Web-Entwicklung

Jil Zerndt, Lucien Perret
January 2025

Übungsaufgaben

JavaScript Grundlagen

Datentypen und Operatoren

Aufgabe 1: Was ist die Ausgabe folgender Ausdrücke?

```
typeof NaN
typeof []
typeof null
typeof undefined
[] == false
null === undefined
"5" + 3
"5" - 3
```

Lösung:

```
"number"      // NaN ist vom Typ number
"object"      // Arrays sind Objekte
"object"      // null ist historisch ein Objekt
"undefined"   // undefined ist ein eigener Typ
true          // [] wird zu 0 konvertiert
false         // === vergleicht auch Typen
"53"          // String-Konkatenation
2             // Numerische Subtraktion
```

Funktionen und Scoping

Aufgabe 2: Was ist die Ausgabe dieses Codes?

```
let x = 1;
const f = () => {
    let x = 2;
    return {
        getX: () => x,
        setX: (val) => { x = val; }
    };
};
const obj = f();
console.log(x);        // ?
console.log(obj.getX()); // ?
obj.setX(3);
console.log(obj.getX()); // ?
console.log(x);        // ?
```

Lösung:

```
1    // Globales x bleibt 1
2    // Closure hat Zugriff auf lokales x
3    // Lokales x wird auf 3 gesetzt
1    // Globales x bleibt unveraendert
```

DOM und Events

DOM Manipulation

Aufgabe 3: Erstellen Sie eine Funktion, die eine ToDo-Liste verwaltet.

```
function createTodoList(containerId) {
    // Container finden
    const container =
        document.getElementById(containerId);

    // Input und Liste erstellen
    const input = document.createElement('input');
    const button = document.createElement('button');
    const list = document.createElement('ul');

    // Button konfigurieren
    button.textContent = 'Add';
    button.onclick = () => {
        if (input.value.trim()) {
            const li = document.createElement('li');
            li.textContent = input.value;
            list.appendChild(li);
            input.value = '';
        }
    };

    // Elemente zusammenfuegen
    container.appendChild(input);
    container.appendChild(button);
    container.appendChild(list);
}
```

Event Handling

Aufgabe 4: Implementieren Sie einen Klick-Zähler mit Event Delegation.

```
document.getElementById('container').addEventListener('clic
    (e) => {
        if (e.target.matches('button')) {
            const count = (
                parseInt(e.target.dataset.count) || 0
            ) + 1;
            e.target.dataset.count = count;
            e.target.textContent = `Clicked ${count}
                times`;
        }
    }
);
```

Client-Server Kommunikation

Fetch API

Aufgabe 5: Implementieren Sie eine Funktion für API-Requests.

```
async function apiRequest(url, method = 'GET', data =
    null) {
    const options = {
        method,
        headers: {
            'Content-Type': 'application/json'
        }
    };

    if (data) {
        options.body = JSON.stringify(data);
    }

    try {
        const response = await fetch(url, options);
        if (!response.ok) {
            throw new Error(`HTTP error:
                ${response.status}`);
        }
        return await response.json();
    } catch (error) {
        console.error('API request failed:', error);
        throw error;
    }
}
```

Formular-Validierung

Aufgabe 6: Erstellen Sie eine Formular-Validierung.

```
function validateForm(formId) {
    const form = document.getElementById(formId);

    form.addEventListener('submit', (e) => {
        e.preventDefault();

        const formData = new FormData(form);
        const errors = [];

        // Email validieren
        const email = formData.get('email');
        if (!email.includes('@')) {
            errors.push('Invalid email');
        }

        // Passwort validieren
        const password = formData.get('password');
        if (password.length < 8) {
            errors.push('Password too short');
        }

        if (errors.length === 0) {
            // Form submission logic
            console.log('Form valid, submitting...');
            form.submit();
        } else {
            alert(errors.join('\n'));
        }
    });
}
```

UI-Komponenten

SuiWeb Komponente **Aufgabe 7:** Erstellen Sie eine Counter-Komponente mit SuiWeb.

```
const Counter = () => {
    const [count, setCount] = useState(0);

    return [
        "div",
        ["h2", `Count: ${count}`],
        ["button",
            {onclick: () => setCount(count + 1)},
            "Increment"
        ],
        ["button",
            {onclick: () => setCount(count - 1)},
            "Decrement"
        ]
    ];
};
```

