Lecture # 1-5

JavaScript Lecture 1-5

Ms. Hafiza Alia
(alia@theprotec.com)

What is scripting ?

Scripting refers to a series of commands that are interpreted and executed sequentially and immediately on occurrence of an event.

This event is an action generated by a user while interacting with a Web page.

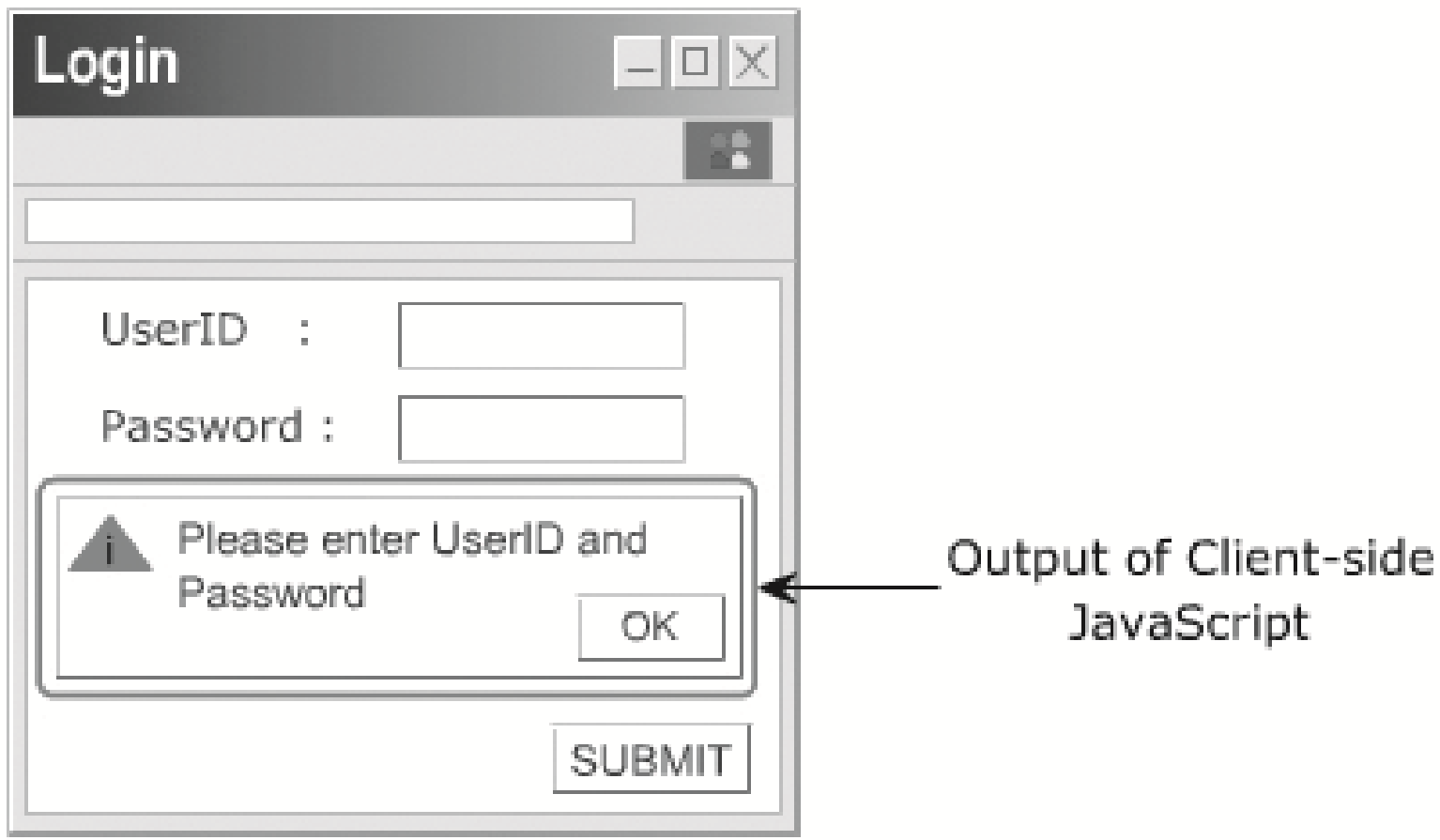
Examples of events include button clicks, selecting a product from a menu, and so on.

