Chapter 5

Objects and Arrays

**Overview of Objects**

**Syntax**

let objectName = {

key1: value1, // Property

key2: value2, // Property

methodName: function() {

// Method body

}

};

**Example**

let human = {

firstName: "Rupali",

lastName: "Chopade",

age: 30,

greet: function() {

console.log(`Hello, my name is ${this.firstName} ${this.lastName}.`);

}

};

console.log(human.firstName); // Output: "Rupali"

human.greet(); // Output: "Hello, my name is Rupali Chopade."

**Key Characteristics of JavaScript Objects**

1. **Dynamic Nature**:

human.middleName = "Mahesh"; // Add new property

human.age = 31; // Update property

delete human.lastName; // Delete property

**Creating Objects**

**1. Object Literals**

**Syntax**

let objectName = {

property1: value1,

property2: value2,

methodName: function() {

// method body

}

};

**Example**

let cars = {

brand: "Toyota",

model: "Corolla",

year: 2023

};

**Example**

cars.color = "red"; // Adding a new property

delete cars.year; // Deleting a property

console.log(cars);// Output: { brand: "Toyota", model: "Corolla", color: "red", start: [Function] }

**2. Object Constructor**

The Object constructor can also be used to create objects.

let cars = new Object();

cars.brand = "Toyota";

cars.model = "Corolla";

**3. Constructor Functions**

**Syntax**

function ConstructorName(parameter1, parameter2) {

this.property1 = parameter1;

this.property2 = parameter2;

this.methodName = function() {

// method body

};

}

**Example**

function Car(brand, model, year) {

this.brand = brand;

this.model = model;

this.year = year;

}

let myCar = new Car("Honda", "Civic", 2020);

**Advanced Techniques**

**Object.create()**

**Example**

let animal = {

speak: function() {

console.log("The animal makes a sound.");

}

};

let dog = Object.create(animal);

dog.speak(); // Output: "The animal makes a sound."

**Class Syntax (ES6+)**

**Example**

class Car {

constructor(brand, model, year) {

this.brand = brand;

this.model = model;

this.year = year;

}

start() {

console.log(`${this.brand} ${this.model} is starting...`);

}

}

let car1 = new Car("Tesla", "Model S", 2022);

car1.start(); // Output: "Tesla Model S is starting..."

1. **Use Prototypes for Methods**
   * **Example**

function Car(brand, model) {

this.brand = brand;

this.model = model;

}

Car.prototype.start = function() {

console.log(`${this.brand} ${this.model} is starting...`);

};

**Accessing Object Properties**

**1. Dot Notation**

* **Syntax:** object.propertyName

**Example**

let person = {

firstName: "Rupali",

age: 25

};

console.log(person.firstName); // Output: "Rupali"

console.log(person.age); // Output: 25

**2. Bracket Notation**

* **Syntax:** object["propertyName"]

**Example**:

let person = {

"full name": "Rupali Chopade",

age: 35

};

console.log(person["full name"]); // Output: "Rupali Chopade"

let key = "age";

console.log(person[key]); // Output: 35

**Key Differences**

**1. Updating Existing Properties**

**Example**

let car = {

brand: "Toyota",

model: "Corolla"

};

car.model = "Camry"; // Using dot notation

car["brand"] = "Honda"; // Using bracket notation

console.log(car); // Output: { brand: "Honda", model: "Camry" }

**2. Adding New Properties**

**Example**:

let car = {

brand: "Toyota"

};

car.year = 2020; // Add using dot notation

car["color"] = "Blue"; // Add using bracket notation

console.log(car);

// Output: { brand: "Toyota", year: 2020, color: "Blue" }

**3. Removing Properties**

**Example**

let car = {

brand: "Toyota",

year: 2020

};

delete car.year; // Removes the "year" property

console.log(car); // Output: { brand: "Toyota" }

**Dynamic Property Access**

**Example**

let propName = "age";

let person = {

name: "Aditi",

age: 25

};

console.log(person[propName]); // Access dynamically: Output: 25

person[propName] = 26; // Modify dynamically

console.log(person.age); // Output: 26

**Iterating Over Object Properties**

**1. Using for...in Loop**

**Example**

let person = {

firstName: "Sujata",

age: 35,

city: "Pune"

