

Programista JavaScript: programowanie od podstaw

Mariusz Poliński



Plan szkolenia

0. Wprowadzenie

- Kilka słów o nas i o naszym stopniu zaawansowania.
- Dlaczego zdecydowaliście się zapisać na ten kurs? Co chcielibyście z niego wynieść?
- Dlaczego JavaScript to solidny wybór do nauki programowania?
- Instalacja i konfiguracja środowiska do pracy (IDE).

Ćwiczenie praktyczne: stworzenie pierwszego projektu w VS Code.

1. HTML

szybkie przypomnienie oraz aspekty istotne dla programistów JavaScript

- Co to są znaczniki (tag)?
- Selektory tagów – id vs klasa vs element
- Semantyka HTML5 (drzewko DOM)
- Tips & Tricks: HTML5

Ćwiczenie praktyczne: Stworzenie strony frontowej bloga z takimi elementami jak: tytuł bloga, opis ogólny, lista wstępniaków z linkiem “Czytaj dalej” i stopka.



Plan szkolenia

2. Wstęp do JavaScriptu

Podstawy i teoria niezbędna do napisania pierwszego skryptu.

- Struktury i typy danych
- Zmienne, operatory, pętle, warunki
- Funkcje, klasy, obiekty
- OOP – Programowanie Obiektowe
- Przechowywanie danych (JSON, XML)
- VanillaJS vs dostępne frameworki
- Semantyka standardów ES5 vs ES6

3. JavaScript – praktycznie

- Animacje, manipulacje stroną (DOM).
- Tworzenie modali (okien dialogowych)
- Różne sposoby przechowywania danych na stronie (storages, cookies)
- Walidacja formularza
- Biblioteka jQuery oraz jQueryUI
- Prosta aplikacja SPA (Single Page Application).

Ćwiczenie praktyczne: Dodajemy sekcję komentowania w naszym blogu i prostą interakcję, walidację po kliknięciu na przycisk.



Plan szkolenia

4. Podstawy tworzenia aplikacji webowych

- Podstawy protokołu HTTP – jak działają strony internetowe?
- Czym jest Webserver, DNS, Certyfikat SSL?
- Czym jest CDN?
- Technologia AJAX
- Przygotowanie środowiska do pracy – Node.js
- Jak instalować moduły? Przydatne polecenia Node.js

5. Piszemy frontend do prostego bloga

- REST – komunikacja z backendem
- Dynamiczne ładowanie treści – bez przeładowywania strony
- Dodawanie nowych postów – Wyświetlanie postów – Edycja postów
- Dodawanie komentarzy
- Usuwanie komentarzy



Plan szkolenia

6. React — Jak zacząć?

- Tworzenie nowego projektu
- Struktura projektu
- Jak działają komponenty?
- Co składa się na komponent?
- Jak działa kompilator JSX?
- Routing — Napisanie prostej aplikacji

Ćwiczenie praktyczne: Przepiszemy naszego bloga używając komponentów biblioteki React.

Wprowadzenie



Wprowadzenie: Poznajmy się

Cześć, jestem (...) i zainteresowałem/am się tym kursem, bo (...).

To moje pierwsze zetknięcie z programowaniem / Kodowałem/am już wcześniej.

Wybrałem/am JavaScript, ponieważ (...)

Mój pierwszy skrypt/aplikacja to (...)





Wprowadzenie: Dlaczego JavaScript to solidny wybór do nauki programowania?

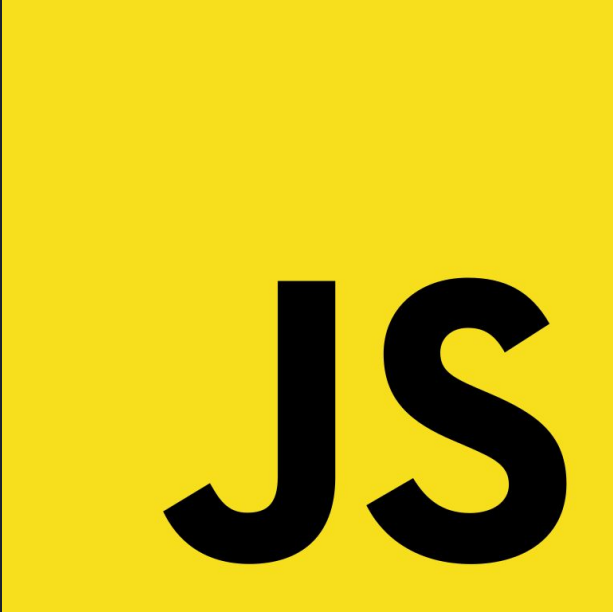
Elastyczny. Stworzony do pisania frontendu, używany także na backendzie, w aplikacjach mobilnych i desktopowych.

Bo każdy ma u siebie zainstalowane środowisko. **JS działa w przeglądarce.**

Ogromna społeczność i masa materiałów. JS to bardzo popularny język.

Przyjemny do pisania. Luźne typowanie i (zwykle) dość zwięzła i zrozumiała składnia.

JS to jeden z najbardziej poszukiwanych języków na rynku (szczególnie React i Node.js)



JS

Wprowadzenie: Dlaczego JavaScript to nie najlepszy język na start?

Czasem dziwna i nieintuicyjna składnia. Dlatego będziemy uczyć się nie używać "dziwnych" części JavaScriptu.

Luźne typowanie = więcej błędów. W JS dzięki luźnemu typowaniu pisze się łatwo i szybko. Bardzo szybko można wyprodukować koszmar.

```
let x = "5";  
let y = 3;  
console.log(x + y); // "53" zamiast 8
```

Nie uczy "solidnych fundamentów". W JS program można napisać na 20 różnych sposobów, z czego 15 będzie bardzo kiepskimi pomysłami.

Zbyt duży ekosystem = chaos na start. JS ma setki frameworków i bibliotek, początkującym łatwo się pogubić.

```
< "number" < true  
> 9999999999999999 > true===1  
< 10000000000000000 < false  
> 0.5+0.1==0.6 > (!+[!]+[!+![]]).length  
< true < 9  
> 0.1+0.2==0.3 > 9+"1"  
< false < "91"  
> Math.max() > 91-"1"  
< -Infinity < 90  
> Math.min() > []==0  
< Infinity < true  
> []+[]  
< ""  
> []+{}  
< "[object Object]"  
> {}+[]  
< 0  
> true+true+true===3  
< true  
> true-true
```



Wprowadzenie: Instalacja IDE

Dlaczego proponuję VS Code?

- Bo jest darmowy i łatwy w użyciu.
- Bo za pomocą wtyczek można dostosować go do różnych języków i stylu pracy.
- Bo spora część materiałów szkoleniowych będzie tworzona z jego użyciem.

Pobieranie: <https://code.visualstudio.com/download>



Wprowadzenie: Przydatne rozszerzenia IDE

1. **HTML CSS Support.** Automatycznie podpowiada klasy CSS w HTML, również z plików zewnętrznych.
2. **Live Server.** Uruchamia lokalny serwer i automatycznie odświeża stronę przy zapisie. Idealne do pracy z HTML/JS/CSS.

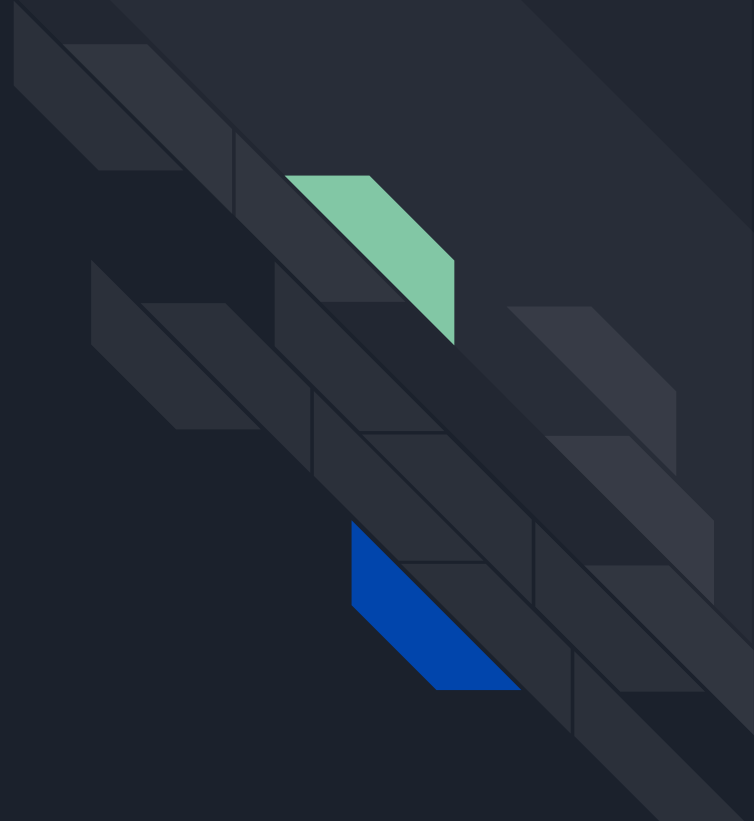




Wprowadzenie: Zadanie praktyczne

1. Stwórzcie katalog o nazwie `Projekty`
2. Wewnątrz stwórzcie katalog `kurs-js`
3. Otwórzcie katalog `kurs-js` w VS Code za pomocą opcji *Open Folder*
4. Stwórzcie plik `index.html` - to będzie główny plik naszego bloga
5. Utwórzcie podstawową strukturę dokumentu HTML
 - a. tag `<!DOCTYPE html>`
 - b. tag `<body>`
 - i. tag `<header>`
 1. tag `<h1>` z tytułem Waszego bloga
6. Pamiętajcie o zamknięciu otwartych tagów `</h1>` `</header>` `</body>`

HTML (i CSS)



HTML: Co to takiego?

HTML (HyperText Markup Language) to język, w którym tworzy się strony internetowe.

Za jego pomocą określamy strukturę treści na stronie – np. gdzie są nagłówki, akapity, obrazki, przyciski itp.

Strukturę dokumentu opisują tak zwane **znaczniki** (ang. **tagi**).





HTML: Quiz

Który znacznik służy do tworzenia linków?

- A) <link>
- B) <a>
- C) <href>
- D) <url>

Co robi ten kod?

```
<p><strong>Ważna</strong>  
informacja</p>
```

- A) Tworzy nagłówek
- B) Tworzy link
- C) Pogrubia słowo "Ważna"
- D) Wstawia nowy akapit

Który znacznik NIE jest semantyczny?

- A) <article>
- B) <section>
- C) <div>
- D) <header>

Jak poprawnie osadzić obrazek?

- A) image.jpg
- B) <image src="image.jpg">
- C)
- D) <pic src="image.jpg">

HTML: Co to są znaczniki (tagi)?

Tag HTML to blok, który służy do oznaczania różnych elementów na stronie.

Na przykład, możemy użyć tagu `<p>` do stworzenia akapitu, tagu `<h1>` do nagłówka, a tagu `` do obrazka.

Większość znaczników ma otwierający i zamykający tag.

```
1  <!DOCTYPE html>
2  <html>
3    <head>
4      <meta charset="UTF-8">
5      <title>Title goes here</title>
6    </head>
7    <body>
8
9    </body>
10 </html>
```




HTML: Identyfikacja tagów

Tagi HTML mogą być identyfikowane na różne sposoby. W tym celu używamy różnych atrybutów. Najczęściej spotykane to:

- **ID (id):** Atrybut id daje unikalną nazwę danemu elementowi na stronie. Ważne: id musi być unikalne na stronie – nie może się powtarzać!
- **Klasa (class):** Atrybut class pozwala przypisać elementom wspólną klasę, co umożliwia stosowanie stylów do wielu elementów na stronie. W przeciwieństwie do id, ta sama klasa może być przypisana wielu elementom.
- Tagi mogą być również wybierane bezpośrednio po nazwie elementu w CSS.
- Inne ważne i często używane atrybuty tagów: href (dla linków), src (dla obrazków i plików), alt dla obrazków,

```
1  <!DOCTYPE html>
2  <html>
3    <head>
4      <meta charset="UTF-8">
5      <title>Title goes here</title>
6    </head>
7    <body>
8
9    </body>
10 </html>
```



HTML: Zagnieżdżanie tagów

Znaczniki HTML mogą być zagnieżdżane w sobie, co pozwala na tworzenie bardziej złożonej struktury strony.

```
<div>  
  <h1>Witaj na mojej stronie!</h1>  
  <p>To jest akapit w obrębie <code>div</code>.</p>  
</div>
```

Uporządkowany zestaw tagów tworzy drzewo dokumentu.



HTML: Zadanie praktyczne

1. Stwórzcie plik index.html - to będzie główny plik naszego bloga
2. Utwórzcie podstawową strukturę dokumentu HTML
 - a. tag `<!DOCTYPE html>`
 - b. tag `<head>`
 - i. `<title>`
 - c. tag `<body>`
 - i. tag `<header>`
 1. tag `<h1>` z tytułem Waszego bloga
 - ii. tag `<div class="container">` Jako blok na listę wstępniaków
 1. tag `<p>` z krótkim podsumowaniem o czym jest blog
 2. tag `<article>` jako blok na pojedynczy wstępniak
 - a. tag `<h2>` jako tytuł artykułu we wstępniaku
 - b. tag `<p>` z treścią wstępniaka
 - c. tag `<a>` z linkiem Czytaj Dalej
 - iii. tag `<footer>` z informacjami o prawach autorskich.



HTML: Semantyka HTML5 i dlaczego to jest ważne

Semantyka oznacza używanie tagów, które opisują znaczenie i strukturę treści, a nie tylko wygląd.

Oznacza to, że tagi HTML5 pozwalają na lepsze rozumienie struktury strony przez przeglądarki, narzędzia do **SEO**, czytniki ekranu i inne systemy.

