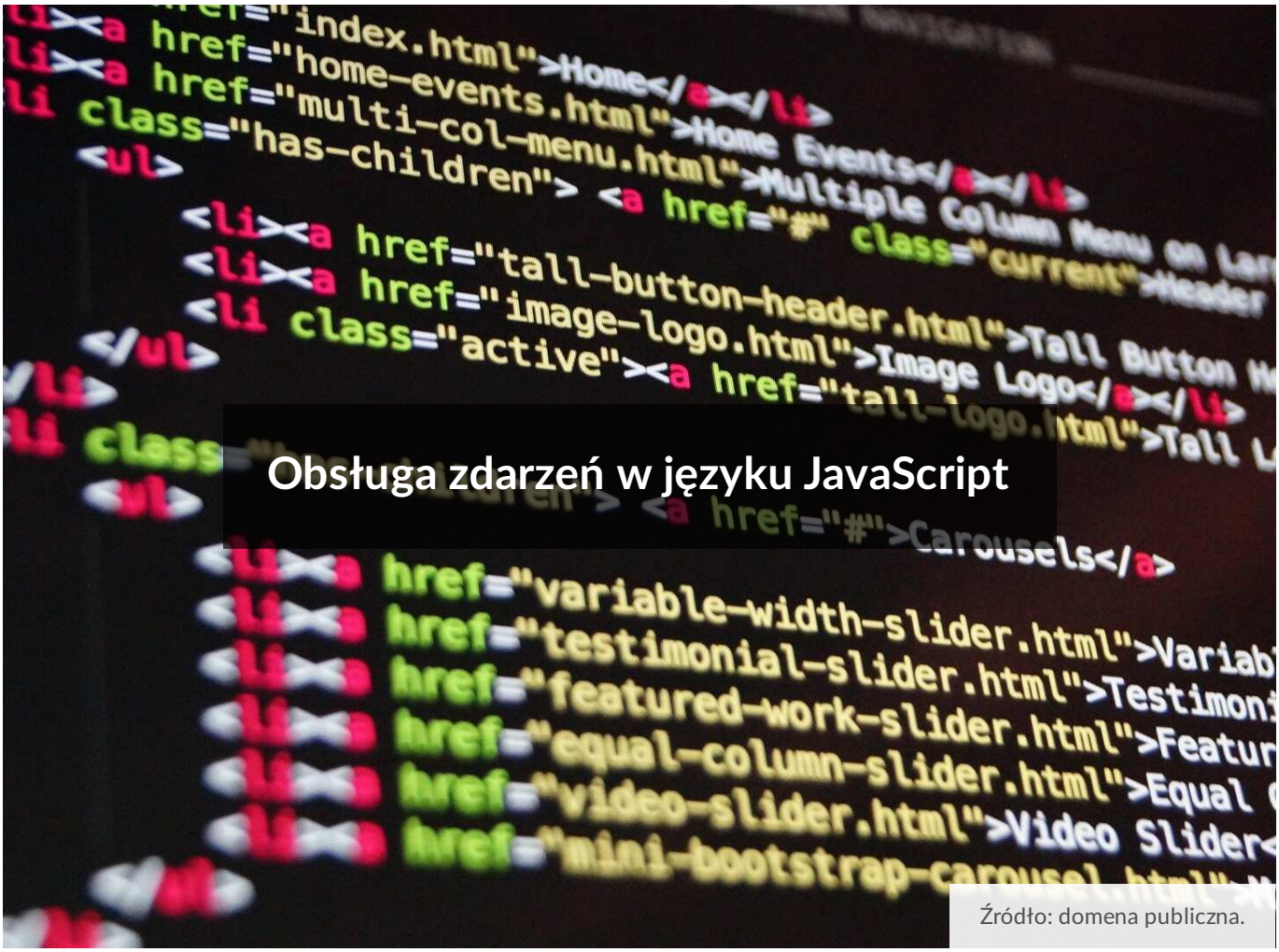
Obsługa zdarzeń w języku JavaScript

- Wprowadzenie
- Przeczytaj
- Infografika
- Sprawdź się
- Dla nauczyciela



Źródło: domena publiczna.

Interfejsy współczesnych witryn internetowych cechuje spora interaktywność – dzięki temu żywo reagują one na działania użytkownika. To właśnie dlatego JavaScript, jako pełnoprawny skryptowy język programowania, potrafi podejmować decyzje w zależności od zaistniałych okoliczności i zdarzeń w przeglądarce.

