Shailesh Diwanji

JavaScript Interview Questions

Part 1: Theory Questions (75)

1. What are the different data types in JavaScript?

JavaScript has 8 data types: 7 primitive types (String, Number, BigInt, Boolean, Undefined, Null, Symbol) and 1 reference type (Object).

```
let str = "hello";
let num = 42;
let bigInt = 9007199254740991n;
let bool = true;
let undef = undefined;
let n = null;
let sym = Symbol("id");
let obj = { name: "John" };

console.log(typeof str);
console.log(typeof num);
console.log(typeof bigInt);
console.log(typeof bool);
console.log(typeof undef);
console.log(typeof n);
console.log(typeof sym);
console.log(typeof obj);
```

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

== performs type coercion before comparison, while === checks both value and type without coercion.

```
console.log(5 == "5");
console.log(5 === "5");
console.log(null == undefined);
console.log(null === undefined);
console.log(0 == false);
console.log(0 === false);
```

3. What is hoisting in JavaScript?

Hoisting is JavaScript's behavior of moving declarations to the top of their scope before code execution. Variables declared with var and function declarations are hoisted.

```
console.log(x);
var x = 5;

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

console.log(y);
let y = 10;
```

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

var is function-scoped and hoisted, let and const are block-scoped. const cannot be reassigned.

```
function example() {
  if (true) {
    var x = 1;
    let y = 2;
    const z = 3;
  }
  console.log(x);
  console.log(y);
}

const arr = [1, 2, 3];
arr.push(4);
console.log(arr);
arr = [5, 6, 7];
```

5. What is a closure?

A closure is a function that has access to variables in its outer (enclosing) lexical scope, even after the outer function has returned.

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

const increment = outer();
console.log(increment());
console.log(increment());
console.log(increment());
```

6. What is the this keyword?

this refers to the object that is executing the current function. Its value depends on how the function is called.

```
const obj = {
  name: "Alice",
  greet: function() {
    console.log(this.name);
  },
  arrowGreet: () => {
    console.log(this.name);
  }
};

obj.greet();
obj.arrowGreet();

const greetFunc = obj.greet;
greetFunc();
```

7. What are arrow functions and how do they differ from regular functions?

Arrow functions have shorter syntax and don't bind their own this, arguments, super, or new.target.

```
const regularFunc = function(a, b) {
  console.log(arguments);
  return a + b;
};

const arrowFunc = (a, b) => {
  console.log(arguments);
  return a + b;
};

regularFunc(1, 2);
arrowFunc(1, 2);

const obj = {
  value: 10,
  regular: function() { setTimeout(function() { console.log(this.value); }, 100);
  arrow: function() { setTimeout(() => { console.log(this.value); }, 100); }
};

obj.regular();
obj.arrow();
```

8. What is event bubbling and capturing?

Event bubbling is when an event propagates from the target element up through its ancestors. Capturing is the opposite - from root to target.

```
document.getElementById("parent").addEventListener("click", () => {
  console.log("Parent clicked - Bubbling");
}, false);

document.getElementById("child").addEventListener("click", () => {
  console.log("Child clicked");
}, false);

document.getElementById("parent").addEventListener("click", () => {
  console.log("Parent clicked - Capturing");
}, true);
```

9. What is the difference between null and undefined?

undefined means a variable has been declared but not assigned a value. null is an assignment value representing no value.

```
let a;
console.log(a);
console.log(typeof a);

let b = null;
console.log(b);
console.log(typeof b);

console.log(null == undefined);
console.log(null === undefined);
```

10. What is the prototype chain?

The prototype chain is the mechanism by which JavaScript objects inherit properties and methods from other objects.

```
function Person(name) {
  this.name = name;
}

Person.prototype.greet = function() {
  console.log("Hello, " + this.name);
};

const john = new Person("John");
john.greet();

console.log(john.__proto__ === Person.prototype);
console.log(Person.prototype.__proto__ === Object.prototype);
```

11. What is the event loop?

The event loop is a mechanism that handles asynchronous operations by continuously checking the call stack and callback queue.

```
console.log("Start");

setTimeout(() => {
  console.log("Timeout");
}, 0);

Promise.resolve().then(() => {
  console.log("Promise");
});

console.log("End");
```

12. What are promises?

Promises are objects representing the eventual completion or failure of an asynchronous operation.

