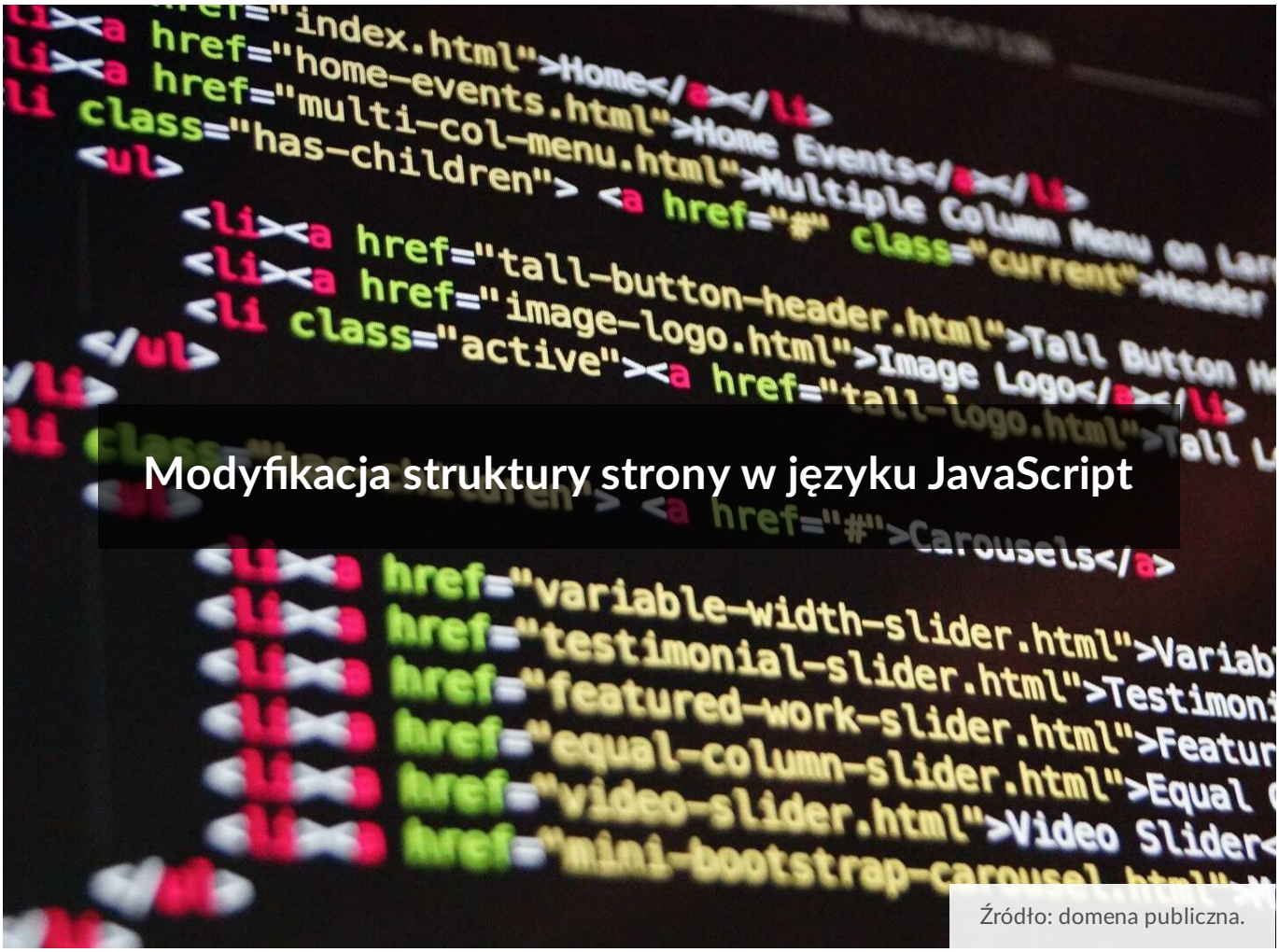

# Modyfikacja struktury strony w języku JavaScript




Źródło: domena publiczna.

Strukturę strony internetowej z serwera przesyłamy w postaci danych tekstowych z użyciem języków znaczników, między innymi HTML i CSS. Dzięki temu witryna internetowa pozostaje niezależna od platformy sprzętowej czy systemu operacyjnego.

Jednak taki tekstowy opis witryny musi zostać docelowo **zamieniony przez przeglądarkę** na **model obiektowy** znajdujący się w pamięci operacyjnej komputera klienckiego. Jest to tzw. Document Object Model, który umożliwia programiście JavaScript modyfikację właściwości oraz zachowania istniejących obiektów.

W tym e-materiale omówimy możliwości, jakie udostępnia wspomniany model – nauczymy się **modyfikować strukturę strony** z użyciem języka JavaScript.

Pozostałe e-materiały z tej serii:

- Tworzenie stron internetowych,
- Język HTML i podstawy CSS,
- Szkielet strony internetowej,
- Podstawy języka JavaScript,
- Obsługa zdarzeń w języku JavaScript,
- Zewnętrzne biblioteki wspierające tworzenie stron internetowych.

**Twoje cele**

- Scharakteryzujesz podstawowe rodzaje uchwytów elementów składowych witryny.
- Omówisz sposoby modyfikowania wartości atrybutów HTML obiektów na stronie internetowej oraz ich stylów CSS.
- Sklasyfikujesz metody zmiany zawartości wewnętrznej elementów.

# Przeczytaj

W internecie **dane witryn przesyłane są w formie tekstowej**. Logika działania strony internetowej nie jest realizowana w postaci skompilowanego programu wykonywalnego, np. z rozszerzeniem `.exe`, lecz przyjmuje formę **niekompilowanych skryptów tekstowych**.

