

jQuery



v 1.1

PLAN

- [Wprowadzenie](#)
- [Szukanie elementów w jQuery](#)
- [Eventy](#)
- [Manipulacja DOM-em](#)

Wprowadzenie

Czym jest jQuery?

jQuery jest biblioteką napisaną w języku JavaScript służącą do operacji na drzewie DOM i ujednoliceniu działania stron na różnych przeglądarkach.

Najważniejsze funkcje, które implementuje jQuery:

- łatwiejsze wyszukiwanie elementów,
- lepsza kontrola dodawania i usuwania elementów,
- animacje,
- łatwiejsze w użyciu eventy,
- Ajax.

JavaScript a jQuery

JavaScript

```
var x, y;
if (self.innerHeight) { // all except Explorer
  x = self.innerWidth;
  y = self.innerHeight;
}
else if (document.documentElement &&
  document.documentElement.clientHeight) {
  // Explorer 6 Strict Mode
  x = document.documentElement.clientWidth;
  y = document.documentElement.clientHeight;
}
else if (document.body) { // other explorers
  x = document.body.clientWidth;
  y = document.body.clientHeight;
}
```

jQuery

```
var x = $(window).width();
var y = $(window).height();
```

Instalacja jQuery

- Ściągnij pliki ze strony <http://jquery.com/download>
- Dodaj ściągnięty skrypt do kodu HTML za pomocą tagu **script**, kolejność jest ważna – jQuery powinno być przed naszymi skryptami,
- Można też korzystać z plików hostowanych przez np. Google (tak zwane CDN).

Aktualnie dostępna w dwóch wersjach:

- 1.x – wspierająca przeglądarki <IE9.
- 2.x – wspierająca tylko nowsze przeglądarki.

Dokumentacja

- jQuery ma bardzo dobrze napisaną dokumentację. Można tam znaleźć dokładne opisy, przykłady użycia i najczęstsze problemy z danymi metodami.
- Przed użyciem każdej metody należy chociaż pobieżnie przeczytać jej dokumentację.
- <http://api.jquery.com>

Szukanie elementów w jQuery

DOMContentLoaded w jQuery

- Zanim przystąpimy do pisania skryptów korzystających z elementów DOM, musimy sprawdzić, czy został on załadowany.
- W **Vanilla JS** (czysty JavaScript) korzystamy m.in. z **DOMContentLoaded**.
- W jQuery możemy wykorzystać następujące dwa sposoby przedstawione po prawej stronie.

```
$(function() {  
    // tu nasz kod  
});
```

//lub

```
$(document).ready(function() {  
    // tu nasz kod  
});
```

Wszystkie zadania i przykłady
będziemy pisać wewnątrz
jednej z takich funkcji.

Wyszukiwanie elementów

- Kiedy korzystamy z jQuery, mamy możliwość łatwego wyszukiwania elementów za pomocą zapytań CSS.

Aby swobodnie korzystać z jQuery trzeba dobrze opanować wyszukiwanie elementów.

// Znajdź element o **id** top

\$('#top');

// Znajdź wszystkie elementy **li**

\$('li');

// Znajdź wszystkie elementy **li** wewnątrz **ul**

\$('ul li');

// Znajdź wszystkie elementy z klasą boxes

\$('.boxes');

Wyszukiwanie elementów

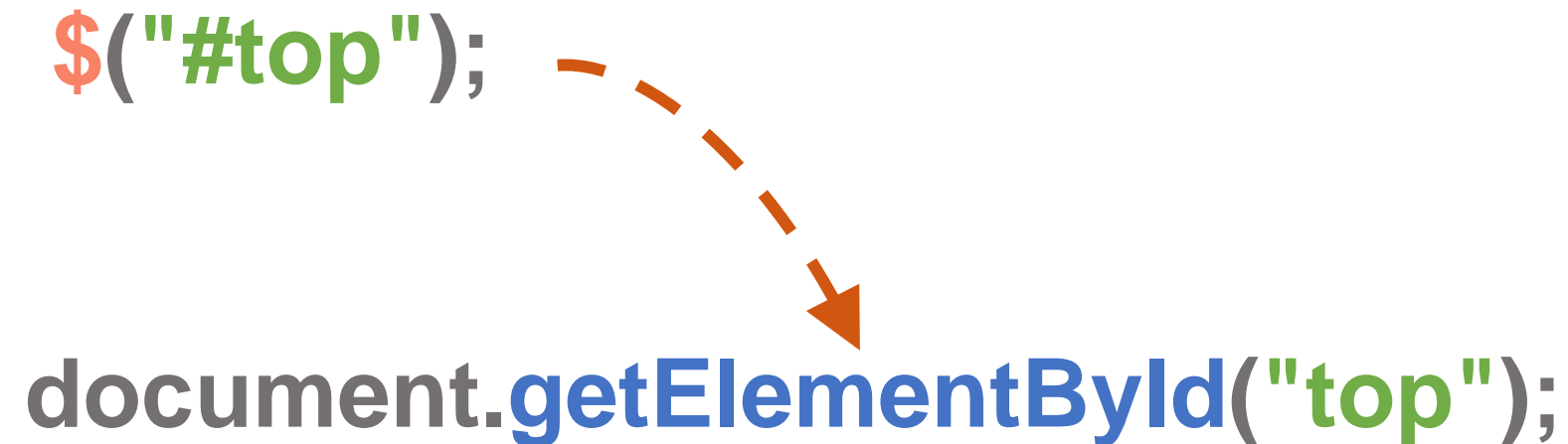
- W jakim celu opakujemy selektory CSS funkcją jQuery?
- Wyszukiwanie elementów za pomocą selektorów CSS daje nam ogromne możliwości, których nie ma w czystym JavaScriptcie.