What is scripting ?

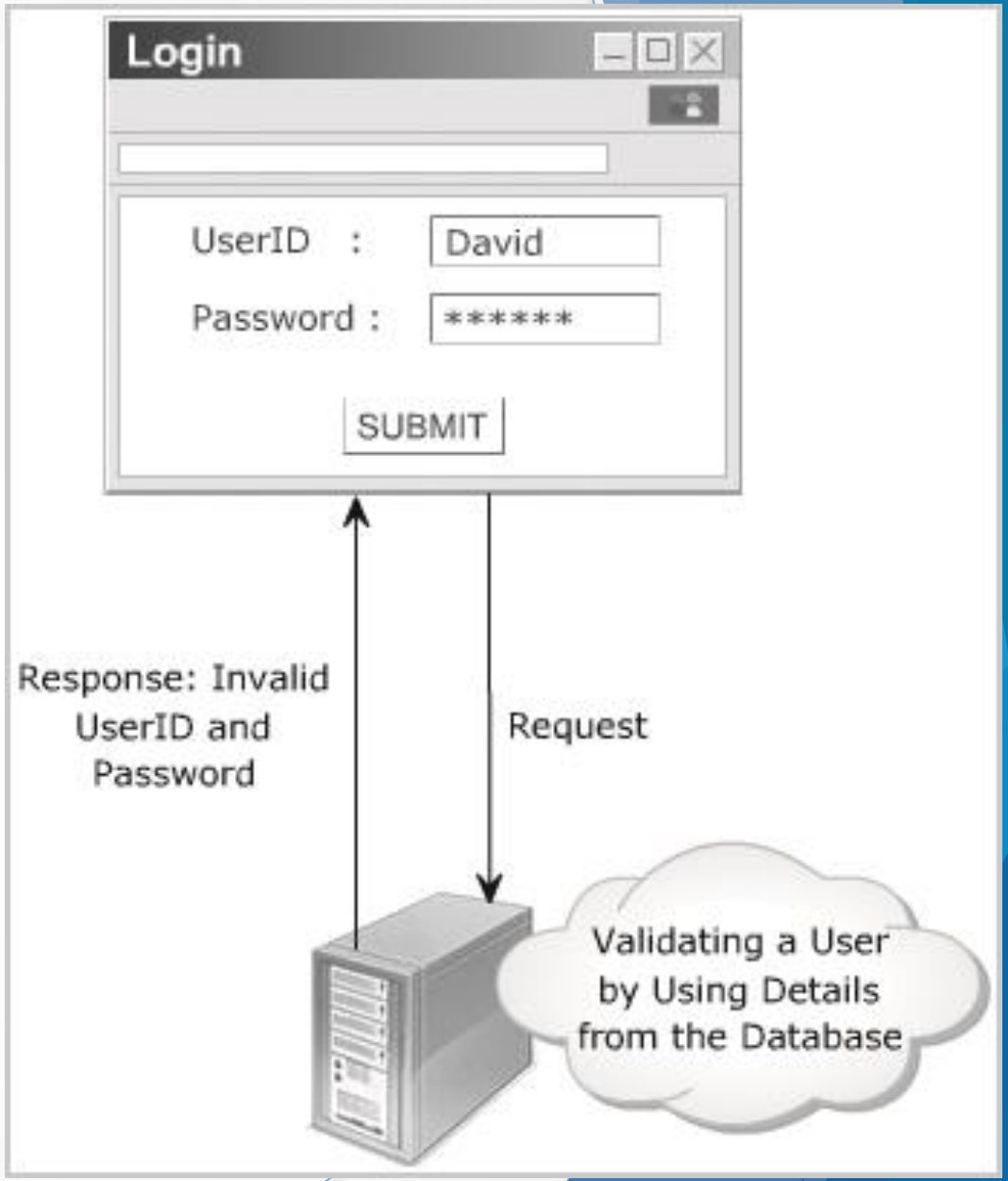
There are two types of scripting languages. They are as follows:

- **Client-side Scripting:**
  - Refers to a script being executed on the client's machine by the browser.
- **Server-side Scripting:**
  - Refers to a script being executed on a Web server to generate dynamic HTML pages.

