Zaawansowany Programista JS

Dzień 2

Plan

- Podstawy HTML
  - Podstawowe tagi HTML
  - Szkielet HTML
  - Podlinkowanie styli i JavaScript
  - Formularze
  - Tabele
- Podstawy JS
  - Deklarowanie zmiennych
  - Typy zmiennych (proste i złozone)
  - Wbudowane metody JavaScript do pracy ze zmiennymi
  - Instrukcje warunkowe
  - Pętle
  - Funkcje
- Podstawy DOM
  - Selectory CSS
  - Łapanie elementów za pomocą `querySelector` i `querySelectorAll`
  - Podstawowe metody DOM na elementach
  - Eventy
  - Praca z formularzami

Podstawy HTML

HTML (Hypertext Markup Language) to język znaczników wykorzystywany do tworzenia stron internetowych. Jest to podstawowy język używany do opisu struktury i zawartości dokumentów na stronach internetowych.

HTML składa się z zestawu znaczników, które opisują różne elementy strony internetowej, takie jak nagłówki, paragrafy, obrazy, linki i wiele innych. Znaczniki te są interpretowane przez przeglądarkę internetową i wykorzystywane do wyświetlania treści na stronie.

Podstawowe tagi HTML

Kod HTML składa się z elementów, które składają się z otwierającego tagu, treści elementu i zamykającego tagu. Na przykład, element nagłówka pierwszego poziomu w HTML wygląda tak:

`<h1>To jest nagłówek pierwszego poziomu</h1>`

W HTML znaczniki zamykające są opcjonalne dla niektórych elementów, na przykład dla znacznika obrazu `<img src="obrazek.jpg" alt="Opis obrazka">`

Nie ma potrzeby zamykania tego tagu, ponieważ nie ma w nim zawartości. Jednak dla większości innych elementów, takich jak nagłówki, paragrafy, listy itp., potrzebne są zarówno tagi otwierające, jak i zamykające. Na przykład, otwarcie i zamknięcie tagu p wygląda następująco: `<p>To jest paragraf z jakąś treścią.</p>` Tag zamykający zawsze ma ten sam format co tag otwierający, ale zawiera ukośnik / po nazwie elementu. Na przykład, tag zamykający dla tagu otwierającego p to `</p>`. To właśnie takie zamknięcie oznacza, że treść pomiędzy tagami p jest wewnątrz tego elementu.

Warto zauważyć, że w niektórych przypadkach tagi zamykające mogą być pominięte, na przykład w przypadku elementów pustych takich jak img lub input. Jednakże, dla większości elementów HTML zaleca się zawsze używanie obu tagów - otwierającego i zamykającego.

Szkielet HTML

Podstawowy szkielet HTML to minimalna struktura pliku HTML, która jest wymagana, aby przeglądarka mogła poprawnie wyświetlić stronę internetową. Oto przykład podstawowego szkieletu HTML:

```
<!DOCTYPE html>
<html>
<head>
  <title>Tytuł strony</title>
</head>
<body>
  Treść strony
</body>
</html>
```

Pierwsza linia `<!DOCTYPE html>` informuje przeglądarkę, że dokument jest napisany w języku HTML5. Następnie mamy znacznik `<html>`, który definiuje początek struktury dokumentu HTML. Wewnątrz tego tagu umieszczamy dwa kolejne tagi: `<head>` i `<body>`. Tag `<head>` zawiera informacje o dokumencie, takie jak tytuł strony, informacje meta i odwołania do zewnętrznych plików CSS i JavaScript. Tag `<body>` zawiera zawartość strony, taką jak tekst, obrazy, filmy i inne elementy.

W powyższym przykładzie w sekcji `<head>` znajduje się tag `<title>`, który definiuje tytuł strony wyświetlany w pasku tytułu przeglądarki. W sekcji `<body>` jest tylko jedna linia z tekstem "Treść strony", ale zwykle w sekcji tej znajduje się znacznie więcej elementów, takich jak nagłówki, paragrafy, listy, obrazy i inne.