Zamiast używać samego `<div>`, możemy użyć tagów takich jak:

```
<header>, <footer>, <section>,  
<article>, <nav>, <main>, itd.
```



HTML: Drzewo DOM (Document Object Model)

Drzewo DOM to hierarchiczna struktura, która reprezentuje dokument HTML.

Wszystkie elementy HTML (tagi, atrybuty, tekst) są traktowane jako węzły w drzewie.

Dzięki DOM, JavaScript może dynamicznie modyfikować zawartość strony, np. dodawać, usuwać lub zmieniać elementy w czasie rzeczywistym.

```
<html>
  <body>
    <header>
      <h1>Witaj na mojej stronie!</h1>
    </header>
    <main>
      <p>To jest główny akapit.</p>
    </main>
  </body>
</html>
```

```
html
├─ body
│   ├── header
│   │   └─ h1 (Witaj na mojej stronie!)
│   └─ main
│       └─ p (To jest główny akapit.)
```



CSS: Co to takiego?

CSS - Cascading Style Sheets (Kaskadowe Arkusze Stylów)

Język służący do opisywania wyglądu stron internetowych

Oddziela strukturę (HTML) od prezentacji (CSS)

Umożliwia zmianę: kolorów, czcionek, układu, animacji itp.



CSS: Jak wygląda kod?

```
body {  
  background-color: #f0f0f0;  
  font-family: Arial, sans-serif;  
}
```

```
p h1 {  
  color: darkblue;  
  text-align: center;  
}
```

```
div.container {  
  background-color: #ffffff}
```

```
#terms-of-use {  
  font-size: 8px  
}
```

```
section > p {  
  font-weight: bold;  
}
```

```
h2 + p {  
  font-style: italic;  
}
```

```
li:first-child {  
  color: orange;  
}
```

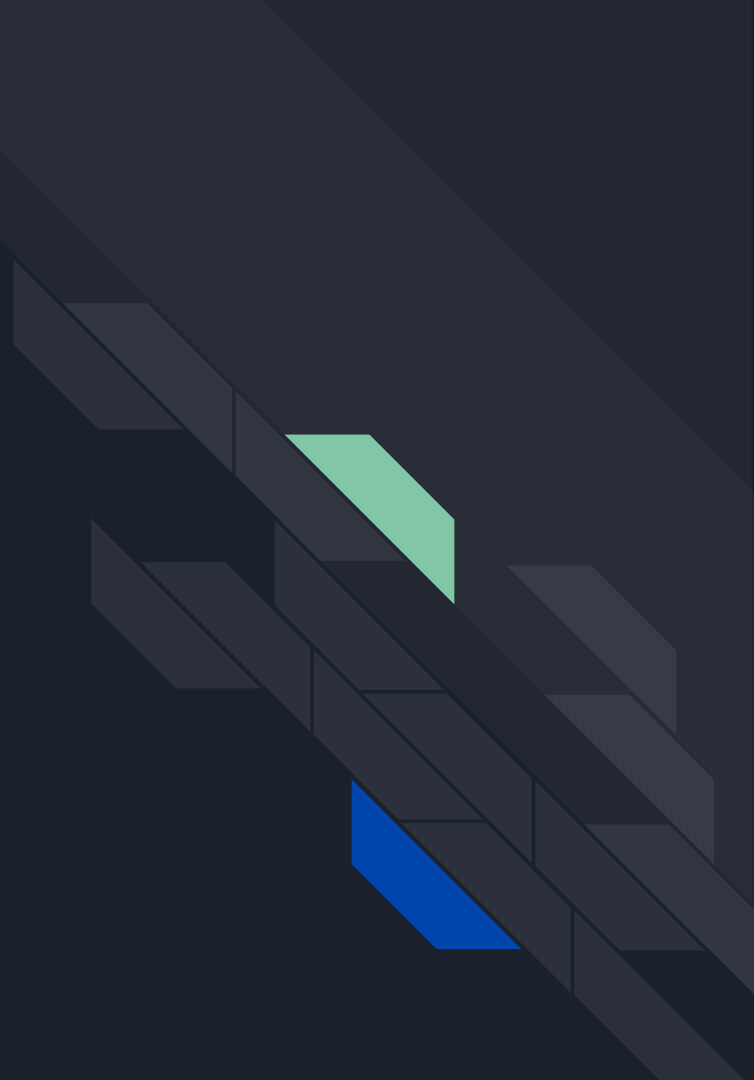
```
.container p:first-child ul.code  
{  
  font-family:
```



CSS: Dodawanie pliku CSS do dokumentu HTML

```
<!DOCTYPE html>
<html>
<head>
  <title>Moja strona</title>
  <link rel="stylesheet" href="style.css">
</head>
<body>
  <h1>Witaj na stronie!</h1>
</body>
</html>
```


Wstęp do JavaScript





JavaScript: Historia

1995 – Narodziny JavaScriptu. Stworzony w zaledwie 10 dni przez Brendana Eichę w firmie Netscape

Pierwotnie nazywał się Mocha, potem LiveScript, a dopiero potem JavaScript (z powodów marketingowych – "moda na Javę")

1996 – Microsoft i JScript

Microsoft tworzy własną wersję (JScript) do przeglądarki Internet Explorer. Zaczyna się wojna przeglądarek

1997 – Standaryzacja (ECMAScript)

JavaScript zostaje sformalizowany jako ECMAScript (ES) przez organizację ECMA. Pierwszy standard: ES1

Lata 2000–2010: Powolny rozwój, jQuery, AJAX

Popularność dynamicznych stron. Powstają biblioteki jak jQuery, które upraszczają użycie JS

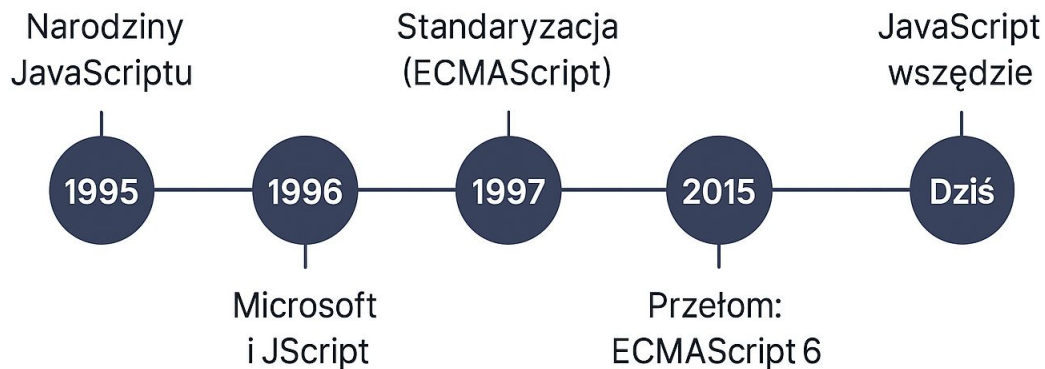
2015 – Przełom: ECMAScript 6 (ES6)

Duża aktualizacja: let/const, klasy, funkcje strzałkowe, import/export. JS staje się dojrzałym językiem.

Dziś: JavaScript wszędzie

Backend: Node.js. Aplikacje webowe: React, Vue, Angular. Nawet aplikacje mobilne i desktopowe (React Native, Electron)

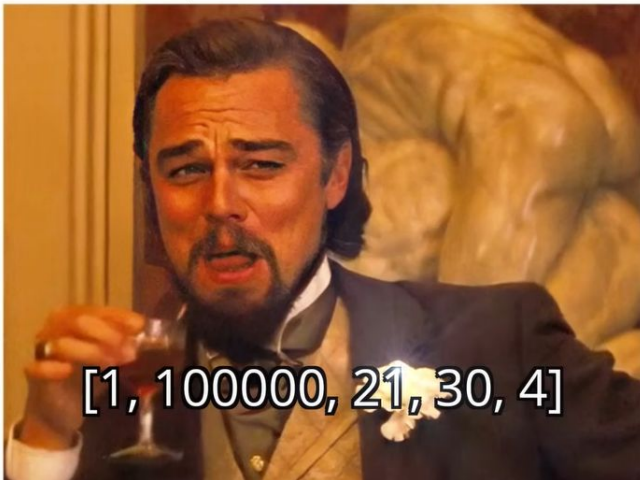
Krótką historia JavaScriptu




JavaScript stworzony w 10 dni

People learning JavaScript:
"I'll use `array.sort()` to
sort this list of numbers!"

JavaScript:



```
> typeof NaN
< "number"
> 9999999999999999
< 10000000000000000
> 0.5+0.1==0.6
< true
> 0.1+0.2==0.3
< false
> Math.max()
< -Infinity
> Math.min()
< Infinity
> []+[]
< ""
> []+{}
< "[object Object]"
> {}+[]
< 0
> true+true+true===3
< true
> true-true
< 0
```

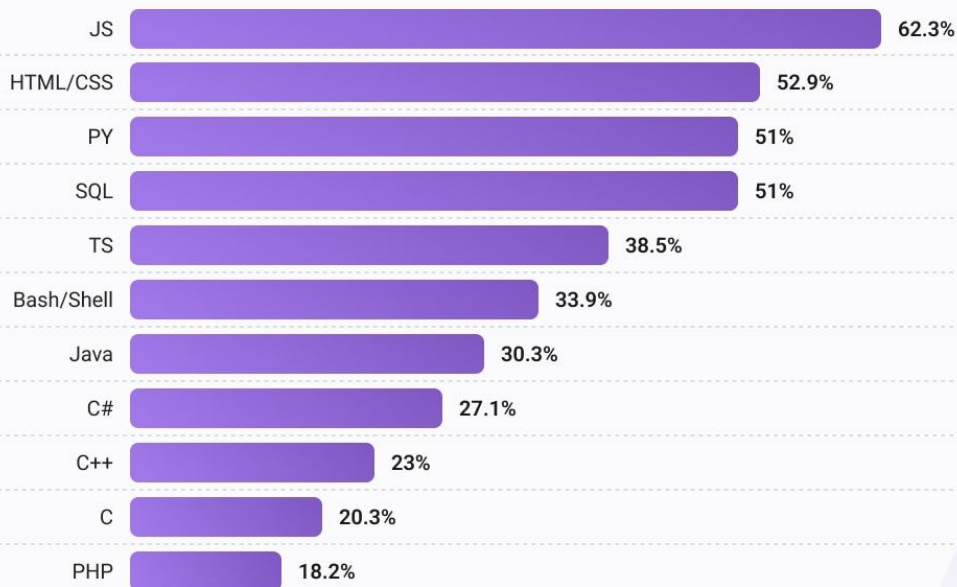


Thanks for inventing Javascript

JavaScript: zdecydowanie popularny i zdecydowanie użyteczny

2024 Developer Survey

Source: [Stack Overflow 2024 Developer Survey](#)



Wstęp do JavaScript: typy i struktury danych





JavaScript: Podstawowe typy danych

JavaScript ma kilka prostych typów danych
(tzw. typy prymitywne):

```
- let name = "Anna";           // string
- let age = 30;                 // number
- let isHappy = true;          // boolean
- let nothing = null;           // null
- let notDefined;              // undefined
- let id = Symbol("id");        // symbol
- let bigNum = 123n;            // bigint
```



JavaScript: String

string to sekwencja znaków, zapisana w cudzysłowie:

```
let a = "tekst";  
let b = 'tekst';  
let c = `tekst`; // tzw. template literal  
  
let str1 = "Hello";      // podwójne cudzysłowy  
let str2 = 'World';      // pojedyncze cudzysłowy  
let str3 = `Hi ${str1}`; // backticks, template literal  
                        (umożliwia interpolację)  
  
let multi = `to jest  
tekst w wielu  
linijkach`;
```




JavaScript: Właściwości stringów

`length` - długość tekstu:

```
"hello".length // 5
```

Dostęp do znaku przez indeks:

```
"hello"[1] // "e"
```

Nie możesz zmienić znaku w stringu bez stworzenia nowego stringa:

```
let word = "kot";  
word[0] = "p";           // nie zadziała  
word = "pot";           // musisz przypisać nową  
wartość
```

JavaScript: Popularne operacje na stringach

Metoda	Co robi	Przykład
<code>toUpperCase()</code>	Duże litery	<code>"abc".toUpperCase()</code> → <code>"ABC"</code>
<code>toLowerCase()</code>	Małe litery	<code>"XYZ".toLowerCase()</code> → <code>"xyz"</code>
<code>includes(substr)</code>	Sprawdza, czy zawiera tekst	<code>"test".includes("es")</code> → <code>true</code>
<code>startsWith(str)</code>	Czy zaczyna się od	<code>"Hello".startsWith("He")</code>
<code>endsWith(str)</code>	Czy kończy się na	<code>"abc".endsWith("c")</code>
<code>indexOf(substr)</code>	Pozycja pierwszego wystąpienia	<code>"abcabc".indexOf("b")</code> → <code>1</code>
<code>slice(start, end)</code>	Wycina fragment (nie modyfikuje oryginału)	<code>"abcdef".slice(1, 4)</code> → <code>"bcd"</code>
<code>substring(start, end)</code>	Podobna do <code>slice</code> (nie działa z ujemnymi liczbami)	
<code>replace(old, new)</code>	Zamienia pierwsze wystąpienie	<code>"abcabc".replace("a", "x")</code> → <code>"xbcabc"</code>
<code>trim()</code>	Usuwa białe znaki z początku i końca	<code>" tekst ".trim()</code>
<code>split(separator)</code>	Dzieli tekst na tablicę	<code>"a,b,c".split(",")</code> → <code>["a", "b", "c"]</code>
<code>repeat(n)</code>	Powielia tekst	<code>"ha".repeat(3)</code> → <code>"hahaha"</code>



JavaScript: Typ złożony, obiekt

Obiekt to kolekcja par *klucz: wartość*:

```
let person = {  
  name: "Anna",  
  age: 30,  
  isHappy: true  
};
```

```
console.log(person.name); // "Anna"
```

Wartościami mogą być również funkcje. Funkcje zawarte w obiekcie często określa się metodami obiektu.