Takim zdarzeniem (ang. *event*) może być np. kliknięcie lub wskazanie elementu kursorem myszy, zmiana rozmiaru obiektów, wczytanie zawartości plików witryny, wciśnięcie wybranych klawiszy na klawiaturze etc.

Zadaniem programisty jest więc przygotować kody źródłowe własnych funkcji, które zrealizują odpowiednią obsługę wybranych zdarzeń - tym właśnie zajmiemy się w niniejszym e-materiale.

Pozostałe e-materiały z tej serii:

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

Twoje cele

- Zidentyfikujesz rodzaje często wykorzystywanych zdarzeń zachodzących w przeglądarce internetowej.
- Sklasyfikujesz sposoby przypisania wywołań własnych funkcji do obsługi zdarzeń.

Przeczytaj

Pełnoprawny, skryptowy język programowania JavaScript umożliwia przeglądarce internetowej **podejmowanie decyzji lub inicjowanie działań** w zależności od akcji podjętych przez użytkownika w interfejsie witryny.

Taką **akcję użytkownika** zarejestrowaną przez przeglądarkę określamy w programowaniu jako **zdarzenie** (ang. *event*). Zadaniem programisty webowego jest więc **przygotowanie obsługi wybranych zdarzeń**, czyli w praktyce stworzenie własnego kodu źródłowego, zazwyczaj w postaci funkcji.

Funkcja to w programowaniu fragment kodu **realizujący konkretne zadanie**. Idealnie więc zgrywa się to ze zdarzeniami zachodzącymi w interfejsie strony – do zajścia wybranego zdarzenia można przecież wygodnie przypisać wywołanie własnej funkcji. Tak właśnie powstaje obsługa zdarzenia zdefiniowana przez programistę.

Czym dokładnie jest zdarzenie?

W praktyce zdarzenia w przeglądarce nie zawsze stanowią jedynie reakcję na akcję internauty. Owszem, najczęściej obsługiwane przez programistę sytuacje rzeczywiście zainicjowane będą bezpośrednimi działaniami użytkownika - na przykład kliknięciem, wskazaniem za pomocą kursora, wciśnięciem klawisza na klawiaturze itd.

Lecz oprócz tego typu bezpośrednich reakcji istnieją także **zdarzenia zachodzące wskutek mechaniki działania** samej witryny - przykładem jest tu zdarzenie `load`, które zachodzi w momencie wczytania np. obrazka, pliku lub nawet całej podstrony. Jaka jest więc pełna i możliwie dokładna definicja zdarzenia?

> **Definicja: Zdarzenie**
>
> (ang. *event*) to **dająca się wyróżnić sytuacja** zachodząca w oknie przeglądarki internetowej, zarejestrowana automatycznie przez wbudowany w przeglądarkę mechanizm tzw. nasłuchiwania zdarzeń (ang. *event listening*).

Gdy się nad tym zastanowić, wspomniane **nasłuchiwanie zdarzeń** nie jest wcale niczym nowym ani zaskakującym w systemach komputerowych. W końcu na tej samej zasadzie działa np. obsługa ikon pulpitu czy paska zadań w systemie Windows albo ekran główny w telefonie z systemem Android.

Choć brzmi to dość niepokojąco, wiele programowalnych urządzeń komputerowych **nieustannie śledzi nasze poczynania**. Oczywiście dokonuje tego jedynie na ściśle określonych i transparentnych zasadach związanych z udostępnionymi użytkownikowi elementami interfejsu.

Lecz co do zasady, zadaniem dobrze napisanej aplikacji (w tym również aplikacji webowej) jest **jak najszybciej reagować na zaistniałe zdarzenia**. To właśnie tak pożądana przez użytkowników **interaktywność** aplikacji, systemu czy urządzenia. Dlatego również przeglądarka internetowa została stworzona według tej samej idei jak najszybszego **obsługiwania zdarzeń**.

