

J a v a s c r i p t o d p i e r w s z e j l i n i j k i

Naucz się jak pisać gry, strony WWW i aplikacje internetowe!

Peter Ringel

Spis treści

Wprowadzenie do JavaScript	8
Historia języka i jego znaczenie.....	9
Gdzie można używać JavaScriptu.....	10
Pierwsze "Hello World".....	11
Konsola przeglądarki i podstawy debugowania.....	13
Dodawanie skryptów do strony HTML.....	14
Zmienne i typy danych	18
let, const i var	19
String, Number, Boolean.....	Błąd! Nie zdefiniowano zakładki.
undefined i null.....	Błąd! Nie zdefiniowano zakładki.
Konwersja typów.....	Błąd! Nie zdefiniowano zakładki.
Symbol i BigInt.....	Błąd! Nie zdefiniowano zakładki.
Przykłady deklaracji i inicjalizacji	Błąd! Nie zdefiniowano zakładki.
Operatory i wyrażenia.....	Błąd! Nie zdefiniowano zakładki.
Operatory arytmetyczne (+, -, *, /).....	Błąd! Nie zdefiniowano zakładki.
Operatory porównania (==, ===, !=, !==).....	Błąd! Nie zdefiniowano zakładki.
Operatory logiczne (&&, , !)	Błąd! Nie zdefiniowano zakładki.
Operator warunkowy (ternary)	Błąd! Nie zdefiniowano zakładki.
Operatory przypisania (+=, -=, *=, /=)	Błąd! Nie zdefiniowano zakładki.
Priorytet operatorów	Błąd! Nie zdefiniowano zakładki.
Instrukcje warunkowe	Błąd! Nie zdefiniowano zakładki.
if, else if, else	Błąd! Nie zdefiniowano zakładki.
switch/case	Błąd! Nie zdefiniowano zakładki.

Zagnieżdżone warunki	Błąd! Nie zdefiniowano zakładki.
Praktyczne przykłady z walidacją formularzy	Błąd! Nie zdefiniowano zakładki.
Obsługa błędów try/catch	Błąd! Nie zdefiniowano zakładki.
 Pętle i iteracje.....	Błąd! Nie zdefiniowano zakładki.
Pętla for.....	Błąd! Nie zdefiniowano zakładki.
Pętla while i do...while	Błąd! Nie zdefiniowano zakładki.
forEach dla tablic	Błąd! Nie zdefiniowano zakładki.
for...in dla obiektów	Błąd! Nie zdefiniowano zakładki.
for...of dla iterowalnych	Błąd! Nie zdefiniowano zakładki.
break i continue.....	Błąd! Nie zdefiniowano zakładki.
Funkcje.....	Błąd! Nie zdefiniowano zakładki.
Deklaracja funkcji.....	Błąd! Nie zdefiniowano zakładki.
Wyrażenia funkcyjne.....	Błąd! Nie zdefiniowano zakładki.
Funkcje strzałkowe.....	Błąd! Nie zdefiniowano zakładki.
Parametry i argumenty	Błąd! Nie zdefiniowano zakładki.
Zwracanie wartości.....	Błąd! Nie zdefiniowano zakładki.
Scope i closure	Błąd! Nie zdefiniowano zakładki.
Funkcje callback.....	Błąd! Nie zdefiniowano zakładki.
Rekurencja.....	Błąd! Nie zdefiniowano zakładki.
Tablice i ich metody.....	Błąd! Nie zdefiniowano zakładki.
Tworzenie i modyfikacja tablic	Błąd! Nie zdefiniowano zakładki.
push, pop, shift, unshift.....	Błąd! Nie zdefiniowano zakładki.
map, filter, reduce.....	Błąd! Nie zdefiniowano zakładki.
sort i reverse.....	Błąd! Nie zdefiniowano zakładki.
slice i splice	Błąd! Nie zdefiniowano zakładki.
Tablice wielowymiarowe	Błąd! Nie zdefiniowano zakładki.

