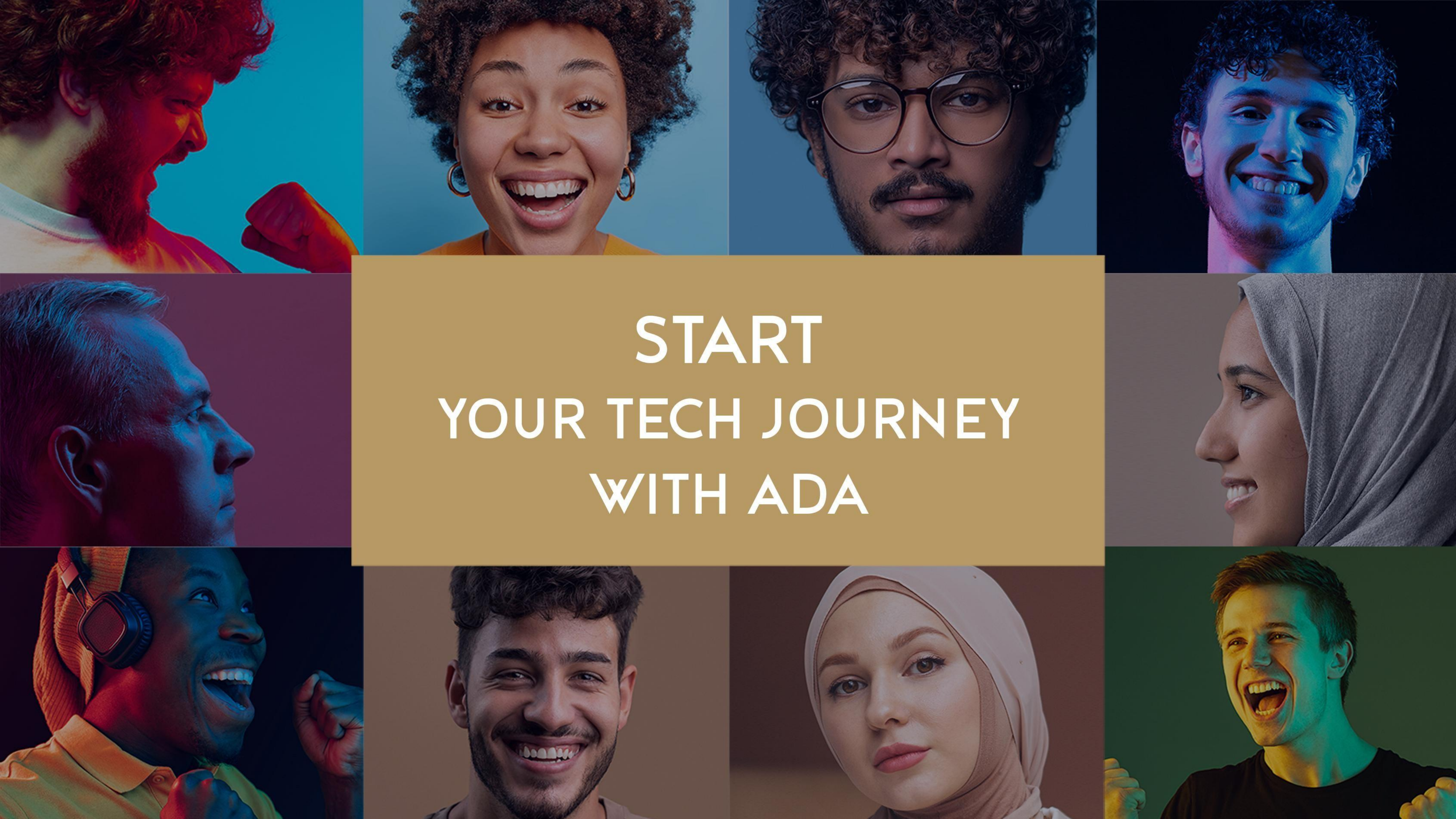
ACADEMY OF DIGITAL ARTS

EGYPT

Google cisco Adobe IBM Microsoft CompTIA



START YOUR TECH JOURNEY WITH ADA

Agenda

- Objects: properties, methods, and the **this** keyword
- JSON basics - working with data interchange format
- Error handling: try/catch, throw statements
- Modules in JavaScript: import/export, require
- Creating reusable and maintainable code
- Best practices for error management
- Hands-on exercise: Create a Book object with error handling

Introduction to Objects

What Are Objects?

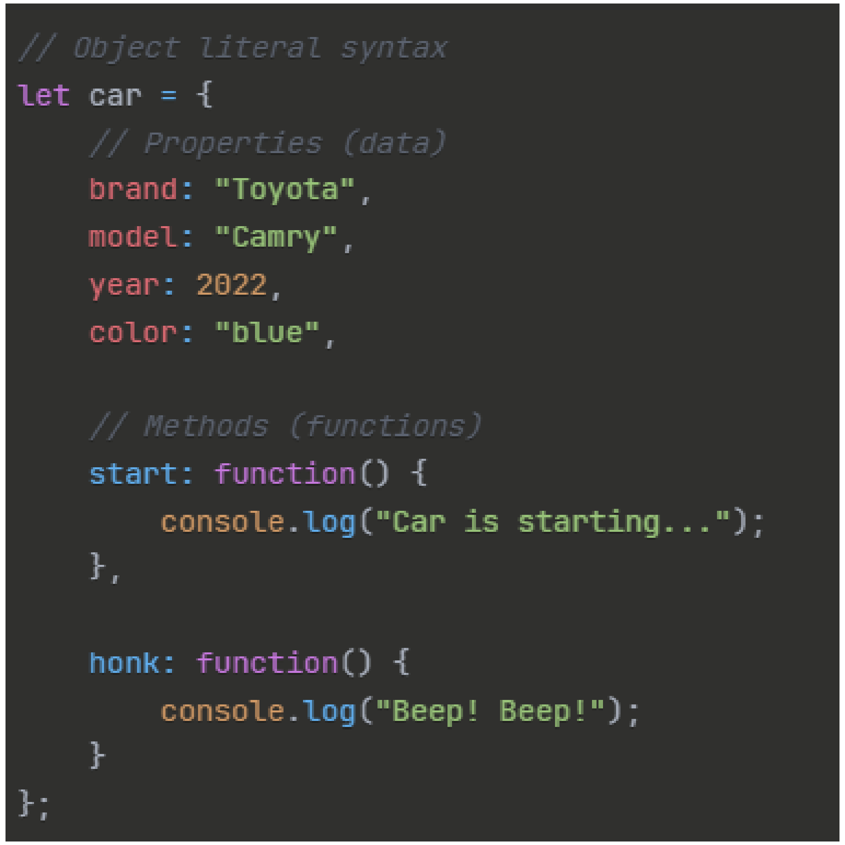
Real-World Analogy:

Think of objects like real-world entities.

