Zestaw Zadań: Weryfikacja i utrwalenie wiedzy o JavaScript

Informacje Ogólne

Materiał źródłowy:	Kompleksowy przegląd JavaScript - od podstaw (zmienne, typy danych, operatory) przez funkcje, DOM, zdarzenia, aż po asynchroniczność, Promises i API
Poziom:	Mieszany (podstawowy → zaawansowany)
Szacowany czas:	8-10 godzin (wszystkie zadania)
Typ zestawu:	Powiązane - każde zadanie rozwija struktury HTML/CSS z poprzednich zestawów
Wymagania wstępne:	Ukończone zadania z zestawów HTML i CSS lub równoważna wiedza

Wprowadzenie do JavaScript

Czym jest JavaScript?

JavaScript to dynamiczny język programowania, który pozwala dodać interaktywność do stron internetowych. W przeciwieństwie do HTML (struktura) i CSS (wygląd), JavaScript odpowiada za zachowanie strony - reakcje na akcje użytkownika, manipulacje treścią, komunikację z serwerami.

JavaScript jest językiem jednowątkowym z asynchronicznym modelem wykonania opartym na pętli zdarzeń (event loop). Działa zarówno w przeglądarce (frontend) jak i na serwerze (Node.js).

Trzy sposoby dodawania JavaScript do HTML

- Inline JavaScript - bezpośrednio w atrybutach zdarzeń HTML: onclick="alert('Cześć!')"
- Internal JavaScript - w znaczniku <script> wewnątrz dokumentu HTML
- External JavaScript - w osobnym pliku .js podłączonym do HTML (zalecane!)

> 💡 **Najlepsza praktyka:** Zawsze używaj zewnętrznego JavaScript (osobny plik .js). Pozwala to na lepsze cache'owanie, separację kodu i łatwiejsze utrzymanie projektu.

Najczęstsze problemy i pułapki JavaScript

1. var vs let vs const

JavaScript ma trzy sposoby deklarowania zmiennych: var (funkcyjny zasięg, hoisting), let (blokowy zasięg, można zmienić), const (blokowy zasięg, nie można zmienić referencji).

> ⚠️ **Ostrzeżenie:** Unikaj var! Może prowadzić do niespodziewanych błędów przez hoisting i zasięg funkcyjny. Używaj const domyślnie, let gdy potrzebujesz zmienić wartość.

2. Koercja typów (Type Coercion)

JavaScript automatycznie konwertuje typy podczas porównywania: '5' == 5 → true, '5' === 5 → false, [] == false → true

> 📌 **Zasada:** Zawsze używaj === (ściśle równy) i !== (ściśle różny) zamiast == i !=. Unikniesz niespodzianek związanych z koercją typów.

3. Truthy i Falsy

Tylko 6 wartości jest falsy: false, 0, "" (pusty string), null, undefined, NaN. Wszystko inne jest truthy, w tym [], {}, "0", "false"!

> 💡 **Wskazówka:** Aby sprawdzić czy tablica jest pusta, używaj array.length === 0, nie !array (pusta tablica jest truthy!).

4. Kontekst this

Wartość this zmienia się w zależności od kontekstu: w metodzie obiektu → obiekt, w funkcji regularnej → window/undefined, w arrow function → dziedziczony z zewnętrznego zasięgu.

> 📌 **Zasada:** Arrow functions nie mają własnego this - dziedziczą go z otaczającego kontekstu. To ważne przy event handlerach i callbackach!

5. Asynchroniczność i Callback Hell

JavaScript obsługuje operacje asynchroniczne przez callbacks, Promises i async/await. Zagnieżdżone callbacks tworzą "piramidę zagłady" - trudny do czytania kod.

> 💡 **Wskazówka:** Preferuj async/await nad zagnieżdżone callbacks. Zawsze obsługuj błędy przez try/catch lub .catch().

Zadanie 1: Pierwsze kroki z JavaScript

Poziom trudności:	1 🟢 (Podstawowy)
Szacowany czas:	30 minut
Punktacja:	15 punktów
Typ:	Praktyczne
Powiązanie:	Rozszerza Zadanie 1 z HTML (Anatomia dokumentu HTML)

Cel dydaktyczny

Zrozumienie podstaw JavaScript: zmienne, typy danych, operatory i konsola przeglądarki.

Polecenie

Wykorzystaj szablon dokumentu HTML5 stworzony w Zadaniu 1 (HTML):

Część A - Konfiguracja (3 pkt)

1. Stwórz plik script.js w tym samym folderze co HTML
2. Podłącz plik JavaScript do HTML przed zamknięciem </body>
3. Sprawdź czy działa: console.log('Skrypt załadowany!')