To jest bardzo prosty szkielet HTML, ale na jego bazie można już zbudować pełnoprawną stronę internetową.

Podlinkowanie styli i JavaScript

Aby podlinkować pliki CSS i JS do podstawowego szkieletu HTML, należy użyć odpowiednich tagów w sekcji `<head>` dokumentu. Przykładowo, jeśli chcemy podlinkować plik CSS o nazwie style.css i plik JS o nazwie script.js, to w sekcji `<head>` musimy dodać następujące tagi:

```
<!DOCTYPE html>
<html>
<head>
  <title>Tytuł strony</title>
  <link rel="stylesheet" type="text/css" href="style.css">
  <script src="script.js"></script>
</head>
<body>
  Treść strony
</body>
</html>
```

Tag `<link>` służy do podlinkowania pliku CSS i zawiera atrybuty `rel="stylesheet"` i `type="text/css"`, które określają, że link dotyczy pliku stylów CSS. Atrybut href zawiera ścieżkę do pliku CSS względem pliku HTML.

Tag `<script>` służy do podlinkowania pliku JS i zawiera atrybut src, który określa ścieżkę do pliku JS względem pliku HTML.

Dzięki tym tagom przeglądarka wie, gdzie ma szukać plików CSS i JS, które mają zostać wykorzystane na stronie. W ten sposób możemy oddzielić stylizację i logikę strony od samej zawartości, co ułatwia zarządzanie i edycję naszego kodu.

Warto umieszczać znacznik `<script>` przed końcem sekcji `<body>` ze względu na kilka powodów:

- Szybsze ładowanie strony - umieszczając skrypty na końcu `<body>`, przeglądarka musi je pobrać i wykonać przed wczytaniem reszty strony. Jeśli skrypty są duże lub złożone, mogą opóźniać ładowanie strony, co wpłynie negatywnie na doświadczenie użytkownika. Dlatego lepiej umieścić skrypty przed końcem `<body>`, tak aby treść strony została wczytana przed ich pobraniem i wykonywaniem.
- Poprawne działanie skryptów - jeśli umieścimy skrypty w sekcji `<head>`, mogą pojawić się problemy z ich działaniem. Przeglądarka wykonuje skrypty w kolejności, w jakiej są zdefiniowane, więc jeśli skrypt odwołuje się do elementów strony, które jeszcze nie zostały utworzone (np. elementów w sekcji `<body>`), może pojawić się błąd. Umieszczając skrypty na końcu `<body>`, zapewniamy, że wszystkie elementy strony zostały już utworzone i można bezpiecznie wykonywać skrypty.
- Ułatwienie pracy z kodem - umieszczając skrypty na końcu `<body>`, ułatwiamy edycję kodu, ponieważ nie trzeba przełączać się między kodem HTML a JS. Skrypty są umieszczone w jednym miejscu, co ułatwia ich zarządzanie i utrzymanie.

Warto jednak pamiętać, że niektóre skrypty, takie jak np. narzędzia analityczne lub śledzące, muszą być umieszczone w sekcji `<head>`, aby działały poprawnie. W takim przypadku warto przeanalizować, które skrypty należy umieścić przed końcem `<body>`, a które w sekcji `<head>`.

Formularze

Formularz w HTML jest to element, który pozwala na zbieranie danych od użytkownika, takich jak tekst, liczby, wybory, daty itp. Formularze składają się z jednego lub więcej pól, w których użytkownik wprowadza informacje oraz z przycisków do ich przesłania. Głównym celem formularzy jest zebranie danych od użytkowników, które mogą być wykorzystane przez aplikację internetową lub wysłane na serwer.

Formularze składają się z kilku elementów HTML, w tym:

- `<form>` - element, który określa formularz i jego właściwości, takie jak metoda przesyłania danych, adres URL docelowego skryptu i inne atrybuty.
- Pole formularza - element, który pozwala na wprowadzenie danych przez użytkownika. Pole formularza może przyjmować różne typy, takie jak tekstowe `<input type="text">`, wyboru (`<input type="radio">` lub `<select>`), przycisku `<button>`, daty `<input type="date">` i wiele innych.
- `<label>` - element, który określa etykietę dla pola formularza. Etykiety te są używane do opisania, co należy wprowadzić w pole formularza.
- `<button>` - element, który umożliwia użytkownikowi wysłanie formularza lub wykonanie innej akcji związanej z formularzem, np. resetowanie formularza.