- A car has properties (color, model, year) and methods (start, stop, honk)
- A person has properties (name, age, height) and methods (walk, talk, eat)

Introduction to Objects

JavaScript Objects:



```
// Object literal syntax
let car = {
    // Properties (data)
    brand: "Toyota",
    model: "Camry",
    year: 2022,
    color: "blue",

    // Methods (functions)
    start: function() {
        console.log("Car is starting...");
    },

    honk: function() {
        console.log("Beep! Beep!");
    }
};
```

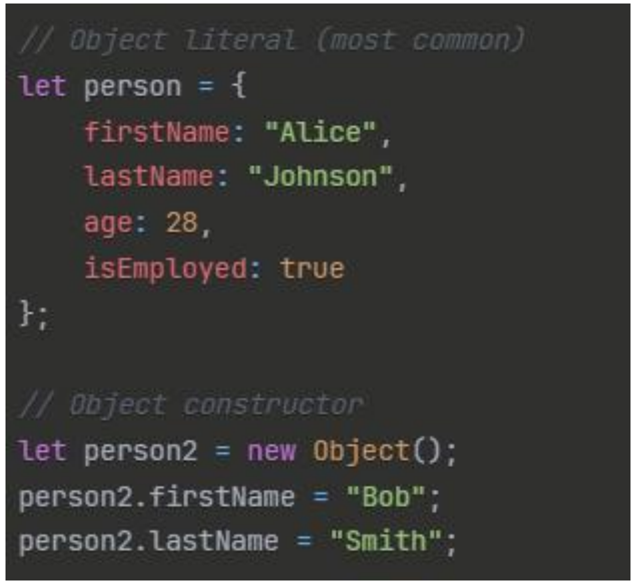
Introduction to Objects

Why Objects?

- Group related data together
- Model real-world entities
- Organize code logically
- Encapsulate data and behavior

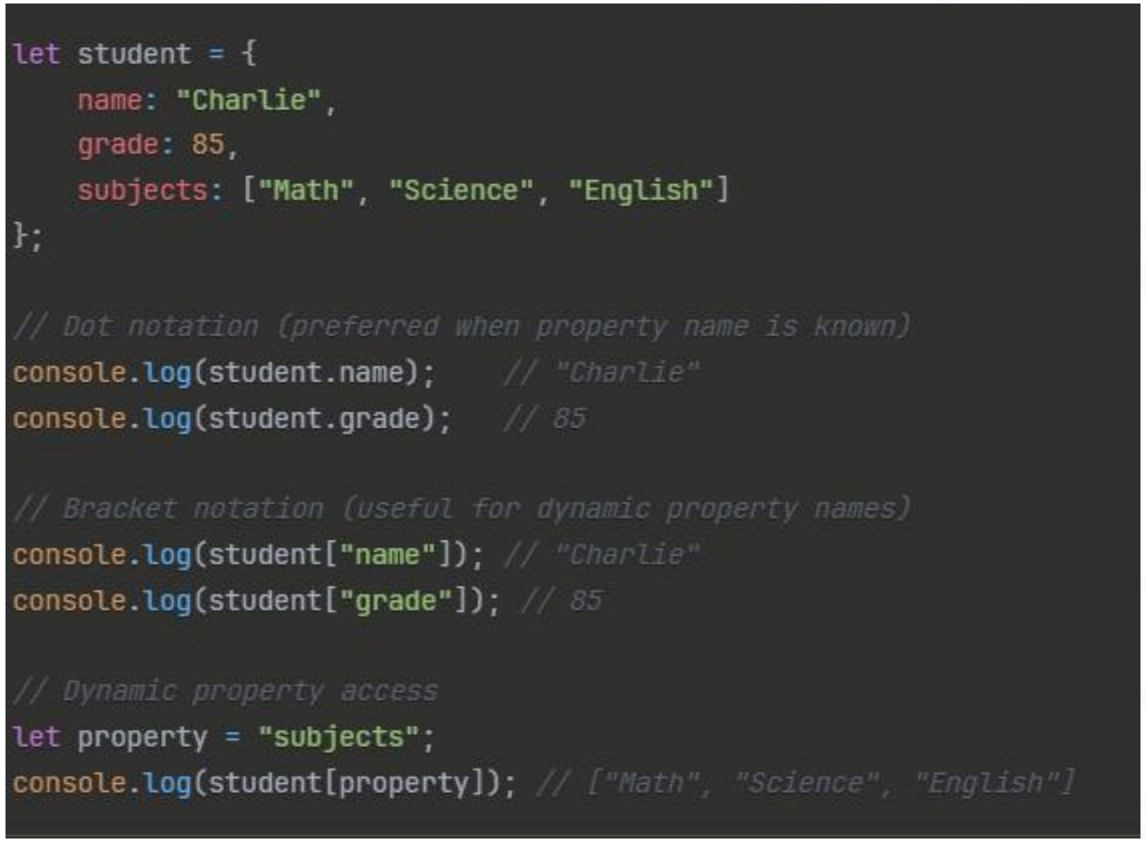
Object Properties

Creating Objects:



```
// Object literal (most common)
let person = {
    firstName: "Alice",
    lastName: "Johnson",
    age: 28,
    isEmployed: true
};

// Object constructor
let person2 = new Object();
person2.firstName = "Bob";
person2.lastName = "Smith";
```

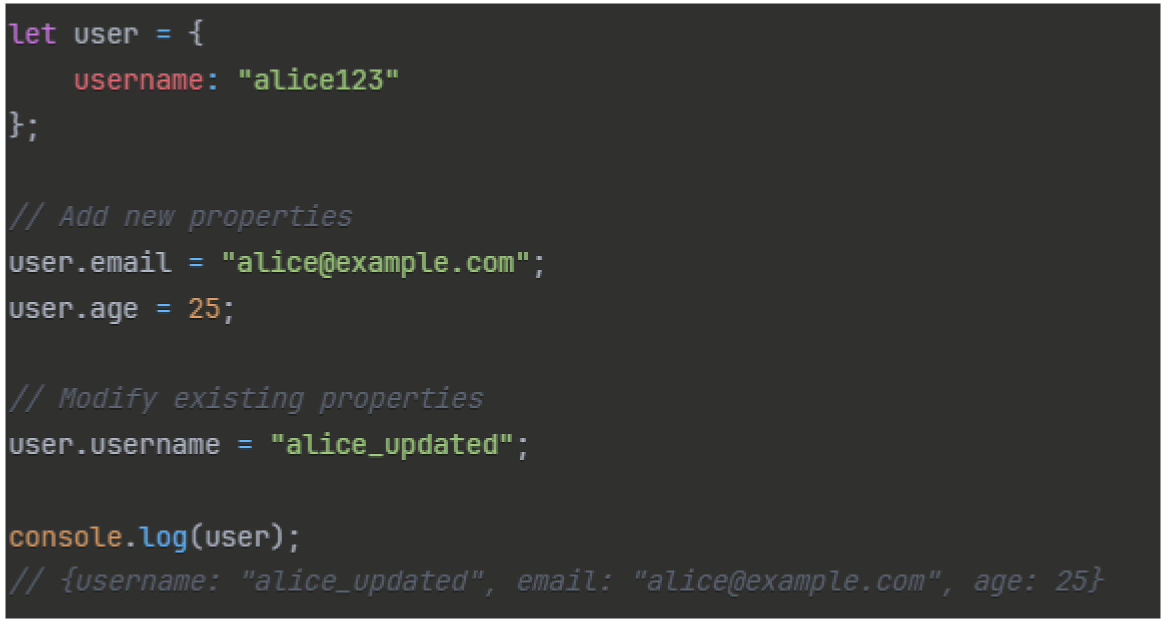
Accessing Properties:



```
let student = {
    name: "Charlie",
    grade: 85,
    subjects: ["Math", "Science", "English"]
};

// Dot notation (preferred when property name is known)
console.log(student.name);    // "Charlie"
console.log(student.grade);   // 85

// Bracket notation (useful for dynamic property names)
console.log(student["name"]); // "Charlie"
console.log(student["grade"]); // 85

// Dynamic property access
let property = "subjects";
console.log(student[property]); // ["Math", "Science", "English"]
```