Rodzaje zdarzeń w przeglądarce

Dokonajmy teraz przeglądu **popularnych zdarzeń**, czyli takich, których obsługę często definiują programiści JavaScript – dla zwiększenia czytelności listy, zdarzenia podzielono na kilka kategorii:

Zdarzenia związane z kursorem myszy

Nazwa zdarzenia	Zastosowanie, czyli kiedy wystąpienie zdarzenia zostanie zarejestrowane podczas nasłuchiwania
`click`	Klasyczne kliknięcie, które rozumiemy jako naciśnięcie przycisku myszy, a następnie jego zwolnienie, które nastąpiło w czasie, gdy wskaźnik (kursor) myszy wciąż znajdował się na elemencie (obiekcie).
`dblclick`	Dwukrotne kliknięcie elementu (z ang. double click).
`mousedown`	Wystąpi w sytuacji, gdy przycisk myszy został wciśnięty, kiedy kursor znajdował się na elemencie. W odróżnieniu od zdarzenia `click` kursor nie musi znajdować się na obiekcie w momencie zwalniania przycisku.
`mouseup`	Wystąpi w sytuacji, gdy przycisk myszy został zwolniony, kiedy kursor znajdował się na elemencie. W odróżnieniu od zdarzenia `click` kursor nie musi znajdować się na obiekcie w momencie, gdy przycisk myszy został wciśnięty.
`mousemove`	Rejestrowane za każdym razem, gdy kursor myszy poruszy się, znajdując się na elemencie.
`mouseenter`	Zachodzi, kiedy wskaźnik myszy wkroczy z obszaru spoza elementu i znajdzie się na obiekcie. Jednak w odróżnieniu od zdarzenia `mousemove` nie zostanie zarejestrowane ponownie, dopóki kursor nie opuści obiektu.
`mouseover`	Zachodzi, kiedy wskaźnik myszy wkroczy z obszaru spoza elementu i znajdzie się na obiekcie lub także na jego elementach potomnych w hierarchii DOM.
`mouseleave`	Zostanie zarejestrowane, gdy kursor myszy opuści dany obiekt.

Nazwa zdarzenia	Zastosowanie, czyli kiedy wystąpienie zdarzenia zostanie zarejestrowane podczas nasłuchiwania
`mouseout`	Zostanie zarejestrowane, gdy kursor myszy opuści dany obiekt lub także jego elementy potomne w hierarchii DOM.

Zdarzenia związane z klawiaturą

Nazwa zdarzenia	Zastosowanie, czyli kiedy wystąpienie zdarzenia zostanie zarejestrowane podczas nasłuchiwania
`keyDown`	Rejestrowane, kiedy klawisz na klawiaturze zostaje wciśnięty.
`keyUp`	Rejestrowane, kiedy wciśnięty klawisz klawiatury zostaje zwolniony.
`keyPress`	Zachodzi pomiędzy zdarzeniami `keyDown` i `keyUp` oraz dodatkowo dla znaków alfanumerycznych przekazuje informację o kodzie `Unicode` wprowadzonego znaku we właściwości `charCode`. Natomiast kody klawiszy innych niż mających reprezentację znakową są rejestrowane identycznie jak w zdarzeniach `keyDown` i `keyUp`, czyli we właściwości `keyCode`.

Zdarzenia w interfejsie witryny

Nazwa zdarzenia	Zastosowanie, czyli kiedy wystąpienie zdarzenia zostanie zarejestrowane podczas nasłuchiwania
`focus`	Wystąpi w momencie, gdy element HTML znajdzie się w centrum uwagi – np. gdy używamy w witrynie klawisza `Tab`, kolejne elementy będą właśnie uzyskiwać wyróżnienie (ostrość, uwagę) w interfejsie.
`blur`	W przeciwieństwie do zdarzenia `focus`, zostanie zarejestrowane w momencie, gdy element przestanie być w centrum uwagi, czyli utraci wyróżnienie (ostrość, uwagę) w interfejsie.
`resize`	Zachodzi w momencie zmiany rozmiaru widoku dokumentu w przeglądarce.
`load`	Rejestrowane, kiedy przeglądarka zakończy wczytywanie podstrony.
`unload`	Zachodzi jednokrotnie w momencie, gdy użytkownik opuszcza interfejs – np. kiedy karta przeglądarki zostaje zamknięta lub gdy naciśnięto przycisk `Wstecz`.
`drag`	Występuje w sytuacji, gdy dany obiekt jest przenoszony w interfejsie.

Nazwa zdarzenia	Zastosowanie, czyli kiedy wystąpienie zdarzenia zostanie zarejestrowane podczas nasłuchiwania
`drop`	Występuje w sytuacji, gdy dany obiekt jest upuszczany na określonym celu w interfejsie.

