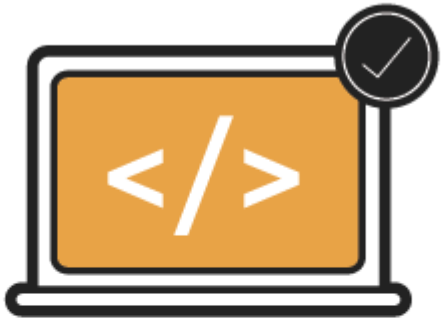
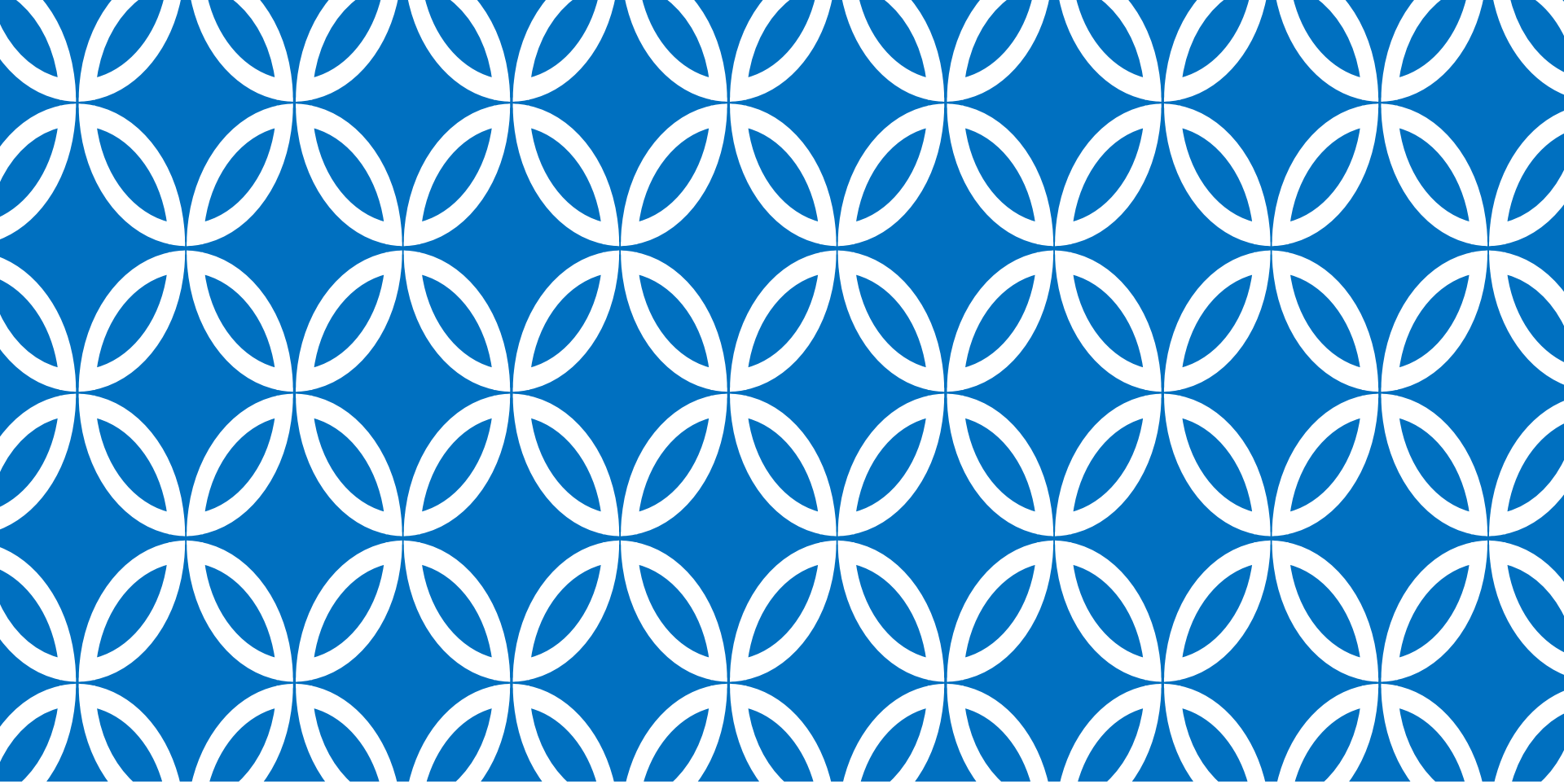
ACADGILD



