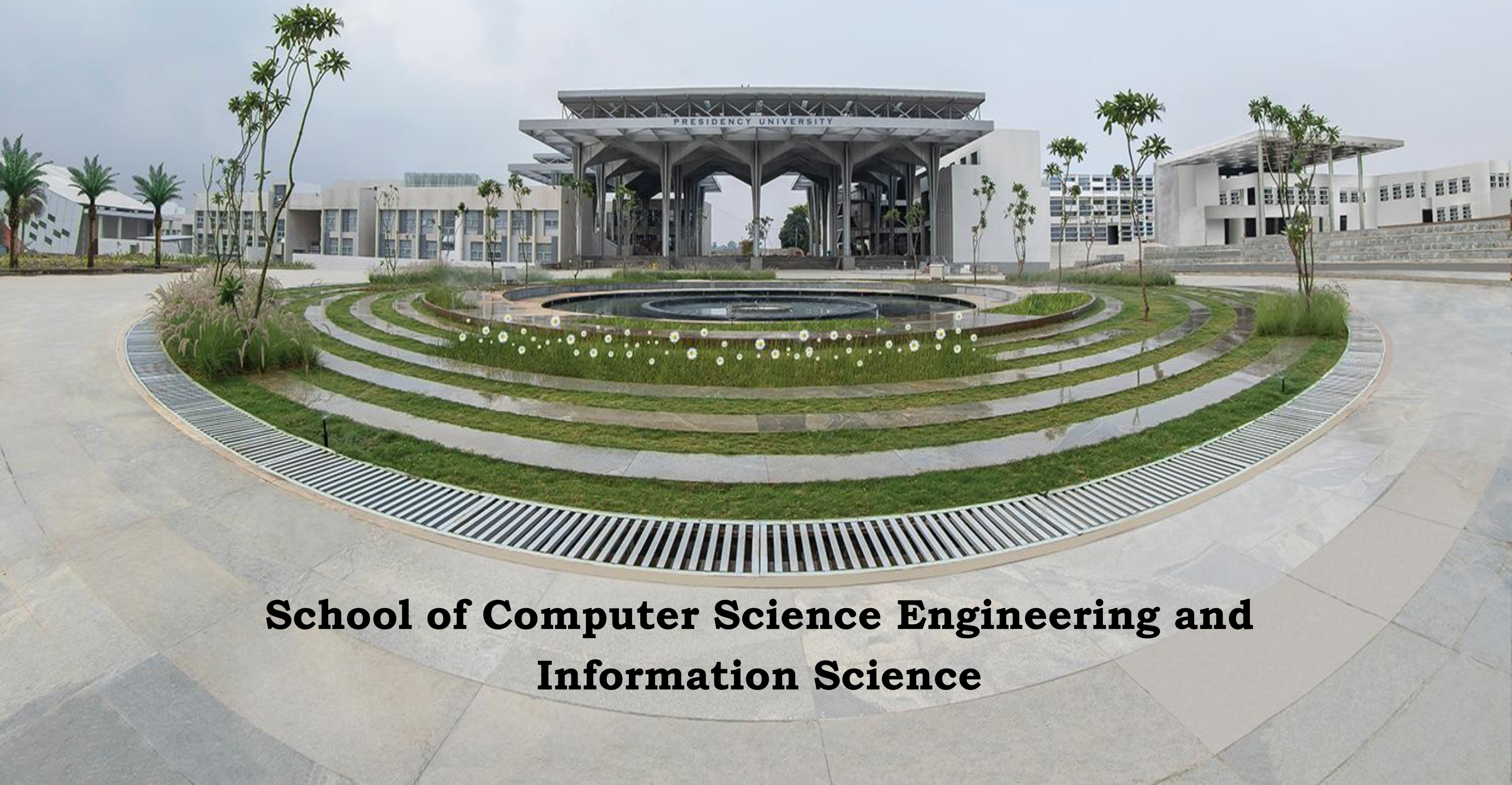
OVER 40 YEARS OF ACADEMIC WISDOM

PRESIDENCY UNIVERSITY

GAIN MORE KNOWLEDGE
REACH GREATER HEIGHTS

CSE3150 – Front-end Full Stack Development



School of Computer Science Engineering and Information Science

Module 2 - Responsive web design

JavaScript – Core syntax, HTML DOM, objects, classes, Async;

BootStrap for Responsive Web Design; Ajax and jQuery Introduction

What is Javascript?