};

for (let key in person) {

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

}

**2. Using Object.keys()**

**Example**:

let person = {

firstName: "Sujata",

age: 35

};

Object.keys(person).forEach(key => {

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

});

**3. Using Object.entries()**

**Example**:

let person = {

firstName: "Sujata",

age: 35

};

Object.entries(person).forEach(([key, value]) => {

console.log(`${key}: ${value}`);

});

**Advanced Concepts**

**1. Accessing Nested Properties**

**Example**

let company = {

name: "TechCorp",

location: {

city: "San Francisco",

state: "California"

}

};

console.log(company.location.city); // Output: "San Francisco"

console.log(company["location"]["state"]); // Output: "California"

**2. Using Optional Chaining (?.)**

**Example**

let company = {

name: "TechCorp"

};

console.log(company.location?.city); // Output: undefined (no error thrown)

**3. Accessing Property Names Dynamically**

**Example**

let propName = "dynamicProp";

let obj = {

[propName]: "Dynamic Value"

};

console.log(obj.dynamicProp); // Output: "Dynamic Value"

**Common Pitfalls**

1. **Accessing Non-Existent Properties**

let obj = { a: 1 };

console.log(obj.b); // Output: undefined

1. **Overwriting Existing Properties**

let obj = { a: 1 };

obj.a = 2;

console.log(obj.a); // Output: 2

1. **Unintentional Mutation**

**Example**

function modify(obj) {

obj.a = 100;

}

let myObj = { a: 1 };

modify(myObj);

console.log(myObj.a); // Output: 100

**Defining Methods**

**1. Adding a Function as a Property**

**Example**

let person = {

firstName: "Anil",

lastName: "Kapoor",

fullName: function() {

return this.firstName + " " + this.lastName;

}

};

console.log(person.fullName()); // Output: "Anil Kapoor"

**Example**

let car = {

brand: "Toyota",

model: "Corolla",

displayInfo: function() {

return `Car: ${this.brand} ${this.model}`;

}

};

console.log(car.displayInfo()); // Output: "Car: Toyota Corolla"

**Creating Methods Using ES6 Shorthand**

**Example**

let person = {

firstName: "Alia",

lastName: "Bhat",

fullName() {

return `${this.firstName} ${this.lastName}`;

}

};

console.log(person.fullName()); // Output: "Alia Bhat"

**Adding Methods Dynamically**

**Example**

let animal = {

type: "Dog"

};

animal.sound = function() {

return "Bark!";

};

console.log(animal.sound()); // Output: "Bark!"

**Calling Methods**

**Example**

let calculator = {

add: function(a, b) {

return a + b;

},

subtract: function(a, b) {

return a - b;

}

};

console.log(calculator.add(5, 3)); // Output: 8

console.log(calculator["subtract"](5, 3)); // Output: 2

**Built-in Object Methods**

**Example**

let person = {

name: "Rashmika",

age: 30

};

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

console.log(Object.values(person)); // Output: ["Rashmika", 30]

**Method Chaining**

**Example**

let calculator = {

result: 0,

add(value) {

this.result += value;

return this; // Return the object to allow chaining

},

subtract(value) {

this.result -= value;

return this;

},

multiply(value) {

this.result \*= value;

return this;

},

getResult() {

return this.result;

}

};

let finalResult = calculator.add(5).subtract(2).multiply(3).getResult();

console.log(finalResult); // Output: 9

**Arrow Functions in Methods**

**Example**

let person = {

name: "Neha",

greet: () => {

console.log(‘Hello, ${this.name}’); // ‘this’ is not bound to the object

}

};

person.greet(); // Output: "Hello, undefined"