Część B - Zmienne i typy danych (7 pkt)

Stwórz zmienne demonstracyjne każdego typu danych:

1. String - Twoje imię
2. Number - Twój wiek
3. Boolean - Czy jesteś studentem
4. Array - Lista 3 ulubionych języków programowania
5. Object - Obiekt z Twoimi danymi (imię, wiek, miasto)
6. null i undefined - Zadeklaruj zmienne z tymi wartościami
7. Użyj typeof dla każdej zmiennej i wyświetl wyniki w konsoli

Część C - Operatory (5 pkt)

1. Wykonaj operacje arytmetyczne: +, -, *, /, %, **
2. Porównaj == z === na przykładzie '5' i 5
3. Użyj operatorów logicznych: &&, ||, !
4. Dodaj komentarze wyjaśniające każdy przykład

Kryteria oceny

- Poprawne podłączenie skryptu do HTML (3 pkt)
- Wszystkie typy danych z prawidłowym użyciem typeof (4 pkt)
- Operatory arytmetyczne i logiczne (4 pkt)
- Rozróżnienie == vs === (2 pkt)
- Komentarze i czytelność kodu (2 pkt)

Wskazówki

- Otwórz DevTools (F12) → zakładka Console aby zobaczyć console.log()
- Użyj let lub const zamiast var
- typeof null zwraca "object" - to znany bug w JavaScript!

Zadanie 2: Instrukcje warunkowe i pętle

Poziom trudności:	2 🟡 (Średniozaawansowany)
Szacowany czas:	45 minut
Punktacja:	20 punktów
Typ:	Praktyczne
Powiązanie:	Rozszerza Zadanie 2 z HTML (Mistrz formatowania tekstu)

Cel dydaktyczny

Opanowanie sterowania przepływem programu: instrukcje warunkowe if/else/switch i różne typy pętli.

Polecenie

Część A - Instrukcje warunkowe (8 pkt)

4. Stwórz funkcję sprawdzającą czy liczba jest parzysta/nieparzysta
5. Stwórz kalkulator ocen (0-100 punktów → ocena słowna)
6. Użyj switch do wyświetlenia dnia tygodnia na podstawie numeru (1-7)
7. Zastosuj operator trójargumentowy (ternary) do sprawdzenia pełnoletności

Część B - Pętle (8 pkt)

8. Użyj for do wyświetlenia liczb od 1 do 10
9. Użyj while do odliczania od 10 do 0
10. Użyj for...of do iteracji po tablicy
11. Użyj for...in do iteracji po właściwościach obiektu
12. Zastosuj break i continue w przykładowych pętlach

Część C - Zadanie praktyczne (4 pkt)

Napisz program generujący tabliczkę mnożenia (1-10) i wyświetlający ją w konsoli w czytelnym formacie.

Kryteria oceny

- Poprawne instrukcje warunkowe (4 pkt)
- Switch z break i default (2 pkt)
- Operator trójargumentowy (2 pkt)
- Wszystkie typy pętli (6 pkt)
- break i continue (2 pkt)
- Tabliczka mnożenia (4 pkt)

Wskazówki

- for...of iteruje po wartościach (tablice, stringi)
- for...in iteruje po kluczach (obiekty) - unikaj dla tablic!
- Pamiętaj o break w switch - inaczej wykona się też następny case

Zadanie 3: Funkcje i zakres zmiennych

Poziom trudności:	2 🟡 (Średniozaawansowany)
Szacowany czas:	50 minut
Punktacja:	25 punktów
Typ:	Praktyczne
Powiązanie:	Rozszerza Zadanie 3 z HTML (Listy i tabele w praktyce)

Cel dydaktyczny

Opanowanie różnych sposobów definiowania funkcji, parametrów, wartości zwracanych oraz zrozumienie zasięgów i domknięć (closures).

