JAVASCRIPT DAY 18

FAQ'S OF DAY 18

1. What are functions in JavaScript?

A function in JavaScript is a block of code designed to perform a particular task. Functions allow you to
reuse code, which can help keep programs clean and manageable. Functions can be defined using the function
keyword or through arrow function syntax.

2. How do you define a function in JavaScript?

Functions can be defined in two main ways:

Function Declaration:

```
function myFunction() {
console.log("Hello World");
}
```

Function Expression:

```
const myFunction = function() {
console.log("Hello World");
};
```

3. What is the difference between function declarations and function expressions?

Function Declarations are hoisted to the top of their scope, meaning they can be called before the function is
defined.

```
myFunction(); // Works fine
function myFunction() {
console.log("Hello");
}
```

Function Expressions are not hoisted, so they can only be called after they are defined.

```
myFunction(); // Error: Cannot call a function before it's defined
const myFunction = function() {
console.log("Hello");
};
```

4. What are arrow functions?

Arrow functions are a more concise way to write functions in JavaScript. They were introduced in ECMAScript 6 (ES6). They are syntactically shorter and do not have their own this context, meaning they inherit
this from the surrounding scope.

Example:

```
const add = (a, b) => a + b;
```

5. What are higher-order functions in JavaScript?

Higher-order functions are functions that can take other functions as arguments or return functions as
their result. They allow you to create more abstract and reusable code.

Example:

```
function applyOperation(a, b, operation) {
return operation(a, b);
}
const sum = (x, y) => x + y;
console.log(applyOperation(5, 3, sum)); // Outputs 8
```

6. What are anonymous functions?

Anonymous functions are functions that do not have a name. They are often used in situations where
a function is needed temporarily or passed as an argument.

Example:

```
const myFunction = function() {
console.log("Anonymous Function");
};
```

7. What is a callback function?

Answer: A callback function is a function that is passed into another function as an argument and is executed
after the completion of that function. They are commonly used in asynchronous programming.

Example:

```
function fetchData(callback) {
setTimeout(() => {
console.log("Data fetched");
callback();
}, 1000);
}
fetchData(() => {
console.log("Callback executed");
});
```

8. What are default parameters in JavaScript?

Default parameters allow you to set default values for function parameters in case they are not provided during function invocation.

Example:

```
function greet(name = "Guest") {
console.log("Hello, " + name);
}

greet(); // Outputs: "Hello, Guest"
greet("Alice"); // Outputs: "Hello, Alice"
```

9. What is the difference between call(), apply(), and bind() methods in JavaScript functions?

call(): Invokes the function with a specified this value and arguments provided individually.

```
function greet(name) {
console.log("Hello " + name);
}
greet.call(this, "Alice"); // Outputs: "Hello Alice"
```

apply(): Similar to call(), but the arguments are passed as an array.

```
greet.apply(this, ["Alice"]); // Outputs: "Hello Alice"
```

bind(): Returns a new function with a specific this value and arguments, but does not immediately invoke it.

```
const boundGreet = greet.bind(this, "Alice");
boundGreet(); // Outputs: "Hello Alice"
```

10. What are IIFE (Immediately Invoked Function Expressions) in JavaScript?

An IIFE is a function that is defined and invoked immediately after its creation. It is commonly used to create a new scope to avoid polluting the global scope.

Example:

```
(function() {
console.log("This is an IIFE");
})();
```

11. What are closures in JavaScript?

A closure is a function that retains access to its lexical scope, even after the outer function has finished

execution. This allows for private variables and functions.

Example:

```
function outer() {
let counter = 0;
return function inner() {
```

```
counter++;
console.log(counter);
};
}
const increment = outer();
increment(); // Outputs: 1
increment(); // Outputs: 2
```

12. What is a function declaration in JavaScript?

A function declaration, also known as a named function, is a way to define a function in JavaScript. It has a specific name, and it is hoisted, meaning the function can be called before its definition in the code.

Key Points:

• Function declarations are hoisted.