```
const promise = new Promise((resolve, reject) => {
  setTimeout(() => {
    resolve("Success!");
  }, 1000);
});

promise
  .then(result => console.log(result))
  .catch(error => console.log(error))
  .finally(() => console.log("Done"));
```

13. What is async/await?

async/await is syntactic sugar for promises, making asynchronous code look synchronous.

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

fetchData().then(data => console.log(data));
```

14. What is destructuring?

Destructuring is a syntax for extracting values from arrays or properties from objects into distinct variables.

```
const arr = [1, 2, 3, 4, 5];
const [first, second, ...rest] = arr;
console.log(first, second, rest);

const person = { name: "Alice", age: 30, city: "NYC" };
const { name, age, country = "USA" } = person;
console.log(name, age, country);
```

15. What is the spread operator?

The spread operator (...) expands iterables into individual elements.

```
const arr1 = [1, 2, 3];
const arr2 = [4, 5, 6];
const combined = [...arr1, ...arr2];
console.log(combined);

const obj1 = { a: 1, b: 2 };
const obj2 = { c: 3, d: 4 };
const merged = { ...obj1, ...obj2 };
console.log(merged);

function sum(a, b, c) {
  return a + b + c;
}
console.log(sum(...arr1));
```

16. What is the rest parameter?

The rest parameter allows a function to accept an indefinite number of arguments as an array.

```
function sum(...numbers) {
  return numbers.reduce((total, num) => total + num, 0);
}

console.log(sum(1, 2, 3));
console.log(sum(1, 2, 3, 4, 5));

function greet(greeting, ...names) {
  return greeting + " " + names.join(", ");
}
console.log(greet("Hello", "Alice", "Bob", "Charlie"));
```

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

Function declarations are hoisted, while function expressions are not.

```
hello();
function hello() {
  console.log("Declaration");
}

greet();
const greet = function() {
  console.log("Expression");
};
```

18. What is an IIFE?

IIFE (Immediately Invoked Function Expression) is a function that runs as soon as it is defined.

```
(function() {
  console.log("IIFE executed");
})();

(function(name) {
  console.log("Hello, " + name);
})("Alice");

const result = (function() {
  return 42;
})();
console.log(result);
```

19. What is the call() method?

call() invokes a function with a specified this value and arguments provided individually.

```
function greet(greeting, punctuation) {
  console.log(greeting + ", " + this.name + punctuation);
}

const person = { name: "Alice" };
greet.call(person, "Hello", "!");
greet.call({ name: "Bob" }, "Hi", ".");
```

20. What is the apply() method?

apply() is similar to call() but accepts arguments as an array.

```
function greet(greeting, punctuation) {
  console.log(greeting + ", " + this.name + punctuation);
}

const person = { name: "Alice" };
greet.apply(person, ["Hello", "!"]);

const numbers = [5, 6, 2, 3, 7];
const max = Math.max.apply(null, numbers);
console.log(max);
```

21. What is the bind() method?

bind() creates a new function with a specified this value and optional arguments.

```
const person = {
  name: "Alice",
  greet: function(greeting) {
    console.log(greeting + ", " + this.name);
  }
};

const greetPerson = person.greet.bind(person);
setTimeout(greetPerson, 1000, "Hello");

function multiply(a, b) {
  return a * b;
}
const double = multiply.bind(null, 2);
console.log(double(5));
```

22. What is currying?

Currying is a technique of transforming a function with multiple arguments into a sequence of functions each taking a single argument.

```
function curry(a) {
  return function(b) {
    return function(c) {
      return a + b + c;
    };
  };
}

console.log(curry(1)(2)(3));

const curriedMultiply = a => b => a * b;
const double = curriedMultiply(2);
console.log(double(5));
```

23. What is memoization?

Memoization is an optimization technique that stores the results of expensive function calls and returns the cached result when the same inputs occur again.

```
function memoize(fn) {
  const cache = {};
  return function(...args) {
    const key = JSON.stringify(args);
    if (cache[key]) {
      console.log("From cache");
      return cache[key];
    }
    const result = fn(...args);
    cache[key] = result;
    return result;
  };
}

const factorial = memoize(n => {
  if (n <= 1) return 1;
  return n * factorial(n - 1);
});

console.log(factorial(5));
console.log(factorial(5));
```

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

Synchronous code executes line by line, blocking subsequent code. Asynchronous code doesn't block execution.

```
console.log("Start");

console.log("Synchronous");

setTimeout(() => {
  console.log("Asynchronous");
}, 1000);

console.log("End");
```

25. What are callbacks?

Callbacks are functions passed as arguments to other functions, to be executed after some operation completes.

```
function fetchData(callback) {
  setTimeout(() => {
    const data = { id: 1, name: "Alice" };
    callback(data);
  }, 1000);
}

fetchData(data => {
  console.log(data);
});
```