JavaScript: Tablica

Tablica przechowuje wiele wartości w jednej zmiennej:

```
let colors = ["red", "green", "blue"];

console.log(colors[0]);      // "red"
console.log(colors.length);  // 3
```

Elementy są indeksowane od 0

Tablica to też obiekt (`typeof colors === "object"`) i ma swoje metody jak:

- `push()`, `pop()`
- `shift()` i `unshift()`
- `map()`, `filter()`
- `forEach()`
- `find()`, `includes()`
- `slice()`
- `join()`



JavaScript: Typy danych - zadanie praktyczne

Stwórz plik `profile.js`, w którym zapiszesz informacje o sobie w postaci zmiennych różnych typów danych.

W pliku powinny się znaleźć:

- Twoje imię jako `string`
- Twój wiek jako `number`
- Czy masz prawo jazdy – `boolean`
- Lista Twoich ulubionych języków programowania – `array`
- Obiekt `person`, który zawiera:
 - imię,
 - wiek,
 - ulubione języki
- Dodaj zmienną `middleName`, ale nie przypisuj jej wartości (`undefined`)
- Dodaj zmienną `luckyNumber`, ale przypisz jej wartość `null`

Wypisz wszystkie dane do konsoli (`console.log`)

Sprawdź typ każdej zmiennej za pomocą `typeof`

Zmień wartość zmiennej `age` i wypisz ją ponownie

Dodaj nowy język do tablicy `favoriteLanguages` metodą `.push(newValue)`



JavaScript: Funkcje

W JavaScript funkcje to też typ danych. Funkcja to blok kodu, który można wykonać wiele razy, często z różnymi danymi (parametrami).

```
function greet(name) {  
    return "Hello " + name;  
}  
  
let sayHi = greet;  
console.log(sayHi("Anna")); // "Hello Anna"
```



JavaScript: Sposoby definiowania funkcji

Deklaracja funkcji (function declaration)


```
function add(a, b) {  
  return a + b;  
}
```

Wyrażenie funkcyjne (function expression)

```
const multiply = function (a, b) {  
  return a * b;  
};
```

Funkcja strzałkowa (arrow function)

```
const greet = (name) => "Hi " + name;
```



JavaScript: Funkcje - argumenty i domyślne wartości

```
function greet(name = "Gość") {  
    return "Cześć, " + name;  
}
```

W JavaScript funkcja może być wywołana:

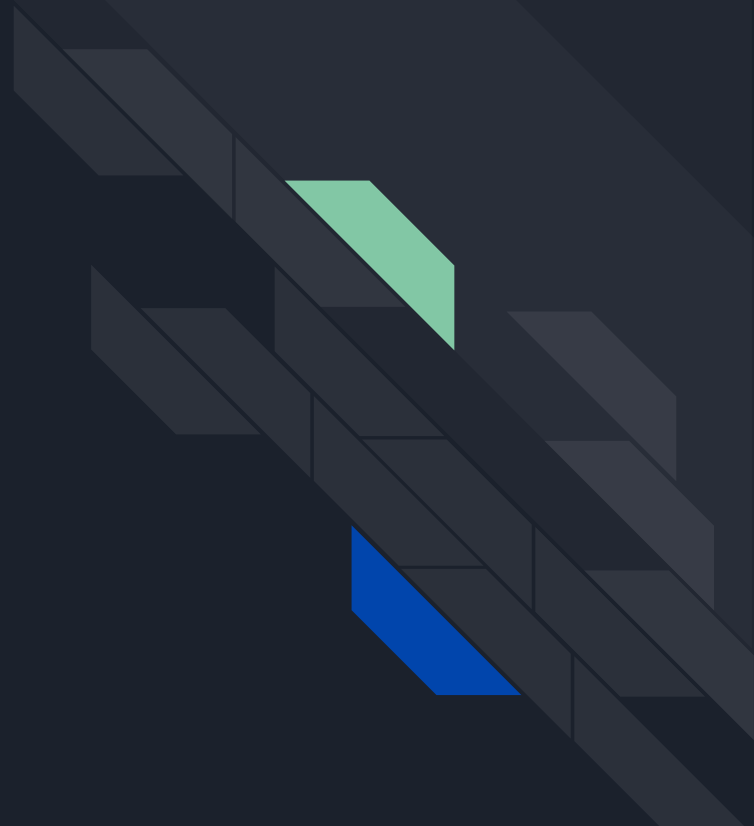
- z dowolną liczbą argumentów
- niezależnie od tego, ile argumentów zostało zadeklarowanych

```
greet();           // Cześć, undefined
```

Każda funkcja (poza arrow functions) ma dostęp do obiektu `arguments`, który zawiera wszystkie przekazane wartości:

```
function showAll() {  
    for (let i = 0; i < arguments.length; i++) {  
        console.log(arguments[i]);  
    }  
}  
  
showAll("a", "b", "c"); // wypisze a, b, c
```


Wstęp do JavaScript: operatory





JavaScript: Operatory arytmetyczne

Operator	Działanie	Przykład
+	Dodawanie	<code>2 + 3</code> → 5
-	Odejmowanie	<code>5 - 1</code> → 4
*	Mnożenie	<code>3 * 4</code> → 12
/	Dzielenie	<code>8 / 2</code> → 4
%	Reszta z dzielenia	<code>5 % 2</code> → 1
**	Potęgowanie	<code>2 ** 3</code> → 8

JavaScript: Operatory porównania

Operator	Znaczenie	Przykład
<code>==</code>	Równość (luźna)	<code>5 == "5" → true</code>
<code>===</code>	Równość (ściśła)	<code>5 === "5" → false</code>
<code>!=</code>	Nierówność	<code>5 != 3 → true</code>
<code>!==</code>	Ściśła nierówność	<code>5 !== "5" → true</code>
<code>></code> <code><</code> <code>>=</code> <code><=</code>	Porównania liczbowe	<code>4 < 5 → true</code>



JavaScript: Operatory logiczne

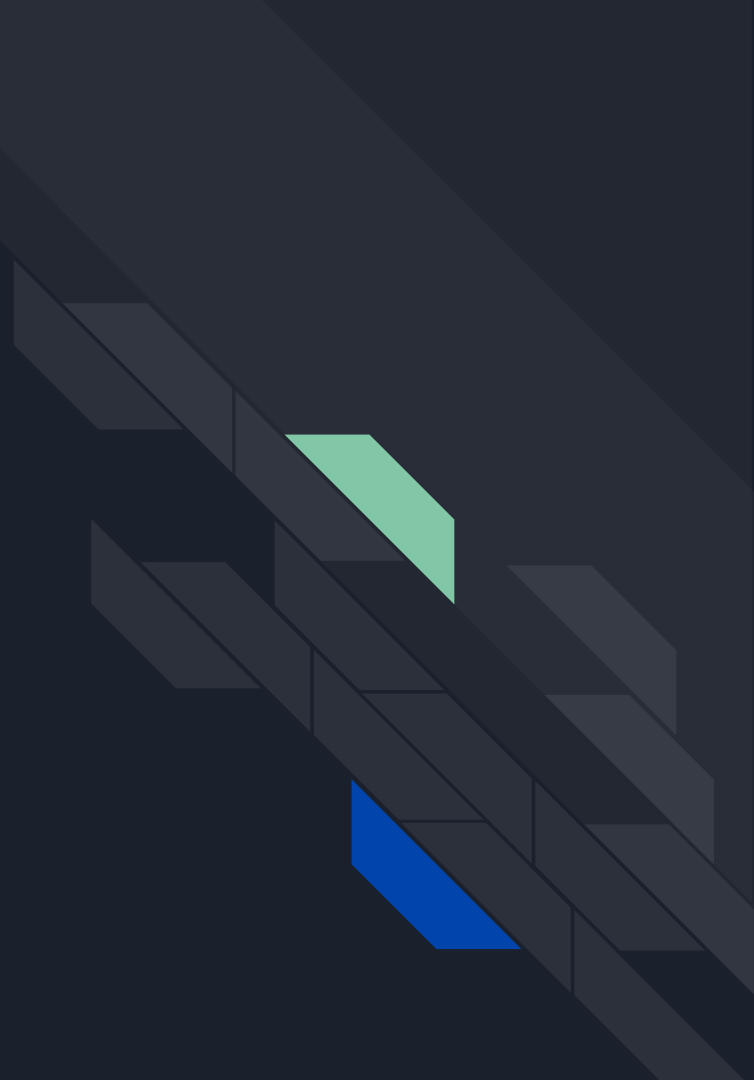
Operator	Znaczenie	Przykład
<code>&&</code>	I (AND)	<code>true && false</code> → <code>false</code>
<code>,</code>		<code>,</code>
<code>!</code>	NIE (NOT)	<code>!true</code> → <code>false</code>



JavaScript: Operatory przypisania

Operator	Znaczenie	Przykład
=	Przypisanie	<code>x = 10</code>
<code>+=</code> <code>-=</code> <code>*=</code> <code>/=</code>	Skrócone przypisania	<code>x += 2</code> (czyli <code>x = x + 2</code>)

Podsumowanie lekcji 1-4





HTML i CSS

HTML – struktura strony

- Znaczniki (tagi): `<p>`, `<h1>`, `<a>`, ``, `<div>`, itd.
- Atrybuty: `id` (unikalne), `class` (wielokrotnego użytku), `href`, `src`, `alt`
- Semantyczne znaczniki HTML5: `<header>`, `<footer>`, `<section>`, `<article>`, `<nav>`, `<main>`
- Skrypty JS osadza się za pomocą taga `<script>`
- Drzewo DOM – struktura dokumentu jako hierarchia elementów

CSS – stylizacja strony

- Oddzielenie wyglądu od struktury. animacje
- Selektory: po tagu, klasie (`.class`), ID (`#id`), zagnieżdżenia (`div > p`, `h2 + p`, `li:first-child`)



Quiz na przypomnienie

Który z poniższych jest typem prymitywnym w JavaScript?

- A) Object
- B) Array
- C) Number
- D) Function

Jaka będzie wartość tego wyrażenia: `typeof "53"`

- A) Object
- B) Number
- C) BigInt
- D) String

Która z instrukcji zwróci wartość `false`?

- A) `"53" == 53`
- B) `typeof 53 == typeof 35`
- C) `typeof 53 === typeof 35`
- D) `53 === "53"`

Które z poniższych porównań zwróci `true`?

- A) `false == 0`
- B) `false === 0`
- C) `[] == false`



Podstawy JavaScript: przypomnienie

Typy danych:

- Prymitywne: `string`, `number`, `boolean`, `null`, `undefined`, `symbol`, `bigint`

- Złożone: `object`, `array`, `function`

```
let name = "Anna";
let age = 30;
let isHappy = true;
let person = { name: "Anna", age: 30 };
let languages = ["JS", "Python"];
```

Operatory arytmetyczne:

`+` `-` `*` `/` `%` `**`

Operatory porównania

`==` `===` `!=` `!==`
`>` `>=` `<` `<=`

Operatory logiczne

`&&` `||` `!`

Operatory przypisania

`=` `+=` `-=` `*=` `/=`



Podstawy JavaScript: przypomnienie

Po co używamy funkcji?

- Unikamy duplikowania kodu — funkcje pozwalają na wielokrotne użycie tego samego kodu w różnych miejscach programu. (zasada DRY)
- Ułatwiają czytelność — dzieląc kod na mniejsze, zrozumiałe fragmenty, które mają jasne cele. (zasada KISS)

Najważniejsze zasady przy tworzeniu funkcji:

- **Semantyczne i zwarte nazwy.** Funkcje powinny mieć nazwy, które jasno opisują, co robią.

Przykład: `calculateTotal()` zamiast `calc()`, `getUserInfo()` zamiast `info()`.

- **Jedno zadanie, jedna funkcja.** Staraj się, żeby każda funkcja realizowała jedno konkretne zadanie. Dzięki temu łatwiej ją przetestować i utrzymać.

Przykład: Funkcja `addItemToCart()` powinna dodawać przedmiot do koszyka, a nie robić nic więcej.

- **Przekazuj niewielką ilość parametrów.** Dążymy do tego, żeby funkcje przyjmowały jak najmniej argumentów. Zbyt wiele parametrów to żółta flaga. Jeśli musimy przekazać zbyt wiele danych, rozważ użycie obiektów lub tablic.

Instrukcje warunkowe





JavaScript: instrukcja if

```
let age = 18;
```

```
if (age >= 18) {  
    console.log("Jesteś pełnoletni.");  
} else if (age >= 13) {  
    console.log("Jesteś nastolatkiem.");  
} else {  
    console.log("Jesteś dzieckiem.");  
}
```

```
let isLoggedIn = true;  
let message = isLoggedIn ? "Witaj z powrotem!" :  
"Zaloguj się, proszę.";
```

```
console.log(message);
```



JavaScript: instrukcja switch

```
let kolor = "zielony";

switch (kolor) {
  case "czerwony":
    console.log("Stop!");
    break;
  case "zielony":
    console.log("Idź!");
    break;
  case "żółty":
    console.log("Uwaga!");
    break;
  default:
    console.log("Nieznany kolor");
}
```

Kiedy używać? Gdy chcesz wykonać różne akcje w zależności od wartości jednej zmiennej.

Co warto zapamiętać:

- break kończy daną gałąź – **bez niego wykonanie „przeleci” dalej!**
- default działa jak „else” – nie jest wymagany, ale przydatny.



Instrukcje warunkowe: zadanie praktyczne

Napisz skrypt, który pyta użytkownika o numer dnia tygodnia (1–7), a następnie wyświetla jego nazwę (np. 1 → "Poniedziałek"). Użyj do tego instrukcji `switch`.

