Atrybut onclick vs. addEventListener

JavaScript pozwala nam reagować w skryptach na różne zdarzenia w przeglądarce – na przykład kliknięcie, ruch kursorem myszy na elemencie, zmiana wartości w obiekcie czy naciśnięcie klawisza na klawiaturze. Ta “reakcja” to oczywiście w praktyce przypisanie konkretnej funkcji wywoływanej w momencie zajścia (*ang. trigger*) zdarznia. W praktyce na egzaminach najczęściej spotykamy się z obsługą kliknięć.

Najprostsza, klasyczna metoda zrealizowania takiego przypisania funkcji to użycie atrybutu HTML – dla kliknięcia jest to oczywiście atrybut

`onclick` – zaś jako wartość podajemy nazwę funkcji – koniecznie z towarzyszącymi nawiasami okrągłymi:

```
<button onclick="pokazKomunikat()">Kliknij mnie</button>
<script>
    function pokazKomunikat() {
        alert("Przycisk został kliknięty!");
    }
</script>
```

Natomiast w takiej sytuacji przypisanie funkcji do obsługi zdarzenia tak naprawdę odbyło się w HTML – to w końcu do tagu przycisku dopisaliśmy atrybut:

`onclick="pokazKomunikat()"` . Jest to więc nienajlepsze rozdzielenie warstwy zawartości witryny (HTML) od jej logiki (JS) – lepiej byłoby takie przypisanie również zdefiniować w skrypcie, zamiast w elemencie HTML.

I dlatego możemy także dokonać naszego przypisania w JavaScript – co prawda nadal używamy atrybutu

`onclick` obiektu, ale nie trzeba tego wpisywać w dokumencie HTML, tylko w samym skrypcie – a skrypt może już znajdować się w oddzielnym pliku `skrypt.js` :

```
<button id="przycisk">Kliknij mnie</button>
<script src="skrypt.js"></script>
```

Zdefiniowaliśmy jedynie atrybut

`id` , aby móc przycisk uchwycić w JS. Po jego uchwyceniu, obsługę zdarzenia naszą własną funkcją zdefiniujemy następująco:

```
let przycisk = document.getElementById("przycisk");
przycisk.onclick = function () {
    alert("Przycisk został kliknięty!");
};
```

Dodatkowo, wykorzystaliśmy tutaj tzw. funkcję anonimową. Więcej na temat rodzajów funkcji w JS (klasyczne, anonimowe, strzałkowe) oraz zdarzeń (najważniejsze *events* do wykorzystania) opowiedzieliśmy w lekcji: Tworzenie własnych funkcji, zdarzenia. Nie będziemy teraz powtarzać zakresu wiedzy z tamtej lekcji – w razie wątpliwości co do rodzajów funkcji czy zdarzeń zajrzyj koniecznie najpierw do tamtego materiału.

Natomiast co jeszcze warto wiedzieć – jeśli przypiszemy nową funkcję do wartości atrybutu

`onclick` , to oczywiście poprzednia zostanie nadpisana. Czyli w takim przykładowym kodzie:

```
let przycisk = document.getElementById("przycisk");
```

```
przycisk.onclick = function () {
    alert("Kliknięcie - funkcja nr 1");
};
przycisk.onclick = function () {
    alert("Kliknięcie - funkcja nr 2");
};
```

Zawsze wykona się tylko alert z napisem

`Kliknięcie - funkcja nr 2`, jako że ta później zdefiniowana funkcja anonimowa nadpisała poprzednią. Dlaczego to ważne? Ponieważ wskutek nadpisywania wartości atrybutu HTML nie mogą zaistnieć dwie (lub więcej) funkcje obsługujące to samo zdarzenie elementu HTML. Natomiast staje się to możliwe w nowocześniejszym sposobie przypisywania funkcji do zdarzeń – poznajmy go zatem szczegółowo.