26. What is callback hell?

Callback hell refers to heavily nested callbacks that make code difficult to read and maintain.

```
getData(function(a) {
  getMoreData(a, function(b) {
    getMoreData(b, function(c) {
      getMoreData(c, function(d) {
        console.log(d);
      });
    });
  });
});
```

27. What are template literals?

Template literals are string literals that allow embedded expressions and multi-line strings.

```
const name = "Alice";
const age = 30;
const greeting = `Hello, my name is ${name} and I am ${age} years old.`;
console.log(greeting);

const multiline = `
  This is a
  multi-line
  string
`;
console.log(multiline);
```

28. What is the difference between map() and forEach()?

map() creates a new array with the results of calling a function on every element. forEach() executes a function on each element but doesn't return anything.

```
const numbers = [1, 2, 3, 4, 5];

const doubled = numbers.map(num => num * 2);
console.log(doubled);
console.log(numbers);

const result = numbers.forEach(num => num * 2);
console.log(result);
```

29. What is the filter() method?

filter() creates a new array with all elements that pass the test implemented by the provided function.

```
const numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];

const evenNumbers = numbers.filter(num => num % 2 === 0);
console.log(evenNumbers);

const greaterThanFive = numbers.filter(num => num > 5);
console.log(greaterThanFive);
```

30. What is the reduce() method?

reduce() executes a reducer function on each element, resulting in a single output value.

```
const numbers = [1, 2, 3, 4, 5];

const sum = numbers.reduce((acc, num) => acc + num, 0);
console.log(sum);

const product = numbers.reduce((acc, num) => acc * num, 1);
console.log(product);

const max = numbers.reduce((acc, num) => num > acc ? num : acc);
console.log(max);
```

31. What is the find() method?

find() returns the first element in an array that satisfies the provided testing function.

```
const users = [
  { id: 1, name: "Alice" },
  { id: 2, name: "Bob" },
  { id: 3, name: "Charlie" }
];

const user = users.find(u => u.id === 2);
console.log(user);

const notFound = users.find(u => u.id === 5);
console.log(notFound);
```

32. What is the some() method?

some() tests whether at least one element in the array passes the test implemented by the provided function.

```
const numbers = [1, 2, 3, 4, 5];

const hasEven = numbers.some(num => num % 2 === 0);
console.log(hasEven);

const hasLargerthan10 = numbers.some(num => num > 10);
console.log(hasLargerthan10);
```

33. What is the every() method?

every() tests whether all elements in the array pass the test implemented by the provided function.

```
const numbers = [2, 4, 6, 8, 10];

const allEven = numbers.every(num => num % 2 === 0);
console.log(allEven);

const allPositive = numbers.every(num => num > 0);
console.log(allPositive);
```

34. What is Object.keys()?

Object.keys() returns an array of a given object's own enumerable property names.

```
const person = {
  name: "Alice",
  age: 30,
  city: "NYC"
};

const keys = Object.keys(person);
console.log(keys);

keys.forEach(key => {
  console.log(key + ": " + person[key]);
});
```

35. What is Object.values()?

Object.values() returns an array of a given object's own enumerable property values.

```
const person = {
  name: "Alice",
  age: 30,
  city: "NYC"
};

const values = Object.values(person);
console.log(values);
```

36. What is Object.entries()?

Object.entries() returns an array of a given object's own enumerable property [key, value] pairs.

```
const person = {
  name: "Alice",
  age: 30,
  city: "NYC"
};

const entries = Object.entries(person);
console.log(entries);

entries.forEach(([key, value]) => {
  console.log(key + ": " + value);
});
```

37. What is the difference between Object.freeze() and Object.seal()?

Object.freeze() prevents modification of existing properties and prevents adding new properties. Object.seal() prevents adding/removing properties but allows modification of existing ones.

```
const frozen = Object.freeze({ name: "Alice" });
frozen.name = "Bob";
frozen.age = 30;
console.log(frozen);

const sealed = Object.seal({ name: "Alice" });
sealed.name = "Bob";
sealed.age = 30;
console.log(sealed);
```

38. What is JSON.stringify()?

JSON.stringify() converts a JavaScript value to a JSON string.

```
const obj = {
  name: "Alice",
  age: 30,
  hobbies: ["reading", "coding"]
};

const jsonString = JSON.stringify(obj);
console.log(jsonString);

const formatted = JSON.stringify(obj, null, 2);
console.log(formatted);
```

39. What is JSON.parse()?

JSON.parse() parses a JSON string and constructs the JavaScript value or object described by the string.

```
const jsonString = '{"name":"Alice","age":30}';
const obj = JSON.parse(jsonString);
console.log(obj);
console.log(obj.name);
```

40. What are classes in JavaScript?

Classes are templates for creating objects, providing a cleaner syntax for constructor functions and prototypes.

