ES6+ and Modern JavaScript Notes

Arrow Functions:

- Introduced in ES6 for shorter function syntax.
- Do not bind their own 'this', 'arguments', 'super', or 'new.target'.
- Cannot be used as constructors (no 'new').
- Syntax:

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

- If there's only one parameter, parentheses are optional:

```
x => x * 2;
```

Classes:

- Syntactic sugar over prototype-based inheritance.
- Defined using the 'class' keyword.
- Supports constructor, methods, getters/setters, static methods.
- Example:

```
class Person {
  constructor(name) {
    this.name = name;
  }
  greet() {
    console.log(`Hello, ${this.name}`);
  }
  static species() {
    return 'Human';
  }
}
```

Modules:

- Allow code to be split into separate files.
- Use 'export' to expose functionality and 'import' to use it.
- Example:

```
// math.js
export function add(a, b) { return a + b; }
// main.js
import { add } from './math.js';
```

- Modules are strict mode by default.
- Can have default exports:

```
export default function() { ... }
```

Optional Chaining (?.):

- Allows safe access to deeply nested properties.
- Prevents errors if a property is null or undefined.
- Example:

```
const user = { profile: { name: 'John' } };
console.log(user.profile?.name);  // John
console.log(user.settings?.theme);  // undefined
```

Nullish Coalescing (??):

- Returns the right-hand side if the left-hand side is null or undefined.
- Different from || (which treats falsy values like '', 0 as false).
- Example:

```
const name = null ?? 'Guest';  // 'Guest'
const age = 0 ?? 18;           // 0 (not replaced)
```

Promises & Async/Await:

- Promise: Represents a value that may be available now, later, or never.

```
const promise = new Promise((resolve, reject) => { ... });
promise.then(...).catch(...);
```

- Async/Await: Cleaner syntax for working with Promises.

```
async function fetchData() {
  try {
    const result = await fetch(url);
    const data = await result.json();
  } catch (err) {
    console.error(err);
  }
}
```

Generators:

- Functions that can be paused and resumed.
- Defined with 'function*' and use 'yield'.
- Example:

```
function* numbers() {
  yield 1;
  yield 2;
  yield 3;
}
const gen = numbers();
console.log(gen.next().value);  // 1
```

- Useful for iterators, async tasks, and state machines.

WeakMap & WeakSet:

- WeakMap:
  - A map where keys must be objects.
  - Keys are weakly referenced (garbage collected if no other reference).
  - No size property; not iterable.
  - Example:

```
const wm = new WeakMap();
const obj = {};
wm.set(obj, 'value');
```

- WeakSet:
  - Similar to Set but only stores objects (no primitives).
  - Weakly referenced (can be garbage collected).
  - Not iterable, no size property.
  - Example:

```
const ws = new WeakSet();
const obj = {};
ws.add(obj);
```