Client-side JavaScript



Server-side JavaScript



What is JavaScript ?

JavaScript is a scripting language that allows building dynamic Web pages by ensuring maximum user interactivity.

Where to JavaScript ?

- The <script> Tag
  - <script>

  document.getElementById("demo").innerHTML = "My First JavaScript";</script>
- JavaScript in <head> or <body>
  - You can place any number of scripts in an HTML document.
  - Scripts can be placed in the <body>, or in the <head> section of an HTML page, or in both.
- External Javascript
  - <script src="myScript.js"></script>

JavaScript Output

- JavaScript can "display" data in different ways:
  - Writing into an HTML element, using **innerHTML**.
  - Writing into the HTML output using **document.write()**.
  - Writing into an alert box, using **window.alert()**.
  - Writing into the browser console, using **console.log()**.

JavaScript Output : **innerHTML.**

```
<script>
document.getElementById("demo").innerHTML =
“I will display a text in an element with ID equals
to demo”;
</script>
```

JavaScript Output : **document.write()**

- For testing purposes, it is convenient to use **document.write()**:

```
<script>
document.write(I will display whatever you write here);
</script>
```

JavaScript Output : **document.write()**

- Using document.write() after an HTML document is fully loaded, will **delete all existing HTML**

- <button type="button" onclick="document.write(5 + 6)">Try it</button>
- The document.write() method should only be used for testing.

JavaScript Output : **window.alert()**

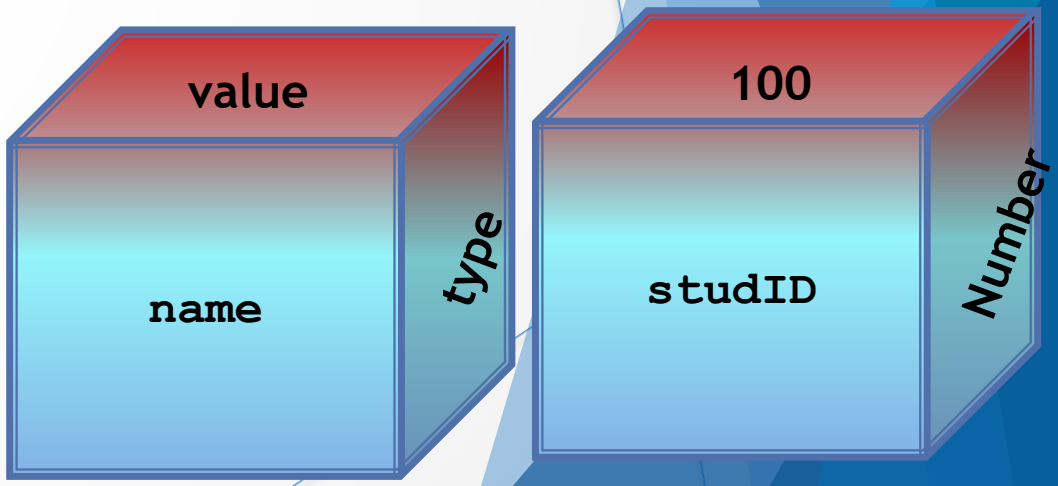
```
<script>
window.alert(I will display an Alert box with
this text you write here );
</script>
```

JavaScript Variables & Variable Declaration

- the var keyword is used to create a variable by allocating memory to it.
- a keyword is a reserved word that holds a special meaning.
- Syntax
  - var <variableName>;

where,

- var: Is the keyword in JavaScript.
- variableName: Is a valid variable name.



