JavaScript Questions

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

   A closure is a function that retains access to its own scope, outer function's scope, and global scope.

   Example:

```
//closure
function outer(){
    let a = 'I am from outer function';

    function inner(){
        let b = 'I am from inner function';
        console.log(a);
    }
    return inner;
}
const closureFunc = outer();
closureFunc(); // output: "I am from outer function"
```

2. What are promises? And how are they useful?

   Promises in JavaScript represent the eventual result of an asynchronous operation, providing a cleaner way to handle success (fulfilled) or failure (rejected) compared to traditional callbacks.

   Example:

```
//promises
let promise = new Promise((resolve, reject) => {
    setTimeout(() => {

        resolve('Promise resolved');
    }, 2000);

});

promise.then((data) => {
    console.log(data); //output: Promise resolved after 2 seconds
});
```

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

   *To check if a key exists in a JavaScript object, you can use the **in** operator or the **hasOwnProperty** method.*

   Example:

```javascript
//checking whether a key exist in javascript object
//using in operator
const obj = {
    name: 'John',
    age: 28
};
console.log('name' in obj); //output: true
console.log('salary' in obj); //output: false

//using hasOwnProperty method
console.log(obj.hasOwnProperty('name')); //output: true
console.log(obj.hasOwnProperty('salary')); //output: false
```

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

```javascript
//Question 4
var employeeId = 'abc123';

function foo() {
 employeeId();
return;

function employeeId() {
console.log(typeof employeeId);

  }
}
 foo();
//output: function
```

Explanation:

In foo, "employeeId" refers to a hoisted local function, not the global variable, so typeof employeeId outputs "function".

5. What is the output of the following? Explain

```javascript
//question 5
(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"]
})();
```

Explanation:

*An IIFE (Immediately Invoke function Expression) with **'use strict'** creates an object person with an initial property name: 'John'. It then adds properties salary: '10000$' and country: 'USA' directly to the object. Using **Object.defineProperty**, it adds the property phoneNo: '8888888888' and makes it enumerable. The **Object.keys(person)** method is*

used to return an array of all enumerable property names of the person object, resulting in ["name", "salary", "country", "phoneNo"].

6. What is the output of the code? Explain

```
//question 6
(function(){
    var objA ={
        foo: 'foo',
        bar: 'bar'
    };
    var objB = {
        foo: 'foo',
        bar: 'bar'
    };
    console.log(objA == objB); //output: false
    console.log(objA === objB); //output: false
}());
```

Explanation:

objA and objB, with identical properties. Both == and === comparisons between objA and objB evaluate to false due to reference inequality.

- ✓ *objA == objB is false because objA and objB are different objects.*
- ✓ *objA === objB is false for the same reason; strict equality also checks for reference equality.*

7. What is the output of the following code:

```
//question 7
function person1(name, age) {
    this.name = name || "John";
    this.age = age || 24;
    this.displayName = function() {
      console.log(this.name);
    }
  }

  person1.name = "John";
  person1.displayName = function() {
    console.log(this.name);
  }

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

Explanation:

The Person constructor creates objects with optional name and age parameters, defaulting to "John" and 24 respectively. It includes an instance method ***displayName*** *to log the object's name. Static properties on the Person function set its own name property and define a method to log it, applicable even without an instance.*

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:

I. 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.

II. 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.

III. 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.

IV. 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?

V. 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.

Link for question 8:

https://github.com/safra-siyam/JS_SUB_3/blob/main/sms.js