FRONT END WEB DEVELOPMENT FUNDAMENTALS

Website : http://www.acadgild.com
LinkedIn : https://www.linkedin.com/company/acadgild
Facebook : https://www.facebook.com/acadgild



Session 5 – JavaScript

Agenda – JavaScript

Sl No	Agenda Title
1	Looping Statement
2	for Loop
3	for-in Loop
4	while Loop
5	do while Loop
6	function
7	Accessing Unnamed Arguments
8	Scope

Looping Statement

- Loops let you execute a block of code certain number of times.
- **Types:**
  - for” Loops
  - “for/in” Loops
  - “while” Loops
  - “do … while” Loops

for Loop

- One of the most used and familiar loops is “for loop”.
- It iterates through a sequence of statements for a number of times controlled by a condition.
- The change_exp determines how much has been added or subtracted from the counter variable.

Syntax :

```
for (initial_expression; test_exp; change_exp)
{  statements;  }
```

for–in Loop

- When the for/in statement is used, the counter and termination are determined by the length of the object.
- The statement begins with 0 as the initial value of the counter variable, terminates with all the properties of the objects have been exhausted.
  - E.g. array → no more elements found

Syntax :

```
for (counter_variable in object)
{ statements; }
```

while Loop

- The **while loop** begins with a termination condition and keeps looping until the termination condition is met.
- The counter variable is managed by the context of the statements inside the curly braces.

Syntax :

initial value declaration;

```
while (condition) {
        statements;
        increment/decrement statement;
}
```

do while loop

- The **do/while** loop always executes statements in the loop in the first iteration of the loop.
- The termination condition is placed at the bottom of the loop.

Syntax :

initial value declaration;

```
while (condition) {
        statements;
        increment/decrement statement;
}
```

function

- 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).
- A **JavaScript** function is defined with the **function** keyword, followed by a **name** and then followed by parentheses **()**.
- Function names can contain letters, digits, underscores, and dollar signs (same rules as variables).
- The parentheses may include parameter names separated by commas: **(*parameter1, parameter2, ..., so on*)**.
- The code to be executed, by the function, is placed inside curly brackets: **{}**

Syntax :

```
function Name(parameter1, parameter2, parameter3) {
        code to be executed
}
```

Accessing Unnamed Arguments

How do we get more arguments than listed in parameters?

- There is a special pseudo-array inside each function called **arguments.**
- It contains all parameters by their
  number: arguments[0], arguments[1] etc.

Example :

```
function sayHi() {
for(var i=0; i<arguments.length; i++) {
    alert("Hi, " + arguments[i]) }
}
say Hi("Ron", "Alice") // 'Hi, Ron', then 'Hi, Alice'
```

Scope

- **“Scope”** refers to the variables that are available to a piece of code at a given time.
- Functions have access to variables defined in the same scope.

Example :

```
var foo = 'hello';

var sayHello = function() {
console.log(foo);
};

sayHello(); // logs 'hello'
console.log(foo); // also logs 'hello'
```

Scope (contd.)

- Code outside the **scope** in which a variable was defined does not have access to the variable

Example :

```
var sayHello = function() {
var foo = 'hello';
console.log(foo);
};

sayHello(); // logs 'hello'
console.log(foo); // doesn't log anything
```

Scope (contd.)