Object Properties

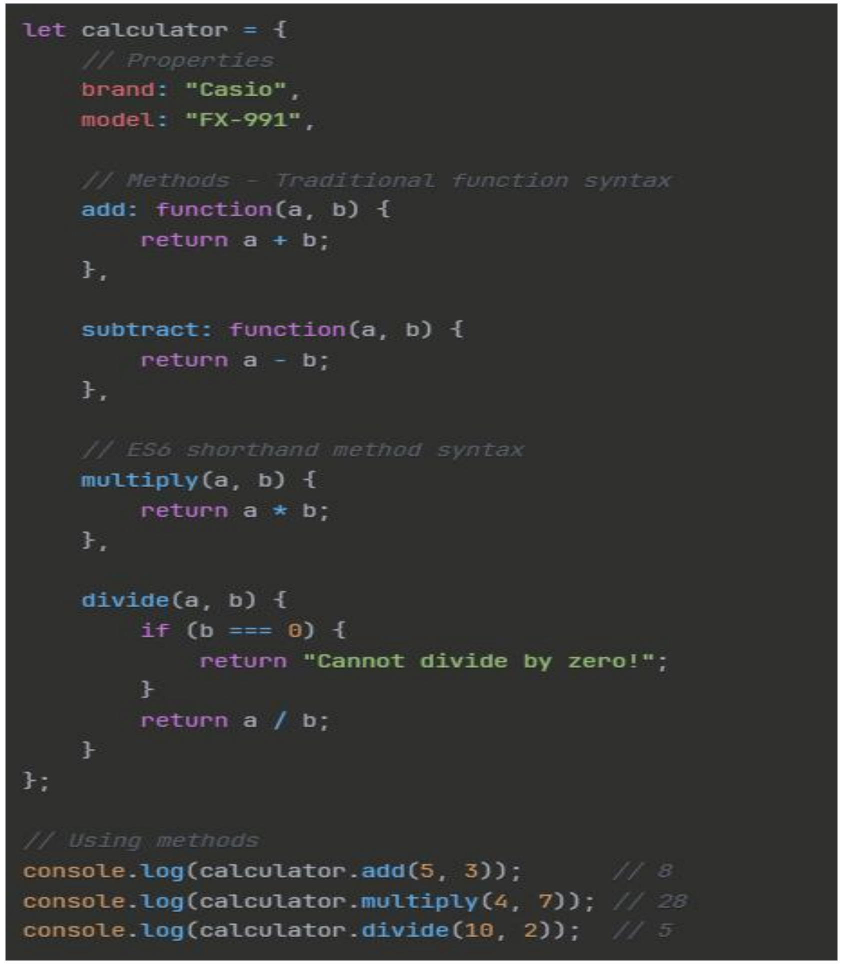
Adding/Modifying Properties: Objects:



```
let user = {
    username: "alice123"
};

// Add new properties
user.email = "alice@example.com";
user.age = 25;

// Modify existing properties
user.username = "alice_updated";

console.log(user);
// {username: "alice_updated", email: "alice@example.com", age: 25}
```

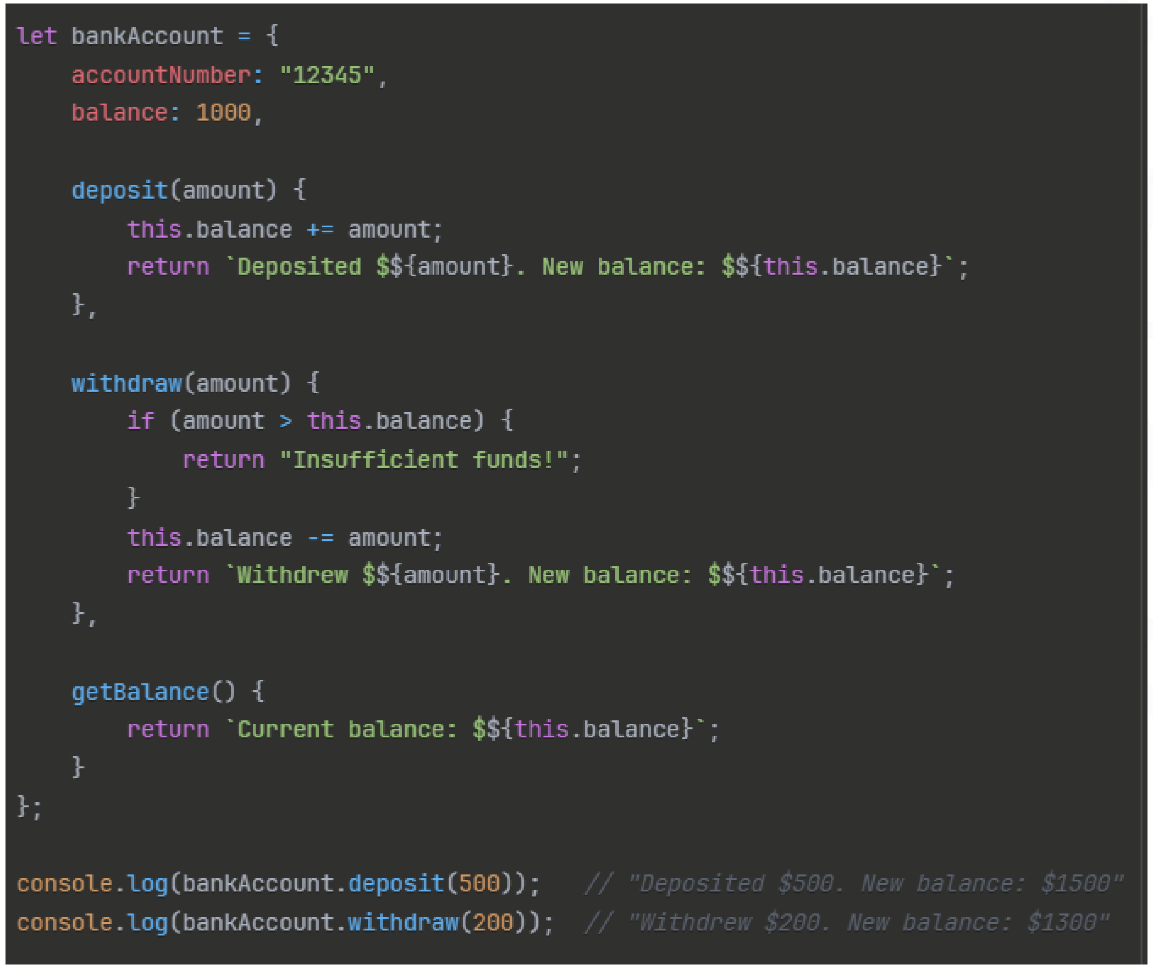
Object Methods

Defining Methods:


```javascript
let calculator = {
    // Properties
    brand: "Casio",
    model: "FX-991",

    // Methods - Traditional function syntax
    add: function(a, b) {
        return a + b;
    },

    subtract: function(a, b) {
        return a - b;
    },

    // ES6 shorthand method syntax
    multiply(a, b) {
        return a * b;
    },

    divide(a, b) {
        if (b === 0) {
            return "Cannot divide by zero!";
        }
        return a / b;
    }
};

// Using methods
console.log(calculator.add(5, 3));      // 8
console.log(calculator.multiply(4, 7)); // 28
console.log(calculator.divide(10, 2));  // 5
```