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

  greet() {
    console.log(`Hello, I'm ${this.name}`);
  }
}

const alice = new Person("Alice", 30);
alice.greet();
```

41. What is inheritance in JavaScript?

Inheritance allows a class to inherit properties and methods from another class using the extends keyword.

```
class Animal {
  constructor(name) {
    this.name = name;
  }

  speak() {
    console.log(`${this.name} makes a sound`);
  }
}

class Dog extends Animal {
  speak() {
    console.log(`${this.name} barks`);
  }
}

const dog = new Dog("Rex");
dog.speak();
```

42. What are getters and setters?

Getters and setters are special methods that provide access to object properties.

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

  get fullName() {
    return `${this.firstName} ${this.lastName}`;
  }

  set fullName(name) {
    const parts = name.split(" ");
    this.firstName = parts[0];
    this.lastName = parts[1];
  }
}

const person = new Person("Alice", "Smith");
console.log(person.fullName);
person.fullName = "Bob Johnson";
console.log(person.fullName);
```

43. What are static methods?

Static methods are called on the class itself, not on instances of the class.

```
class MathUtils {
  static add(a, b) {
    return a + b;
  }

  static multiply(a, b) {
    return a * b;
  }
}

console.log(MathUtils.add(5, 3));
console.log(MathUtils.multiply(5, 3));
```

44. What is the super keyword?

super is used to call functions on a parent class and access parent properties.

```
class Animal {
  constructor(name) {
    this.name = name;
  }

  speak() {
    console.log(`${this.name} makes a sound`);
  }
}

class Dog extends Animal {
  constructor(name, breed) {
    super(name);
    this.breed = breed;
  }

  speak() {
    super.speak();
    console.log(`${this.name} barks`);
  }
}

const dog = new Dog("Rex", "Labrador");
dog.speak();
```

45. What is the new keyword?

The new keyword creates an instance of a user-defined object type or constructor function.

```
function Person(name, age) {
  this.name = name;
  this.age = age;
}

const alice = new Person("Alice", 30);
console.log(alice);

const obj = new Object();
const arr = new Array(1, 2, 3);
console.log(obj);
console.log(arr);
```

46. What is the instanceof operator?

instanceof tests whether an object is an instance of a specific constructor.

```
class Person {}
class Animal {}

const alice = new Person();
const dog = new Animal();

console.log(alice instanceof Person);
console.log(alice instanceof Animal);
console.log(dog instanceof Animal);

console.log([] instanceof Array);
console.log({} instanceof Object);
```

47. What is the typeof operator?

typeof returns a string indicating the type of the operand.

```
console.log(typeof 42);
console.log(typeof "hello");
console.log(typeof true);
console.log(typeof undefined);
console.log(typeof null);
console.log(typeof {});
console.log(typeof []);
console.log(typeof function() {});
```

48. What are Set objects?

Set is a collection of unique values where each value may occur only once.

```
const set = new Set([1, 2, 3, 3, 4, 4, 5]);
console.log(set);

set.add(6);
set.delete(2);
console.log(set.has(3));
console.log(set.size);

set.forEach(value => console.log(value));
```

49. What are Map objects?

Map is a collection of key-value pairs where keys can be any type.

```
const map = new Map();
map.set("name", "Alice");
map.set("age", 30);
map.set(1, "one");

console.log(map.get("name"));
console.log(map.has("age"));
console.log(map.size);

map.forEach((value, key) => {
  console.log(key + ": " + value);
});
```

50. What is the difference between Set and Array?

Set stores unique values and has O(1) lookup time. Array allows duplicates and has O(n) lookup time.

```
const arr = [1, 2, 3, 3, 4, 4, 5];
const set = new Set(arr);

console.log(arr);
console.log([...set]);

console.log(arr.includes(3));
console.log(set.has(3));
```

51. What are WeakMap and WeakSet?

WeakMap and WeakSet hold weak references to objects, allowing garbage collection if no other references exist.

```
let obj1 = { name: "Alice" };
let obj2 = { name: "Bob" };

const weakMap = new WeakMap();
weakMap.set(obj1, "value1");
weakMap.set(obj2, "value2");

console.log(weakMap.get(obj1));

obj1 = null;
```