Tekst pozostaje **niezależny** od systemu operacyjnego czy środowiska uruchomieniowego, a ponadto nie zawiera kodów sterujących, czyli znaków mogących wpłynąć na logikę działania aplikacji.

To właśnie dlatego odpowiedź na wysłane do serwera żądanie HTTP odsyłana jest do klienta w postaci tekstowych plików HTML, CSS oraz skryptów JavaScript.

> **Ciekawostka**
>
> Inaczej oczywiście zachowują się skrypty uruchamiane po stronie serwera – pliki te na zawsze pozostaną utajnione dla użytkownika serwisu, gdyż zwyczajnie nigdy nie trafią do komputera klienta. Tego typu skrypty również są tekstem, lecz zdefiniowanym i używanym tylko na potrzeby realizacji zadań serwerowych, np. komunikacji z bazą danych.

Dopiero w komputerze użytkownika końcowego odebrany tekst zostaje zamieniony na **widok witryny**. Interpretacji dokonuje **przeglądarka internetowa**. Aplikacja ta oczywiście uruchomiona jest już w ramach konkretnego systemu operacyjnego (Windows, Linux, Unix, macOS).

## Obiektowy model dokumentu

Przeglądarka internetowa **na podstawie przysłanego do niej z serwera tekstu** tworzy w pamięci RAM komputera klienta liczne **obiekty**. Powstają one według przepisu **klas**, posiadających **atrybuty** (właściwości, cechy) oraz **metody**, czyli funkcje wewnątrz klas.

> **Ważne!**
>
> Metody w języku JavaScript to funkcje, które przetwarzają dany obiekt (są do niego przypisane jako własność). Zestaw metod jest definiowany dla konkretnych typów obiektów.
>
> Stworzenie obiektu przechowującego datę:
>
> - `const d = new Date();`
>
> Przykładowe metody wywołane na rzecz obiektu:
>
> - `d.getDate();` – zwraca dzień miesiąca,

> - `d.getHours();` – zwraca godzinę.
> - `d.setFullYear(2022);` – ustawia rok w dacie na wartość podaną jako argument w nawiasie.

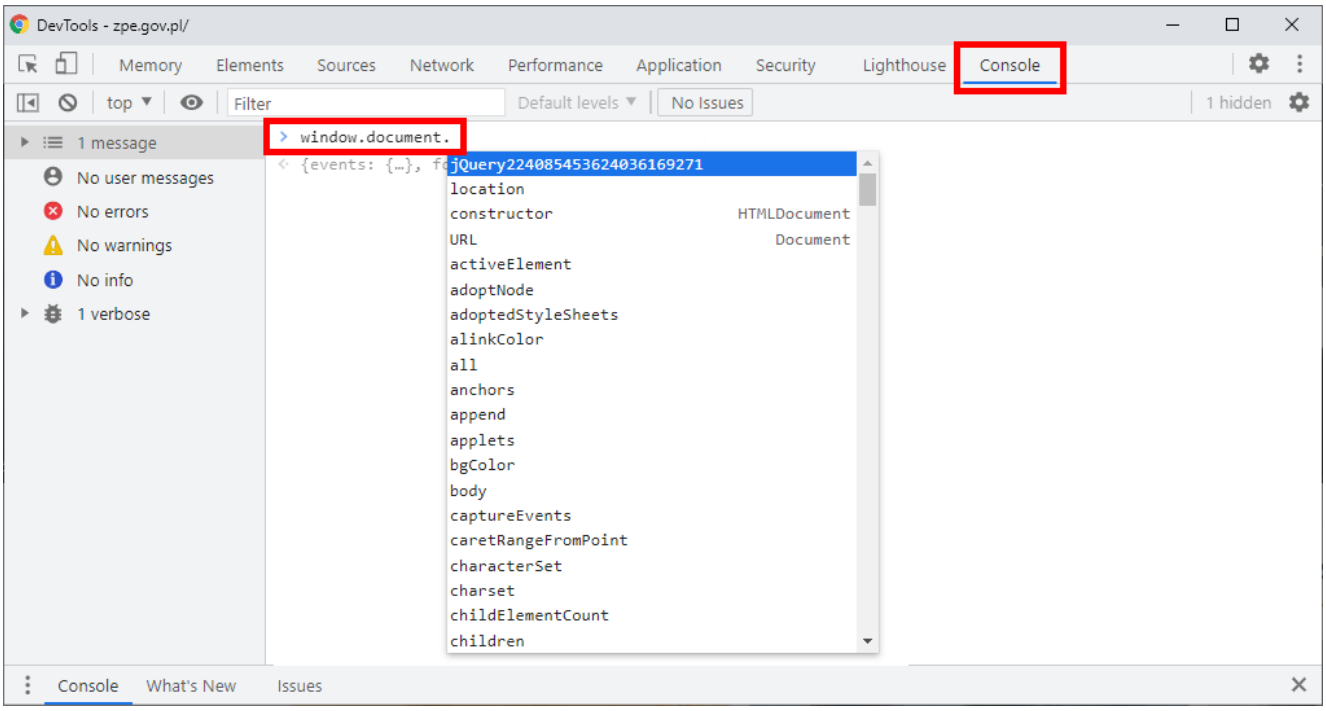
W pamięci komputera stworzony zostaje wirtualny **model struktury** strony internetowej. Zawiera on wszystkie zdefiniowane w dokumencie HTML **elementy** (obiekty), ich **style** (wygląd i położenie) określone w CSS oraz **zachowania** (funkcje, metody) zapisane w JavaScript , przydatne do realizowania mechaniki zachowania interfejsu po stronie klienta.

Model ten (ang. Document Object Model) umożliwia wprawnemu programiście JavaScript modyfikację wielu aspektów wyglądu oraz zachowania istniejących w dokumencie obiektów składowych. Można więc śmiało powiedzieć, że wraz z językiem JavaScript obiektowy model DOM tworzy API dedykowane wygodnej komunikacji programisty webowego z witryną.