Object Methods

Methods with Object Properties:



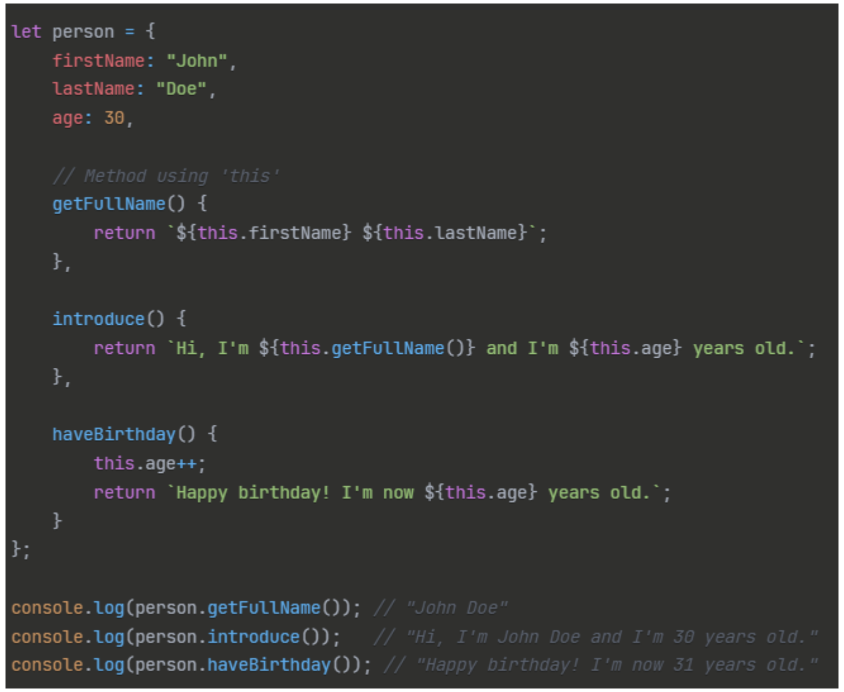
```
let bankAccount = {
    accountNumber: "12345",
    balance: 1000,

    deposit(amount) {
        this.balance += amount;
        return `Deposited $${amount}. New balance: $${this.balance}`;
    },

    withdraw(amount) {
        if (amount > this.balance) {
            return "Insufficient funds!";
        }
        this.balance -= amount;
        return `Withdrew $${amount}. New balance: $${this.balance}`;
    },

    getBalance() {
        return `Current balance: $${this.balance}`;
    }
};

console.log(bankAccount.deposit(500));   // "Deposited $500. New balance: $1500"
console.log(bankAccount.withdraw(200));  // "Withdrew $200. New balance: $1300"
```

The 'this' Keyword

What is 'this’?

‘this’ refers to the current object the method is being called on.

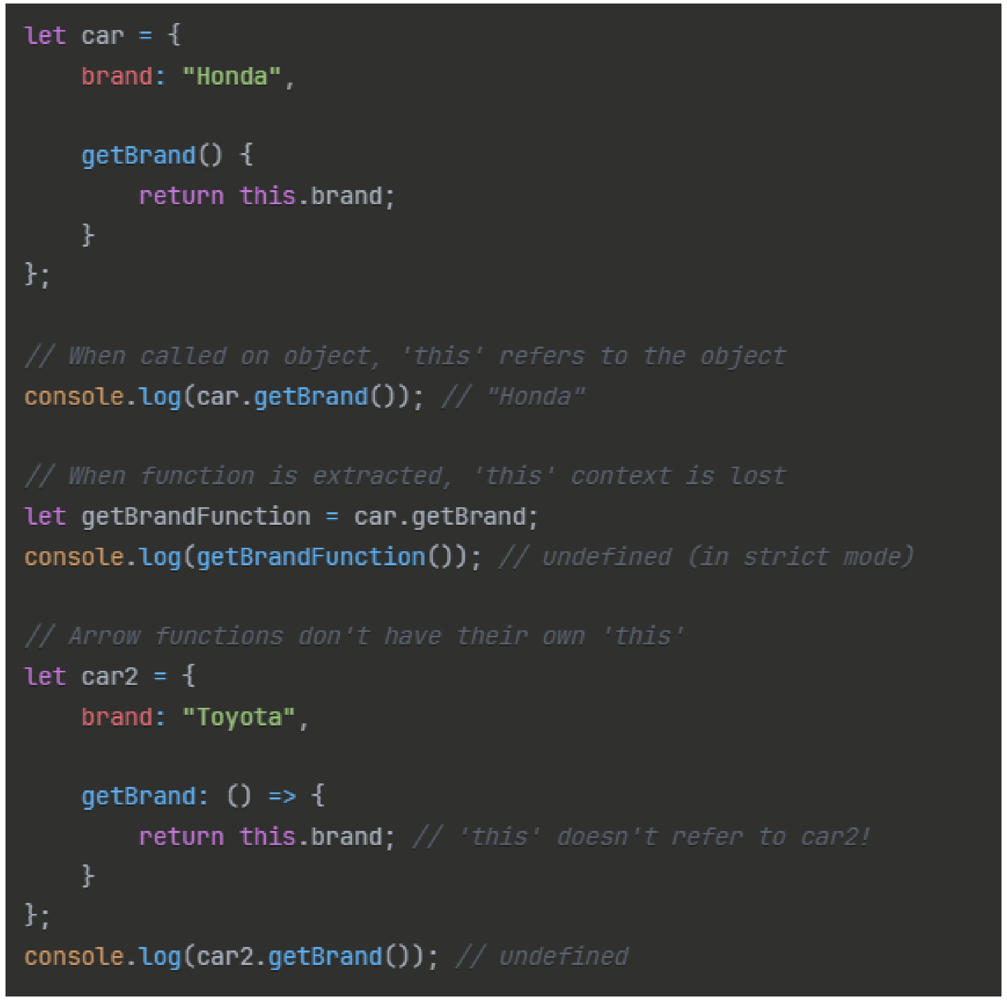
Basic 'this' Usage:



```
let person = {
    firstName: "John",
    lastName: "Doe",
    age: 30,

    // Method using 'this'
    getFullName() {
        return `${this.firstName} ${this.lastName}`;
    },

    introduce() {
        return `Hi, I'm ${this.getFullName()} and I'm ${this.age} years old.`;
    },

    haveBirthday() {
        this.age++;
        return `Happy birthday! I'm now ${this.age} years old.`;
    }
};

console.log(person.getFullName()); // "John Doe"
console.log(person.introduce());   // "Hi, I'm John Doe and I'm 30 years old."
console.log(person.haveBirthday()); // "Happy birthday! I'm now 31 years old."
```