Zdarzenia związane z formularzami

Nazwa zdarzenia	Zastosowanie, czyli kiedy wystąpienie zdarzenia zostanie zarejestrowane podczas nasłuchiwania
`submit`	Wystąpi w momencie podsumowania (wysłania) danych całego formularza.
`reset`	Zachodzi w sytuacji zresetowania stanu formularza, czyli wprowadzonych do jego kontrolek danych wejściowych (tekstu, wyborów, wskazań).
`change`	Zachodzi w momencie, gdy zmienia się zawartość lub stan kontrolki formularza – np. zmieniono tekst w polu `<input>` lub `<textarea>` albo wskazano opcję na liście wyboru `<select>`.
`select`	Rejestrowane w sytuacji, gdy użytkownik zaznaczy tekst w polu `<input>` lub `<textarea>`.
`input`	Zostanie zarejestrowane, gdy w polu `<input>` formularza pojawią się dane wejściowe, czyli w praktyce wpisany przez użytkownika tekst.

Jak widać, do naszej dyspozycji oddano wiele różnorodnych zdarzeń, których obsługę możemy zdefiniować w języku JavaScript. Powyższa lista stanowi przegląd jedynie najpopularniejszych zdarzeń – w dokumentacji języka JS odnajdziemy ich jeszcze więcej!

Teraz możemy przejść do stworzenia obsługi sytuacji automatycznie rejestrowanych w interfejsie witryny. Dokonamy tego z użyciem własnych funkcji.

Własne funkcje obsługujące zdarzenia

W języku JavaScript zrealizowanie **obsługi zdarzenia** polega najczęściej na **przypisaniu** do automatycznej rejestracji tego zajścia w przeglądarce **wywołania własnej funkcji**.

Jak już wspomnieliśmy, funkcja to **wydzielony fragment kodu** spełniający **konkretne zadanie**. Pasuje to wręcz idealnie do stworzenia różnych reakcji elementów interfejsu w odpowiedzi na poczynania internauty czy zajścia w samej przeglądarce.

Istnieje klika metod zdefiniowania tego przypisania wywołania własnej funkcji do momentu zajścia zdarzenia (ang. *trigger*, - dosł. naciśnięcie spustu broni). Zacznijmy od metody klasycznej, choć nieoptymalnej.

Przypisanie z użyciem atrybutu HTML

Zrealizujmy **przypisanie własnej funkcji** do obsługi najpopularniejszego zdarzenia `click`. Załóżmy, że naszym celem jest **wyświetlenie małego okienka dialogowego** (ang. *alert*), za pomocą którego przywitamy się ze światem. Jednak ma to nastąpić nie przy wczytaniu strony do przeglądarki, lecz w **momencie kliknięcia w przycisk** z napisem `Test`.

Klasyczna metoda połączenia wykonania funkcji z obsługą zajścia zdarzenia, jakim jest kliknięcie przycisku, polega na **zdefiniowaniu dla przycisku atrybutu** `HTML`, który wywołanie funkcji ma ustawione jako swoją **wartość**.

Przycisk zdefiniowany w HTML:

```
<input type="button" value="Test" onclick="hello()">
```

Zwróćmy uwagę na obecność **dodatkowego atrybutu** `HTML` o nazwie `onclick`, którego wartość ustawiono na **wywołanie** funkcji o nazwie `hello()`. Ponieważ jest to wywołanie funkcji, zawiera nawiasy okrągłe.

Samą funkcję oczywiście zdefiniujemy już w skrypcie:

```
<script>

  function hello()
  {
    alert("Witaj Świecie!");
  }

</script>
```

Ta klasyczna metoda jest bardzo prosta w zapisie. Reasumując: połączenie **funkcji** zapisanej w JavaScript z **zajściem zdarzenia** następuje dzięki zdefiniowaniu w HTML odpowiedniej **wartości dodatkowego atrybutu**.

> **Ciekawostka**
>
> Może nas nieco dziwić nazwa tego dodatkowego atrybutu `HTML` - dlaczego jest to `onclick`, a nie `click`, czyli nazwa samego zdarzenia w JavaScript? To właśnie dla odróżnienia tego zapisu w HTML od kodu źródłowego JS użyto dodatkowego przedrostka `on`, który należy odczytać jako: podczas zajścia zdarzenia.
>
> Zasada ta obowiązuje oczywiście także w przypadku innych zdarzeń - przykładowe atrybuty możliwe do wykorzystania to: `onmousedown`, `onload`, `onsubmit`.

> Częstym skrótem myślowym stosowanym przez wielu programistów jest używanie w mowie potocznej nazw atrybutów jako nazw zdarzeń. Nie jest to jednak formalnie poprawne, gdyż przedrostek `on` zawierają tylko atrybuty `HTML`, a nie zdarzenia w JavaScript.