52. What is the ternary operator?

The ternary operator is a shorthand for if-else statements.

```
const age = 18;
const canVote = age >= 18 ? "Yes" : "No";
console.log(canVote);

const number = 5;
const result = number % 2 === 0 ? "Even" : "Odd";
console.log(result);
```

53. What is short-circuit evaluation?

Short-circuit evaluation stops evaluating an expression as soon as the result is determined.

```
const result1 = true || console.log("Not executed");
const result2 = false && console.log("Not executed");

const name = "" || "Default Name";
console.log(name);

const user = null;
const userName = user && user.name;
console.log(userName);
```

54. What is the nullish coalescing operator?

The nullish coalescing operator (??) returns the right operand when the left is null or undefined.

```
const value1 = null ?? "default";
console.log(value1);

const value2 = 0 ?? "default";
console.log(value2);

const value3 = "" || "default";
console.log(value3);
```

55. What is optional chaining?

Optional chaining (?.) allows safe access to nested object properties without checking each level.

```
const user = {
  name: "Alice",
  address: {
    city: "NYC"
  }
};

console.log(user?.address?.city);
console.log(user?.phone?.number);

const arr = null;
console.log(arr?.[0]);
```

56. What is the difference between deep copy and shallow copy?

Shallow copy copies only the first level, while deep copy copies all nested levels.

```
const original = { a: 1, b: { c: 2 } };

const shallow = { ...original };
shallow.b.c = 3;
console.log(original.b.c);

const deep = JSON.parse(JSON.stringify(original));
deep.b.c = 4;
console.log(original.b.c);
```

57. What is the difference between slice() and splice()?

slice() returns a shallow copy without modifying the original array. splice() modifies the original array.

```
const arr1 = [1, 2, 3, 4, 5];
const sliced = arr1.slice(1, 3);
console.log(sliced);
console.log(arr1);

const arr2 = [1, 2, 3, 4, 5];
const spliced = arr2.splice(1, 2);
console.log(spliced);
console.log(arr2);
```

58. What is the concat() method?

concat() merges two or more arrays and returns a new array.

```
const arr1 = [1, 2, 3];
const arr2 = [4, 5, 6];
const arr3 = [7, 8, 9];

const combined = arr1.concat(arr2, arr3);
console.log(combined);
console.log(arr1);
```

59. What is the join() method?

join() creates a string by concatenating all array elements separated by a specified separator.

```
const arr = ["Hello", "World", "JavaScript"];
const str1 = arr.join(" ");
console.log(str1);

const str2 = arr.join("-");
console.log(str2);
```

60. What is the split() method?

split() divides a string into an array of substrings.

```
const str = "Hello World JavaScript";
const arr1 = str.split(" ");
console.log(arr1);

const str2 = "a,b,c,d";
const arr2 = str2.split(",");
console.log(arr2);
```

61. What is the includes() method?

includes() determines whether an array or string includes a certain value.

```
const arr = [1, 2, 3, 4, 5];
console.log(arr.includes(3));
console.log(arr.includes(10));

const str = "Hello World";
console.log(str.includes("World"));
console.log(str.includes("world"));
```

62. What is the indexOf() method?

indexOf() returns the first index at which a given element can be found, or -1 if not present.

```
const arr = [1, 2, 3, 4, 5, 3];
console.log(arr.indexOf(3));
console.log(arr.indexOf(10));

const str = "Hello World";
console.log(str.indexOf("o"));
console.log(str.indexOf("z"));
```

63. What is the lastIndexOf() method?

lastIndexOf() returns the last index at which a given element can be found.

```
const arr = [1, 2, 3, 4, 5, 3];
console.log(arr.lastIndexOf(3));
console.log(arr.lastIndexOf(10));

const str = "Hello World";
console.log(str.lastIndexOf("o"));
```

64. What is the reverse() method?

reverse() reverses the order of elements in an array in place.

```
const arr = [1, 2, 3, 4, 5];
arr.reverse();
console.log(arr);

const str = "Hello";
const reversed = str.split("").reverse().join("");
console.log(reversed);
```

65. What is the sort() method?

sort() sorts the elements of an array in place and returns the sorted array.

```
const arr1 = [3, 1, 4, 1, 5, 9, 2, 6];
arr1.sort();
console.log(arr1);

const arr2 = [3, 1, 4, 1, 5, 9, 2, 6];
arr2.sort((a, b) => a - b);
console.log(arr2);

const arr3 = [3, 1, 4, 1, 5, 9, 2, 6];
arr3.sort((a, b) => b - a);
console.log(arr3);
```