The 'this' Keyword

'this' Context Rules:



```javascript
let car = {
    brand: "Honda",

    getBrand() {
        return this.brand;
    }
};

// When called on object, 'this' refers to the object
console.log(car.getBrand()); // "Honda"

// When function is extracted, 'this' context is lost
let getBrandFunction = car.getBrand;
console.log(getBrandFunction()); // undefined (in strict mode)

// Arrow functions don't have their own 'this'
let car2 = {
    brand: "Toyota",

    getBrand: () => {
        return this.brand; // 'this' doesn't refer to car2!
    }
};
console.log(car2.getBrand()); // undefined
```

The 'this' Keyword

Best Practices:

- Use regular functions for object methods
- Use arrow functions for callbacks inside methods
- Be careful when passing methods as callbacks

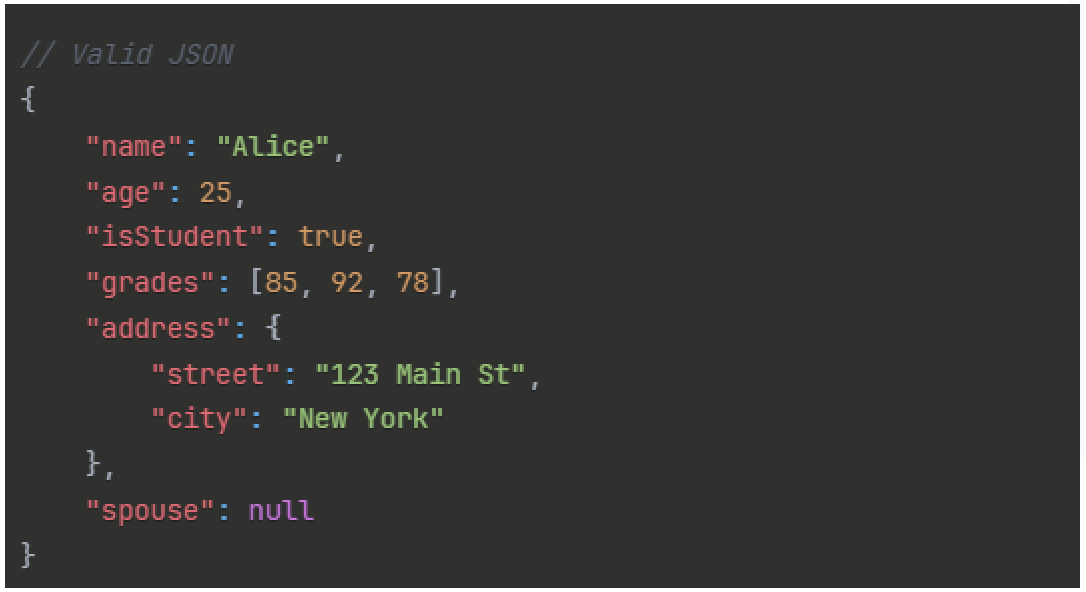
JSON Basics

What is JSON?

- **J**avaScript **O**bject **N**otation
- Text-based data interchange format
- Language independent (used by many programming languages)
- Lightweight alternative to XML

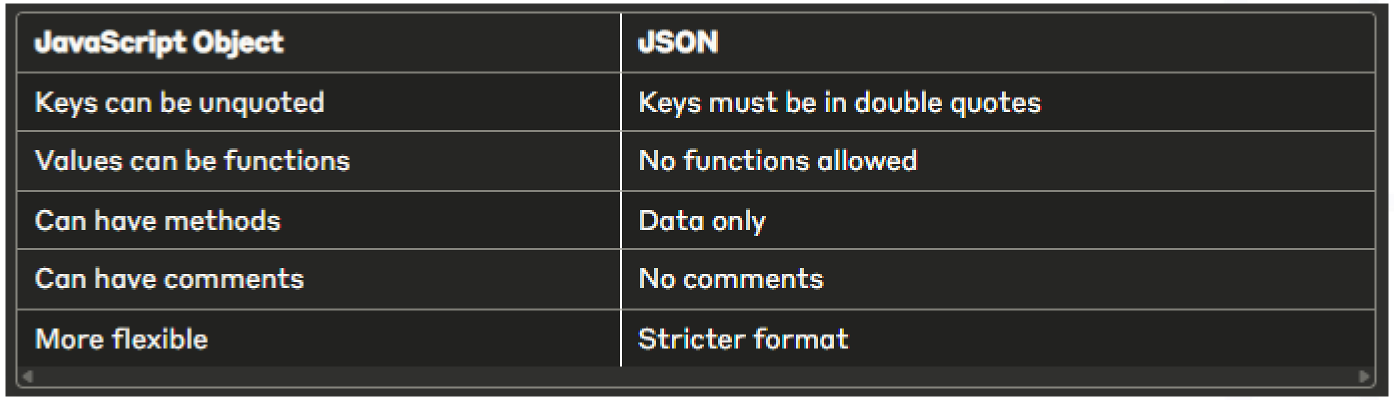
JSON Basics

JSON Syntax Rules:


```
// Valid JSON
{
    "name": "Alice",
    "age": 25,
    "isStudent": true,
    "grades": [85, 92, 78],
    "address": {
        "street": "123 Main St",
        "city": "New York"
    },
    "spouse": null
}
```

JSON Basics

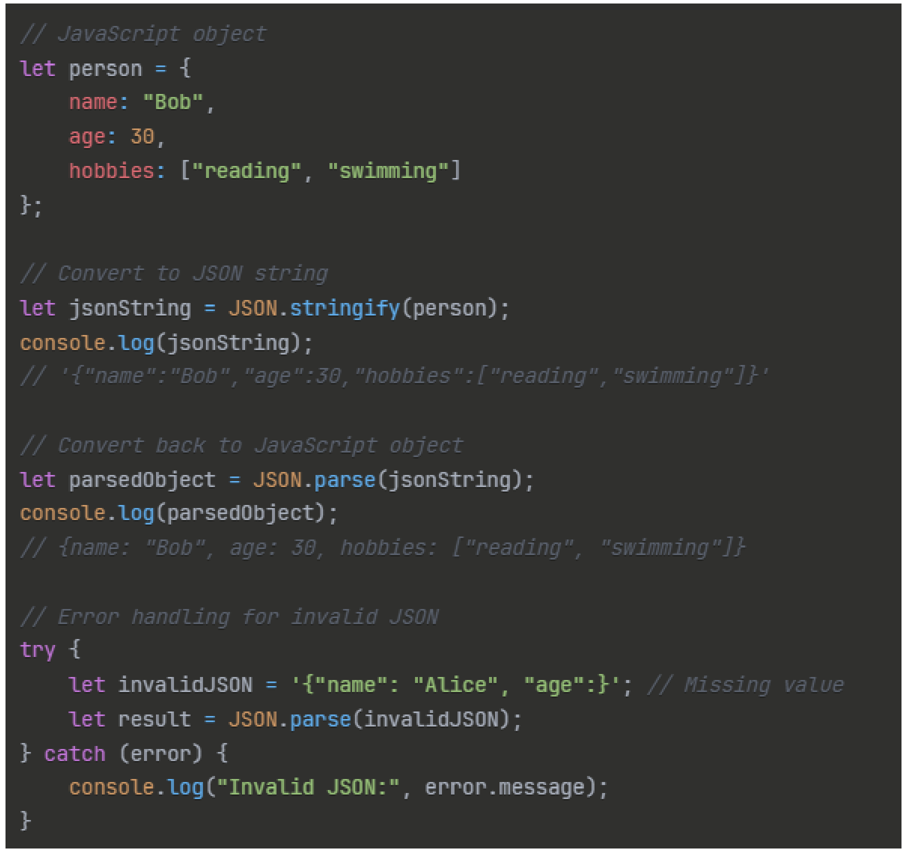
JSON vs JavaScript Object:



