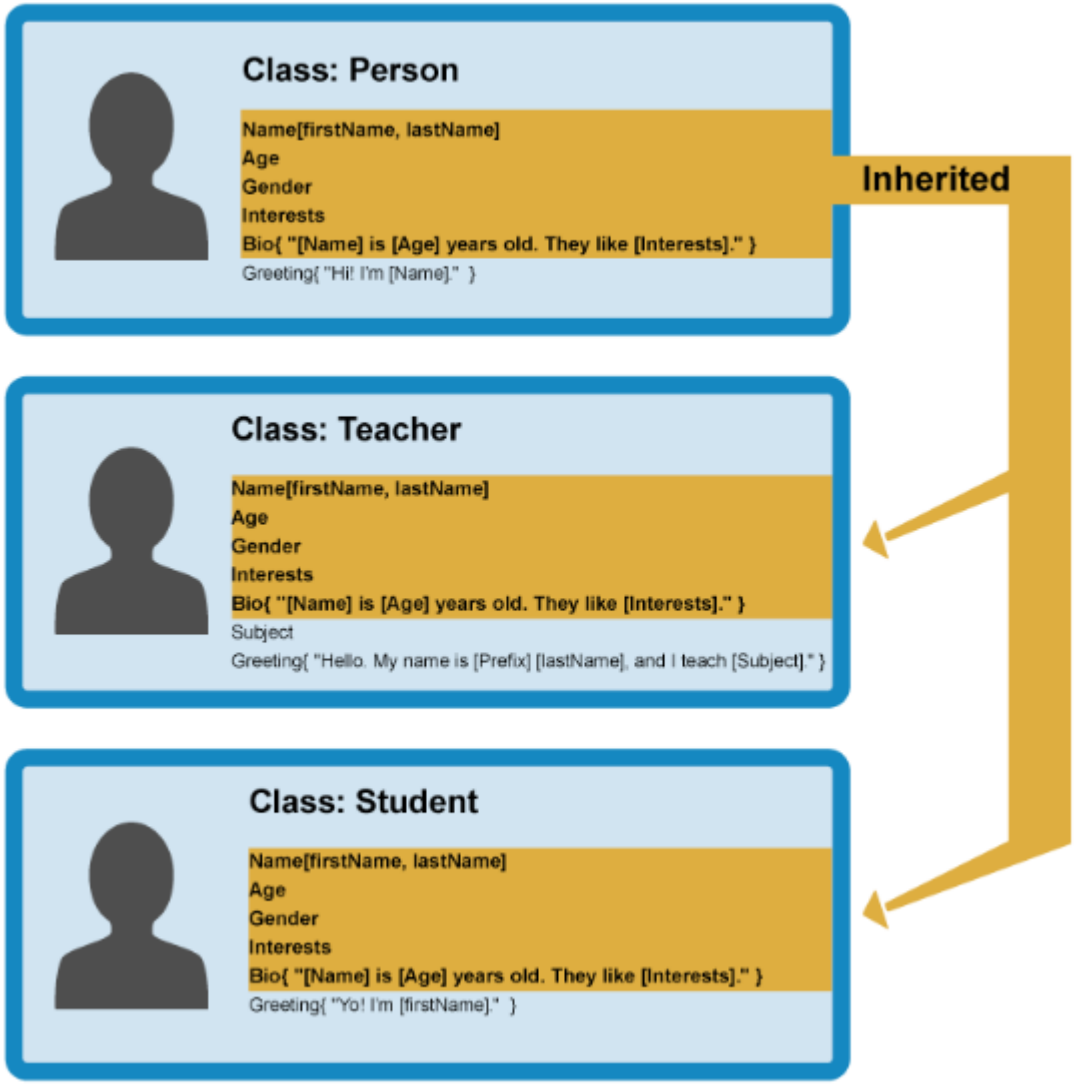
```
var person2 = Object.create(person);
```

Prototype Based Language

- Prototype Chaining

Inheritance

- In OOP, we can create new classes based on other classes
  - Existing class – **parent class**
  - New class - **child class**
- Child class can be made to **inherit** the data and code features of their **parent class**



Implementation using Java Script

```
function Brick() {
  this.width = 10;
  this.height = 20;
}

function BlueGlassBrick() {
  Brick.call(this);

  this.opacity = 0.5;
  this.color = 'blue';
}
```