Polecenie

Część A - Deklaracje funkcji (8 pkt)

Stwórz te same funkcjonalności na 4 różne sposoby:

8. Function Declaration: function nazwaFunkcji() {}
9. Function Expression: const funkcja = function() {}
10. Arrow Function: const funkcja = () => {}
11. IIFE: (function() { /* kod */ })()

Część B - Parametry i zwracanie wartości (9 pkt)

13. Stwórz funkcję z parametrami domyślnymi
14. Użyj parametru rest (...args) do przyjęcia dowolnej liczby argumentów
15. Zwróć obiekt z wieloma wartościami
16. Stwórz funkcję przyjmującą callback
17. Stwórz funkcję zwracającą inną funkcję (higher-order function)

Część C - Zakres i domknięcia (8 pkt)

5. Zademonstruj różnicę między var, let i const w pętli for
6. Stwórz przykład pokazujący zasięg blokowy
7. Stwórz closure (licznik z zapamiętanym stanem)
8. Wyjaśnij w komentarzu jak działa domknięcie

Kryteria oceny

- Wszystkie typy deklaracji funkcji (6 pkt)
- Parametry: domyślne i rest (4 pkt)
- Callbacks i higher-order functions (5 pkt)
- Demonstracja zasięgów var/let/const (4 pkt)
- Closures z wyjaśnieniem (4 pkt)
- Czytelność i komentarze (2 pkt)

Wskazówki

- Arrow functions nie mają własnego this ani arguments
- Closure "zamyka" zmienne z zewnętrznego zasięgu
- Klasyczny problem: var w pętli z setTimeout - użyj let

Zadanie 4: Manipulacja DOM - Interaktywna galeria

Poziom trudności:	3 🟠 (Zaawansowany)
Szacowany czas:	90 minut
Punktacja:	30 punktów
Typ:	Praktyczne
Powiązanie:	Rozszerza Zadanie 4 z HTML/CSS (Galeria multimedialna)

Cel dydaktyczny

Opanowanie manipulacji DOM: wybieranie elementów, modyfikacja treści, stylowanie, obsługa zdarzeń oraz tworzenie i usuwanie elementów dynamicznie.

Polecenie

Weź galerię multimedialną z Zadania 4 (HTML/CSS) i dodaj interaktywność:

Część A - Wybieranie elementów (6 pkt)

12. Użyj getElementById do pobrania kontenera galerii
13. Użyj querySelectorAll do pobrania wszystkich obrazków
14. Użyj querySelector z selektorami CSS
15. Użyj closest do nawigacji po drzewie DOM w górę

Część B - Modyfikacja elementów (10 pkt)

18. Zmień tytuł galerii dynamicznie (textContent, innerHTML)
19. Dodaj/usuń klasy CSS na hover (classList.add/remove/toggle)
20. Zmień style inline przez element.style
21. Modyfikuj atrybuty (setAttribute, getAttribute)
22. Użyj data-* atrybutów do przechowywania informacji o obrazkach

Część C - Lightbox (14 pkt)

Stwórz w pełni funkcjonalny lightbox:

9. Kliknięcie w obrazek otwiera overlay z powiększonym obrazem
10. Przyciski Poprzedni/Następny do nawigacji
11. Przycisk X lub kliknięcie poza obrazem zamyka lightbox
12. Obsługa klawiatury: strzałki do nawigacji, Escape do zamknięcia
13. Wyświetlanie tytułu/opisu obrazka pod powiększeniem

Kryteria oceny

- Metody wybierania elementów (5 pkt)
- Modyfikacja treści i atrybutów (5 pkt)
- Manipulacja klasami CSS (4 pkt)
- Lightbox - podstawowa funkcjonalność (8 pkt)
- Lightbox - nawigacja i klawiatura (6 pkt)
- Czytelność kodu i komentarze (2 pkt)

Wskazówki

- Event delegation: dodaj jeden listener do rodzica zamiast wielu do dzieci

- e.target to kliknięty element, e.currentTarget to element z listenerem
- e.stopPropagation() zatrzymuje bąbelkowanie

Zadanie 5: Formularze i walidacja

Poziom trudności:	3 🟠 (Zaawansowany)
Szacowany czas:	90 minut
Punktacja:	30 punktów
Typ:	Praktyczne
Powiązanie:	Rozszerza Zadanie 5 z HTML/CSS (Formularz kontaktowy)

Cel dydaktyczny

Opanowanie obsługi formularzy: pobieranie wartości, walidacja danych po stronie klienta, obsługa zdarzeń formularza i wyświetlanie komunikatów.

