**REACT COURSE**

**BASIC SKILLS**

**.ARRAY: Looping through array elements:**

1. **Traversing an Array**

const arr = [1, 2, 3, 4, 5];

**// Using for loop**

for (let i = 0; i < arr.length; i++) {

console.log(arr[i]);

}

**// Using forEach**

arr.forEach((num) => console.log(num));

**// Using for...of**

for (let num of arr) {

console.log(num);

}

**2. Adding Elements**

push() → Add at the end

unshift() → Add at the beginning

splice() → Add at a specific position

**Example:**

let numbers = [10, 20, 30];

numbers.push(40); // [10, 20, 30, 40]

numbers.unshift(0); // [0, 10, 20, 30, 40]

numbers.splice(2, 0, 15); // [0, 10, 15, 20, 30, 40]

**3. Removing Elements**

pop() → Remove last element

shift() → Remove first element

splice() → Remove at a specific position

**Example:**

let items = ["apple", "banana", "cherry"];

items.pop(); // ["apple", "banana"]

items.shift(); // ["banana"]

items.splice(0, 1); // [] (removes first element)

**4. Updating Elements**

let colors = ["red", "green", "blue"];

colors[1] = "yellow"; // ["red", "yellow", "blue"]

**5. Searching for Elements**

indexOf() → Find index of an element

lastIndexOf() → Find last occurrence

includes() → Check if an element exists

find() → Find first match

findIndex() → Find index of first match

**Example:**

let nums = [10, 20, 30, 40];

console.log(nums.indexOf(30)); // 2

console.log(nums.includes(20)); // true

console.log(nums.find((n) => n > 25)); // 30

console.log(nums.findIndex((n) => n > 25)); // 2

**6. Filtering Elements**

filter() → Returns an array of matching elements

**Example:**

let nums = [10, 20, 30, 40];

**7. Transforming Elements**

map() → Apply a function to each element

let squared = nums.map((n) => n \* n); // [100, 400, 900, 1600]

**8. Sorting Elements**

sort() → Sorts alphabetically by default

reverse() → Reverses the array

custom sort() → Sorts numerically

**Example:**

let numbers2 = [5, 1, 8, 3];

numbers2.sort(); // [1, 3, 5, 8] (default alphabetical order)

numbers2.reverse(); // [8, 5, 3, 1]

numbers2.sort((a, b) => a - b); // [1, 3, 5, 8] (ascending order)

numbers2.sort((a, b) => b - a); // [8, 5, 3, 1] (descending order)

**9. Combining and Slicing**

concat() → Merge arrays

slice() → Extract a portion

let arr1 = [1, 2, 3];

let arr2 = [4, 5, 6];

let combined = arr1.concat(arr2); // [1, 2, 3, 4, 5, 6]

let sliced = combined.slice(2, 4); // [3, 4]

**10. Reducing an Array**

reduce() → Aggregate values

let sum = nums.reduce((acc, num) => acc + num, 0); // Sum of elements

let max = nums.reduce((acc, num) => (num > acc ? num : acc), nums[0]); // Find max

**11. Checking Conditions**

every() → Checks if all elements match a condition

some() → Checks if any element matches a condition

console.log(nums.every((n) => n > 0)); // true (all are positive)

console.log(nums.some((n) => n > 50)); // false (none are > 50)

**12. Converting an Array**

join() → Convert to string

toString() → Convert to a comma-separated string

let fruits = ["apple", "banana", "cherry"];

console.log(fruits.join(" - ")); // "apple - banana - cherry"

console.log(fruits.toString()); // "apple,banana,cherry"

**13. Filling an Array**

fill() → Fill with a value

let emptyArray = new Array(5).fill(0); // [0, 0, 0, 0, 0]

**14. Flattening Nested Arrays**

flat() → Flattens an array

let nested = [1, [2, 3], [4, [5, 6]]];

console.log(nested.flat(2)); // [1, 2, 3, 4, 5, 6]

