

Lab session 1 – JavaScript

Objective: Teach a set of basic concepts in the JavaScript programming language.

Prerequisites: Students should have basic JavaScript knowledge.

a. JavaScript Objects

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

<h2>JavaScript Objects</h2>
<!--In JavaScript, an object is a standalone entity, with properties
and type.-->
<p id="demo"></p>

<script>

// Animal properties and method encapsulation
const Animal = {
  type: "Invertebrates", // Default value of properties
  displayType() {
    // Method which will display type of Animal
    console.log(this.type);
  },
};

// Create new animal type called animal1
const animal1 = Object.create(Animal);
animal1.displayType(); // Logs: Invertebrates

// Create new animal type called fish
const fish = Object.create(Animal);
fish.type = "Fishes";
fish.displayType(); // Logs: Fishes

</script>

</body>
```

```
</html>
```

b. JavaScript Closure

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

<h2>JavaScript Closure</h2>

<!--
  A closure is a function having access to the parent scope,
  even after the parent function has closed.
-->

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

<script>

//a closure gives you access to an outer function's scope from an
inner function.

function greeting() {
  let message = 'Hi';

  function sayHi() {
    console.log(message);
  }

  return sayHi;
}
let hi = greeting();
hi(); // still can access the message variable'

</script>

</body>
</html>
```

c. JSON Placeholder API

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

```

<h2>JSON Placeholder API</h2>

<!--An application programming interface is a way for two or more
computer programs to communicate with each other.
    It is a type of software interface,
    offering a service to other pieces of software. -->

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

<script>

//https://jsonplaceholder.typicode.com/
//Free fake API for testing and prototyping.

fetch('https://jsonplaceholder.typicode.com/todos/1')
    .then(response => response.json())
    .then(json => console.log(json))

</script>

</body>
</html>

```

ES6 New Features

d. Classes

a. Create a simple class constructor.

```

<!DOCTYPE html>
<html>

<body>

<script>

//What is this? In JavaScript, the this keyword refers to an
object.
//Which object depends on how this is being invoked (used or
called).
//The this keyword refers to different objects depending on how it
is used:
//In an object method, this refers to the object.

class Car {
  constructor(name) {
    this.brand = name;
  }
}

```

```

    }

    present() {
        return 'I have a ' + this.brand;
    }
}

const mycar = new Car("Ford");
document.write(mycar.present());
</script>

</body>
</html>

```

- b. Create a class and, define a method inside the class, after that, create an object from the class and execute the methods.

```

<!DOCTYPE html>
<html>

<body>

<script>
class Car {
    constructor(name) {
        this.brand = name;
    }

    present() {
        return 'I have a ' + this.brand;
    }
}

const mycar = new Car("Ford");
document.write(mycar.present());
</script>

</body>
</html>

```

- c. Class inheritance – create a class (base class) and create another class, derived from base class that you created and make a method within each class and, execute method within derived class by creating an object of derived class and then, execute the base class's method via that object.

```

<!DOCTYPE html>

```

```

<html>

<body>

<script>
class Car {
  constructor(name) {
    this.brand = name;
  }

  present() {
    return 'I have a ' + this.brand;
  }
}

class Model extends Car {
  constructor(name, mod) {
    super(name);
    this.model = mod;
  }
  show() {
    return this.present() + ', it is a ' + this.model
  }
}

const mycar = new Model("Ford", "Mustang");
document.write(mycar.show());
</script>

</body>
</html>

```

e. Variables

- a. “var”, “let” and “const” variables try their behaviors.

```

<!DOCTYPE html>
<html>

<body>

<script>

let a = 10;
  function f() {
    if (true) {

```

```
        let b = 9

        // It prints 9
        console.log(b);
    }

    // It gives error as it
    // defined in if block
    console.log(b);
}
f()

// It prints 10
console.log(a)

</script>

<p>Press F12 and see the result in the console view.</p>

</body>
</html>
```

```
<!DOCTYPE html>
<html>

<body>

<script>

const a = {
    prop1: 10,
    prop2: 9
}

// It is allowed
a.prop1 = 3

// It is not allowed
a = {
    b: 10,
    prop2: 9
}

</script>
```

```
<p>Press F12 and see the result in the console view.</p>
</body>
</html>
```

f. Array methods

a. Map a list of items from an array.

```
<!DOCTYPE html>
<html>

<body>

    <h1 id="demo"></h1>

<script>
const array1 = [1, 4, 9, 16];

// Pass a function to map
const map1 = array1.map(x => x * 2);

document.getElementById("demo").innerHTML = map1;
// Expected output: Array [2, 8, 18, 32]

</script>

</body>
</html>
```

g. Destructuring

a. Use destructuring when a function returns an array.

```
<!DOCTYPE html>
<html>

<body>

<script>
function calculate(a, b) {
    const add = a + b;
    const subtract = a - b;
    const multiply = a * b;
    const divide = a / b;
```

```

    return [add, subtract, multiply, divide];
}

const [add, subtract, multiply, divide] = calculate(4, 7);

document.write("<p>Sum: " + add + "</p>");
document.write("<p>Difference " + subtract + "</p>");
document.write("<p>Product: " + multiply + "</p>");
document.write("<p>Quotient " + divide + "</p>");
</script>

</body>
</html>

```

- b. Destructure deeply nested objects by referencing the nested object then using a colon and curly braces to again destructure the items needed from the nested object.

```

<!DOCTYPE html>
<html>

<body>

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

<script>
const vehicleOne = {
  brand: 'Ford',
  model: 'Mustang',
  type: 'car',
  year: 2021,
  color: 'red',
  registration: {
    city: 'Houston',
    state: 'Texas',
    country: 'USA'
  }
}

myVehicle(vehicleOne)

function myVehicle({ model, registration: { state } }) {
  const message = 'My ' + model + ' is registered in ' + state +
  '.';

  document.getElementById("demo").innerHTML = message;
}

```



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