JavaScript Object	JSON
Keys can be unquoted	Keys must be in double quotes
Values can be functions	No functions allowed
Can have methods	Data only
Can have comments	No comments
More flexible	Stricter format

JSON Basics

Converting Between JSON and Objects:



```
// JavaScript object
let person = {
    name: "Bob",
    age: 30,
    hobbies: ["reading", "swimming"]
};

// Convert to JSON string
let jsonString = JSON.stringify(person);
console.log(jsonString);
// '{"name":"Bob","age":30,"hobbies":["reading","swimming"]}'

// Convert back to JavaScript object
let parsedObject = JSON.parse(jsonString);
console.log(parsedObject);
// {name: "Bob", age: 30, hobbies: ["reading", "swimming"]}

// Error handling for invalid JSON
try {
    let invalidJSON = '{"name": "Alice", "age":}'; // Missing value
    let result = JSON.parse(invalidJSON);
} catch (error) {
    console.log("Invalid JSON:", error.message);
}
```

Error Handling - try/catch

Why Error Handling?

- Prevent crashes - Keep your program running
- User experience - Show helpful error messages
- Debugging - Log errors for troubleshooting
- Data validation - Handle invalid input

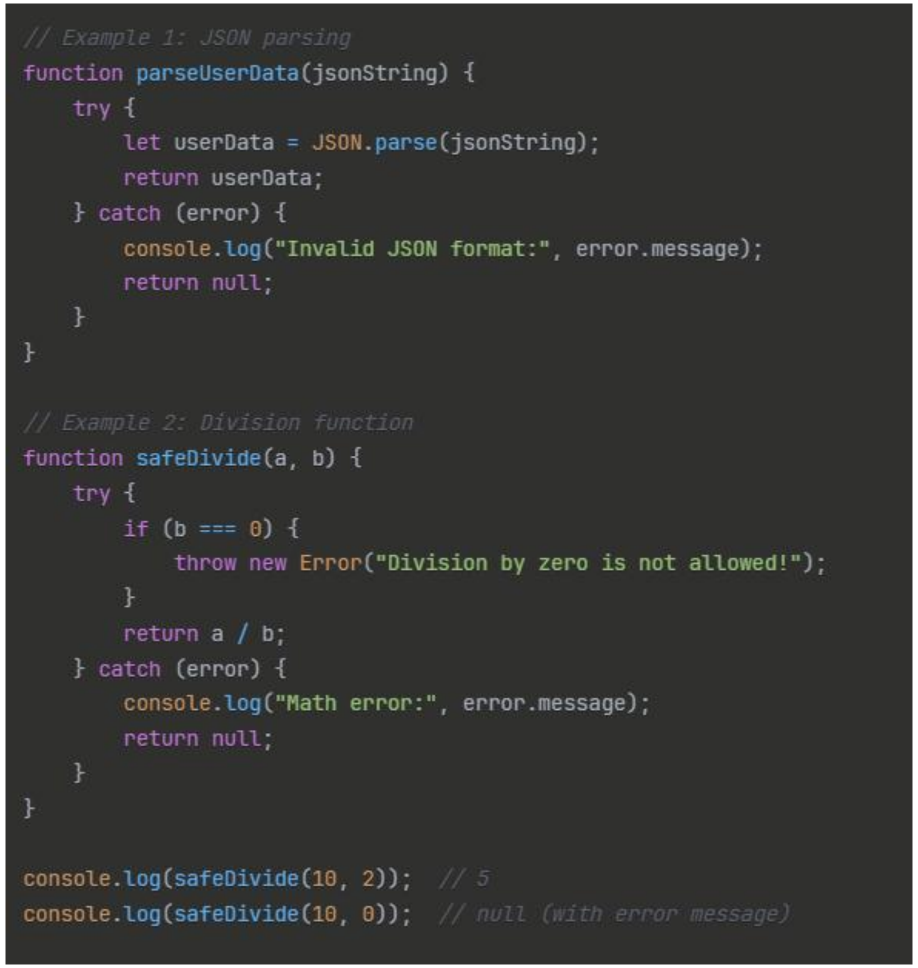
Error Handling - try/catch

Basic try/catch Syntax:

```javascript
try {
    // Code that might throw an error
    let result = riskyOperation();
    console.log("Success:", result);
} catch (error) {
    // Handle the error
    console.log("Error occurred:", error.message);
} finally {
    // This always runs (optional)
    console.log("Cleanup code here");
}
```

Error Handling - try/catch

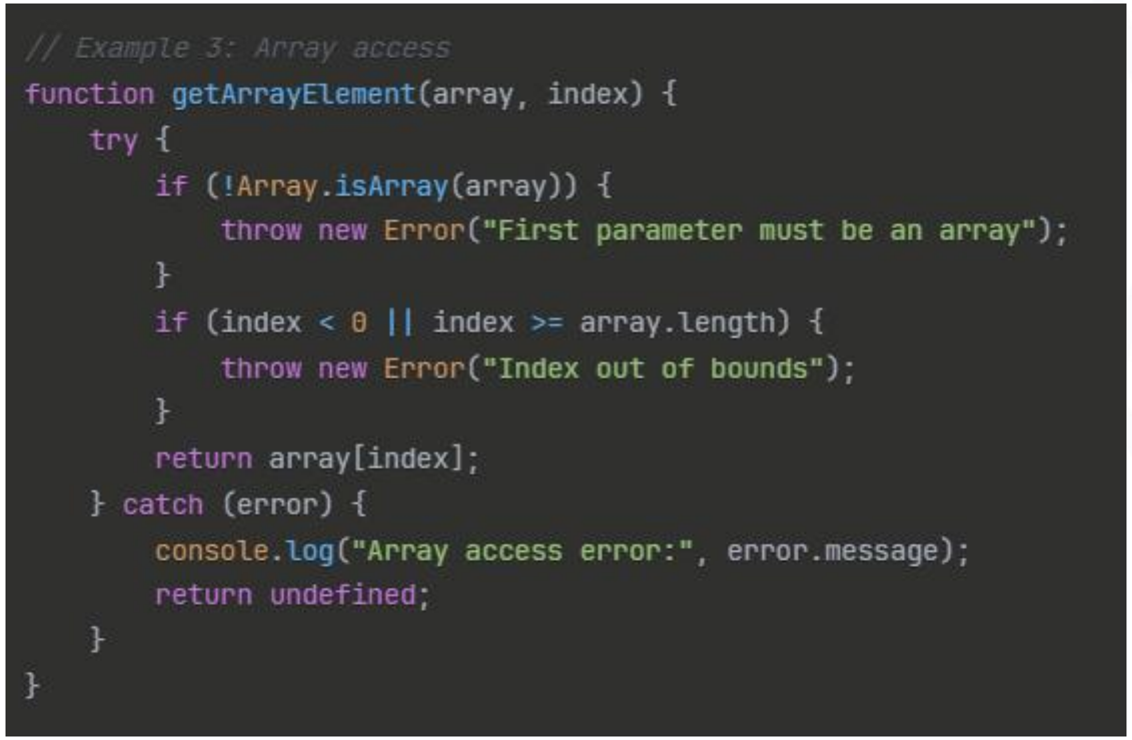
Real-World Examples:



```
// Example 1: JSON parsing
function parseUserData(jsonString) {
    try {
        let userData = JSON.parse(jsonString);
        return userData;
    } catch (error) {
        console.log("Invalid JSON format:", error.message);
        return null;
    }
}

// Example 2: Division function
function safeDivide(a, b) {
    try {
        if (b === 0) {
            throw new Error("Division by zero is not allowed!");
        }
        return a / b;
    } catch (error) {
        console.log("Math error:", error.message);
        return null;
    }
}

console.log(safeDivide(10, 2));  // 5
console.log(safeDivide(10, 0));  // null (with error message)
```