- a lightweight programming language ("scripting language")
  - used to make web pages interactive
  - insert dynamic text into HTML (ex: user name)
  - **react to events** (ex: page load **user click**)
  - get information about a user's computer (ex: browser type)
  - perform calculations on user's computer (ex: form validation)

Javascript vs Java

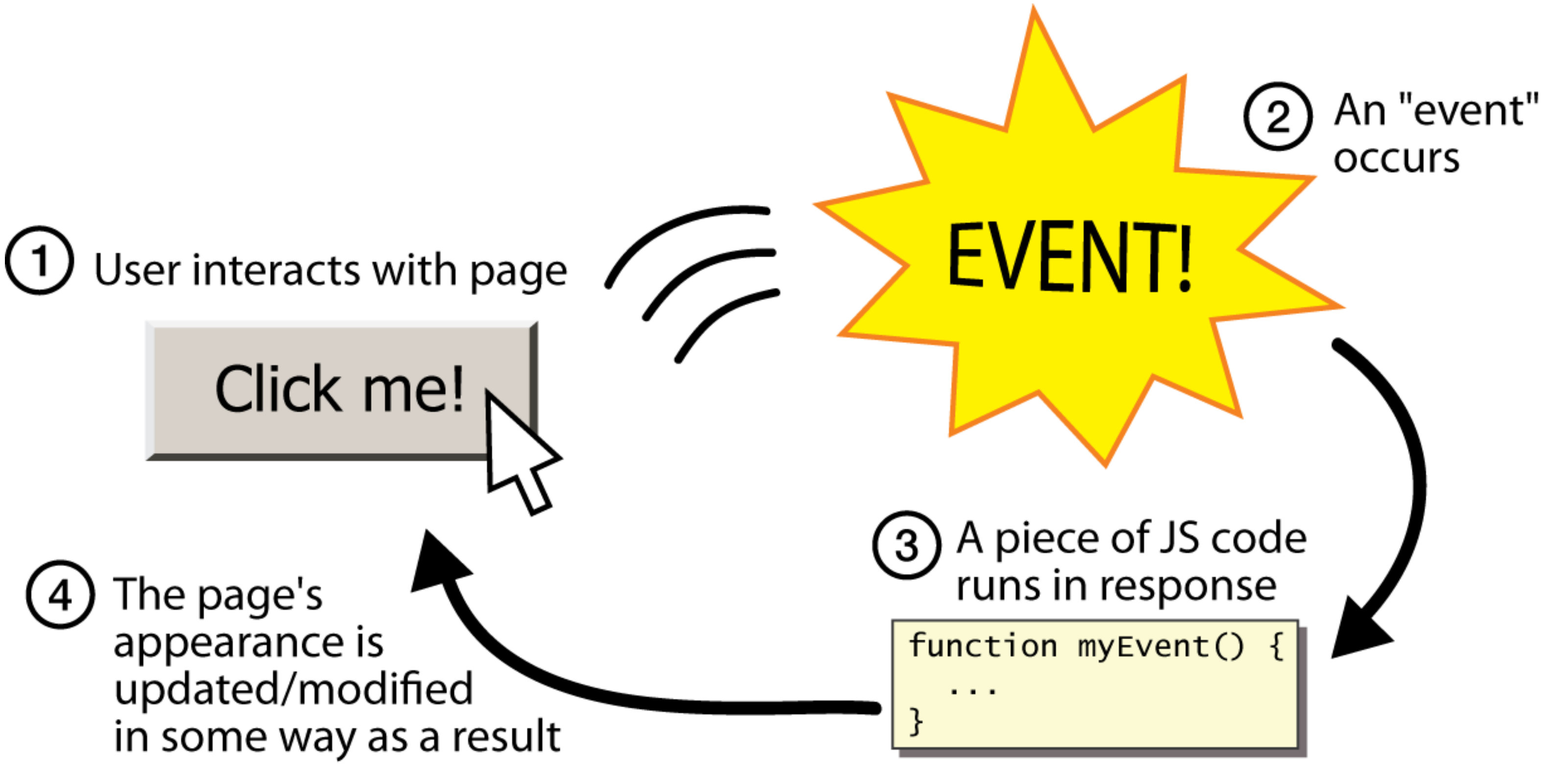
- interpreted, not compiled
- more relaxed syntax and rules
  - fewer and "looser" data types
  - variables don't need to be declared
  - errors often silent (few exceptions)
- key construct is the function rather than the class
  - "first-class" functions are used in many situations
- contained within a web page and integrates with its HTML/CSS content

Linking to a JavaScript file: `script`

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

- script tag should be placed in HTML page's head
- script code is stored in a separate .js file
- JS code can be placed directly in the HTML file's body or head (like CSS)

Event-driven programming



1 User interacts with page
Click me!
EVENT!
2 An "event" occurs
3 A piece of JS code runs in response
4 The page's appearance is updated/modified in some way as a result

```
function myEvent() {
  ...
}
```

Event-driven programming

- you are used to programs start with a main method (or implicit main like in PHP)
- JavaScript programs instead wait for user actions called *events* and respond to them
- event-driven programming: writing programs driven by user events
- Let's write a page with a clickable button that pops up a "Hello, World" window...

