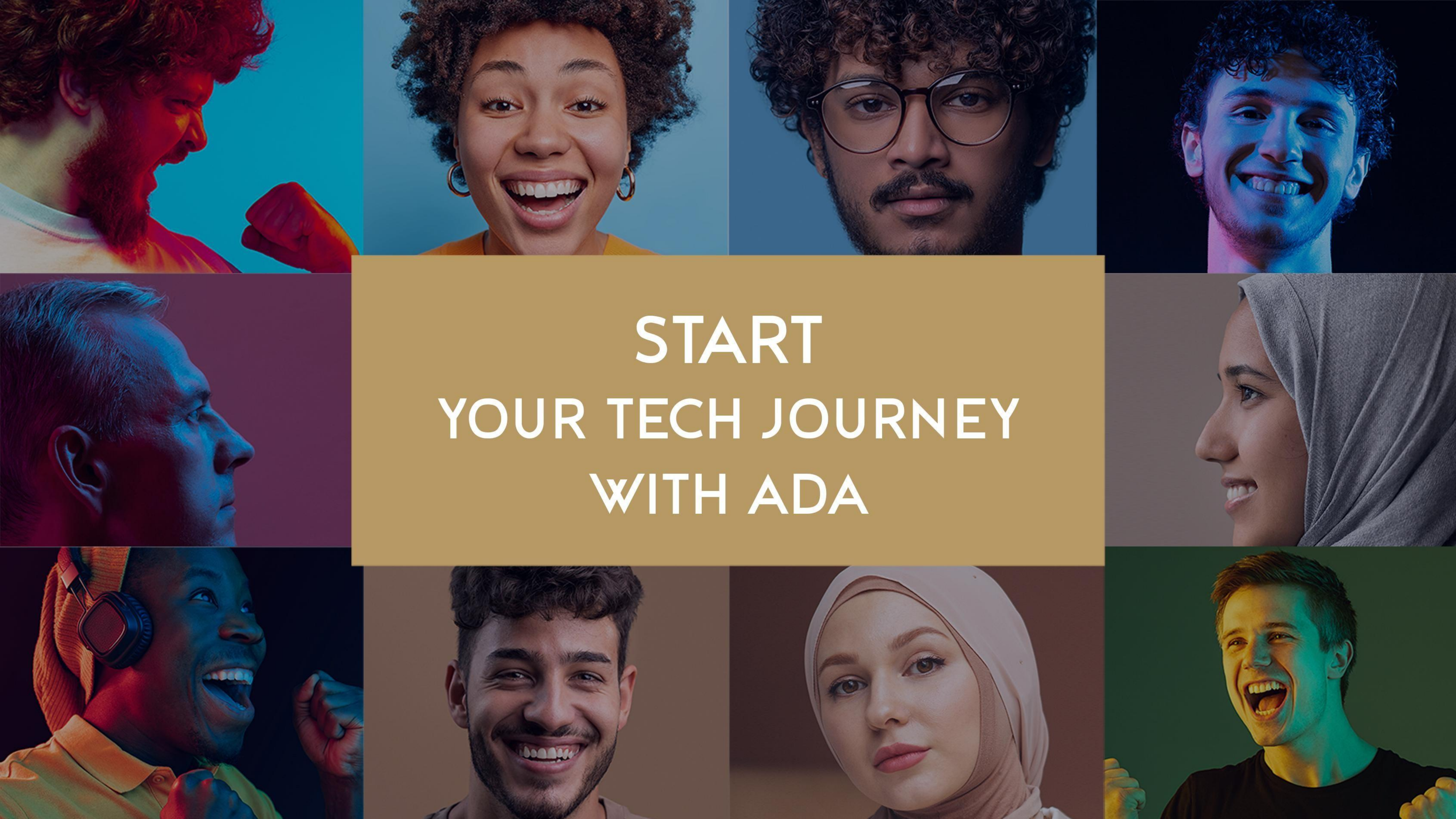
ACADEMY OF DIGITAL ARTS EGYPT

Google cisco Adobe IBM Microsoft CompTIA



START
YOUR TECH JOURNEY WITH ADA

Agenda

- ➢Array operations: push, pop, shift, unshift, slice, splice
- ➢Array iteration methods: forEach, map, filter, reduce
- ➢ES6 Features: Template literals, destructuring, spread & rest operators
- ➢Working with complex data structures
- ➢Functional programming concepts with arrays
- ➢Hands-on exercise: Extract names of students with grades above 50

Arrays Review

Creating Arrays:

```
// Array literal (most common)
let fruits = ["apple", "banana", "orange"];

// Array constructor
let numbers = new Array(1, 2, 3, 4, 5);

// Empty array
let emptyArray = [];

// Mixed data types
let mixed = ["hello", 42, true, null, {name: "Alice"}];
```

Accessing Array Elements:

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

console.log(colors[0]);         // "red" (first element)
console.log(colors[1]);         // "green"
console.log(colors[2]);         // "blue"
console.log(colors.length);     // 3
console.log(colors[colors.length - 1]); // "blue" (last element)
```

Basic Array Properties:

- Zero-indexed - First element is at index 0
- Dynamic length - Can grow and shrink
- Mixed types - Can contain different data types

Array Operations - Adding Elements

push() - Add to End:

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

fruits.push("orange");          // Returns new length (3)
console.log(fruits);            // ["apple", "banana", "orange"]

// Add multiple elements
fruits.push("grape", "mango");
console.log(fruits);            // ["apple", "banana", "orange", "grape", "mango"]
```

unshift() - Add to Beginning:

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

numbers.unshift(1);             // Returns new length (4)
console.log(numbers);           // [1, 2, 3, 4]

// Add multiple elements
numbers.unshift(-1, 0);
console.log(numbers);           // [-1, 0, 1, 2, 3, 4]
```

Array Operations - Removing Elements

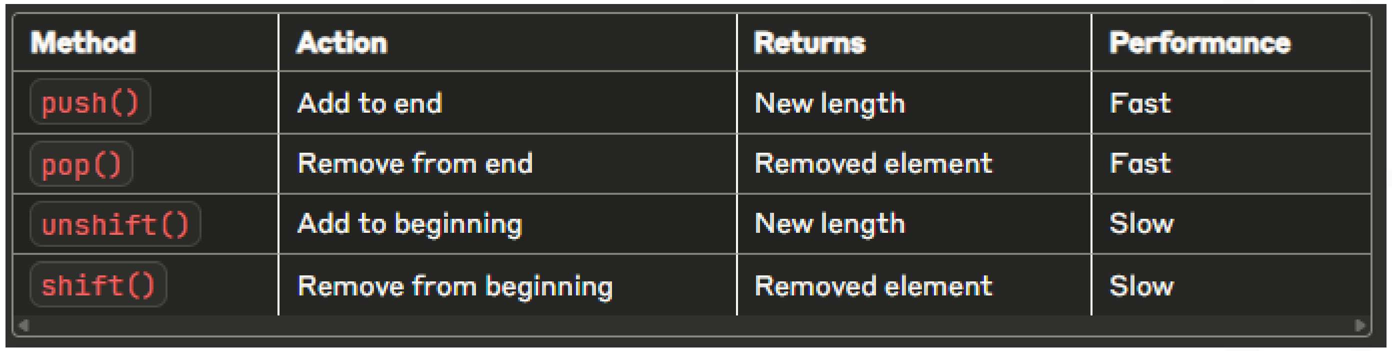
pop() - Remove from End:

```javascript
let fruits = ["apple", "banana", "orange"];

let lastFruit = fruits.pop();  // Returns removed element
console.log(lastFruit);        // "orange"
console.log(fruits);           // ["apple", "banana"]
```

shift() - Remove from Beginning:

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

let firstNumber = numbers.shift(); // Returns removed element
console.log(firstNumber);           // 1
console.log(numbers);               // [2, 3, 4]
```

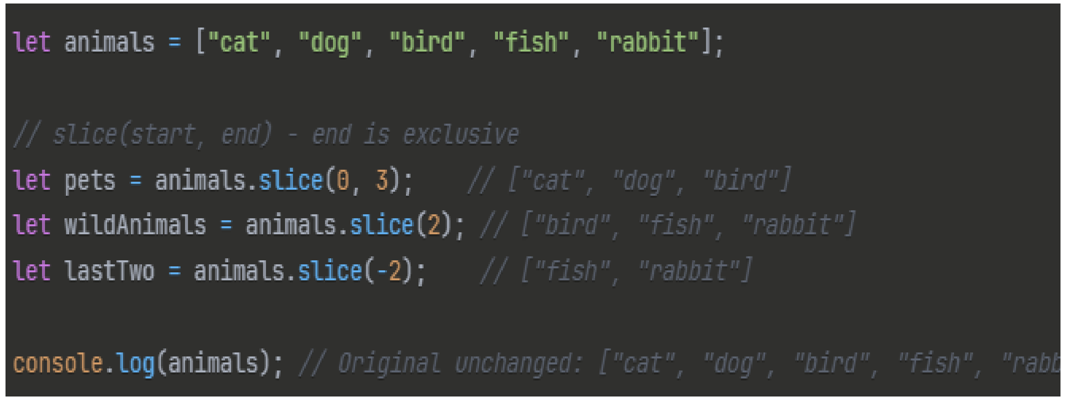
Summary of Basic Operations:



Method	Action	Returns	Performance
`push()`	Add to end	New length	Fast
`pop()`	Remove from end	Removed element	Fast
`unshift()`	Add to beginning	New length	Slow
`shift()`	Remove from beginning	Removed element	Slow

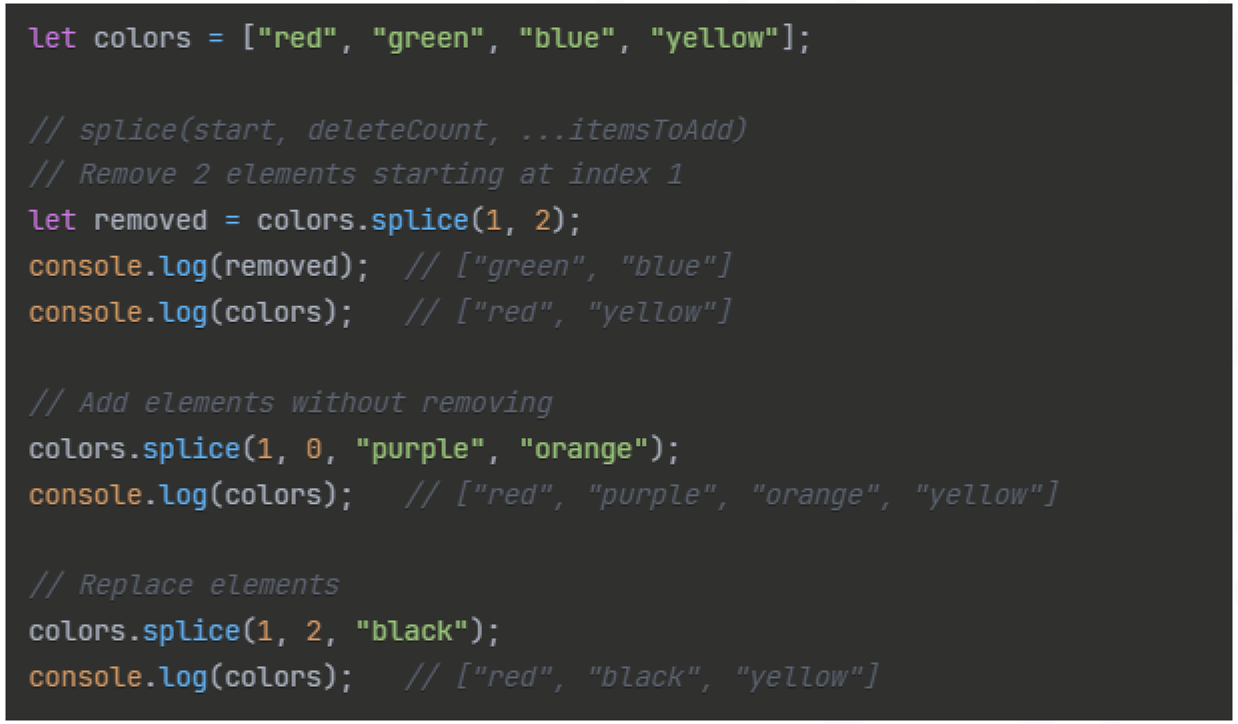
Array Operations - slice() and splice()

slice() - Extract Portion (Non-destructive):



```javascript
let animals = ["cat", "dog", "bird", "fish", "rabbit"];

// slice(start, end) - end is exclusive
let pets = animals.slice(0, 3);    // ["cat", "dog", "bird"]
let wildAnimals = animals.slice(2); // ["bird", "fish", "rabbit"]
let lastTwo = animals.slice(-2);    // ["fish", "rabbit"]

console.log(animals); // Original unchanged: ["cat", "dog", "bird", "fish", "rabb
```

splice() - Add/Remove Elements (Destructive):



