JavaScript

mgr inż. Katarzyna Dadek

2025

1 Wprowadzenie do JavaScript. Podstawowe zastosowania i struktury językowe.

JavaScript (JS) to język programowania stworzony w 1995 roku przez w firmie Netscape. Początkowo został zaprojektowany jako prosty skryptowy język do użytku w przeglądarkach internetowych, aby umożliwić tworzenie dynamicznych stron WWW. Dzięki JavaScript można:

- Dynamicznie manipulować zawartością strony internetowej (zmieniać tekst, obrazy, kolory, układ).
- Obsługiwać interakcje użytkownika (np. kliknięcia, ruch myszką, wprowadzanie danych).
- Wysyłać i odbierać dane z serwera bez przeładowywania strony (AJAX).
- Tworzyć efekty wizualne i animacje.
- Budować aplikacje webowe typu Single Page Application (SPA).
- Wykorzystywać w backendzie (np. za pomocą Node.js).

1.1 Krótka historia JavaScript

1995 Brendan Eich stworzył pierwszą wersję języka w zaledwie 10 dni. Początkowo nosił nazwę Mocha, potem LiveScript, aż w końcu został przemianowany na JavaScript ze względu na popularność języka Java (choć technicznie nie mają ze sobą nic wspólnego).

1997 ECMA (European Computer Manufacturers Association) opublikowała pierwszą specyfikację języka jako ECMAScript (ES1) – JavaScript to implementacja ECMAScript.

2009 Powstaje Node.js, umożliwiający używanie JavaScript na serwerze (backend).

2015 Wydano ES6 (ECMAScript 2015), która wprowadziła kluczowe nowości, jak let, const, klasy, funkcje strzałkowe i obietnice (Promise).

Obecnie JavaScript stał się podstawowym językiem programowania na frontendzie i backendzie. Jest używany w popularnych frameworkach takich jak React, Angular i Vue oraz w narzędziach serwerowych (Node.js).

Ważne

JavaScript jest językiem **interpretowanym**, dynamicznym i prototypowym — co oznacza, że kod jest wykonywany bezpośrednio w przeglądarce bez potrzeby wcześniejszej kompilacji. Ponadto JS działa na zasadzie Event-Driven i jest oparty na modelu **asynchronicznym**.

1.2 Podstawowe typy danych w JavaScript

W JavaScript istnieje siedem podstawowych typów danych — sześć typów prymitywnych + jeden obiektowy.

1.2.1 String

Tekst (ciąg znaków), zapisywany w pojedynczych (') lub podwójnych (") cudzysłowach albo w tzw. backtickach (`).

```
let text = "Hello, world!";
let name = 'John';
let template = `Witaj, ${name}`;
```

1.2.2 Number

Liczba (całkowita lub zmiennoprzecinkowa), JavaScript ma tylko jeden typ liczbowy (nie ma oddzielnego typu dla liczb całkowitych i zmiennoprzecinkowych).

```
let a = 42;
let b = 3.14;
```

1.2.3 Boolean

Wartość logiczna *true* lub *false*.

```
let isActive = true;
let isLoggedIn = false;
```

1.2.4 Undefined

Zmienna została zadeklarowana, ale nie nadano jej wartości.

```
let value;
console.log(value); // undefined
```

1.2.5 Null

Oznacza “brak wartości” (ustawiane ręcznie).

```
let data = null;
```

1.2.6 Symbol

Unikatowa wartość (rzadko używane).

```
let id = Symbol('unique');
```

1.2.7 BigInt

Liczby większe niż Number.MAX_SAFE_INTEGER.

```
let bigNumber =
  12345678901234567890123456789012345678901234567890n;
```

1.3 Typ obiektowy (Object)

Obiekty w JavaScript to kolekcje par klucz-wartość. Klucz jest zawsze typu string lub Symbol, a wartość może być dowolnym typem. Na obiekcie można wykonywać różne operacje. W przypadku gdy obiekt jest zdefiniowany jako *const* to nie można zmienić przypisania tego obiektu, jednakże można zmieniać jego pola.

```
let person = {
  firstName: "John",
  lastName: "Doe",
  age: 30,
  isEmployed: true
};
```

1.3.1 Dostęp do właściwości obiektu

```
console.log(person.firstName); // John
console.log(person["age"]); // 30
```

1.3.2 Dodawanie nowych właściwości

```
person.job = "Developer";
```

1.3.3 Usuwanie właściwości

```
delete person.age;
```

1.3.4 Dostęp do obiektów zagnieżdżonych

```
let company = {
  name: "TechCorp",
  location: {
    city: "New York"
  }
};
console.log(company.location.city); // New York
```

1.4 JSON (JavaScript Object Notation)

JSON to tekstowy format zapisu danych oparty na składni obiektów JavaScript. Służy do wymiany danych między aplikacjami (np. frontend ↔ backend).

```
{
  "name": "John",
  "age": 30,
  "isEmployed": true
}
```

JavaScript udostępnia dwie wbudowane metody do konwersji JSON ↔ obiekt.

