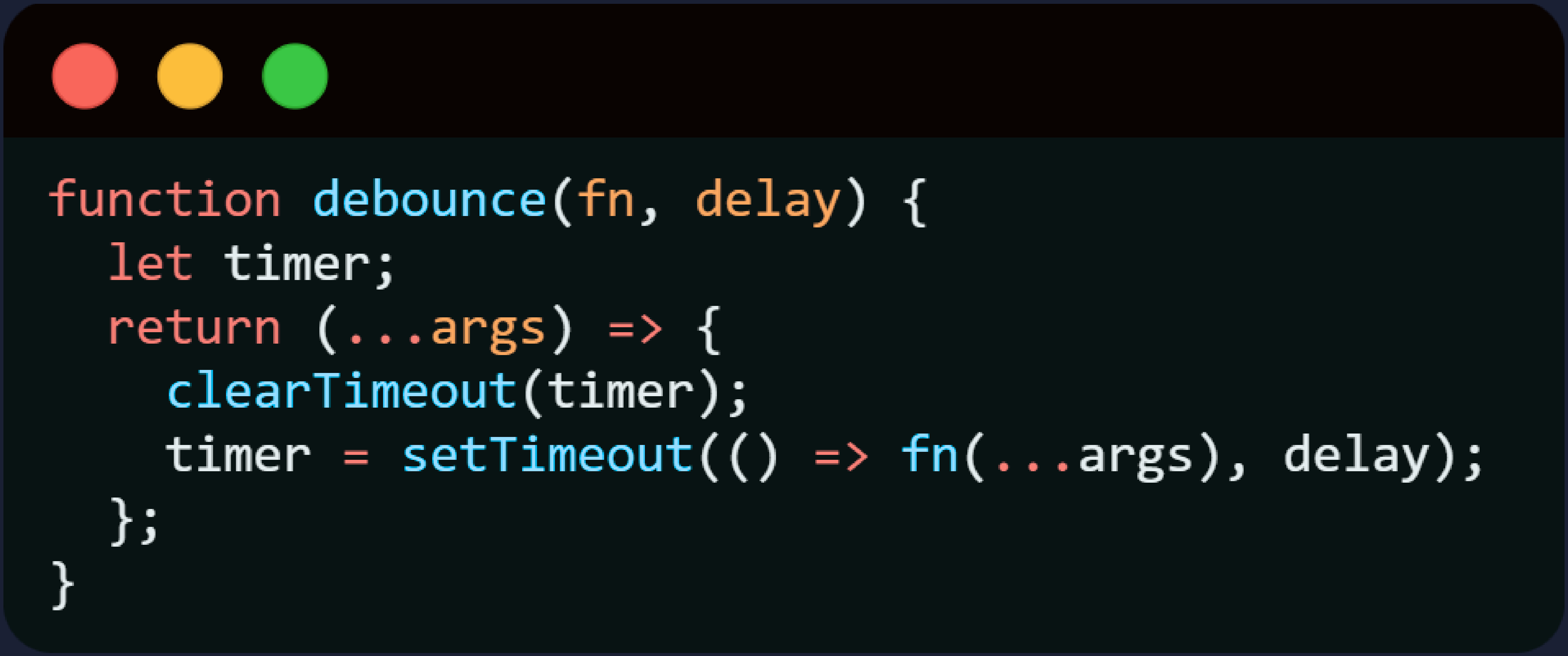
Debounce

- Debounce is a technique used to delay the execution of a function until after a certain time has passed since it was last called.’
- Prevents a function from running too often.

- Example use cases:
  - Search input (wait until user stops typing before sending API call).
  - Window resize events.

