# JS Assignment

**What will this code log? Explain your reasoning.**

*const person = {*

*name: 'Alice',*

*sayHi: function() {*

*console.log(`Hi, my name is ${this.name}.`);*

*}*

*};*

*person.sayHi();*

person is an **object** with two properties:

* name: a string 'Alice'
* sayHi: a **function** that logs a greeting using this.name

When we call person.sayHi(), the function runs in the **context of the** person **object,** so this refers to person.

Therefore, this.name is 'Alice', and the output becomes: Hi, my name is Alice.

**What is logged to the console, and why? How would you fix it so it logs "Alice"?**

*const person = {*

*name: 'Alice',*

*greet: function() {*

*console.log(`Hello, ${this.name}`);*

*}*

*};*

*const greetFunction = person.greet;*

*greetFunction();*

This will log: Hello, undefined.

When greetFunction() is called, **this** no longer refers to object person. Instead, it becomes **undefined.** So we could use bind to connect the object to the function in following way;

Const greetFunction=person.greet.bind(person)

Else we could also call the function directly via the object as;

Person.greet();

**What will this code log and why?**

*const user = {*

*name: 'Bob',*

*logName: function() {*

*setTimeout(() => {*

*console.log(this.name);*

*}, 100);*

*}*

*};*

*user.logName()*

This will log ‘Bob’ with a delay of 100ms due to the setTimeout function set to 100.

**You have a User object with a method that logs a welcome message. There's also a "Login" button in your HTML. You want the welcome method to be called when the button is clicked, and the message should use the User object's data.**

*<button id="loginBtn">Click me</button>*

*const User = {*

*name: 'Alice',*

*welcome: function() {*

*console.log(`Welcome, ${this.name}!`);*

*}*

*}*

*document.getElementById('loginBtn').addEventListener('click', User.welcome.bind(User));*

**What will be logged to the console? Explain!**

*let animal = 'Cat';*

*function showAnimal() {*

*let animal = 'Dog';*

*console.log(animal);*

*}*

*showAnimal();*

*console.log(animal);*

The variable animal is 'Cat' in the global scope. Inside the function showAnimal(), a new local variable animal is declared with the value 'Dog'. So when showAnimal() is called, it logs 'Dog'. After that, logging animal from the global scope prints 'Cat' because the global variable remains unchanged. So it logged:

Dog

Cat

**What will this code output? Why?**

*function test() {*

*console.log(a);*

*console.log(foo());*

*var a = 1;*

*function foo() {*

*return 2;*

*}*

*}*

*test();*

The code logs undefined and 2 because of hoisting. The variable a declared with var is hoisted but not initialized, so console.log(a) prints undefined. The function foo is fully hoisted, so console.log(foo()) calls the function and prints 2.

**Explain the output of this for loop.**

*for (var i = 0; i < 5; i++) {*

*setTimeout(function() {*

*console.log(i);*

*}, 10);*

*}*

The variable i is declared with var, which means **function-scoped**, not block-scoped. By the time the setTimeout callbacks run (after ~10ms), the for loop has **already completed**, and i has the value 5. All the functions in setTimeout **share the same** i, so each logs 5.

**What will the following code log to the console on the last two lines? Explain why the count variable is not reset.**

*function createCounter() {*

*let count = 0;*

*return function() {*

*count++;*

*console.log(count);*

*};*

*}*

*const counter = createCounter();*

*counter();*

*counter();*

The code logs 1 and 2 because of how JavaScript closures work. When createCounter() is called, it defines a local variable count and returns a function that increments and logs this variable. Even though createCounter() finishes executing, the returned function retains access to its original count variable through a closure. This means that each time counter() is called, it continues using and updating the same count variable instead of resetting it. As a result, the first call logs 1, and the second call logs 2. The count variable is preserved between calls because it lives in the function’s closed-over scope, not in the global or re-created context.

**What does this code log? Explain!**

*const myObject = {*

*id: 'my-object',*

*createLogger: function() {*

*return () => {*

*console.log(`Logger for ${this.id}`);*

*};*

*}*

*};*

*const logger = myObject.createLogger();*

*logger();*

It logs "Logger for my-object" because the arrow function inside createLogger captures this from its outer scope (myObject), and retains that reference even when called outside.