1. Użytkownik wprowadza liczbę od 1 do 7 (możesz użyć `prompt()` w przeglądarce lub stałej w kodzie).
2. Program używa `switch`, aby przypisać odpowiednią nazwę dnia.
3. Jeśli użytkownik poda liczbę spoza zakresu 1–7, wyświetl komunikat: "Nieprawidłowy numer dnia."
4. Bonus: Obsłuż sytuację, gdy użytkownik wpisze tekst albo nic nie wpisze (np. `null`).

Przydatne w zadaniu funkcje:

- `prompt()` - do pobrania zmiennej od użytkownika
- `console.log()` - do wypisania wartości w konsoli JS
- `alert()` - do zwrócenia komunikatu użytkownikowi w formie okienka wyskakującego
- `isNaN()` - sprawdza, czy wartość przekazana w parametrze to NaN (Not a Number)



JavaScript: pętla for

Do czego służy? Do powtarzania kodu określoną liczbę razy, np. przy pracy z tablicami lub liczeniem.

```
for (inicjalizacja; warunek; zmiana) {  
  // kod do wykonania w każdej iteracji  
}
```

```
for (let i = 0; i < 5; i++) {  
  console.log("Liczba: " + i);  
}
```

let i = 0 — start od 0

i < 5 — pętla działa, dopóki i < 5

i++ — po każdej iteracji zwiększ i o 1

console.log(...) — wypisuje liczby: 0, 1, 2, 3, 4



JavaScript: pętla while

Do czego służy? Pętla `while` wykonuje kod dopóki warunek jest prawdziwy. Idealna, gdy nie wiesz, ile razy trzeba powtórzyć operację.

```
while (warunek) {  
    // kod do wykonania  
}
```

- Warunek jest sprawdzany przed każdą iteracją.
- Jeśli warunek jest `true`, kod w pętli się wykona.
- Jeśli warunek jest `false`, pętla kończy działanie.



Pętle: Zadanie praktyczne

1. Użyj pętli, która będzie iterować od 1 do 50.
2. Sprawdź, czy liczba jest parzysta (użyj operatora %).
3. Jeśli liczba jest parzysta, wypisz ją na ekranie.



Pętle: instrukcje break i continue

Instrukcja `break` natychmiastowo przerywa działanie pętli, niezależnie od tego, czy warunek pętli jest spełniony, czy nie.

```
for (let i = 0; i < 10; i++) {  
  if (i === 5) {  
    break;  
  }  
  
  console.log(i);  
}
```

Instrukcja `continue` przerywa bieżącą iterację pętli i przechodzi do następnej, pomijając kod, który znajduje się poniżej w tej iteracji.

```
for (let i = 0; i < 5; i++) {  
  if (i === 3) {  
    continue; // Pomiń tę iterację, gdy i  
    == 3  
  }  
  
  console.log(i);  
}
```

break: Kiedy chcesz natychmiast przerwać pętlę, np. po znalezieniu konkretnego elementu w tablicy.

continue: Kiedy chcesz pominąć część iteracji, ale kontynuować kolejne, np. przy sprawdzaniu danych.



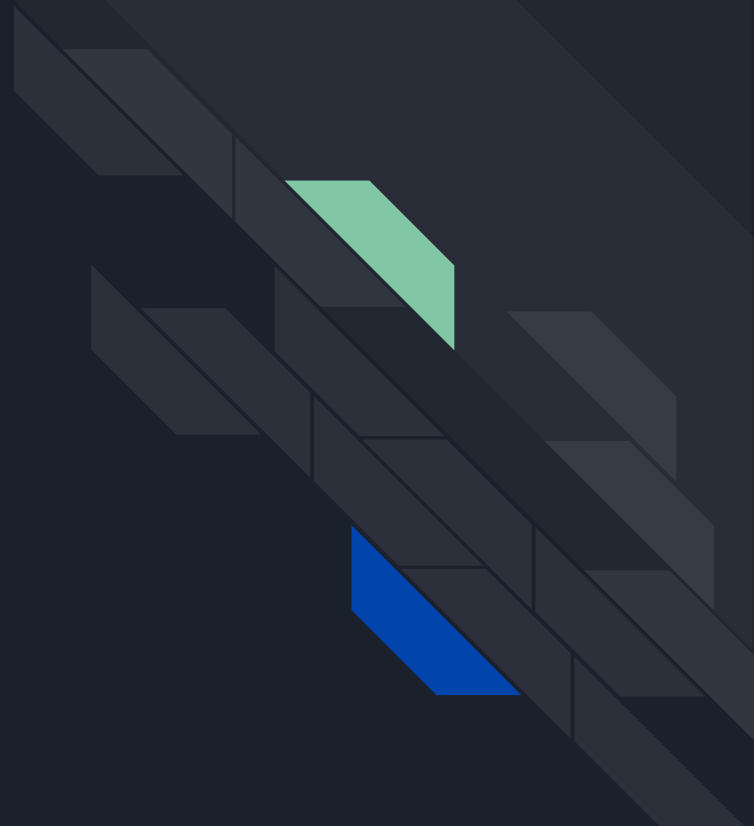
Zadanie praktyczne: instrukcje break i continue

Napisz program, który będzie sprawdzał liczby od 1 do 50, aby znaleźć liczby pierwsze. Zastosuj instrukcje `break` i `continue`.

Instrukcje:

1. Liczba pierwsza to taka liczba, która jest większa niż 1 i nie ma dzielników oprócz 1 i samej siebie.
2. Użyj pętli `for` do iteracji przez liczby od 2 do 50.
3. Dla każdej liczby sprawdź, czy jest liczbą pierwszą:
 - a. Jeśli liczba nie jest pierwsza, pomiń dalsze sprawdzanie tej liczby (użyj `continue`).
 - b. Jeśli znajdziesz dzielnik, przerwij pętlę dla tej liczby (użyj `break`).
4. Wypisz każdą liczbę pierwszą.

Programowanie obiektywne w JS





Co to jest programowanie obiektowe (OOP)?

Programowanie obiektowe to styl programowania, w którym kod organizujemy wokół obiektów, które łączą dane (właściwości) i logikę (metody).

Kluczowe pojęcia:

- Obiekt – reprezentuje „rzecz” z cechami i zachowaniami
- Klasa – szablon do tworzenia obiektów
- Metody – funkcje należące do obiektu



Tworzenie obiektów w JavaScript

W JS można tworzyć obiekty na różne sposoby. Najprostszy to tzw. „object literal”.

```
const car = {  
  brand: "Toyota",  
  year: 2021,  
  start: function() {  
    console.log("Car is starting...");  
  }  
};
```

```
car.start(); // Car is starting...
```

Obiekt car ma właściwości (brand, year) i metodę (start).



Klasy w JavaScript (ES6+)

Od ES6 możemy używać słowa kluczowego `class`, by tworzyć obiekty w stylu klasycznego OOP.

```
class Person {  
  
    constructor(name, age) {  
        this.name = name;  
        this.age = age;  
    }  
  
    greet() {  
        console.log(`Hi, I'm ${this.name}`);  
    }  
  
}  
  
const john = new Person("John", 30);  
  
john.greet(); // Hi, I'm John
```

Person to klasa. Obiekt `john` jest jej instancją i ma dostęp do metody `greet`.



Co oznacza this w klasach JavaScript?

W klasach `this` odwołuje się do bieżącego obiektu, czyli instancji klasy, na której wywoływana jest metoda.

```
class Dog {  
  constructor(name) {  
    this.name = name; // `this` oznacza obiekt, który tworzymy  
  }  
  
  bark() {  
    console.log(`${this.name} says: Woof!`);  
  }  
}  
  
const rex = new Dog("Rex");  
rex.bark(); // Rex says: Woof!
```

W metodzie `bark()` słowo `this.name` odwołuje się do właściwości `name` konkretnego psa (np. "Rex"). `this` zawsze odnosi się do obiektu, który wywołuje metodę.



Programowanie obiektowe: zadanie praktyczne

Stwórz klasę `Article` z następującymi właściwościami:

- `title` – tytuł artykułu
- `content` – treść artykułu
- `author` – autor artykułu
- `publishedDate` – data publikacji (może być przekazana jako `Date` lub `string`)

Dodaj metody:

- `getSummary()` – zwraca pierwsze 100 znaków treści zakończone ...
- `isRecent()` – zwraca `true`, jeśli artykuł został opublikowany w ciągu ostatnich 7 dni

Stwórz przynajmniej 3 różne artykuły (instancje klasy), dodaj je do tablicy i wypisz w pętli:

- Tytuł
- Autor
- Czy jest świeży (`isRecent()`)
- Skrócony opis (`getSummary()`)

Pomocne funkcje/metody:

- `now = new Date(); now.getTime()`
- `slice()` lub `substring()`



Programowanie obiektowe

Dlaczego warto używać OOP?

- Kod jest czytelniejszy
- Łatwiej zarządzać większymi projektami
- Promuje wielokrotne użycie kodu
- Pomaga w organizacji danych i funkcji razem



Programowanie obiektowe: Filary OOP

Abstrakcja (Abstraction)

Ukrywanie szczegółów implementacji i pokazywanie tylko tego, co istotne.

Użytkownik klasy nie musi wiedzieć jak coś działa, tylko co robi.

Przykład: klasa Car ma metodę drive(), ale nie pokazuje, co dzieje się wewnątrz silnika.

Enkapsulacja (Encapsulation)

Grupowanie danych (właściwości) i metod w jednej jednostce – obiekcie.

Ochrona danych przed nieautoryzowanym dostępem.

Często realizowana przez modyfikatory dostępu (private, # w JS).

Dziedziczenie (Inheritance)

Możliwość tworzenia nowych klas na podstawie już istniejących.

Pozwala unikać duplikowania kodu.

Przykład: klasa Admin dziedziczy po klasie User.

Polimorfizm (Polymorphism)

Ta sama metoda może zachowywać się inaczej w zależności od kontekstu.



Quiz

Co wypisze poniższy kod?

```
let i = 0;
while (i < 3) {
  console.log(i);
  i++;
}
```

- A) 1 2 3
- B) 0 1 2
- C) 0 1 2 3
- D) Nie wypisze nic

Co wypisze poniższy kod?

```
let value = 0;
if (value) {
  console.log("Prawda");
} else {
  console.log("Fałsz");
}
```

- A) Prawda
- B) Fałsz
- C) 0
- D) Błąd



Zadanie praktyczne: Znajdź największą liczbę

Masz tablicę z liczbami. Napisz funkcję, która znajdzie największą liczbę za pomocą pętli.

```
const numbers = [3, 18, 7, 21, 14];
```

Oczekiwany wynik: Największa liczba to: 21



Modyfikacja DOM za pomocą JavaScript

DOM (Document Object Model) to struktura dokumentu HTML w postaci drzewa obiektów, z którymi JavaScript może pracować.

HTML staje się dostępny w JS jako obiekty

Każdy element to węzeł: div, p, h1, itp.

Możemy je dynamicznie tworzyć, modyfikować, usuwać

```
<!-- HTML -->
```

```
<p>Hello!</p>
```

```
// JavaScript
```

```
const p = document.querySelector("p");
```

```
p.textContent = "Cześć!";
```



Najczęstsze operacje na DOM

Cel	Przykład w JS
Znaleźć element	<code>document.getElementById("id")</code>
Utworzyć nowy element	<code>document.createElement("div")</code>
Ustawić tekst	<code>element.textContent = "Tekst"</code>
Ustawić HTML	<code>element.innerHTML = "<p>Tekst</p>"</code>
Dodać element do DOM	<code>parent.appendChild(child)</code>
Zmienić klasę CSS	<code>element.classList.add("nazwa-klasy")</code>



Przykład dynamicznego tworzenia elementu

```
const atricleTitle = "Programiści go nienawidzą!"
const articleContent = "Lorem Ipsum ..."
const article = document.createElement("article");

article.innerHTML = `
  <h2>${articleTitle}</h2>
  <p>${articlkeContent}</p>
`;

document.body.appendChild(article);
```



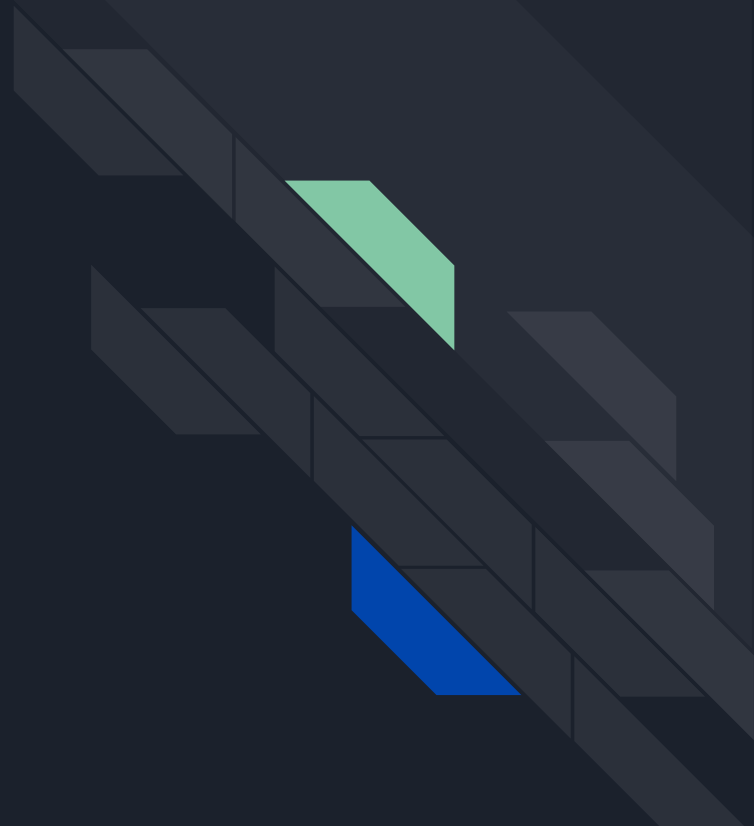

Zadanie praktyczne: dynamiczne generowanie treści bloga

1. Stwórz klasę `Article` z właściwościami:
 - `title`
 - `content`
 - `author`
 - `publishedDate`
2. Dodaj metody `getSummary()` (pierwsze 100 znaków contentu) i `isRecent()` (publikacja w ciągu ostatnich 7 dni)
3. Stwórz kilka artykułów i dodaj je do tablicy.
4. Wypisz je na stronie w postaci HTML-a – każdy jako `<article>` z tytułem, autorem, datą i opisem.
5. Bonus: dodaj do elementu `<article>` klasę `recent`, jeżeli artykuł był opublikowany w ciągu ostatnich 7 dni

```
<article class="recent">
  <h2>How to Learn JavaScript</h2>
  <p><strong>Autor:</strong> John Doe</p>
  <p><strong>Opublikowano:</strong> 2025-04-20</p>
  <p>Lorem Ipsum ...</p>
</article>
```

1. Użyj `document.getElementById()` lub `document.querySelector()` do złapania kontenera.
2. Dla każdego artykułu stwórz nowy element `article`
3. Wypełnij element `article` kodem HTML (`h2`, `p`, itd) na przykład za pomocą *template literals*
4. Użyj `element.innerHTML` do wstawiania wygenerowanej treści.
5. Na koniec użyj `container.appendChild(article)`.