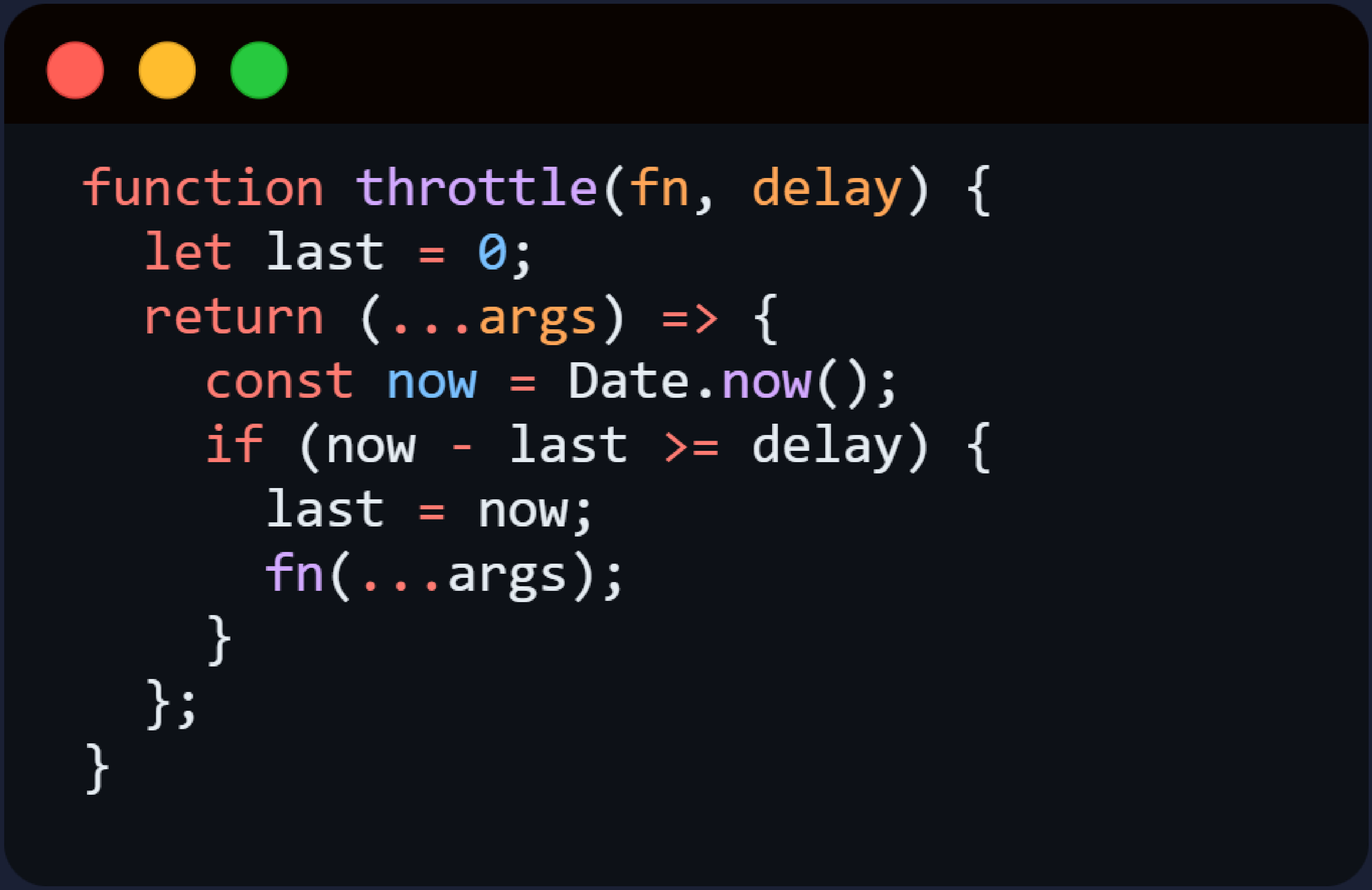


```javascript
function debounce(fn, delay) {
  let timer;
  return (...args) => {
    clearTimeout(timer);
    timer = setTimeout(() => fn(...args), delay);
  };
}
```

Throttle

- Throttle ensures a function runs at most once in a specified interval, no matter how many times it's triggered.
- Limits execution rate for performance.

- Example use cases:
    - Scroll events.
    - Button clicks (to prevent double submission).



```
function throttle(fn, delay) {
  let last = 0;
  return (...args) => {
    const now = Date.now();
    if (now - last >= delay) {
      last = now;
      fn(...args);
    }
  };
}
```

Deep Clone

- A method of creating a copy of an object along with all nested objects, instead of just referencing the original.
- Prevents accidental changes to the original object when modifying the clone.

- Example use cases:
  - State management in apps (React/Angular).
  - Copying configuration objects.