Oto prosty przykład formularza, który składa się z pola tekstowego i przycisku do przesłania danych:

```
<form>
  <label for="name">Imię:</label>
  <input type="text" id="name" name="name"><br><br>
  <button type="submit">Wyślij</button>
</form>
```

W tym przykładzie formularz składa się z jednego pola tekstowego `<input type="text">` i przycisku do przesłania danych `<button type="submit">`. Pole tekstowe jest oznaczone etykietą `<label>` o wartości "Imię:", która pomaga użytkownikowi zrozumieć, co należy wprowadzić do pola tekstowego. Pole tekstowe posiada atrybut id i name, które są używane do odwoływania się do jego wartości w kodzie JavaScript lub w przetwarzaniu danych po stronie serwera. Przycisk "Wyślij" powoduje przesłanie danych z formularza na adres URL określony w atrybucie action elementu `<form>`.

Tabele

Tabela w HTML to element, który pozwala na wyświetlanie danych w postaci tabelarycznej, składającej się z wierszy (`<tr>`) i kolumn (`<td>` lub `<th>`). Elementy `<td>` definiują poszczególne komórki w tabeli, natomiast elementy `<th>` definiują nagłówki kolumn lub wierszy.

Oto przykładowa tabela HTML z trzema wierszami i trzema kolumnami

```
<table>
  <tr>
    <th>Imię</th>
    <th>Nazwisko</th>
    <th>Wiek</th>
  </tr>
  <tr>
    <td>Adam</td>
    <td>Kowalski</td>
```

```
    <td>25</td>
  </tr>
  <tr>
    <td>Marta</td>
    <td>Nowak</td>
    <td>30</td>
  </tr>
</table>
```

W powyższym przykładzie pierwszy wiersz definiuje nagłówki kolumn, które są wyświetlane wytłuszczoną czcionką dzięki użyciu elementu `<th>`. Następne wiersze definiują komórki tabeli (`<td>`) z wartościami dla każdej kolumny. Warto zwrócić uwagę na zastosowanie elementu `<table>`, który definiuje całą tabelę, oraz elementów `<tr>`, które definiują kolejne wiersze.

Podstawy JS

JavaScript jest językiem skryptowym, co oznacza, że kod źródłowy jest kompilowany i wykonywany bezpośrednio w przeglądarce internetowej. Dlatego też, każdy użytkownik przeglądający stronę internetową może wykonać kod JavaScript na swoim urządzeniu bez potrzeby instalacji żadnych dodatkowych narzędzi.

JavaScript został stworzony przez Brendana Eicha w 1995 roku i początkowo służył jedynie do prostych walidacji formularzy na stronach internetowych. Od tamtej pory stał się niezwykle popularnym i wszechstronnym językiem programowania stosowanym na całym świecie.

Deklarowanie zmiennych

W JavaScript zmienne są deklarowane przy pomocy słowa kluczowego `var`, `let`, lub `const`. Oto kilka przykładów

```
var name = 'John'; // zmienna "name" typu string
let age = 30; // zmienna "age" typu number
const PI = 3.14; // stała "PI" typu number
```

Słowo kluczowe var było pierwotnie jedynym sposobem deklarowania zmiennych w JavaScript, ale wprowadzono również słowa kluczowe let i const, aby ulepszyć obsługę zmiennych.