66. What is the push() method?

push() adds one or more elements to the end of an array and returns the new length.

```
const arr = [1, 2, 3];
const newLength = arr.push(4, 5);
console.log(arr);
console.log(newLength);
```

67. What is the pop() method?

pop() removes the last element from an array and returns that element.

```
const arr = [1, 2, 3, 4, 5];
const removed = arr.pop();
console.log(removed);
console.log(arr);
```

68. What is the shift() method?

shift() removes the first element from an array and returns that element.

```
const arr = [1, 2, 3, 4, 5];
const removed = arr.shift();
console.log(removed);
console.log(arr);
```

69. What is the unshift() method?

unshift() adds one or more elements to the beginning of an array and returns the new length.

```
const arr = [3, 4, 5];
const newLength = arr.unshift(1, 2);
console.log(arr);
console.log(newLength);
```

70. What is the fill() method?

fill() changes all elements in an array to a static value.

```
const arr1 = [1, 2, 3, 4, 5];
arr1.fill(0);
console.log(arr1);

const arr2 = [1, 2, 3, 4, 5];
arr2.fill(9, 2, 4);
console.log(arr2);
```

71. What is the flat() method?

flat() creates a new array with all sub-array elements concatenated into it recursively up to the specified depth.

```
const arr1 = [1, 2, [3, 4]];
console.log(arr1.flat());

const arr2 = [1, 2, [3, 4, [5, 6]]];
console.log(arr2.flat());
console.log(arr2.flat(2));
```

72. What is the flatMap() method?

flatMap() maps each element using a mapping function, then flattens the result into a new array.

```
const arr = [1, 2, 3];
const result = arr.flatMap(x => [x, x * 2]);
console.log(result);

const sentences = ["Hello World", "JavaScript"];
const words = sentences.flatMap(s => s.split(" "));
console.log(words);
```

73. What is the charAt() method?

charAt() returns the character at a specified index in a string.

```
const str = "Hello World";
console.log(str.charAt(0));
console.log(str.charAt(6));
console.log(str.charAt(20));
```

74. What is the substring() method?

substring() extracts characters from a string between two specified indices.

```
const str = "Hello World";
console.log(str.substring(0, 5));
console.log(str.substring(6));
console.log(str.substring(6, 11));
```

75. What is the trim() method?

trim() removes whitespace from both ends of a string.

```
const str = "   Hello World   ";
console.log(str.trim());
console.log(str.trimStart());
console.log(str.trimEnd());
```

Part 2: Tricky Coding Questions (25)

1. Write a function to reverse a string

```
function reverseString(str) {
  return str.split("").reverse().join("");
}

console.log(reverseString("hello"));
console.log(reverseString("JavaScript"));
```

2. Write a function to check if a string is a palindrome

```
function isPalindrome(str) {
  const cleaned = str.toLowerCase().replace(/[^a-z0-9]/g, "");
  return cleaned === cleaned.split("").reverse().join("");
}

console.log(isPalindrome("racecar"));
console.log(isPalindrome("A man a plan a canal Panama"));
console.log(isPalindrome("hello"));
```

3. Write a function to find the largest number in an array

```
function findLargest(arr) {
  return Math.max(...arr);
}

function findLargestLoop(arr) {
  let max = arr[0];
  for (let i = 1; i < arr.length; i++) {
    if (arr[i] > max) {
      max = arr[i];
    }
  }
  return max;
}

console.log(findLargest([3, 7, 2, 9, 1]));
console.log(findLargestLoop([3, 7, 2, 9, 1]));
```

4. Write a function to remove duplicates from an array

```
function removeDuplicates(arr) {
  return [...new Set(arr)];
}

function removeDuplicatesFilter(arr) {
  return arr.filter((item, index) => arr.indexOf(item) === index);
}

console.log(removeDuplicates([1, 2, 3, 2, 4, 3, 5]));
console.log(removeDuplicatesFilter([1, 2, 3, 2, 4, 3, 5]));
```

5. Write a function to flatten a nested array

```
function flattenArray(arr) {
  return arr.reduce((acc, val) =>
    Array.isArray(val) ? acc.concat(flattenArray(val)) : acc.concat(val), []
  );
}

function flattenArrayFlat(arr) {
  return arr.flat(Infinity);
}

console.log(flattenArray([1, [2, [3, [4]], 5]]));
console.log(flattenArrayFlat([1, [2, [3, [4]], 5]]));
```