• Can be called before being defined in the code.

```
function greet(name) {
console.log("Hello " + name);
}
greet("John"); // Output: Hello John
```

13. What is a function expression in JavaScript?

A: A function expression involves creating a function and assigning it to a variable. This function is often
anonymous, meaning it doesn't have a name. Unlike function declarations, function expressions are not
hoisted.

Key Points:

• Not hoisted.

• Can be anonymous.

• Assigned to a variable.

```
const greet = function(name) {
console.log("Hello " + name);
};
greet("Jane"); // Output: Hello Jane
```

14. What are arrow functions in JavaScript, and how are they different from regular functions?

Arrow functions are a concise way of writing functions in JavaScript. They are introduced in ES6 and differ
from regular functions in the following ways:

• They don't have their own this context, instead, they inherit this from the enclosing scope.
• They are shorter and don't require the function keyword.

Key Points:

• Concise syntax.
• Inherits this from the surrounding scope.
• Cannot be used as constructor functions.

```
const greet = (name) => {
console.log("Hello " + name);
};
greet("Alice"); // Output: Hello Alice
```

15. What is a constructor function in JavaScript?

A constructor function is used to create multiple instances of an object with similar properties and methods.
The function is invoked with the new keyword, which sets up a new object and binds this to it.

Key Points:

• Used for creating objects.
• Invoked with the new keyword.
• this refers to the newly created object.

```
function Person(name, age) {
this.name = name;
this.age = age;
}
const person1 = new Person("John", 30);
console.log(person1.name); // Output: John
```

16. What are generator functions in JavaScript?

Generator functions are special functions that allow you to pause and resume their execution. They are defined using the function
* syntax and yield multiple values over time using the yield keyword.

Key Points:

• Defined with function*.

• Use yield to return multiple values lazily.

• Can be paused and resumed.

```
function* greet() {
yield "Hello";
yield "Hi";
}
const greetGen = greet();
console.log(greetGen.next().value); // Output: Hello
console.log(greetGen.next().value); // Output: Hi
```

17. What is an IIFE in JavaScript?

An IIFE (Immediately Invoked Function Expression) is a function that is defined and executed immediately
after its creation. It helps avoid polluting the global scope.

Key Points:

• Invoked immediately after being defined.

• Often used to create a local scope.

```
(function() {
console.log("This is an IIFE");
}) (); // Output: This is an IIFE
```

18. What are callback functions in JavaScript?

A callback function is a function that is passed as an argument to another function and is executed after the completion of the outer function. It is commonly used in asynchronous programming (e.g., with setTimeout, event listeners, or AJAX).

```
function greet(name, callback) {
console.log("Hello " + name);
callback();
}
function farewell() {
console.log("Goodbye");
}
greet("John", farewell);
// Output:
// Hello John
// Goodbye
```

Key Points:

- Passed as an argument to another function.
- Executed after the outer function finishes.
- Commonly used in asynchronous operations.

MCQ'S OF DAY 18

1. Function Declaration

Q1. What is a function declaration in JavaScript?

Answer: B) A function defined using the function keyword

Q2: How can you call a function declaration before its definition?

Answer: C) Due to hoisting

Q3: What is the syntax for a named function declaration?

Answer: C) function myFunc() {}

2. Function Expression

Q4: What is a function expression?

Answer: A) A function stored in a variable

Q5: Which of the following correctly defines a function expression?

Answer: A) const sum = function(a, b) { return a + b; }

Q6: Can function expressions be anonymous?

Answer: B) Yes

3. Arrow Functions

Q7: How is an arrow function different from a regular function?

Answer: D) Both A and B

Q8: How do you define an arrow function?

Answer: B) const greet = (name) => {}

Q9: Can arrow functions be used as constructors?

Answer: B) No

4. Constructor Functions

Q10: What is a constructor function in JavaScript?

Answer: B) A function used to create new objects

Q11: How do you define a constructor function?

Answer: A) function Person(name) { this.name = name; }

Q12: How do you create an instance of a constructor function?

Answer: B) new Person()

5. Generator Functions

Q13: What keyword is used to define a generator function?

Answer: A) function*

Q14: What does the yield keyword do in a generator function?