Praktyczne przykłady przetwarzania danych**Błąd! Nie zdefiniowano zakładki.**

Obiekty i programowanie obiektowe..... **Błąd! Nie zdefiniowano zakładki.**

Tworzenie obiektów..... **Błąd! Nie zdefiniowano zakładki.**

Właściwości i metody **Błąd! Nie zdefiniowano zakładki.**

Prototypy i dziedziczenie **Błąd! Nie zdefiniowano zakładki.**

Klasy w ES6+ **Błąd! Nie zdefiniowano zakładki.**

Gettery i settery **Błąd! Nie zdefiniowano zakładki.**

This i wiązanie kontekstu..... **Błąd! Nie zdefiniowano zakładki.**

Przykłady projektowania obiektowego **Błąd! Nie zdefiniowano zakładki.**

Metody obsługi stringów..... **Błąd! Nie zdefiniowano zakładki.**

Konkatenacja i template literals..... **Błąd! Nie zdefiniowano zakładki.**

indexOf i includes **Błąd! Nie zdefiniowano zakładki.**

slice, substring, split **Błąd! Nie zdefiniowano zakładki.**

toLowerCase i toUpperCase **Błąd! Nie zdefiniowano zakładki.**

replace i replaceAll..... **Błąd! Nie zdefiniowano zakładki.**

Wyrażenia regularne..... **Błąd! Nie zdefiniowano zakładki.**

Praktyczne przykłady formatowania tekstu**Błąd! Nie zdefiniowano zakładki.**

Zdarzenia i interakcje z użytkownikiem..... **Błąd! Nie zdefiniowano zakładki.**

addEventListener **Błąd! Nie zdefiniowano zakładki.**

Typy zdarzeń (click, submit, keyup)..... **Błąd! Nie zdefiniowano zakładki.**

Event bubbling i capturing **Błąd! Nie zdefiniowano zakładki.**

preventDefault i stopPropagation **Błąd! Nie zdefiniowano zakładki.**

Tworzenie własnych zdarzeń...	Błąd! Nie zdefiniowano zakładki.
Praktyczne przykłady interakcji	Błąd! Nie zdefiniowano zakładki.
Manipulacja elementami DOM	Błąd! Nie zdefiniowano zakładki.
Selektory (getElementById, querySelector).....	Błąd! Nie zdefiniowano zakładki.
Modyfikacja zawartości (innerHTML, textContent).....	Błąd! Nie zdefiniowano zakładki.
Nawigacja po DOM	Błąd! Nie zdefiniowano zakładki.
Praktyczne przykłady dynamicznej zawartości	Błąd! Nie zdefiniowano zakładki.
Asynchroniczność i Promise	Błąd! Nie zdefiniowano zakładki.
setTimeout i setInterval.....	Błąd! Nie zdefiniowano zakładki.
Callbacks.....	Błąd! Nie zdefiniowano zakładki.
Promise i łańcuchy then.....	Błąd! Nie zdefiniowano zakładki.
async/await.....	Błąd! Nie zdefiniowano zakładki.
Obsługa błędów w kodzie asynchronicznym.....	Błąd! Nie zdefiniowano zakładki.
Przykłady równoległego wykonywania zadań	Błąd! Nie zdefiniowano zakładki.
Fetch API i komunikacja z serwerem	Błąd! Nie zdefiniowano zakładki.
GET, POST, PUT, DELETE	Błąd! Nie zdefiniowano zakładki.
Wysyłanie i odbieranie danych	Błąd! Nie zdefiniowano zakładki.
Headers i opcje fetch	Błąd! Nie zdefiniowano zakładki.
Obsługa odpowiedzi JSON.....	Błąd! Nie zdefiniowano zakładki.
Obsługa błędów sieciowych	Błąd! Nie zdefiniowano zakładki.
Przykłady komunikacji z REST API	Błąd! Nie zdefiniowano zakładki.
Storage i zarządzanie danymi	Błąd! Nie zdefiniowano zakładki.