6. Write a function to count the occurrences of each element in an array

```
function countOccurrences(arr) {
  return arr.reduce((acc, val) => {
    acc[val] = (acc[val] || 0) + 1;
    return acc;
  }, {});
}

console.log(countOccurrences([1, 2, 2, 3, 3, 3, 4]));
console.log(countOccurrences(["a", "b", "a", "c", "b", "a"]));
```

7. Write a function to find the factorial of a number

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

function factorialIterative(n) {
  let result = 1;
  for (let i = 2; i <= n; i++) {
    result *= i;
  }
  return result;
}

console.log(factorial(5));
console.log(factorialIterative(5));
```

8. Write a function to generate Fibonacci sequence

```
function fibonacci(n) {
  if (n <= 1) return n;
  return fibonacci(n - 1) + fibonacci(n - 2);
}

function fibonacciSequence(n) {
  const fib = [0, 1];
  for (let i = 2; i < n; i++) {
    fib[i] = fib[i - 1] + fib[i - 2];
  }
  return fib.slice(0, n);
}

console.log(fibonacci(7));
console.log(fibonacciSequence(10));
```

9. Write a function to check if two strings are anagrams

```
function areAnagrams(str1, str2) {
  const normalize = str => str.toLowerCase().replace(/[^a-z0-9]/g, "").split("").s
  return normalize(str1) === normalize(str2);
}

console.log(areAnagrams("listen", "silent"));
console.log(areAnagrams("hello", "world"));
```

10. Write a function to find the second largest number in an array

```
function secondLargest(arr) {
  const unique = [...new Set(arr)].sort((a, b) => b - a);
  return unique[1];
}

function secondLargestLoop(arr) {
  let first = -Infinity, second = -Infinity;
  for (let num of arr) {
    if (num > first) {
      second = first;
      first = num;
    } else if (num > second && num < first) {
      second = num;
    }
  }
  return second;
}

console.log(secondLargest([3, 7, 2, 9, 1, 9]));
console.log(secondLargestLoop([3, 7, 2, 9, 1, 9]));
```

11. Write a function to chunk an array into smaller arrays

```
function chunkArray(arr, size) {
  const result = [];
  for (let i = 0; i < arr.length; i += size) {
    result.push(arr.slice(i, i + size));
  }
  return result;
}

console.log(chunkArray([1, 2, 3, 4, 5, 6, 7, 8], 3));
```

12. Write a function to capitalize the first letter of each word

```
function capitalizeWords(str) {
  return str.split(" ").map(word =>
    word.charAt(0).toUpperCase() + word.slice(1).toLowerCase()
  ).join(" ");
}

console.log(capitalizeWords("hello world javascript"));
console.log(capitalizeWords("the quick brown fox"));
```

13. Write a function to find the missing number in an array of 1 to n

```
function findMissingNumber(arr, n) {
  const expectedSum = (n * (n + 1)) / 2;
  const actualSum = arr.reduce((sum, num) => sum + num, 0);
  return expectedSum - actualSum;
}

console.log(findMissingNumber([1, 2, 3, 5, 6], 6));
console.log(findMissingNumber([1, 2, 4, 5, 6, 7, 8, 9, 10], 10));
```

14. Write a function to merge two sorted arrays

```
function mergeSortedArrays(arr1, arr2) {
  const result = [];
  let i = 0, j = 0;

  while (i < arr1.length && j < arr2.length) {
    if (arr1[i] < arr2[j]) {
      result.push(arr1[i++]);
    } else {
      result.push(arr2[j++]);
    }
  }

  return result.concat(arr1.slice(i)).concat(arr2.slice(j));
}

console.log(mergeSortedArrays([1, 3, 5, 7], [2, 4, 6, 8]));
```

15. Write a function to debounce a function

```
function debounce(func, delay) {
  let timeoutId;
  return function(...args) {
    clearTimeout(timeoutId);
    timeoutId = setTimeout(() => func.apply(this, args), delay);
  };
}

const debouncedLog = debounce(() => console.log("Debounced!"), 1000);
debouncedLog();
debouncedLog();
debouncedLog();
```

16. Write a function to throttle a function

```
function throttle(func, limit) {
  let inThrottle;
  return function(...args) {
    if (!inThrottle) {
      func.apply(this, args);
      inThrottle = true;
      setTimeout(() => inThrottle = false, limit);
    }
  };
}

const throttledLog = throttle(() => console.log("Throttled!"), 1000);
throttledLog();
throttledLog();
throttledLog();
```

17. Write a function to deep clone an object