## Przeglądanie struktury modelu

Aby przyjrzeć się strukturze obiektowej modelu DOM, wystarczy uruchomić **narzędzia deweloperskie** – np. podczas przeglądania dowolnej strony w przeglądarce Google Chrome wystarczy nacisnąć klawisz `F12` lub z menu kontekstowego myszy wybrać opcję `Zbadaj`.

W nowym oknie wybieramy zakładkę `Console`, po czym wpisujemy z klawiatury: `window.document.` (zwróćmy uwagę na dopisany na końcu operator kropki, który oddaje w JavaScript hierarchię elementów). Na liście podpowiedzi dostępnej po wpisaniu kropki (widocznej na zrzucie ekranu poniżej) warto rozejrzeć się wśród udostępnionych właściwości, kolekcji i metod, każdorazowo zatwierdzając swój wybór klawiszem `Enter`.

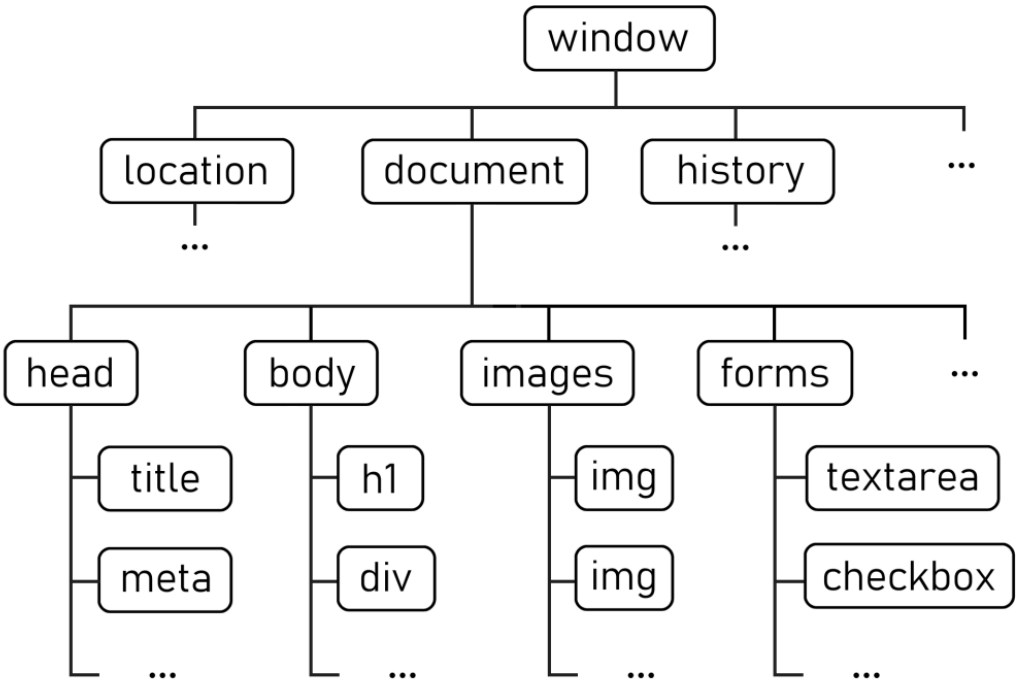


Źródło: Contentplus.pl sp. z o.o., licencja: CC BY-SA 3.0.

Warto zajrzeć np. do następujących elementów w strukturze `window.document.`:

- `window.document.head` – zwraca zawartość sekcji `head` dokumentu,
- `window.document.body` – zwraca zawartość sekcji `body` dokumentu,
- `window.document.images` – kolekcja wszystkich obrazów umieszczonych na podstronie,
- `window.document.links` – kolekcja hiperłączy w dokumencie, w tym adresów określonych w atrybucie `href` oraz pól `area`,
- `window.document.forms` – kolekcja zawierająca listę formularzy.

Szybko przekonujemy się, że struktura modelu **przypomina drzewo**. Jest ono bardzo rozbudowane, dlatego przedstawiamy jedynie kilka węzłów w postaci graficznej:




Źródło: Contentplus.pl Sp. z o.o., licencja: CC BY-SA 3.0.

Po tej wstępnej eksploracji modelu DOM nauczymy się **modyfikacji** elementów witryny z użyciem JavaScript. Każda taka zmiana zostanie poprzedzona **uchwyceniem** danego obiektu znajdującego się w dokumencie.

## Uchwycenie elementów witryny

Uchwyt (ang. *handle*) pozwala nam uzyskać łatwy **dostęp do obiektu** zdefiniowanego w HTML, a więc dostępnego także w modelu obiektowym DOM. Dzięki istnieniu uchwytu zyskamy możliwość **odczytu bądź modyfikacji licznych atrybutów** obiektu.

W języku JavaScript istnieje kilka sposobów definiowania uchwytów. Warto je poznać i przećwiczyć ich stosowanie, gdyż w codziennej pracy z kodem na pewno będziemy ich używać.

> **Polecenie 1**
>
> Wszystkie przedstawione poniżej przykłady uchwycenia obiektów możesz przetestować we własnej przeglądarce internetowej – wystarczy pobrać archiwum:

Plik o rozmiarze 1.38 KB w języku polskim

W pliku o nazwie `uchwyty.html` zdefiniowano łącznie sześć pojemników `<div>` o następujących wartościach atrybutów:

```
<div id="a" class="czerwony" name="x">Jeden</div>
```

```
<div id="b" class="zielony" name="x">Dwa</div>
<div id="c" class="niebieski" name="x">Trzy</div>
<div id="d" class="czerwony" name="y">Cztery</div>
<div id="e" class="zielony" name="y">Pięć</div>
<div id="f" class="niebieski" name="y">Sześć</div>
```

## Metoda `getElementById()`

Metoda `getElementById()`, jak wskazuje jej nazwa, pozwala uchwycić obiekt, jeżeli w znaczniku HTML ustanowiono dodatkowo atrybut `id`, koniecznie o **unikalnej** wartości. Zapis w postaci:

```
document.getElementById("a").innerHTML
```

pozwala uchwycić tylko pierwszy z pojemników, gdyż jako jedyny posiada atrybut `id` o wartości `a`.