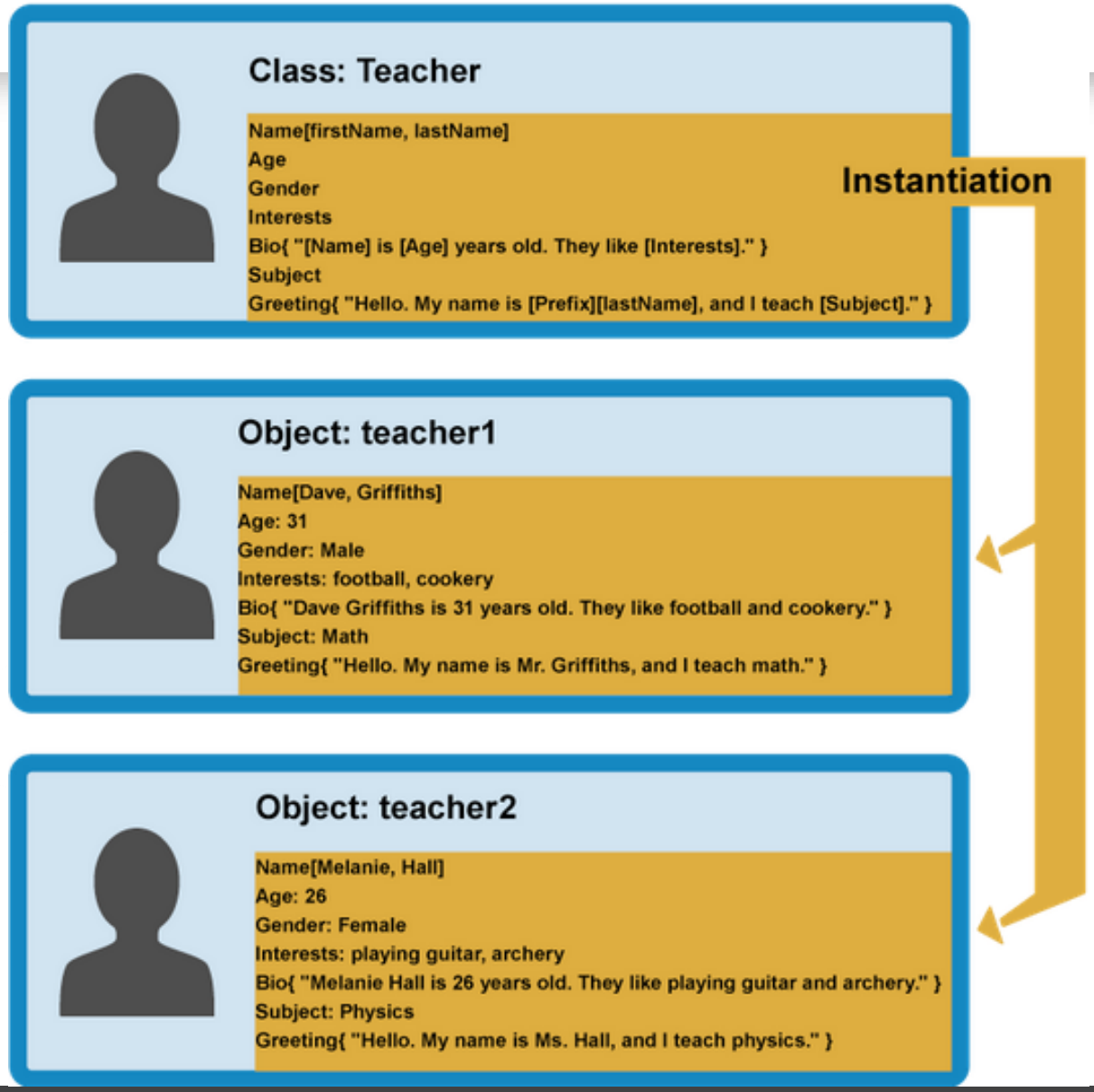
```
function Teacher(first, last, age, gender, interests, subject) {
  Person.call(this, first, last, age, gender, interests);

  this.subject = subject;
}
```

Setting prototype and constructor reference

```
Teacher.prototype = Object.create(Person.prototype);
Teacher.prototype.constructor = Teacher;
```

Cloning the prototype

Instantiation of Child Class



Thank You

Customer Engagement Reimagined

Our Locations: U.S.A | U.K | China | Costa Rica | India | Mauritius | Philippines | Poland | Singapore