localStorage i sessionStorage ...	Błąd! Nie zdefiniowano zakładki.
JSON.stringify i JSON.parse	Błąd! Nie zdefiniowano zakładki.
Cookies.....	Błąd! Nie zdefiniowano zakładki.
IndexedDB podstawy	Błąd! Nie zdefiniowano zakładki.
Przykłady persist.....	Błąd! Nie zdefiniowano zakładki.
Security i Data Privacy.....	Błąd! Nie zdefiniowano zakładki.
Performance Optimization	Błąd! Nie zdefiniowano zakładki.
Obsługa formularzy	Błąd! Nie zdefiniowano zakładki.
Walidacja pól	Błąd! Nie zdefiniowano zakładki.
Obsługa submit.....	Błąd! Nie zdefiniowano zakładki.
Dynamiczne formularze.....	Błąd! Nie zdefiniowano zakładki.
File upload	Błąd! Nie zdefiniowano zakładki.
Autouzupełnianie	Błąd! Nie zdefiniowano zakładki.
Przykłady formularzy kontaktowych.....	Błąd! Nie zdefiniowano zakładki.
Form State Management.....	Błąd! Nie zdefiniowano zakładki.
Debugowanie i narzędzia deweloperskie.....	Błąd! Nie zdefiniowano zakładki.
Console methods.....	Błąd! Nie zdefiniowano zakładki.
Breakpoints	Błąd! Nie zdefiniowano zakładki.
Network tab	Błąd! Nie zdefiniowano zakładki.
Performance profiling.....	Błąd! Nie zdefiniowano zakładki.
Memory leaks.....	Błąd! Nie zdefiniowano zakładki.
Przykłady debugowania typowych problemów	Błąd! Nie zdefiniowano zakładki.
Source Maps.....	Błąd! Nie zdefiniowano zakładki.
Optymalizacja i dobre praktyki	Błąd! Nie zdefiniowano zakładki.
Clean code.....	Błąd! Nie zdefiniowano zakładki.
DRY i SOLID	Błąd! Nie zdefiniowano zakładki.

Optymalizacja wydajności	Błąd! Nie zdefiniowano zakładki.
Code review	Błąd! Nie zdefiniowano zakładki.
Przykłady refaktoryzacji.....	Błąd! Nie zdefiniowano zakładki.
Error Handling.....	Błąd! Nie zdefiniowano zakładki.
Frameworki i biblioteki JavaScript	Błąd! Nie zdefiniowano zakładki.
Wprowadzenie do React.....	Błąd! Nie zdefiniowano zakładki.
Podstawy Vue.....	Błąd! Nie zdefiniowano zakładki.
jQuery w nowoczesnym JS	Błąd! Nie zdefiniowano zakładki.
Porównanie popularnych rozwiązań	Błąd! Nie zdefiniowano zakładki.
Kiedy używać frameworków	Błąd! Nie zdefiniowano zakładki.
Tworzenie pierwszej gry	Błąd! Nie zdefiniowano zakładki.
Game loop	Błąd! Nie zdefiniowano zakładki.
Obsługa kolizji.....	Błąd! Nie zdefiniowano zakładki.
Animacje.....	Błąd! Nie zdefiniowano zakładki.
Sterowanie.....	Błąd! Nie zdefiniowano zakładki.
Punktacja.....	Błąd! Nie zdefiniowano zakładki.
Przykład prostej gry platformowej	Błąd! Nie zdefiniowano zakładki.
Audio.....	Błąd! Nie zdefiniowano zakładki.
Debug i Testing.....	Błąd! Nie zdefiniowano zakładki.
Tworzenie aplikacji internetowej	Błąd! Nie zdefiniowano zakładki.
Architektura aplikacji	Błąd! Nie zdefiniowano zakładki.
Routing.....	Błąd! Nie zdefiniowano zakładki.
Stan aplikacji.....	Błąd! Nie zdefiniowano zakładki.
Komponenty wielokrotnego użytku	Błąd! Nie zdefiniowano zakładki.
Deployment.....	Błąd! Nie zdefiniowano zakładki.