Zmienne zadeklarowane przy pomocy var i let mogą zmieniać swoją wartość, natomiast zmienne zadeklarowane przy użyciu const są stałe i nie mogą zmienić swojej wartości.

```
let x = 5; // deklaracja zmiennej "x" i przypisanie wartości 5
x = 10; // zmiana wartości zmiennej "x" na 10

const y = 20; // deklaracja stałej "y" i przypisanie wartości 20
y = 30; // to spowoduje błąd, ponieważ stałe nie mogą zmieniać swojej wartości
```

Typy zmiennych (proste i złozone)

W JavaScript mamy kilka typów zmiennych, w tym:

- String - reprezentuje łańcuch znaków, np. 'Hello, World!'
- Number - reprezentuje liczby, np. 42
- Boolean - reprezentuje wartość true lub false
- Null - reprezentuje brak wartości, jest to specjalny typ, który zawiera tylko jedną wartość: null
- Undefined - reprezentuje zmienną, która nie ma wartości przypisanej, jest to również specjalny typ z jedną wartością: undefined
- Object - reprezentuje złożone struktury danych, np. obiekty, funkcje, tablice

Tablica w JavaScript jest obiektem i ma wiele wbudowanych metod, które ułatwiają operacje na elementach. Można do niej dodawać, usuwać i modyfikować elementy oraz wyszukiwać elementy o określonej wartości lub indeksie.

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

console.log(fruits.length); // 3
console.log(fruits[0]); // 'apple'

fruits.push('grape'); // dodaje nowy element na koniec tablicy
console.log(fruits); // ['apple', 'banana', 'orange', 'grape']

fruits.pop(); // usuwa ostatni element z tablicy
console.log(fruits); // ['apple', 'banana', 'orange']

fruits.splice(1, 1); // usuwa jeden element z indeksem 1 z tablicy
console.log(fruits); // ['apple', 'orange']

fruits.reverse(); // odwraca kolejność elementów w tablicy
console.log(fruits); // ['orange', 'apple']
```

Wbudowane metody JavaScript do pracy ze zmiennymi

Metody do pracy ze stringami

- `length` : zwraca długość ciągu znaków.
- `charAt()` : zwraca znak na podanej pozycji.
- `concat()` : łączy dwa lub więcej ciągów znaków.
- `indexOf()` : zwraca pozycję pierwszego wystąpienia określonego ciągu znaków w łańcuchu lub -1, jeśli ciąg nie został znaleziony.
- `lastIndexOf()` : zwraca pozycję ostatniego wystąpienia określonego ciągu znaków w łańcuchu lub -1, jeśli ciąg nie został znaleziony.
- `replace()` : zastępuje określony ciąg znaków innym ciągiem znaków.
- `slice()` : zwraca wycinek ciągu znaków, zaczynając od określonej pozycji do końca lub do innej określonej pozycji.
- `substring()` : zwraca wycinek ciągu znaków, zaczynając od określonej pozycji do końca lub do innej określonej pozycji.
- `toLowerCase()` : zwraca ciąg znaków, w którym wszystkie litery zostały przekształcone na małe litery.
- `toUpperCase()` : zwraca ciąg znaków, w którym wszystkie litery zostały przekształcone na wielkie litery.

Oto przykład użycia niektórych z tych metod

```
let str = "Hello, world!";
console.log(str.length); // wyświetli 13
console.log(str.indexOf("o")); // wyświetli 4
console.log(str.replace("world", "John")); // wyświetli "Hello, John!"
console.log(str.slice(0, 5)); // wyświetli "Hello"
console.log(str.toLowerCase()); // wyświetli "hello, world!"
console.log(str.toUpperCase()); // wyświetli "HELLO, WORLD!"
```

Metody do pracy z numberami

- `toFixed(n)` - metoda ta zwraca wartość liczby z dokładnością do n miejsc po przecinku. Na przykład 3.14159.toFixed(2) zwróci 3.14.
- `toString()` - metoda ta zwraca wartość liczby jako ciąg znaków.
- `parseInt(string)` - metoda ta konwertuje ciąg znaków na liczbę całkowitą.
- `parseFloat(string)` - metoda ta konwertuje ciąg znaków na liczbę zmiennoprzecinkową.
- `isNaN(number)` - metoda ta zwraca wartość true, jeśli przekazany argument nie jest liczbą, w przeciwnym wypadku zwraca false.
- `Math.round(number)` - metoda ta zaokrągla liczbę do najbliższej liczby całkowitej.
- `Math.ceil(number)` - metoda ta zaokrągla liczbę do najbliższej większej liczby całkowitej.
- `Math.floor(number)` - metoda ta zaokrągla liczbę do najbliższej mniejszej liczby całkowitej.
- `Math.max(number1, number2, ...)` - metoda ta zwraca największą wartość spośród przekazanych argumentów.
- `Math.min(number1, number2, ...)` - metoda ta zwraca najmniejszą wartość spośród przekazanych argumentów.

Instrukcje warunkowe

Instrukcje warunkowe to konstrukcje w języku JavaScript, które pozwalają na wykonanie określonego bloku kodu tylko wtedy, gdy spełniony jest określony warunek. W JavaScript mamy dwa główne typy instrukcji warunkowych: if i switch.

Instrukcja if pozwala na wykonanie bloku kodu tylko wtedy, gdy spełniony jest określony warunek. Warunek ten określamy w nawiasach po słowie kluczowym if. Na przykład

