Biblioteka jQuery

Celem ćwiczenia jest zapoznanie studenta z biblioteką jQuery, przeznaczoną dla języka JavaScript. W trakcie wykonywania zadań student zdobędzie następujące umiejętności:

- posługiwanie się różnymi selektorami, aby odnaleźć pożądane fragmenty dokumentu,
- odseparowanie zawartości dokumentu od jego wyglądu oraz zachowania,
- dynamiczne modyfikowanie wyglądu strony,
- zaawansowana obsługa zdarzeń,
- dodawanie animacji do strony,
- zaawansowane zarządzanie DOM.

Do wykonania ćwiczenia potrzebny jest dowolny edytor plików tekstowych oraz przeglądarka internetowa.

Wprowadzenie

1. Stwórz plik `Selektory.html` i wypełnij go poniższym kodem.

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8" />
  <title>jQuery</title>
  <script type="text/javascript"
src="https://ajax.googleapis.com/ajax/libs/jquery/3.4.1/jquery.min.js"></script>
  <script type="text/javascript">
    $(document).ready(function () {
      alert('Czy wiesz, że możesz skopiować ten tekst przy pomocy skrótu Ctrl + C.');
    });
  </script>
</head>
<body>
</body>
</html>
```

Przyjrzyjmy się skryptowi zamieszczonemu w nagłówku dokumentu. Znak `$` jest funkcją, która jako parametr przyjmuje obiekt lub selektor i zwraca obiekt jQuery. Jest to konieczne aby móc korzystać z funkcji zawartych w bibliotece. Równoważnymi zapisami do powyższego są:

```
$().ready(function () {...});
$(function () {...});
jQuery(document).ready(function () {...});
jQuery().ready(function () {...});
jQuery(function () {...});
```

Zadaniem skryptu z przykładu jest wyświetlenie wiadomości w momencie, w którym całe drzewo dokumentu zostanie przygotowane (zdarzenie `ready`). Bardzo zbliżony efekt można osiągnąć umieszczając skrypt na samym końcu ciała dokumentu, albo przypisując funkcję do zdarzenia `onload` elementu `window` lub `body`. Istnieje jednak zasadnicza różnica między przedstawionymi podejściami. Zdarzenie `ready` nie czeka na załadowanie wszystkich obiektów danego elementu (na przykład pobranie obrazków), podczas gdy zdarzenie `onload` wywoływane jest dopiero po załadowaniu całej zawartości. Korzystaj więc `$(document).ready`, gdy potrzebujesz drzewa

dokumentu, a z `window.onload`, gdy chcesz na przykład odczytać wysokość załadowanego obrazka.

Selektory

2. Przedstawiona wcześniej funkcja `$` przyjmuje jako parametr obiekt lub selektor, czyli wyrażenie, które pozwala wybrać elementy z dokumentu. Selektory w jQuery wzorowane są na tych znanych Tobie z arkuszy stylistycznych CSS.

3. Dodaj do ciała dokumentu następującą zawartość:

```
<p id="a">a</p><p>b</p>
<p>c</p><p>d</p>
<div>e</div><div class="c">f</div>
<table>
    <tr><td>Pierwszy</td></tr>
    <tr><td>Drugi</td></tr>
    <tr><td>Trzeci</td></tr>
    <tr><td>Czwarty</td></tr>
    <tr><td>Piąty</td></tr>
    <tr><td>Szósty</td></tr>
</table>
<a href="wakalaka.html">Wakalaka</a>
<a href="asdf.pdf">PDF</a>
<a href="asdf.html">Asdf</a>
```

Zastąp skrypt w dokumencie poniższym kodem.

```
$(function () {
    $('#a').css('color', 'red').append(' doklejone');
});
```

Jest to przykład selektora wyszukującego elementy według identyfikatora. Dla odnalezionego elementu ustawiamy kolor czerwony i doklejamy dodatkowy tekst. Przykład ten ilustruje bardzo ważną własność funkcji z biblioteki jQuery, mianowicie większość z nich zwraca obiekt, na rzecz którego była wywoływana. Pozwala to na stosowanie wygodnych, „łańcuchowych” wywołań funkcji dla tego samego obiektu.

4. Dodaj do skryptu poniższy kod.

```
$('div').css('color', 'green');
$('.c').css('font-size', 'x-large');
$('tr + tr').css('color', 'aqua');
```

Są to przykłady wykorzystania selektorów znanych Tobie z CSS’a. `$('div')` zwraca kolekcję elementów `div`, `$('.c')` zwraca kolekcję elementów o klasie `c`, natomiast `$('tr + tr')` zwraca kolekcję elementów `tr` występujących po elemencie `tr`.

5. Teraz przyjrzymy się selektorom stworzonym dla grup elementów.