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

// splice(start, deleteCount, ...itemsToAdd)
// Remove 2 elements starting at index 1
let removed = colors.splice(1, 2);
console.log(removed);  // ["green", "blue"]
console.log(colors);   // ["red", "yellow"]

// Add elements without removing
colors.splice(1, 0, "purple", "orange");
console.log(colors);   // ["red", "purple", "orange", "yellow"]

// Replace elements
colors.splice(1, 2, "black");
console.log(colors);   // ["red", "black", "yellow"]
```

Array Iteration - forEach()

Basic forEach() Usage:

```javascript
let fruits = ["apple", "banana", "orange"];

// Traditional for loop
for (let i = 0; i < fruits.length; i++) {
    console.log(fruits[i]);
}

// forEach method (cleaner!)
fruits.forEach(function(fruit) {
    console.log(fruit);
});

// forEach with arrow function (even cleaner!)
fruits.forEach(fruit => console.log(fruit));
```

forEach() with Index and Array:

```javascript
let numbers = [10, 20, 30];

numbers.forEach((number, index, array) => {
    console.log(`Index ${index}: ${number}`);
    console.log(`Array length: ${array.length}`);
});

// Output:
// Index 0: 10
// Array length: 3
// Index 1: 20
// Array length: 3
// Index 2: 30
// Array length: 3
```

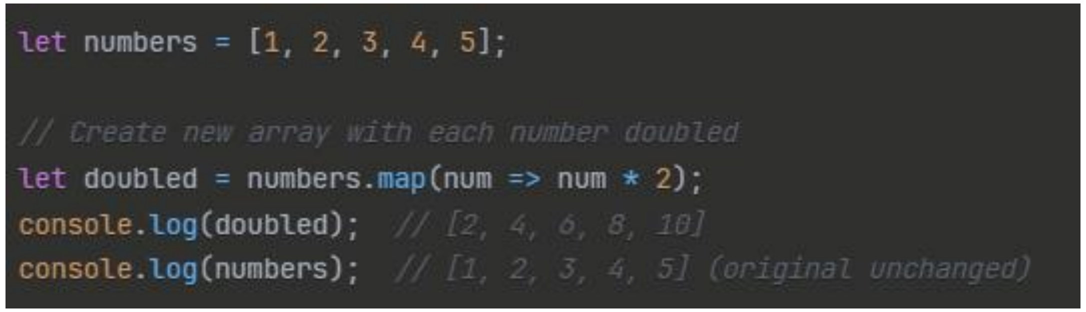
Array Iteration - forEach()

When to Use forEach():

- Simple iteration without creating new array.
- Side effects - logging, updating DOM, etc.
- Cannot break early - runs for all elements

Array Iteration - map()

Basic map() Usage:



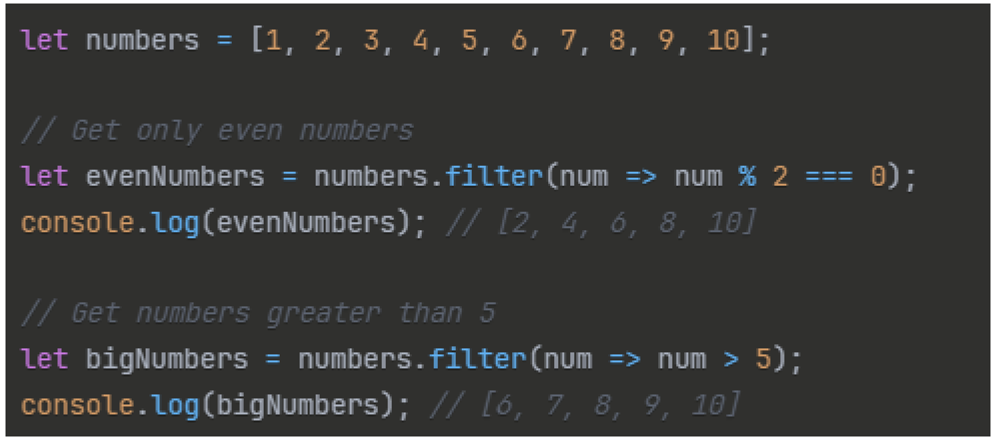
```
let numbers = [1, 2, 3, 4, 5];

// Create new array with each number doubled
let doubled = numbers.map(num => num * 2);
console.log(doubled);  // [2, 4, 6, 8, 10]
console.log(numbers);  // [1, 2, 3, 4, 5] (original unchanged)
```

Key Points:

- Returns new array - same length as original
- Transforms each element - one-to-one mapping
- Pure function - doesn't modify original

Array Iteration - filter()

Basic filter() Usage:



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

// Get only even numbers
let evenNumbers = numbers.filter(num => num % 2 === 0);
console.log(evenNumbers); // [2, 4, 6, 8, 10]

// Get numbers greater than 5
let bigNumbers = numbers.filter(num => num > 5);
console.log(bigNumbers); // [6, 7, 8, 9, 10]
```

Key Points:

- Returns new array - potentially smaller than original
- Returns new array - potentially smaller than original
- Callback returns boolean - true to keep, false to exclude

Array Iteration - reduce()

Basic reduce() Usage:

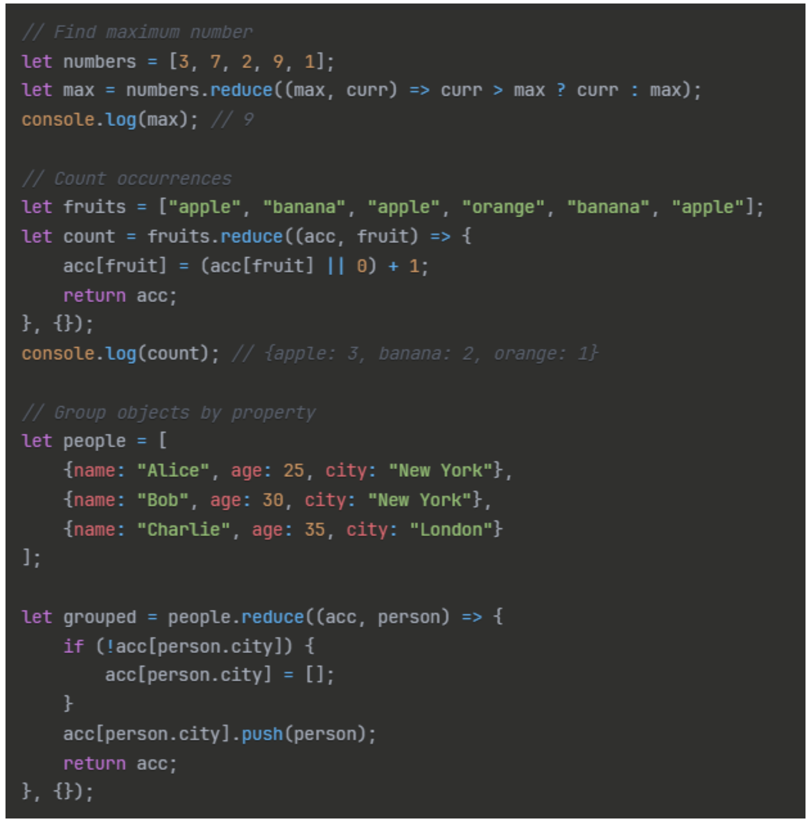
```javascript
let numbers = [1, 2, 3, 4, 5];

// Sum all numbers
let sum = numbers.reduce((accumulator, current) => {
    return accumulator + current;
}, 0); // 0 is the initial value

console.log(sum); // 15

// Shorter version
let sum2 = numbers.reduce((acc, curr) => acc + curr, 0);
```