Ile elementów zostanie znalezionych?

```
$('#top');
```

Wyszukiwanie elementów

- W podanym przykładzie jQuery napotyka na selektor ID, uruchamia zatem funkcję JavaScript **document.getElementById("top")**.
- Jeśli selektor jest bardziej zaawansowany, jQuery uruchamia silnik Sizzle służący do wyszukiwania elementów DOM.



A vertical dashed line separates the text on the left from the code on the right. On the right side, the code `$("#top");` is shown in red and green. A dashed orange arrow points from this code down to the code `document.getElementById("top");` which is shown in blue and green.

```
$("#top");  
  
document.getElementById("top");
```

Wyszukiwanie elementów

- jQuery uruchamia standardowe funkcje JavaScript, jeśli przeglądarka je obsługuje.
- W poniższym przypadku uruchomiona zostanie funkcja **querySelectorAll()**
- W przypadku starszych przeglądarek np. IE8, który nie obsługuje funkcji `querySelectorAll()`, jQuery skorzysta z silnika Sizzle.
- Ważne aby przypisywać znalezione elementy do zmiennych, jeśli chcemy ich potem wielokrotnie użyć, zapobiegnie to ponownemu przeszukaniu drzewa DOM. Jest to tzw. **caching**.

```
$('#top a');
```

Optymalizacja wyszukiwania

Jak jQuery interpretuje ten selektor?

- jQuery przeszukuje od prawej do lewej, chyba że pierwszy element to **ID**.
- W tym przykładzie jQuery znajduje wszystkie elementy **a**, następnie sprawdza, które z nich mają rodzica z klasą **menu**.
- Nieoptymalnie, prawda?

`$('.menu a');`

Optymalizacja wyszukiwania

- A co, gdybyśmy mieli na stronie 100 linków i tylko pięć z nich znajdowałoby się w menu?
- jQuery i tak przeszuka cały dokument.
- Można zoptymalizować wyszukiwanie na kilka sposobów.

//Użyć funkcji np. find() lub children()
`$('.menu').find('a');`

//Zagnieździć selektor
`$('a', $('.menu'));`

//Zmienić klasę na id
`$('#menu a');`

Co możemy zrobić z tymi elementami?

Po co wyszukujemy elementy?
Dlaczego jest to ważne?

Wyszukujemy elementy aby:

- nimi manipulować,
- animować je,
- usuwać,
- zmieniać,
- nadawać style lub klasy,
- itp.

```
$('.menu').find('a').css('color', 'red');
```

```
$('.menu').find('a').addClass('crazyColors');
```

```
$('.menu').find('a').removeClass('crazyColors');
```

```
$('.menu').find('a').toggleClass('crazyColors');
```

Czy element ma klasę?

Możemy również sprawdzić, czy dany element ma klasę za pomocą funkcji **hasClass()**

```
if ($('.menu').hasClass('crazyColors')) {  
    console.log('Menu ma klasę crazyColors');  
} else {  
    console.log('Menu nie ma klasy crazyColors');  
}
```


Czy element może zniknąć?

Funkcje powodujące przenikanie:

- fadeIn() – pojawienie się ukrytego elementu z efektem przenikania
- fadeOut() – zniknięcie widocznego elementu z efektem przenikania

```
$('.menu').find('a').fadeIn('slow');
```

```
$('.menu').find('a').fadeOut(1000);
```

Ustawianie atrybutów elementów?

Możemy pobierać i ustawiać atrybuty elementów np.

