Javascript:

1. What is **hoisting** ?

   When we execute a javascript code, it goes through 2 phases. One is the creation phase and the next is the execution phase. In this creation phase, javascript skims through our code and hoists our variables at the top.
   So this is the reason when we try to print a variable declared using var before the variable has been initialized we get undefined.

2. What is **var, let and const** ?

   var -
   - Global scope.
   - In the memory allocation phase, the variable declared using var will be hoisted and initialized with undefined.

   let -
   - In the memory allocation phase, the variable will be hoisted and stored in temporal dead zone.
   - Temporal dead zone a temporary storage area where the let and const variable will be stored.
   - Using let and const, we cannot assign the same variable name again in the same scope, it will throw a syntax error.
   - Using let we can initialize a variable and declare a value at a later point of time.
   - If we try to print a let variable before initializing, it will throw a reference error.

   const -
   - Just like let, the variable will be hoisted and stored in temporal dead zone.
   - Const is more strict than let.
   - When we initialize a const variable, we also have to declare a value at the same place. Not like let, else a type error will be throwed.
   - We cannot modify a const variable at a later point of time, it needs to remain constant.

3. What is a **closure** ?

   - A function along with its lexical scope bound together are called closures.
   - That is a function with a scope inside it and another nested function. Now this nested function has access to the variable in its lexical area, which is in its ancestor elements.

4. What is a **block scope** ?

    - Block scope is defined as { … }, and the set of code written within the block are considered as block scope. Eg. if { … }, for loop { … }.,
    - The let and const declared within the block will be considered as a separate block scope, whereas var declared within the block will remain as a global scope.

5. What is **shadowing** ?

    ```
    var a = 100;
    {
            var a = 10;
            console.log(a);
    }
    console.log(a);
    ```

    - If we declare a variable var in a global scope and if we try to declare the same variable within a block scope, then the value of this variable in local scope will be considered.

6. **Illegal shadowing** ?

7. What is **event bubbling** and **event capturing** or **event delegation** ?

    Technical explanation : Bubbling propagates from the innermost to outermost element, whereas capturing propagates from outermost to innermost element.

    Logical understanding:
    Imagine you have a child element, a parent element and a grandparent element. Now you add an event listener to all these elements.
    When you click on a child element, along with the child element the ancestors elements will also be executed when attribute capture is false.
    This concept is called bubbling.

    Event capturing is the vice versa of bubbling with attribute capture as true.

8. What is **import** and **export** in js ?

    - In order to maintain reusable code.

      Export :
    - For example if we have to create a separate component for navbar, we have to write this code as a function enclosed with backticks.
    - Now to export one function, default is used to export one function.

export default navbar

- To export more than one function,

  export { function1, function2 }

Import:

- The import must happen in a script type module.

  <script type="module"> …. </script>
- To import this navbar in html file:

  Import navbar from "./components/navbar.js"

  Import { function1, function2 } from "./components/navbar.js"
- To use this in your html file, mention where you have to append this element and invoke the function exported from the component.

  document.body.innerHTML = navbar();

9. What is a **higher order function** ?

- A function takes another function as an argument or returns a function is called as higher order functions
- A function passed as an argument is a callback function.
- Some important HOF's are map, filter, reduce and foreach.

Map:

- Map transforms an array and returns another array.
- For eg., if we want to convert all the elements in an array to its square, then we can fo for map.
- Map syntax is .map with a function as an argument.

Filter:

- Filter is used to filter a specific value from an array.
- For eg., if we want to filter even numbers alone from the given array, we can go with .filter

Reduce:

- Reduce is used when we have to compound the given array into a single value.
- For eg., if we want to take the sum of all the values from an array, then we can use reduce.
- Reduce takes two parameters, accumulator and current.
- Current points to a specific element in the array and accumulates the set of operations as it proceeds forward.

Foreach:

- Foreach is similar to map but it doesn't return an array, it returns undefined.

- Foreach can be used when we want to access all the values for various purposes.
- For eg., in an E-commerce page, when we fetch the data from the database and show all the details of every product in such a case we have to use foreach.

10. What is a **promise**?

- Promise is an asynchronous operation that returns a resolved value or a value which is not resolved.
- There are 3 states in a promise - Fulfilled, pending and rejected.
- Fulfilled is when a promise is successful, pending is when a promise is yet to achieve or fail and rejection is when promise is not resolved.
- Promise works in such a way that its syntax is new Promise and it takes 2 parameters - resolve and reject.
- When a promise is successful it goes into resolve and not successful it enters reject.
- Promise provides 2 methods - .then and .catch.
- If a promise is resolved, .then prints the output. If a promise is not successful then we can catch the error using .catch.
- Promises are pushed into the Micro task queue. Whereas setTime out and setInterval are pushed into the callback queue.
- Promise eg:

```
let a = 10;
const isEven = new Promise((res, rej) => {
        if(a % 2 == 0) res("Success");
        else rej("Failure");
});

isEven.then((res) => console.log(res))
.catch((err) => console.log(err));
```

11. What is a **constructor**?

- Constructor function is a function to create or construct a pattern of an object.
- Constructor function has to be in pascal notation, which is the first character should be in caps.
- We use 2 important key words in a constructor function - this and new.
- this keywords points it to an empty object and new is like return - to return a newly created object.
- Eg for constructor function:

```
function Circle(radius) {
        this.radius = radius,
}

const circle1 = new Circle(5);
```

12. What is **this** ?

- This is a keyword which points to that object.
- If we are creating an object with first name and last name and if we have a function in this object to print this object name, here we will be using this. Keyword to point at the first name of that particular object.
- In the constructor function, this points to an empty object or towards all the new objects we create having constructor as an object pattern.