Lepsza kontrola nad zdarzeniami – addEventListener()

Jest to nowoczesny i profesjonalny sposób przypisywania funkcji do zdarzeń w JavaScript. Dlaczego jest lepszy od używania atrybutów HTML

`onclick`, `onchange`, `onkeydown` dla elementów? Co najmniej z kilku powodów:

- Dobrze oddzielamy HTML od JavaScript – osobno jest definicja obiektów (elementów) a osobno logika.
- Ten sposób pozwala dodać więcej niż jedną funkcję do tego samego zdarzenia!
- Oczywiście metoda zadziała z różnymi typami zdarzeń (*click*, *keydown*, *mouseover*, *submit*) podawanymi jako pierwszy parametr metody.

Zobaczmy zatem ten najnowszy sposób w akcji – dla każdego przypadku załóżmy, że w dokumencie HTML istnieje przycisk oraz podpięto skrypt z pliku

`skrypt.js`:

```
<button id="przycisk">Kliknij mnie</button>
<script src="skrypt.js"></script>
```

Rozpocznijmy od wykorzystania klasycznej funkcji, czyli posiadającej swoją nazwę (czyli nie jest to ani funkcja anonimowa, ani strzałkowa):

```
function pokazKomunikat() {
    alert("Kliknięto przycisk!");
}
document.getElementById("przycisk").addEventListener("click", pokazKomunikat);
```

Co ważne – nie dodajemy już nawiasów okrągłych – w drugim argumencie metody

`addEventListener()` zapisaliśmy przecież: `pokazKomunikat` – jest to tzw. referencja do funkcji, czyli wskazanie w stylu: "*Tu jest funkcja, ale uruchom ją później, kiedy zajdzie zdarzenie*". Wpisanie tam `pokazKomunikat()` oznaczałoby natychmiastowe wywołanie funkcji – czyli uruchomienie jej od razu w momencie, gdy przeglądarka "czyta" kod pliku `skrypt.js`.

Ponadto – zwróćmy uwagę – pierwszym argumentem metody jest nazwa zdarzenia, czyli np. "*click*", "*keydown*", "*mouseover*", "*submit*" – bez "on" na początku. Bo nie chodzi tu o atrybut HTML, tylko nazwę samego zdarzenia! Listę zarówno zdarzeń, jak i atrybutów HTML im odpowiadających znajdziemy tutaj: Tworzenie własnych funkcji, zdarzenia.

A teraz pora przetestować kolejną cechę metody – możliwość przypisania kilku funkcji do obsługi tego samego zdarzenia. Rozważmy taki skrypt:

```
// Funkcja nr 1: pokazuje alert
function pokazAlert() {
    alert("Kliknięto przycisk!");
}
// Funkcja nr 2: zmienia kolor tła przycisku
function zmienKolor() {
    document.getElementById("przycisk").style.backgroundColor = "lightgreen";
}
// Dodajemy obie funkcje do zdarzenia "click"
let przycisk = document.getElementById("przycisk");
przycisk.addEventListener("click", pokazAlert);
przycisk.addEventListener("click", zmienKolor);
```

Co się stanie po kliknięciu? Najpierw uruchomi się funkcja

`pokazAlert()` i wyświetli komunikat, a następnie zadziała `zmienKolor()` – i tło przycisku zmieni się na zielone. Dlaczego zadziałają obie, bez nadpisania wcześniejszej kolejną? Ponieważ w odróżnieniu od nadpisywania atrybutów HTML, metoda `addEventListener()` pozwala nasłuchiwać tego samego zdarzenia wiele razy – każda dodana funkcja zostaje “zapamiętana” i uruchomiona w kolejności, w jakiej została dodana.

Jeszcze więcej użytecznych informacji na temat metody

`addEventListener()` znajdziemy pod tym adresem.