JavaScript Statements

- JavaScript statements are composed of:
- Values, Operators, Expressions, Keywords, and Comments.

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

JavaScript Programs

- A **computer program** is a list of "instructions" to be "executed" by a computer.
- In a programming language, these programming instructions are called **statements**.
- A **JavaScript program** is a list of programming **statements**.

Semicolons ;

- Semicolons separate JavaScript statements.
- Add a semicolon at the end of each executable statement:

```
var a, b, c;     // Declare 3 variables
a = 5;           // Assign the value 5 to a
b = 6;           // Assign the value 6 to b
c = a + b;       // Assign the sum of a and b to c
```

Semicolons ;

- When separated by semicolons, multiple statements on one line are allowed:

```
var a, b, c;
a = 5; b = 6; c = a + b;
```

JavaScript White Space

- JavaScript ignores multiple spaces. You can add white space to your script to make it more readable.

```
var person = “Page";
var person=“Page";
```

- A good practice is to put spaces around operators (= + - * /):

```
var x = y + z;
```

JavaScript Line Length and Line Breaks

- For best readability, programmers often like to avoid code lines longer than 80 characters.
- The best place to break a code line is after an **operator** or a **comma**.

```
document.getElementById("demo").innerHTML =
"Hello World!";
```

Variable Initialization

- Syntax
  - <variableName> = <value>;

- =: Is the assignment operator used to assign values.
- value: Is the data that is to be stored in the variable

Example

- var studID;
- var studName;
- studID = 1;
- studName = “Abcdeg”;

Variable Naming Rules

- JavaScript is a case-sensitive language.
- The variables X and x are treated as two different variables.

Variable Naming Rules

- ✓ can consist of digits, underscore, and alphabets.
- ✓ must begin with a letter or the underscore character.
- ✓ cannot begin with a number and cannot contain any punctuation marks.
- ✓ cannot contain any kind of special characters such as +, *, %, and so on.
- ✓ cannot contain spaces.
- ✓ cannot be a JavaScript keyword.

Data Types in JavaScript

- To identify the type of data that can be stored in a variable, JavaScript provides different data types.
- Data types in JavaScript are classified into two broad categories namely, primitive and composite data types.
- Primitive data types contain only a single value, whereas the composite data types contain a group of values.

Data Types in JavaScript

- PRIMITIVE DATA TYPES
  - A primitive data type contains a single literal value such as a number or a string.
  - A literal is a static value that you can assign to variables.

PRIMITIVE DATA TYPES

Primitive Data Type	Description
`boolean`	Contains only two values namely, true or false
`null`	Contains only one value namely, null. A variable of this value specifies that the variable has no value. This null value is a keyword and it is not the same as the value, zero
`number`	Contains positive and negative numbers and numbers with decimal point. Some of the valid examples include 6, 7.5, -8, 7.5e-3, and so on
`string`	Contains alphanumeric characters in single or double quotation marks. The single quotes is used to represent a string

COMPOSITE DATA TYPES

- A composite data type stores a collection of multiple related values.
- In JavaScript, all composite data types are treated as objects.
- A composite data type can be either predefined or user-defined in JavaScript.

COMPOSITE DATA TYPES

Composite Data Type	Description
Objects	Refers to a collection of properties and functions. Properties specify the characteristics and functions determine the behavior of a JavaScript object
Functions	Refers to a collection of statements, which are instructions to achieve a specific task
Arrays	Refers to a collection of values stored in adjacent memory locations

JavaScript Data Types

- JavaScript variables can hold many **data types**: numbers, strings, objects and more:
- Example

```
var length = 16;                                    // Number
var lastName = "Johnson";                           // String
var x = {firstName:"John", lastName:"Doe"};         // Object

Var x= true                                         //Boolean
```

JavaScript Types are Dynamic.

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

Using Comments

- SINGLE LINE COMMENTS
  - //This is a single line comment
- MULTI-LINE COMMENTS
  - /* This is a multi line comment*/

JavaScript Values

- The JavaScript syntax defines two types of values:
  - Fixed values and variable values.
- Fixed values are called literals.
- Variable values are called variables.

