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,
- asynchroniczne ładowanie i modyfikowanie poszczególnych komponentów strony,
- dodawanie animacji do strony,
- zaawansowane zarządzanie DOM,
- posługiwanie się obiektami JSON.

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="http://ajax.googleapis.com/ajax/libs/jquery/1.9.1/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>
```

Zamień 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 (`^=`) 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 stylów 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: Pink;}
```

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`.

12. Spróbuj teraz samodzielnie napisać następujący selektor:

- Wybierz wszystkie punkty numerowanej listy, która ma przypisaną klasę `a`, które nie zawierają linków wewnątrz dowolnego elementu, który ma przypisaną klasę `b`.

Zdarzenia

13. Zdarzenia definiują zachowanie strony. Na zajęciach dotyczących języka JavaScript przypisywaliśmy 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. Jak wspomniano już wcześniej, 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).

14. 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 (np. pola `<input>`). Aby najlepiej zrozumieć różnicę pomiędzy metodami `text()` oraz `html()` przeprowadź eksperyment. Wstaw poniższy tekst do dowolnego z pól na stronie w aktualnej postaci.

```
<script>alert('!DANGER!')</script>
```

Następnie zamień metodę, w której wpisywana jest zawartość pola `<input>` do komórki tabeli z `text()` na `html()` i ponownie przetestuj wprowadzenie powyższego tekstu w dowolne pole.

Zamień deklarację zmiennej `row` na stałą. Czy program działa tak samo? Zmień deklarację funkcji przekazywanej do metody `blur` na funkcję ze strzałką. Czy program działa tak samo?

15. jQuery umożliwia również definiowanie własnych zdarzeń. Do tego celu służy metoda `bind()`, która w ogólnej postaci przyjmuje dwa parametry: nazwę zdarzenia oraz funkcję jego obsługi. Zdarzenie takie można następnie wywołać wykorzystując funkcję `trigger()`, przyjmującą jako parametr nazwę zdarzenia. Dodaj do skryptu wykonywanego po załadowaniu dokumentu poniższą linijkę tworzącą nowe zdarzenie:

```
$('button').bind('asdf', function () { alert('Asdf!'); });
```

Następnie wstaw wywołanie uprzednio utworzonego zdarzenia wewnątrz metody `edit()`:

```
$('button').trigger('asdf');
```

Dodaj na stronie element `<button>` i przetestuj działanie utworzonego zdarzenia.

Zauważ, że w ten sposób możesz przypisać wielokrotnie obsługę tego samego zdarzenia. Funkcje obsługi nie będą się wówczas nadpisywały tylko dodawały. Można w ten sam sposób wiązać funkcję obsługi z już istniejącymi zdarzeniami, takimi jak `click` czy `load`.

Dodane do elementów zdarzenia można łatwo usunąć, zarówno pojedynczo jak i wszystkie naraz. Służy do tego metoda `unbind()`. Wywołana z parametrem będącym nazwą zdarzenia wyłącza jego obsługę, natomiast użyta bez parametrów wyłącza obsługę wszystkich zdarzeń związanych z danym elementem. Przetestuj funkcję usuwania obsługi zdarzeń dodając na końcu funkcji `$()` wywołanie metody `unbind()` dla wszystkich elementów na stronie.

```
$('*').unbind();
```

AJAX

16. Technologia AJAX pozwala w zaawansowany sposób zarządzać asynchroniczną interakcją z serwerem. jQuery nie odbiera tej możliwości, dając do dyspozycji metodę `$.ajax()`. Udostępnia jednak szereg funkcji upraszczających asynchroniczną komunikację.

17. Stworzymy teraz stronę zawierającą kilka podstron, których zawartość będzie ładowana dynamicznie. Najpierw stwórz po jednym pliku dla każdej z utworzonych do tej pory stron i nazwij go tak samo jak oryginalną stronę, dodając na początku przedrostek *AJAX*. Na przykład dla strony *Selektory.html* nazwij nową stronę *AJAXSelektory.html*. Skopiuj do każdego z utworzonych plików samą treść stron źródłowych, czyli kod znajdujący się wewnątrz elementu body, a następnie stwórz nowy plik o nazwie *AJAX.html* i umieść w nim poniższy kod. Przetestuj działanie strony w przeglądarce Firefox.