1.4.1 JSON.parse() – konwersja JSON → obiekt JavaScript

- Metoda JSON.parse() zamienia tekstowy JSON na obiekt JavaScript.
- Wartości w JSON zostaną przekonwertowane na odpowiednie typy w JS.
- Dla niepoprawnej struktury zostanie wygenerowany błąd (SyntaxError).

```
const jsonString = '{"name":"John","age":30,"isEmployed":
    true}';
const obj = JSON.parse(jsonString);

console.log(obj.name); // John
console.log(obj.age); // 30
console.log(obj.isEmployed); // true
```

1.4.2 JSON.stringify() – konwersja obiekt JavaScript → JSON

- Metoda JSON.stringify() zamienia obiekt JavaScript na tekst JSON.
- undefined, funkcje, symbole i inne niestandardowe typy nie będą uwzględnione
- Można użyć drugiego argumentu (replacer).

```
const person = {
  name: 'John',
  age: 30,
  isEmployed: true,
  skills: ['JavaScript', 'TypeScript', 'React']
};

const jsonString = JSON.stringify(person);
console.log(jsonString);
// {"name":"John","age":30,"isEmployed":true,"skills":["
    JavaScript","TypeScript","React"]}
```

1.5 Zmienne w JavaScript – var, let, const

W JavaScript zmienne służą do przechowywania danych w pamięci, aby można było ich użyć i modyfikować w kodzie. Każda zmienna ma swoją nazwę, zakres (ang. scope) i typ przypisanej wartości. W JavaScript mamy trzy sposoby deklarowania zmiennych:

1. **var** – to najstarszy sposób deklarowania zmiennych w JavaScript, wprowadzony w pierwszej wersji języka. Ma zakres funkcyjny, co oznacza że zmienna zadeklarowana za pomocą var **jest dostępna w całej funkcji**, w której została zadeklarowana. Zmienne zadeklarowane przez var są **hoistowane** (czyli przenoszone na początek zakresu).

```
function example() {
  if (true) {
    var x = 10;
  }
  console.log(x); // 10
}

example();
```

W powyższym przykładzie zmienna x jest dostępna poza blokiem *if*, ponieważ *var* ma zakres funkcyjny (a nie blokowy).

> **Ważne**
>
> **Hoisting** to mechanizm w JavaScript, w którym deklaracje zmiennych i funkcji są automatycznie "podnoszone" (ang. hoisted) na początek swojego zakresu (scope) podczas fazy kompilacji — przed wykonaniem kodu. Oznacza to, że zmienne zadeklarowane za pomocą var, let, const oraz funkcje są dostępne w kodzie przed ich deklaracją, choć ich zachowanie różni się w zależności od sposobu deklaracji.

2. **let** – zmienne o zakresie blokowym (block scope). Został wprowadzony w ES6 (2015) i rozwiązał problemy związane z var ponieważ ma zakres blokowy – zmienna zadeklarowana w bloku {} jest dostępna tylko wewnątrz tego bloku. Jest on również hoistowany, ale nie można użyć zmiennej przed jej deklaracją.

```
function example() {
  if (true) {
    let x = 10;
    console.log(x); // 10
  }
  console.log(x); // Error: x is not defined
}
example();
```

3. **const** – deklaracja stałych (zmienne, których wartości nie można zmienić). Został wprowadzony w ES6 (2015) i służy do deklarowania stałych – zmiennych, których wartość nie może zostać zmieniona po przypisaniu. Ma zakres blokowy – tak jak let, wartość przypisana do const jest niezmienna (chociaż w przypadku

obiektów i tablic – ich zawartość może być modyfikowana). Co ważne - konieczne jest przypisanie wartości już podczas deklaracji.

```
const x = 10;
x = 20; // TypeError: Assignment to constant variable.
```

Ważne

Problemy z var:

- nie ma zakresu blokowego → może prowadzić do nieoczekiwanych rezultatów.
- może zostać nadpisany przypadkowo w innym miejscu kodu

1.5.1 Kiedy używać let, const, var?

- ✔ Używaj const zawsze, gdy wartość nie będzie zmieniana.
- ✔ Używaj let kiedy wiesz, że wartość zmiennej będzie zmieniana.
- ✘ Unikaj var – w nowoczesnym kodzie JavaScript var jest przestarzałe i należy go unikać.