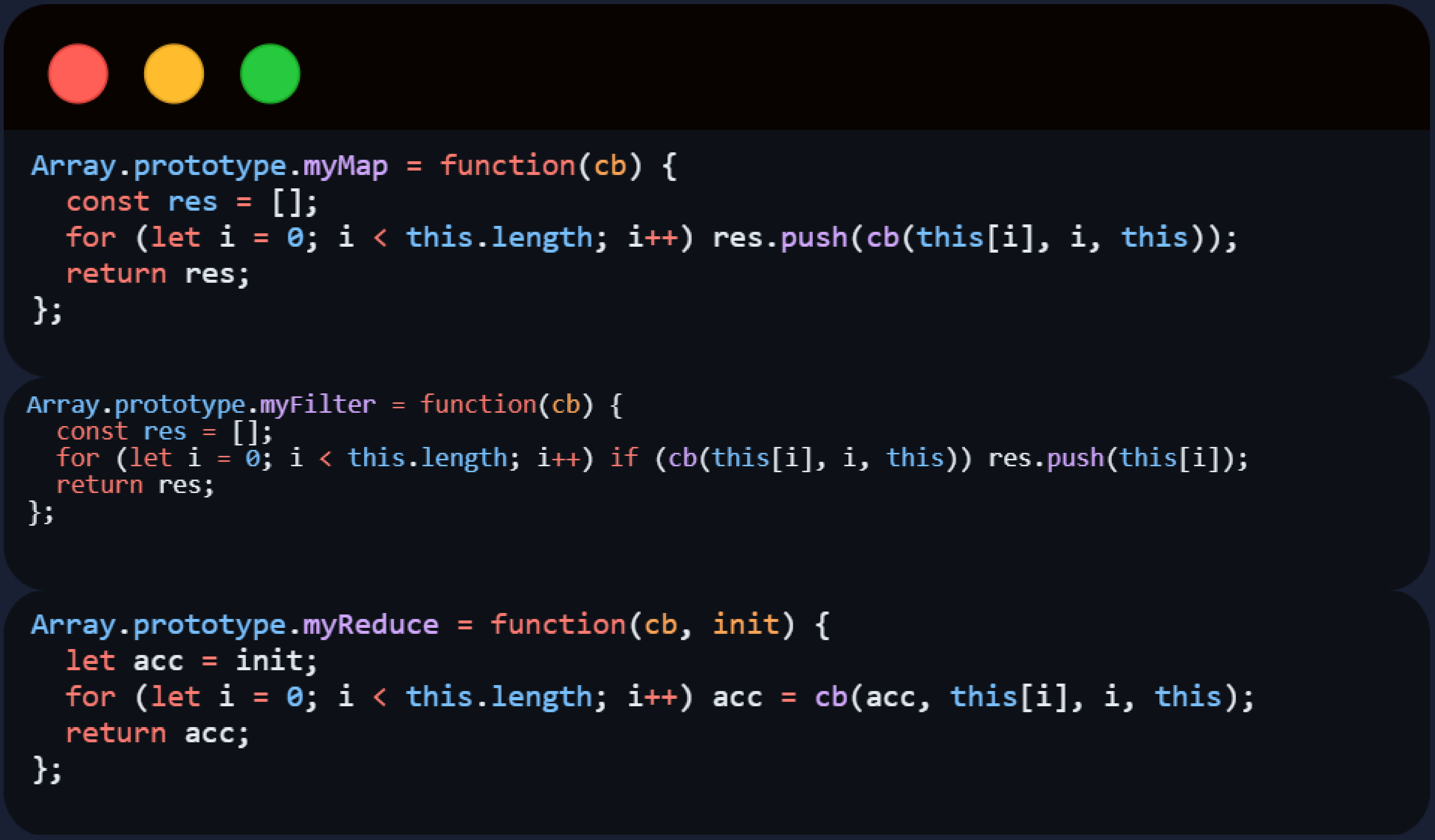
```
function deepClone(obj) {
  if (obj === null || typeof obj !== "object") return obj;
  if (Array.isArray(obj)) return obj.map(deepClone);

  const clone = {};
  for (let key in obj) {
    clone[key] = deepClone(obj[key]);
  }
  return clone;
}
```

Array Polyfills

- Custom implementations of built-in array methods.
- Useful for interview prep and core JavaScript understanding.

- Example use cases:
  - map → transform array values.
  - filter → return values matching a condition.
  - reduce → accumulate values into a single result (sum, average, object building).



```javascript
Array.prototype.myMap = function(cb) {
  const res = [];
  for (let i = 0; i < this.length; i++) res.push(cb(this[i], i, this));
  return res;
};
```

```javascript
Array.prototype.myFilter = function(cb) {
  const res = [];
  for (let i = 0; i < this.length; i++) if (cb(this[i], i, this)) res.push(this[i]);
  return res;
};
```

```javascript
Array.prototype.myReduce = function(cb, init) {
  let acc = init;
  for (let i = 0; i < this.length; i++) acc = cb(acc, this[i], i, this);
  return acc;
};
```

Promise.all

- Helps you handle multiple asynchronous operations concurrently.
- Ensures that results from all promises are collected in the same order as the input array.

- Example use cases:
    - Fetching data from multiple APIs at once and processing them together.
    - Running several independent asynchronous tasks in parallel (e.g., reading multiple files, processing images).

```
function myPromiseAll(promises) {
  return new Promise((resolve, reject) => {
    let results = [];
    let count = 0;

    promises.forEach((p, i) => {
      Promise.resolve(p).then(res => {
        results[i] = res;
        count++;
        if (count === promises.length) resolve(results);
      }).catch(reject);
    });
  });
}
```