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <script type="text/javascript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.9.1/jquery.min.js"></script>
    <style>
        a { margin: 10px; }
        #content { border: 1px solid black; padding: 20px; width: 50%; }
    </style>
    <script type="text/javascript">
        $(function () {
            $('a').click(function () {
```

```
                $('#content').load('AJAX' + $(this).attr('href'));
                return false;
            });
        });
    </script>
</head>
<body>
    <header><nav>
        <a href="Selektory.html">Strona1</a>
        <a href="Style.html">Strona2</a>
        <a href="Zdarzenia.html">Strona3</a>
    </nav></header>
    <div id="content"></div>
</body>
</html>
```

Jak widzisz stworzenie dynamicznej strony przy pomocy jQuery jest bardzo proste. W powyższym przykładzie wykorzystujemy metodę `load()`, która przyjmuje jeden parametr obowiązkowy i dwa opcjonalne. Pierwszy parametr to adres url zasobu, który ma zostać załadowany do danego elementu. Drugi parametr to ewentualne parametry wywołania żądania, natomiast trzeci parametr to funkcja, która ma zostać wykonana po zakończeniu żądania (po załadowaniu zawartości). Dodatkowo, w funkcji obsługi zdarzenia `click` elementów `<a>` dodajemy instrukcję `return false;`, która zapobiega wywołaniu domyślnej obsługi zdarzenia. Możesz usunąć tę linijkę aby przekonać się o jej działaniu.

Spróbuj teraz uruchomić ten sam przykład w przeglądarce Opera lub Chrome. Jak widzisz strona nie działa. Dzieje się tak dlatego, że interakcje asynchroniczne możliwe są jedynie w obrębie jednej domeny (wyjątkiem jest funkcja `getJSONP()`). Aby strona działała poprawnie musi zostać zatem umieszczona na serwerze. Umieść teraz wszystkie 7 plików w folderze *XAMPP\htdocs*. Następnie uruchom program XAMPP Control Panel i sprawdź, czy serwer Apache jest uruchomiony. Jeśli nie, to go uruchom, a następnie udaj się pod adres http://localhost/AJAX.html w celu przetestowania strony. Teraz witryna powinna działać we wszystkich przeglądarkach.

Przedstawione rozwiązanie ma swoje wady oraz zalety. Dynamiczne zachowanie strony jest bardzo efektowne i na początku przyjemne dla użytkownika, gdyż daje iluzję „lokalności" aplikacji. Spróbuj jednak skorzystać z przycisku wstecz przeglądarki. Zauważ, że adres strony pozostaje niezmieniony. Nawigacja jest więc wygodna, jednak należałoby również pomyśleć o obsłużeniu nawigacji poza stroną. Zaletą zaprezentowanego rozwiązania jest natomiast fakt, iż będzie ono działało również wtedy, gdy wyłączona zostanie obsługa JavaScript w przeglądarce. Przetestuj działanie strony dla takiego przypadku.

Animacje

18. Aby jeszcze bardziej uatrakcyjnić interfejs naszej strony można dodać do jej zachowania animacje. jQuery udostępnia kilka predefiniowanych animacji (`fadeIn`/`fadeOut`, `slideUp`/`slideDown`), których wykorzystanie wydaje się dziś już dość powszechne na stronach internetowych, jak również animację niestandardową, w której można wykonać dowolną modyfikację stylów elementów strony (metoda `animate()`).

    Najpierw skorzystamy z predefiniowanych animacji `fadeIn`/`fadeOut`. Dodaj wewnątrz funkcji obsługi kliknięcia po jednej linijce kodu przed oraz po wywołaniu metody load, tak jak pokazano poniżej i przetestuj działanie strony.

```
$('#content').fadeOut();
$('#content').load('AJAX' + $(this).attr('href'));
$('#content').fadeIn();
```

Animacja działa, jednak prawdopodobnie nie tak jak byśmy się tego spodziewali. Najpierw pojawia się nowa zawartość, a dopiero potem całość znika i pojawia się ponownie. Dzieje się tak dlatego, że wywołanie animacji nie jest operacją blokującą. Po wywołaniu funkcji `fadeOut()` wykonanie programu przechodzi do kolejnej linii, która ładuje zawartość strony. Aby osiągnąć pożądany efekt musimy zatem skorzystać z dwuparametrowego wywołania funkcji `fadeOut()`, które jako drugi parametr przyjmuje funkcję, która ma zostać wykonana po zakończeniu animacji. Zamień definicję funkcji obsługi zdarzenia na poniższą.