**Methods Inside Nested Objects**

**Example**

let library = {

name: "City Library",

books: {

count: 100,

displayCount: function() {

return `Total books: ${this.count}`;

}

}

};

console.log(library.books.displayCount()); // Output: "Total books: 100"

**Iterating Over Methods**

**Example**

let obj = {

name: "Rupali",

greet() {

return "Hello!";

},

farewell() {

return "Goodbye!";

}

};

for (let key in obj) {

if (typeof obj[key] === "function") {

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

}

}

**Working of this in Object Methods**

**Basic Example**

let person = {

name: "Alice",

greet: function() {

return `Hello, my name is ${this.name}`;

}

};

console.log(person.greet()); // Output: "Hello, my name is Alice"

**Using this with Multiple Methods**

**Example**

let calculator = {

result: 0,

add(value) {

this.result += value;

},

subtract(value) {

this.result -= value;

},

getResult() {

return this.result;

}

};

calculator.add(10);

calculator.subtract(4);

console.log(calculator.getResult()); // Output: 6

**Dynamic Context of this**

**Example: Calling a Method as a Standalone Function**

let person = {

name: "Alice",

greet: function() {

return `Hello, my name is ${this.name}`;

}

};

let standaloneGreet = person.greet;

console.log(standaloneGreet()); // Output: "Hello, my name is undefined"

**Binding this Explicitly**

Using bind()

let person = {

name: "Monali",

greet: function() {

return `Hello, my name is ${this.name}`;

}

};

let standaloneGreet = person.greet.bind(person);

console.log(standaloneGreet()); // Output: "Hello, my name is Monali"

**this in Arrow Functions**

Example

let person = {

name: "Alice",

greet: () => {

return `Hello, my name is ${this.name}`;

}

};

console.log(person.greet()); // Output: "Hello, my name is undefined"

**Proper Usage**

let person = {

name: "Rupali",

hobbies: ["reading", "traveling", "cycling"],

showHobbies() {

this.hobbies.forEach((hobby) => {

console.log(`${this.name} enjoys ${hobby}`);

});

}

};

person.showHobbies();

// Output:

**this in Nested Functions**

**Example**

let person = {

name: "Rupali",

greet: function() {

function innerGreet() {

return `Hello, my name is ${this.name}`;

}

return innerGreet();

}

};

console.log(person.greet()); // Output: "Hello, my name is undefined"

**Solution: Use Arrow Functions**

let person = {

name: "Rupali",

greet: function() {

let innerGreet = () => {

return `Hello, my name is ${this.name}`;

};

return innerGreet();

}

};

console.log(person.greet()); // Output: "Hello, my name is Rupali"

**this in Object Creation and Constructors**

**Example: Constructor Function**

function Person(name, age) {

this.name = name;

this.age = age;

this.greet = function() {

return `Hello, my name is ${this.name} and I am ${this.age} years old.`;

};

}

let person1 = new Person("Shubhangi", 45);

console.log(person1.greet()); // Output: "Hello, my name is Shubhangi and I am 45 years old."

**Example: Class**

class Person {

constructor(name, age) {

this.name = name;

this.age = age;

}

greet() {

return `Hello, my name is ${this.name} and I am ${this.age} years old.`;

}

}

let person1 = new Person("Mahesh", 44);

console.log(person1.greet()); // Output: "Hello, my name is Mahesh and I am 44 years old."

**About Array**

**Example**

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

console.log(fruits); // Output: ["apple", "banana", "cherry"]

**Creating Arrays**

**1. Using Array Literals**

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

**2. Using the new Array() Constructor**

let numbers = new Array(1, 2, 3);

console.log(numbers); // Output: [1, 2, 3]

**Note:** The literal syntax ([]) is preferred because it is shorter and easier to read.

**3. Creating an Empty Array**

let emptyArray = [];

console.log(emptyArray); // Output: []

**4. Creating Arrays with Predefined Length**

let fixedSizeArray = new Array(5);

console.log(fixedSizeArray);