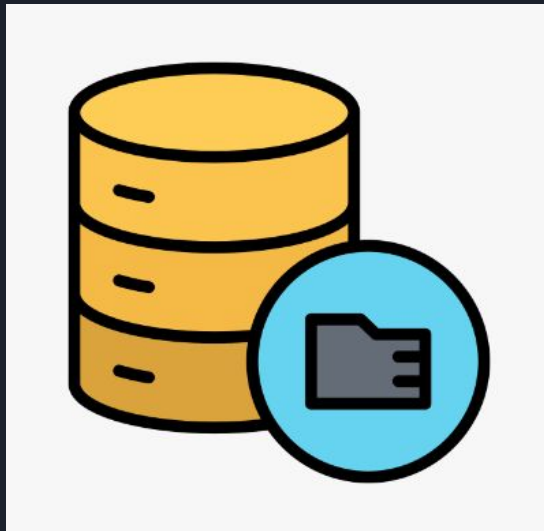
Przechowywanie danych




JavaScript: przechowywanie danych

Dane można przechowywać:

- Po stronie klienta (w przeglądarce)
 - Ciasteczka (cookies)
 - localStorage
 - sessionStorage
- Po stronie serwera
 - pamięć tymczasowa
 - baza danych (SQL, noSQL)





JavaScript: przechowywanie danych po stronie klienta

JavaScript umożliwia przechowywanie danych w przeglądarce użytkownika za pomocą:

- Cookies – małe pliki tekstowe, wysyłane przy każdym żądaniu do serwera
- localStorage – przechowywanie danych na stałe (dopóki użytkownik ich nie usunie)
- sessionStorage – dane dostępne tylko w ramach jednej sesji (zamykanie karty je usuwa)

Cookies – podstawy

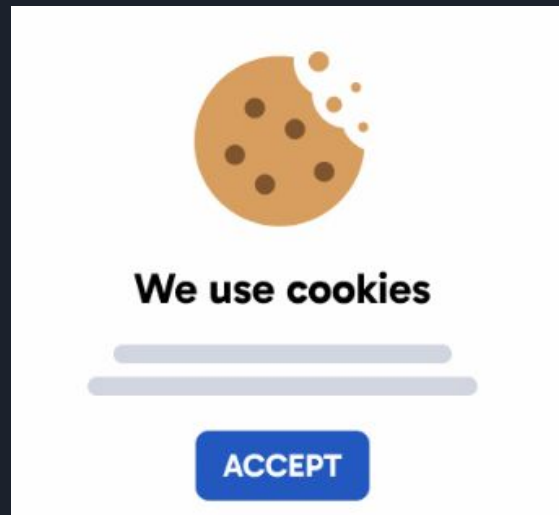
- Maksymalny rozmiar: ~4 KB
- Przechowywane przez przeglądarkę, ale **wysyłane do serwera przy każdym żądaniu**
- Używane np. do:
 - śledzenia logowania,
 - preferencji,
 - analityki

Zapis:

```
document.cookie = "username=Anna; expires=Fri, 31 Dec 2025 23:59:59 GMT; path="/;
```

Odczyt:

```
console.log(document.cookie);
```





localStorage – trwałe przechowywanie danych

- Dostępne dopóki użytkownik nie wyczyści danych przeglądarki
- Pojemność: ok. 5-10 MB
- Dane są nigdy nie wysyłane do serwera
- Klucz-wartość, tylko ciągi znaków

```
localStorage.setItem("theme", "dark");  
let theme = localStorage.getItem("theme"); // "dark"  
localStorage.removeItem("theme");
```



sessionStorage – dane tymczasowe

Działa jak localStorage, ale:

- Dane są przechowywane tylko do momentu zamknięcia karty
- Przydatne do przechowywania np. stanu formularzy, koszyka w danej sesji

```
sessionStorage.setItem("step", "2");  
let step = sessionStorage.getItem("step"); // "2"
```

Pamiętajmy: Dane staramy się przechowywać najkrócej jak się da i tylko te, które są nam niezbędne do działania aplikacji.



localStorage / sessionStorage – najpopularniejsze metody

`setItem(key, value)` - Zapisuje dane pod określonym kluczem. Nadpisuje, jeśli klucz już istnieje.

`localStorage.setItem("username", "Anna");`

`getItem(key)` - Odczytuje wartość zapisaną pod danym kluczem. Zwraca null, jeśli klucz nie istnieje.

`localStorage.getItem("username");`

`removeItem(key)` - Usuwa wpis z localStorage na podstawie klucza.

`localStorage.removeItem("username");`

`clear()` - Usuwa wszystkie dane z localStorage dla danej domeny.

`localStorage.clear();`

`key(index)` - Zwraca nazwę klucza na danym indeksie (kolejność zapisu).

`localStorage.key(0);`



Przechowywanie danych - zadanie praktyczne

1. wprowadź imię użytkownika do zmiennej,
2. zapisz je w `localStorage`,
3. odczytaj zapisane imię i wypisz je w konsoli,
4. usuń je z `localStorage`.



Przechowywanie danych: złote zasady

1. **Nigdy nie przechowuj wrażliwych danych w localStorage lub cookies.**
 - Hasła, tokeny sesyjne, numery kart kredytowych – to dane, które powinny być przechowywane w bezpiecznym miejscu, np. na serwerze lub w odpowiednich bazach danych.
 - Jeśli musisz przechować dane użytkownika, rozważ szyfrowanie danych przed zapisaniem.
2. **Ogranicz rozmiar przechowywanych danych**
 - Nie przechowuj zbyt dużych danych w localStorage – ma on ograniczenie rozmiaru (około 5 MB na domenę).
 - Wykorzystuj odpowiednie narzędzia do zarządzania dużymi zbiorami danych, np. przechowuj tylko najważniejsze dane w localStorage, a resztę w bazach danych lub na serwerze.
3. **Bądź świadomy czasu życia danych**
 - localStorage przechowuje dane na stałe, dopóki użytkownik nie wyczyści danych przeglądarki, podczas gdy sessionStorage jest tymczasowy i dane znikają po zamknięciu karty.
 - Zarządzaj danymi z wyraźnym określeniem ich czasu życia – np. dane, które mają być przechowywane tylko przez sesję, przechowuj w sessionStorage.



Przechowywanie danych: złote zasady

4. **Używaj JSON do przechowywania obiektów**
localStorage przechowuje tylko ciągi znaków (stringi), więc do przechowywania obiektów lub tablic musisz użyć `JSON.stringify()` przy zapisie, a `JSON.parse()` przy odczycie.
5. **Pamiętaj o prywatności użytkownika**
 - Zbieraj tylko te dane, które są niezbędne – unikaj zbierania informacji, które nie są potrzebne do działania aplikacji.
 - Pozwól użytkownikowi na kontrolowanie danych – umożliwaj im łatwe usuwanie danych z `localStorage` i cookies.
6. **Zabezpiecz dane w cookies**
 - Używaj flagi `HttpOnly` i `Secure` w cookies, by zapobiec atakom XSS i umożliwić bezpieczne przesyłanie cookies tylko przez HTTPS.
 - Określ datę wygaśnięcia cookies, jeśli chcesz, by dane były przechowywane tylko przez określony czas.



Przechowywanie danych: złote zasady

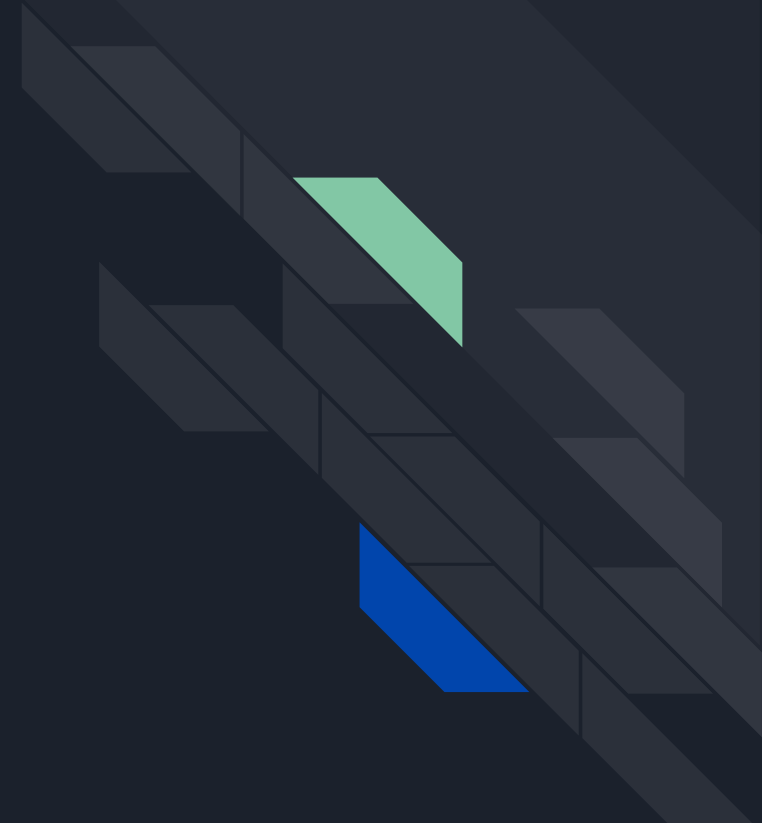
7. Regularnie przeglądaj dane

- Zarządzaj danymi aktywnie – regularnie usuwaj dane, które już nie są potrzebne.
- Monitoruj przestrzeń w `localStorage` – kontroluj, ile danych przechowujesz, aby nie przekroczyć limitów przeglądarki.

8. Zachowaj spójność danych

Synchronizuj dane pomiędzy różnymi źródłami przechowywania – np. jeśli zapisujesz dane w `localStorage`, upewnij się, że będą one również synchronizowane z danymi na serwerze, jeśli to konieczne.

Formaty danych
używane do przesyłania i
przechowywania





Przechowywanie danych: Co to jest JSON?

JSON (JavaScript Object Notation) to format zapisu danych:

- Prosty
- Czytelny dla człowieka
- Obsługiwany przez większość języków programowania

Używany m.in. do:

- komunikacji z API
- przechowywania danych (np. w localStorage)
- zapisu plików konfiguracyjnych

```
{
  "volume": "blaring",
  "current" : {
    "band": "rednex",
    "song": "cotton eye joe",
    "members":[
      {"firstname":"Kent","lastname":"Olander"},
      {"firstname":"Urban","lastname":"Landgren"},
      {"firstname":"Jonas","lastname":"Lundstrom"},
      {"firstname":"Tor","lastname":"Nilsson"}
    ]
  },
  "next" : {
    "band": "the dubliners",
    "song": "finnegan's wake",
    "members":[
      {"firstname":"Ronnie","lastname":"Drew"},
      {"firstname":"Luke","lastname":"Kelly"},
      {"firstname":"Ciaran","lastname":"Bourke"},
      {"firstname":"Barney","lastname":"McKenna"}
    ]
  }
}
```



Przechowywanie danych: Jak wygląda JSON?

```
{  
  "name": "Ola",  
  "age": 25,  
  "isStudent": true,  
  "skills": ["JavaScript", "HTML", "CSS"]  
}
```

- Klucze: **zawsze w cudzysłowach**
- Wartości: string, liczba, boolean, null, tablica, obiekt



Przechowywanie danych: JSON vs JavaScript

1. JSON to tekst (string)
2. Obiekt JS to struktura w kodzie

```
const user = {  
  name: "Ola",  
  age: 25  
};
```

```
// Zamiana obiektu JS na JSON (tekst)  
const json = JSON.stringify(user);
```

```
// Zamiana JSON-a na obiekt JS  
const parsed = JSON.parse(json);
```




Przechowywanie danych: Najczęstsze zastosowania JSON-a

Wysyłanie danych do serwera:

```
fetch("/api", {  
  method: "POST",  
  body: JSON.stringify({ name: "Ola" }),  
  headers: { "Content-Type": "application/json" }  
});
```

Przechowywanie danych:

```
localStorage.setItem("user", JSON.stringify(user));
```

Odczyt danych:

```
const user =  
JSON.parse(localStorage.getItem("user"));
```



Przechowywanie danych: Typowe błędy w JSON

Brak cudzysłowów wokół kluczy


```
// Błąd:  
{ name: "Ola" }  
  
// Poprawnie:  
{ "name": "Ola" }
```

Zły format wartości (np. zamiast true → "true")

```
// Błąd:  
{ "isAdmin": "true", "age": twenty }  
  
// Poprawnie:  
{ "isAdmin": true, "age": 20 }
```

Przecinek po ostatnim elemencie

Użycie undefined, function, Date, Symbol
JSON obsługuje tylko: string, number, boolean, null, array, object



Przechowywanie danych: Jak VS Code pomaga przy pracy z formatem JSON

Co VS Code robi "z pudełka"?