```
$('a').click(function () {
    var thisA = $(this);
    $('#content').fadeOut('medium', function () {
        $('#content').load('AJAX' + thisA.attr('href'), function () {
            $('#content').fadeIn();
        });
    });
    return false;
});
```

Teraz strona zachowuje się zgodnie z naszymi oczekiwaniami. Na kliknięcie dowolnego odnośnika uruchamiana jest animacja ukrycia zawartości. Gdy animacja się skończy, zawartość zacznie być ładowana. Gdy asynchroniczne wywołanie dobiegnie końca zostanie uruchomiona metoda ponownego pokazania zawartości.

W tym przykładzie pojawia się bardzo ważna kwestia, mianowicie zasięg działania zmiennej `this`. W poprzedniej wersji skryptu mogliśmy bez problemu z niej korzystać aby pobrać zawartość atrybutu `href` odnośnika. Teraz jednak, wykonujemy to wewnątrz obsługi zdarzenia dotyczącego innego elementu o identyfikatorze `content`. Zmienna `this` oznacza tam zatem zupełnie inny element! Dlatego też wewnątrz obsługi kliknięcia zapamiętaliśmy kliknięty odnośnik jako obiekt jQuery pod zmienną `thisA`, aby później móc z niego skorzystać wewnątrz funkcji obsługi zdarzenia zakończenia animacji `fadeOut`. Jest to kwestia, na którą należy zwrócić szczególną uwagę podczas zagnieżdżonych definicji obsługi zdarzeń.

19. Zamienimy teraz wywołanie predefiniowanych funkcji na własną animację, wykorzystując metodę `animate()`. Występuje ona w kilku wariantach. My skorzystamy z opcji przyjmującej jako parametry właściwości CSS oraz funkcję wywoływaną na zakończenie animacji. Zamień skrypt na stronie poniższym kodem.

```
$('a').click(function () {
    var thisA = $(this);
    $('#content').animate({height: '0px', width: '0px'}, function () {
        $('#content').load('AJAX' + thisA.attr('href'), function () {
            $('#content').animate({ height: '500px', width: '500px'});
        });
    });
    return false;
});
```

Jak widzisz zasada działania jest bardzo podobna jak w przypadku predefiniowanych funkcji animujących. Tym razem jednak, funkcja jako pierwszy parametr przyjmuje wartości właściwości stylistycznych, które mają zostać zmienione. Istotnym szczegółem tego sposobu animowania jest fakt, iż można jedynie manipulować właściwościami przyjmującymi wartości numeryczne. (Teoretycznie nie ma zatem na przykład możliwości zmiany kolorów. Można jednak temu zaradzić korzystając z dodatkowej biblioteki jQuery-ui).

20. Usuń linijkę, w której deklarowana jest zmienna `thisA`. Zamień funkcje przekazywane do metod `animate()` i `load()` na funkcje ze strzałką i zamień odwołanie do `thisA` na referencję `$(this)`.

DOM

21. jQuery udostępnia również szereg metod pozwalających przeszukiwać drzewo dokumentu. Do metod istniejących w JavaScripcie dochodzi szereg użytecznych i ciekawych funkcji, jak chociażby `parents()` – zwracająca wszystkich przodków elementu aż do korzenia, `closest()` – szukająca najbliższego elementu, czy `nextAll()` – zwracająca wszystkie elementy występujące po aktualnym, na danym poziomie. Co istotne, do każdej z tych metod można podać jako parametr wyrażenie, które wyszukiwane elementy mają spełniać.

    Oprócz wyszukiwania pojawiają się również metody umożliwiające edycję drzewa, takie jak `insertBefore()` – dołączająca element przed dopasowanymi elementami, `replaceWith()` – zastępująca dane elementy lub `clone()` – tworząca dokładną kopię elementu z możliwością skopiowania przypisanych funkcji obsługi zdarzeń.

    Zaprezentujemy działanie kilku z opisanych metod tworząc przykładową aplikację typu Drag&Drop. Stwórz nowy plik o nazwie *DOM.html* i wstaw do niego poniższą zawartość.

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>Q&A</title>
    <style>
        #r{ background-color: Red; }
        #g{ background-color: Green; }
        #b{ background-color: Blue; }
        #r, #g, #b { height: 100px; width: 500px;
                     border: 1px solid black; text-align: center; }
    </style>
    <script type="text/javascript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.9.1/jquery.min.js"></script>
    <script type="text/javascript">
        function Down() {
            var div = $(this);
            $('div').mousemove(function (e) {
                div.remove();
                div.insertAfter(e.target);
                div.mousedown(Down);
            });
            $('div').mouseup(function (e) {
                $('div').unbind('mousemove');
            });
        }
        $(document).ready(function () {
            $('div').mousedown(Down);
        });
    </script>
</head>
<body>
    <div id="r"></div>
    <div id="g"></div>
    <div id="b"></div>
</body></html>
```