Event handlers

```
<element attributes onclick="function();">...
```

```
<button onclick="myFunction();">Click me!</button>
```

- JavaScript functions can be set as event handlers
  - when you interact with the element, the function will execute
- onclick is just one of many event HTML attributes we'll use
- but popping up an alert window is disruptive and annoying
  - A better user experience would be to have the message appear on the page...

JavaScript functions

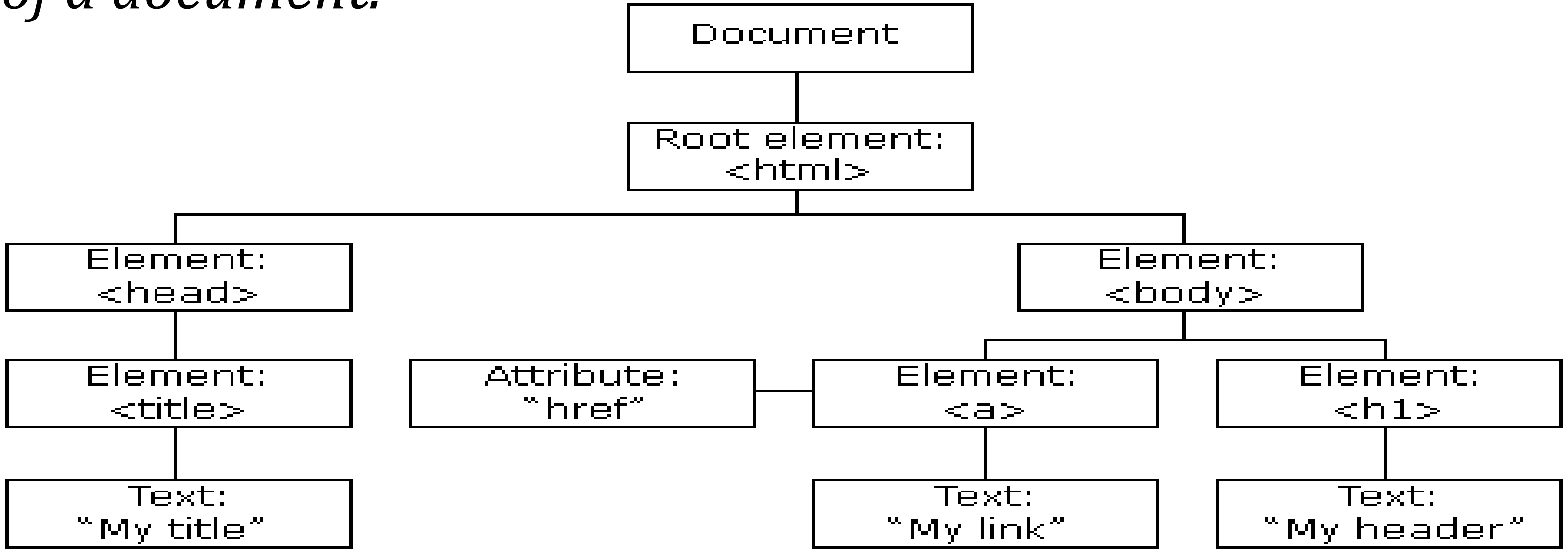
```
function name() {
statement ;
statement ;
...
statement ;
}                                                    JS
```

```
function myFunction() {
      alert("Hello!");
      alert("How are you?");
}                                                    JS
```

- the above could be the contents of example.js linked to our HTML page
- statements placed into functions can be evaluated in response to user events

The HTML DOM (Document Object Model)