- Variables with the same name can exist in different scopes with different values

Example :

```
var foo = 'world';

var sayHello = function() {
var foo = 'hello';
console.log(foo);
};

sayHello(); // logs 'hello'
console.log(foo); // logs 'world'
```

Agenda – JavaScript

1. DOM
2. How DOM works?
3. DOM Manipulation
4. Modifying HTML Using innerHTML
5. Events
6. Event Types
7. Event Bubbling and Event Capturing
8. Action Dialog
9. Form Validation

DOM (Document Object Model)

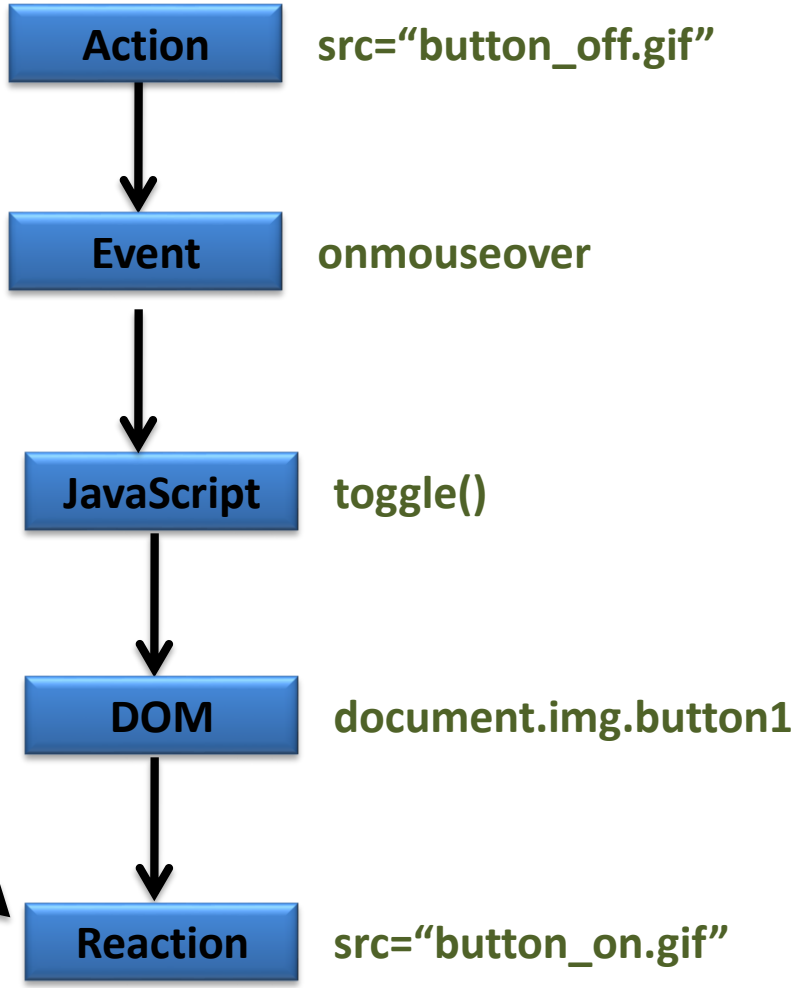
- **DOM** is cross-platform and language-independent programming interface for building, accessing, and manipulating valid HTML and well-formed XML documents.
- Ultimate goal is to make it possible for programmers to write applications that work properly on all browsers and servers, and on all platforms.
- When a web page is loaded, Browser creates a **D**ocument **O**bject **M**odel of the page.

DOM (contd.)

- With Object model, JavaScript gets power that it need to create dynamic HTML.
- Now JavaScript can:
  - change the HTML elements in the page
  - change the HTML attributes in the page
  - change the CSS styles in the page
  - remove existing HTML elements and attributes
  - add new HTML elements and attributes
  - react to all existing HTML events in the page
  - create new HTML events in the page

How DOM Works ?

```
<head>
        <script>
  function toggle()
  { document.img.button1.src=
  "button_on.gif"; }
  </script>
 </head>
<body>  <a href="test.html"
    onmouseover="toggle()"> <img
    name="button1"
    src="button_off.gif">
</a>  </body>
```