    W powyższym przykładzie wykorzystujemy dwie metody modyfikujące drzewo: `remove()` oraz `insertAfter()`. Metoda `remove()` usuwa element z drzewa dokumentu, ale podobnie jak w przypadku JavaScript nie jest on niszczony, więc można z niego później skorzystać. Robimy to w kolejnym kroku, dodając usunięty element przed elementem, nad którym się aktualnie znajdujemy (`insertAfter()`). Istotne jest w tym przypadku, aby pamiętać o ponownym przypisaniu obsługi zdarzenia do dodanego elementu, ponieważ w chwili usunięcia go z drzewa wszystkie związane z nim zdarzenia zostają usunięte!

JSON

22. Ostatnią funkcją oferowaną przez jQuery, która zostanie omówiona, jest pobieranie i przetwarzanie danych w formacie JSON. Podobnie jak metoda `load()`, dane w formacie JSON pobierane są w sposób asynchroniczny. Służy do tego funkcja `$.getJSON()`. Zanim jednak z niej skorzystamy przyjrzyjmy się najpierw budowie obiektów JSON. Ich budowa opiera się na zasadzie słownika. Dla każdego klucza może istnieć pojedyncza wartość, tablica wartości lub kolejny klucz. Taka struktura powoduje, że bardzo łatwo można zserializować dowolne obiekty do tej postaci.

    Jako prosty przykład ilustrujący budowę obiektu JSON stworzymy stronę prezentującą listę prowadzących naszego ulubionego przedmiotu. Stwórz nowy plik *JSON1.html* i wstaw do niego poniższą zawartość.

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>JSON</title>
    <script type="text/javascript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.9.1/jquery.min.js"></script>
    <script type="text/javascript">
        var d =
            { "Pracownicy": {
                "Pracownik": [
                    { "Imie": "Maciej", "Nazwisko": "Zakrzewicz" },
                    { "Imie": "Maciej", "Nazwisko": "Piernik" }

                ]
            }
        };

        $(function () {
            for (var i = 0; i < d.Pracownicy.Pracownik.length; i++) {
                $('#x').append('<div>' + d.Pracownicy.Pracownik[i].Imie + ' ' +
d.Pracownicy.Pracownik[i].Nazwisko + '</div>');
            }
        });
    </script>
</head>
<body>
    <h1>Prowadzący mojego ulubionego przedmiotu</h1>
    <div id="x"></div>
</body>
</html>
```

    Zwróć uwagę na sposób nawigacji po polach utworzonego obiektu. Zmień powyższy kod w taki sposób, aby zastąpić pętlę funkcją `map()`.

23. Znając już sposób budowy obiektów JSON, możemy zacząć z nich korzystać. Stwórz plik *JSON2.html* i wypełnij go poniższym kodem.

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>JSON</title>
    <script type="text/javascript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.9.1/jquery.min.js"></script>
    <script>
        $(function () {
$.getJSON("http://api.flickr.com/services/feeds/photos_public.gne?jsoncallback=?", { tags:
"cars", format: "json" }, function (data) {
                $.each(data.items, function (i, item) {
                    $('div').append($("<img/>").attr("src", item.media.m));
                });
            });
        });
    </script>
</head>
<body>
    <div></div>
</body>
</html>
```

Powyższy prosty przykład ilustruje sposób pobrania obiektów w formacie JSON z przykładowej usługi (w tym wypadku jest to api flickr'a). Jego działanie opiera się na asynchronicznej metodzie `$.getJSON()`, która wykonuje żądanie do podanej jako parametr strony, oczekując na odpowiedź w postaci obiektu JSON. Gdy taka odpowiedź nadejdzie, uruchamiana jest funkcja, która została podana jako trzeci parametr wywołania.