13. What is **call, apply and bind** ?

- Call, apply and bind is a concept of function borrowing.
- If there is a function within an object and if we want to access this function in another object, here using call we can access this function.
- Also, if we have a function separately and an object, and if we want to use this function binded with our object, call, apply and bind comes into the picture.

Call:

- The syntax follows with the function name followed by the method and the object where we have to access this function is passed as an argument.
- Eg:

  name.getFullName.call(name1); //When we access a function from an object.
  getFullName.call(name1); //When we access a function which is not in a function into any object.
- If we want to pass other arguments then we can pass it from the second argument. First parameter points to the object to use this function whereas from the second parameter we can pass any arguments to be used in that function.

Apply:

- Apply is similar to call but the arguments will be passed as an array.
- Eg: getFullName.call(name1, [“Hi”, “Bye”]);

Bind:

- Bind is different from call and apply, whereas call and apply were making use of that pointer function but bind makes a copy of this function.
- As bind makes a copy of that function, this can be used later by invoking this function.
- Eg:

  let later = getFullName.call(name1);

later();

14. What is a **pure function** ?

- A function is called as a pure function when it takes the same argument and returns the same pattern of result. For eg. Math.max(), or to add 2 numbers.
- It shouldn't have any side effects, like mutating data or sending a network request.

15. Some **ES6 features** ?

- let and const - Block scope variables.
- Arrow function.
- Object and Array destructuring.
- Spread Operator - Separates the array values and combines with another array.
  Eg: [...arr1, …arr2]

16. What is a **Polyfill** ?
- Polyfill is a functionality where we write a piece of code to standardize our function to run in old and current browsers.
- That is, some browsers might not support the latest functions of javascript. For eg., arrow function and promises does not support on Internet explorer.
- So here we optimize our code in order to run in all browsers.

17. What is **currying** ?
- It is a technique to transform a function with multiple arguments into different functions taking separate arguments.
- For eg. If we want to take a sum of arguments a and b, here we will be creating a function which takes an argument 'a' and returns a function which takes the argument 'b' and returns the sum of a its lexical scope and b.

```
const addCurry =(a)  => {
    return (b)=>{
        return (c)=>{
            return a+b+c
        }
    }
}
console.log(addCurry(2)(3)(5))  // 10
```

18. What is the **async and await** function ?
- Async is a keyword used in a function to make it an asynchronous function.

- Just like promises we can use try and catch blocks inside an async function to handle the operation. Asynchronous functions will also be used in fetch network requests.
- Await is a keyword which goes along with async functions which waits for the asynchronous operation to perform and store the data.

19. What is an **arrow function** and a **regular function** ?

- Regular function is a block of code which can be declared with a keyword function, function name followed by curly braces.
- Whereas arrow function is an ES6 feature which takes the syntax of parentheses pointing with an arrow at the curly braces.
- Both function types work in the same way yet there are few limitations.
- Arrow function doesn't have its own this keyword which will throw undefined.
- Because of this limitation, the arrow function also cannot be used to create constructor functions.
- In a regular function we can use argument objects which will return our arguments. But the same is not allowed in arrow function.

```
function func1(a, b, c) {
  console.log(arguments[0]);
  // expected output: 1

  console.log(arguments[1]);
  // expected output: 2

  console.log(arguments[2]);
  // expected output: 3
}

func1(1, 2, 3);
```

20. **Rest Operator** ?

- Rest operator is an ES6 feature used to pass n number of input as a parameter in a function.
- It is used as '…input'. So now if we pass a, b, c, d as an argument to this function, then …input will form this argument as an array to use it in this function.

21. What is a **prototype** ?

- Prototype is a blueprint of an object. In Javascript using prototype we can add methods and properties to a constructor. By creating an object we can inherit these properties and methods from a prototype.
- An example to create our own method using prototype:

```
//push function
Array.prototype.myPush = function(val) {
    this[this.length] = val;
};
```

22. What is **proto** ?
  - Proto itself is an object provided by javascript with the data types corresponding methods and properties.
  - For eg. When we create an array, we can see the properties of the array we have created along with a proto which the array has inherited from.

23. What is a **Prototype chain** ?
  - When we create an array this array inherits all the array properties and methods from proto and this array property inherits prototype of object.
    This chaining process between prototypes is called prototype chain.

24. What is **shallow cloning** and **deep cloning**?
  - Shallow cloning (Pass by reference) - When we update a cloned object then the original object will also be updated as they both are pointing towards the same object reference.
  - Deep cloning (Pass by Value) - When we update the cloned object the original object will not be updated as they both are pointing towards different object references.
  - In the spread operator, deep cloning will happen.