Answer: D) Both B and C

Q15: How do you call a generator function?

Answer: B) Using .next()

6. IIFE (Immediately Invoked Function Expression)

Q16: What is the purpose of an IIFE?

Answer: B) To execute a function immediately

Q17: What is the correct syntax for an IIFE?

Answer: D) Both B and C

Q18: Why use an IIFE?

Answer: A) To avoid polluting the global scope

7. Callback Functions

Q19: What is a callback function?

Answer: B) A function passed as an argument to another function

Q20: Which of the following is an example of a callback function?

Answer: A) setTimeout(function() { console.log('Hello') }, 1000)

Q21: Why use callback functions?

Answer: B) To handle asynchronous operations

8. Higher-Order Functions

Q22: What is a higher-order function in JavaScript?

Answer: D) Both A and B

Q23: Which of the following is an example of a higher-order function?

Answer: A) map()

9. Function Parameters

Q24: What happens if a function is called with fewer arguments than defined parameters?

Answer: C) The missing arguments will be undefined

Q25: How can you set default values for parameters in a function?

Answer: C) Using default parameter syntax

10. Rest Parameters

Q26: What does the rest parameter (...) do in a function?

Answer: A) Collects all arguments into a single array

Q27: Which of the following is a correct use of the rest parameter?

Answer: D) All of the above

11. Function Scope

Q28: What is the scope of a variable declared inside a function?

Answer: B) Local scope of the function

Q29: What will happen if you try to access a variable declared with let outside its block?

Answer: A) It will throw a ReferenceError

12. Closures

Q30: What is a closure in JavaScript?

Answer: B) A function that returns another function and maintains its environment

Q31: Which of the following is an example of a closure?

Answer: A) A function returning another function that accesses its outer variable

13. Function Binding

Q32: What is the bind() method used for in JavaScript?

Answer: A) To bind a function to a specific object and set its this context

Q33: What will the following code output?

```
const obj = { x: 10 };
const func = function() { console.log(this.x); };
const boundFunc = func.bind(obj);
boundFunc();
```

Answer: B) 10

14. Anonymous Functions

Q34: What is an anonymous function in JavaScript?

Answer: A) A function without a name

Q35: Which of the following is an example of an anonymous function?

Answer: D) Both A and C

15. Function Arguments

Q36: Which property of the function object contains the arguments passed to the function?

Answer: A) arguments

Q37: What is the result of calling add(1, 2, 3) if add is defined as:

```
function add(a, b) { return a + b; }
```

Answer: A) 3

16. Function Call Method

Q38: What does the call() method do in JavaScript?

Answer: A) Calls a function immediately and sets its this context

Q39: How do you call a function with call() passing arguments to it?

Answer: B) func.call(this, ...args)

CODING QUESTIONS

1. What is a function?

A function is a block of reusable code that performs a specific task. It is executed when it is called or invoked.

2. Set of statements called as?

A **function** is a set of statements grouped together to perform a task.

3. Are functions used to reuse business logic?

Yes, functions help in reusing business logic and avoid repetition of code.

4. What are the types of functions?

1. **Named Functions**
2. **Anonymous Functions**
3. **Arrow Functions**
4. **Constructor Functions**
5. **IIFE (Immediately Invoked Function Expressions)**

5. Write the syntax for function declaration.

```
function functionName(parameters) {
   // code block
}
```

6. Write the syntax for function expression.

```
const functionName = function(parameters) {
   // code block
};
```

7. Write the syntax for arrow functions.

```
const functionName = (parameters) => {
   // code block
};
```

8. Can I store function expression and arrow functions to variables?

Yes, both can be stored in variables.

9. How to represent arrow functions?

Arrow functions are represented using the => syntax.

Example:

```
const greet = (name) => `Hello, ${name}`;
```

10. Arrow functions are introduced in which version?

Arrow functions were introduced in **ES6**.