```
if (x > 5) {
  console.log("x jest większe niż 5");
}
```

W tym przykładzie, jeśli zmienna x jest większa niż 5, zostanie wyświetlony komunikat w konsoli.

Można także dodać blok kodu, który zostanie wykonany w przypadku, gdy warunek nie jest spełniony, używając konstrukcji else

```
if (x > 5) {
  console.log("x jest większe niż 5");
} else {
  console.log("x jest mniejsze lub równe 5");
}
```

W tym przykładzie, jeśli zmienna x jest mniejsza lub równa 5, zostanie wyświetlony drugi komunikat.

Możemy także użyć instrukcji else if do przetestowania wielu warunków

```
if (x > 10) {
  console.log("x jest większe niż 10");
} else if (x > 5) {
  console.log("x jest większe niż 5, ale mniejsze lub równe 10");
} else {
  console.log("x jest mniejsze lub równe 5");
}
```

W tym przykładzie, jeśli zmienna x jest większa niż 10, zostanie wyświetlony pierwszy komunikat. Jeśli jest większa niż 5, ale mniejsza lub równa 10, zostanie wyświetlony drugi komunikat. W przeciwnym razie zostanie wyświetlony trzeci komunikat.

Instrukcja switch jest alternatywną metodą przetestowania wielu warunków. Pozwala na przetestowanie jednej zmiennej wobec wielu różnych przypadków. Na przykład

```
switch (day) {
  case "poniedziałek":
    console.log("Dzisiaj jest poniedziałek");
    break;
  case "wtorek":
    console.log("Dzisiaj jest wtorek");
    break;
  case "środa":
    console.log("Dzisiaj jest środa");
    break;
  default:
    console.log("Nieznany dzień tygodnia");
}
```

W tym przykładzie, na podstawie wartości zmiennej day, zostanie wyświetlony odpowiedni komunikat w konsoli. Jeśli wartość zmiennej nie odpowiada żadnemu z przypadków, zostanie wyświetlony komunikat "Nieznany dzień tygodnia".

Pętle

Pętla `for` to pętla sterowana licznikiem, która wykorzystuje trzy elementy: inicjalizację licznika, warunek kontynuacji i instrukcję zmiany licznika.

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

W powyższym przykładzie zmienna i jest inicjalizowana na początku pętli na wartość 0. Warunek kontynuacji i < 10 sprawdza, czy zmienna i jest mniejsza niż 10. Jeśli warunek jest spełniony, to instrukcje w bloku kodu pętli są wykonywane, a następnie wartość zmiennej i jest zwiększana o 1 za pomocą instrukcji i++.

Pętla for-of służy do iterowania po kolekcji elementów, takiej jak tablica, ciąg znaków lub inna kolekcja, która posiada właściwość Symbol.iterator. W każdym przebiegu pętli zmienna iteracyjna przyjmuje kolejną wartość z kolekcji. Pętla for-of może wyglądać następująco:

```
const arr = [1, 2, 3, 4, 5];
for (const element of arr) {
  console.log(element);
}
```

Metoda forEach jest dostępna dla większości kolekcji w JavaScript, w tym dla tablic, Map i Set. Jest to metoda wywoływana na kolekcji, która jako argument przyjmuje funkcję zwrotną, nazywaną również funkcją zwrotną lub funkcją zwrotną iteracji, która jest wywoływana dla każdego elementu w kolekcji. W funkcji zwrotnej można wykonywać dowolną operację na elemencie. Poniżej przykład użycia metody forEach:

```
const arr = [1, 2, 3, 4, 5];
arr.forEach(element => {
  console.log(element);
});
```

Funkcje

Funkcje są blokami kodu, które można wywołać w różnych miejscach programu, aby wykonać określone zadania. Funkcje są jednymi z podstawowych elementów programowania w JavaScript i są często wykorzystywane do organizowania kodu w moduły i zwiększenia jego czytelności.

Funkcje w JavaScript mogą być definiowane w różnych formach, na przykład jako funkcje anonimowe, wyrażenia funkcyjne lub jako funkcje nazwane. Funkcje mogą przyjmować argumenty (dane wejściowe), a następnie wykonywać określone operacje na tych argumentach. Mogą również zwracać wartości (dane wyjściowe) do miejsca, z którego zostały wywołane.

Funkcje w JavaScript mogą być wykorzystywane w różnych sposobach, na przykład do:

Wykonywania operacji na danych, takich jak dodawanie, odejmowanie, mnożenie, dzielenie itp. Tworzenia funkcji pomocniczych, takich jak funkcje sortowania, filtrowania i mapowania. Definiowania zdarzeń (event handlers), które zostaną wywołane w odpowiedzi na interakcje użytkownika z aplikacją. Tworzenia obiektów i klas, które są podstawowymi elementami programowania obiektowego w JavaScript. Funkcje w JavaScript są bardzo elastyczne i mogą być używane w różnych kontekstach, dzięki czemu są niezwykle przydatne w tworzeniu zaawansowanych aplikacji internetowych.

Oto kilka przykładów funkcji w JavaScript:

Funkcja, która dodaje dwie liczby i zwraca wynik:

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

const suma = dodaj(2, 3);
console.log(suma); // 5
```

Funkcja, która zwraca długość podanego napisu:

```
function dlugoscNapisu(napis) {
  return napis.length;
}

const dlugosc = dlugoscNapisu('JavaScript');
console.log(dlugosc); // 10
```

Funkcja, która wypisuje na konsolę wartości z podanej tablicy:

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

const liczby = [1, 2, 3, 4, 5];
wypiszTablice(liczby);
// 1
// 2
// 3
// 4
// 5
```

Funkcja, która zwraca wartość bezwzględną liczby:

```
function bezwzgledna(liczba) {
  if (liczba < 0) {
    return -liczba;
  } else {
    return liczba;
  }
}

const x = -5;
const y = 10;
console.log(bezwzgledna(x)); // 5
console.log(bezwzgledna(y)); // 10
```

Funkcja strzałkowa, która mnoży dwie liczby i zwraca wynik:

```
const pomnoz = (a, b) => a * b;