25. What is **pass by value** and **pass by reference** ?
  - All primitive data types - number, string, boolean, null, undefined are passed by value.
  - Updating the cloned object will not affect the original object.

```
//for example
let a = 10;
let b = a;
b++;

console.log(a); //output - 10
console.log(b); //output - 11
```

  - All non primitive data types - arrays and objects are passed by reference.
  - Updating the cloned object will also affect the original object.

```
//Pass by reference - for example
let arr1 = [1, 2, 3];
let arr2 = arr1;
arr2.push(5);

```

```
console.log(arr1); //1, 2, 3, 5
console.log(arr2); //1, 2, 3, 5
```

26. What is **JSON** ?
    - JSON is JavaScript Object Notation.
    - JSON is a format used to store data and transfer from server to web.

27. Difference between **debounce** and **throttling**?
    - Debounce - Is delayed as per the event emitter.
      I.e., every time when we enter something, the delay will be performed after this click.

28. Throttling - Is a function that is called for a set of intervals regardless of the event that happened.

29. What are classes in Javascript?
    - Classes are templates to create objects and methods in javascript.

```
class Users {
    constructor(name, age) {
        this.name = name;
        this.age = age;
    }

    details() {
        console.log(this.name, this.age);
    }
}

let user1 = new Users("Taher", 26);
console.log(user1);
user1.details();
```

30. What is DOM?
    - DOM represents the HTML document as a tree structure.
    - Using DOM we can read and access the content of the document.
    - Some of the DOM methods we use are .getElementById, .querySelector, .setAttribute.

31. What is ‘use strict’ in Javascript ?
    - Use strict in Javascript is used to strictly follow the latest ECMAscript code.

- For eg. We can declare a variable without using any let, var or const. - name = “Taher”. This is accepted in older versions and also accepted now if we don't use ‘use strict’ at top of our script. But if we use ‘use script’ at the top of our script, this statement declaration will not be valid.
- Use strict has to be mentioned at the top of our Javascript file. Else wrap it inside a function.

32. What are Browser specific functions ?
    - Alert - alert(“You will have a good day!”);
      Alerts the user with a message.
    - Prompt - prompt(“How was your day?”, “Good”);
      Prompts with an input field to enter. We can get this value and use it.
    - Confirm - confirm(“How was your day?”);
      By clicking ok, we get a boolean value true and cancel we get a boolean value false.

33. What is a callback function ?
    - A function passed into another function as an argument and invoked within this outer function.
    - Eg. A function passed as an argument within a higher order function.

34. What is an asynchronous function?
    - Javascript is a single threaded language, meaning it runs line by line from top to bottom. Async functions take time to complete, for eg. setTimeout or promises.
    - So now, in order to avoid blocking of our code, these async functions will be handled parallelly by web API and once the synchronous code is executed, event loop will monitor and execute these async functions.

35. What is fetch ?
    - Fetch is promise based used to fetch a network request.
    - The response can be handled using .then .catch or async await methods.
    - .then .catch -

      ```
      fetch(“”).then(res => res.json()).then(data => console.log(data));
      ```
    - Async await -

      ```
      async function getData() {
              let res = await fetch(“”);
              let data = await res.json();
      }
      ```

All Fetch methods:

```
//get request
function getData() {
    fetch("/")
    .then(res => res.json())
```

```
    .then(data => console.log(data))
    .catch(err => console.log(err));
}

//post request
function postData() {
    fetch("/", {
        method: "POST",
        headers: {
            "Content-Type": "application/json"
        },
        body: JSON.stringify({
            name: "Taher"
        })
    })
    .then(res => res.json())
    .then(data => console.log(data))
    .catch(err => console.log(err));
}

//update request
function updateData() {
    fetch("/", {
        method: "PATCH",
        headers: {
            "Content-Type": "application/json"
        },
        body: JSON.stringify({
            name: "Ahmed"
        })
    })
    .then(res => res.json())
    .then(data => console.log(data))
    .catch(err => console.log(err));
}

//Delete request
function updateData() {
    fetch("/", {
        method: "DELETE",
    })
    .then(res => res.json())
    .then(data => console.log(data))
    .catch(err => console.log(err));
}
```

Fundamentals of Javascript:

Operators

1. Rules of AND Operator (&&) -
    - If both the values are truthy, then prints the last truth value.
    - If there is any falsy value, it finds and prints the first falsy value.

2. Rules of OR Operator (||) -
    - Finds for the first truthy value and prints.
    - If both the values are falsy, then prints the last falsy value.

3. Nullish Coalescing Operator (??) -
    - Returns the first defined value. I.e., will ignore the null/undefined value.

General

4. Alert returns undefined.

5. When a value is passed as a function parameter, it's called an argument.

Loop

6. Difference between while loop and do while loop -
    - While loop first checks the condition then executes the value.
    - Do while loop first executes the value then checks the condition.

7. continue/break is not allowed in ternary operators.