11. How to read data and write from <input> tag?

To read:

```
const value = document.getElementById("inputId").value;
```

To write:

```
document.getElementById("inputId").value = "New Value";
```

12. How to get the reference of <input type="text" id="username">?

```
const usernameRef = document.getElementById("username");
```

13. How to set the content to an HTML element?

```
document.getElementById("elementId").innerHTML = "New Content";
```

14. What are the differences between innerHTML and innerText in JavaScript?

innerHTML	**innerText**
Returns/sets the HTML markup.	Returns/sets only text.
Parses HTML tags.	Ignores HTML tags.

15. Display full name in Input3 after entering first and last name.

```
function displayFullName() {
   const firstName = document.getElementById("input1").value;
   const lastName = document.getElementById("input2").value;
   document.getElementById("input3").value = `${firstName} ${lastName}`;
}
```

16. Display full name in <h1> element.

```
function displayFullName() {
   const firstName = document.getElementById("input1").value;
   const lastName = document.getElementById("input2").value;
   document.getElementById("result").innerHTML = `${firstName} ${lastName}`;
}
```

17. Display operation result in an output area.

```
function calculate(operation) {
   const a = parseFloat(document.getElementById("a").value);
   const b = parseFloat(document.getElementById("b").value);
   let result;
   if (operation === "add") result = a + b;
   else if (operation === "subtract") result = a - b;
   else if (operation === "multiply") result = a * b;
   else if (operation === "divide") result = a / b;
   document.getElementById("output").value = result;
}
```

18. Swap two numbers and display in input controls.

```
function swapNumbers() {
    const a = document.getElementById("a").value;
    const b = document.getElementById("b").value;
    document.getElementById("a").value = b;
    document.getElementById("b").value = a;
}
```

19. Calculate monthly salary.

```
function calculateSalary() {
    const annualSalary = parseFloat(document.getElementById("salary").value);
    document.getElementById("monthlySalary").value = (annualSalary / 12).toFixed(2);
}
```

20. Display product of two values.

```
function calculateProduct() {
    const p = parseFloat(document.getElementById("p").value);
    const q = parseFloat(document.getElementById("q").value);
    document.getElementById("result").value = p * q;
}
```

21. Calculate bill amount.

```
function calculateBill() {
    const cost = parseFloat(document.getElementById("cost").value);
    const quantity = parseFloat(document.getElementById("quantity").value);
    document.getElementById("bill").value = cost * quantity;
}
```

22. Calculate total and average of marks.

```
function calculateMarks(action) {
    const m1 = parseFloat(document.getElementById("m1").value);
    const m2 = parseFloat(document.getElementById("m2").value);
    const m3 = parseFloat(document.getElementById("m3").value);
    const total = m1 + m2 + m3;

    if (action === "total") {
        document.getElementById("total").value = total;
    } else if (action === "average") {
        document.getElementById("average").value = (total / 3).toFixed(2);
    }
}
```

23. Calculate interest and final amount.

```
function calculateLoan(action) {
    const principal = parseFloat(document.getElementById("principal").value);
    const rate = parseFloat(document.getElementById("rate").value);
    const time = parseFloat(document.getElementById("time").value);
    const interest = (principal * rate * time) / 100;

    if (action === "interest") {
        document.getElementById("interest").value = interest.toFixed(2);
    } else if (action === "finalAmount") {
        document.getElementById("finalAmount").value = (principal + interest).toFixed(2);
    }
}
```

24. Calculate GST and final cost.

```
function calculateGST(action) {
    const cost = parseFloat(document.getElementById("cost").value);
    const gstRate = parseFloat(document.getElementById("gstRate").value);
    const gstAmount = (cost * gstRate) / 100;

    if (action === "gstAmount") {
        document.getElementById("gstAmount").value = gstAmount.toFixed(2);
    } else if (action === "finalCost") {
        document.getElementById("finalCost").value = (cost + gstAmount).toFixed(2);
    }
}
```

25. Function examples

```
function func_one(param1, param2, param3) {
    document.write(`${param1} ${param2} ${param3}<br>`);
}
func_one("ReactJS", "NodeJS", "MongoDB"); // ReactJS NodeJS MongoDB
func_one(100, 200, 300, 400); // 100 200 300
func_one(); // undefined undefined undefined
func_one(undefined, "Hello"); // undefined Hello undefined
func_one(null, null, null); // null null null
```

26. What is the output of the following function examples?

Example 1:

```
function func_one(param1, param2, param3) {
   document.write(`${param1} ${param2} ${param3}<br>`);
}


func_one("ReactJS", "NodeJS", "MongoDB");
```

Output:

```
ReactJS NodeJS MongoDB
```

Example 2:

```
func_one(100, 200, 300, 400);
```

Output:

```
100                    200                    300
```

(Extra arguments are ignored.)