- class,
- id,
- type
- i inne, za pomocą funkcji **attr()**

```
<input value="" type="text" class="user-name">
```

```
var userName = $('input.user-name');
```

```
userName.attr('type');
```

```
userName.attr('type', 'password');
```

Pobranie

Ustawienie

each()

Oto jedna z ważniejszych
i popularniejszych funkcji jQuery:
each(index, element)

Jest bardzo wygodna przy rozbudowanej
strukturze DOM-u. Funkcja to tak naprawdę pętla,
która iteruje po elementach znalezionych
w DOM-ie.


Index zawiera numer kolejnego elementu, a
element to obiekt tego elementu.

HTML

```
<a href="http://jsfiddle.net">JSFiddle</a>  
<a href="http://codepen.io">CodePen</a>  
<a href="http://jsbin.com">JS Bin</a>
```

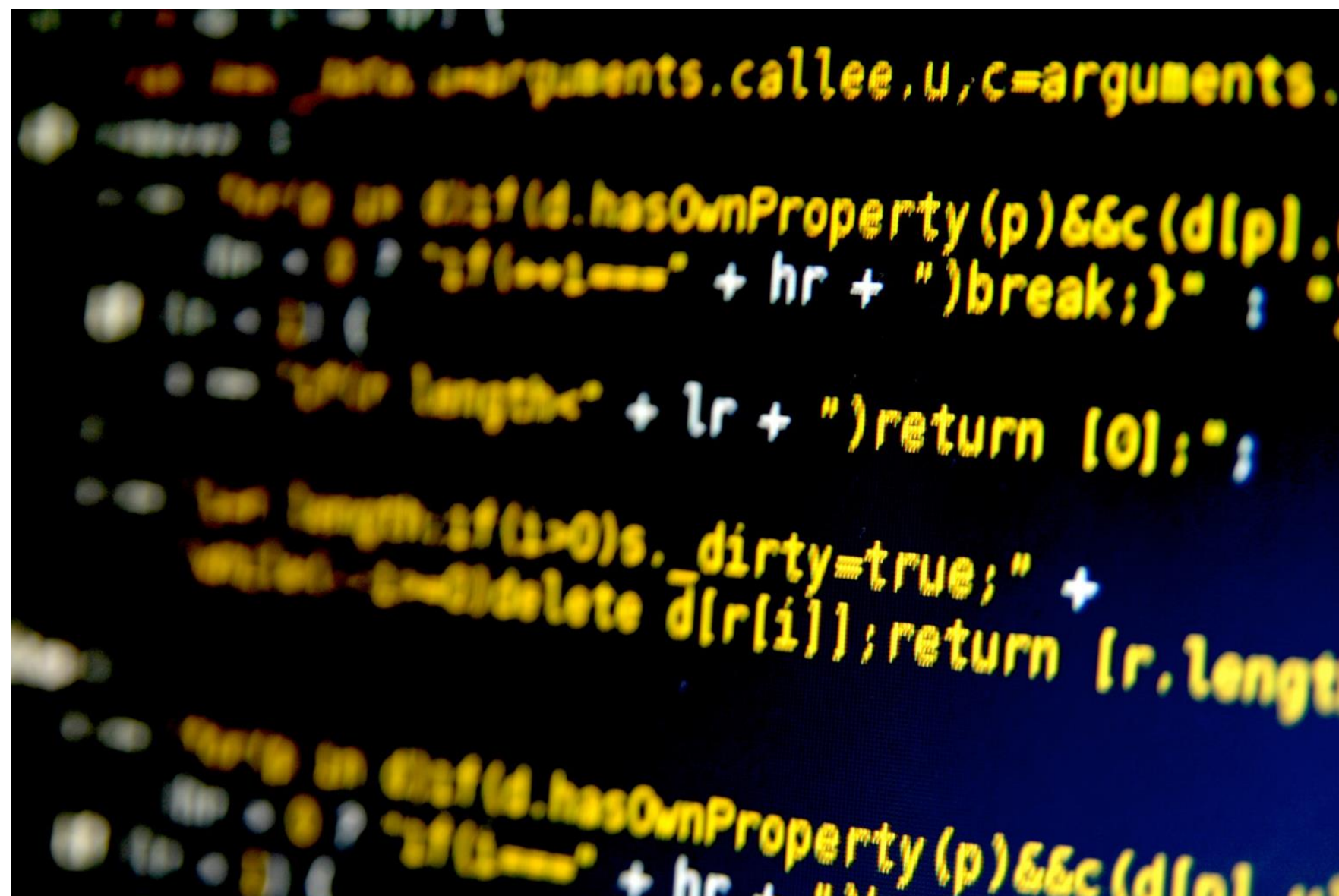
JavaScript

```
var links = $('a');  
  
links.each(function (index, element) {  
    console.log($(this).attr('href'));  
});
```



