### **1. Let and Const**

#### **Explanation:**

* **let** and **const** are new ways to declare variables in JavaScript.
* **let** is block-scoped, meaning it is only accessible within the block it is defined.
* **const** is also block-scoped but is used to declare constants, meaning the variable's value cannot be reassigned after its initial assignment.

#### **Real-World Example:**

javascript

Copy code

function getDiscount(price) {

let discount = 0;

if (price > 100) {

let discount = 20; // Only accessible within this block

}

return price - discount;

}

const TAX\_RATE = 0.07; // Constant value

TAX\_RATE = 0.08; // This will throw an error because TAX\_RATE is constant

### **2. Arrow Functions**

#### **Explanation:**

* Arrow functions provide a shorter syntax for writing functions.
* They do not have their own this context, making them useful for methods that don’t require their own this or for passing as callbacks.

#### **Real-World Example:**

javascript

Copy code

const products = ['apple', 'banana', 'mango'];

const getProductLengths = products.map(product => product.length);

// Output: [5, 6, 5]

### **3. Template Literals**

#### **Explanation:**

* Template literals allow for easier string creation with embedded expressions, multi-line strings, and string interpolation.

#### **Real-World Example:**

javascript

Copy code

const name = "John";

const age = 30;

const greeting = `Hello, my name is ${name} and I am ${age} years old.`;

// Output: "Hello, my name is John and I am 30 years old."

### **4. Default Parameters**

#### **Explanation:**

* Default parameters allow named parameters to be initialized with default values if no value or undefined is passed.

#### **Real-World Example:**

javascript

Copy code

function calculatePrice(price, tax = 0.07) {

return price + (price \* tax);

}

console.log(calculatePrice(100)); // Output: 107

console.log(calculatePrice(100, 0.1)); // Output: 110

### **5. Destructuring Assignment**

#### **Explanation:**

* Destructuring allows you to unpack values from arrays or properties from objects into distinct variables.

#### **Real-World Example:**

javascript

Copy code

const person = { name: 'Alice', age: 25 };

const { name, age } = person;

console.log(name); // Output: 'Alice'

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

const numbers = [1, 2, 3];

const [first, second] = numbers;

console.log(first); // Output: 1

console.log(second); // Output: 2

### **6. Rest and Spread Operators**

#### **Explanation:**

* **Rest (...)** operator collects all remaining elements into an array.
* **Spread (...)** operator expands an array or object into its elements.

#### **Real-World Example:**

javascript

Copy code

function sum(...numbers) {

return numbers.reduce((acc, num) => acc + num, 0);

}

console.log(sum(1, 2, 3)); // Output: 6

const arr1 = [1, 2];

const arr2 = [3, 4];

const combined = [...arr1, ...arr2]; // Output: [1, 2, 3, 4]

### **7. Classes**

#### **Explanation:**

* Classes in JavaScript provide a simpler syntax for creating objects and handling inheritance, allowing for more intuitive and readable object-oriented programming.

#### **Real-World Example:**

javascript

Copy code

class Car {

constructor(brand, model) {

this.brand = brand;

this.model = model;

}

displayInfo() {

console.log(`Car: ${this.brand} ${this.model}`);

}

}

const car = new Car('Toyota', 'Corolla');

car.displayInfo(); // Output: 'Car: Toyota Corolla'

### **8. Modules**

#### **Explanation:**

* ES6 modules allow the import and export of code across different files, enabling better code organization and reuse.

#### **Real-World Example:**

javascript

Copy code

// math.js

export function add(a, b) {

return a + b;

}

// main.js

import { add } from './math.js';

console.log(add(2, 3)); // Output: 5

### **9. Promises**

#### **Explanation:**

* Promises represent the eventual completion (or failure) of an asynchronous operation and its resulting value, offering a cleaner and more robust way to handle asynchronous operations than callbacks.

#### **Real-World Example:**

javascript

Copy code

const fetchData = new Promise((resolve, reject) => {

const data = { id: 1, name: 'Alice' };

if (data) {

resolve(data);

} else {

reject('No data found');

}

});

fetchData

.then(data => console.log(data)) // Output: { id: 1, name: 'Alice' }

.catch(error => console.error(error));

### **10. Symbol**

#### **Explanation:**

* A Symbol is a unique and immutable data type used to create unique object property keys, which avoids property name collisions.

#### **Real-World Example:**

javascript

Copy code

const uniqueId = Symbol('id');

const user = {

[uniqueId]: 1,

name: 'Bob'

};

console.log(user[uniqueId]); // Output: 1

console.log(Object.keys(user)); // Output: ['name']

### **11. Iterators and Generators**

#### **Explanation:**

* **Iterators** are objects that enable iteration over collections like arrays or strings.
* **Generators** are functions that can be paused and resumed, allowing for asynchronous programming or handling sequences.

#### **Real-World Example:**

javascript

Copy code

function\* numberGenerator() {

yield 1;

yield 2;

yield 3;

}

const gen = numberGenerator();

console.log(gen.next().value); // Output: 1

console.log(gen.next().value); // Output: 2

console.log(gen.next().value); // Output: 3

### **12. Map and Set**

#### **Explanation:**

* **Map** is a collection of keyed data items, similar to an object, but allows keys of any type and remembers the original insertion order.
* **Set** is a collection of unique values, preventing duplicates.

#### **Real-World Example:**

javascript

Copy code

const map = new Map();

map.set('name', 'Alice');

map.set('age', 25);

console.log(map.get('name')); // Output: 'Alice'

const set = new Set();

set.add(1);

set.add(2);

set.add(2); // Duplicate, won't be added

console.log(set.size); // Output: 2

This study material covers the foundational aspects of ES6 with explanations and examples that illustrate how these features are used in real-world applications. You can expand each topic with more examples or exercises to enhance learning.