Funkcja anonimowa w nasłuchiwaniu zdarzeń

Oczywiście w ramach metody nie musimy koniecznie używać referencji do klasycznej funkcji posiadającej swoją nazwę – możemy w samej metodzie zastosować funkcję anonimową:

```
document.getElementById("przycisk").addEventListener("click", function() {
    alert("Kliknięto przycisk!");
});
```

Nie tworzymy już osobno funkcji o nazwie na przykład

`pokazKomunikat()` – zamiast tego bezpośrednio w miejscu przypisania zdarzenia wpisujemy w stworzonej “w locie” funkcji anonimowej, co ma się wydarzyć. To tak, jakbyśmy powiedzieli przeglądarce: “*Kiedy ktoś kliknie ten przycisk, wykonaj taki oto fragment kodu*“. Funkcja nie ma nazwy, ale zadziała bezproblemowo dokładnie dla tego konkretnego zdarzenia kliknięcia. Jeśli zależy nam na krótkim i czytelnym kodzie, a dana funkcja nie będzie używana w innych miejscach – funkcja anonimowa jest idealnym wyborem!

Funkcja strzałkowa w nasłuchiwaniu zdarzeń

Najkrótszy zapis nasłuchiwania zdarzenia kliknięcia przycisku uzyskamy oczywiście stosując funkcję strzałkową:

```
document.getElementById("przycisk").addEventListener("click", () => {
    alert("Kliknięto przycisk!");
});
```

To zdecydowanie najkrótszy możliwy zapis funkcji – elegancko i nowocześnie. Tylko pamiętajmy, że funkcja strzałkowa nie ma swojego własnego kontekstu

`this` – mówiliśmy o tym w lekcji: Tworzenie własnych funkcji, zdarzenia. Załóżmy, że chcemy (zamiast pokazania okna typ alert) zmienić tło kliknięte przycisku:

```
document.getElementById("przycisk").addEventListener("click", () => {
    this.style.backgroundColor = "lightblue"; // Błąd! this nie wskazuje na przycisk!
});
```

Nie możemy wówczas użyć funkcji strzałkowej – kontekst

`this` wewnątrz niej nie odnosi się do przycisku! Funkcja strzałkowa nie “widzi” przycisku, bo dziedziczy `this` z kontekstu zewnętrznego, czyli np. z window. A przecież nie chcemy zmienić koloru tła całego okna, tylko konkretnego przycisku! Dlatego w takim przypadku musimy użyć funkcji anonimowej albo referencji do funkcji klasycznej posiadającej swoją nazwę.

Usunięcie nasłuchiwania – removeEventListener()

Kiedy dodajemy funkcję obsługującą zdarzenie elementu za pomocą

`addEventListener()`, to tworzymy istniejącą “reakcję” witryny na zajście zdarzenia. Jednak czasami zechcemy później tę reakcję usunąć. I właśnie do tego służy metoda `removeEventListener()`. Załóżmy istnienie na stronie dwóch przycisków:

```
<button id="przycisk">Pokaż komunikat</button>
<button id="usun">Usuń obsługę komunikatu</button>
<script src="skrypt.js"></script>
```

I rozważmy następujący skrypt:

```
function pokazKomunikat() { alert("Kliknięto!"); }
document.getElementById("przycisk").addEventListener("click", pokazKomunikat);
document.getElementById("usun").addEventListener("click", function() {
    document.getElementById("przycisk").removeEventListener("click", pokazKomunikat);
});
```

Na początku definiujemy funkcję nazwaną:

`pokazKomunikat()` i w globalnym zasięgu skryptu przypisujemy ją do obsługi zdarzenia kliknięcia pierwszego przycisku. Jednak w obsłudze kliknięcia przycisku drugiego, usuwamy tego “nasłuchiwacza” – po jednokrotnym kliknięciu przycisku drugiego, naciskanie pierwszego już nie spowoduje pokazania komunikatu.

Ale uwaga – usuwanie nasłuchiwania nie zawsze zadziała! Wszystko zależy od tego, czy kiedy dodawaliśmy zdarzenie w

`addEventListener()` to zastosowaliśmy klasyczną funkcję, anonimową czy strzałkową. I tu pojawiają się trzy różne przypadki, które omówmy.

Przypadek #1 – Funkcja nazwana (klasyczna)

Taka funkcja zadziała świetnie z

`removeEventListener()`, co widzieliśmy zresztą w przykładzie powyżej. Dysponując nazwą funkcji bez problemu możemy jej użyć w metodzie usuwającej nasłuchiwanie kliknięcia.