JavaScript Values

- JavaScript Literals
  - document.getElementById("demo").innerHTML = 100.0;
- JavaScript Variables
  - In a programming language, variables are used to store data values.
  - var x;

    x = 6;

JavaScript Operators

- JavaScript **arithmetic operators**
- JavaScript **assignment operators**
- JavaScript **String Operators**
- JavaScript **Comparison Operators**
- JavaScript **Logical Operators**
- JavaScript **Type Operators**

JavaScript Arithmetic Operators

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

JavaScript Arithmetic Operators

- Arithmetic operators perform arithmetic on numbers (literals or variables).
- Examples
  - var x = 100 + 50;
  - var x = a + b;
- Operators and Operands

Operand	Operator	Operand
10	%	10

JavaScript Arithmetic Operators

- Examples
  - var z = x + y;
  - var z = x - y;
  - var z = x * y;
  - var z = x / y;
  - var z = x % y;
  - x++;
  - x--;

JavaScript Expressions

- An **expression** produces a value
- The computation is called an evaluation.
  - For example, 5 * 10 evaluates to 50:
- Expressions can also contain variable values:
  - x * 10
- The values can be of various types, such as numbers and strings.
  - “ABCd” + “Defg”
- Eval(2+2)

JavaScript Assignment Operators

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

JavaScript Assignment Operators

- The = assignment operator assigns a value to a variable.
- The += assignment operator adds a value to a variable.
- The -= assignment operator subtracts a value from a variable.
- The *= assignment operator multiplies a variable.
- The /= assignment divides a variable.
- The %= assignment operator assigns a remainder to a variable.

JavaScript String Operators

- The + operator can also be used to add (concatenate) strings.

```
var txt1 = ”Abcs";
var txt2 = ”Efgh";
var txt3 = txt1 + " " + txt2;
```

- The += assignment operator can also be used to add (concatenate) strings:

```
var txt1 = "What a very ";
txt1 += "nice day";
```

When used on strings, the + operator is called the concatenation operator.

JavaScript Comparison Operators

==	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

Conditional (Ternary) Operator

- (*condition*) ? *value1*:*value2*

- var voteable = (age < 18) ? "Too young":"Old enough";

Conditional (Ternary) Operator

- Exercises
  - Check if marks are grater to 30 then assign status Pass else assign Status fail
  - Check if the kid is eligible for kindergarten or not
  - Check if a person is eligible to drive or not

JavaScript Type Operators

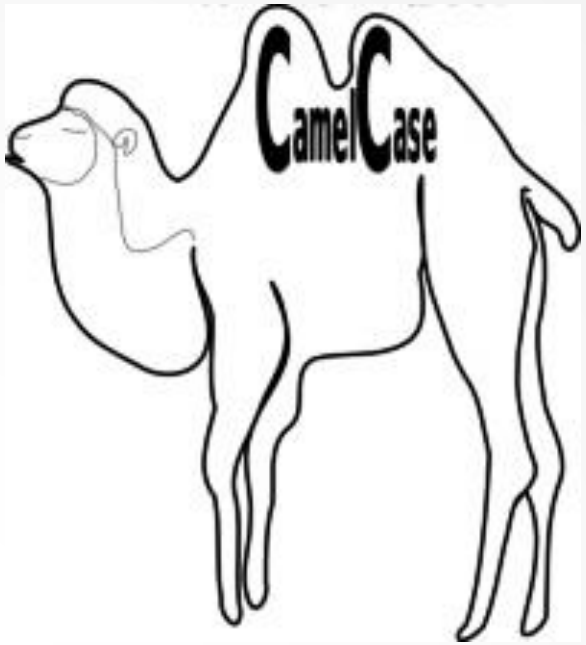
typeof	Returns the type of a variable

JavaScript and Camel Case

- Historically, programmers have used different ways of joining multiple words into one variable name:

- **Hyphens:**
  - first-name, last-name, master-card, inter-city. (not allowed in JS)
- Underscore:
  - first_name, last_name, master_card, inter_city.