Understanding reduce():

```javascript
// Step by step breakdown
let numbers = [1, 2, 3, 4];

let sum = numbers.reduce((acc, curr) => {
    console.log(`Acc: ${acc}, Curr: ${curr}, New Acc: ${acc + curr}`);
    return acc + curr;
}, 0);

// Output:
// Acc: 0, Curr: 1, New Acc: 1
// Acc: 1, Curr: 2, New Acc: 3
// Acc: 3, Curr: 3, New Acc: 6
// Acc: 6, Curr: 4, New Acc: 10
// Final result: 10
```

Array Iteration - reduce()

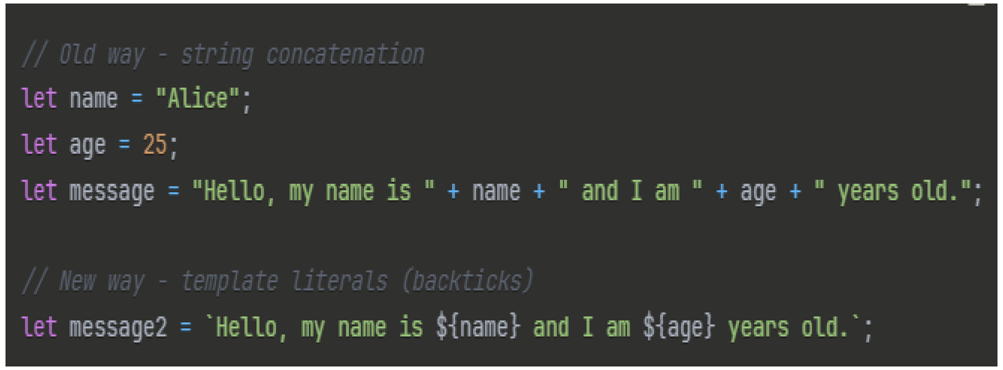
Advanced reduce() Examples:



```
// Find maximum number
let numbers = [3, 7, 2, 9, 1];
let max = numbers.reduce((max, curr) => curr > max ? curr : max);
console.log(max); // 9

// Count occurrences
let fruits = ["apple", "banana", "apple", "orange", "banana", "apple"];
let count = fruits.reduce((acc, fruit) => {
    acc[fruit] = (acc[fruit] || 0) + 1;
    return acc;
}, {});
console.log(count); // {apple: 3, banana: 2, orange: 1}

// Group objects by property
let people = [
    {name: "Alice", age: 25, city: "New York"},
    {name: "Bob", age: 30, city: "New York"},
    {name: "Charlie", age: 35, city: "London"}
];

let grouped = people.reduce((acc, person) => {
    if (!acc[person.city]) {
        acc[person.city] = [];
    }
    acc[person.city].push(person);
    return acc;
}, {});
```

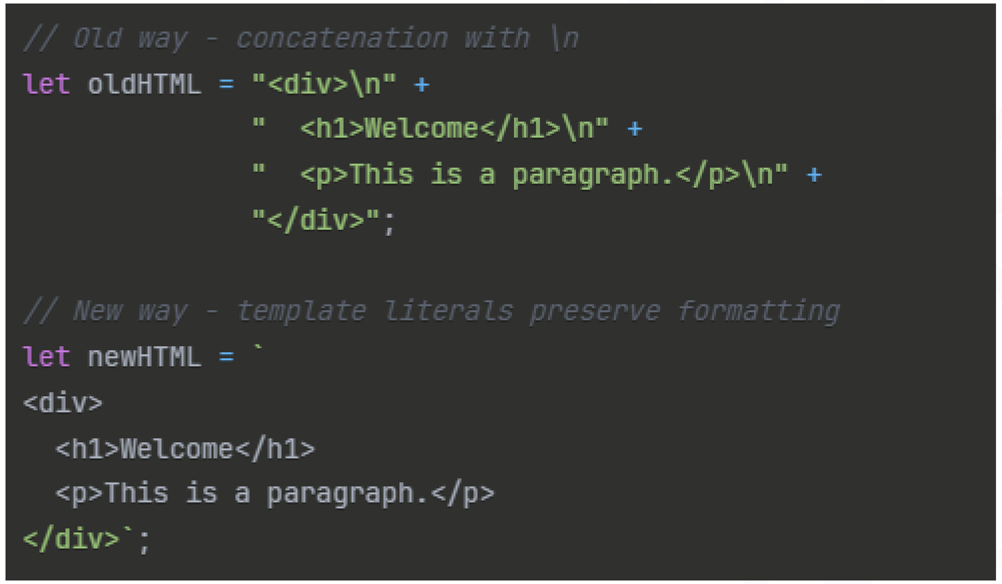
ES6 Feature - Template Literals

Basic Template Literals:



```javascript
// Old way - string concatenation
let name = "Alice";
let age = 25;
let message = "Hello, my name is " + name + " and I am " + age + " years old.";

// New way - template literals (backticks)
let message2 = `Hello, my name is ${name} and I am ${age} years old.`;
```

