

# JavaScript for QA Problem Set 2 Solutions

## 1 More Object & Array Handling

**Problem 1.1:** Filter only users with valid emails and age above 18.

```
function getValidAdultUsers(users) {  
  const emailRegex = /^\\S+@\\S+\\.\\S+$/;  
  return users.filter(user => user.age > 18 &&  
    emailRegex.test(user.email));  
}
```

**Problem 1.2:** Transform test results into grouped counts.

```
function countTestResults(tests) {  
  return tests.reduce((acc, test) => {  
    acc[test.status] = (acc[test.status] || 0) + 1;  
    return acc;  
  }, {});  
}
```

**Problem 1.3:** Convert array of key-value pairs to object.

```
function pairsToObject(pairs) {  
  return Object.fromEntries(pairs);  
}
```

## 2 Deep Validation and Assertions

**Problem 2.1:** Validate response contains nested properties.

```
function hasNestedFields(obj, fields) {
  return fields.every(field => {
    let keys = field.split('.');
    let value = obj;
    for (let key of keys) {
      if (!value || !value.hasOwnProperty(key))
        return false;
      value = value[key];
    }
    return true;
  });
}
```

**Problem 2.2:** Assert array has only unique values (like Postman test).

```
function assertUniqueValues(arr) {
  return new Set(arr).size === arr.length;
}
```

**Problem 2.3:** Compare two responses deeply.

```
function deepEqual(a, b) {
  if (a === b) return true;
  if (typeof a !== "object" || typeof b !== "object")
    return false;

  let keysA = Object.keys(a);
  let keysB = Object.keys(b);

  if (keysA.length !== keysB.length) return false;

  for (let key of keysA) {
    if (!deepEqual(a[key], b[key])) return false;
  }
}
```

```
    return true;
}
```

### 3 String Parsing and Regex

**Problem 3.1:** Extract domain from email.

```
function getEmailDomain(email) {
    const match = email.match(/@(.+)$/);
    return match ? match[1] : null;
}
```

**Problem 3.2:** Mask sensitive parts of a string (e.g., password).

```
function maskPassword(pw) {
    if (pw.length <= 2) return '*'.repeat(pw.length);
    return pw[0] + '*'.repeat(pw.length - 2) + pw[pw.
        length - 1];
}
```

**Problem 3.3:** Parse log entries and return error messages.

```
function extractErrors(logs) {
    return logs
        .filter(entry => entry.level === 'error')
        .map(err => err.message);
}
```

### 4 Functional Programming Patterns

**Problem 4.1:** Chain filter, map, and reduce to compute average age of valid users.

```
function averageAge(users) {
  const valid = users.filter(u => u.age > 0);
  const sum = valid.reduce((acc, u) => acc + u.age, 0)
  ;
  return valid.length ? sum / valid.length : 0;
}
```

#### Problem 4.2: Group items by property.

```
function groupBy(arr, key) {
  return arr.reduce((acc, item) => {
    const val = item[key];
    acc[val] = acc[val] || [];
    acc[val].push(item);
    return acc;
  }, {});
}

// This one is perhaps more intuitive:

function groupBy(arr, key) {
  const result = {};
  arr.forEach(item => {
    const value = item[key];
    if (!result[value]) {
      result[value] = [];
    }
    result[value].push(item);
  });
  return result;
}
```

#### Problem 4.3: Flatten a deeply nested array (1 level).

```
function flattenArray(arr) {
  return arr.reduce((flat, toFlatten) => flat.concat(
    toFlatten), []);
}
```

```

}

// Or the solution we've already seen from the general
// Javascript practice:

function flattenArray(arr) {
  let result = [];
  arr.forEach(item => {
    if (Array.isArray(item)) {
      result.push(...item);
    } else {
      result.push(item);
    }
  });
  return result;
}

```

## 5 Asynchronous QA Concepts

**Problem 5.1:** Simulate delayed API test and validate the response.

```

async function simulateAPICheck() {
  function mockFetch() {
    return new Promise(resolve => {
      setTimeout(() => resolve({ status: 200, data
        : { user: 'QA' } })), 1000);
    });
  }

  const response = await mockFetch();
  return response.status === 200 && response.data.user
    === 'QA';
}

```

**Problem 5.2:** Retry a function up to 3 times if it fails.

```
async function retry(fn, retries = 3) {  
  for (let i = 0; i < retries; i++) {  
    try {  
      return await fn();  
    } catch (e) {  
      if (i === retries - 1) throw e;  
    }  
  }  
}
```

**Problem 5.3:** Run multiple async validations and collect results.

```
async function runAllValidations(validations) {  
  const results = await Promise.all(validations.map(v  
    => v()));  
  return results;  
}
```