Task – 3

Iterable Protocol in JavaScript

Explanation of the concept:

Iteration Protocol is a protocol that can be implemented by any object that follows the required conventions.
It allows JavaScript objects to define or customize their iteration behavior, even if they are not built-in iterables in JavaScript.
To make an object iterable, it must implement the Symbol.iterator() method, and it must return an iterator used to obtain the values to be iterated.

Real code examples with short descriptions of what they do:

```
const customIterable = {
  *[Symbol.iterator]() {
    yield 1;
    yield 2;
    yield 3;
  },
};

for (const value of customIterable) {
  console.log(value);
}
```

Output:

```
1
2
3
```

In this example, the object customIterable becomes iterable by defining the Symbol.iterator method using a generator. When we use for...of, it automatically calls this method and starts the iteration.

The importance or use cases of the concept in real-world JavaScript development:

It plays a critical role in libraries and frameworks that need to traverse complex data sources or return lazy-loaded data on demand.

Generators in JavaScript

Explanation of the concept:

Generators are special functions in JavaScript that can pause and resume execution. They are defined using the function* syntax and use the yield keyword to return values one at a time. Unlike regular functions that run to completion.

Generator is a subclass of the hidden *Iterator* class.

Real code examples with short descriptions of what they do:

```
function* countToThree() {
  yield 1;
  yield 2;
  yield 3;
}

const counter = countToThree();
console.log(counter.next());
console.log(counter.next());
console.log(counter.next());
console.log(counter.next());
```

Output:

```
{ value: 1, done: false }
{ value: 2, done: false }
{ value: 3, done: false }
{ value: undefined, done: true }
```

Each next() call resumes the generator from the last yield. When there's no more code, it returns { done: true }.

The importance or use cases of the concept in real-world JavaScript development:

Generators are useful in real-world JavaScript because they maintain their state between yield calls, making them ideal for state machines or step-by-step logic without extra variables or complex control flow.