**Write a function *makeAdder(x)* that takes a number x and returns a new function. The new function should take a number y and return the *sum x + y*. Use a closure to achieve this.**

*Eg:*

*let add = makeAdder(5);*

*console.log(add(2));*

*Answer: 7*

*=>*

*function makeAdder(x){*

*return(y)=>{*

*return x+y*

*}*

*}*

*Let add=makeAdder(5)*

*console.log(add(2))*

**Implement a parent Animal and a child Dog relationship in two different ways:**

* Using the Constructor/Prototype pattern.
* Using the ES6 class syntax.

Your solution must satisfy two conditions for both patterns:

* The Animal must have an eat() method that the Dog inherits.
* The Dog must also have its own bark() method.

Using Constructor method:

function Animal(name) {

this.name = name;

}

Animal.prototype.eat = function() {

console.log(`${this.name} is eating.`);

};

// Child constructor

function Dog(name, breed) {

Animal.call(this, name); // Call parent constructor

this.breed = breed;

}

// Inherit from Animal

Dog.prototype = Object.create(Animal.prototype);

Dog.prototype.constructor = Dog;

// Child method

Dog.prototype.bark = function() {

console.log(`${this.name} is barking.`);

};

const dog1 = new Dog('Buddy', 'Labrador');

dog1.eat(); // Buddy is eating.

dog1.bark(); // Buddy is barking.

Using ES6 class syntax:

// Parent class

class Animal {

constructor(name) {

this.name = name;

}

eat() {

console.log(`${this.name} is eating.`);

}

}

// Child class

class Dog extends Animal {

constructor(name, breed) {

super(name); // Call parent constructor

this.breed = breed;

}

bark() {

console.log(`${this.name} is barking.`);

}

}

// Example

const dog2 = new Dog('Max', 'Beagle');

dog2.eat(); // Max is eating.

dog2.bark(); // Max is barking.

**Design a content feed/post UI (similar to X/Twitter) involving Post, Comment, and User details.[Using either fetch API or axios and DOM APIs]**

Use the **API** from Resources section: <https://jsonplaceholder.typicode.com/>

**Features:**

* Feed View (like X/Twitter)
* Render a list of posts
* Each post should display
  + The user (owner) who created the post
  + The comments related to the post
  + When a user's name is clicked, navigate to their profile page.

**User Profile View:**

* Display the basic details of the user
* Show a photo gallery of the user(**/albums** and **/photos API**)
* Show a list of todos associated with the user**(/todos api)**

**Hint**: *All entities (posts, comments, photos, todos) are interconnected using the userId(users). Use userId to associate and map the relevant data.*

**Let there be two files in the same folder: *index.html* and *index.js*. Update code in index.js so that the counter app works perfectly.**

***index.html***

*<!DOCTYPE html>*

*<html lang="en">*

*<head>*

*<meta charset="UTF-8" />*

*<meta name="viewport" content="width=device-width, initial-scale=1.0" />*

*<title>Counter</title>*

*<script src="https://cdn.tailwindcss.com"></script>*

*</head>*

*<body class="bg-gray-100 h-screen flex items-center justify-center font-sans">*

*<div class="bg-white rounded-xl shadow-lg text-center py-10 px-16">*

*<div id="counter-value" class="text-6xl font-bold text-gray-800 mb-8">*

*0*

*</div>*

*<div class="flex justify-center gap-5">*

*<button*

*id="decrement"*

*class="py-4 px-8 text-lg bg-indigo-500 text-white rounded-lg transition duration-200 hover:bg-indigo-600 active:scale-95"*

*>*

*Decrement*

*</button>*

*<button*

*id="increment"*

*class="py-4 px-8 text-lg bg-indigo-500 text-white rounded-lg transition duration-200 hover:bg-indigo-600 active:scale-95"*

*>*

*Increment*

*</button>*

*</div>*

*</div>*

*<script src="./index.js"></script>*

*</body>*

*</html>*

***index.js***

*class Counter {*

*constructor() {*

*this.value = 0;*

*this.decrementBtn = document.getElementById("decrement");*

*this.incrementBtn = document.getElementById("increment");*

*this.counter = document.getElementById("counter-value");*

*this.counter.innerText = this.value;*

*this.incrementBtn.addEventListener("click", () => this.increment());*

*this.decrementBtn.addEventListener("click", () => this.decrement());*

*}*

*increment() {*

*this.value++;*

*this.counter.innerText = this.value;*

*}*

*decrement() {*

*this.value--;*

*this.counter.innerText = this.value;*

*}*

*}*

*const counter = new Counter();*