Wadą tworzenia atrybutu `HTML` jest brak oddzielenia kodu JavaScript od źródła HTML, czyli zmieszanie warstwy zawartości strony (HTML) z front-endową mechaniką działania witryny (JavaScript).

Znacznie lepszym wyjściem jest **przypisanie wywołania własnej funkcji** do **obsługi zdarzenia** również za pomocą kodu JavaScript, czyli bez konieczności użycia dodatkowych zapisów HTML. W tym celu trzeba użyć tzw. **nasłuchiwacza zdarzeń**.

Zdefiniowanie nasłuchiwacza w JavaScript

Naszym celem ponownie jest **wyświetlenie małego okienka dialogowego**, za pomocą którego przywitamy się ze światem po **kliknięciu w przycisk** z napisem `Test`.

Przycisk w HTML:

```
1 <input type="button" id="test" value="Test">
```

Tym razem jednak nie użyjemy atrybutu `onclick`, tylko metody `addEventListener()` wywołanej na rzecz obiektu, którym jest nasz przycisk. Aby móc **uchwycić obiekt**, do kontrolki formularza dodaliśmy klasyczny atrybut `id="test"`.

> **Ciekawostka**
>
> Może się to wydawać trochę dziwne - nowy sposób miał przecież zaoszczędzić nam tworzenia dodatkowego atrybutu `HTML`, a tymczasem do przycisku dodano atrybut `id`.
>
> Owszem, jednak zadaniem atrybutu `id` jest **stworzenie identyfikatora**, który może posłużyć do **uchwycenia obiektu**, a nie przypisanie własnej funkcji do obsługi zdarzenia.
>
> Ponadto, gdyby zamiast metody `getElementById()` użyć do uchwycenia przycisku np. metody `querySelector()`, to atrybut `id` okazałby się całkowicie zbędny.

Pora zatem na podpięcie wykonania naszej funkcji `hello()` w nasłuchiwaczu zdarzeń - tym razem zostanie to już zrealizowane za pomocą kodu `JavaScript`, a nie z użyciem atrybutu `HTML`:

```
1 <script>
2
```

```
  function hello()
  {
    alert("Witaj Świecie!");
  }

  var przycisk = document.getElementById("test");
  przycisk.addEventListener("click", hello);

</script>
```

W tej metodzie lepiej oddzielono źródło JavaScript od języka opisowego HTML – nie mieszamy warstwy zawartości witryny oraz jej mechaniki. Przypisanie funkcji do zdarzenia nastąpiło przy użyciu **instrukcji języka JavaScript**, a nie dodatkowego **atrybutu HTML**.

Przeglądarka „nasłuchuje" zajścia zdarzenia i w momencie, gdy kliknięcie rzeczywiście następuje, wywołana zostaje do pracy funkcja o nazwie `hello`.

> **Ciekawostka**
>
> Zwróćmy uwagę, że tym razem odniesienie nastąpiło przez tzw. referencję (odniesienie po nazwie), a więc bez dodatkowych nawiasów okrągłych, jak to miało miejsce w wartości atrybutu `onclick`.

Referencja, funkcja anonimowa, funkcja strzałkowa

Oprócz przekazania funkcji do nasłuchiwacza poprzez referencję, można tego dokonać także na dwa inne sposoby – przyjrzyjmy się im bliżej.

Pierwszy sposób już znamy - wskazujemy przypisanie własnej funkcji do nasłuchiwacza poprzez **referencję**, czyli podając samą **nazwę funkcji**, bez obecności nawiasów:

```
<script>

  function hello()
  {
    alert("Witaj Świecie!");
  }

  var przycisk = document.getElementById("test");

  //Przypisanie przez referencję do własnej funkcji
  przycisk.addEventListener("click", hello);

```

```
</script>
```

Możemy także skorzystać z tzw. **funkcji strzałkowej**, czyli takiej, która zawiera uproszczoną, krótszą składnię aniżeli klasyczne wyrażenie funkcji:

```
<script>

  var przycisk = document.getElementById("test");

  //Użycie funkcji strzałkowej zamiast funkcji hello()
  przycisk.addEventListener("click", () => {
    alert("Witaj Świecie!");
  });

</script>
```