1.5.2 Dobre praktyki

- ✔ Używaj const domyślnie – jeśli wartość musi się zmienić → zmień na let..
- ✔ Unikaj deklaracji zmiennych globalnych → ogranicz zakres zmiennych.
- ✔ Używaj camelCase do nazw zmiennych (*myVariable*).
- ✔ Nazwy stałych pisz wielkimi literami (*MAX_SIZE*).

1.6 Instrukcje warunkowe (if, else, switch)

Instrukcje warunkowe w JavaScript pozwalają na podejmowanie decyzji w zależności od spełnienia określonych warunków. Pozwalają one na kontrolę przepływu programu w oparciu o wartość wyrażeń logicznych (*true* lub *false*).

JavaScript udostępnia trzy podstawowe konstrukcje warunkowe:

1. **if** – wykonuje blok kodu, jeśli warunek jest prawdziwy.

```
let age = 20;

if (age >= 18) {
  console.log("You can vote!");
}
```

2. **if...else** – wykonuje jeden z dwóch bloków kodu w zależności od wartości warunku.

```
let age = 20;

if (age >= 18) {
  console.log("You can vote!");
} else {
  console.log("You can't vote!");
}

let score = 75;

if (score >= 90) {
  console.log("Ocena: A");
} else if (score >= 75) {
  console.log("Ocena: B");
} else if (score >= 50) {
  console.log("Ocena: C");
} else {
  console.log("Ocena: F");
}

let message = age >= 18 ? "Can vote" : "Can't vote";
console.log(message); // Can vote
```

3. **switch** – pozwala na obsługę wielu przypadków (case).

```
let day = 3;
switch (day) {
  case 1:
  console.log("Monday");
  break;
  case 2:
  console.log("Tuesday");
  break;
  default:
  console.log("Other");
}
```

```
let color = 'red';
switch (color) {
  case 'red':
  console.log("Kolor czerwony");
  case 'blue':
  console.log("Kolor niebieski");
  default:
  console.log("Inny kolor");
}
```

1.7 Pętle (for, while)

Pętle w JavaScript pozwalają na powtarzanie fragmentu kodu wielokrotnie, dopóki spełniony jest określony warunek. Są one niezwykle przydatne do automatyzacji zadań, pracy z tablicami, przetwarzania danych i wielu innych operacji. JavaScript udostępnia kilka rodzajów pętli:

1. **for** – klasyczna pętla z licznikiem

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

```
const fruits = ['apple', 'banana', 'orange'];

for (let i = 0; i < fruits.length; i++) {
  console.log(fruits[i]);
}
```

2. **while** – wykonuje blok kodu, dopóki podany warunek jest prawdziwy (*true*).

```
let i = 0;

while (i < 5) {
  console.log(i);
  i++;
}
```

3. **do...while** - działa jak while, z tą różnicą, że warunek jest sprawdzany po wykonaniu bloku kodu. Pętla **wykona się przynajmniej raz**, nawet jeśli warunek jest fałszywy na początku.

```
let i = 10;

do {
  console.log(i);
  i++;
} while (i < 5);
```

4. **for...in** – służy do iteracji po **kluczach** obiektów. Działa na wszystkich własnościach obiektu (włącznie z właściwościami odziedziczonymi z prototypu). Nie używaj for...in do tablic → używaj for...of lub klasycznego for.

```
const person = {
  name: 'John',
  age: 30,
  job: 'Developer'
};

for (let key in person) {
  console.log(`${key}: ${person[key]}`);
}
```

5. **for...of** – iteruje po **wartościach** elementów w tablicach, ciągach znaków i innych strukturach iterowalnych.

```
const fruits = ['apple', 'banana', 'orange'];

for (let fruit of fruits) {
  console.log(fruit);
}
```

Ważne

W pętlach można używać instrukcji takich jak ***break*** (przerywa działanie pętli) i ***continue*** (pominięcie bieżącej iteracji i przejście do następnej).

Zadania

1.1. Utwórz plik script.js i połącz go z plikiem HTML za pomocą tagu <script>.

1.2. W konsoli przeglądarki wyświetl komunikat "Witaj w świecie JavaScript!" za pomocą console.log().

1.3. Sprawdź typ wartości *42*, *"Hello"*, *true* oraz *null* za pomocą typeof.

1.4. Sprawdź, co się stanie, jeśli spróbujesz dodać liczbę i tekst (np. 10 + "5").

1.5. Zadeklaruj zmienne *name*, *age*, *isStudent* i przypisz do nich odpowiednio: tekst, liczbę i wartość logiczną.

1.6. Utwórz tablicę *colors* z trzema nazwami kolorów i wyświetl pierwszy oraz ostatni element tablicy.

1.7. Utwórz obiekt *person* zawierający klucze *name*, *age*, *isEmployed* i przypisz im odpowiednie wartości.

1.8. Utwórz zmienną *data* i przypisz do niej wartość *null*, a następnie wyświetl jej typ.

1.9. Sprawdź, co się stanie, jeśli spróbujesz podzielić liczbę przez 0.

1.10. Zadeklaruj zmienną *x* za pomocą *var* i przypisz jej wartość 5. Następnie zmień jej wartość na 10 i wyświetl wynik w konsoli.