Przykład Todo App **Błąd! Nie zdefiniowano zakładek.**
Testing..... **Błąd! Nie zdefiniowano zakładek.**

Wprowadzenie do JavaScript

JavaScript narodził się w bardzo ciekawym momencie rozwoju internetu. W 1995 roku programista Brendan Eich, pracujący dla firmy Netscape, otrzymał niezwykle ambitne zadanie - stworzenie języka programowania, który działałby bezpośrednio w przeglądarce internetowej. W ciągu zaledwie 10 dni stworzył pierwszą wersję języka, początkowo nazwanego LiveScript. Był to moment przełomowy, ponieważ strony internetowe przestały być tylko statycznymi dokumentami. Gdy firma Netscape nawiązała współpracę z firmą Sun Microsystems (twórcą popularnego wówczas języka Java), LiveScript zmienił nazwę na JavaScript - był to sprytny zabieg marketingowy, choć oba języki nie miały ze sobą wiele wspólnego. W 1997 roku, aby uporządkować rozwój języka, powstał standard ECMAScript. JavaScript stał się jego najbardziej znaną implementacją, a organizacja ECMA International zajęła się dalszym rozwojem specyfikacji.

Początkowo JavaScript był niedoceniany przez programistów. Służył głównie do prostych zadań, takich jak walidacja formularzy czy tworzenie efektów wizualnych na stronach. Jednak z biegiem lat jego możliwości znacząco wzrosły. Prawdziwa rewolucja nastąpiła w 2009 roku wraz z powstaniem Node.js - środowiska pozwalającego uruchamiać JavaScript poza przeglądarką. Nagle okazało się, że ten sam język może służyć do tworzenia zarówno interfejsu użytkownika, jak i logiki serwerowej. Rozwój technologii webowych sprawił, że pojawiły się potężne frameworki frontendowe, umożliwiające tworzenie zaawansowanych aplikacji działających w przeglądarce. JavaScript przestał być tylko dodatkiem do HTML-a, a stał się pełnoprawną platformą programistyczną.

Dziś JavaScript jest jednym z najpopularniejszych języków programowania na świecie. Według indeksu TIOBE regularnie znajduje się w pierwszej dziesiątce, a na GitHubie jest językiem z największą liczbą repozytoriów. Jego wszechstronność jest imponująca - służy do tworzenia interfejsów użytkownika, aplikacji serwerowych, aplikacji mobilnych (poprzez frameworki hybrydowe), a nawet aplikacji desktopowych. W zasadzie każda nowoczesna strona internetowa wykorzystuje JavaScript, a znajomość tego języka jest jednym z podstawowych wymagań na stanowiskach związanych z tworzeniem oprogramowania.

Gdzie można używać JavaScriptu

JavaScript to wyjątkowy język, który może działać w różnych środowiskach wykonawczych. Najpopularniejszym z nich jest oczywiście przeglądarka internetowa - każda nowoczesna przeglądarka posiada wbudowany silnik JavaScript, który interpretuje i wykonuje kod. Jednym z najważniejszych silników jest V8, stworzony przez Google, który napędza nie tylko przeglądarkę Chrome, ale stał się również podstawą Node.js. Node.js to środowisko, które pozwala uruchamiać JavaScript poza przeglądarką, bezpośrednio na komputerze czy serwerze. To właśnie dzięki Node.js JavaScript wyrwał się z ram przeglądarki i może być używany praktycznie wszędzie.