Dodatkowo odczytano także atrybut `innerHTML` uchwyconego pierwszego bloku, czyli w praktyce znajdujący się w nim wewnętrzny kod HTML.

## Metoda `getElementsByName()`

Tym razem zyskujemy możliwość uchwycenia **wszystkich obiektów** w dokumencie, które mają taką samą wartość atrybutu `name`. A ponieważ atrybut ten **nie musi być unikalny** w dokumencie, zmienna `k` może stać się **kolekcją wielu uchwytów**:

```
var k = document.getElementsByName("x");
```

W omawianym ćwiczeniu istnieją dokładnie trzy pojemniki o wartości `"x"` atrybutu `name`.

Dostęp do poszczególnych uchwyconych obiektów zgromadzonych w kolekcji uzyskamy podobnie jak do **szuflad klasycznej tablicy**, stąd:

- pierwszy pojemnik to `k[0]`,
- drugi pojemnik to `k[1]`,
- trzeci pojemnik to `k[2]` - itd.

Jest to standardowe indeksowanie tablicowe, rozpoczynające się zawsze od wartości zero. Jak widać także w kodzie ćwiczenia, możemy dodatkowo odczytać **liczbę uchwyconych elementów**, a to za sprawą dodatkowego atrybutu `length`, który oznacza dosłownie **długość kolekcji**:

```
wypisz += "<li> Uchwycono " + k.length + " obiekty: ";
wypisz += k[0].innerHTML + ", " + k[1].innerHTML + ", " + k[2].
```

## Metoda `querySelector()`

Bardzo często elementy zdefiniowane w języku HTML zdarza nam się wybierać z całego dokumentu na potrzeby **arkuszy stylów** – są to tzw. **selektory CSS**. Stąd także w języku JavaScript powstała metoda używająca właśnie tych zapisów.

W przykładzie użyto operatora #, który oznacza w CSS wybranie z dokumentu obiektu o zadanej wartości `id`.

```
document.querySelector("#d").innerHTML
```

Ponieważ atrybut `id` **musi być unikalny**, nie mamy do czynienia z kolekcją, tylko z pojedynczym obiektem.

> **Ciekawostka**
>
> Zwróćmy uwagę, że w metodzie `querySelector()` możemy posługiwać się nie tylko znakiem # reprezentującym w selektorze CSS identyfikator elementu, lecz całym silnikiem selektorów CSS - możliwe jest zatem uchwycenie znaczników, klas, pseudoklas itd.

## Metoda `getElementsByTagName()`

Tym razem zyskujemy możliwość uchwycenia **wszystkich obiektów** w dokumencie, które zdefiniowano za pomocą **tego samego znacznika** (tagu) HTML.

Częstym błędem jest mylenie tej metody z `getElementsByName()`, która wykorzystuje atrybut `name`. Warto zwracać na to szczególną uwagę – mowa tu o `TagName`, czyli nazwie tagu, nie zaś o atrybucie `name`.

Podobnie jak w poprzednim przykładzie, tutaj także mamy do czynienia z kolekcją elementów, przez co również dysponujemy dodatkowym atrybutem `length` całej kolekcji o nazwie `zestaw`:

```
var zestaw = document.getElementsByTagName("div");
wypisz += "<li> Uchwycono " + zestaw.length + " obiektów - ";
wypisz += "pierwszy z nich w kolekcji: " + zestaw[0].innerHTML
```

### Metoda `getElementsByClassName()`

Obiekty możemy także uchwycić po wartości atrybutu `class`, który oczywiście **nie musi być unikalny** w dokumencie – mamy zatem ponownie do czynienia z kolekcją uchwytów oraz możemy używać zapisu tablicowego i nawiasów kwadratowych.

Dodatkowy atrybut `length` określający liczbę uchwyconych obiektów również jest dla kolekcji dostępny.

W naszym przykładzie oba pojemniki mają przypisaną klasę o nazwie `zielony`, a zatem oba obiekty zostają przekazane do kolekcji uchwytów:

```
var kolekcja = document.getElementsByClassName("zielony");
wypisz += "<li> Uchwycono " + kolekcja.length + " obiekty: ";
wypisz += kolekcja[0].innerHTML + ", " + kolekcja[1].innerHTML
```

> **Ciekawostka**
>
> Zwróćmy uwagę, że metody mogące zwrócić wiele uchwytów do obiektów, czyli będące kolekcjami, mają w nazwie na końcu literę `s` – jest to zawsze: `getElements`, w przeciwieństwie do `getElement`. W języku angielskim obecność tej dodatkowej litery `s` sugeruje właśnie liczbę mnogą.

### Metoda `querySelectorAll()`

Metoda ta, wzbogacona w porównaniu do `querySelector()` o słówko `All` (ang. wszystko), sprawdzi się w przypadku uchwycenia więcej niż jednego elementu z użyciem selektorów CSS. Służy zatem również do definiowania kolekcji.

Zgodnie z zasadą znaną ze wcześniejszych metod, także i tutaj dysponujemy dodatkowym atrybutem `length` utworzonej kolekcji:

```
var obiekty = document.querySelectorAll(".niebieski");
wypisz += "<li> Uchwycono " + obiekty.length + " obiekty: ";
wypisz += obiekty[0].innerHTML + ", " + obiekty[1].innerHTML +
```

Skoro sposoby definiowania uchwytów mamy już sklasyfikowane, możemy przejść do ćwiczeń praktycznych w kolejnej sekcji tego e-materiału.