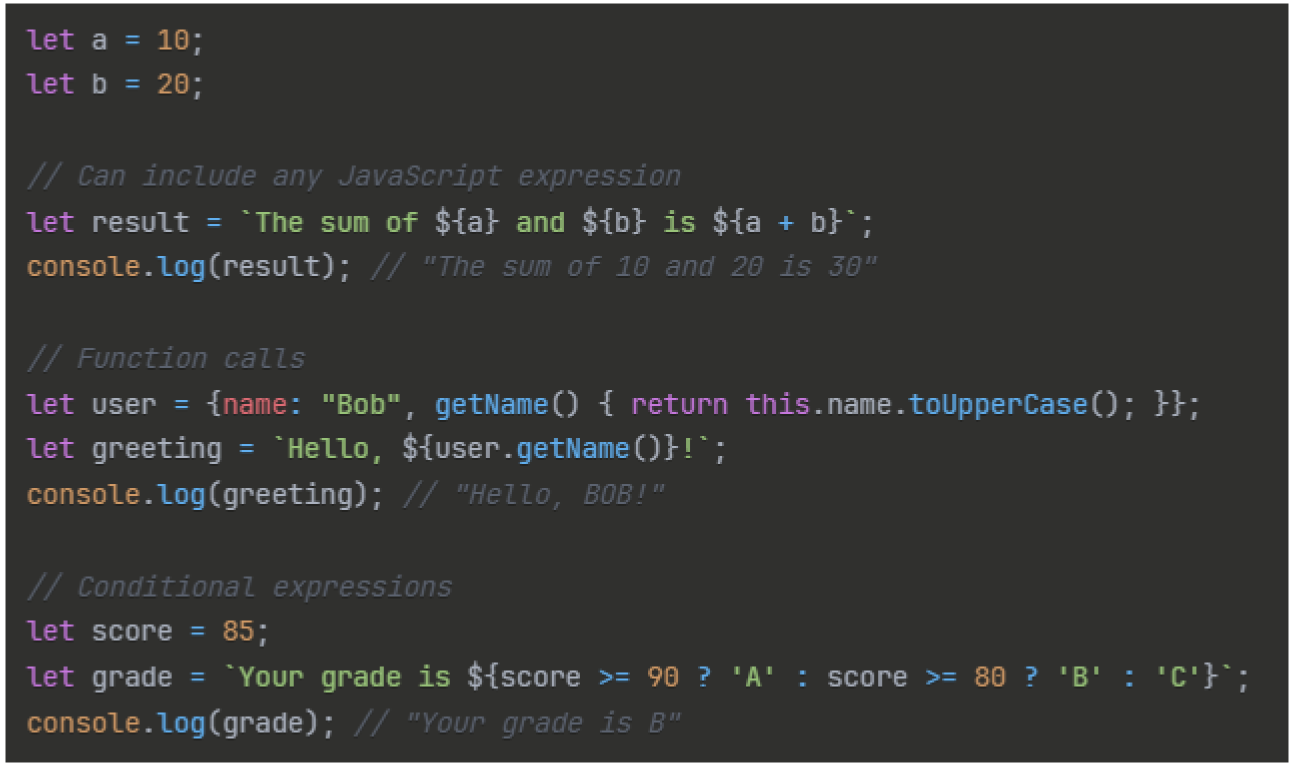
Multi-line Strings:



```javascript
// Old way - concatenation with \n
let oldHTML = "<div>\n" +
              "  <h1>Welcome</h1>\n" +
              "  <p>This is a paragraph.</p>\n" +
              "</div>";

// New way - template literals preserve formatting
let newHTML = `
<div>
  <h1>Welcome</h1>
  <p>This is a paragraph.</p>
</div>`;
```

ES6 Feature - Template Literals

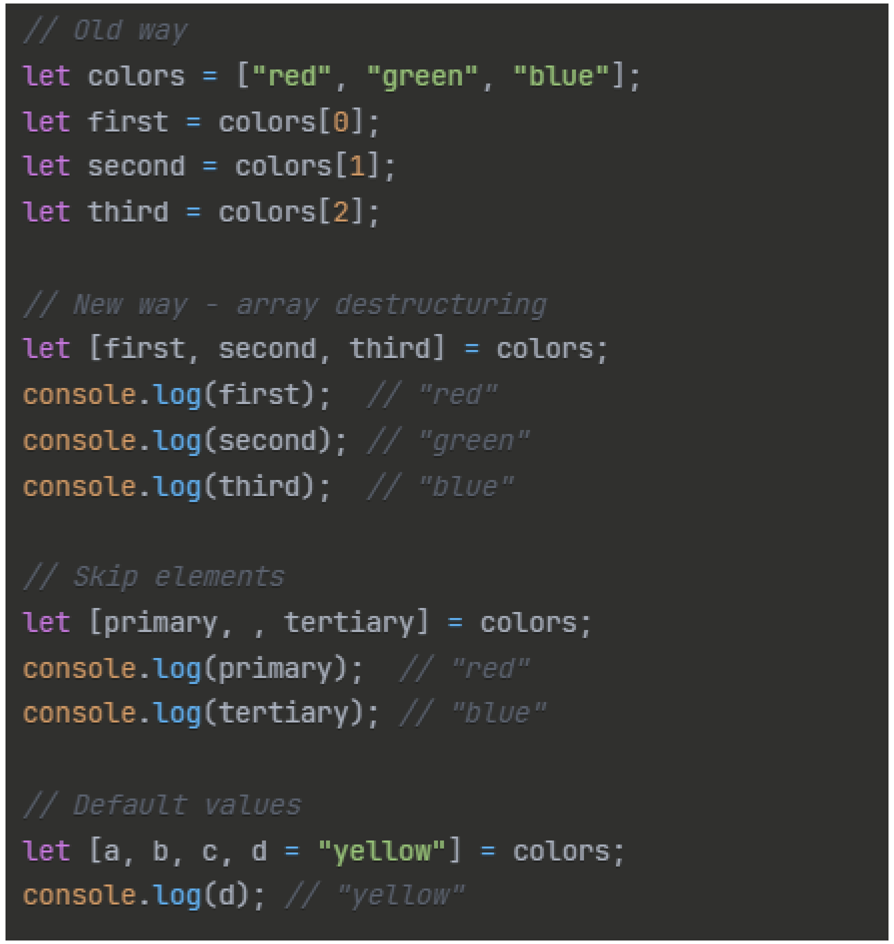
Expression Evaluation:



```javascript
let a = 10;
let b = 20;

// Can include any JavaScript expression
let result = `The sum of ${a} and ${b} is ${a + b}`;
console.log(result); // "The sum of 10 and 20 is 30"

// Function calls
let user = {name: "Bob", getName() { return this.name.toUpperCase(); }};
let greeting = `Hello, ${user.getName()}!`;
console.log(greeting); // "Hello, BOB!"

// Conditional expressions
let score = 85;
let grade = `Your grade is ${score >= 90 ? 'A' : score >= 80 ? 'B' : 'C'}`;
console.log(grade); // "Your grade is B"
```

