1.What are the three types of execution contexts in JavaScript? Briefly explain each type.	1. Global execution context 2. Function execution context 3. Eval Execution context	1. The default context when the script runs and includes the global object(window or global) and any other global variables/global functions. 2. This context is created when a function is called and includes any arguments, local variables and “this”’s value 3. This context is when the “eval” is used to executes a string of JS code

2.Explain the concept of variable hoisting in JavaScript. Provide an example to illustrate your explanation.	Variable hoisting: is when variables and functions are declared and moved to the tops of their scopes. This makes it so that the variable can be used before being defined and the code runs without throwing an error. The values is still undefined, though.	Example:
		console.log(y); // undefined
		let y = 14;
		console.log(y); // 14

3.What is the difference between 'var', 'let', and 'const' when declaring variables in JavaScript? Provide examples for each.	One difference is scope. Variables declared with “**var**” are apparently function scoped (var is hoisted). On the other hand, “**let** and “**const**” are block-scoped so are limited in being accessible to only the block of code the variable is declared in (they are not hoisted).
	
	Variables declared with “const” cannot change their values after being declared while variables declared with “let” and “var” can.
	
	It is better to use “let” and “const” to declare variables so that information is contained and avoids hoisting bugs.
		//below is example code to demonstrate the differences in values of the 3 types of variable declarations based on it’s scope
		//declare variables “x”, “y”, “z” using var,let, and const in the global scope
		var x = 10;
		let y = 20;
		const z = 30;
		
		//Define a function called 'example'
		function example() {
		// Declares variables “x”, “y”, “z” 'using var within the function scope
		var x = 5;
		let y = 15;
		const z = 25;
		
		//starts a new block of code:(if statement) still in the function.
		if (true) {
		//Declare variables “x”, “y”, “z” within the block scope
		var x = 1;
		let y = 2;
		const z = 3;
		}
		
		//print the value of 'x' within the function (1)
		console.log(x);
		
		//print the value of 'y' within the function (15)
		console.log(y);
		
		//print the value of 'z' within the function (25) //because const cannot be changed once //declared
		console.log(z);
		}
		
		//call the 'example' function
		example();
		
		//print the value of 'x' in the global scope(10)
		console.log(x);
		
		/* attempting to access 'y' and 'z' outside of the function results in a ReferenceError because they are not defined in the global scope */
		console.log(y);
		console.log(z);

4. Explain the concept of scope in JavaScript. How does it relate to execution context and variable environment?	Scope helps explain when variables are visible in different locations of the code. Scope related to execution context and variable environment in that if a variable is declared inside a function, then that variable is only visible inside that function, or if a variable is declared at the top of a code then that variable is visible globally.	

5. Write a JavaScript code snippet to demonstrate the use of an if/else statement and a for loop.	const numbers = [2, 5, 8, 10, 3];
	
	for (let i = 0; i < numbers.length; **i++)** {
	  if (numbers[i] % 2 === 0) {
	  console.log(numbers[i] + " is even");
	  } else {
	  console.log(numbers[i] + " is odd");
	  }
	}
	
	// Output: "2 is even", "5 is odd", "8 is even", "10 is even", "3 is odd"
		//array of numbers to check
		
		//starts at index of [0] and goes through the length of array and **increases the index each time**
		
		// checks if the number at array index i is divisible evenly by 2. If So, it will then console log the number at index [i] + “is even”.
		//else, the only other case is that the number at index [i] would be odd, so it console logs the number at [i] + “is odd”.

6.What is the difference between a named function, an anonymous function expression, and an arrow function? Provide examples for each.	A named function is assigned a name during declaration and formatted differently.
	
	An anonymous function does not have a name.
	
	An arrow function has a name but is also written differently and uses an arrow like so: =>. The arrow is read as “goes to”
		Named function:
		function **add**(a, b) {
		  return a + b;
		}
		
		Anonymous function
		let greet = **function(name) {**
		  return `Hello, ${name}!`;
		}; //this function is stored in the variable greet-it is not named.
		
		Arrow function:

Briefly explain the role of the JavaScript engine in processing a script, and how it handles asynchronous tasks using callback functions, Promises, and async/await syntax.	The JavaScript engine executes code first by interpreting it, compiling it, and then running it. With asynchronous tasks, the engine does not stop the execution of the total code and instead adds it to the “event loop” of tasks needing to be executed. An event loop is a process that is constantly running and constantly monitoring the queue of callbacks and the call stack. Callback functions are called when a task is complete, but are also passed as arguments in another function. Promises represent a place for a future value and are eventually completed in an async operation. Async functions can be chained together by calling .then method. The word “async” is used to define a function when the function will return with a promise. The word “await” is used to put a pause on the code executing until a Promise is fulfilled or rejected.