## Jak modyfikować style CSS?

Sklasyfikujmy metody wpływania przy użyciu języka JavaScript na style CSS uchwyconych obiektów.

> **Polecenie 2**
>
> Wszystkie przedstawione przykłady wpływania na style elementów hierarchii DOM możesz przetestować we własnej przeglądarce internetowej – wystarczy pobrać archiwum.

Plik o rozmiarze 2.67 KB w języku polskim

## Metoda `classList.add()`

Kod źródłowy znajdziesz w archiwum w pliku o nazwie `c01.html`:

```
<head>
  <style>.czerwony { color: red; }</style>
</head>

<body>

  <div id="a" class="klasa" name="x">Hello</div>
  <div id="b" class="klasa" name="y">World</div>

  <script>
    var obiekt = document.getElementById("a");
    obiekt.classList.add("czerwony");
  </script>
```

Za pomocą tej instrukcji jesteśmy w stanie **dodać klasę** o zadanej nazwie **do listy wszystkich klas** obowiązujących w uchwyconym obiekcie. W naszym przykładzie do pierwszego pojemnika dodano klasę zmieniającą kolor tekstu na czerwony.

## Metoda `classList.remove()`

Kod źródłowy znajdziesz w archiwum w pliku o nazwie `c02.html`:

```
<head>
  <style>.czerwony { color: red; }</style>
</head>

```

```
<body>

  <div id="a" class="czerwony" name="x">Hello</div>
  <div id="b" class="klasa" name="y">World</div>

  <script>
    var obiekt = document.getElementById("a");
    obiekt.classList.remove("czerwony");
  </script>
```

Tym razem możemy **usunąć klasę** z listy aktualnie obowiązujących w obiekcie. W przedstawionym przykładzie klasa o nazwie `class="czerwony"` obowiązywała w HTML, lecz skrypt usunął ją z obiektu, przez co tekst pozostał zapisany czarnym kolorem czcionki.

## Metoda `classList.contains()`

Kod źródłowy znajdziesz w archiwum w pliku o nazwie `c03.html`:

```
<head>
  <style>.czerwony { color: red; }</style>
</head>

<body>

  <div id="a" class="czerwony" name="x">Hello</div>
  <div id="b" class="klasa" name="y">World</div>

  <script>
    var obiekt = document.getElementById("a");
    //Odkomentowanie linii poniżej sprawi, że spełni się else:
    //obiekt.classList.remove("czerwony");
    if (obiekt.classList.contains("czerwony"))
      alert("Posiada klasę: czerwony");
    else
      alert("Nie posiada klasy: czerwony");
  </script>
```

Angielskie słowo *contains* oznacza „zawiera” – stąd metoda ta służy programiście do uzyskania odpowiedzi na pytanie, czy obiekt posiada aktualnie przypisaną pewną klasę.

Metoda zwraca wartość `true` lub `false`, co pozwala na wygodne skorzystanie z niej w instrukcji warunkowej `if`. W omawianym przykładzie sprawdzamy, czy obiekt ma przypisaną klasę o nazwie `czerwony`.

## Metoda `classList.toggle()`

Kod źródłowy znajdziesz w archiwum w pliku o nazwie `c04.html`:

```
<head>
  <style>.czerwony { color: red; }</style>
</head>

<body>

  <div id="a" class="klasa" name="x">Hello</div>
  <div id="b" class="klasa" name="y">World</div>

  <script>
    var obiekt = document.getElementById("a");
    //Dodaj klasę: .czerwony
    obiekt.classList.toggle("czerwony");
    //Odkomentowanie linii poniżej usunie klasę:
    //obiekt.classList.toggle("czerwony");
  </script>
```

Metoda ta działa jak **przełącznik stanu przypisania klasy do obiektu**. Jeżeli klasy nie ma na liście w elemencie, to zostanie dodana. W przeciwnym wypadku, jeśli klasa aktualnie obowiązuje, nastąpi usunięcie jej z obiektu.

## Właściwość `style`

Kod źródłowy znajdziesz w archiwum w pliku o nazwie `c05.html`:

```
<div id="a" class="klasa" name="x">Hello</div>
<div id="b" class="klasa" name="y">World</div>

<script>
  var obiekt = document.getElementById("a");
  obiekt.style.color = "red";
  obiekt.style.fontWeight = "bold";
```

```
    obiekt.style.marginBottom = "30px";
</script>
```

Dzięki właściwości `style` zyskujemy możliwość prostej **modyfikacji konkretnych właściwości** CSS z użyciem języka JavaScript. Oczywiście problemem może być operator `-`, który w CSS **rozdziela właściwości główne od potomnych** w kaskadzie, zaś w JavaScript jest **operatorem odejmowania**.

Kilka przykładów przedstawionych w tabeli pokazuje, w jaki sposób poradzono sobie z tym problemem:

| Właściwości | JavaScript |
|---|---|
| background-color | backgroundColor |
| font-family | fontFamily |
| list-style-type | listStyleType |
| text-decoration | textDecoration |

Jak widać, zasada jest prosta – tam, gdzie jest to wymagane, **brak minusa zaznaczono przez użycie wielkiej litery**. Ten sam sposób formułowania nazw właściwości zastosowano w omawianym przykładzie.

## Właściwość `style.cssText`

Kod źródłowy znajdziesz w archiwum w pliku o nazwie `c06.html`:

```
<div id="a" style="color: red; font-weight: bold;">Hello</div>
<div id="b" class="klasa" >World</div>

<script>
 //Odczytaj style CSS elementu
 var obiekt = document.getElementById("a");
 alert(obiekt.style.cssText);
 //Zmień style CSS elementu
 //Odkomentowanie linii poniżej zmieni wygląd tekstu:
 //obiekt.style.cssText = "color: green;";
</script>
```

Ta metoda **odczytuje w formie tekstowej** deklaracje ustawionych w elemencie stylów, ale może je także **modyfikować**, jeśli znajdzie się po lewej stronie operatora: = (jak

w prezentowanym przykładzie).