```
$('table tr:first').css('color', 'yellow');
$('tr:last').css('color', 'yellow');
$('p:even').css('color', 'orange');
```

jQuery udostępnia pseudoklasy `:first`, `:last`, `:even` oraz `:odd`, które z kolekcji elementów wybierają odpowiednio pierwsze, ostatnie, parzyste lub nieparzyste. W pierwszym przykładzie wybierane są wszystkie pierwsze elementy `tr` będące potomkami `table`, drugi przykład wybiera wszystkie ostatnie elementy `tr`, natomiast trzeci przykład zwraca wszystkie parzyste elementy `p`.

Dodatkowo, w celu bardziej szczegółowego wyboru elementów z kolekcji, można posłużyć się pseudoklasami `:eq(n)`, `:gt(n)` oraz `:lt(n)`, które wybierają odpowiednio elementy równe, większe bądź mniejsze od podanego indeksu `n`.

6. Kolejną grupą są selektory sprawdzające atrybuty.

```
$('[href]').css('border', '1px solid black').css('padding', '3px');
$('a[href*=kalaka]').css('border', '1px solid pink');
$('a[href$=\'.pdf\']').css('border', '1px solid red');
```

W pierwszym przypadku wyszukujemy wszystkie elementy zawierające atrybut `href`. Drugi oraz trzeci przykład ilustrują wyszukanie elementów `a` o wartości atrybutu `href` zawierającej ciąg znaków „`kalaka`” oraz kończących się na „`.pdf`”. Można również wyszukać atrybuty rozpoczynające się zadanym ciągiem znaków się (`^=`) oraz takie, które są równe bądź różne od danego ciągu znaków (`=` oraz `!=`).

7. jQuery dodatkowo udostępnia szereg pseudoklas pozwalających wyszukiwać elementy `input` (`:input`, `:test`, `:submit`, `:checkbox` itp.), które dodatkowo mogą spełniać określone kryteria (`:enabled`, `:disabled`, `:selected`, `:checked`).

Style

8. W poprzednich przykładach działanie selektorów ilustrowane było dodaniem odpowiedniego stylu do wybranych elementów. Do tego celu wykorzystywana była funkcja `css`, która występuje w dwóch typowych dla jQuery wariantach: pobierającym oraz ustawiającym. Wywołanie funkcji z pojedynczym parametrem będącym nazwą właściwości CSS, np. `color`, spowoduje pobranie wartości tego parametru. Natomiast, wywołanie tej samej metody z dwoma parametrami, z których pierwszy jest nazwą a drugi wartością, spowoduje ustawienie tego parametru na zadaną wartość. Dodaj poniższą linijkę do skryptu aby zilustrować działanie pobierania wartości.

```
alert('Wartość atrybutu color elementu o id="a" to ' + $('#a').css('color'));
```

9. Ustawianie styli w sposób dynamiczny może być efektowne, jednak należy również zadbać o to, aby było wykonane w sposób czytelny w kodzie. Podobnie jak przy omawianiu języka HTML oraz arkuszy CSS mówiliśmy o separacji treści od stylu, tak samo tutaj musimy zadbać o to, aby możliwie jak najbardziej odseparować style od zachowania strony, czyli od skryptów. Dlatego też zaleca się możliwie jak najrzadsze korzystanie z funkcji `css()`. Zamiast niej zdecydowanie lepiej użyć klas zdefiniowanych w osobnym arkuszu. Służą do tego metody `addClass`, `removeClass` oraz `hasClass`, które odpowiednio dodają klasę, usuwają klasę bądź sprawdzają, czy dany element ją posiada.

10. Stwórz nowy plik `Style.html` i wypełnij go szkieletem strony z punktu pierwszego. Dodaj w nagłówku strony sekcję `style`, a wewnątrz niej zamieść poniższą klasę.

```
.zielona{background: Green;}
```

Do ciała dokumentu wstaw poniższy kod,

```
<table>
    <tr><td>Pierwszy</td></tr>
    <tr><td>Drugi</td></tr>
    <tr><td>Trzeci</td></tr>
    <tr><td>Czwarty</td></tr>
    <tr><td>Piąty</td></tr>
    <tr><td>Szósty</td></tr>
</table>
<div><button>Zielony</button><button>mosteczek</button><span id="classic">Zmiana przez
dodawanie/usuwanie klasy.</span></div>
<div><button>ugina się.</button><span id="toggle">Zmiana przez przełączanie
klas.</span></div>
```

natomiast do sekcji ze skryptami następujący kod.

