Clean Code

1. Avoid if/else branching, use early termination instead.

```
function processUser(user) {
  if (user != null) {
    if (user.hasSubscription) {
      if (user.age >= 18) {
        showFullVersion();
      } else {
        showChildrenVersion();
      }
    } else {
      throw new Error('User needs a subscription');
    }
  } else {
    throw new Error('No user found');
  }
}
```

→

```
function processUser(user) {
  if (user == null)
    throw new Error('No user found');

  if (!user.hasSubscription)
    throw new Error('User needs a subscription');

  if (user.age < 18)
    return showChildrenVersion();

  showFullVersion();
}
```

2. Boolean methods should start with ‘is’

```
const MIN_PASSWORD = 6;

function checkPasswordLength(password) {
  return password.length >= MIN_PASSWORD;
}
```

→

```
const MIN_PASSWORD_LENGTH = 6;

function isPasswordLongEnough(password) {
  return password.length >= MIN_PASSWORD_LENGTH;
}
```

3. Write necessary comments only

```
// Function to check if a number is prime
function isPrime(number) {
  // Check if number is less than 2
  if (number < 2) {
    // If less than 2, not a prime number
    return false;
  }

  // At least 1 divisor must but less than square root, so we can stop there
  for (let i = 2; i <= Math.sqrt(number); i++) {
    // Check if number is divisible by i
    if (number % i === 0) {
      // If divisible, number is not prime
      return false;
    }
  }

  // After all checks, if not divisible by any i, number is prime
  return true;
}
```

→

```
function isPrime(number) {
  if (number < 2) {
    return false;
  }

  // At least 1 divisor must but less than square root, so we can stop there
  for (let i = 2; i <= Math.sqrt(number); i++) {
    if (number % i === 0) {
      return false;
    }
  }

  return true;
}
```

4. Consistent formatting

```
const name = "Conner";
let age=26;

function getUserInfo() {
  console.log("User Info:");
  console.log('Name: ' + name)
  console.log(`Age: ${age}`);
}
```

→

```
const name = "Conner";
const age = 26;

function getUserInfo() {
  console.log("User Info:");
  console.log(`Name: ${name}`)
  console.log(`Age: ${age}`);
}
```

5. DRY (don't repeat yourself)

```
function logLogin() {
  console.log('User logged in at ' + new Date());
}

function logLogout() {
  console.log('User logged out at ' + new Date());
}

function logSignUp() {
  console.log('User signed up at ' + new Date());
}
```

→

```
function logAction(action) {
  console.log(`User ${action} at ${new Date()}`);
}
```

6. Early returns

```
function getUppercaseInput(input) {
  const result = input?.toUpperCase?.();

  if (typeof input !== 'string' || input.trim() === '') {
    throw new Error('Invalid input');
  }

  return result;
}
```

→

```
function getUppercaseInput(input) {
  if (typeof input !== 'string' || input.trim() === '') {
    throw new Error('Invalid input');
  }

  const result = input.toUpperCase();

  return result;
}
```

7. Avoid magic values - declare and use CONSTANTS instead

```
let price = 10;
if (transactionType === 1) {
  price *= 1.1;
}
```

→

```
const TAXABLE_TRANSACTION_TYPE = 1;
const TAX_MULTIPLE = 1.1;

let price = 10;
if (transactionType === TAXABLE_TRANSACTION_TYPE) {
  price *= TAX_MULTIPLE;
}
```

8. Avoid violating single responsibility. Prefer to use pure functions (no side effects)

```
let area = 0;

function calculateAndUpdateArea(radius) {
  const newArea = Math.PI * radius * radius;
  area = newArea;
  return newArea;
}
```

→

```
let area = 0;

function calculateArea(radius) {
  return Math.PI * radius * radius;
}

area = calculateArea(5);
```

9. Avoid overly clever code and always go for readable ones

```
const numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];

const result = numbers.reduce((acc, n) => n & 1 ? [...acc, n * n] : acc, []);
console.log(result); // Output: [ 1, 9, 25, 49, 81 ]

```

→

```
// More readable approach
const filteredAndSquared = numbers.filter(n => n % 2 !== 0).map(n => n * n);
console.log(filteredAndSquared); // Output: [ 1, 9, 25, 49, 81 ]
```

10. Avoid useless optimizations

```
function countingSort(arr, min, max) {
  let count = new Array(max - min + 1).fill(0);
  arr.forEach((element) => {
    count[element - min]++;
  });

  let index = 0;
  for (let i = min; i <= max; i++) {
    while (count[i - min] > 0) {
      arr[index++] = i;
      count[i - min]--;
    }
  }

  return arr;
}

const arr = [4, 2, 2, 8, 3, 3, 1];
console.log(countingSort(arr, 1, 8)); // Output:
```

Here the array is already small in size, so all this optimization is not really needed instead these extra lines of code can cause bugs down the line.