## Jak zmienić zawartość wewnętrzną?

Modyfikacja modelu DOM może także polegać na **zmianie tekstu znajdującego się wewnątrz** elementu (np. bloku czy akapitu) lub nawet na **redefinicji znaczników** HTML wewnątrz niego.

> **Polecenie 3**
>
> Wszystkie przedstawione przykłady wpływania na zawartość wewnętrzną obiektów możesz przetestować we własnej przeglądarce internetowej – wystarczy pobrać znajdujące się tu archiwum:

Plik o rozmiarze 1.83 KB w języku polskim

### Właściwość `innerText`

Kod źródłowy znajdziesz w archiwum w pliku o nazwie `z01.html`:

```
<div id="a">
  <p id="b">Hello</p>
  <p id="c">World</p>
</div>

<script>
  var obiekt = document.getElementById("a");
  //Odczyt właściwości
  alert(obiekt.innerText);
  //Przykładowa modyfikacja (ustawienie) właściwości
  obiekt.innerText = "<p>Podmieniono!</p>";
</script>
```

Ten atrybut pozwala **odczytać lub ustawić tekstową zawartość** elementu. Tekstową – czyli pozbawioną znaczników. Zwróćmy uwagę, że przy **odczycie** akapity traktowane są jako znaki końca linii, a nie jako tagi.

Podobnie przy **modyfikacji** wartości, pomimo wstawienia tagów `<p>Podmieniono!</p>` przeglądarka traktuje nawiasy ostre jako encje w tekście, a nie znaczniki HTML.

### Właściwość `innerHTML`

Kod źródłowy znajdziesz w archiwum w pliku o nazwie `z02.html`:

```
<div id="a">
  <p id="b">Hello</p>
  <p id="c">World</p>
</div>

<script>
  var obiekt = document.getElementById("a");
  //Odczyt właściwości
  alert(obiekt.innerHTML);
  //Przykładowa modyfikacja (ustawienie) właściwości
  obiekt.innerHTML = "<p>Witaj</p><p>Świecie</p>";
</script>
```

Tym razem uzyskujemy dostęp do **wewnętrznej zawartości HTML**, czyli zawierającej także znaczniki. Podczas odczytu wnętrza pojemnika w oknie dialogowym pojawiają się tagi, tak też w przypadku modyfikacji możliwa staje się całkowita **redefinicja struktury znaczników** elementu.

## Właściwość `outerHTML`

Kod źródłowy znajdziesz w archiwum w pliku o nazwie `z03.html`:

```
<div id="a">
  <p id="b">Hello</p>
  <p id="c">World</p>
</div>

<script>
  var obiekt = document.getElementById("a");
  //Odczyt właściwości
  alert(obiekt.outerHTML);
  //Przykładowa modyfikacja (ustawienie) właściwości
  obiekt.outerHTML = "<p>Witaj</p><p>Świecie</p>";
</script>
```

Jest to również sposób na dostanie się do **zawartości HTML elementu**, czyli zawierającej zdefiniowane znaczniki. Tym razem jednak **odczyt zwraca także sam element**, a więc w naszym przykładzie `<div id="a">...</div>`.

Warto zwrócić uwagę, że przeprowadzona tu zmiana zawartości zewnętrznej bloku na treść **pozbawioną tego pojemnika** w praktyce **wymazuje ten obiekt** ze zmodyfikowanego kodu HTML.

### Właściwość `textContent`

Kod źródłowy znajdziesz w archiwum w pliku o nazwie `z04.html`:

```
<div id="a">
  <p id="b">Hello</p>
  <p id="c">World</p>
</div>

<script>
  var obiekt = document.getElementById("a");
  //Odczyt właściwości
  alert(obiekt.textContent);
  //Przykładowa modyfikacja (ustawienie) właściwości
  obiekt.textContent = "<p>Witaj Świecie</p>";
</script>
```

Jest to atrybut wprowadzony wcześniej w standardy od właściwości `innerText`, lecz podobnie z jego pomocą można **odczytać lub ustawić tekstową zawartość** elementu pozbawioną znaczników.

Istnieje kilka różnic pomiędzy `innerText` a `textContent`, najważniejsza z nich jest taka, że właściwość `innerText` **zwróci tylko widoczną** zawartość tekstową, a `textContent` **pełny tekst**, w tym np. elementy tekstowe ukryte w pojemniku z użyciem właściwości `display:none;` w arkuszu stylów CSS.

## Słownik

> **API**
>
> (ang. *Application Programming Interface*) zestaw udostępnionych klas, metod, funkcji, zmiennych, parametrów, których aplikacja używa w celu zrealizowania zaplanowanych przez programistę zadań
> **encje HTML**
>
> kombinacje tekstowe znaków, używane w celu zastąpienia znaku sterującego w tekście krótką formą tekstową; przykładem może być zastąpienie znaku `<` encją `<` – dzięki

temu przeglądarka zrozumie, że naszą intencją jest wstawienie otwierającego nawiasu ostrego w tekście witryny, a nie rozpoczynanie w tym miejscu nowego znacznika HTML

**kody sterujące**

znaki w danej tablicy kodowej, których celem jest dokonanie jakieś zmiany w sterowaniu urządzeniem przetwarzającym dane; przykładami znaków sterujących w popularnej tablicy ASCII mogą być kody: `Escape`, `Carriage Return` czy `Line Feed`

**kompilacja**

proces tłumaczenia kodu źródłowego zapisanego w danym języku programowania na inny język, najczęściej język maszynowy; dzięki kompilacji na język maszynowy (przy udziale kilku innych programów wspomagających) powstaje plik wykonywalny, czyli aplikacja możliwa do uruchomienia w ramach danego systemu operacyjnego

**metoda**

funkcja należąca do klasy i służąca do operowania na jej obiektach (zmiennych)

# Animacja

Teraz, gdy już rozumiemy, czym jest obiektowy model DOM, oraz gdy dowiedzieliśmy się, w jaki sposób w języku JavaScript:

- uchwycić obiekty,
- zmodyfikować zawartość wewnętrzną obiektów,
- wpłynąć na style CSS elementów,

– pora nauczyć się odczytu i modyfikacji **atrybutów** obiektów DOM.