Container Component **Aufgabe 8:** Implementieren Sie eine UserList-Komponente.

```
const UserList = () => {
    const [users, setUsers] = useState([]);
    const [loading, setLoading] = useState(true);

    if (loading) {
        fetchUsers()
            .then(data => {
                setUsers(data);
                setLoading(false);
            })
            .catch(error => {
                console.error(error);
                setLoading(false);
            });
    }

    if (loading) {
        return ["div", "Loading..."];
    }

    return [
        "div",
        ["h2", "Users"],
        ["ul",
            ...users.map(user =>
                ["li", `${user.name} (${user.email})`]
            )
        ]
    ];
};
```

Theoriefragen

Konzeptfragen **1. Erklären Sie den Unterschied zwischen == und === in JavaScript.**
Antwort: == vergleicht Werte mit Typumwandlung, === vergleicht Werte und Typen ohne Umwandlung.
2. Was ist Event Bubbling?
Antwort: Events werden von dem auslösenden Element durch den DOM-Baum nach oben weitergeleitet.
3. Was ist der Unterschied zwischen localStorage und sessionStorage?
Antwort: localStorage persistiert Daten auch nach Schließen des Browsers, sessionStorage nur während der Session.
4. Erklären Sie den Unterschied zwischen synchronem und asynchronem Code.
Antwort: Synchroner Code wird sequentiell ausgeführt, asynchroner Code ermöglicht parallele Ausführung ohne Blockierung.

Praktische Aufgaben

Implementierungsaufgaben **1. Implementieren Sie eine Funktion zur Deep Copy von Objekten.**
2. Erstellen Sie eine Funktion, die prüft ob ein String ein Palindrom ist.
3. Implementieren Sie eine debounce-Funktion.
4. Erstellen Sie eine Komponente für einen Image Slider.

Debugging-Aufgaben **1. Finden Sie den Fehler im folgenden Code:**

```
const getData = () => {
    fetch('api/data')
        .then(response => response.json())
        .then(data => {
            return data;
        });
}
// Warum kommt undefined zurueck?
```

Antwort: Die Funktion hat kein explizites return Statement. Sollte entweder async/await verwenden oder die Promise zurückgeben.

Example Exercises

JavaScript Fundamentals

Basic Array Manipulation Write a function that takes an array of numbers and returns a new array containing only the even numbers, doubled.

```
// Example solution
function processArray(numbers) {
    return numbers
        .filter(num => num % 2 === 0)
        .map(num => num * 2);
}

// Test
console.log(processArray([1, 2, 3, 4, 5, 6])); // [4, 8, 12]
```

Closure Implementation Create a function that generates unique IDs with a given prefix. Each call should return a new ID with an incrementing number.

```
// Example solution
function createIdGenerator(prefix) {
    let counter = 0;
    return function() {
        counter++;
        return `${prefix}${counter}`;
    };
}

// Test
const generateUserId = createIdGenerator('user_');
console.log(generateUserId()); // "user_1"
console.log(generateUserId()); // "user_2"
```

Async Programming Write an async function that fetches user data from two different endpoints and combines them. Handle potential errors appropriately.

```
async function getUserData(userId) {
    try {
        const [profile, posts] = await Promise.all([
            fetch(`/api/profile/${userId}`).then(r =>
                r.json()),
            fetch(`/api/posts/${userId}`).then(r =>
                r.json())
        ]);

        return {
            ...profile,
            posts: posts
        };
    } catch (error) {
        console.error('Failed to fetch user data:',
            error);
        throw new Error('Failed to load user data');
    }
}
```

DOM Manipulation

Dynamic List Creation Write a function that takes an array of items and creates a numbered list in the DOM. Add a button to each item that removes it from the list.

```
function createList(items, containerId) {
    const container =
        document.getElementById(containerId);
    const ul = document.createElement('ul');

    items.forEach((item, index) => {
        const li = document.createElement('li');
        li.textContent = `${index + 1}. ${item} `;

        const button =
            document.createElement('button');
        button.textContent = 'Remove';
        button.onclick = () => li.remove();

        li.appendChild(button);
        ul.appendChild(li);
    });

    container.appendChild(ul);
}
```