- Automatycznie sprawdza składnię .json i .jsonc (JSON z komentarzami)
- Podkreśla błędy (np. brak cudzysłowów, przecinki)
- Podpowiada klucze i wartości w znanych plikach (np. package.json)
- Koloruje składnię – łatwiej coś zauważyć

Przydatne funkcje:

- Formatowanie dokumentu – Shift + Alt + F
- Skrót do składni JSON – np. {} i [] z auto-uzupełnianiem
- Wbudowany JSON Schema – np. dla plików konfiguracyjnych

Przydatne rozszerzenia:

- JSON Tools – do sortowania, formatowania i walidacji
- Prettier – może też formatować JSON według ustalonych zasad



Przechowywanie danych: Co to jest XML?

XML (eXtensible Markup Language) to język znaczników do przechowywania i wymiany danych.

Zbudowany podobnie do HTML, ale jego głównym celem jest strukturalne przechowywanie danych, **a nie ich prezentacja.**

Popularny w starszych systemach, usługach webowych (SOAP), plikach konfiguracyjnych.

```
<user>
  <name>Anna</name>
  <age>30</age>
</user>
```



Przechowywanie danych: XML vs JSON

Cecha	XML	JSON
<hr/>		
Składnia	Znakowa, znacznikowa	Lekka, obiektowa
Łatwość użycia w JS	Mniej wygodny	Naturalna integracja
Rozmiar danych	Większy	Mniejszy
Czytelność	OK	Bardzo dobra

Wniosek? XML był w porządku, JSON jednak robi to co XML tylko lepiej i wygodniej się z nim pracuje.



Przechowywanie danych: Rehydracja

Dane zapisane w localStorage są przechowywane jako tekst (string). Podczas odczytu otrzymujemy **ciąg znaków, a nie oryginalny obiekt**.

```
// Zapis do localStorage
const user = { name: "Ala", age: 25 };
localStorage.setItem("user", JSON.stringify(user));
```

```
// Odczyt z localStorage
const raw = localStorage.getItem("user");
const parsedUser = JSON.parse(raw);
```

Co się dzieje z metodami obiektu przy zapisie do localStorage?



Przechowywanie danych: Rehydracja

Co się dzieje z metodami obiektu przy zapisie do localStorage?

```
const user = {  
  name: "Ala",  
  sayHi() {  
    console.log(`Cześć, jestem ${this.name}`);  
  }  
};  
  
localStorage.setItem("user", JSON.stringify(user));
```

Metody (sayHi) nie zostaną zapisane!
JSON.stringify ignoruje funkcje — zapisze tylko dane:

```
{  
  "name": "Ala"  
}
```



Przechowywanie danych: Rehydracja

Jak odzyskać metody po odczycie?

1. Rehydracja przez klasę:

```
class User {  
  constructor(data) {  
    Object.assign(this, data);  
  }  
  
  sayHi() {  
    console.log(`Cześć, jestem ${this.name}`);  
  }  
}
```

```
const raw = localStorage.getItem("user");  
const parsed = JSON.parse(raw);  
const user = new User(parsed);
```

```
user.sayHi(); // Działa!
```

```
const dane = { name: "Ala", age: 25 };  
const user = {};
```

```
Object.assign(user, dane);  
console.log(user);  
// { name: "Ala", age: 25 }
```


Przechowywanie danych: Rehydracja

Jak odzyskać metody po odczycie?

Rehydracja:

```
class User {
  constructor(data) {
    Object.assign(this, data);
    /*
     * zamiast Object.assign() moglibyśmy
     * przepisać ręcznie wszystkie atrybuty
     * obiektu.
     */
  }

  sayHi() {
    console.log(`Cześć, jestem ${this.name}`);
  }
}
```

```
const raw = localStorage.getItem("user");
const parsed = JSON.parse(raw);
const user = new User(parsed);
```

```
user.sayHi(); // Działa!
```

```
const dane = { name: "Ala", age: 25 };
const user = {};
```

```
Object.assign(user, dane);
console.log(user);
// { name: "Ala", age: 25 }
```



Zadanie praktyczne: Formularz i zapis/odczyt z localStorage

Formularz dodawania artykułu: Użytkownik wypełnia formularz (podając autora i treść artykułu). Po kliknięciu przycisku „Dodaj artykuł” dane są zapisywane w localStorage.

Przechowywanie danych w localStorage: Artykuły są zapisywane jako tablica obiektów klasy Article (zawierająca autora, treść oraz datę dodania) w localStorage.

1. Zapisz tablicę artykułów w localStorage (`localStorage.setItem(key, value)`, `JSON.stringify()`)
2. Odczytaj listę artykułów z localStorage (`localStorage.getItem(key)`)
3. Zamień pobrany JSON na tablicę obiektów (`JSON.parse()`)
4. Odtwórz obiekty klasy article, aby mieć dostęp do metod `isRecent()` i `getSummary()` - Rehydracja

Eventy w JavaScript





Eventy w JavaScript

Co to jest event?

Event (zdarzenie) to reakcja przeglądarki na działanie użytkownika, np. kliknięcie, wpisanie tekstu, przesunięcie myszką.

Jak nasłuchiwać eventów?

```
element.addEventListener("nazwaZdarzenia", funkcja);
```

Przykład

```
function handleSubmit(e) {  
  e.preventDefault();  
  const title = document.getElementById("title").value;  
  const article = document.createElement("article");  
  article.textContent = title;  
  document.getElementById("articlesContainer").appendChild(article);  
}  
  
document.getElementById("newArticleForm").addEventListener("submit", handleSubmit);
```



Eventy w JavaScript

Najpopularniejsze eventy w JavaScript

Zdarzenia myszy:

- `click` - kliknięcie
- `dblclick` - podwójne kliknięcie
- `mousedown` / `mouseup` - naciśnięcie / puszczenie przycisku myszy
- `mousemove` - poruszenie myszą
- `mouseover` / `mouseout` - najazd / zjazd kursorem z elementu

Zdarzenia klawiatury:

- `keydown` - klawisz wciśnięty
- `keyup` - klawisz puszczoney

Zdarzenia formularzy:

- `submit` - wysłanie formularza
- `change` - zmiana wartości w polu (np. `select`, `checkbox`)
- `input` - każda zmiana w polu tekstowym (na bieżąco)

Inne często używane:

- `load` - załadowanie strony lub zasobu
- `DOMContentLoaded` - załadowanie struktury DOM
- `scroll` - przewinięcie strony
- `resize` - zmiana rozmiaru okna



Eventy w JavaScript

Przykłady

```
document.getElementById("box").addEventListener("mouseover", function() {  
    this.style.backgroundColor = "lightblue";  
});
```

```
document.getElementById("nameInput").addEventListener("input", function() {  
    console.log("Wpisano:", this.value);  
});
```

```
function domLoaded(event) {  
    console.log("DOM gotowy - można manipulować elementami.");  
    console.log(e)  
}
```

```
document.addEventListener("DOMContentLoaded", domLoaded);
```



Eventy w JavaScript

Co zawiera parametr event?

- `target` – element, który wywołał zdarzenie (np. kliknięty przycisk).
- `type` – typ zdarzenia (np. "click", "keydown").
- `bubbles` – informacja, czy zdarzenie "bąbelkuje" (czyli propaguje się w górę drzewa DOM).
- `cancelable` – informacja, czy zdarzenie można anulować (np. `e.preventDefault()`).
- `timeStamp` – czas, kiedy zdarzenie zostało wywołane (w milisekundach).
- `defaultPrevented` – czy zostało wywołane `preventDefault()`.

W zależności od typu obiekt event może zawierać dodatkowe informacje, na przykład:

Zdarzenia myszy:

- `clientX` i `clientY` – współrzędne kursora w obrębie okna przeglądarki.
- `pageX` i `pageY` – współrzędne kursora w odniesieniu do całej strony (z uwzględnieniem przewinięcia).
- `button` – numer przycisku myszy, który został wciśnięty (0 - lewy, 1 - środkowy, 2 - prawy).

Zdarzenia klawiatury:

- `key` – naciśnięty klawisz (np. "Enter", "a").
- `keyCode` – kod klawisza (starsze właściwości, obecnie zaleca się używanie `key`).




Zadanie praktyczne

1. Przechwyć event kliknięcia myszą na całym elemencie body Twojego bloga.
2. Wypiszcie cały obiekt event przekazywany w eventListenerze w konsoli
3. Wypiszcie koordynaty punktu w którym nastąpiło kliknięcie
4. Jeżeli kliknięcie nastąpiło na elemencie typu text input, wypiszcie w konsoli wartość tego pola tekstowego

Przydatne funkcje i atrybuty:

- `document.body` lub `document.querySelector("body")`
- `addEventListener("click", function)`
- `e.clientX` i `e.clientY`
- `e.target.type` i `e.target.value`

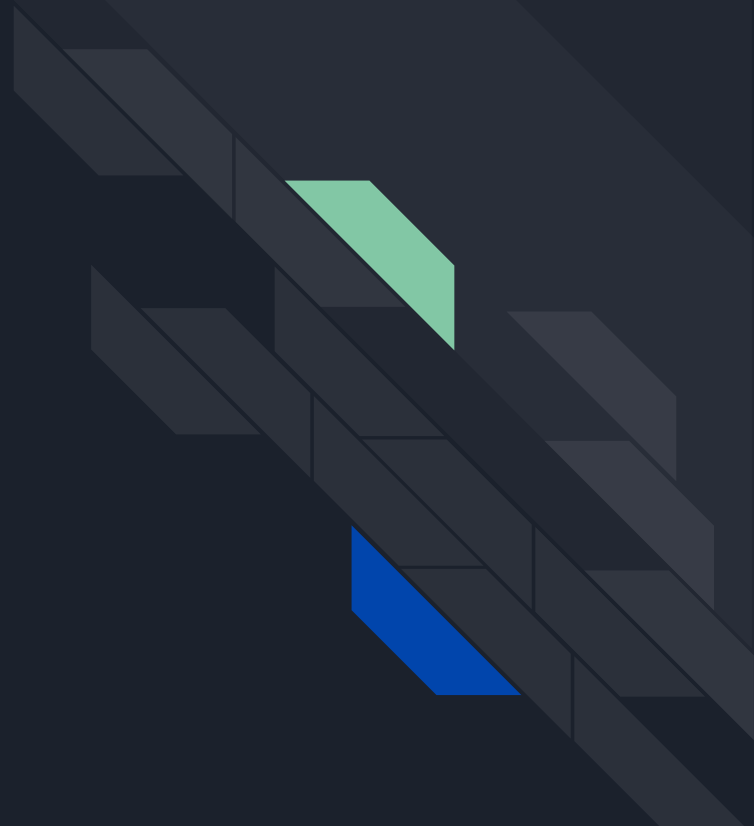


Zadanie praktyczne: dynamiczne dodawanie artykułów

1. Odczytaj listę artykułów z `localStorage` i przeprowadź **rehydrację**
2. Na podstawie odtworzonej listy obiektów klasy `Article` wygeneruj za pomocą JavaScriptu elementy DOM reprezentujące artykuły
 - a. `document.getElementById()`,
`document.querySelector()`
 - b. `document.createElement()`,
`document.innerHTML`
 - c. `domElement.appendChild()`
3. Przycisk **Dodaj** w formularzu powinien dodać do `localStorage` nowy obiekt z artykułem o wprowadzonych danych (autor, tytuł, treść) i bieżącą datą oraz wyświetlić nowo dodany artykuł na liście
4. Bonus: Dodaj przycisk **reset**, który po naciśnięciu wyczyści `localStorage` i spowoduje zniknięcie artykułów

```
<article class="recent">
  <h2>Tytuł</h2>
  <p>Autor</p>
  <p>Wstepniak (getSummary())</p>
  <p><small>Data
publikacji</small></p>
</article>
```

Walidacja danych





Walidacja danych

Walidacja danych w formularzach to proces sprawdzania poprawności wprowadzonych danych przed ich wysłaniem do serwera.

Walidacja może być:

- Po stronie klienta (w przeglądarce, z użyciem JavaScript),
- Po stronie serwera (gdzie dane są ostatecznie weryfikowane przed zapisaniem).

Walidacja po stronie klienta poprawia doświadczenie użytkownika, ale nigdy nie powinna zastępować walidacji po stronie serwera.



Walidacja danych

Rodzaje walidacji

- Walidacja wymagalności (required): Sprawdzenie, czy pole zostało wypełnione.
- Walidacja formatu: Sprawdzenie, czy dane mają odpowiedni format (np. e-mail, numer telefonu, data).
- Walidacja zakresu: Sprawdzenie, czy wartość mieści się w określonym zakresie (np. liczba pomiędzy 1 a 100).
- Walidacja długości: Sprawdzenie, czy dane mają odpowiednią długość (np. hasło musi mieć przynajmniej 8 znaków).



Walidacja danych

Walidacja po stronie klienta za pomocą HTML5

HTML5 wprowadza nowe atrybuty, które ułatwiają walidację danych po stronie klienta:

- required – pole jest wymagane.
- type – określa typ danych, np. email, url, number, date.
- min i max – definiują zakres wartości dla liczb i dat.
- pattern – umożliwia określenie wyrażenia regularnego do walidacji tekstu.