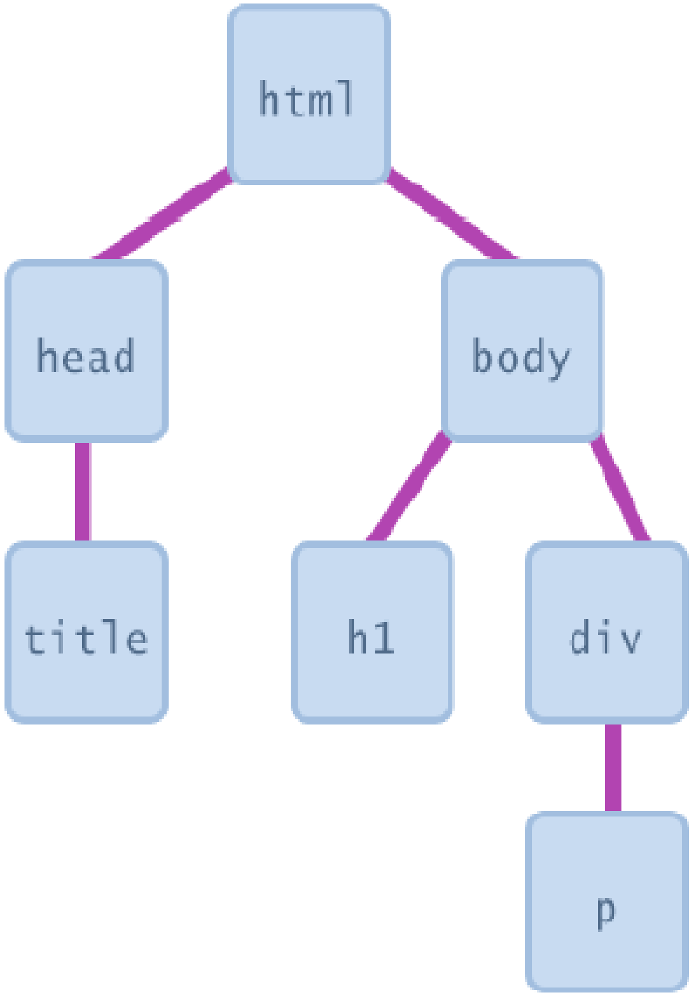
Document Object Model (DOM) is a platform and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure, and style of a document.



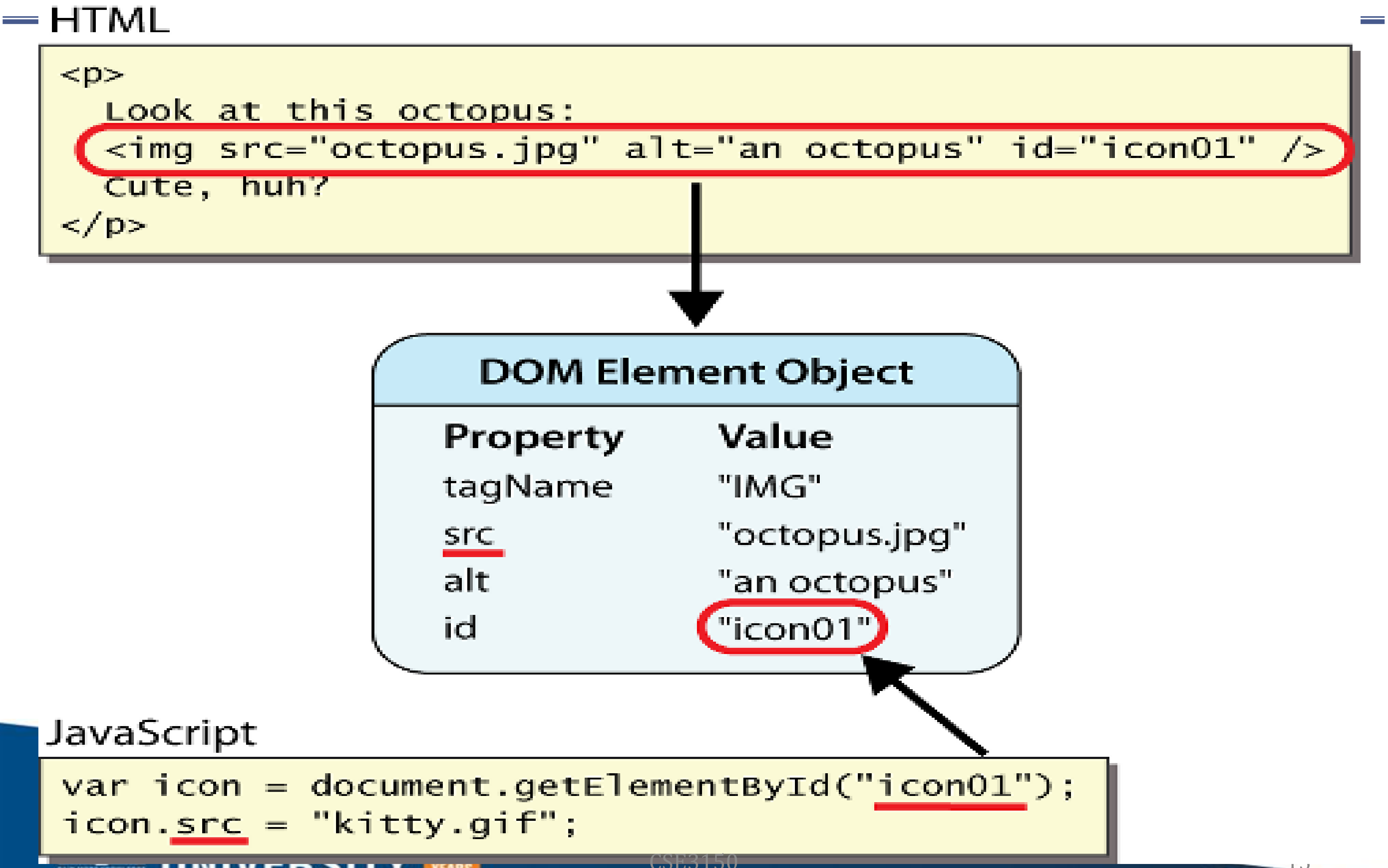
- The HTML elements as **objects**
- The **properties** of all HTML elements
- The **methods** to access all HTML elements
- The **events** for all HTML elements