**Polecenie 1**

Zapoznaj się z filmem wyjaśniającym przeznaczenie podanych metod. Wszystkie przedstawione w nim przykłady wpływania na atrybuty obiektów możesz przetestować w przeglądarce internetowej – w tym celu wystarczy pobrać archiwum.

Plik o rozmiarze 2.21 KB w języku polskim

**Metoda** `getAttribute()` - pobiera wartość danego atrybutu lub zwraca null, jeżeli taki atrybut nie istnieje.

Kod źródłowy znajdziesz w archiwum w pliku o nazwie `a01.html`.

**Metoda** `setAttribute()` - ustawia nową, pożądaną wartość wskazanego atrybutu.

Kod źródłowy znajdziesz w archiwum w pliku o nazwie `a02.html`.

**Metoda** `removeAttribute()` - usuwa wskazany nazwą atrybut z obiektu.

Kod źródłowy znajdziesz w archiwum w pliku o nazwie `a03.html`.

**Metoda** `hasAttribute()` - sprawdza, czy obiekt posiada wskazany atrybut - zwraca `true`, jeśli istnieje, lub `false` - w przeciwnym wypadku.

Kod źródłowy znajdziesz w archiwum w pliku o nazwie `a04.html`.

**Metoda** `toggleAttribute()` - przełącza stan atrybutu - jeśli istnieje, usuwamy go z obiektu, jeśli nie istnieje, dodajemy go.

Kod źródłowy znajdziesz w archiwum w pliku o nazwie `a05.html`.


Film dostępny pod adresem /preview/resource/R9ladxDi7H84o



Film nawiązujący do treści materiału: Jak modyfikować atrybuty html.

---

Plik o rozmiarze 257.00 B w języku polskim

Plik o rozmiarze 336.00 B w języku polskim

Plik o rozmiarze 372.00 B w języku polskim

Plik o rozmiarze 398.00 B w języku polskim

Plik o rozmiarze 283.00 B w języku polskim

> **Polecenie 2**
>
> Zapoznaj się z zawartością archiwum i przetestuj działanie metod.

# Sprawdź się

Pokaż ćwiczenia:

## Ćwiczenie 1

Wstaw w odpowiednie miejsca nazwy kilku podstawowych kolekcji modelu DOM.

`window.document.` ____ - zawartość sekcji nagłówkowej dokumentu

`window.document.` ____ - kolekcja obrazów na stronie

`window.document.` ____ - kolekcja hiperłączy i pól `area`

`window.document.` ____ - zawartość ciała dokumentu

`window.document.` ____ - lista formularzy i ich kontrolek

history | links | forms | location | body | head | images

## Ćwiczenie 2

Połącz każdą nazwę metody definiowania uchwytów obiektów w JavaScript z właściwym opisem jej działania.

| | |
|---|---|
| `getElementsByClassName()` | może uchwycić tylko jeden obiekt w dokumencie |
| `querySelector()` | uchwyci wszystkie obiekty mające przypisaną taką samą klasę |
| `getElementsByTagName()` | kolekcja uchwytów do elementów zdefiniowanych takim samym znacznikiem |
| `getElementById()` | do wskazania obiektu używa selektora CSS |
| `getElementsByName()` | kolekcja uchwytów do elementów o takiej samej wartości znacznika `name` |

## Ćwiczenie 3

Wskaż, której metody w języku JavaScript użyjemy do sprawdzenia, czy atrybut `disabled` istnieje aktualnie w uchwyconym obiekcie.

- ◯ `obiekt.usesAttribute("disabled")`
- ◯ `obiekt.containsAttribute("disabled")`
- ◯ `obiekt.toggleAttribute("disabled")`
- ◯ `obiekt.hasAttribute("disabled")`

## Ćwiczenie 4

Uzupełnij tekst brakującymi fragmentami, korzystając z podanych poniżej opcji.

Przeglądarka internetowa na podstawie [ ] przysłanego do niej z internetu tworzy w pamięci RAM komputera klienta liczne [ ] powstające według [ ] klas, które zawierają [ ] (właściwości, cechy) oraz [ ], czyli funkcje wewnątrz klas.

Tak stworzony zostaje wirtualny model [ ] strony internetowej, który określamy w języku angielskim jako Document [ ] Model. Można powiedzieć, że prawdziwa jest zależność:

Model DOM + JavaScript = [ ], czyli interfejs dedykowany do wygodnego realizowania komunikacji programisty webowego z witryną.

przepisu | metody | WWW | programu komputerowego | struktury | API | tekstu | atrybuty | Object | obiekty

## Ćwiczenie 5

Dokończ zdanie.
**Pole edycyjne typu `checkbox` posiadające atrybut `id="x"` jest aktualnie zaznaczone.**
**Wywołanie metody:**

```
document.getElementById("x").toggleAttribute("checked");
```

**spowoduje w przeglądarce...**

- ◯ wyłączenie aktywności pola.
- ◯ ukrycie widoczności pola.
- ◯ ponowne zaznaczenie pola, czyli utrzymanie stanu zaznaczenia.
- ◯ odznaczenie pola.