ES6 Feature - Destructuring

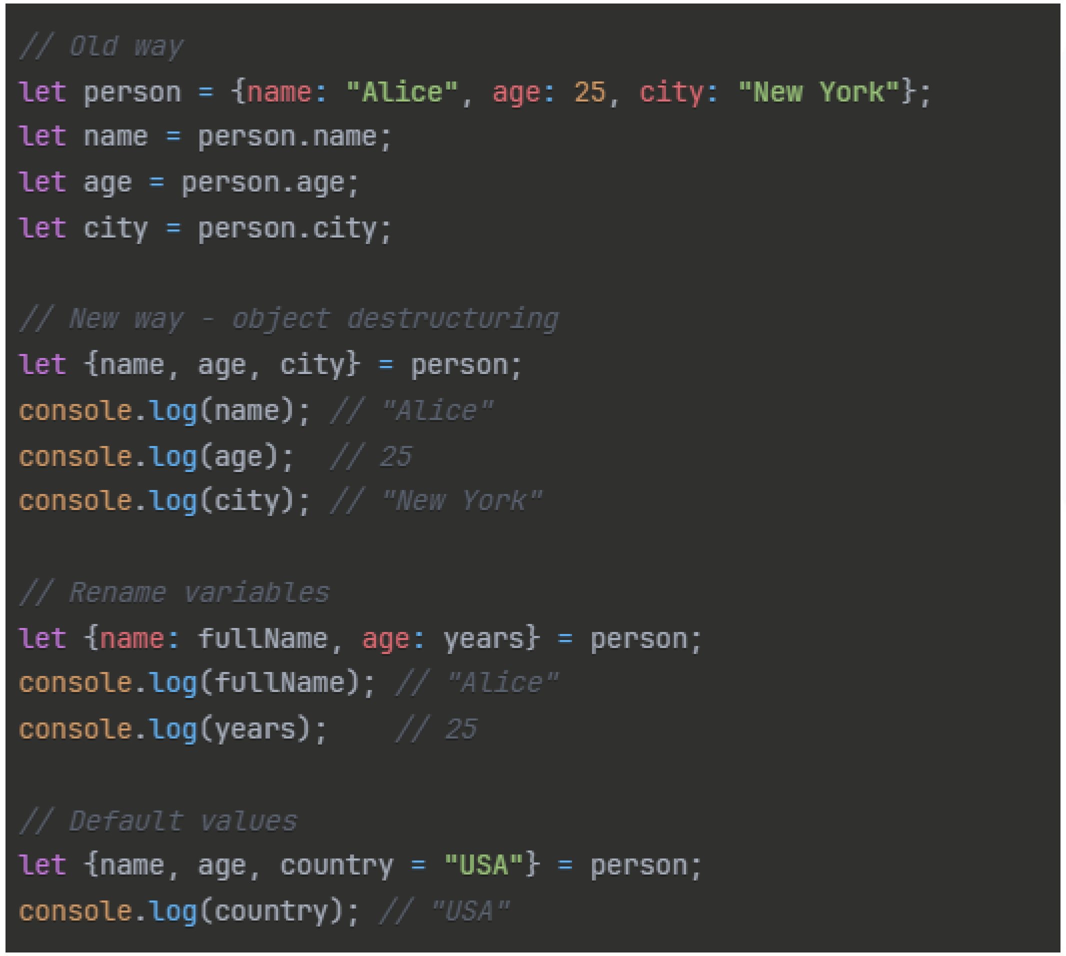
Array Destructuring:



```
// Old way
let colors = ["red", "green", "blue"];
let first = colors[0];
let second = colors[1];
let third = colors[2];

// New way - array destructuring
let [first, second, third] = colors;
console.log(first);  // "red"
console.log(second); // "green"
console.log(third);  // "blue"

// Skip elements
let [primary, , tertiary] = colors;
console.log(primary);  // "red"
console.log(tertiary); // "blue"

// Default values
let [a, b, c, d = "yellow"] = colors;
console.log(d); // "yellow"
```

ES6 Feature - Destructuring

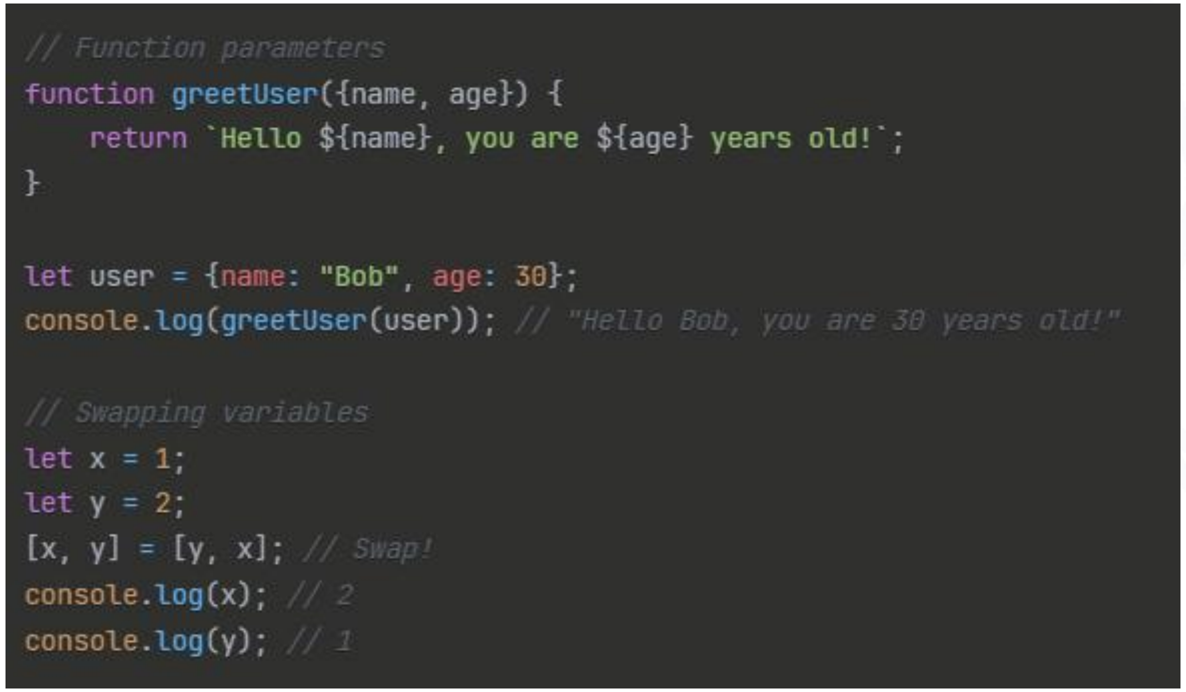
Object Destructuring:



```
// Old way
let person = {name: "Alice", age: 25, city: "New York"};
let name = person.name;
let age = person.age;
let city = person.city;

// New way - object destructuring
let {name, age, city} = person;
console.log(name); // "Alice"
console.log(age);  // 25
console.log(city); // "New York"

// Rename variables
let {name: fullName, age: years} = person;
console.log(fullName); // "Alice"
console.log(years);    // 25

// Default values
let {name, age, country = "USA"} = person;
console.log(country); // "USA"
```