Error Handling - try/catch

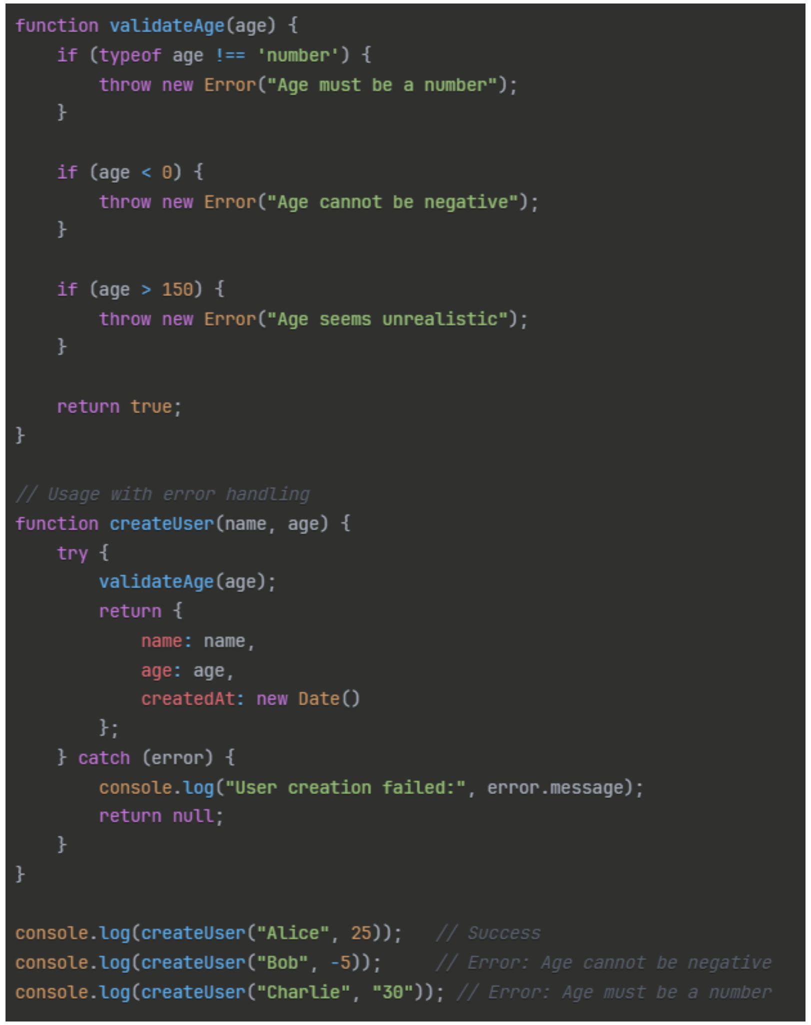
Real-World Examples:



```
// Example 3: Array access
function getArrayElement(array, index) {
    try {
        if (!Array.isArray(array)) {
            throw new Error("First parameter must be an array");
        }
        if (index < 0 || index >= array.length) {
            throw new Error("Index out of bounds");
        }
        return array[index];
    } catch (error) {
        console.log("Array access error:", error.message);
        return undefined;
    }
}
```

Throwing Custom Errors

Using 'throw' Statement:



```javascript
function validateAge(age) {
    if (typeof age !== 'number') {
        throw new Error("Age must be a number");
    }

    if (age < 0) {
        throw new Error("Age cannot be negative");
    }

    if (age > 150) {
        throw new Error("Age seems unrealistic");
    }

    return true;
}

// Usage with error handling
function createUser(name, age) {
    try {
        validateAge(age);
        return {
            name: name,
            age: age,
            createdAt: new Date()
        };
    } catch (error) {
        console.log("User creation failed:", error.message);
        return null;
    }
}

console.log(createUser("Alice", 25));   // Success
console.log(createUser("Bob", -5));     // Error: Age cannot be negative
console.log(createUser("Charlie", "30")); // Error: Age must be a number
```

Modules - Introduction Custom Errors

Why Modules?

- Code organization - Split large files into smaller, focused files
- Reusability - Use code across multiple projects
- Maintainability - Easier to update and debug
- Namespace - Avoid variable name conflicts
- Dependency management - Clear dependencies between files

Module Systems in JavaScript:

- CommonJS (Node.js traditional) - require() and module.exports
- ES6 Modules (Modern) - import and export

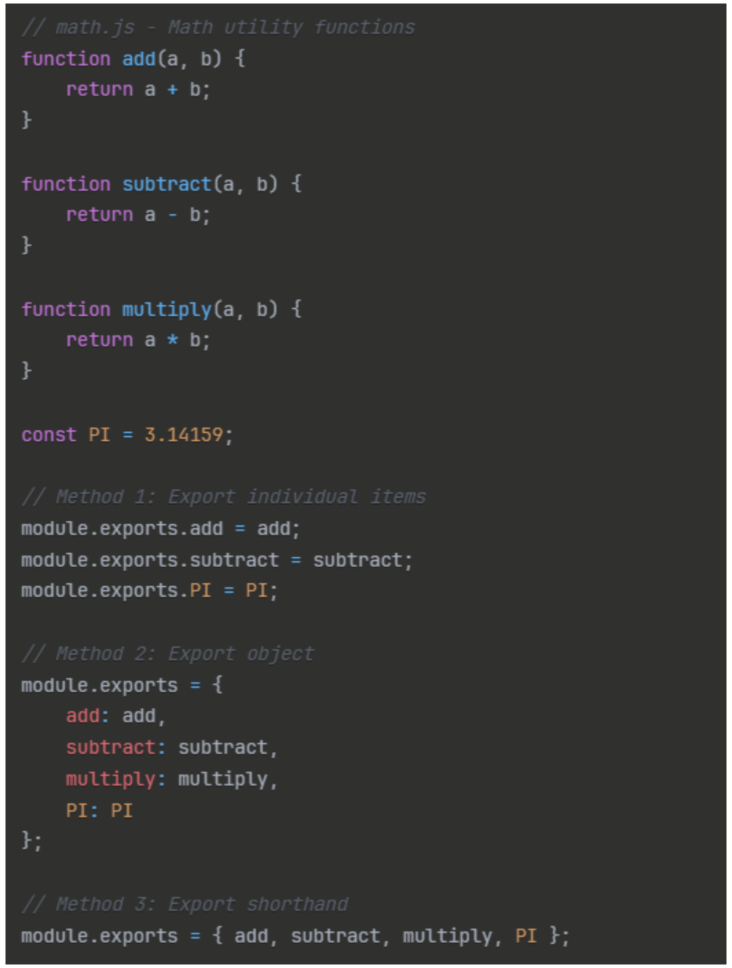
Modules - Introduction Custom Errors

Module Principles:

- Single Responsibility - Each module has one clear purpose
- High Cohesion - Related functionality grouped together
- Loose Coupling - Modules don't depend heavily on each other
- Clear Interface - Well-defined inputs and outputs

CommonJS Modules (require)

Exporting from Modules:



```javascript
// math.js - Math utility functions
function add(a, b) {
    return a + b;
}

function subtract(a, b) {
    return a - b;
}

function multiply(a, b) {
    return a * b;
}

const PI = 3.14159;

// Method 1: Export individual items
module.exports.add = add;
module.exports.subtract = subtract;
module.exports.PI = PI;

// Method 2: Export object
module.exports = {
    add: add,
    subtract: subtract,
    multiply: multiply,
    PI: PI
};

// Method 3: Export shorthand
module.exports = { add, subtract, multiply, PI };
```

CommonJS Modules (require)

Importing Modules:

```
// app.js - Main application file

// Import entire module
const math = require('./math.js');
console.log(math.add(5, 3));        // 8
console.log(math.PI);               // 3.14159

// Import with destructuring
const { add, subtract, PI } = require('./math.js');
console.log(add(10, 5));            // 15
console.log(subtract(10, 5));       // 5
console.log(PI);                    // 3.14159

// Import built-in Node.js modules
const fs = require('fs');           // File system
const path = require('path');       // Path utilities
const os = require('os');           // Operating system utilities
```

ES6 Modules (import/export)

Named Exports:

```javascript
// utils.js - Utility functions
export function formatCurrency(amount) {
    return `$${amount.toFixed(2)}`;
}

export function formatDate(date) {
    return date.toLocaleDateString();
}

export const TAX_RATE = 0.08;

// Alternative syntax
function calculateTip(amount, percentage) {
    return amount * (percentage / 100);
}

const COMPANY_NAME = "Tech Corp";

export { calculateTip, COMPANY_NAME };
```

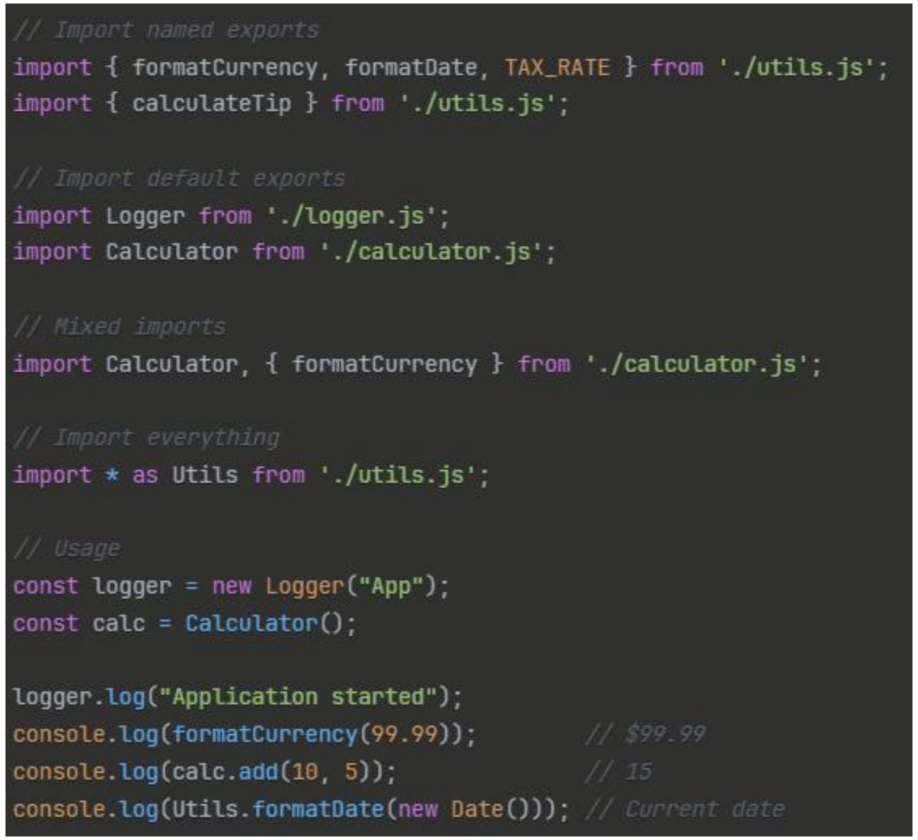
ES6 Modules (import/export)

Default Exports:

```javascript
// logger.js - Logging utility
class Logger {
    constructor(name) {
        this.name = name;
    }

    log(message) {
        console.log(`[${this.name}] ${new Date().toISOString()}: ${message}`);
    }

    error(message) {
        console.error(`[${this.name}] ERROR: ${message}`);
    }
}

export default Logger;

// calculator.js - Simple calculator
function Calculator() {
    return {
        add: (a, b) => a + b,
        subtract: (a, b) => a - b,
        multiply: (a, b) => a * b,
        divide: (a, b) => b !== 0 ? a / b : "Cannot divide by zero"
    };
}

export default Calculator;
```

ES6 Modules (import/export)

Importing ES6 Modules:



```
// Import named exports
import { formatCurrency, formatDate, TAX_RATE } from './utils.js';
import { calculateTip } from './utils.js';

// Import default exports
import Logger from './logger.js';
import Calculator from './calculator.js';

// Mixed imports
import Calculator, { formatCurrency } from './calculator.js';

// Import everything
import * as Utils from './utils.js';

// Usage
const logger = new Logger("App");
const calc = Calculator();

logger.log("Application started");
console.log(formatCurrency(99.99));        // $99.99
console.log(calc.add(10, 5));              // 15
console.log(Utils.formatDate(new Date())); // Current date
```

Practice: Book Object with Error Handling

Your Task:
Create a Book object with properties and methods, including error handling

Requirements:
- Create a Book constructor function or class
- Properties: title, author, year, pages
- Method: getDetails() - returns formatted book information
- Method: getAge() - calculates how old the book is
- Validate that year is a number and reasonable (1450-2025)
- Throw custom errors for invalid data
- Use try/catch when creating books
- Create a separate module for book-related functions

THANK YOU

ADAEGY