Document Object Model (DOM)

- JavaScript can change all the HTML elements in the page
- JavaScript can change all the HTML attributes in the page
- JavaScript can change all the CSS styles in the page
- JavaScript can remove existing HTML elements and attributes
- JavaScript can add new HTML element and attributes
- JavaScript can react to all existing HTML events in the page
- JavaScript can create new HTML event in the page



DOM element objects



HTML

```
<p>
  Look at this octopus:
  <img src="octopus.jpg" alt="an octopus" id="icon01" />
  Cute, huh?
</p>
```

DOM Element Object

Property	Value
tagName	"IMG"
src	"octopus.jpg"
alt	"an octopus"
id	"icon01"

JavaScript

```
var icon = document.getElementById("icon01");
icon.src = "kitty.gif";
```

Accessing elements: `document.getElementById`

- document.getElementById returns the DOM object for an element with a given id
- can change the text inside most elements by setting the innerHTML property
- can change the text in form controls by setting the value property

Accessing elements: document.getElementById

- In the DOM, all HTML elements are defined as objects.

```
<body>
<p id="demo"></p>
<script>
document.getElementById("demo").innerHTML = "
Hello World!";
</script>
</body>
```

The innerHTML property is useful for getting or replacing the content of HTML elements.

Accessing elements: document.getElementById

```
var name = document.getElementById("id");
```

```
<button onclick="changeText();">Click me!</button>
<span id="output">replace me</span>
<input id="textbox" type="text" />
</body>
<script>
function changeText() {
        var span = document.getElementById("output");
        var textBox = document.getElementById("textbox");
        textbox.style.color = "red";
}
</script>
```

Changing element style: `element.style`

Attribute	Property or style object
color	color
padding	padding
background-color	backgroundColor
border-top-width	borderTopWidth
Font size	fontSize
Font famiy	fontFamily

```
function changeText() {
       //grab or initialize text here

       // font styles added by JS:
       text.style.fontSize = "13pt";
       text.style.fontFamily = "Comic Sans MS";
       text.style.color = "red"; // or pink?
}
```
JS

Reading Properties with JavaScript

Sample HTML

```
<ul id="t1">
<li> Item 1 </li>
</ul>
```

```
1. document.getElementById('t1').nodeName
2. document.getElementById('t1').nodeValue
3. document.getElementById('t1').firstChild.nodeName
4. document.getElementById('t1').firstChild.firstChild.nodeName
5. document.getElementById('t1').firstChild.firstChild.nodeValue
```

- Example 1 returns "ul"
- Example 2 returns "null"
- Example 3 returns "li"
- Example 4 returns "text"
  - A text node below the "li" which holds the actual text data as its value
- Example 5 returns " Item 1 "

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().

Using innerHTML

To access an HTML element, JavaScript can use the document.getElementById(id) method.