**Accessing Array Elements**

**Syntax**

arrayName[index];

**Example**

let animals = ["cat", "dog", "rabbit"];

console.log(animals[0]); // Output: "cat"

console.log(animals[1]); // Output: "dog"

console.log(animals[2]); // Output: "rabbit"

**Example: Out-of-Bounds Access**

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

console.log(colors[5]); // Output: undefined

**Modifying Array Elements**

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

fruits[1] = "orange";

console.log(fruits); // Output: ["apple", "orange", "cherry"]

**Adding Elements to an Array**

**1. Using Index**

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

fruits[2] = "cherry";

console.log(fruits); // Output: ["apple", "banana", "cherry"]

**2. Using push()**

let numbers = [1, 2, 3];

numbers.push(4);

console.log(numbers); // Output: [1, 2, 3, 4]

**Removing Elements from an Array**

**1. Using pop()**

let numbers = [1, 2, 3];

let lastElement = numbers.pop();

console.log(numbers); // Output: [1, 2]

console.log(lastElement); // Output: 3

**2. Using shift()**

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

let firstElement = fruits.shift();

console.log(fruits); // Output: ["banana", "cherry"]

console.log(firstElement); // Output: "apple"

**Checking Array Length**

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

console.log(colors.length); // Output: 3

**Iterating Over Arrays**

**1. Using a for Loop**

let numbers = [10, 20, 30];

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

console.log(numbers[i]);

}

**2. Using forEach()**

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

colors.forEach(function(color) {

console.log(color);

});

**Checking for Element Existence**

**1. Using includes()**

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

console.log(fruits.includes("banana")); // Output: true

console.log(fruits.includes("grape")); // Output: false

**2. Using indexOf()**

let animals = ["cat", "dog", "rabbit"];

console.log(animals.indexOf("dog")); // Output: 1

console.log(animals.indexOf("lion")); // Output: -1

**Working with Nested Arrays**

**Example**

let matrix = [

[1, 2, 3],

[4, 5, 6],

[7, 8, 9]

];

console.log(matrix[0]); // Output: [1, 2, 3]

console.log(matrix[0][1]); // Output: 2

**Common Pitfalls**

let numbers = [1, 2, 3];

console.log(numbers[5]); // Output: undefined

**Mixing Data Types**

let mixedArray = [1, "two", true];

console.log(mixedArray); // Output: [1, "two", true]

**1. push() Method**

**Syntax**

array.push(element1, element2, ..., elementN);

**Example**

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

let newLength = fruits.push("cherry", "orange");

console.log(fruits); // Output: ["apple", "banana", "cherry", "orange"]

console.log(newLength); // Output: 4

**2. pop() Method**

**Syntax**

let removedElement = array.pop();

**Example**

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

let lastFruit = fruits.pop();

console.log(fruits); // Output: ["apple", "banana"]

console.log(lastFruit); // Output: "cherry"

**3. slice() Method**

**Syntax**

let newArray = array.slice(start, end);

**Example**

let numbers = [1, 2, 3, 4, 5];

let subArray = numbers.slice(1, 4);

console.log(subArray); // Output: [2, 3, 4]

console.log(numbers); // Output: [1, 2, 3, 4, 5] (original array unchanged)

**4. splice() Method**

**Syntax**

array.splice(start, deleteCount, item1, item2, ..., itemN);

**Example: Removing Elements**

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

let removed = colors.splice(1, 2);

console.log(colors); // Output: ["red", "yellow"]

console.log(removed); // Output: ["green", "blue"]

**Example: Adding Elements**

let colors = ["red", "yellow"];

colors.splice(1, 0, "green", "blue");

console.log(colors); // Output: ["red", "green", "blue", "yellow"]

**Example: Replacing Elements**

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

colors.splice(1, 1, "yellow");

console.log(colors); // Output: ["red", "yellow", "blue"]

**Practical Example Using All Methods**

let items = ["a", "b", "c", "d"];