JavaScript and Camel Case



- **Upper Camel Case (Pascal Case):**
  - FirstName, LastName, MasterCard, InterCity.

- **Lower Camel Case:**
  - JavaScript programmers tend to use camel case that starts with a lowercase letter:
  - f irstName, lastName, masterCard, interCity.

Multiple variables with single var keyword

- Start the statement with **var** and separate the variables by **comma:**
  - var person = "John Doe", carName = "Volvo", price = 200;

Print a variable in div with Id demo

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

<script>

var stuName = “Unknown”;

document.getElementById("demo").innerHTML = stuName;

</script>
```

Create a variable called **number**, assign the value **50** to it, and display it.

```
Var number=50;

Document.write(number);
```

Display the sum of 5 + 10, using two variables x and y.

```
Var x=5,y=10;
document.getElementById(‘demo’).innerHTM
L= eval(x+y) ;
```

Create a third variable called **z**, assign x + y to it, and display it.

Use a **single** var keyword to create three variables with the following values:

firstName = "John"

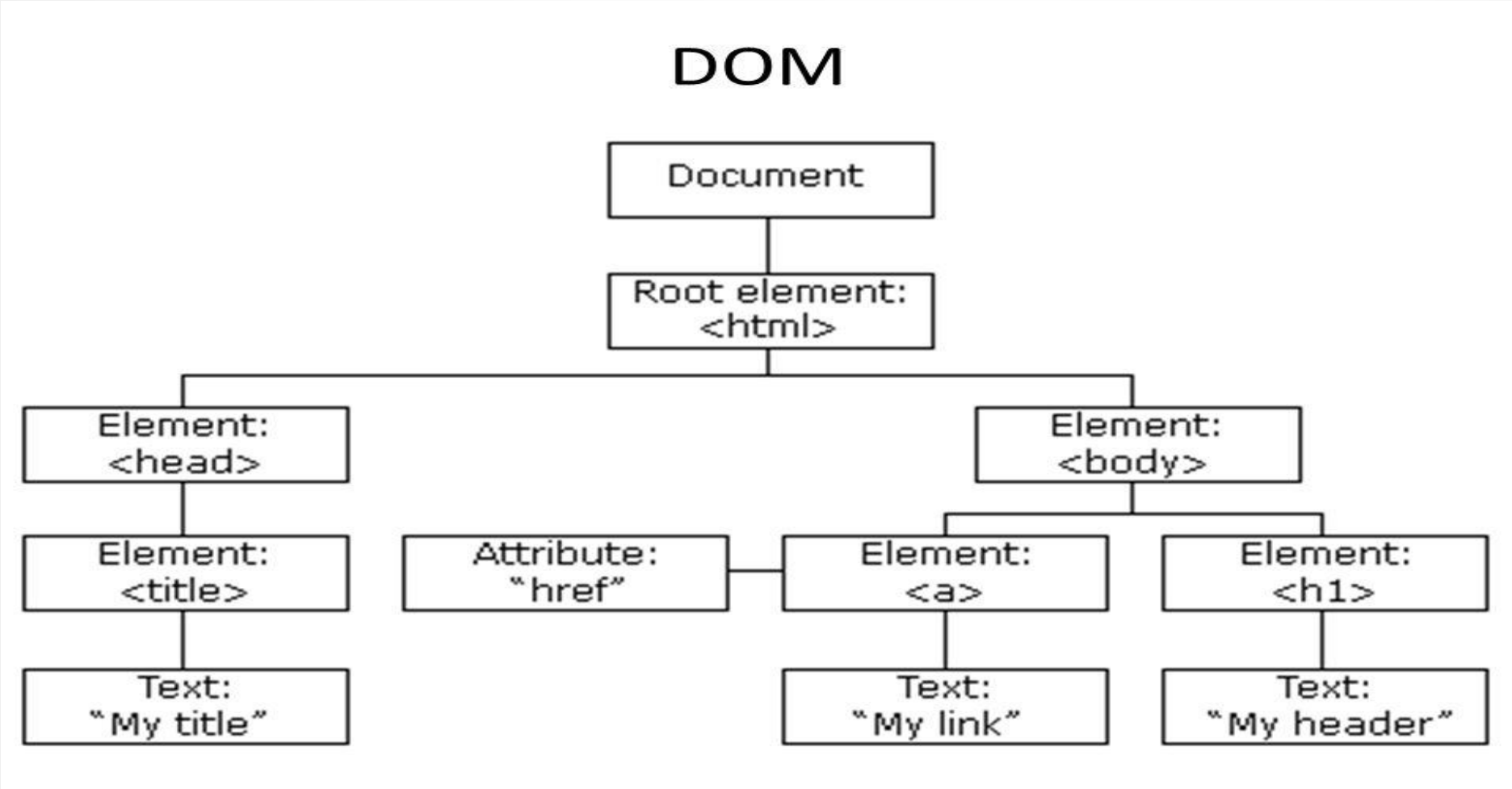
lastName = "Doe"

age = 35

What is the HTML DOM?

- The DOM defines a standard for accessing documents
- The HTML DOM is a standard **object** model and **programming interface** for HTML.
- It defines:
  - The HTML elements as **objects**
  - The **properties** of all HTML elements
  - The **methods** to access all HTML elements
  - The **events** for all HTML elements
- In other words: **The HTML DOM is a standard for how to get, change, add, or delete HTML elements.**

The HTML DOM Tree of Objects Document Object Model



The DOM Programming Interface

- The HTML DOM can be accessed with JavaScript (and with other programming languages).
- In the DOM, all HTML elements are defined as **objects**.
- The programming interface is the properties and methods of each object.
- A **property** is a value that you can get or set (like changing the content of an HTML element).
- A **method** is an action you can do (like add or deleting an HTML element).

JavaScript Objects

- JavaScript objects are written with curly braces.
- Object properties are written as name:value pairs, separated by commas.
