upGrad ENTERPRISE

# JavaScript Advanced : ES6

By
Narasimha Rao T
***Microsoft.Net FSD Trainer***
Professional Development Trainer
tnrao.trainer@gmail.com

# Arrays and Array Methods

## Concept

Arrays are ordered lists of values. They can store multiple types — numbers, strings, objects, even other arrays.

```
const fruits = ["apple", "banana", "cherry"];
```

## Common Methods

| Method | Description | Example |
| --- | --- | --- |
| `push()` | Adds to the end | `fruits.push("mango")` |
| `pop()` | Removes from the end | `fruits.pop()` |
| `shift()` | Removes first element | `fruits.shift()` |
| `unshift()` | Adds to the start | `fruits.unshift("kiwi")` |
| `map()` | Transforms each element | `fruits.map(f => f.toUpperCase())` |
| `filter()` | Keeps items that match a condition | `fruits.filter(f => f.includes("a"))` |

## Example (Usage of map method)

```
const numbers = [1, 2, 3, 4, 5];
const squared = numbers.map(n => n * n);
console.log(squared); // [1,4,9,16,25]
```

# Object Manipulation

## Concept

Objects store key–value pairs.

```
const user = { name: "Alice", age: 25, city: "Paris" };
```

## Access, Modify, Add, Remove properties

```
console.log(user.name); // Alice
user.age = 26;          // Update value
user.country = "France"; // Add property
delete user.city;       // Remove property
```

## Iterate over Objects

```
for (let key in user) {
  console.log(`${key}: ${user[key]}`);
}
```

## Useful Methods

```
Object.keys(user);   // ["name", "age", "country"]
Object.values(user); // ["Alice", 26, "France"]
Object.entries(user); // [["name","Alice"],["age",26]]
```

# Loops

## Types of Loops

| Loop | Description | Example |
|---|---|---|
| `for` | Classic loop with counter | `for (let i=0; i<5; i++) {}` |
| `for...of` | Iterates over values (arrays, strings) | `for (let fruit of fruits)` |
| `for...in` | Iterates over keys (objects) | `for (let key in user)` |
| `while` | Runs while condition true | `while (count < 5)` |
| `do...while` | Runs once before checking | `do { ... } while()` |

## Example

```
const fruits = ["apple", "banana", "cherry"];
for (const fruit of fruits) {
  console.log(fruit);
}
```

# Destructuring

## Arrays

```
const colors = ["red", "green", "blue"];
const [first, second] = colors;
console.log(first); // red
```

## Objects

```
const user = { name: "Alice", age: 25 };
const { name, age } = user;
console.log(name); // Alice
```

# Spread and Rest Operators

## Spread (`...`)

Expands arrays or objects.

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

const user = { name: "Bob" };
const details = { age: 30 };
const person = { ...user, ...details };
```

# Template Literals

## Concept

Allows embedded expressions using backticks `` ` ``.

```
const name = "Alice";
console.log(`Hello, ${name}!`);
```

File Name : **Multiline Support**

```
const message = `
Dear ${name},
Welcome to JavaScript learning!
`;
```

# Let and const

## `let`

Block-scoped, can be reassigned.

```
let count = 0;
count = 5;
```

## `const`

Block-scoped, cannot be reassigned (but object contents can change).

```
const user = { name: "Tom" };
user.name = "Jerry"; // File Name :  Allowed
// user = {}   Not allowed
```

## Best Practice

- Use `const` by default.
- Use `let` only when reassignment is needed.

# Arrow Functions

Arrow functions in JavaScript, introduced in ES6, provide a concise syntax for writing function expressions.
They are particularly useful for anonymous functions

JavaScript

```javascript
// Basic arrow function
const add = (a, b) => {
  return a + b;
};

// Concise arrow function with implicit return (for single expressions)
const multiply = (a, b) => a * b;

// Arrow function with no parameters
const greet = () => console.log("Hello!");

// Arrow function with a single parameter (parentheses optional)
const square = num => num * num;
```

# Mini Practice Challenge

**Task:**

Create an array of people objects (name, age).

1. Use `filter()` to get adults.
2. Use `map()` to get their names.
3. Use `template literals` to print a sentence.
4. Use `destructuring` to access properties.

```
const people = [
  { name: "Alice", age: 17 },
  { name: "Bob", age: 25 },
  { name: "Carol", age: 19 }
];

const adults = people.filter(p => p.age >= 18);
const names = adults.map(({ name }) => name);
console.log(`Adults: ${names.join(", ")}`);
```

upGrad ENTERPRISE

# Q & A

Narasimha Rao T
tnrao.trainer@gmail.com