const wynik = pomnoz(2, 3);
console.log(wynik); // 6
```

Podstawy DOM

DOM (Document Object Model) to interfejs programistyczny, który umożliwia dostęp do elementów strony internetowej (takich jak nagłówki, obrazy, przyciski itp.) oraz manipulowanie nimi za pomocą języka JavaScript. W DOM każdy element strony jest

reprezentowany jako obiekt, który można modyfikować za pomocą właściwości i metod dostępnych w API DOM.

DOM umożliwia programistom tworzenie interaktywnych aplikacji internetowych poprzez reagowanie na zdarzenia, takie jak kliknięcia myszą czy wprowadzanie danych do formularzy. Dzięki temu, że można manipulować elementami strony internetowej, można zmieniać wygląd i zawartość strony bez konieczności przeładowywania całej strony.

W JavaScript istnieje wiele metod i właściwości, które pozwalają na dostęp i manipulowanie elementami strony za pomocą DOM. Przykładowe metody to `createElement()`, `appendChild()`, `removeChild()`, a przykładowe właściwości to `innerHTML`, `value`, `style`. Dzięki nim można tworzyć, dodawać, usuwać oraz modyfikować elementy strony internetowej w sposób dynamiczny.

Selectory CSS

Selektory CSS to mechanizmy służące do wybierania elementów na stronie internetowej, do których mają być zastosowane reguły stylów. Poniżej przedstawiam kilka podstawowych selektorów CSS:

Selektor elementu - wybiera elementy na podstawie nazwy tagu, np. p wybierze wszystkie akapity na stronie.

Selektor klasy - wybiera elementy na podstawie nazwy klasy, np. .red wybierze wszystkie elementy z klasą "red".

Selektor identyfikatora - wybiera elementy na podstawie identyfikatora, np. #main wybierze element z id "main".

Selektor atrybutu - wybiera elementy na podstawie wartości atrybutu, np. [type="text"] wybierze wszystkie elementy z atrybutem "type" równym "text".

Selektory CSS pozwalają nam na wybieranie elementów z drzewa DOM, tak aby móc wykonywać na nich operacje

Łapanie elementów za pomocą `querySelector` i `querySelectorAll`

Metoda `querySelector` w DOM pozwala na wyszukanie pierwszego elementu pasującego do podanego selektora CSS i zwrócenie go w postaci obiektu. Można ją wywołać na dowolnym elemencie DOM lub na całej stronie. Przykładowo

```
// znalezienie pierwszego elementu o klasie "example"
const element = document.querySelector('.example');
```

Metoda `querySelectorAll` pozwala na wyszukanie wszystkich elementów pasujących do podanego selektora CSS i zwrócenie ich w postaci kolekcji obiektów. Można ją wywołać na dowolnym elemencie DOM lub na całej stronie. Przykładowo

```
// znalezienie wszystkich elementów o klasie "example"
const elements = document.querySelectorAll('.example');
```

Obydwie metody przyjmują jako argument selektor CSS, który może być dowolnym z dostępnych selektorów CSS. Metoda `querySelector` zwraca tylko pierwszy pasujący

element, a metoda `querySelectorAll` zwraca wszystkie pasujące elementy w postaci kolekcji, która można przetwarzać za pomocą pętli lub metod kolekcji, takich jak `forEach()`.

Podstawowe metody DOM na elementach

- `getAttribute()` - zwraca wartość atrybutu określonego elementu.
- `setAttribute()` - ustawia wartość atrybutu określonego elementu.
- `removeAttribute()` - usuwa atrybut z określonego elementu.
- `createElement()` - tworzy nowy element.
- `appendChild()` - dodaje nowy element jako ostatnie dziecko innego elementu.
- `removeChild()` - usuwa określony element potomny.
- `innerHTML` - zwraca lub ustawia zawartość elementu w postaci kodu HTML.
- `textContent` - zwraca lub ustawia tekst znajdujący się wewnątrz elementu, bez kodu HTML.
- `style` - umożliwia ustawianie stylów CSS dla elementu.
- `classList.add(className)` - dodaje określoną klasę do elementu.
- `classList.remove(className)` - usuwa określoną klasę z elementu.
- `classList.toggle(className)` - dodaje klasę, jeśli jej nie ma, lub usuwa ją, jeśli już istnieje.
- `classList.contains(className)` - zwraca wartość logiczną true lub false, w zależności od tego, czy element ma określoną klasę.

Eventy

Eventy w JavaScript to zdarzenia, które są wywoływane w odpowiedzi na interakcje użytkownika z elementami strony lub na zmiany zachodzące na stronie.

W praktyce oznacza to, że kiedy użytkownik wykonuje jakąś akcję, na przykład klikając na przycisk, przesuwając myszką nad elementem, wprowadzając wartość do formularza itp., wtedy wywoływany jest określony event, który można przechwycić i obsłużyć za pomocą kodu JavaScript.

W celu obsługi eventów w JavaScript, można użyć metody `addEventListener()` na elementach DOM. Ta metoda pozwala na rejestrowanie funkcji do wywołania w momencie, gdy określone zdarzenie wystąpi.

Przykłady zdarzeń (eventów) w JavaScript to

`click` - wywoływany po kliknięciu na element `mouseover` - wywoływany po najechaniu myszką na element `mouseout` - wywoływany po zjechaniu myszką z elementu `keyup` - wywoływany po puszczeniu klawisza na klawiaturze `submit` - wywoływany po naciśnięciu przycisku "submit" w formularzu

Wykorzystanie eventów w JavaScript jest kluczowe dla interaktywnych aplikacji internetowych. Pozwala to na tworzenie reaktywnych i dynamicznych interfejsów, które odpowiadają na interakcje użytkownika i zmiany na stronie.

Przykład obsługi zdarzenia "click" za pomocą metody `addEventListener()`

html