- var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};

JavaScript Objects (Real life Obj example)

Object	Properties	Methods/Funstions
	car.name = Fiat car.model = 500 car.weight = 850kg car.color = white	car.start() car.drive() car.brake() car.stop()

JavaScript Objects

- var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};
- Line breaks are not important
- var person = {
  firstName:"John",
  lastName:"Doe",
  age:50,
  eyeColor:"blue"
  };

Task Create an Object and define all properties (chose any object)

JavaScript Objects

- Variables
  - Var car =Volvo;
  - Var carColor=“Gray”;
  - Var carModel=“Def”;
- Object (Car is an Object)
  - Var Car ={type:“Volvo", model:“fx106", color:“Gray"}
- The values are written as **name:value** pairs (name and value separated by a colon).

Accessing Object Properties

- You can access object properties in two ways:
  - *objectName.propertyName*
  - *objectName["propertyName"]*
- *Example*
  - *person.lastName;*
  - *person["lastName"];*

Null

- In JavaScript null is "nothing". It is supposed to be something that doesn't exist.
- Unfortunately, in JavaScript, the data type of null is an object.

Difference Between Undefined and Null

- Undefined and null are equal in value but different in type:

JavaScript Functions

- A JavaScript function is a block of code designed to perform a particular task.
- A JavaScript function is executed when "something" invokes it (calls it).
- function myFunction(p1, p2) {
  return p1 * p2; // The function returns the product of p1 and p2
  }
- Syntax ? Parameters ?

Function Invocation / Func Call

- The code inside the function will execute when "something" **invokes** (calls) the function:

Function Return

- When JavaScript reaches a **return statement**, the function will stop executing.
-

```
var x = myFunction(4, 3);    // Function is called, return value will end up in x

function myFunction(a, b) {
    return a * b;                // Function returns the product of a and b
}
```

Why Functions?

- You can reuse code: Define the code once, and use it many times.
- You can use the same code many times with different arguments, to produce different results.
-