Component Implementation

Form Component Create a form component in SuiWeb that handles user input with validation and submits data to a server.

```
const UserForm = () => {
    const [formData, setFormData] = useState({
        username: '',
        email: ''
    });
    const [errors, setErrors] = useState({});

    const validate = () => {
        const newErrors = {};
        if (!formData.username) {
            newErrors.username = 'Username is
                required';
        }
        if (!formData.email.includes('@')) {
            newErrors.email = 'Valid email is
                required';
        }
        setErrors(newErrors);
        return Object.keys(newErrors).length === 0;
    };

    const handleSubmit = async (e) => {
        e.preventDefault();
        if (!validate()) return;

        try {
            await fetch('/api/users', {
                method: 'POST',
                headers: {'Content-Type':
                    'application/json'},
                body: JSON.stringify(formData)
            });
        } catch (error) {
            setErrors({submit: 'Failed to submit
                form'});
        }
    };

    return [
        "form",
        {onsubmit: handleSubmit},
        ["div",
            ["label", {for: "username"}, "Username:"],
            ["input", {
                id: "username",
                value: formData.username,
                oninput: (e) => setFormData({
                    ...formData,
                    username: e.target.value
                })
            }],
            errors.username && ["span", {class:
                "error"}, errors.username]
        ],
        ["div",
            ["label", {for: "email"}, "Email:"],
            ["input", {
                id: "email",
                type: "email",
                value: formData.email,
                oninput: (e) => setFormData({
                    ...formData,
                    email: e.target.value
                })
            }],
            errors.email && ["span", {class: "error"},
                errors.email]
```

API Implementation

REST API with Express Create a simple REST API for a todo list with Express.js, including error handling and basic validation.

```
const express = require('express');
const app = express();
app.use(express.json());

let todos = [];

// Get all todos
app.get('/api/todos', (req, res) => {
    res.json(todos);
});

// Create new todo
app.post('/api/todos', (req, res) => {
    const { title } = req.body;

    if (!title) {
        return res.status(400).json({
            error: 'Title is required'
        });
    }

    const todo = {
        id: Date.now(),
        title,
        completed: false
    };

    todos.push(todo);
    res.status(201).json(todo);
});

// Update todo
app.patch('/api/todos/:id', (req, res) => {
    const { id } = req.params;
    const { completed } = req.body;

    const todo = todos.find(t => t.id ===
        parseInt(id));

    if (!todo) {
        return res.status(404).json({
            error: 'Todo not found'
        });
    }

    todo.completed = completed;
    res.json(todo);
});

app.use((err, req, res, next) => {
    console.error(err);
    res.status(500).json({
        error: 'Internal server error'
    });
});

app.listen(3000);
```

State Management Implement a shopping cart component that manages products, quantities, and total price calculation.

```
const ShoppingCart = () => {
    const [items, setItems] = useState([]);

    const addItem = (product) => {
        setItems(current => {
            const existing = current.find(
                item => item.id === product.id
            );

            if (existing) {
                return current.map(item =>
                    item.id === product.id
                        ? {...item, quantity:
                            item.quantity + 1}
                        : item
                );
            }

            return [...current, {...product, quantity:
                1}];
        });
    };

    const removeItem = (productId) => {
        setItems(current =>
            current.filter(item => item.id !==
                productId)
        );
    };

    const total = items.reduce(
        (sum, item) => sum + item.price *
            item.quantity,
        0
    );

    return [
        "div",
        ["h2", "Shopping Cart"],
        ["ul",
            ...items.map(item => [
                "li",
                ["span", `${item.name} x
                    ${item.quantity}`],
                ["span", `$${item.price *
                    item.quantity}`],
                ["button",
                    {onclick: () =>
                        removeItem(item.id)},
                    "Remove"
                ]
            ])
        ],
        ["div", `Total: $${total.toFixed(2)}`]
    ];
};
```

Browser APIs and Events

Custom Event System Implement a publish/subscribe system using browser events.

```
class EventBus {
    constructor() {
        this.eventTarget = new EventTarget();
    }

    publish(eventName, data) {
        const event = new CustomEvent(eventName, {
            detail: data,
            bubbles: true
        });
        this.eventTarget.dispatchEvent(event);
    }

    subscribe(eventName, callback) {
        const handler = (e) => callback(e.detail);
        this.eventTarget.addEventListener(eventName,
            handler);
        return () => {
            this.eventTarget.removeEventListener(eventName,
                handler);
        };
    }
}

// Usage
const bus = new EventBus();
const unsubscribe = bus.subscribe('userLoggedIn',
    (user) => {
    console.log(`Welcome, ${user.name}!`);
});

bus.publish('userLoggedIn', { name: 'John' });
unsubscribe(); // Cleanup
```