The id attribute defines the HTML element. The innerHTML property defines the HTML content:

```
<html>
<body>
<h1>My First Web Page</h1>
<p>My First Paragraph</p>
<p id="demo"></p>
<script>
document.getElementById("demo").innerHTML = 5 +
6;
</script>
</body>
</html>
```

Using document.write()

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

```
<button type="button" onclick="document.write(5 + 6)">
Try it</button>
```

Displaying text

- The document.write() method writes a string of text to the browser

```
<script type="text/javascript">
  <!--
   document.write("<h1>Hello, world!</h1>");
  //-->
</script>
```

document.write()

Ends in a semicolon

```
document.write("<h1>Hello,world!</h1>");
```

Enclosed in quotes -- denotes a "string"

Using window.alert()

```
<script>
window.alert(7 +12);
</script>
```

Using console.log()

For debugging purposes, you can call the console.log() method in the browser to display data.

Comments in JavaScript

- Two types of comments
  - Single line
    - Uses two forward slashes (i.e. **//**)
  - Multiple line
    - Uses **/*** and ***/**

Language Basics

- JavaScript is case sensitive
  - onClick, ONCLICK, … are HTML, thus not case-sensitive
- Statements terminated by returns or semi-colons
  - x = x+1;

  "Blocks" of statements enclosed in { …}
- Variables
  - Define using the var statement
  - Define implicitly by its first use, which must be an assignment
    - Implicit defn has global scope, even if occurs in nested scope!

JavaScript Primitive Datatypes

- Boolean: true and false
- Number: 64-bit floating point
  - Similar to Java double and Double
  - No integer type
  - Special values NaN (not a number) and Infinity
- String: sequence of zero or more Unicode chars
  - No separate character type (just strings of length 1)
  - Literal strings using ' or " characters (must match)
- Special objects: null and undefined

4 Ways to Declare a JavaScript Variable

- Using var
- Using let
- Using const
- Using nothing

Variables

```
var name = expression;                                    JS
```

```
var clientName = "Connie Client";
var age = 32;
var weight = 127.4;                                        JS
```

- variables are declared with the var keyword (case sensitive)
- types are not specified, but JS does have types ("loosely typed")
  - `Number, Boolean, String, Array, Object, Function, Null, Undefined`
  - can find out a variable's type by calling `typeof()`

- let x = 5;
- let y = 6;
- let z = x + y;

When to Use?

- Always declare JavaScript variables with var,let, or const.
- The let and const keywords were added to JavaScript in 2015.
- If you want your code to run in older browsers, you must use var.
- If you want a general rule: always declare variables with const.
- If you think the value of the variable can change, use let.
  - const price1 = 5;
  - const price2 = 6;
  - let total = price1 + price2;

Number type

```
var enrollment = 99;
var medianGrade = 2.8;
var credits = 5 + 4 + (2 * 3);
                                                    JS
```

- integers and real numbers are the same type (no int vs. double)
- same operators: + - * / % ++ -- = += -= *= /= %=
- similar precedence to Java
- many operators auto-convert types: "2" * 3 is 6

Comments (same as Java)

```
// single-line comment
/* multi-line comment */
                                                   JS
```

- identical to Java's comment syntax
- recall: 4 comment syntaxes
  - HTML: <!-- comment -->
  - CSS/JS/PHP: /* comment */
  - Java/JS/PHP: // comment
  - PHP: # comment

Math object

```
var rand1to10 = Math.floor(Math.random() * 10 + 1);
var three = Math.floor(Math.PI);
                                                         JS
```

- methods: `abs, ceil, cos, floor, log, max, min, pow, random, round, sin, sqrt, tan`
- properties: `E, PI`

```
// Numbers:
let length = 16;
let weight = 7.5;
// Strings:
let color = "Yellow";
let lastName = "Johnson";
// Booleans
let x = true;
let y = false;
// Object:
const person = {firstName:"John", lastName:"Doe"};
// Array object:
const cars = ["Saab", "Volvo", "BMW"];
// Date object:
const date = new Date("2022-03-25");
```

Special values: null and undefined

```
var ned = null;
var benson = 9;
// at this point in the code,
// ned is null
// benson's 9
// caroline is undefined
                                                    JS
```

- `undefined` : has not been declared, does not exist
- `null` : exists, but was specifically assigned an empty or null value
- Why does JavaScript have both of these?