DOM Manipulation

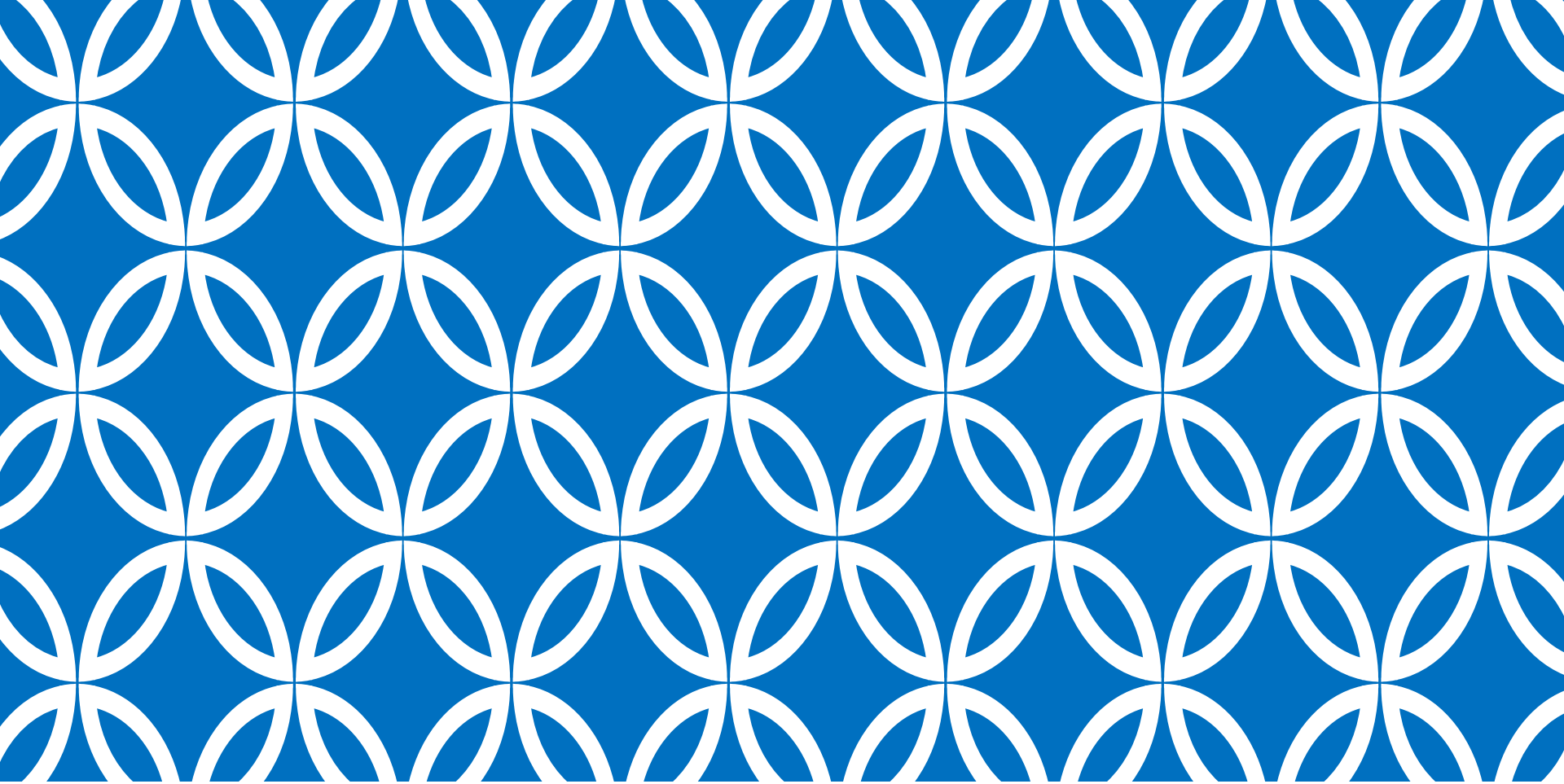
- **document.getElementsByTagName(tagname)** - This method returns a collection of all elements reference in the document with the specified tag name.
- **document.getElementsByClassName(classname)** - This method returns a collection of all elements reference in the document with the specified class name.
- **document.getElementById(id)** - This method returns a element reference in the document with the specified id.

Modifying HTML Using innerHTML

- innerHTML is the property of DOM object nodes. Using this property we can get/set the html inside a tag.