Przykład:

```
<form>
  <input type="email" required placeholder="Wprowadź email">
  <input type="password" minlength="8" required placeholder="Wprowadź hasło">
  <input type="number" min="1" max="100" required>
  <input type="text" id="phone" name="phone" required placeholder="(123) 456-7890"
    pattern="^\(\d{2}\) \d{3}-\d{3}-\d{3}$">
  <button type="submit">Wyślij</button>
</form>
```

UI Kit





Czym są UI Kity?

UI Kit to zestaw gotowych komponentów interfejsu użytkownika, które pomagają programistom tworzyć aplikacje z jednolitym wyglądem i zachowaniem.

UI Kity zawierają elementy takie jak przyciski, formularze, modale, nawigacje, tabelki i inne interaktywne komponenty, które można łatwo używać w aplikacjach.

Zalety korzystania z UI Kitów:

- Szybszy rozwój aplikacji: Dzięki gotowym komponentom nie trzeba budować wszystkiego od podstaw.
- Spójność UI: UI Kit zapewnia jednolity wygląd aplikacji.
- Skalowalność: UI Kity są projektowane z myślą o łatwym skalowaniu aplikacji.



UI Kity w React

- Material-UI (MUI)
- Ant Design
- Chakra UI
- React Bootstrap

Material-UI (MUI):

Styl oparty na Material Design stworzonym przez Google.
Oferuje szeroki wachlarz gotowych komponentów, które można łatwo dostosować.

Ant Design:

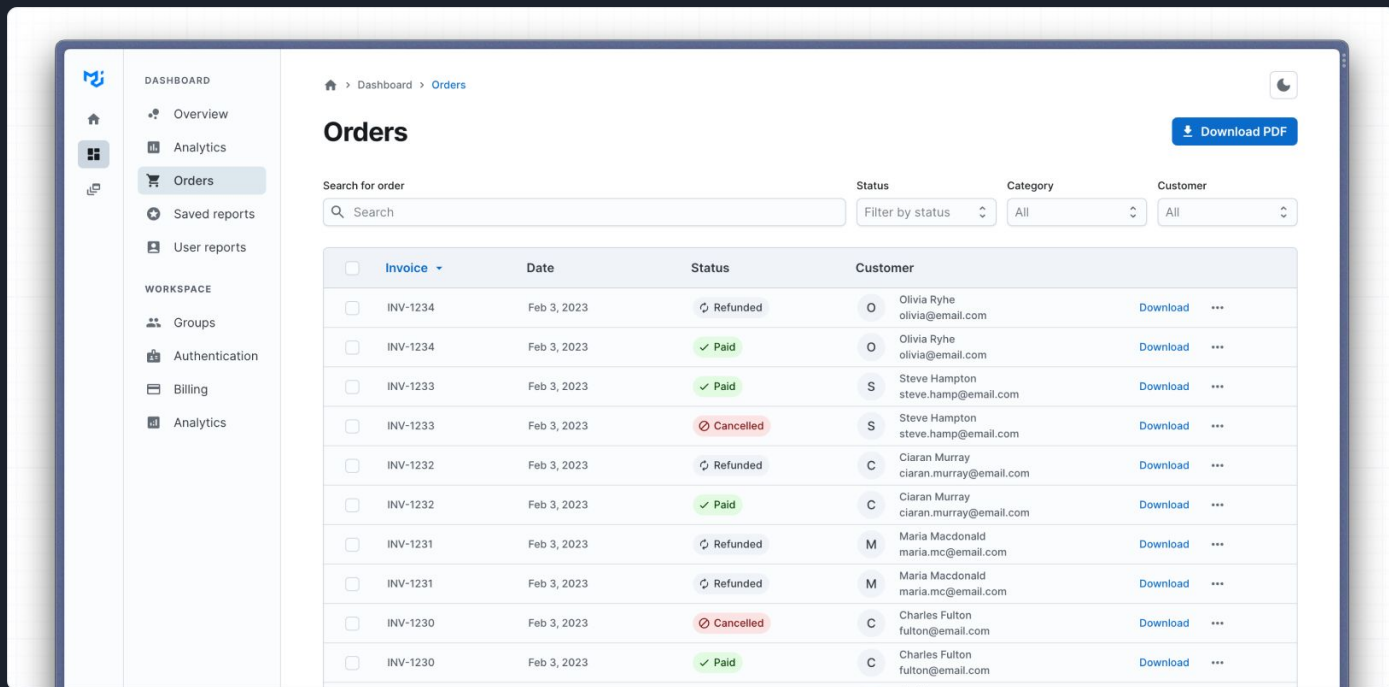
Zestaw komponentów o eleganckim, nowoczesnym wyglądzie.
Używane głównie w aplikacjach korporacyjnych.

Chakra UI:

Bardzo elastyczny i łatwy w użyciu UI Kit dla Reacta, który kładzie duży nacisk na dostępność.
Skupia się na prostocie i elastyczności.

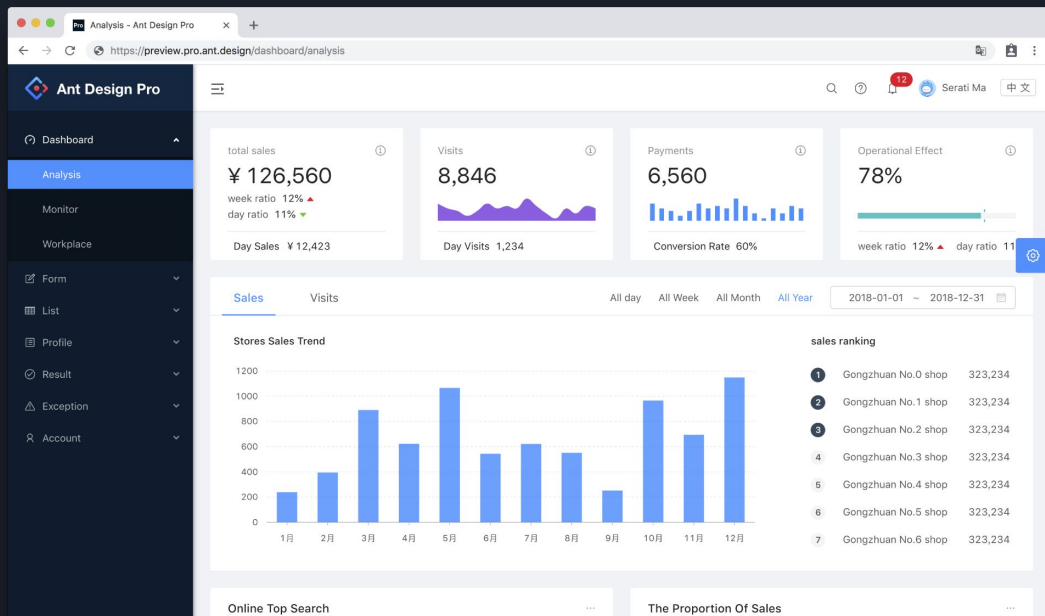
MUI

<https://mui.com/material-ui/all-components/>



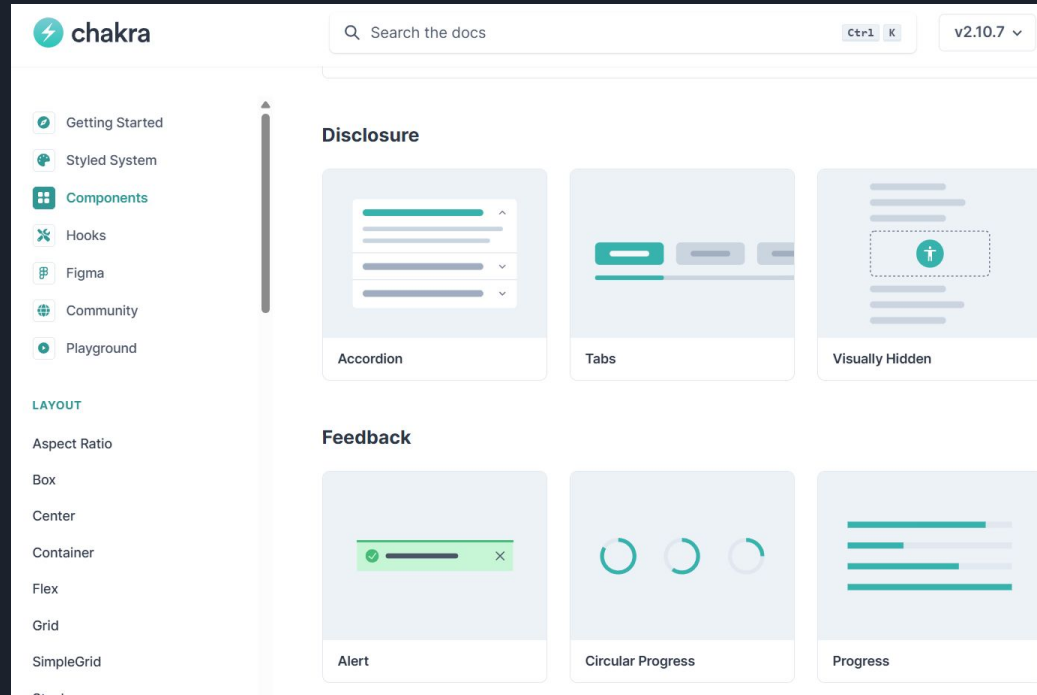
Ant Design

<https://ant.design/>



Chakra UI

<https://v2.chakra-ui.com/docs/components>





jQuery UI

jQuery UI to biblioteka rozszerzająca możliwości jQuery, która dostarcza gotowe komponenty i efekty interfejsu użytkownika. Jest wykorzystywana do tworzenia interaktywnych i dynamicznych elementów na stronach internetowych.

Chociaż jQuery UI była kiedyś bardzo popularną biblioteką do tworzenia interaktywnych elementów UI w aplikacjach webowych, obecnie nie jest już tak szeroko stosowana, a to z kilku powodów:

- Zwiększenie możliwości nowoczesnych frameworków JS
- Brak wsparcia dla nowoczesnych standardów
- Brak wydajności i rozmiar pliku
- Brak aktywnego rozwoju
- Dostępność nowoczesnych alternatyw



Bootstrap

Bootstrap to darmowy, open-source'owy framework, który umożliwia szybkie tworzenie responsywnych i mobilnych stron internetowych oraz aplikacji webowych.

Został stworzony przez Twitter i oferuje gotowe komponenty UI, system siatki responsywnej oraz narzędzia CSS i JavaScript, które ułatwiają tworzenie nowoczesnych stron.

JavaScript jako
technologia serwerowa:
Node.js



Node.js

Node.js to środowisko uruchomieniowe, które pozwala na uruchamianie kodu JavaScript poza przeglądarką (czyli na serwerze).

Zbudowane na Silniku **V8** (używany przez Google Chrome), który kompiluje JavaScript do kodu maszynowego, zapewniając dużą szybkość.

Zastosowanie:

- Tworzenie serwerów HTTP
- Aplikacje internetowe i API
- Rozwiązywanie problemów z dużą liczbą równoczesnych połączeń



Node.js: instalacja

Pobierz Node.js: Wejdź na stronę nodejs.org i pobierz wersję LTS (Long-Term Support).

Windows: Uruchom instalator i postępuj zgodnie z instrukcjami.

Otwórz terminal i sprawdź poprawność instalacji wykonując komendy:

```
node -v  
npm -v
```

Stwórz plik z kodem JavaScript i wykonaj go za pomocą polecenia `node plik.js` w terminalu





Node.js

Szybkość działania (w oparciu o silnik V8)

Node.js korzysta z silnika V8 (tego samego, który jest używany w Google Chrome) do kompilowania JavaScriptu na kod, który działa bardzo szybko.

To oznacza, że aplikacje stworzone w Node.js potrafią przetwarzać dane w czasie rzeczywistym, szybko reagując na działania użytkowników, co jest kluczowe w nowoczesnych aplikacjach internetowych.

Asynchroniczność i obsługa wielu zapytań

Node.js jest asynchroniczny, co oznacza, że może równocześnie obsługiwać wiele zapytań bez blokowania pracy. Wyobraź sobie, że serwer zajmuje się kilkoma rzeczami naraz, nie czekając, aż każda z nich zostanie zakończona przed rozpoczęciem kolejnej.

Na przykład, kiedy użytkownik wysyła żądanie do serwera, ten nie musi czekać na zakończenie odpowiedzi przed obsługą innych użytkowników. To sprawia, że Node.js jest idealne do aplikacji, które muszą obsługiwać dużo użytkowników w tym samym czasie, np. aplikacje do czatowania, gry online czy media społecznościowe.



Node.js

Jednolity język – JavaScript

Node.js pozwala na pisanie kodu zarówno na serwerze, jak i w przeglądarkach w tym samym języku – JavaScript.

Dla programistów to ogromna zaleta, bo mogą wykorzystać ten sam kod i narzędzia na obu stronach (serwerze i kliencie). Dla osób zarządzających projektami oznacza to, że nie trzeba zatrudniać specjalistów do obsługi różnych języków na serwerze i kliencie – wszystko jest napisane w JavaScript.

Ogromna społeczność i ekosystem

Dzięki npm (Node Package Manager), który jest częścią Node.js, masz dostęp do milionów gotowych paczek i bibliotek, które pomagają rozwiązywać różne problemy. Chcesz zrobić coś specjalnego? W większości przypadków ktoś już to napisał, i wystarczy to zaimplementować w projekcie.

Np. jeśli chcesz, by aplikacja wysyłała e-maile, możesz łatwo zainstalować paczkę, która to umożliwia, zamiast pisać całą logikę od podstaw.



Node.js

Skalowalność – idealne dla rosnących aplikacji

Node.js jest wyjątkowo skalowalny, co oznacza, że łatwo można go używać do aplikacji, które muszą radzić sobie z dużą liczbą użytkowników. Na przykład, możesz łatwo dodać kolejne serwery, by rozdzielić obciążenie, co umożliwia skalowanie aplikacji w miarę jak rośnie liczba użytkowników.