// Push: Add an element to the end

items.push("e");

console.log(items); // Output: ["a", "b", "c", "d", "e"]

// Pop: Remove the last element

let popped = items.pop();

console.log(items); // Output: ["a", "b", "c", "d"]

console.log(popped); // Output: "e"

// Slice: Extract a portion of the array

let sliced = items.slice(1, 3);

console.log(sliced); // Output: ["b", "c"]

console.log(items); // Output: ["a", "b", "c", "d"]

// Splice: Remove and replace elements

items.splice(1, 2, "x", "y");

console.log(items); // Output: ["a", "x", "y", "d"]

**1. forEach() Method**

**Syntax**

array.forEach(callback(element, index, array));

* **callback: A function that executes for each element.**

**Example: Logging Array Elements**

let numbers = [1, 2, 3, 4];

numbers.forEach((number, index) => {

console.log(`Index ${index}: ${number}`);

});

**2. map() Method**

**Syntax**

let newArray = array.map(callback(element, index, array));

**Example: Doubling Array Values**

let numbers = [1, 2, 3, 4];

let doubled = numbers.map((number) => number \* 2);

console.log(doubled); // Output: [2, 4, 6, 8]

console.log(numbers); // Output: [1, 2, 3, 4] (original array unchanged)

**3. filter() Method**

**Syntax**

let newArray = array.filter(callback(element, index, array));

**Example: Filtering Even Numbers**

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

let evens = numbers.filter((number) => number % 2 === 0);

console.log(evens); // Output: [2, 4, 6]

console.log(numbers); // Output: [1, 2, 3, 4, 5, 6] (original array unchanged)

**Practical Example: Using All Three Methods**

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

// forEach: Log each number

numbers.forEach((num) => console.log(`Number: ${num}`));

// map: Create an array of squares

let squares = numbers.map((num) => num \* num);

console.log(squares); // Output: [1, 4, 9, 16, 25, 36]

// filter: Extract numbers greater than 3

let greaterThanThree = numbers.filter((num) => num > 3);

console.log(greaterThanThree); // Output: [4, 5, 6]

**Creating Multidimensional Arrays**

let matrix = [

[1, 2, 3], // Row 1

[4, 5, 6], // Row 2

[7, 8, 9] // Row 3

];

console.log(matrix);

**Accessing Elements**

console.log(matrix[0][1]); // Output: 2 (1st row, 2nd column)

console.log(matrix[2][0]); // Output: 7 (3rd row, 1st column)

**Manipulating Elements**

matrix[1][2] = 10; // Change the value in the 2nd row, 3rd column

console.log(matrix);

**Iterating Over Multidimensional Arrays**

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

for (let j = 0; j < matrix[i].length; j++) {

console.log(`Element at (${i}, ${j}): ${matrix[i][j]}`);

}

}

**Practical Example: Representing a Grid**

let grid = [

['X', 'O', 'X'],

['O', 'X', 'O'],

['X', 'O', 'X']

];

console.log(grid[1][1]); // Output: X (middle element)

**Creating Nested Objects**

let student = {

name: "Alice",

grades: {

math: 90,

science: 85,

english: 92

},

address: {

city: "New York",

zip: 10001

}

};

**Accessing Properties**

console.log(student.grades.math); // Output: 90

console.log(student.address['city']); // Output: New York

**Updating Properties**

student.grades.math = 95;

console.log(student.grades.math); // Output: 95

**Adding Properties**

student.address.state = "NY";

console.log(student.address);

**Iterating Over Nested Objects**

for (let subject in student.grades) {

console.log(`${subject}: ${student.grades[subject]}`);

}

**Combining Multidimensional Arrays and Nested Objects**

let school = [

{

name: "John",

grades: { math: 88, science: 75 }

},

{

name: "Jane",

grades: { math: 92, science: 89 }

}

];

// Accessing data

console.log(school[1].grades.science); // Output: 89

// Updating data

school[0].grades.math = 90;

console.log(school[0].grades.math); // Output: 90