Here's a more structured and detailed set of notes on the JavaScript topics you should know before diving into React:

---

1. ES6+ Features

Arrow Functions

- **Syntax:** `(parameters) => { statements }`
- **Example:**

```javascript
const add = (a, b) => a + b;
```

- **Behavior:** Arrow functions do not have their own `this`, `arguments`, or `super`. Useful for functions where you need to retain the lexical `this`.

Destructuring

- **Object Destructuring:**

```javascript
const person = { name: 'Alice', age: 30 };
const { name, age } = person;
```

- **Array Destructuring:**

```javascript
const [first, second] = [1, 2, 3];
```

- **Usage:** Simplifies extracting values from arrays and objects.

Template Literals

- **Syntax:** Use backticks (`` ` ``) instead of quotes.
- **Example:**

```javascript
const name = 'Bob';
const greeting = `Hello, ${name}!`;
```

- **Features:** Supports multi-line strings and embedded expressions.

Classes

- **Syntax:**

```javascript
class Person {
  constructor(name) {
    this.name = name;
  }
  greet() {
    return `Hello, ${this.name}`;
  }
}
```

- **Usage:** Provides a syntax for creating objects with methods and inheritance.

Modules

- **Exporting:**

```javascript
export const pi = 3.14;
export function add(x, y) { return x + y; }
```

- **Importing:**

```javascript
import { pi, add } from './math';
```

- **Usage:** Helps in organizing code into reusable modules.

---

2. Asynchronous JavaScript

Promises

- **Syntax:**

```javascript
const promise = new Promise((resolve, reject) => {
  // async operation
  if (success) resolve(result);
  else reject(error);
});
promise.then(result => console.log(result)).catch(error =>
console.error(error));
```

- **Usage:** Handles asynchronous operations by chaining `.then()` and `.catch()`.

Async/Await

- **Syntax:**

```javascript
async function fetchData() {
  try {
    const response = await fetch('https://api.example.com/data');
    const data = await response.json();
    console.log(data);
  } catch (error) {
    console.error(error);
  }
}
```

- **Usage:** Provides a more readable way to work with asynchronous code using `await` inside `async` functions.

---

3. Array Methods

`.map()`

- **Usage:** Creates a new array with the results of calling a provided function on every element.
- **Example:**

```javascript
const numbers = [1, 2, 3];
const doubled = numbers.map(num => num * 2);
```

`.filter()`

- **Usage:** Creates a new array with all elements that pass the test implemented by the provided function.
- **Example:**

```javascript
const numbers = [1, 2, 3, 4];
const evenNumbers = numbers.filter(num => num % 2 === 0);
```

`.reduce()`

- **Usage:** Executes a reducer function on each element of the array, resulting in a single output value.
- **Example:**

```javascript
const numbers = [1, 2, 3];
const sum = numbers.reduce((accumulator, current) => accumulator + current,
0);
```

---

4. Object Manipulation

Creating Objects

- **Example:**

```javascript
const person = { name: 'Alice', age: 30 };
```

Updating Properties

- **Example:**

```javascript
person.age = 31;
```

Accessing Properties

- **Example:**

```javascript
const name = person.name;
```

Destructuring

- **Example:**

```javascript
const { name, age } = person;
```

---

5. Event Handling

Adding Event Listeners

- **Syntax:**

```javascript
button.addEventListener('click', () => {
  alert('Button clicked!');
});
```

- **Event Object:** Provides information about the event (e.g., `event.target` to get the element that triggered the event).

---

6. Understanding the DOM

Selecting Elements

- **Syntax:**

```javascript
const element = document.querySelector('.className');
```

Manipulating Elements

- **Syntax:**

```javascript
element.textContent = 'New Text';
```

Handling Events

- **Syntax:**

```javascript
element.addEventListener('click', () => {
  console.log('Element clicked');
});
```

These notes should give you a solid foundation in JavaScript before you start working with React.