Short Answer:

Answer the following questions with complete sentences in **your own words**. You are encouraged to conduct your own research online or through other methods before answering the questions. If you research online, please consult multiple sources before you write down your answers.

1. **What are the different type of scopes?**

There global scope, which covers all code, functional scope, which covers inside a function, and block scope, which covers inside {} within if, for, and while

2. **What is hoisting?**

Hosting allows you to use functions and variables before they're declared.

3. **Explain the differences between var, let, & const.**

-var: scoped defined and it's limited to a function; if it's defined outside of function, it's global. it's a function scoped variable
-let: scoped defined and limited to inside the block: curly braced inside will work. outside curly brace will not work
-const: scoped defined and limit to inside block. However, it cannot be changed or reassigned. It can be mutated though;

4. **What is an execution context? How does it relate to the call stack?**

It is the environment where a function executes variable scope function arguments, and the values of this object.

5. **What is the scope chain? How does lexical scoping work?**

Scope chain is a way of looking for variable within the current scope, then out to the outer scope until it reaches the global scope.

Lexical scoping can access variable that is defined outside. However, Outside variables cannot access variable defined inside a function, in then clause, loops or anything within {} because it is not within its scope.

6. **What is a closure? Can you provide an example use-case in words?**

Closure helps give access to outer function's scope from an inner function. Examples include imitating private data and methods in class for currying and callback functions.

7. **What is an IIFE? When would you use it?**

IIIFE is a function that runs as soon it is defined by the user. It is used to prevent global variable definition issues, alias variable protect private data, and avoid conflict with program that have the same object or variable name.

11. What does 'this' refer to in the cases that were discussed in lecture?

This refers to an object that is executing the current piece of code. This refers depends on where it is invoked and where in the scope it is defined.

13. **What are the differences between call, apply & bind?**

call: binds the this value, invokes the function, and allows you to pass a list of arguments.
apply: binds the this value, invokes the function, and allows you to pass arguments as an array.
bind: binds the this value, returns a new function, and allows you to pass in a list of arguments.

14. **Can you name the new ES6 features?**

ES6 uses the variables such as Let and Cost, uses Arrow Function, Template string literals, Map and Sets, and Spread Operators

15. **What is 'use strict'? What are the major effects that it has?**

"use strict" is used to help write secure codes by putting it in strict mode. Due to this, it cannot use undeclared values, cannot delete variables or objects or cannot have duplicate function argument names

16. **What is currying?**

Currying involves a process that transforms a function with multiple arguments into a sequence of nesting functions. It returns a new function that expects the next argument inline.

17. **What are ES6 modules? Why are they useful? How do we make these modules available in other JS files?**

ES6 modules are files containg Javascript. They are useful because it allows you to load and manage dependencies through import and export keywords. Both the import and export keyword make these modules available in other JS files.

18. **What is Object-Oriented Programming (OOP)?**

OOP is a programming paradigm that relies on the concept of classes and objects. It provides structure a software program into simple, reusable pieces of code through classes, which are used to create individual instances of objects

19. **What is prototype-based OOP in JS?**

Prototype-based OOP is a style of object-oriented programming where inheritance is implemented by cloning existing objects that serve as prototypes. Two methods of constructing new objects include creating an object from nothing or through cloning an existing object.

20. **What is the prototype chain?**

Prototype chain is a cycle of looking up object's property on the current object until you either reach null, or you find the property you're looking for.

21. **How do you implement inheritance in JavaScript before ES6 and with ES6?**

Before ES6, inheritance would involve using Constructor functions and prototypal inheritance. ES6 made this Easier by using Classes with extends and super keywords.

Coding Questions:

Use HTML/CSS/JS to solve the following problems. You are highly encouraged to present

more than one way to answer the questions. Please follow best practices when you write the code so that it would be easily readable, maintainable, and efficient. Clearly state your assumptions if you have any. You may discuss with others on the questions, but please write your own code.

1. Consider the following code snippet. Assume that it works and was imported into a .html file with the proper button IDs.

```
for (var i = 1; i <= 3; i++) {
  document.getElementById(`btn${i}`).addEventListener('click', function () {
  console.log(`you just clicked #${i} button`);
}) }
```

a) What is the console output when the user clicks on “Button 3” and why?

b) How would we fix the issue before ES6? How do we fix it after ES6?

2. Consider the following code snippet. Create a function named **add2** that does the exact same thing but can be invoked in this way: **add2(num1)(num2)(num3)**

```
function add(num1, num2, num3) {
  return num1 + num2 + num3;
}
```

3. Please implement the following classes in both pre-ES6 (prototypes) & ES6 (class) styles.
- Create a Vehicle class that contains the properties **engine** and **speed**.
  - Add a method **info()**, which logs the engine & speed values.
- Create a Car class that inherits from the Vehicle class.
  - Add more properties **wheels** and **brake**
  - Add a method **honk()**, which logs “Honk!”.
  - Add a static method **isTesla(car)**, which takes an argument car object and returns true if its brake property is true, otherwise false.

Note: Static methods are invoked by calling it on the class itself, so Car.isTesla().

Paired Programming:

Use JS to solve the following problems with your partner. Please remember to label who was acting as the driver and navigator for each problem and write down your feedback on their performance. **Feedback will be confidential**.

Leetcode #13: Roman to Integer
Leetcode #14: Longest Common Prefix

Optional Questions:

The following are very similar to what you may see during an interview. Feel free to type out the following code and run it to see the output. You still need to explain why the output is what it is.

```
// 1. What is the output of this code?
var dataObj = {
  data: "xyz",
  functionA: function () {
    var self = this;
    console.log("outer function: this.data = " + this.data);
    console.log("outer function: self.data = " + self.data);
    (function () {
      console.log("inner function: this.data = " + this.data);
      console.log("inner function: self.data = " + self.data);
    })();
  }
};
dataObj.functionA();

// 2. What is the output of this code and why?
var x = 1;
var fn = function () {
  console.log(x);
  var x = 2;
};
fn();

// 3. What is the output of this code and why?
var b = 1;
function outer() {
  var b = 2;
  function inner() {
    b++;
    var b = 3;
    console.log(b);
  }
  inner();
}
outer();
```

```
// 4. What is the output of this code and why?
(function (x) {
  return (function (y) {
    console.log(y);
  })(2);
})(1);

(function (x) {
  return (function (y) {
    console.log(x);
  })(2);
})(1);

// 5. What is the output of this code and why?
var user = {
  _name: 'Username1',
  printName: function () {
    console.log(this._name);
  }
};
var printName = user.printName;
printName();
user.printName();

var user2 = {
  _name: 'Username2',
  printName2: () => {
    console.log(this._name);
  }
};
var printName2 = user2.printName2;
printName2();
user2.printName2();
```