Różnorodność środowisk wykonawczych przekłada się na szerokie spektrum aplikacji, jakie możemy tworzyć w JavaScript. W przeglądarce możemy budować zarówno proste strony internetowe wzbogacone o interaktywne elementy, jak i złożone aplikacje webowe działające jak programy desktopowe. Po stronie serwera, dzięki Node.js, możemy tworzyć wydajne API RESTowe obsługujące tysiące zapytań. JavaScript świetnie sprawdza się też w tworzeniu aplikacji desktopowych - technologia Electron pozwala pakować aplikacje webowe w samodzielne programy działające na komputerze. W świecie mobile, dzięki technologiom jak React Native, możemy tworzyć natywne aplikacje mobilne używając znajomej składni JavaScriptu.

Do tworzenia aplikacji w JavaScript mamy do dyspozycji bogaty zestaw narzędzi programistycznych. Podstawowym narzędziem jest edytor kodu - od prostych jak Visual Studio Code po zaawansowane środowiska IDE jak WebStorm. Nieocenioną pomocą są narzędzia deweloperskie wbudowane w przeglądarki internetowe - pozwalają one na debugowanie kodu, analizę wydajności, podgląd zmiennych i struktury DOM. Console w narzędziach deweloperskich to pierwsze miejsce, gdzie zwykle testujemy nasz kod i sprawdzamy jego działanie. Większość edytorów oferuje kolorowanie składni, autouzupełnianie i podpowiedzi, co znacząco ułatwia pisanie kodu JavaScript.

```
// Przykład kodu, który możemy przetestować w konsoli przeglądarki
console.log("Hello World!");
// Prosty przykład interakcji z DOM-em
document.querySelector('button').addEventListener('click', () => {
```

```
    alert('Przycisk został kliknięty!');
  });
  // Przykład kodu Node.js
  const http = require('http');
  const server = http.createServer((req, res) => {
    res.writeHead(200);
    res.end('Hello World!');
  });
  server.listen(8080);
```

Pierwsze "Hello World"

Zacznijmy od stworzenia najprostszej strony internetowej z JavaScript. Kod JavaScript możemy umieścić w pliku HTML na kilka sposobów. Najpopularniejszym jest użycie znacznika script w sekcji head lub na końcu body dokumentu. Możemy też napisać kod JavaScript bezpośrednio między znacznikami script lub wskazać zewnętrzny plik .js. Oto przykład prostej strony z osadzonym kodem:

```
<!DOCTYPE html>
<html>
<head>
  <title>Moja pierwsza strona z JavaScript</title>
  <!-- JavaScript w sekcji head -->
  <script>
    // Tu możemy pisać kod JS
  </script>
</head>
<body>
  <h1>Hello World!</h1>
  <!-- JavaScript na końcu body (zalecane) -->
  <script src="moj-skrypt.js"></script>
</body>
</html>
```

JavaScript oferuje kilka podstawowych sposobów wyświetlania danych. Najprostszym jest użycie funkcji `console.log()`, która wyświetla informacje w konsoli przeglądarki - jest to podstawowe narzędzie podczas nauki i debugowania kodu. Funkcja `alert()` wyświetla okienko z komunikatem, które użytkownik musi zaakceptować. Z kolei `document.write()` pozwala wpisać treść bezpośrednio do dokumentu HTML. Oto przykłady użycia każdej z tych metod:

```
// Wyświetlenie w konsoli  
console.log("Hello World w konsoli!");  
// Wyświetlenie alertu  
alert("Hello World w alercie!");  
// Wyświetlenie na stronie  
document.write("Hello World na stronie!");  
// Możemy też wyświetlać liczby i wyrażenia  
console.log(2 + 2); // wyświetli 4  
console.log("Wynik:", 2 + 2); // wyświetli "Wynik: 4"
```

Pisząc kod JavaScript, warto od początku przywyknąć do pewnych zasad. Instrukcje kończymy średnikiem (;) - choć JavaScript często wybaczy nam jego brak, to jest to dobra praktyka. Kod warto formatować używając wcięć (najczęściej 2 lub 4 spacje) dla lepszej czytelności. Komentarze są niezwykle ważne - pomagają nam i innym zrozumieć kod. Możemy używać komentarzy jednoliniowych rozpoczynających się od `//` lub wieloliniowych między `/*` a `*/`. Przykład:

```
// To jest komentarz jednoliniowy  
console.log("Hello"); // To też jest komentarz jednoliniowy  
/* To jest komentarz wieloliniowy, który może zajmować wiele linii */  
/* Przykład wcięć w kodzie */  
if (true) {  
    console.log("Pierwsza linia");  
    console.log("Druga linia");  
}
```