**15. Spreading and Destructuring**

Spread Operator **(...)** → Copy or merge arrays

Destructuring → Assign variables

let arr3 = [...arr1, ...arr2]; // Merge arrays

let [first, second, ...rest] = arr3; // Destructuring

**16. Converting Objects to Arrays**

Object.keys() → Get keys

Object.values() → Get values

Object.entries() → Get key-value pairs

let person = { name: "John", age: 30 };

console.log(Object.keys(person)); // ["name", "age"]

console.log(Object.values(person)); // ["John", 30]

console.log(Object.entries(person));// [["name", "John"], ["age", 30]]

**let evenNumbers = nums.filter((n) => n % 2 === 0); // [10, 20, 30, 40]**

**OBJECTS**

**Here’s a comprehensive list of operations on objects in JavaScript,**

**JavaScript Object Operations**

**1. Creating an Object**

**const person = {**

**name: "John",**

**age: 30,**

**job: "Developer"**

**};**

**// Using Object Constructor**

**const user = new Object();**

**user.name = "Alice";**

**user.age = 25;**

**2. Accessing Object Properties**

**Dot notation**

**Bracket notation**

**console.log(person.name); // "John"**

**console.log(person["age"]); // 30**

**3. Adding/Updating Properties**

**person.city = "New York"; // Adding new property**

**person.age = 35; // Updating property**

**4. Removing Properties**

**delete person.job;**

**console.log(person); // { name: "John", age: 35, city: "New York" }**

**5. Checking if a Property Exists**

**hasOwnProperty()**

**'key' in object**

**console.log(person.hasOwnProperty("name")); // true**

**console.log("age" in person); // true**

**6. Looping Through an Object**

**for...in**

**for (let key in person) {**

**console.log(`${key}: ${person[key]}`);**

**}**

**7. Getting Keys, Values, and Entries**

**Object.keys() → Get an array of keys**

**Object.values() → Get an array of values**

**Object.entries() → Get an array of key-value pairs**

**console.log(Object.keys(person)); // ["name", "age", "city"]**

**console.log(Object.values(person)); // ["John", 35, "New York"]**

**console.log(Object.entries(person));// [["name", "John"], ["age", 35], ["city", "New York"]]**

**8. Copying an Object**

**Spread operator (...)**

**Object.assign()**

**const newPerson = { ...person };**

**const clonedPerson = Object.assign({}, person);**

**9. Merging Objects**

**const details = { country: "USA", gender: "Male" };**

**const merged = { ...person, ...details };**

**console.log(merged);**

**// { name: "John", age: 35, city: "New York", country: "USA", gender: "Male" }**

**10. Converting an Object to an Array**

**const personArray = Object.entries(person);**

**console.log(personArray);**

**// [["name", "John"], ["age", 35], ["city", "New York"]]**

**11. Converting an Array to an Object**

**const personFromArray = Object.fromEntries(personArray);**

**console.log(personFromArray);**

**// { name: "John", age: 35, city: "New York" }**

**12. Checking if an Object is Empty**

**console.log(Object.keys(person).length === 0); // false**

**13. Freezing and Sealing Objects**

**Object.freeze() → Prevents adding, updating, or deleting properties**

**Object.seal() → Prevents adding/deleting but allows modifying existing properties**

**Object.freeze(person);**

**person.name = "Mike"; // Won't change**

**Object.seal(person);**

**person.age = 40; // Can modify existing properties**

**delete person.city; // Won't work**

**14. Object Destructuring**

**const { name, age } = person;**

**console.log(name, age); // "John" 35**

**15. Nesting Objects**

**const userInfo = {**

**name: "Alice",**

**address: {**

**city: "Los Angeles",**

**country: "USA"**

**}**

**};**

**console.log(userInfo.address.city); // "Los Angeles"**

**16. Using Object Methods**

**const student = {**

**name: "Emma",**

**getInfo: function () {**

**return `${this.name} is a student.`;**

**}**

**};**

**console.log(student.getInfo()); // "Emma is a student."**

**17. Object with Dynamic Keys**

**let keyName = "email";**

**const profile = {**

**[keyName]: "john@example.com"**

**};**

**console.log(profile.email); // "john@example.com"**