Przypadek #2 – Funkcja anonimowa

Domyślnie nie można jej usunąć poprzez

`removeEventListener()` – nasz przykład musiałby dla funkcji anonimowej (nadającej obsługę kliknięcia) wyglądać tak:

```
document.getElementById("przycisk").addEventListener("click", function() {
    alert("Kliknięto!");
});
document.getElementById("usun").addEventListener("click", function() {

    document.getElementById("przycisk").removeEventListener("click", function() {
    alert("Kliknięto!");});
});
```

Dlaczego

`removeEventListener()` nie zadziała? Bo te dwie funkcje są różnymi obiektami (instancjami) w pamięci! Choć wyglądają identycznie, to nie mają tej samej "tożsamości" – JS widzi obie jako oddzielne, niezależne, dwie niepowiązane funkcje anonimowe. A przecież metoda `removeEventListener()` potrzebuje wskazać dokładnie tę samą funkcję, która została przypisana do zdarzenia wcześniej – oby ją "odpiąć" od nasłuchiwania. A zatem funkcji anonimowej nie da się skutecznie usunąć później, jeśli nie zapisaliśmy jej wcześniej np. do zmiennej.

Lecz jeśli faktycznie ją "uratujemy" (zapiszemy w zmiennej, nadając jej "nazwę") to już uda nam się ją potem usunąć z obsługi kliknięcia:

```
// Przypisujemy funkcję anonimową do zmiennej
const reakcjaKlik = function() {
    alert("Kliknięto przycisk!");
};
// Dodajemy funkcję jako listener do przycisku
document.getElementById("przycisk").addEventListener("click", reakcjaKlik);
// Dodajemy drugi przycisk, który usuwa zdarzenie
document.getElementById("usun").addEventListener("click", function() {
    document.getElementById("przycisk").removeEventListener("click", reakcjaKlik);
});
```

Przypadek #3 – Funkcja strzałkowa

Można usunąć nasłuchiwanie, ale również tylko jeśli przypisaliśmy wcześniej funkcję do zmiennej o konkretnej nazwie:

```
const klikFunkcja = () => {
    alert("Kliknięto!");
};
document.getElementById("przycisk").addEventListener("click", klikFunkcja);
document.getElementById("usun").addEventListener("click", () => {
    document.getElementById("przycisk").removeEventListener("click", klikFunkcja);
});
```

Podsumowanie

- Atrybut HTML `onclick` pozwala wprost przypisać funkcję do kliknięcia przycisku – i analogicznie zadziałają atrybuty: `onchange`, `onchange`, `onkeydown` itd.

- Nieco lepszym podejściem jest przypisanie funkcji w samym skrypcie JS, z użyciem: `element.onclick = function() { ... }`. Jednak zmiana wartości atrybutu pozwala przypisać tylko jedną funkcję do danego zdarzenia – nowa funkcja nadpisze poprzednią.
- Profesjonalnym sposobem przypisywania zdarzeń jest metoda: `addEventListener()`, która pozwala też dodawać wiele reakcji do tego samego zdarzenia i świetnie oddziela logikę JS od warstwy HTML.
- Składnia metody dla dodania obsługi kliknięcia to: `element.addEventListener("click", funkcja);` – bez "on" przed nazwą zdarzenia!
- Przypisać możemy:
  - funkcję klasyczną (z nazwą): `function pokaz() { ... }`
  - funkcję anonimową: `function() { ... }`
  - funkcję strzałkową: `() => { ... }`

- Funkcja strzałkowa nie posiada własnego `this` – nie sprawdzi się np. przy zmianie tła klikniętego przycisku: `this.style.backgroundColor = "lightblue"; // nie działa`
- Aby usunąć nasłuchiwanie, stosujemy: `removeEventListener("click", funkcja);`. Jednak funkcję anonimową lub strzałkową możemy usunąć z obsługi zdarzenia tylko, jeśli wcześniej przypiszemy ją do zmiennej.