Zacznijmy od tego, jak dotrzeć do konsoli przeglądarki. W większości nowoczesnych przeglądarek możemy ją otworzyć naciskając F12 lub kombinację Ctrl+Shift+I (Cmd+Option+I na MacOS). Możemy też kliknąć prawym przyciskiem myszy na stronie i wybrać "Zbadaj element" lub "Inspect", a następnie przejść do zakładki "Console". Konsola oferuje różne typy komunikatów, które różnią się wizualnie i ważnością: zwykłe logi (console.log), informacje (console.info), ostrzeżenia (console.warn) i błędy (console.error). Każdy z nich ma inne przeznaczenie i jest oznaczony inną ikoną, co pomaga w szybkiej identyfikacji typu komunikatu.

Konsola JavaScript oferuje wiele przydatnych metod, które ułatwiają debugowanie kodu. Podstawowa metoda console.log() wyświetla przekazane wartości w najprostszej formie. Metoda console.table() jest świetna do wyświetlania danych tabelarycznych - automatycznie formatuje tablice i obiekty w czytelną tabelę. Console.group() i console.groupEnd() pozwalają grupować powiązane komunikaty, co jest szczególnie przydatne przy debugowaniu większych fragmentów kodu. Metody console.time() i console.timeEnd() służą do mierzenia czasu wykonania kodu. Oto przykłady:

```
// Różne typy komunikatów
console.log("Zwykły komunikat");
console.info("Informacja");
console.warn("Ostrzeżenie!");
console.error("Błąd!");
// Wyświetlanie danych w tabeli
const users = [
  {name: "Jan", age: 25},
  {name: "Anna", age: 30}
];
console.table(users);
// Grupowanie komunikatów
console.group("Testowanie funkcji");
console.log("Test 1 passed");
console.log("Test 2 passed");
console.groupEnd();
```

```
// Mierzenie czasu  
console.time("Pętla");  
for (let i = 0; i < 1000000; i++) {}  
console.timeEnd("Pętla");
```

Kiedy w naszym kodzie występuje błąd, konsola jest pierwszym miejscem, gdzie powinniśmy szukać informacji o problemie. Błędy w JavaScript zawierają nazwę błędu (np. `SyntaxError`, `ReferenceError`, `TypeError`) oraz opis problemu. Stack trace (stos wywołań) pokazuje dokładną ścieżkę, która doprowadziła do błędu - od miejsca jego wystąpienia, przez wszystkie funkcje, które były po drodze wywołane. Najczęstsze błędy składniowe to brakujące nawiasy, przecinki czy średniki, literówki w nazwach zmiennych lub próba użycia zmiennej, która nie została zadeklarowana. Przykładowo:

```
// Przykład błędu składniowego  
function test() {  
  console.log("Start");  
  undefinedVariable; // ReferenceError: undefinedVariable is not  
defined  
  console.log("Koniec");  
}  
// Przykład błędu typu  
const number = 42;  
number.toUpperCase(); // TypeError: number.toUpperCase is not a  
function  
// Przykład błędu składni  
if (true { // SyntaxError: missing ) after condition  
  console.log("Błąd!");  
}
```