Polecenie

Weź formularz kontaktowy z Zadania 5 (HTML/CSS) i dodaj kompleksową walidację JavaScript:

Część A - Pobieranie danych (6 pkt)

16. Pobierz wartości ze wszystkich typów pól (text, email, textarea, select, radio, checkbox)
17. Użyj FormData API do zbierania danych
18. Wyświetl zebrane dane w konsoli jako obiekt

Część B - Walidacja (14 pkt)

Zaimplementuj walidację dla każdego pola:

23. Imię: minimum 2 znaki, tylko litery (z polskimi znakami)
24. Email: poprawny format (użyj regex)
25. Telefon: opcjonalny, ale jeśli podany - 9 cyfr
26. Wiadomość: minimum 10 znaków
27. Checkbox zgody: musi być zaznaczony
28. Walidacja w czasie rzeczywistym (on blur/input) oraz przy wysyłce

Część C - UX walidacji (10 pkt)

14. Wyświetl komunikaty błędów pod każdym polem
15. Dodaj wizualne oznaczenie błędnych pól (czerwona ramka)
16. Dodaj wizualne oznaczenie poprawnych pól (zielona ramka)
17. Przy wysyłce - zablokuj przycisk i pokaż "Wysyłanie..."
18. Po (symulowanej) wysyłce - pokaż komunikat sukcesu i wyczyść formularz

Kryteria oceny

- Pobieranie danych z różnych typów pól (5 pkt)
- FormData API (2 pkt)
- Walidacja wszystkich pól (8 pkt)
- Regex dla email i telefonu (3 pkt)
- Walidacja w czasie rzeczywistym (4 pkt)
- Komunikaty i wizualna informacja zwrotna (5 pkt)
- Obsługa wysyłki i reset (3 pkt)

Wskazówki

- Regex dla email: /^[^\s@]+@[^\s@]+\.[^\s@]+$/
- e.preventDefault() w onsubmit zapobiega przeładowaniu strony
- Symulacja wysyłki: setTimeout(() => { /* sukces */ }, 1500)

Zadanie 6: Asynchroniczność i API

Poziom trudności:	3 🟠 (Zaawansowany)
Szacowany czas:	90 minut
Punktacja:	30 punktów
Typ:	Praktyczne
Powiązanie:	Może być połączone z dowolnym poprzednim zadaniem

Cel dydaktyczny

Zrozumienie asynchroniczności w JavaScript: Promises, async/await, Fetch API oraz obsługa błędów przy komunikacji z serwerem.

Polecenie

Część A - Promises (8 pkt)

19. Stwórz własną Promise symulującą ładowanie danych (setTimeout)
20. Użyj .then(), .catch() i .finally()
21. Połącz kilka Promises za pomocą Promise.all()
22. Użyj Promise.race() do implementacji timeout

Część B - Async/Await (8 pkt)

29. Przepisz kod z Promises na async/await
30. Użyj try/catch do obsługi błędów
31. Stwórz sekwencyjne wykonanie kilku operacji asynchronicznych
32. Stwórz równoległe wykonanie z await Promise.all()

Część C - Fetch API (14 pkt)

Stwórz aplikację pobierającą dane z publicznego API (JSONPlaceholder):

19. Pobierz listę użytkowników z /users
20. Wyświetl dane w tabeli HTML (stworzonej dynamicznie)
21. Dodaj loader (animacja ładowania) podczas pobierania
22. Obsłuż błędy (brak sieci, błąd serwera) z odpowiednim komunikatem
23. Dodaj przycisk Refresh do odświeżenia danych
24. Kliknięcie w użytkownika pokazuje jego posty (/users/{id}/posts)

Kryteria oceny

- Własne Promises z resolve/reject (4 pkt)
- Promise.all i Promise.race (3 pkt)
- Async/await z try/catch (5 pkt)
- Fetch API - pobieranie i wyświetlanie (6 pkt)
- Loader i obsługa błędów (4 pkt)
- Nested API calls (posty użytkownika) (5 pkt)
- Czytelność i struktura kodu (3 pkt)

Wskazówki

- Fetch zwraca Promise, ale response.json() też zwraca Promise!

- Sprawdzaj response.ok - fetch nie rzuca błędu dla 404/500
- Promise.all czeka na wszystkie, Promise.race na pierwszą

Zadanie 7: Projekt kompleksowy - SPA

Poziom trudności:	4 🔴 (Ekspert)
Szacowany czas:	180 minut
Punktacja:	50 punktów
Typ:	Syntetyczne/Twórcze
Powiązanie:	Łączy wszystkie elementy z zestawów HTML, CSS i JavaScript

Cel dydaktyczny

Połączenie wszystkich umiejętności w kompleksowy projekt: architektura aplikacji, organizacja kodu, zarządzanie stanem, routing i integracja z API.