Oprócz tego możemy także zdecydować się na **funkcję anonimową**, czyli zdefiniowaną klasycznie z użyciem klauzuli `function`, lecz **bez podania nazwy funkcji**:

```
<script>

  var przycisk = document.getElementById("test");

  //Użycie funkcji anonimowej (bez nazwy)
  przycisk.addEventListener("click", function() {
    alert("Witaj Świecie!");
  });

</script>
```

Zapisy skrócone wydają się atrakcyjne, co zatem zyskujemy, **stosując referencję** do własnej funkcji? Otóż wówczas możliwe staje się **odpięcie wywołania funkcji** od nasłuchiwacza zdarzeń:

```
<script>

  function hello()
  {
    alert("Witaj Świecie!");
```

```
    przycisk.removeEventListener("click", hello);
  }

  var przycisk = document.getElementById("test");
  przycisk.addEventListener("click", hello);

</script>
```

Tym razem funkcja `hello()` po wyświetleniu okna dialogowego „odpina" swoje przypisanie do nasłuchiwacza zdarzenia kliknięcia przycisku – kolejne kliknięcia nie spowodują już pojawienia się okienka.

Jak zauważyliśmy, wywołanie metody `removeEventListener()` wymaga przesłania do niej dwóch argumentów - najpierw jest rodzaj zdarzenia, a później nazwa funkcji przypiętej do jego obsługi. To właśnie dlatego użycie funkcji anonimowych wyklucza użycie metody - nie dysponujemy przecież nazwą funkcji.

Kiedy zatem warto użyć zapisów skróconych? Logiczna odpowiedź nasuwa się sama - przede wszystkim tam, gdzie absolutnie nie zakładamy odpięcia wykonywanych instrukcji od nasłuchiwania zdarzenia.

Ponadto istnieje jeszcze kilka dodatkowych **ograniczeń zaawansowanych**, na przykład niemożność użycia w funkcjach strzałkowych słów kluczowych `this` oraz `super`, jak również brak dostępu do specjalnego obiektu dostępnego w funkcjach, który nosi nazwę `arguments`.

Warto także pamiętać, że funkcje strzałkowe **nie są obsługiwane** w przeglądarkach z rodziny Internet Explorer.

Słownik

> **hierarchia DOM**
>
> (ang. Document Object Model) określony przez organizację W3C standardowy model struktury całego dokumentu HTML, wykorzystywany przez współczesne przeglądarki internetowe; model ten jest obiektowy i niezależny od platformy sprzętowej czy używanego w projekcie języka programowania
> **front-end**
>
> ogół technologii webowych (HTML, CSS oraz w wielu zastosowaniach JavaScript), w których kody źródłowe wykonywane są przez przeglądarkę internetową klienta witryny oraz pozostają jawne, czyli może do nich zajrzeć każdy użytkownik serwisu, niezależnie od posiadanych uprawnień