```
$(function () {
    $('button:contains(\'Zielony\')').click(function () {
        $('#classic').addClass('zielona');
    });
    $('button:contains(\'mosteczek\')').click(function () {
        $('#classic').removeClass('zielona');
    });
    $('button:contains(\'ugina\')').click(function () {
        $('#toggle').toggleClass('zielona');
    });
});
```

Na kliknięcie przycisków „`Zielony`” oraz „`mosteczek`” dodajemy lub usuwamy daną klasę CSS. Sprawdź, jaki będzie rezultat wielokrotnego kliknięcia na przycisk `Zielony`. Czy klasa została przypisana do elementu wiele razy?

Kliknięcie przycisku „`ugina się.`” powoduje wywołanie funkcji `toggleClass`, która co drugie zdarzenie dodaje i usuwa klasę przekazaną jako parametr.

Dodatkowo wykorzystujemy w tym przykładzie selektor z pseudoklasą `:contains`, który zwraca elementy zawierające w treści podany jako parametr ciąg znaków.

11. Teraz, wykorzystując odpowiednie selektory oraz zarządzanie klasami CSS, pokolorujemy co drugi wiersz tabeli. Dodaj do skryptu poniższą linijkę.

```
$('tr:even').addClass('zielona');
```

Jak widzisz, realizacja tego zadania została znacznie uproszczona dzięki wprowadzeniu selektorów `:even` oraz `:odd`.

Zdarzenia

12. Zdarzenia definiują zachowanie strony. Tradycyjnie w języku JavaScript przypisywano elementom zachowanie definiując obsługę zdarzeń w następujący sposób:

```
<button onclick="funkcja()">Przycisk</button>
```

Jest to sposób poprawny z punktu widzenia działania strony. Akcją na naciśnięcie przycisku będzie wywołanie funkcji `funkcja`. Przy większych serwisach internetowych nad takim kodem będzie jednak trudniej zapanować, gdyż będzie trzeba szukać zachowania strony wewnątrz jej treści. Podobnie jak w przypadku arkuszy stylistycznych i tu należy zadbać o możliwie jak największą separację definicji zachowania strony od jej zawartości. Aby to osiągnąć należy definiować obsługę zdarzeń w jednym miejscu, możliwie jak najszybciej. Można to osiągnąć wykorzystując zdarzenie `ready` dokumentu, opisane w punkcie 1. Przyjrzyj się ponownie skryptowi w dokumencie `Style.html` i zauważ, że definicja obsługi zdarzeń została tam właśnie w ten sposób zrealizowana.

Wszystkie funkcje dotyczące zdarzeń (np. `click`, `dbclick`, `focus`, `keyup`, `keydown`, `load`, `ready`, `hover`, `submit`) występują w dwóch postaciach. Pierwsza – bezparametrowa – służy jako wywołanie danego zdarzenia i powoduje jego obsługę, jeśli taka została zdefiniowana. Druga wersja przyjmuje jako parametr funkcję, która jest wykonywana w chwili zajścia zdarzenia (np. bezparametrowego wywołania).

13. Stwórz nowy plik `Zdarzenia.html`, wypełnij go szkieletem strony z punktu 1 i wstaw do wnętrza następującą tabelkę.

```
<table>
    <tr><td>Wlazł</td></tr>
    <tr><td>kotek</td></tr>
    <tr><td>na</td></tr>
    <tr><td>...</td></tr>
</table>
```

Następnie do sekcji `script` wstaw poniższy kod.

```
function edit() {
    var row = $(this);
    row.html('<input id="edit" value="' + row.text() + '">');

    $('#edit').blur(function () {
        row.text($(this).val());
        row.one('click', edit);
    }).focus();
}

$(function () {
    $('td').one('click', edit);
});
```

W tym przykładzie, podobnie jak w poprzednich, przypisujemy do elementów `td` obsługę zdarzenia `click` jako wywołanie funkcji `edit`. Tym razem robimy to jednak wykorzystując metodę `one`, która wiąże zdarzenie z funkcją obsługi tylko na jedno wywołanie. Wewnątrz funkcji `edit`, która odpowiada za obsługę kliknięcia, musimy więc ponownie przypisać obsługę tego zdarzenia do elementu w momencie, w którym straci focus (zdarzenie `blur`).

Korzystamy tutaj z trzech metod do manipulowania zawartością: `text`, `html` oraz `val`. Metoda `val` służy do pobrania/ustawienia wartości danego elementu formularza (np. pola input). Metody `text` oraz `html` pobierają/ustawiają zawartość elementu HTML. Pierwsza w formie tekstowej (traktując znaczniki HTML jako tekst), a druga w formie HTML.