1.11. Zadeklaruj zmienną y za pomocą *let* wewnątrz bloku . Sprawdź, czy jest dostępna poza blokiem.

1.12. Zadeklaruj stałą z za pomocą *const* i przypisz jej wartość "JavaScript". Spróbuj zmienić wartość tej stałej i sprawdź, co się stanie.

1.13. Utwórz tablicę za pomocą const, dodaj nowy element do tablicy i sprawdź, czy operacja się powiedzie.

1.14. Napisz kod, który pokaże różnicę między *var* i *let* w zakresie blokowym (scope).

1.15. Użyj hoistingu, aby zadeklarować zmienną za pomocą *var* po jej użyciu w konsoli. Sprawdź wynik.

1.16. Napisz program, który sprawdzi, czy podana liczba *n* jest parzysta czy nieparzysta (użyj if...else).

1.17. Napisz program, który sprawdzi, czy dana liczba x jest dodatnia, ujemna czy równa zero (użyj if...else).

1.18. Napisz kod, który na podstawie wartości zmiennej *grade* (od 1 do 6) zwróci ocenę słowną (1 = niedostateczny, 6 = celujący) przy użyciu *switch*.

1.19. Napisz kod, który sprawdzi, czy podana zmienna *year* jest rokiem przestępnym (użyj if...else).

1.20. Napisz program, który użyje operatora warunkowego (?) do przypisania wartości "pełnoletni" lub "niepełnoletni" na podstawie wieku.

1.21. Użyj pętli *for*, aby wyświetlić liczby od 1 do 10 w konsoli.

1.22. Użyj pętli *for*, aby obliczyć sumę liczb od 1 do 100.

1.23. Użyj pętli *while*, aby wyświetlić liczby od 5 do 0 w konsoli (malejąco).

1.24. Napisz program, który użyje pętli *for*, aby wyświetlić liczby podzielne przez 3 od 1 do 30.

1.25. Użyj pętli *for...of*, aby wyświetlić elementy tablicy fruits = ["apple", "banana", "orange"].

1.26. Użyj pętli *for...in*, aby wyświetlić klucze i wartości obiektu name: "John", age: 30, city: "New York" .

1.27. Napisz program, który użyje *while* do wygenerowania liczb od 1 do 10, ale przerwie działanie po osiągnięciu 7 (użyj break).

1.28. Użyj pętli *for*, aby znaleźć największą liczbę w tablicy [12, 45, 67, 2, 89, 34].

1.29. Napisz program, który użyje *continue*, aby pominąć liczby podzielne przez 4 w zakresie od 1 do 20.

1.30. Napisz kod, który za pomocą pętli *for...in* i *typeof* sprawdzi typ każdej właściwości obiektu name: "John", age: 30, isEmployed: true .

1.31. QUIZ Napisz program, który poprosi użytkownika o wprowadzenie liczby i sprawdzi, czy jest ona liczbą pierwszą.

1.32. QUIZ Napisz program, który wygeneruje tablicę z liczbami od 1 do 50, a następnie użyje pętli *for...of* do wyświetlenia tylko liczb parzystych.

1.33. QUIZ Napisz kod, który przy użyciu *switch* przypisze do zmiennej *dayType* wartość "weekend" lub "weekday" na podstawie numeru dnia tygodnia.

1.34. QUIZ Napisz program, który obliczy sumę wszystkich liczb podzielnych przez 3 i 5 w zakresie od 1 do 100.

1.35. QUIZ Napisz kod, który sprawdzi, czy podane słowo jest palindromem (czyta się tak samo od przodu i od tyłu).

1.36. QUIZ Napisz program, który utworzy obiekt z danymi użytkownika (name, age, email), a następnie za pomocą *for...in* wyświetli wszystkie klucze i wartości.

1.37. QUIZ Napisz kod, który użyje pętli *for*, aby odwrócić kolejność elementów w tablicy bez użycia wbudowanej metody *.reverse()*.

1.38. QUIZ Napisz kod, który przy użyciu operatora warunkowego (?) zwróci informację, czy podana liczba jest podzielna przez 2 i 3.

1.39. QUIZ Napisz program, który znajdzie najmniejszą i największą liczbę w tablicy [15, 42, 7, 23, 67, 1, 90].

1.40. QUIZ Napisz kod, który użyje instrukcji switch do wyświetlenia nazwy miesiąca na podstawie podanego numeru (np. 1 = styczeń).