Dodawanie skryptów do strony HTML

Umieszczenie skryptu JavaScript w dokumencie HTML ma kluczowe znaczenie dla działania naszej strony. Możemy umieścić skrypt w sekcji head lub na końcu body - każde rozwiązanie ma swoje zalety. Skrypty w

head są ładowane wcześniej, ale mogą opóźnić renderowanie strony. Skrypty na końcu body pozwalają stronie załadować się szybciej, ale kod wykona się później. Atrybuty `async` i `defer` dają nam dodatkową kontrolę: `async` powoduje asynchroniczne ładowanie skryptu, a `defer` odkłada wykonanie do momentu załadowania całego HTML-a. Oto przykłady różnych sposobów dodawania skryptów:

```
<!DOCTYPE html>
<html>
<head>
  <!-- Standardowy skrypt w head -->
  <script src="pierwszy.js"></script>
  <!-- Skrypt asynchroniczny -->
  <script async src="drugi.js"></script>
  <!-- Skrypt z defer -->
  <script defer src="trzeci.js"></script>
</head>
<body>
  <!-- Zawartość strony -->
  <!-- Skrypt na końcu body -->
  <script src="czwarty.js"></script>
</body>
</html>
```

Przechowywanie kodu JavaScript w zewnętrznych plikach `.js` ma wiele zalet. Przede wszystkim poprawia to czytelność i utrzymanie kodu - łatwiej znaleźć i poprawić błędy w osobnych plikach. Dodatkowo, przeglądarka może cachować zewnętrzne pliki JS, co przyspieszy ładowanie strony przy kolejnych wizytach. Podczas linkowania plików możemy używać ścieżek względnych (relatywnych do aktualnego pliku HTML) lub bezwzględnych (pełny URL). Przykład struktury projektu i linkowania:

```
<!-- Ścieżka względna do pliku w tym samym folderze -->
<script src="skrypt.js"></script>
<!-- Ścieżka względna do pliku w podfolderze -->
<script src="js/skrypt.js"></script>
<!-- Ścieżka względna poziom wyżej -->
<script src="../skrypt.js"></script>
```

```
<!-- Ścieżka bezwzględna -->  
<script src="https://mojadowena.pl/skrypt.js"></script>
```

Przy organizacji kodu JavaScript warto kierować się kilkoma zasadami. Dla małych projektów jeden plik może wystarczyć, ale w większych projektach lepiej podzielić kod na mniejsze, tematyczne pliki. Ważna jest kolejność ładowania - jeśli jeden skrypt zależy od drugiego (np. używa jego funkcji), musi być załadowany po nim. Dobrą praktyką jest umieszczanie bibliotek zewnętrznych przed własnym kodem. Przykładowa struktura:

```
<!-- Najpierw biblioteki zewnętrzne -->  
<script src="vendor/jquery.js"></script>  
<script src="vendor/bootstrap.js"></script>  
<!-- Potem własne skrypty, w odpowiedniej kolejności -->  
<script src="js/utilities.js"></script>  
<script src="js/main.js"></script>  
<script src="js/app.js"></script>
```

Gdy skrypty nie ładują się poprawnie, pierwszym krokiem powinno być sprawdzenie konsoli przeglądarki. Błąd 404 oznacza, że plik nie został znaleziony - warto wtedy sprawdzić czy ścieżka jest poprawna i czy plik faktycznie istnieje w podanej lokalizacji. Jeśli skrypt się ładuje, ale nie działa poprawnie, może to oznaczać problem z kolejnością ładowania - na przykład próbujemy użyć funkcji z biblioteki, która jeszcze się nie załadowała. Pomocne jest sprawdzenie zakładki Network w narzędziach deweloperskich, która pokazuje kolejność i status ładowania wszystkich plików:

```
<!-- Przykład częstego błędu - próba użycia jQuery przed jego  
załadowaniem -->  
<script>  
  // To nie zadziała, bo jQuery jeszcze nie jest załadowane  
  $(document).ready(function() {  
    console.log("Strona gotowa!");  
  });  
</script>  
<script src="jquery.js">  
<!-- Poprawna kolejność -->
```

```
<script src="jquery.js"></script>
<script>
  // Teraz zadziała poprawnie
  $(document).ready(function() {
    console.log("Strona gotowa!");
  });
</script>
```


let, const i var