ES6 Feature - Destructuring

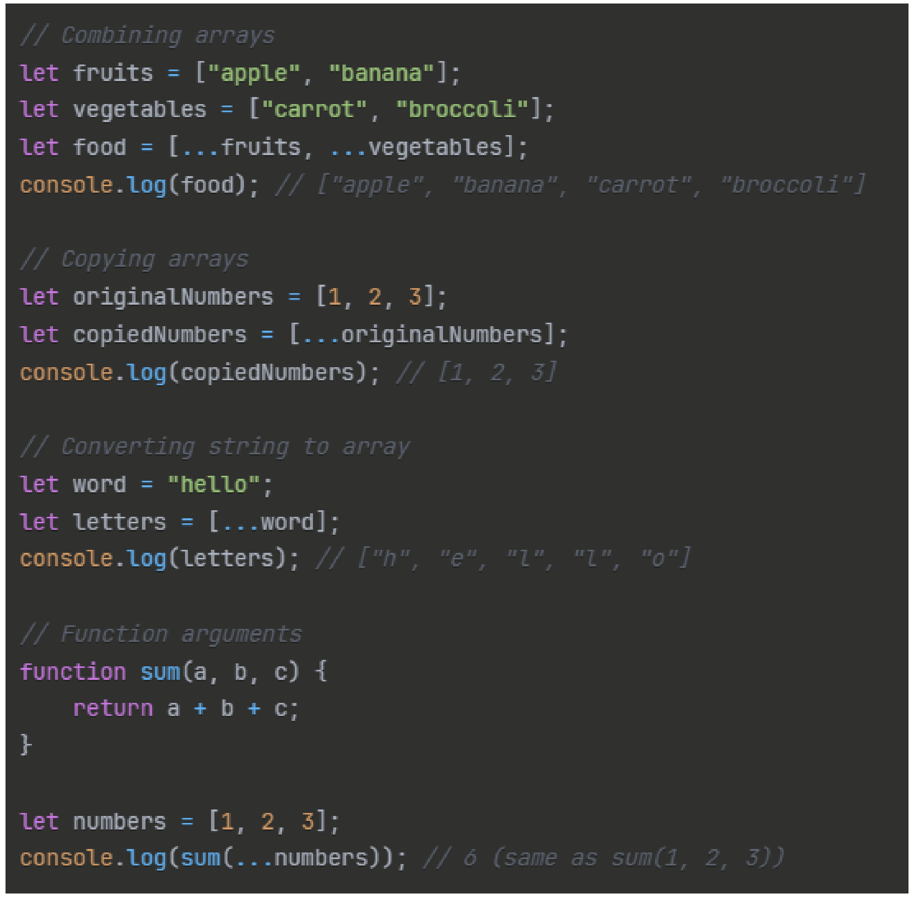
Practical Uses:



```
// Function parameters
function greetUser({name, age}) {
    return `Hello ${name}, you are ${age} years old!`;
}

let user = {name: "Bob", age: 30};
console.log(greetUser(user)); // "Hello Bob, you are 30 years old!"

// Swapping variables
let x = 1;
let y = 2;
[x, y] = [y, x]; // Swap!
console.log(x); // 2
console.log(y); // 1
```

ES6 Feature - Spread & Rest Operators

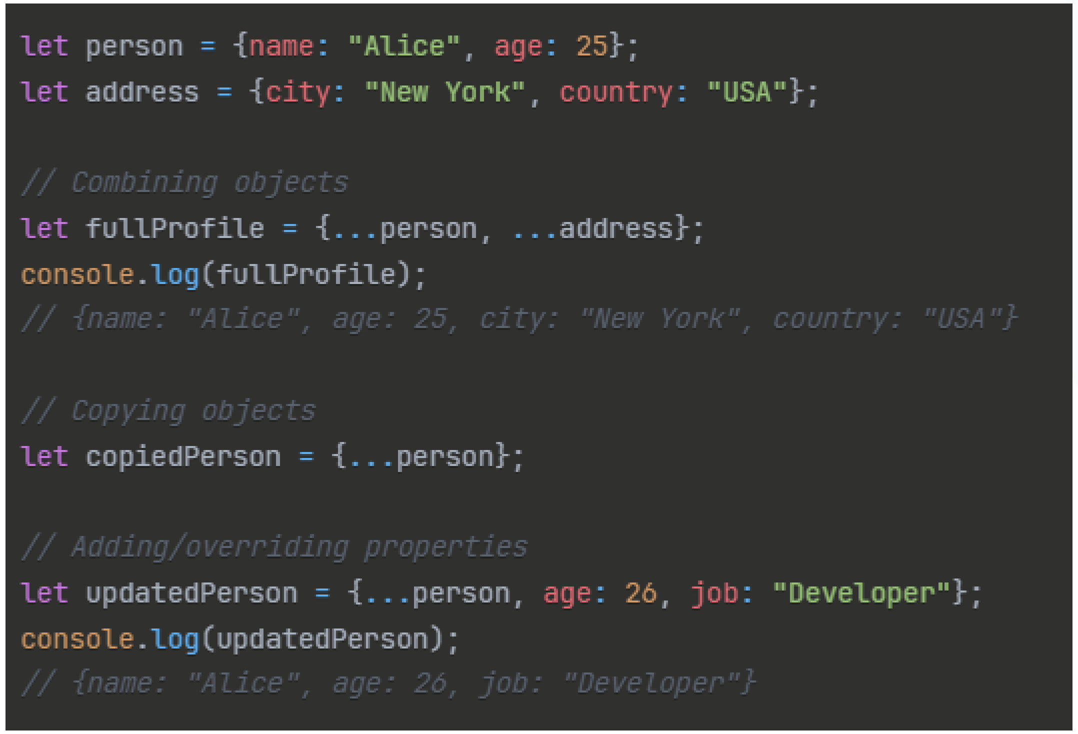
Spread Operator - Arrays:



```
// Combining arrays
let fruits = ["apple", "banana"];
let vegetables = ["carrot", "broccoli"];
let food = [...fruits, ...vegetables];
console.log(food); // ["apple", "banana", "carrot", "broccoli"]

// Copying arrays
let originalNumbers = [1, 2, 3];
let copiedNumbers = [...originalNumbers];
console.log(copiedNumbers); // [1, 2, 3]

// Converting string to array
let word = "hello";
let letters = [...word];
console.log(letters); // ["h", "e", "l", "l", "o"]

// Function arguments
function sum(a, b, c) {
    return a + b + c;
}

let numbers = [1, 2, 3];
console.log(sum(...numbers)); // 6 (same as sum(1, 2, 3))
```