Drag and Drop Implement a simple drag and drop system for list items.

```
function initDragAndDrop(containerId) {
    const container =
        document.getElementById(containerId);
    let draggedItem = null;

    container.addEventListener('dragstart', (e) => {
        draggedItem = e.target;
        e.target.classList.add('dragging');
    });

    container.addEventListener('dragend', (e) => {
        e.target.classList.remove('dragging');
    });

    container.addEventListener('dragover', (e) => {
        e.preventDefault();
        const afterElement =
            getDragAfterElement(container, e.clientY);
        if (afterElement) {
            container.insertBefore(draggedItem,
                afterElement);
        } else {
            container.appendChild(draggedItem);
        }
    });

    function getDragAfterElement(container, y) {
        const draggableElements = [
            ...container.querySelectorAll('li:not(.dragging)')
        ];

        return draggableElements.reduce((closest,
            child) => {
            const box = child.getBoundingClientRect();
            const offset = y - box.top - box.height /
                2;

            if (offset < 0 && offset > closest.offset)
                {
                return { offset, element: child };
            }
            return closest;
        }, { offset: Number.NEGATIVE_INFINITY
            }).element;
    }
}
```

Data Manipulation and Algorithms

Deep Object Comparison Implement a function that deeply compares two objects for equality.

```
function deepEqual(obj1, obj2) {
    // Handle primitives and null
    if (obj1 === obj2) return true;
    if (obj1 == null || obj2 == null) return false;
    if (typeof obj1 !== 'object' || typeof obj2 !==
        'object')
        return false;

    const keys1 = Object.keys(obj1);
    const keys2 = Object.keys(obj2);

    if (keys1.length !== keys2.length) return false;

    return keys1.every(key => {
        if (!keys2.includes(key)) return false;
        return deepEqual(obj1[key], obj2[key]);
    });
}

// Test
const obj1 = {
    a: 1,
    b: { c: 2, d: [3, 4] },
    e: null
};
const obj2 = {
    a: 1,
    b: { c: 2, d: [3, 4] },
    e: null
};
console.log(deepEqual(obj1, obj2)); // true
```

Custom Promise Implementation Create a simplified version of the Promise API.

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

        const resolve = (value) => {
            if (this.state === 'pending') {
                this.state = 'fulfilled';
                this.value = value;
                this.handlers.forEach(handler =>
                    this.handle(handler));
            }
        };

        const reject = (error) => {
            if (this.state === 'pending') {
                this.state = 'rejected';
                this.value = error;
                this.handlers.forEach(handler =>
                    this.handle(handler));
            }
        };

        try {
            executor(resolve, reject);
        } catch (error) {
            reject(error);
        }
    }

    handle(handler) {
        if (this.state === 'pending') {
            this.handlers.push(handler);
        } else {
            const cb = this.state === 'fulfilled'
                ? handler.onSuccess
                : handler.onFail;
            if (cb) {
                try {
                    const result = cb(this.value);
                    handler.resolve(result);
                } catch (error) {
                    handler.reject(error);
                }
            }
        }
    }

    then(onSuccess, onFail) {
        return new MyPromise((resolve, reject) => {
            this.handle({
                onSuccess: onSuccess || (val => val),
                onFail: onFail || (err => { throw err;
                    }),
                resolve,
                reject
            });
        });
    }

    catch(onFail) {
        return this.then(null, onFail);
    }
}

// Usage
new MyPromise((resolve, reject) => {
    setTimeout(() => resolve('Success!'), 1000);
```

Component Testing

Unit Testing Components Write tests for a form component using Jasmine.

```
describe('UserForm Component', () => {
    let form;

    beforeEach(() => {
        form = new UserForm();
    });

    it('should initialize with empty values', () => {
        expect(form.state.username).toBe('');
        expect(form.state.email).toBe('');
        expect(Object.keys(form.state.errors)).toHaveSize(0)
    });

    it('should validate email format', () => {
        form.state.email = 'invalid-email';
        const isValid = form.validate();

        expect(isValid).toBe(false);
        expect(form.state.errors.email)
            .toContain('Valid email is required');
    });

    it('should submit form with valid data', async ()
        => {
        form.state.username = 'testuser';
        form.state.email = 'test@example.com';

        spyOn(window, 'fetch').and.returnValue(
            Promise.resolve({ ok: true })
        );

        await form.handleSubmit();

        expect(window.fetch).toHaveBeenCalledWith(
            '/api/users',
            jasmine.any(Object)
        );
        expect(form.state.errors).toEqual({});
    });
});
```