```
function deepClone(obj) {
  if (obj === null || typeof obj !== "object") return obj;
  if (obj instanceof Date) return new Date(obj);
  if (obj instanceof Array) return obj.map(item => deepClone(item));

  const cloned = {};
  for (let key in obj) {
    if (obj.hasOwnProperty(key)) {
      cloned[key] = deepClone(obj[key]);
    }
  }
  return cloned;
}

const original = { a: 1, b: { c: 2, d: [3, 4] } };
const cloned = deepClone(original);
cloned.b.c = 99;
console.log(original.b.c);
console.log(cloned.b.c);
```

18. Write a function to implement Array.prototype.map

```
Array.prototype.myMap = function(callback) {
  const result = [];
  for (let i = 0; i < this.length; i++) {
    result.push(callback(this[i], i, this));
  }
  return result;
};

const arr = [1, 2, 3, 4, 5];
const doubled = arr.myMap(x => x * 2);
console.log(doubled);
```

19. Write a function to implement Array.prototype.filter

```
Array.prototype.myFilter = function(callback) {
  const result = [];
  for (let i = 0; i < this.length; i++) {
    if (callback(this[i], i, this)) {
      result.push(this[i]);
    }
  }
  return result;
};

const arr = [1, 2, 3, 4, 5, 6];
const even = arr.myFilter(x => x % 2 === 0);
console.log(even);
```

20. Write a function to implement Array.prototype.reduce

```
Array.prototype.myReduce = function(callback, initialValue) {
  let accumulator = initialValue !== undefined ? initialValue : this[0];
  let startIndex = initialValue !== undefined ? 0 : 1;

  for (let i = startIndex; i < this.length; i++) {
    accumulator = callback(accumulator, this[i], i, this);
  }
  return accumulator;
};

const arr = [1, 2, 3, 4, 5];
const sum = arr.myReduce((acc, val) => acc + val, 0);
console.log(sum);
```

21. Write a function to find the longest word in a string

```
function longestWord(str) {
  const words = str.split(" ");
  return words.reduce((longest, current) =>
    current.length > longest.length ? current : longest
  );
}

console.log(longestWord("The quick brown fox jumped over the lazy dog"));
console.log(longestWord("JavaScript is awesome"));
```

22. Write a function to check if an object is empty

```
function isEmpty(obj) {
  return Object.keys(obj).length === 0;
}

function isEmptyFor(obj) {
  for (let key in obj) {
    if (obj.hasOwnProperty(key)) {
      return false;
    }
  }
  return true;
}

console.log(isEmpty({}));
console.log(isEmpty({ name: "Alice" }));
console.log(isEmptyFor({}));
```

23. Write a function to group objects by a property

```
function groupBy(arr, key) {
  return arr.reduce((acc, obj) => {
    const groupKey = obj[key];
    if (!acc[groupKey]) {
      acc[groupKey] = [];
    }
    acc[groupKey].push(obj);
    return acc;
  }, {});
}

const people = [
  { name: "Alice", age: 25 },
  { name: "Bob", age: 30 },
  { name: "Charlie", age: 25 }
];

console.log(groupBy(people, "age"));
```

24. Write a function to implement Promise.all

```
function promiseAll(promises) {
  return new Promise((resolve, reject) => {
    const results = [];
    let completed = 0;

    promises.forEach((promise, index) => {
      Promise.resolve(promise)
        .then(result => {
          results[index] = result;
          completed++;
          if (completed === promises.length) {
            resolve(results);
          }
        })
        .catch(reject);
    });
  });
}

const p1 = Promise.resolve(1);
const p2 = Promise.resolve(2);
const p3 = Promise.resolve(3);

promiseAll([p1, p2, p3]).then(results => console.log(results));
```

25. Write a function to rotate an array by k positions

```
function rotateArray(arr, k) {
  k = k % arr.length;
  return [...arr.slice(-k), ...arr.slice(0, -k)];
}

function rotateArrayReverse(arr, k) {
  k = k % arr.length;
  const reverse = (start, end) => {
    while (start < end) {
      [arr[start], arr[end]] = [arr[end], arr[start]];
      start++;
      end--;
    }
  };

  reverse(0, arr.length - 1);
  reverse(0, k - 1);
  reverse(k, arr.length - 1);
  return arr;
}

console.log(rotateArray([1, 2, 3, 4, 5], 2));
console.log(rotateArrayReverse([1, 2, 3, 4, 5, 6, 7], 3));
```