```
function toCelsius(fahrenheit) {
    return (5/9) * (fahrenheit-32);
}
document.getElementById("demo").innerHTML =
toCelsius(77);
```

The () Operator Invokes the Function

- Accessing a function without () will return the function definition instead of the function result:
-

```
function toCelsius(fahrenheit) {
    return (5/9) * (fahrenheit-32);
}
document.getElementById("demo").innerHTML = toCelsius;
```

Functions Used as Variable Values

- Examples
  - var x = toCelsius(77);
    var text = "The temperature is " + x + " Celsius";
  - var text = "The temperature is " + toCelsius(77) + " Celsius";

Object Methods

- Objects can also have **methods**.
- Methods are **actions** that can be performed on objects.

Property	Property Value
firstName	John
lastName	Doe
age	50
eyeColor	blue
fullName	function() {return this.firstName + " " + this.lastName;}

Example

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

This Keyword

- In a function definition, **this** refers to the "owner" of the function.
- In the example above, **this** is the **person object** that "owns" the **fullName** function.
- In other words, **this.firstName** means the **firstName** property of **this object**.

Accessing Object Methods

- You access an object method with the following syntax:
  - objectName.methodName()
  - name = person.fullName();
- If you access a method **without** the () parentheses, it will return the **function definition**:
  - name = person.fullName;
  -

Exercise

- Create an object called **person** with name = John, age = 50. Then, access the object to display "John is 50 years old".

Exercise

- Create an object Student , assign a section and Id and create a function which will return the Sec+ID as a full ID of Studnet

What is an Array?

- An array is a special variable, which can hold more than one value at a time.
- JavaScript arrays are written with square brackets.
- Array items are separated by commas.
- var cars = ["Saab", "Volvo", "BMW"];

- var cars = [
  "Saab",
  "Volvo",
  "BMW"
  ];

Access the Elements of an Array

- var cars = [
  "Saab",
  "Volvo",
  "BMW"
  ];
- var name = cars[0];
- var name = cars[1];
- var name = cars[2];

Array indexes start with 0.
[0] is the first element.
[1] is the second element.

Access the Full Array

- Just print the name of Array;
- var cars = ["Saab", "Volvo", "BMW"];
  document.getElementById("demo").innerHTML = cars;

- **Length of Array**
- var fruits = ["Banana", "Orange", "Apple", "Mango"];
  fruits.length;

Changing an Array Element

- var cars = [
  "Saab",
  "Volvo",
  "BMW"
  ];
- cars[0]=“Alpha”

Adding Element in Array

- The easiest way to add a new element to an array is using the push method:

- var fruits = ["Banana", "Orange", "Apple", "Mango"];
  fruits.push("Lemon");

- var fruits = ["Banana", "Orange", "Apple", "Mango"];
  fruits[6] = "Lemon";

Adding elements with high indexes can create undefined "holes" in an array:

Associative Arrays

- Many programming languages support arrays with named indexes.
- Arrays with named indexes are called associative arrays (or hashes).
- JavaScript does **not** support arrays with named indexes.
- In JavaScript, **arrays** always use **numbered indexes**.

The Difference Between Arrays and Objects

- In JavaScript, **arrays** use **numbered indexes**.
- In JavaScript, **objects** use **named indexes**.
- **When to Use Arrays. When to use Objects.**
- JavaScript does not support associative arrays.
- You should use **objects** when you want the element names to be **strings (text)**.
- You should use **arrays** when you want the element names to be **numbers**

How to Recognize an Array

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

  typeof fruits; // returns object
- **Solution 1:**
- To solve this problem ECMAScript 5 defines a new method **Array.isArray()**:
- Array.isArray(fruits); // returns true
- **Solution 2:**
- The **instanceof** operator returns true if an object is created by a given constructor:
- var fruits = ["Banana", "Orange", "Apple", "Mango"];

  fruits instanceof Array // returns true

JavaScript
End Of Lecture 1 to 5