Operators (self study)

- Refer all operators you studied in java

Logical operators

- > < >= <= && || ! == != === !==
- most logical operators automatically convert types:
  - 5 < "7" is true
  - 42 == 42.0 is true
  - "5.0" == 5 is true
- === and !== are strict equality tests; checks both type and value
  - "5.0" === 5 is false

JavaScript Strings

- <p>The length of the string is:</p>
- <p id="demo"></p>

- <script>
- let text = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
- document.getElementById("demo").innerHTML = text.length;
- </script>

JavaScript Functions

-
  ```
  function name(parameter1, parameter2, parameter3) {
    // code to be executed
  }
  ```

Function Invocation

- The code inside the function will execute when "something" **invokes** (calls) the function:
- When an event occurs (when a user clicks a button)
- When it is invoked (called) from JavaScript code
- Automatically (self invoked)

Example 1

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

<script>
var x = myFunction(4, 3);
document.getElementById("demo").innerHTML = x;

function myFunction(a, b) {
  return a * b;
}
</script>
```

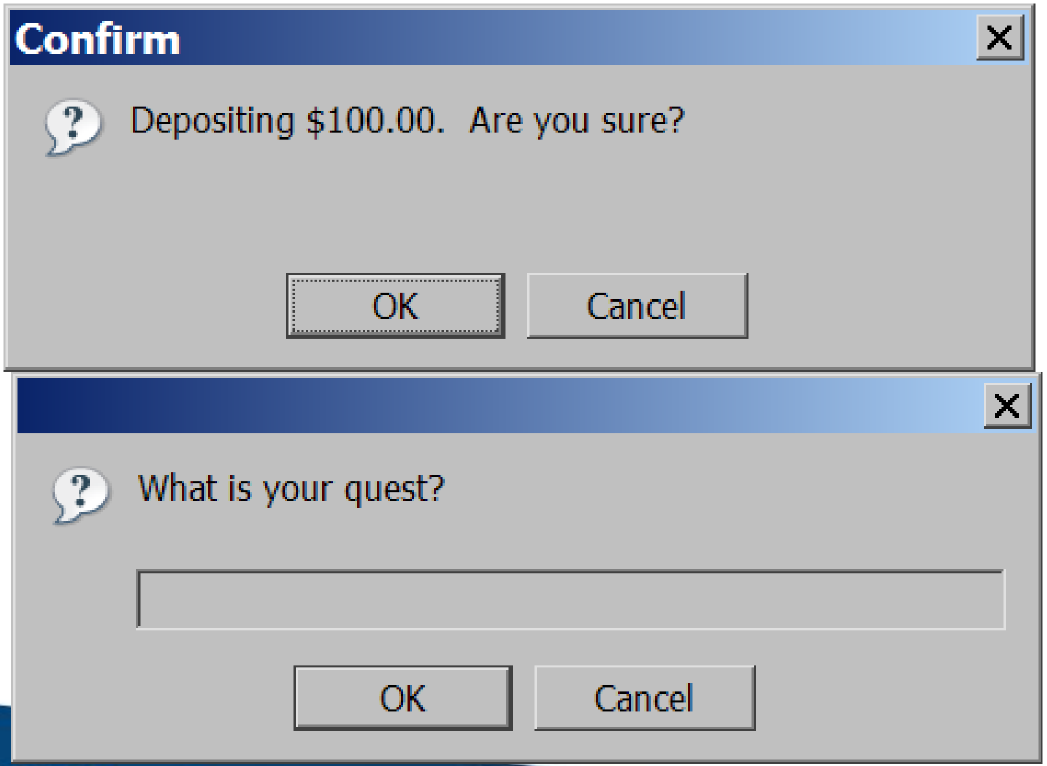
Example 2

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

<script>
document.getElementById("demo").innerHTML =
"The temperature is " + toCelsius(77) + " Celsius";

function toCelsius(fahrenheit) {
  return (5/9) * (fahrenheit-32);
}
</script>
```

Popup boxes

```
alert("message"); // message
confirm("message"); // returns true or false
prompt("message"); // returns user input string
```

JS



Confirm

Depositing $100.00. Are you sure?

What is your quest?

JavaScript Objects

- Real Life Objects, Properties, and Methods
- In real life, a student is an object.
- Student can have properties name, roll_no., marks, phone_num, age...
- Student can have methods to operate on properties like Calcualte_cgpa(), diplayInfo()....
- All student have same properties, values may change
- Assume car is an object

- `let car = "Fiat";`
- Objects are variables too. But objects can contain many values.
- `const car = {type:"Fiat", model:"500", color:"white"};`
- The values are written as **name:value** pairs (name and value separated by a colon).

```
<p id="demo"></p>
<script>
// Create an object:
const person = {
  firstName: "John",
  lastName: "Doe",
  age: 50,
  eyeColor: "blue"
};
// Display some data from the object:
document.getElementById("demo").innerHTML =
person.firstName + " is " + person.age + " years old.";
</script>
```

JavaScript Events

- HTML events are "things" that happen to HTML elements.
- When JavaScript is used in HTML pages, JavaScript can "react" on these events.