```
<button id="my-button">Kliknij mnie!</button>
```

javascript

```
const button = document.querySelector('#my-button');

button.addEventListener('click', () => {
  console.log('Kliknięto przycisk!');
});
```

W tym przykładzie tworzymy przycisk za pomocą znacznika button w HTML. Następnie, używając metody `querySelector()`, pobieramy ten przycisk w kodzie JavaScript i zapisujemy go do zmiennej "button". W końcu, wywołujemy metodę `addEventListener()` na zmiennej "button" i przekazujemy do niej dwa argumenty - nazwę zdarzenia, czyli "click", oraz funkcję, która zostanie wywołana, gdy zdarzenie click zostanie wykryte.

W tym przypadku funkcja wyświetli w konsoli informację "Kliknięto przycisk!" za każdym razem, gdy użytkownik kliknie na przycisk "Kliknij mnie!".

Praca z formularzami

Pracowanie z formularzami HTML za pomocą JavaScript jest częstym zadaniem podczas tworzenia aplikacji internetowych. W JavaScript istnieje kilka sposobów na pobieranie danych z formularzy i manipulowanie nimi. Oto kilka sposobów na pracę z formularzami HTML w JavaScript:

Pobieranie wartości z formularza

Aby pobrać wartość pola formularza, można użyć właściwości value. Na przykład, aby pobrać wartość pola tekstowego

```
const input = document.querySelector('#my-input');
const inputValue = input.value;
```

Przechwytywanie zdarzeń formularza

Aby przechwycić zdarzenie wysłania formularza, należy użyć metody `addEventListener()` i zdarzenia submit.

```
<form id="my-form">
  <input type="text" name="name" placeholder="Imię">
  <input type="text" name="surname" placeholder="Nazwisko">
  <button type="submit">Wyślij</button>
</form>
```

```
const form = document.querySelector('#my-form');

form.addEventListener('submit', (event) => {
  event.preventDefault(); // zapobiegamy domyślnej akcji formularza

  const name = form.elements.name.value;
  const surname = form.elements.surname.value;

  console.log(`Wysłano formularz z imieniem ${name} i nazwiskiem ${surname}.`);
});
```

W tym przykładzie pobieramy wartości z pól "name" i "surname" formularza, a następnie wyświetlamy je w konsoli. Używamy też metody `preventDefault()`, aby zapobiec domyślnej

akcji formularza - w tym przypadku przekierowania na inną stronę.

Manipulowanie wartościami formularza

Aby zmienić wartość pola formularza za pomocą JavaScript, możesz użyć właściwości value, tak jak w przypadku pobierania wartości. Przykład

```
const input = document.querySelector('#my-input');
input.value = 'Nowa wartość';
```

Walidacja formularzy

JavaScript umożliwia również walidację formularzy - sprawdzanie, czy wartości wprowadzone przez użytkownika są prawidłowe. Praca z formularzami HTML jest ważnym elementem tworzenia aplikacji internetowych, a JavaScript dostarcza wiele narzędzi i metod, które ułatwiają tę pracę.