Polecenie

Stwórz aplikację SPA (Single Page Application) - Manager Zadań:

Część A - Architektura (10 pkt)

23. Podziel kod na moduły (ES6 modules lub wzorzec modułu)
24. Stwórz strukturę folderów: /js, /css, /components
25. Zaimplementuj prosty system routingu (hash-based: #/tasks, #/completed)
26. Stwórz główny kontroler aplikacji (app.js)

Część B - Funkcjonalności CRUD (20 pkt)

33. Create: Dodawanie nowych zadań z walidacją (tytuł, priorytet, kategoria)
34. Read: Lista zadań z filtrowaniem (wszystkie/aktywne/ukończone)
35. Update: Edycja istniejących zadań (inline editing)
36. Delete: Usuwanie zadań z potwierdzeniem
37. Toggle: Oznaczanie jako wykonane
38. Kategorie/tagi dla zadań (praca, dom, zakupy, inne)
39. Wyszukiwanie zadań po tytule
40. Sortowanie (priorytet, data, alfabetycznie)

Część C - Przechowywanie danych (10 pkt)

25. Zapisuj dane w LocalStorage
26. Wczytuj dane przy starcie aplikacji
27. Synchronizuj zmiany automatycznie (debounce)
28. Dodaj eksport/import danych (JSON download/upload)

Część D - UI/UX (10 pkt)

1. Responsywny design (mobile-first)
2. Animacje CSS dla interakcji (dodawanie, usuwanie, toggle)
3. Dark mode (toggle z zapisem preferencji)
4. Skróty klawiszowe (n - nowe zadanie, Escape - zamknij modal)
5. Komunikaty dla użytkownika (toast notifications)

Kryteria oceny

- Architektura i organizacja kodu (8 pkt)

- Routing hash-based (4 pkt)
- CRUD operacje na zadaniach (10 pkt)
- Filtrowanie, wyszukiwanie, sortowanie (6 pkt)
- LocalStorage - zapis i odczyt (4 pkt)
- Eksport/import JSON (3 pkt)
- Responsywność (3 pkt)
- Dark mode (3 pkt)
- Animacje i UX (3 pkt)
- Czytelność kodu i komentarze (6 pkt)

Wskazówki

- Hash routing: window.addEventListener('hashchange', handleRoute)
- Debounce dla zapisu: nie zapisuj przy każdej zmianie, poczekaj 500ms
- Dark mode: użyj CSS custom properties i toggle klasy na <body>
- Toast notifications: stwórz komponent z auto-hide po 3 sekundach

Dodatkowe Informacje

Sugerowana kolejność wykonywania

Zadania są zaprojektowane sekwencyjnie - każde następne buduje na poprzednich:

Zadanie 1 → Zadanie 2 → Zadanie 3 → Zadanie 4 → Zadanie 5 → Zadanie 6 → Zadanie 7

Zadanie 6 (Asynchroniczność) może być połączone z dowolnym wcześniejszym zadaniem jako rozszerzenie.

Opcjonalne zadania rozszerzające

- Dodaj testy jednostkowe (Jest lub Mocha)
- Zaimplementuj Service Worker dla trybu offline (PWA)
- Dodaj Web Components dla komponentów wielokrotnego użytku
- Stwórz wersję TypeScript projektu końcowego
- Dodaj synchronizację z Firebase lub inną bazą NoSQL
- Zaimplementuj drag & drop do zmiany kolejności zadań

Pomocne zasoby

Dokumentacja i referencje

- MDN JavaScript: developer.mozilla.org/en-US/docs/Web/JavaScript
- JavaScript.info: javascript.info
- Can I Use: caniuse.com

Narzędzia do nauki

- FreeCodeCamp: freecodecamp.org
- Codecademy: codecademy.com
- Exercism: exercism.org/tracks/javascript

Publiczne API do ćwiczeń

- JSONPlaceholder: jsonplaceholder.typicode.com
- PokeAPI: pokeapi.co
- REST Countries: restcountries.com
- Open Weather API: openweathermap.org/api

Debugowanie

- Chrome DevTools: developer.chrome.com/docs/devtools
- Firefox Developer Tools

Walidacja i linting

- ESLint: eslint.org
- Prettier: prettier.io

Legenda poziomów trudności

Poziom	Emoji	Nazwa	Opis
1	🟢	Podstawowy	Sprawdzenie zrozumienia podstawowych koncepcji
2	🟡	Średniozaawansowany	Zastosowanie wiedzy w typowych sytuacjach
3	🟠	Zaawansowany	Analiza, synteza, rozwiązywanie złożonych problemów
4	🔴	Ekspert	Tworzenie nowej wiedzy, innowacyjne podejścia