Example 3:

```
func_one();
```

Output:

```
undefined              undefined              undefined
```

(No arguments are passed, so undefined is used.)

Example 4:

```
func_one(undefined, "Hello");
```

Output:

```
undefined              Hello                  undefined
```

(undefined is explicitly passed for param1 and param3.)

Example 5:

```
func_one(null, null, null);
```

Output:

```
null                   null                   null
```

(null is explicitly passed for all parameters.)

27. What are default parameters in functions?

Default parameters allow you to initialize parameters with default values if no value or undefined is passed during a function call.

Example:

```
function greet(name = "Guest") {
   return `Hello, ${name}!`;
}
```

```
greet();           // Output: Hello, Guest!
greet("Alice");   // Output: Hello, Alice!
```

28. How to return a value from a function?

Use the return statement.

Example:

```
function add(a, b) {
   return a + b;
}

const sum = add(5, 10); // sum = 15
```

29. Write a function to calculate the square of a number.

```
function square(num) {
   return num * num;
}

const result = square(5); // result = 25
```

30. What is recursion?

Recursion is a process where a function calls itself to solve a smaller instance of the same problem.

Example: Factorial using Recursion

```
function factorial(n) {
   if (n === 0) return 1;
   return n * factorial(n - 1);
}

const result = factorial(5); // result = 120
```

31. What is an anonymous function?

An anonymous function is a function without a name, often used as a value.

Example:

```
const greet = function(name) {
   return `Hello, ${name}`;
};
```

32. What is an IIFE (Immediately Invoked Function Expression)?

An IIFE is a function that is executed immediately after it is defined.

Example:

```
(function () {
    console.log("This is an IIFE");
})();
```

33. What is a higher-order function?

A higher-order function is a function that takes another function as an argument or returns a function.

Example:

```
function greet(name) {
    return `Hello, ${name}`;
}

function execute(func, arg) {
    return func(arg);
}

console.log(execute(greet, "Alice")); // Output: Hello, Alice
```

34. What is a callback function?

A callback function is a function passed as an argument to another function.

Example:

```
function processUserInput(callback) {
    const name = "Alice";
    callback(name);
}

processUserInput((name) => {
    console.log(`Hello, ${name}`);
}); // Output: Hello, Alice
```

35. What is the difference between var, let, and const?

Feature	**var**	**let**	**const**
Scope	Function scope	Block scope	Block scope
Redeclaration	Allowed	Not allowed	Not allowed
Reassignment	Allowed	Allowed	Not allowed
Hoisting	Hoisted with undefined	Hoisted but not usable	Hoisted but not usable

36. How to create an object in JavaScript?

```
const person = {
   firstName: "John",
   lastName: "Doe",
   age: 25,
   greet() {
      return `Hello, ${this.firstName} ${this.lastName}`;
   }
};
```

37. What is a constructor function?

A constructor function is used to create objects with similar properties and methods.

Example:

```
function Person(firstName, lastName) {
   this.firstName = firstName;
   this.lastName = lastName;
}

const john = new Person("John", "Doe");
```

38. What is the this keyword?

The this keyword refers to the object it belongs to.