  Example :

```
<head> <script type = "text/javascript">
    function addHeading(){
    var ref = document.getElementById("container");
    var htmlToInsert= '<h3> This is the Heading</h3>';
    ref.innerHTML = htmlToInsert;
        }  </script>
</head>
<body>  <button  onclick = "addHeading()">Add Heading</button>
            <div id="container"></div>    </body>
```



Events

Events

- **JavaScript** can respond to events which can also be actions by the user.
  - **Example**, clicking on an element, hovering over an element are all actions by user and JavaScript uses events which can react to these actions.
- **JavaScript** attaches a function called an event listener or event handler to a specific event and the function invokes when the event occurs.

Events can be attached in the following ways:

1. Inline HTML attributes
2. Adding to element properties with JavaScript
3. Using DOM Event Listeners

Inline HTML Elements

- Events can be attached as attributes to the elements as shown below:

  Click

Adding to Element Properties

- We can also assign a function to the onclick property of a DOM node element. **Have a look at the code snippet below:**

```
<div id = “container”>click here</div>
<script type = “text/javascript”>var ref = document.getElementById('container');
ref.onclick = function () {
alert('The div area is clicked');
};
</script>
```

Using DOM Event Listener

- The best way to handle events is to use the event listener approach.
- We can assign listeners to the click event using the **addEventListener()** method.

  ref.addEventListener(event,function)
- **addEventListener()** method attaches an event handler to the specified element.
- You can add event listeners to any DOM object.
- The **removeEventListener() method** removes event handlers that have been attached with the addEventListener() method.

Syntax:

element.removeEventListener(“Event Name", function);

Event Types

- **Mouse Events** – mouseup, mousedown, mouseover, mousemove, etc.
- **Keyboard Events** – keydown, keypress and keyup
- **Window Events** – load, unload
- **Form events** – focus, change

Event Bubbling and Event Capturing

- **Event Propagation** is the way of defining the element order when an event occurs.
- Two ways of event propagation in the HTML DOM:
  - Bubbling and Capturing

- **Event Bubbling:** inner most element's event is handled first and then the outer element's event.
- **Event Capturing:** outer most element's event is handled first and then the inner element's event.

Action Dialog

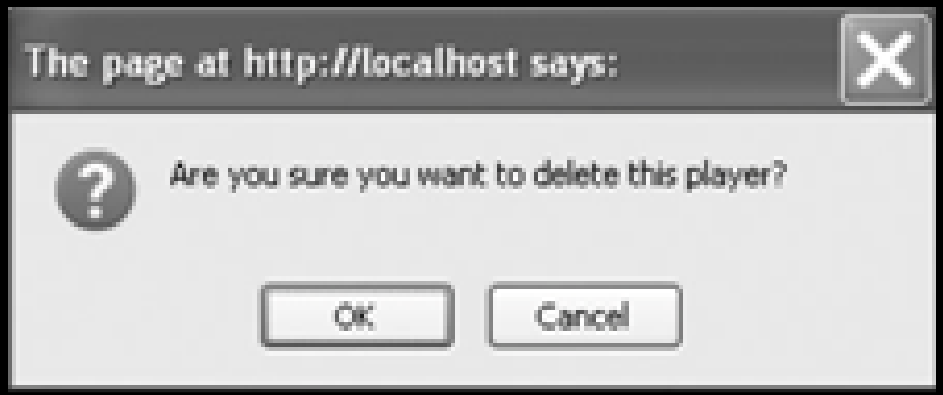
```
<script type="text/javascript">

        function confirmDelete() {
          var answer = confirm("Are you sure you want"
            + "to delete this player?");
          return answer
        }




</script>

<form method="post" action="/delet
            <p>
            <input type="submit" value="Delete“  onclick="return confirmDelete()" />
            </p>
</form>
```



Form Validation

```
<script>
 function validate() {
   if (document.getElementById("name").value.length == 0) {
     alert("Please complete the required fields\n" +
       "and resubmit.");
     return false;
   }
   return true;
 }
</script>
```


```
<h3>Add Player:</h3>
<form id="form1" action="addplayer“ onsubmit="return  validate()" >
 <p>Name: <input type="text" id="name" /></p>
  ...
 <p><input type="submit" value="Register" /></p>
</form>
```

Lets Discuss Assignments

Assignment