Infografika

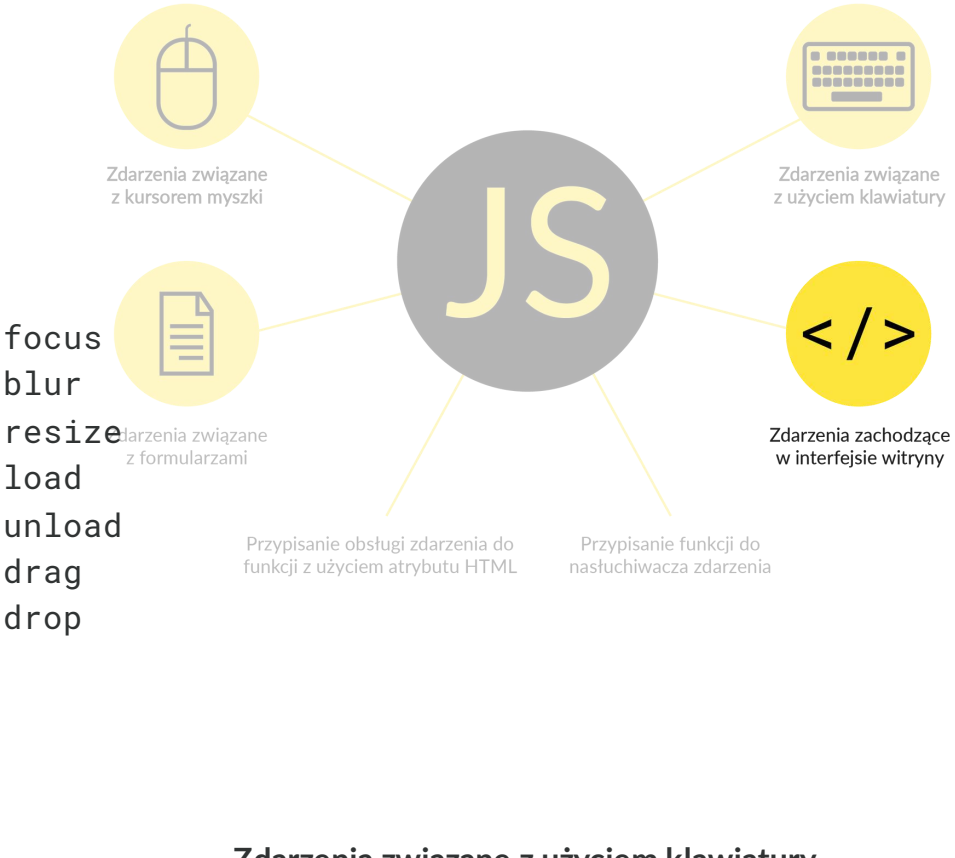
Polecenie 1

Zapoznaj się z infografiką przedstawiającą najczęściej wykorzystywane zdarzenia w języku JavaScript oraz ich obsługę z użyciem własnych funkcji.

Zdarzenia związane z kursorem myszki


```
click
dblclick
mousedown
mouseup
mousemove
mouseenter
mouseover
mouseleave
mouseout
```

Zdarzenia związane z formularzami



Zdarzenia związane z użyciem klawiatury

`keyDown`
`keyUp`
`keyPress`

Przypisanie funkcji do nasłuchiwacza zdarzenia

Metoda `addEventListeren()` posiada 2 parametry: nazwę zdarzenia oraz funkcję zrealizowaną jako:

- referencję (nazwa bez nawiasów),
- funkcję anonimową (bez nazwy),
- funkcję strzałkową.

Przypisanie obsługi zdarzenia do funkcji z użyciem atrybutu HTML

Sprawdź się

Pokaż ćwiczenia:

Ćwiczenie 1

Przyporządkuj wymienione zdarzenia do odpowiednich grup.

zdarzenia związane z kursorem myszy

zdarzenia związane z klawiaturą

zdarzenia związane z formularzami

`keyDown` `reset` `submit` `keyPress` `click` `dblclick` `mouseenter`

Ćwiczenie 2

Wskaż, które zdarzenie zostanie zarejestrowane w `JavaScript`, kiedy kursor myszy opuści dany obiekt.

- ◯ `mouseenter`
- ◯ `mousemove`
- ◯ `mousedown`
- ◯ `mouseout`

Ćwiczenie 3

Połącz każde ze zdarzeń rejestrowanych przez przeglądarkę z odpowiednim opisem jego działania.

`change`	wystąpi przy podsumowaniu (wysłaniu) danych całego formularza
`submit`	rejestrowane w momencie, gdy element znajdzie się w centrum uwagi
`focus`	zachodzi, gdy zmieni się zawartość lub stan kontrolki formularza

Ćwiczenie 4

Uzupełnij tekst brakującymi elementami, korzystając z podanych opcji.

W języku [ ] zrealizowanie [ ] zdarzenia polega zwykle na przypisaniu do automatycznej rejestracji tego zajścia w przeglądarce [ ] własnej funkcji. Funkcja to wydzielony fragment kodu, spełniający konkretne [ ]. Połączenie funkcji zapisanej w `JavaScript` z zajściem [ ] może nastąpić dzięki zdefiniowaniu w języku [ ] odpowiedniej wartości dodatkowego [ ].

atrybutu | zdarzenia | obsługi | wywołania | selektora | identyfikatora | `HTML` | `JavaScript` | zadanie

Ćwiczenie 5

Wskaż, który rodzaj przypisania własnej funkcji do wybranego nasłuchiwacza zapewnia programiście możliwość późniejszego odpięcia funkcji od obsługi zdarzenia.

- ◯ referencja do funkcji
- ◯ funkcja strzałkowa
- ◯ funkcja referencyjna
- ◯ funkcja anonimowa

Ćwiczenie 6

Uzupełnij tekst definicji brakującymi wyrazami.

Zdarzenie (ang. `event`) to dająca się wyróżnić sytuacja zachodząca w oknie przeglądarki internetowej, zarejestrowana automatycznie przez wbudowany w przeglądarkę mechanizm tzw. [ ] zdarzeń, ang. `event listening`.

