BONUS: COMMON CODING CHALLENGES

1. Debounce Function Implementation

javascript

```javascript
function debounce(func, delay) {
  let timeoutId;

  return function(...args) {
    clearTimeout(timeoutId);
    timeoutId = setTimeout(() => func.apply(this, args), delay);
  };
}

// Usage Example
const searchInput = document.getElementById('search');
const debouncedSearch = debounce((query) => {
  console.log('Searching for:', query);
  // API call here
}, 300);

searchInput.addEventListener('input', (e) => {
  debouncedSearch(e.target.value);
});
```

2. Deep Clone Object

javascript

```javascript
function deepClone(obj) {
  if (obj === null || typeof obj !== 'object') {
    return obj;
  }

  if (obj instanceof Date) {
    return new Date(obj.getTime());
  }

  if (obj instanceof Array) {
    return obj.map(item => deepClone(item));
  }

  if (typeof obj === 'object') {
    const clonedObj = {};
    for (let key in obj) {
      if (obj.hasOwnProperty(key)) {
        clonedObj[key] = deepClone(obj[key]);
      }
```

```
    }
    return clonedObj;
  }
}

// Usage
const original = {
  name: 'John',
  age: 30,
  hobbies: ['reading', 'coding'],
  address: {
    city: 'New York',
    country: 'USA'
  }
};

const cloned = deepClone(original);
cloned.hobbies.push('swimming');

console.log(original.hobbies); // ['reading', 'coding'] - unchanged
```

3. Custom Promise Implementation

javascript

```
class MyPromise {
  constructor(executor) {
    this.state = 'pending';
    this.value = undefined;
    this.onFulfilledCallbacks = [];
    this.onRejectedCallbacks = [];

    const resolve = (value) => {
      if (this.state === 'pending') {
        this.state = 'fulfilled';
        this.value = value;
        this.onFulfilledCallbacks.forEach(cb => cb(value));
      }
    };

    const reject = (reason) => {
      if (this.state === 'pending') {
        this.state = 'rejected';
        this.value = reason;
        this.onRejectedCallbacks.forEach(cb => cb(reason));
      }
    };

    try {
      executor(resolve, reject);
```

```
    } catch (error) {
      reject(error);
    }
  }

  then(onFulfilled, onRejected) {
    return new MyPromise((resolve, reject) => {
      if (this.state === 'fulfilled') {
        try {
          const result = onFulfilled ? onFulfilled(this.value) :
this.value;
          resolve(result);
        } catch (error) {
          reject(error);
        }
      } else if (this.state === 'rejected') {
        try {
          const result = onRejected ? onRejected(this.value) :
this.value;
          resolve(result);
        } catch (error) {
          reject(error);
        }
      } else {
        this.onFulfilledCallbacks.push((value) => {
          try {
            const result = onFulfilled ? onFulfilled(value) :
value;
            resolve(result);
          } catch (error) {
            reject(error);
          }
        });

        this.onRejectedCallbacks.push((reason) => {
          try {
            const result = onRejected ? onRejected(reason) :
reason;
            resolve(result);
          } catch (error) {
            reject(error);
          }
        });
      }
    });
  }

  catch(onRejected) {
```

```
    return this.then(null, onRejected);
  }
}
```