FUNCTIONAL PROGRAMMING

Functional Programming (FP)

It is a programming paradigm that focuses on using functions to solve problems. It emphasizes the use of pure functions (functions that have no side effects and return the same output for the same input), immutability (data that cannot be changed once it is created), and the avoidance of shared state and mutable data. In FP, functions are treated as first-class citizens and can be passed around as arguments to other functions, returned as values, and assigned to variables.

Functional Methods

forEach()

In JavaScript, the **forEach()** method is a higher-order function that is available on arrays. It allows you to iterate over the elements of an array and execute a callback function for each element. The **forEach()** method is a convenient way to perform an operation on each item in an array without the need for writing traditional **for** loops.

```
// Example 1: Simple forEach() usage
numbers.forEach((number) => {
  console.log(number);
});

// Example 2: Performing operations using forEach()
let sum = 0;
numbers.forEach((number) => {
  sum += number;
});
console.log("Sum:", sum);

// Example 3: Modifying the original array using forEach()
const squaredNumbers = [];
numbers.forEach((number) => {
  squaredNumbers.push(number * number);
});
```

Functional Methods

sort()

In JavaScript, the **sort()** method is available on arrays and is used to sort the elements of an array in place. By default, it sorts the elements as strings and modifies the original array.

```
const numbers = [5, 2, 8, 1, 4];

// Example 1: Sorting numbers in ascending order
const sortedAscending = numbers.sort((a, b) => a - b);
console.log("Ascending Order:", sortedAscending);

// Example 2: Sorting numbers in descending order
const sortedDescending = numbers.sort((a, b) => b - a);
console.log("Descending Order:", sortedDescending);

// Example 3: Sorting an array of objects based on a property
const students = [
  { name: "Alice", age: 25 },
  { name: "Bob", age: 21 },
  { name: "Charlie", age: 23 }
];

const sortedByAge = students.sort((a, b) => a.age - b.age);
console.log("Sorted by Age:", sortedByAge);
```

Functional Methods

Map()

map() is a built-in JavaScript method that is used to create a new array by applying a function to each element of an existing array. It takes a callback function as its argument. The callback function takes three arguments: the current element, the index of the current element, and the array being looped through.

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

// Example 1: Mapping numbers to their squares
const squares = numbers.map((number) => number * number);
console.log("Squared Numbers:", squares);

// Example 2: Mapping numbers to strings
const strings = numbers.map((number) => String(number));
console.log("Strings:", strings);

// Example 3: Mapping objects to a specific property
const books = [
  { title: "The Alchemist", author: "Paulo Coelho" },
  { title: "To Kill a Mockingbird", author: "Harper Lee" },
  { title: "1984", author: "George Orwell" }
];

const titles = books.map((book) => book.title);
console.log("Titles:", titles);
```

Functional Methods

Filter()

filter() is a built-in JavaScript method that is used to create a new array with all the elements of an existing array that pass a certain test. It takes a callback function as its argument.The callback function takes three arguments: the current element, the index of the current element, and the array being looped through.

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

// Example 1: Filtering even numbers
const evenNumbers = numbers.filter((number) => number % 2 === 0);
console.log("Even Numbers:", evenNumbers);

// Example 2: Filtering numbers greater than 3
const greaterThanThree = numbers.filter((number) => number > 3);
console.log("Numbers Greater Than Three:", greaterThanThree);

// Example 3: Filtering objects based on a condition
const books = [
  { title: "The Alchemist", author: "Paulo Coelho", rating: 4.18 },
  { title: "To Kill a Mockingbird", author: "Harper Lee", rating: 4.27 },
  { title: "1984", author: "George Orwell", rating: 4.16 },
  { title: "The Great Gatsby", author: "F. Scott Fitzgerald", rating: 3.90 },
  { title: "Pride and Prejudice", author: "Jane Austen", rating: 4.25 }
];

const highlyRatedBooks = books.filter((book) => book.rating >= 4.2);
console.log("Highly Rated Books:", highlyRatedBooks);
```

Functional Methods

reduce()

reduce() is a built-in JavaScript method that is used to reduce an array to a single value by applying a function to each element of the array. It takes a callback function as its argument, which is called for each element in the array. The callback function takes two arguments: the accumulator and the current element. The accumulator is the value returned by the previous invocation of the callback function, and it is initialized to the initial value provided as the second argument to reduce(). The value returned by the final invocation of the callback function is the final value of the reduce operation.

Functional Methods

reduce()

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

// Example 1: Summing up all numbers
const sum = numbers.reduce((accumulator, number) => accumulator + number, 0);
console.log("Sum:", sum);

// Example 2: Finding the maximum number
const max = numbers.reduce((accumulator, number) => Math.max(accumulator, number));
console.log("Max:", max);

// Example 3: Concatenating strings
const words = ["Hello", "World", "I", "am", "JavaScript"];
const concatenated = words.reduce((accumulator, word) => accumulator + " " + word);
console.log("Concatenated:", concatenated);
```