Example:

```
const person = {
   name: "Alice",
   greet() {
      return `Hello, ${this.name}`;
   }
};
```

39. What is the difference between function declaration and function expression?

Feature	Function Declaration	Function Expression
Syntax	function greet() {}	const greet = function() {}
Hoisting	Hoisted	Not hoisted

40. How to access object properties in JavaScript?

1. Dot notation: object.property
2. Bracket notation: object["property"]

Example:

```
const person = { name: "Alice", age: 25 };
```

```
console.log(person.name);       // Dot notation
console.log(person["age"]);     // Bracket notation
```

41. What is the difference between == and ===?

Comparison	**==**	**===**
Type conversion	Converts types before comparison	Does not convert types
Example	5 == "5" → true	5 === "5" → false

42. What is a closure in JavaScript?

A closure is a function that remembers its outer scope even after the outer function has executed.

Example:

```
function outer() {
   const message = "Hello";

   return function inner() {
      console.log(message);
   };
}

const greet = outer();
greet(); // Output: Hello
```

43. What is the difference between call(), apply(), and bind()?

Method	**Description**
call()	Calls a function with arguments provided individually.
apply()	Calls a function with arguments provided as an array.
bind()	Returns a new function with a specified this value.

44. What is the difference between synchronous and asynchronous code?

Feature	**Synchronous**	**Asynchronous**
Execution	Runs line-by-line.	Can run tasks concurrently.
Example	Blocking API calls.	Promises, setTimeout().

45. What are promises in JavaScript?

Promises are used to handle asynchronous operations. They can be in three states:

1. Pending
2. Fulfilled
3. Rejected

Example:

```
const promise = new Promise((resolve, reject) => {
  if (true) resolve("Success");
  else reject("Failure");
});
```

46. Write a basic example of async/await.

```
async function fetchData() {
  const response = await fetch("https://api.example.com/data");
  const data = await response.json();
  console.log(data);
}
```

<u>CHEAT SHEET FOR INTERVIEW</u>

1. Function Declaration (Named Functions)

A function declaration is explicitly named and can be hoisted, meaning it can be called before its definition.

Syntax:

```
function functionName() {
  // function body
}
```

Example:

```
function greet() {
  console.log("Hello, World!");
}
greet(); // Output: Hello, World!
```

2. Function Expression (Anonymous Functions)

A function expression assigns an anonymous function to a variable. These are not hoisted, so they must be defined before being called.

Syntax:

```
const functionName = function() {
  // function body
};
```

Example:

```
const greet = function() {
  console.log("Hello, World!");
};
greet(); // Output: Hello, World!
```

3. Arrow Functions

Arrow functions provide a shorter syntax for defining functions. They don't have their own this context; instead, they inherit this from their enclosing scope.

Syntax:

```
const functionName = (param1, param2) => {
   // function body
};
```

Example:

```
const greet = () => {
   console.log("Hello, World!");
};
greet(); // Output: Hello, World!
```

4. Constructor Functions

Constructor functions are used to create and initialize objects. They are called with the new keyword.

Syntax:

```
function ConstructorFunction(param1, param2) {
   this.param1 = param1;
   this.param2 = param2;
}
```

Example:

```
function Person(name, age) {
   this.name = name;
   this.age = age;
}
const person1 = new Person("Alice", 25);
console.log(person1.name); // Output: Alice
console.log(person1.age);  // Output: 25
```

5. Generator Functions

Generator functions are special functions that can pause execution using yield and resume later. These are useful for creating iterators.

Syntax:

```
function* generatorFunction() {
   yield value;
}
```

Example:

```
function* numbers() {
   yield 1;
   yield 2;
   yield 3;
}
const numGen = numbers();
console.log(numGen.next().value); // Output: 1
console.log(numGen.next().value); // Output: 2
console.log(numGen.next().value); // Output: 3
```

6. IIFE (Immediately Invoked Function Expression)

An IIFE runs immediately after being defined, often used to create a private scope.

Syntax:

```
(function() {
   // function body
})();
```

Example:

```
(function() {
   console.log("This is an IIFE!");
})(); // Output: This is an IIFE!
```

7. Callback Functions

A callback function is passed as an argument to another function and executed after the outer function completes. Commonly used in asynchronous programming.

Syntax:

```
function mainFunction(callback) {
   callback();
}
```

Example:

```
function greet() {
   console.log("Hello, Callback!");
}

function executeCallback(callback) {
   callback();
}
executeCallback(greet); // Output: Hello, Callback!
```