## Ćwiczenie 6

Przyporządkuj do odpowiednich grup podane metody i właściwości z języka JavaScript, zgodnie z ich właściwym zastosowaniem.

modyfikacja wartości atrybutów HTML

uchwycenie obiektów

modyfikacja zawartości wewnętrznej

outerHTML | innerText | getElementById() | classList.add() | toggleAttribute() | querySelectorAll() | setAttribute() | textContent

## Ćwiczenie 7

W strukturze dokumentu HTML określono akapit tekstu, który posiada atrybut `id="a"`, zaś w swoim wnętrzu przechowuje tylko napis: "`Hello`". Na rzecz tego obiektu wywołano metodę:

```
alert(document.getElementById("a").outerHTML);
```

Wskaż, co zostanie zwrócone w oknie dialogowym jako rezultat jej wykonania.

- ◯ `Hello`
- ◯ `<body><p id="a">Hello</p></body>`
- ◯ `null`
- ◯ `<p id="a">Hello</p>`

## Ćwiczenie 8

Przyporządkuj do odpowiedniej grupy elementy wpływające na styl obiektu, znane z języków JavaScript oraz CSS.

atrybut JavaScript

właściwość CSS

marginBottom | font-family | text-decoration | backgroundColor | list-style-type | fontWeight

# Dla nauczyciela

**Autor:** Mirosław Zelent
**Przedmiot:** Informatyka

**Temat: Modyfikacja struktury strony w języku JavaScript**

**Grupa docelowa:**

Szkoła ponadpodstawowa, liceum ogólnokształcące, technikum, zakres podstawowy

**Podstawa programowa:**

Cele kształcenia – wymagania ogólne

> II. Programowanie i rozwiązywanie problemów z wykorzystaniem komputera oraz innych urządzeń cyfrowych: układanie i programowanie algorytmów, organizowanie, wyszukiwanie i udostępnianie informacji, posługiwanie się aplikacjami komputerowymi.

Treści nauczania – wymagania szczegółowe

> II. Programowanie i rozwiązywanie problemów z wykorzystaniem komputera i innych urządzeń cyfrowych.
>
> Zakres podstawowy. Uczeń:
>
> 3) przygotowuje opracowania rozwiązań problemów, posługując się wybranymi aplikacjami:
>
> f) tworzy stronę internetową zgodnie ze standardami, wzbogaconą tabelami, listami, elementami dynamicznymi, posługuje się arkuszem stylów, korzysta z oprogramowania i serwisów przeznaczonych do tworzenia stron; potrafi opublikować własną stronę w internecie;

**Kształtowane kompetencje kluczowe:**

- kompetencje cyfrowe;
- kompetencje osobiste, społeczne i w zakresie umiejętności uczenia się;
- kompetencje matematyczne oraz kompetencje w zakresie nauk przyrodniczych, technologii i inżynierii.

**Cele operacyjne (językiem ucznia):**

- Scharakteryzujesz podstawowe rodzaje uchwytów elementów składowych witryny.

- Omówisz sposoby modyfikowania wartości atrybutów HTML obiektów na stronie internetowej oraz ich stylów CSS.
- Sklasyfikujesz metody zmiany zawartości wewnętrznej elementów.

**Strategie nauczania:**

- konstruktywizm;
- konektywizm.

**Metody i techniki nauczania:**

- dyskusja;
- rozmowa nauczająca z wykorzystaniem multimedium i ćwiczeń interaktywnych;
- ćwiczenia praktyczne.

**Formy pracy:**

- praca indywidualna;
- praca w parach;
- praca w grupach;
- praca całego zespołu klasowego.

**Środki dydaktyczne:**

- komputery z głośnikami, słuchawkami i dostępem do internetu;
- zasoby multimedialne zawarte w e-materiale;
- tablica interaktywna/tablica, pisak/kreda.

**Przebieg lekcji**

**Przed lekcją:**

1. **Przygotowanie do zajęć.** Nauczyciel loguje się na platformie i udostępnia e-materiał: „Modyfikacja struktury strony w języku JavaScript". Nauczyciel prosi uczniów o zapoznanie się z treściami w sekcji „Przeczytaj".

**Faza wstępna:**

1. Nauczyciel wyświetla temat i cele zajęć. Prosi uczniów, by na podstawie wiadomości zdobytych przed lekcją zaproponowali kryteria sukcesu.

**Faza realizacyjna:**

1. **Praca z multimedium.** Nauczyciel wyświetla zawartość sekcji „Animacja", czyta treść polecenia nr 1 „" i omawia kolejne kroki rozwiązania.
2. **Ćwiczenie umiejętności.** Nauczyciel przechodzi do sekcji „Sprawdź się". Zapowiada uczniom, że w kolejnym kroku będą rozwiązywać ćwiczenia nr 1-8 – od najprostszych

do najtrudniejszych – i będą to robić wspólnie. Wybrany uczestnik zajęć czyta po kolei polecenia. Po każdym przeczytanym poleceniu ochotnik udziela odpowiedzi. Reszta uczniów ustosunkowuje się do niej, proponując swoje pomysły. Nauczyciel w razie potrzeby koryguje odpowiedzi, dopowiada istotne informacje, udziela uczniom informacji zwrotnej.

**Faza podsumowująca:**

1. Nauczyciel ponownie wyświetla na tablicy temat lekcji zawarty w sekcji „Wprowadzenie” i inicjuje krótką rozmowę na temat zrealizowanych celów (czego uczniowie się nauczyli).

**Praca domowa:**

1. Uczniowie wykonują ćwiczenia 1 i 2 z sekcji „Animacja”.

**Wskazówki metodyczne:**

- Uczniowie mogą wykorzystać multimedium w sekcji „Animacja” do przygotowania się do lekcji powtórkowej.