HTML Events

- An HTML event can be something the browser does, or something a user does.

Here are some examples of HTML events:

- An HTML web page has finished loading
- An HTML input field was changed
- An HTML button was clicked

- HTML allows event handler attributes, **with JavaScript code**, to be added to HTML elements.

```
<element event='some JavaScript’>
```

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

Try this

```
<button onclick="displayDate()">The time
is?</button>

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

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

- Objects use names to access its "members".
- In this example, person.firstName returns John:

```
<p id="demo"></p>
<script>
const person = {firstName:"John", lastName:"Doe",
age:46};
document.getElementById("demo").innerHTML =
person.firstName;
</script>
```

JavaScript Arrays

Syntax:

- `const array_name = [item1, item2, ...];`

```
const cars = [
  "Saab",
  "Volvo",
  "BMW"
];
```

Accessing Array Elements

-
  ```
  const cars = ["Saab", "Volvo", "BMW"];
  let car = cars[0];  //Saab
  ```

-
  ```
  const fruits = ["Banana", "Orange", "Apple", "Mango"];
  let length = fruits.length;   //4
  ```

Looping Array Elements

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

```
<p id="demo"></p>
<script>
const numbers = [65, 44, 12, 4];
numbers.forEach(myFunction)
document.getElementById("demo").innerHTML =
numbers;

function myFunction(item, index, arr) {
  arr[index] = item * 10;
}
</script>
```

Adding elements to array

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

  OR

-
  ```
  const fruits = ["Banana", "Orange", "Apple"];
  fruits[fruits.length] = "Lemon";
  ```

- 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**.

Array methods

```
var a = ["Stef", "Jason"];          // Stef, Jason
a.push("Brian");               // Stef, Jason, Brian
a.unshift("Kelly");            // Kelly, Stef, Jason, Brian
a.pop();                  // Kelly, Stef, Jason
a.shift();                // Stef, Jason
a.sort();                 // Jason, Stef
```

- array serves as many data structures: list, queue, stack, ...
- methods: `concat, join, pop, push, reverse, shift, slice, sort, splice, toString, unshift`
  - push and pop add / remove from back
  - unshift and shift add / remove from front
  - shift and pop return the element that is removed

if/else statement (same as Java)

```
if (condition) {
      statements;
} else if (condition) {
      statements;
} else {
      statements;
}
                                                        JS
```

- identical structure to Java's if/else statement
- JavaScript allows almost anything as a condition

for loop (same as Java)

```
var sum = 0;
for (var i = 0; i < 100; i++) {
      sum = sum + i;
}
```

```
var s1 = "hello";
var s2 = "";
for (var i = 0; i < s.length; i++) {
      s2 += s1.charAt(i) + s1.charAt(i);
}   // s2 stores "hheelllloo"
```

```
<script>
const cars = ["BMW", "Volvo", "Saab", "Ford",
"Fiat", "Audi"];
let text = "";
for (let i = 0; i < cars.length; i++) {
  text += cars[i] + "<br>";
}
document.getElementById("demo").innerHTML = text;
</script>
```

The For In Loop

-
  ```
  for (key in object) {
    // code block to be executed
  }
  ```

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

- Output for text is John Doe 25

The For Of Loop

-
  ```
  for (variable of iterable) {
    // code block to be executed
  }
  ```

```
const cars = ["BMW", "Volvo", "Mini"];
let text = "";
for (let x of cars) {
  text += x;
}  // BMW Volvo Mini
```

```
let language = "JavaScript";
let text = "";
for (let x of language) {
 text += x ; }                    // JavaScript
```

while loops (same as Java)

```
while (condition) {
          statements;
}                                                                    JS
```

```
do {
     statements;
} while (condition);
                                                                     JS
```

- break and continue keywords also behave as in Java

String type

```
var s = "Connie Client";
var fName = s.substring(0, s.indexOf(" ")); // "Connie"
var len = s.length; // 13
var s2 = 'Melvin Merchant';
                                                          JS
```

- methods: `charAt`, `charCodeAt`, `fromCharCode`, `indexOf`, `lastIndexOf`, `replace`, `split`, `substring`, `toLowerCase`, `toUpperCase`
  - charAt returns a one-letter String (there is no char type)
- length property (not a method as in Java)
- Strings can be specified with "" or ''
- concatenation with + :
  - 1 + 1 is 2, but "1" + 1 is "11"

THANK YOU