Skalowalność jest kluczowa dla aplikacji, które mają potencjał do szybkiego wzrostu (np. startupy, aplikacje mobilne).

Zastosowanie w aplikacjach w czasie rzeczywistym

Node.js jest idealne do tworzenia aplikacji, które wymagają natychmiastowej komunikacji z użytkownikiem, np. czaty, powiadomienia w czasie rzeczywistym, aplikacje do współpracy, czy gry online.

Dzięki asynchronicznemu modelowi działania, Node.js umożliwia natychmiastowe przesyłanie danych między użytkownikami bez opóźnień.



Node.js

Wielu deweloperów używa go do prototypowania i rozwoju MVP (Minimum Viable Product)

Ze względu na swoją prostotę, szybkość działania i duży zbiór gotowych bibliotek, Node.js jest popularnym wyborem wśród startupów do szybkiego tworzenia prototypów i minimalnych wersji produktów (MVP). Daje to możliwość testowania pomysłów na rynku i dostosowywania produktów w zależności od reakcji użytkowników.



Zmienne i ich zakres (scope)

Scope (zakres) zmiennych w JavaScript odnosi się do obszaru, w którym zmienne są widoczne i dostępne.

Określa, gdzie zmienna może być używana w kodzie.

W JavaScript istnieją różne typy zakresów zmiennych, które określają, gdzie zmienne są dostępne.

- Global Scope
- Function Scope
- Block Scope



Zmienne i ich zakres: global scope

Zmienna zadeklarowana poza funkcją lub blokiem jest dostępna w całym kodzie.

Można jej używać w różnych częściach programu.

```
let globalVar = "Jestem zmienną globalną";

function example() {
  console.log(globalVar); // Dostęp do zmiennej globalnej
}

example(); // Wypisze: Jestem zmienną globalną
```



Zmienne i ich zakres: function scope

Zmienna zadeklarowana wewnątrz funkcji jest dostępna tylko w tej funkcji. Nie jest dostępna poza funkcją.

```
function myFunction() {  
  let functionVar = "Jestem zmienną funkcji";  
  console.log(functionVar); // Działa  
}
```

```
myFunction();  
console.log(functionVar); // Błąd! Zmienna poza funkcją nie jest dostępna
```




Zmienne i ich zakres: block scope

Wprowadzone w ES6 (ECMAScript 2015) z `let` i `const`.

Zmienna zadeklarowana w obrębie bloku (`{}`) jest dostępna tylko w tym bloku.

```
if (true) {  
  let blockVar = "Jestem zmienną blokową";  
  console.log(blockVar); // Dostępna tylko wewnątrz bloku  
}
```

```
console.log(blockVar); // Błąd! Zmienna nie jest dostępna poza blokiem
```

Zmienne i ich zakres: dlaczego warto ograniczać scope zmiennych?

Zalety:

- Dostępność w całej aplikacji
- Prostota w małych projektach

Wady:

- Potencjalne kolizje nazw
- Trudności w debugowaniu
- Zwiększenie ryzyka błędów
- Trudniejsze utrzymanie kodu



Zadania praktyczne: tablice

Napisz funkcję `groupEvenOdd(numbers)`, która zwróci obiekt `{ even: [...], odd: [...] }` gdzie:

- `even` — tablica liczb parzystych,
- `odd` — tablica liczb nieparzystych.

Przydatne funkcje:

- `a % b` - zwraca resztę z dzielenia `a` przez `b`

Stwórz funkcję `removeDuplicates(arr)`, która zwróci nową tablicę bez powtarzających się elementów.

Przydatne funkcje:

- `Array.includes()`



Zadania praktyczne: tablice

Napisz funkcję `flattenOneLevel(arr)`, która "spłaszczy" tablicę zagnieżdżoną o jeden poziom.

```
flattenOneLevel([1, [2, 3], [4, 5], 6]);  
  
// [1, 2, 3, 4, 5, 6]
```

Wskazówki:

- `typeof`
- `Array.isArray()`

Stwórz histogram występowania liter w napisie.

Napisz funkcję `letterFrequency(text)`, która zwróci **obiekt**, w którym kluczami będą litery, a wartościami liczba ich wystąpień w napisie.

```
letterFrequency("hello");  
  
// { h: 1, e: 1, l: 2, o: 1 }  
  
// let obj = {x: 1, y: 2}  
  
// do kluczy w obiekcie można odwoływać się  
// za pomocą nawiasów kwadratowych  
// obj["z"] = 3; obj.z = 3 - te zapisy są  
// tożsame
```



Zadania praktyczne: obiekty

Napisz funkcję `remapValues(obj, mapping)`, która zwróci nowy obiekt, gdzie każda wartość z `obj` zostanie zamieniona zgodnie z regułami zapisanymi w `mapping`.

```
remapValues({status: "pending"}, {pending:
"w toku", done: "zakończone"})
```

```
// wynik: {status: "w toku"}
```

Wskazówki:

- Nie modyfikuj oryginalnego `obj`, zwróć nowy obiekt.
- Iteracja po kluczach obiektu: `for...in` albo `Object.entries()`.
- `obiett[k] == obiekt.k`
- Operator `in` lub `Object.hasOwnProperty` do sprawdzania czy obiekt posiada konkretny klucz

Symulacja gry - rzut kostką

Stwórz obiekt `diceGame`, który ma metodę `roll()`, symulującą rzut sześcienną kostką (losowanie liczby od 1 do 6), oraz metodę `play()`, która wywołuje `roll()` 5 razy i wypisuje wyniki.

Wskazówki:

- `Math.random()` - losowa wartość między 0 a 1
- `Math.floor(x)` - podłoga, najbliższa liczba całkowita mniejsza lub równa `x`



Zadania praktyczne: obiekty

Stwórz obiekt `shoppingCart`, który przechowuje tablicę produktów oraz ma metodę `addProduct(product)`, dodającą produkt do koszyka, oraz metodę `getTotal()`, obliczającą łączną cenę wszystkich produktów w koszyku.

Produkty:

- `{ name: "Laptop", price: 2000 }`
- `{ name: "Tablet", price: 1500 }`
- `{ name: "Mouse", price: 150 }`

Stwórz obiekt `bankAccount`, który ma właściwości:

- `balance` (początkowa wartość to 0),
- `transactions` (tablica przechowująca historię transakcji).

Dodaj do obiektu następujące metody:

- `deposit(amount)` — wpłata na konto,
- `withdraw(amount)` — wypłata z konta,
- `getBalance()` — zwraca saldo konta,
- `getTransactions()` — zwraca historię transakcji zawierającą typ transakcji (depozyt | pobranie) i kwotę operacji



Funkcje strzałkowe: przypomnienie

Skrócona składnia dla funkcji anonimowych w JavaScript. Wprowadzone w ECMAScript 6 (ES6).

- Nie posiadają własnego `this`, `arguments`, `super` ani `new.target`
- `this` jest dziedziczone z otaczającego kontekstu (tzw. "lexical this")

```
// Tradycyjna funkcja
function dodaj(a, b) {
  return a + b;
}
```

```
// Funkcja strzałkowa
const dodaj = (a, b) => a + b;

const dodaj = (a, b) => {
  const suma = a + b;
  return suma; // wymagany return
};
```



Funkcje asynchroniczne: setTimeout

Funkcja służąca do jednorazowego opóźnienia wykonania kodu. Część wbudowanego API JavaScript (funkcja asynchroniczna).

```
setTimeout(callback, delay, [arg1, arg2, ...]);
```

- callback - funkcja do wykonania
- delay - opóźnienie w milisekundach
- [arg1, arg2, ...] - opcjonalne argumenty przekazywane do callback

```
setTimeout(() => {  
  console.log("Witaj po 2 sekundach!");  
}, 2000);
```

```
setTimeout((name) => {  
  console.log("Witaj po 2 sekundach, ", name);  
}, 2000, "Janek");
```



Zadania praktyczne: funkcje

Użyj funkcji strzałkowej w metodzie `map()` do podwojenia wszystkich liczb w tablicy.

tablica wejściowa:

```
const numbers = [1, 2, 3, 5, 8, 13, 21];
```

Użyj funkcji strzałkowej w metodzie `filter()`, aby zwrócić tylko liczby większe niż 10 z tablicy po podwojeniu wartości.

Napisz funkcję, która wypisze w konsoli cyfry od 1 do 5 w odstępach 1-sekundowych.

Wskazówki:

- `setTimeout (callback)`
- można użyć funkcji anonimowej lub strzałkowej jako `callback`
- funkcje anonimowe i strzałkowe mogą korzystać ze zmiennych globalnych

Obsługa wyjątków





Co to są wyjątki?

Wyjątek to sytuacja, w której program napotyka błąd lub problem podczas wykonywania, który uniemożliwia kontynuowanie normalnego przepływu programu.

Wyjątki mogą występować z różnych powodów, np. dzielenie przez zero, brak dostępu do pliku, czy niepoprawne dane wejściowe.

Typowe przykłady błędów w JavaScript:

- `SyntaxError` – błąd składniowy.
- `ReferenceError` – odwołanie do niezdefiniowanej zmiennej.
- `TypeError` – próba wykonania nieprawidłowej operacji na zmiennej.
- `RangeError` – błędny zakres liczby (np. przekroczenie dozwolonego zakresu dla tablicy).



Jak działają wyjątki w JavaScript?

Kiedy wystąpi błąd, program zatrzymuje swoje działanie.

JavaScript przechodzi do najbliższego bloku `catch`, jeśli wyjątek jest obsługiwany i wystąpił w bloku `try`.

Jeśli wyjątek nie zostanie obsługowany, program zakończy działanie i wyświetli błąd w konsoli.

- `try` – blok, w którym wykonujemy kod, który może wygenerować wyjątek.
- `catch` – blok, w którym przechwytyjemy wyjątek i obsługujemy go.
- `finally` – blok, który zawsze się wykona, niezależnie od tego, czy wyjątek wystąpił, czy nie.

```
try {  
    // Kod, który może generować wyjątek  
    throw new Error("Coś poszło nie tak!"); //  
    Rzucenie wyjątku  
} catch (error) {  
    // Obsługa wyjątku  
    console.log("Błąd:", error.message);  
}  
finally {  
    // Kod, który zawsze się wykona  
    console.log("Zawsze się wykona");  
}
```



Przykład: Obsługa wyjątku

```
try {  
  let jsonString = "Niepoprawny JSON";  
  let object = JSON.parse(jsonString);  
  console.log(object);  
} catch (error) {  
  console.log("Wystąpił błąd:", error.message);  
} finally {  
  // blok finally jest opcjonalny  
  console.log("Blok finally wykonany");  
}
```



Rzucanie wyjątków (Throwing Exceptions)

Używając słowa kluczowego `throw`, możemy rzucić własny wyjątek, jeśli napotkamy nieoczekiwany stan w programie.

```
function checkAge(age) {  
  if (age < 0) {  
    throw new Error("Wiek nie może być mniejszy niż 0");  
  }  
  console.log("Wiek:", age);  
}  
  
try {  
  checkAge(-5);  
} catch (error) {  
  console.log(error.message); // "Wiek nie może być mniejszy niż 0"  
}
```



Obsługa różnych typów wyjątków

Możemy dostosować blok catch, aby obsługiwał różne rodzaje wyjątków w zależności od ich typu.

```
try {
  let obj = {};
  console.log(obj.someMethod()); // Błąd typu (TypeError)
} catch (error) {
  if (error instanceof TypeError) {
    console.log("Błąd typu:", error.message);
  } else if (error instanceof ReferenceError) {
    console.log("Błąd odniesienia:", error.message);
  } else {
    console.log("Inny błąd:", error.message);
  }
}
```

Obsługa wyjątków: Typy błędów

Wbudowane typy błędów w JavaScript

Typ błędu	Opis	Przykład
Error	Bazowy typ błędu. Można go używać do własnych błędów	<code>throw new Error("Coś poszło nie tak")</code>
TypeError	Operacja wykonana na niewłaściwym typie	<code>null.funkcja()</code>
ReferenceError	Odwwołanie do niezdefiniowanej zmiennej	<code>console.log(x)</code> gdzie <code>x</code> nie istnieje
SyntaxError	Błąd składni (np. w <code>eval</code> lub podczas parsowania)	<code>eval("if) {")</code>
RangeError	Wartość spoza dozwolonego zakresu	<code>new Array(-1)</code>
URIError	Nieprawidłowe kodowanie/dekodowanie URI	<code>decodeURIComponent('%E0%A4%A')</code>
EvalError	Błędy związane z <code>eval()</code> (praktycznie nieużywany)	Występuje bardzo rzadko



Obsługa wyjątków: Tworzenie własnych typów błędów

```
class MojBlad extends Error {  
    constructor(komunikat) {  
        super(komunikat);  
        this.name = "MojBlad";  
    }  
}
```




Popularne funkcje rzucające wyjątki

`JSON.parse()`

```
JSON.parse('{"x":1}'); // OK
JSON.parse('x:1');     // SyntaxError
```

`decodeURIComponent()`

```
decodeURIComponent('%E0%A4%A'); // URIError
```

`fs.readFileSync`

```
const fs = require('fs');
fs.readFileSync('nieistnieje.txt'); // Error: ENOENT (Error NO ENTry)
```



Walidacja danych: zadanie praktyczne

1. Stwórz listę słów zakazanych w kodzie JS naszego bloga.
2. Przy dodawaniu nowego artykułu sprawdź, czy pola autor, tytuł i treść nie zawierają żadnego ze słów zakazanych.
3. Powiadom użytkownika, jeżeli tak jest.
4. Bonus: Przeprowadź walidację rzucając i obsługując wyjątek.

Wskazówki:

- `document.body`, `document.querySelector()`, `document.getElementById()`
- `addEventListener("submit", function)`
- `e.target.type` i `e.target.value`
- `pętle`, `Array.forEach()` lub `Array.some()`
- `String.includes(keyword)`