# JavaScript Submission 3

1. What is “closure” in javascript? Can you provide an example?

A closure in JavaScript is a function that retains access to its lexical scope, even when executed outside that scope. This means the function can remember and access variables from its outer function, even after the outer function has completed. Closures enable data privacy and the creation of functions with preserved state.

```
function createCounter() {
  let count = 0;
  return {
    increment: function() {
      count++;
      return count;
    },
    decrement: function() {
      count--;
      return count;
    },
    getCount: function() {
      return count;
    }
  };
}

const counter = createCounter();
console.log(counter.increment()); // Outputs: 1
console.log(counter.increment()); // Outputs: 2
console.log(counter.decrement()); // Outputs: 1
console.log(counter.getCount());  // Outputs: 1
```

2. What are promises and how are they useful?

Promises in JavaScript are objects representing the eventual completion or failure of an asynchronous operation. They allow you to write asynchronous code in a more manageable and readable way, avoiding deeply nested callbacks. Promises provide methods like `.then()` and `.catch()` to handle the resolved value or error, enabling cleaner and more structured code flow.

3. How to check whether a key exists in a JavaScript object or not.
To check if a key exists in a JavaScript object, use the `in` operator: `"key" in object`.
Another way is to use `hasOwnProperty`: `object.hasOwnProperty("key")`.
Additionally,and if the key is `undefined` using `object["key"] !== undefined`.

4. What is the output of this code? **Please explain**

```
var employeeId = 'abc123';


  function foo() {

   employeeId();

   return;


   function employeeId() {

   console.log(typeof employeeId);

   }

  }

  foo();




  // output function
```

When `foo` is called, the hoisted function `employeeId` within `foo` shadows the outer variable `employeeId`. Therefore, the `employeeId` inside `foo` refers to the function `employeeId` defined within it, not the global variable `employeeId`.

Inside the function `employeeId`, `console.log(typeof employeeId)` is executed. Within this function, `employeeId` refers to the function itself, so `typeof employeeId` evaluates to `"function"`.

Therefore output is function

5. What is the output of the following? **Please explain**

```
(function() {
'use strict';

var person = {
name: 'John'
};
person.salary = '10000$';
person['country'] = 'USA';

Object.defineProperty(person, 'phoneNo', {
value: '8888888888',
enumerable: true
})

console.log(Object.keys(person));
})();

// output
['name', 'salary', 'country', 'phoneNo']
```

6. What is the output of the code? Explain

```
(function() {
var objA = {
foo: 'foo',
bar: 'bar'
};
var objB = {
foo: 'foo',
bar: 'bar'
};
console.log(objA == objB);
console.log(objA === objB);
```

```
       }());

       // false
       // false
```

7. What is the output of the following code:

```
function Person(name, age){
 this.name = name || "John";
 this.age = age || 24;
 this.displayName = function(){
 console.log(this.name);
 }
}

Person.name = "John";
Person.displayName = function(){
 console.log(this.name);
}

var person1 = new Person('John');
 person1.displayName();
 Person.displayName();

// output
// John
// Person
```

## 8. In-Class Exercise: Designing a School Management System

**Scenario:**

You are tasked with designing a School Management System for a school. The system should manage students, teachers, courses, and their interactions.

**Exercise Instructions:**

1. **Identify Classes:**
   ○ List down the main entities (classes) that you think are necessary for the School Management System. Consider entities like `Student`, `Teacher`, `Course`, etc. 2. **Define Class Properties:**
   ○ For each identified class, define the properties (attributes) that would be essential to store information. For example, `Student` class might have properties like `id`, `name`, `email`, etc.

3. **Define Class Methods:**
   ○ Specify the methods (functions) that each class should have. Think about what actions each class needs to perform. For instance, `Student` might need methods like `enroll(course)`, `getGrades()`, etc.
4. **Class Relationships:**
   ○ Determine how classes will interact with each other. For example, how will a `Teacher` assign a `Course` to a `Student`? How will a `Course` keep track of enrolled `Students`?
5. **Write Sample Code:**
   ○ Write a basic implementation in JavaScript using classes and methods you've defined. This step can help reinforce understanding through practical application.

```
class Student {
    constructor(id, name, email) {
        this.id = id;
        this.name = name;
        this.email = email;
        this.courses = [];
    }

    enroll(course) {
        this.courses.push(course);
        course.addStudent(this);
    }

    getCourses() {
        return this.courses;
    }
}

class Teacher {
    constructor(id, name, email) {
        this.id = id;
        this.name = name;
        this.email = email;
        this.courses = [];
    }

    assignCourse(course) {
        this.courses.push(course);
        course.setTeacher(this);
    }

    getCourses() {
        return this.courses;
    }
}

class Course {
    constructor(id, name, description) {
        this.id = id;
        this.name = name;
        this.description = description;
        this.students = [];
        this.teacher = null;
    }
    addStudent(student) {
        this.students.push(student);
    }

    setTeacher(teacher) {
        this.teacher = teacher;
    }

    getStudents() {
        return this.students;
    }

    getTeacher() {
        return this.teacher;
    }
}

class School {
    constructor(name) {
        this.name = name;
        this.students = [];
        this.teachers = [];
        this.courses = [];
    }

    addStudent(student) {
        this.students.push(student);
    }

    addTeacher(teacher) {
        this.teachers.push(teacher);
    }

    addCourse(course) {
        this.courses.push(course);
    }
```

```javascript
    getStudents() {
        return this.students;
    }

    getTeachers() {
        return this.teachers;
    }

    getCourses() {
        return this.courses;
    }
}

// Example usage:
const school = new School('Jaffna College');

const student1 = new Student(1, 'Juvi', 'juvi@example.com');
const student2 = new Student(2, 'Juju', 'juju@example.com');

const teacher1 = new Teacher(1, 'Teacher1', 'teacher@example.com');

const course1 = new Course(1, 'Mathematics', 'An introduction to
mathematics.');
const course2 = new Course(2, 'Science', 'An introduction to science.');

school.addStudent(student1);
school.addStudent(student2);

school.addTeacher(teacher1);

school.addCourse(course1);
school.addCourse(course2);

student1.enroll(course1);
student2.enroll(course2);
```

```
teacher1.assignCourse(course1);

console.log(school.getStudents());
console.log(school.getTeachers());
console.log(school.getCourses());
console.log(course1.getStudents());
console.log(course1.getTeacher());
console.log(teacher1.getCourses());
```