ES6 Feature - Spread & Rest Operators

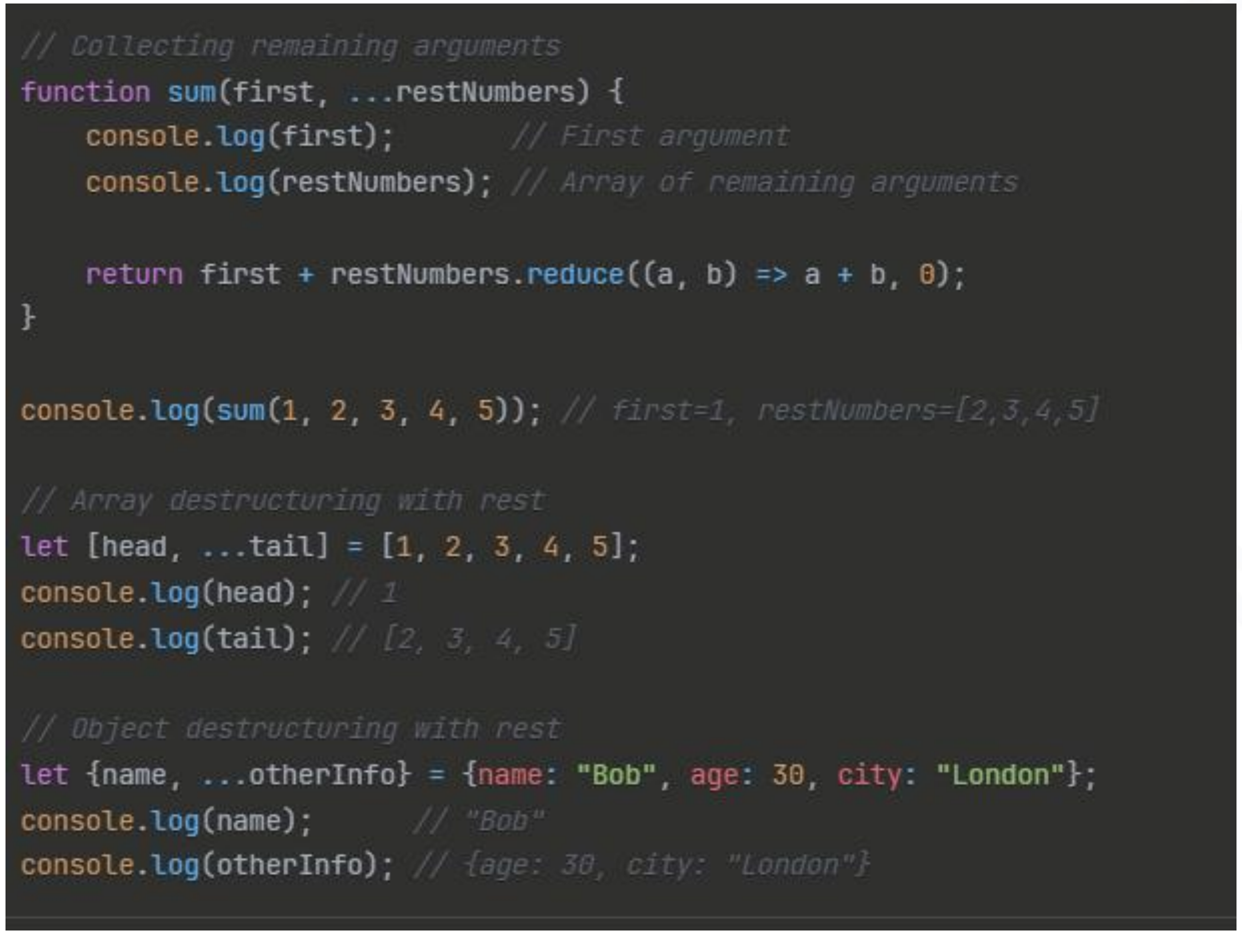
Spread Operator - Objects:



```javascript
let person = {name: "Alice", age: 25};
let address = {city: "New York", country: "USA"};

// Combining objects
let fullProfile = {...person, ...address};
console.log(fullProfile);
// {name: "Alice", age: 25, city: "New York", country: "USA"}

// Copying objects
let copiedPerson = {...person};

// Adding/overriding properties
let updatedPerson = {...person, age: 26, job: "Developer"};
console.log(updatedPerson);
// {name: "Alice", age: 26, job: "Developer"}
```

ES6 Feature - Spread & Rest Operators

Rest Parameters:



```
// Collecting remaining arguments
function sum(first, ...restNumbers) {
    console.log(first);        // First argument
    console.log(restNumbers); // Array of remaining arguments

    return first + restNumbers.reduce((a, b) => a + b, 0);
}

console.log(sum(1, 2, 3, 4, 5)); // first=1, restNumbers=[2,3,4,5]

// Array destructuring with rest
let [head, ...tail] = [1, 2, 3, 4, 5];
console.log(head); // 1
console.log(tail); // [2, 3, 4, 5]

// Object destructuring with rest
let {name, ...otherInfo} = {name: "Bob", age: 30, city: "London"};
console.log(name);       // "Bob"
console.log(otherInfo); // {age: 30, city: "London"}
```

Practice: Student Grade Filter

Your Task:
Given an array of students with grades, extract the names of students who scored above 50

Requirements:
- Use the provided array of student objects
- Filter students with grades above 50
- Extract only the names using map()
- Use ES6 features (arrow functions, destructuring, template literals)
- Log the result with a nice message
- **Bonus:** Also show their grades in the format "Name: Grade"

```
let students = [ {name: "Alice", grade: 85},
{name: "Bob", grade: 45}, {name: "Charlie",
grade: 72}, {name: "Diana", grade: 38}, {name:
"Eve", grade: 91}, {name: "Frank", grade: 55},
{name: "Grace", grade: 42}, {name: "Henry",
grade: 78} ];
```

THANK YOU

ADAEGY