Oprócz bezpośrednich reakcji na poczynania internauty istnieją także zdarzenia zachodzące wskutek mechaniki działania samej witryny - np. zdarzenie o nazwie [ ], które zachodzi w momencie wczytania do przeglądarki obrazka, pliku lub nawet całej podstrony.

Ćwiczenie 7

Dopasuj wybrane zapisy do właściwej grupy.

atrybuty HTML przypisujące funkcję do obsługi zdarzenia

[ ]

nazwy zdarzeń w języku JavaScript

[ ]

`onclick` `mousemove` `dblclick` `onkeypress` `onload` `unload`

Ćwiczenie 8

Uzupełnij brakujące elementy kodu źródłowego, tak aby poprawnie realizował obsługę zdarzenia kliknięcia przycisku.

```
<input type="button" id="y" value="Kliknij mnie!">
<script>
  function x()
{
    alert("Witaj Świecie!");
  }
  var przycisk = document.getElementById("              ");
  przycisk.addEventListener("click",              );
</script>
```

Dla nauczyciela

Autor: Mirosław Zelent
Przedmiot: Informatyka

Temat: Obsługa zdarzeń w języku JavaScript

Grupa docelowa:

Szkoła ponadpodstawowa, liceum ogólnokształcące, technikum, zakres podstawowy i rozszerzony

Podstawa programowa:

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;

III. Posługiwanie się komputerem, urządzeniami cyfrowymi i sieciami komputerowymi.

Zakres rozszerzony. Uczeń spełnia wymagania określone dla zakresu podstawowego, a ponadto:

1) projektuje rozbudowę i zakup nowego zestawu komputerowego oraz oprogramowania;

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):

- Zidentyfikujesz rodzaje często wykorzystywanych zdarzeń zachodzących w przeglądarce internetowej.
- Sklasyfikujesz sposoby przypisania wywołań własnych funkcji do obsługi zdarzeń.

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;
- telefony z dostępem do internetu;
- oprogramowanie dla języka Java SE 8 (lub nowszej wersji), w tym Eclipse 4.4 (lub nowszej wersji).

Przebieg lekcji

Przed lekcją:

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

Faza wstępna:

1. Ustalenie celu lekcji i kryteriów sukcesu.
2. Prowadzący prosi uczniów, aby zgłaszali swoje propozycje pytań do tematu. Jedna osoba może zapisywać je na tablicy. Gdy uczniowie wyczerpią swoje pomysły, a pozostały jakieś ważne kwestie do poruszenia, nauczyciel je dopowiada.

Faza realizacyjna:

1. **Praca z tekstem.** Nauczyciel ocenia, na podstawie informacji na platformie, stan przygotowania uczniów do zajęć. Jeżeli jest ono niewystarczające prosi o ciche zapoznanie się z treścią w sekcji „Przeczytaj".
2. **Praca z multimedium.** Nauczyciel wyświetla zawartość sekcji „Infografika". Uczniowie wspólnie zapoznają się z jej treścią. Zapisują ewentualne problemy i pytania. Następuje dyskusja, w trakcie której nauczyciel wyjaśnia niezrozumiałe treści.
3. **Ćwiczenie umiejętności.** Poszukiwanie najefektywniejszego rozwiązania problemu. Uczniowie wykonują w indywidualnie ćwiczenia nr 1-5 z sekcji „Sprawdź się", a następnie omawiają je na forum. Nauczyciel ocenia efektywność zastosowanego rozwiązania.

Faza podsumowująca:

1. Na koniec zajęć nauczyciel raz jeszcze wyświetla na tablicy temat lekcji i cele zawarte w sekcji „Wprowadzenie". W odniesieniu do ich realizacji dokonuje szczegółowej oceny rozwiązania zastosowanego przez wybranego ucznia.
2. Wybrany uczeń podsumowuje zajęcia, zwracając uwagę na nabyte umiejętności, omawia ewentualne problemy podczas rozwiązania ćwiczeń.

Praca domowa:

1. Uczniowie wykonują ćwiczenia 6-8 z sekcji „Sprawdź się".

Materiały pomocnicze:

- Oficjalna dokumentacja techniczna dla języka Java SE 8 (lub nowszej wersji).
- Oficjalna dokumentacja techniczna dla oprogramowania Eclipse 4.4 (lub nowszej wersji).

Wskazówki metodyczne:

- Uczniowie mogą wykorzystać treści w sekcjach: „Przeczytaj", „Infografika", „Sprawdź się" jako materiał do lekcji powtórkowej.