JS Submission 3

Jegapalini Jegatheeswaran

UKI STU 873

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

A closure is the combination of a function bundled together (enclosed) with references to its surrounding state (the lexical environment). In other words, a closure gives you access to an other function’s scope from an inner function. In JavaScript, closures are created every time a function is created, at function creation time.

function outerFunction() {

let outerVariable = 'I am outside!';

function innerFunction() {

console.log(outerVariable); // Accesses the variable from the outer function

}

return innerFunction;

}

const myClosure = outerFunction();

myClosure(); // Logs: 'I am outside!'

**2. What are promises and how are they useful?**

Promises in JavaScript are objects that represent the eventual completion (or failure) of an asynchronous operation and its resulting value. They are a way to handle asynchronous operations more effectively and avoid callback hell, providing a cleaner and more readable code structure.

To handle the result of a promise, you use the **.then(), .catch(), and .finally()** methods.

**myPromise**

**.then((result) => {**

**console.log(result); // Logs: "Operation was successful!"**

**})**

**.catch((error) => {**

**console.error(error); // Logs: "Operation failed."**

**})**

**.finally(() => {**

**console.log("Operation completed.");**

**});**

**3. How to check whether a key exists in a JavaScript object or not.**

\* Using the ‘in’ Operator

\*Using ‘hasOwnProperty’ method

\*Using ‘Object.hasOwn’

\*Using ‘underfined’ Check

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

var employeeId = 'abc123';

function foo() {

employeeId();

return;

function employeeId() {

console.log(typeof employeeId);

}

}

foo();

Out put: function

**Variable Declaration:**

var employeeId = 'abc123';

**Function Declaration:**

function foo() {

employeeId();

return;

function employeeId() {

console.log(typeof employeeId);

}

}

**Calling foo:**

foo();

**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));

})();

**1.Self-Invoking Anonymous Function:**

(function() {

'use strict';

// Function body

})();

**2.Object Creation and Property Assignment:**

var person = {

name: 'John'

};

**3.Adding Properties:**

person.salary = '10000$';

person['country'] = 'USA';

**4.Using Object.defineProperty to Add Property:**

Object.defineProperty(person, 'phoneNo', {

value: '8888888888',

enumerable: true

});

**5.Logging the Keys of the Object:**

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

the final output is:

["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);

}());

The output of the provided code is:

false

false

**1.Self-Invoking Anonymous Function:**

(function() {

// Function body

})();

**2.Object Creation:**

var objA = {

foo: 'foo',

bar: 'bar'

};

var objB = {

foo: 'foo',

bar: 'bar'

};

**3.Comparison Using ==:**

console.log(objA == objB);

**4.Comparison Using ===:**

console.log(objA === objB);

**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();

The output of the provided code is:

John

John

**1.Function Constructor Definition:**

function Person(name, age){

this.name = name || "John";

this.age = age || 24;

this.displayName = function(){

console.log(this.name);

}

}

**2.Adding Static Properties and Methods to Person:**

Person.name = "John";

Person.displayName = function(){

console.log(this.name);

}

**3.Creating an Instance of Person:**

var person1 = new Person('John');

person1.displayName();

**4.Calling the Static Method on Person:**

Person.displayName();

**8.**

**// Define the Student class**

**class Student {**

**constructor(id, name, email) {**

**this.id = id;**

**this.name = name;**

**this.email = email;**

**this.courses = [];**

**}**

**enroll(course) {**

**this.courses.push(course);**

**course.enrollStudent(this);**

**}**

**drop(course) {**

**this.courses = this.courses.filter(c => c !== course);**

**course.dropStudent(this);**

**}**

**getGrades(course) {**

**// Implementation to get grades for the student in a specific course**

**// Assuming grades are stored somewhere**

**}**

**}**

**// Define the Teacher class**

**class Teacher {**

**constructor(id, name, email) {**

**this.id = id;**

**this.name = name;**

**this.email = email;**

**this.coursesTaught = [];**

**}**

**assignCourse(course) {**

**this.coursesTaught.push(course);**

**course.assignTeacher(this);**

**}**

**getStudents(course) {**

**return course.getStudents();**

**}**

**enterGrades(student, course, grades) {**

**// Implementation to enter grades for a student in a specific course**

**// Assuming grades are stored somewhere**

**}**

**}**

**// Define the Course class**

**class Course {**

**constructor(id, name) {**

**this.id = id;**

**this.name = name;**

**this.teacher = null;**

**this.students = [];**

**}**

**enrollStudent(student) {**

**this.students.push(student);**

**}**

**dropStudent(student) {**

**this.students = this.students.filter(s => s !== student);**

**}**

**assignTeacher(teacher) {**

**this.teacher = teacher;**

**}**

**getStudents() {**

**return this.students;**

**}**

**getTeacher() {**

**return this.teacher;**

**}**

**}**

**// Example usage:**

**let student1 = new Student(1, 'Alice', 'alice@example.com');**

**let teacher1 = new Teacher(101, 'Mr. Smith', 'smith@example.com');**

**let course1 = new Course(1001, 'Mathematics');**

**teacher1.assignCourse(course1);**

**student1.enroll(course1);**

**console.log(course1.getTeacher()); // Output: Teacher { id: 101, name: 'Mr. Smith', email: 'smith@example.com', coursesTaught: [ Course { id: 1001, name: 'Mathematics', teacher: [Circular] } ] }**

**console.log(course1.getStudents()); // Output: [ Student { id: 1, name: 'Alice', email: 'alice@example.com', courses: [ [Circular] ] } ]**