W jQuery działa również **this** reprezentujące bierzący
element ale musimy stworzyć z niego obiekt jQuery
poprzez **\$(this)**

Czas na zadania



Wykonajcie zadania z działu:

3. jQuery

Katalog

1. Wyszukiwanie elementów

Zaawansowane wyszukiwanie elementów

Metody do wyszukiwania elementów

- **find()** – znajduje elementy, które są zagnieżdżone w innym,
- **closest()** – znajduje elementy najbliższe o zadanej **klasie** lub **id**,
- **children()** – znajduje wszystkie dzieci danego elementu,
- **parent()** – znajduje rodzica elementu,
- **siblings()** – znajduje rodzeństwo elementu,
- **next()** – znajduje następny element,
- **prev()** – znajduje poprzedni element.

Czas na zadania



Wykonajcie zadania z działu:

3. jQuery

Katalog

2. Zaawansowane wyszukiwanie



Eventy

Eventy w jQuery

jQuery znacznie usprawniło korzystanie z eventów. Do sterowania eventami wystarczą trzy metody:

- **on(event, function)** – pozwala na dodanie callbacka do eventu,
- **one(event, function)** – pozwala na zapięcie nowego eventu, który zadziała tylko i wyłącznie raz, po czym zostanie automatycznie usunięty,
- **off(event, function)** – usuwa wszystkie callbacki, które były podpięte pod dany event (nawet te anonimowe).

- Nazwy eventów, które przeglądarka nam udostępnia, są w większości identyczne jak te wykorzystujące metodę **addEventListener**.

Propagacja eventów

Tak samo jak w czystym JavaScript eventy są propagowane. Jesteśmy w stanie zatrzymać propagację eventu, jeżeli użyjemy jednej z następujących metod:

- **stopPropagation()** – zatrzymuje propagację eventu w górę drzewa DOM (callbacki rodziców nie zostaną uruchomione).
- **stopImmediatePropagation()** – zatrzymuje propagację eventu oraz **każdy inny event**, który powinien zostać uruchomiony.

preventDefault()

Możemy zapobiec domyślnej akcji eventu np.

```
$('a').on('click', function() {
```

```
    // jakaś akcja po kliknięciu, np. przeniesienie
```

```
    // pod adres znajdujący się w href
```

```
});
```

```
$('a').on('click', function(event) {
```

```
    event.preventDefault();
```

```
    // jakaś akcja po kliknięciu, np. przeniesienie
```

```
    // pod adres znajdujący się w href zostanie anulowana
```

```
});
```

preventDefault() vs return false

Zapobiega domyślnej akcji eventu np.:

```
$('a').on('click', function(event) {  
    event.preventDefault();  
})
```

Zapobiega domyślnej akcji eventu
oraz zapobiega propagacji eventu w górę.

```
$('a').on('click', function() {  
    return false;  
})
```

Przetestuj zadanie:

<http://jsfiddle.net/CodersLab/cw7z5g9x>

on()

Metoda **on** zaczepia określony event do elementu jQuery.

\$(elements).on(events [, selector] [, data], handler)

Gdzie:

- **events** to typ eventu, może być jeden lub więcej,
- **selector** – to opcjonalny parametr, określa selektory, na których możemy zaczepić event, a których np. nie ma jeszcze w dokumencie,
- **data** – to również opcjonalny parametr. Możemy przekazać do funkcji handler jakieś dane np. `{foo: "bar"}`,
- **handler** – to funkcja, która zostanie wykonana w momencie wywołania eventu.

on()

1. Dla każdego elementu, który znajdziesz, ustaw event **click**.
2. Dla każdego elementu, który ma ustawiony event, zostaje przypisana funkcja anonimowa.
3. Funkcja ta zostanie wywołana dopiero wtedy, gdy event **click** zostanie wywołany.

```
$('.menu').find('li').on('click', function() {  
    // jakaś akcja po kliknięciu  
});
```

one()

Metoda **one** zaczepia określony event do elementu jQuery tylko raz.

`$(elements).one(events [, selector] [, data], handler);`

Metoda przyjmuje te same parametry co **on()**.

```
$('.menu').find('li').one('click', function() {  
    // jakaś akcja po kliknięciu  
});
```

off()

Metoda **off** odczepia określony event od elementu jQuery.

`$(elements).off([events] [, selector] [, handler]);`

- Wszystkie parametry są opcjonalne.
- Wywołanie samego `$(element).off()` usunie wszystkie eventy z elementu.

```
$('.menu').find('li').on('click', function() {  
    // jakaś akcja po kliknięciu  
});
```

```
$('.menu').find('li').off('click');
```

Metoda one – przykład

`<button id="ourButton">Click me!</button>`

`$('#ourButton').one('click', function (event) {
 alert('You clicked me!');
});`

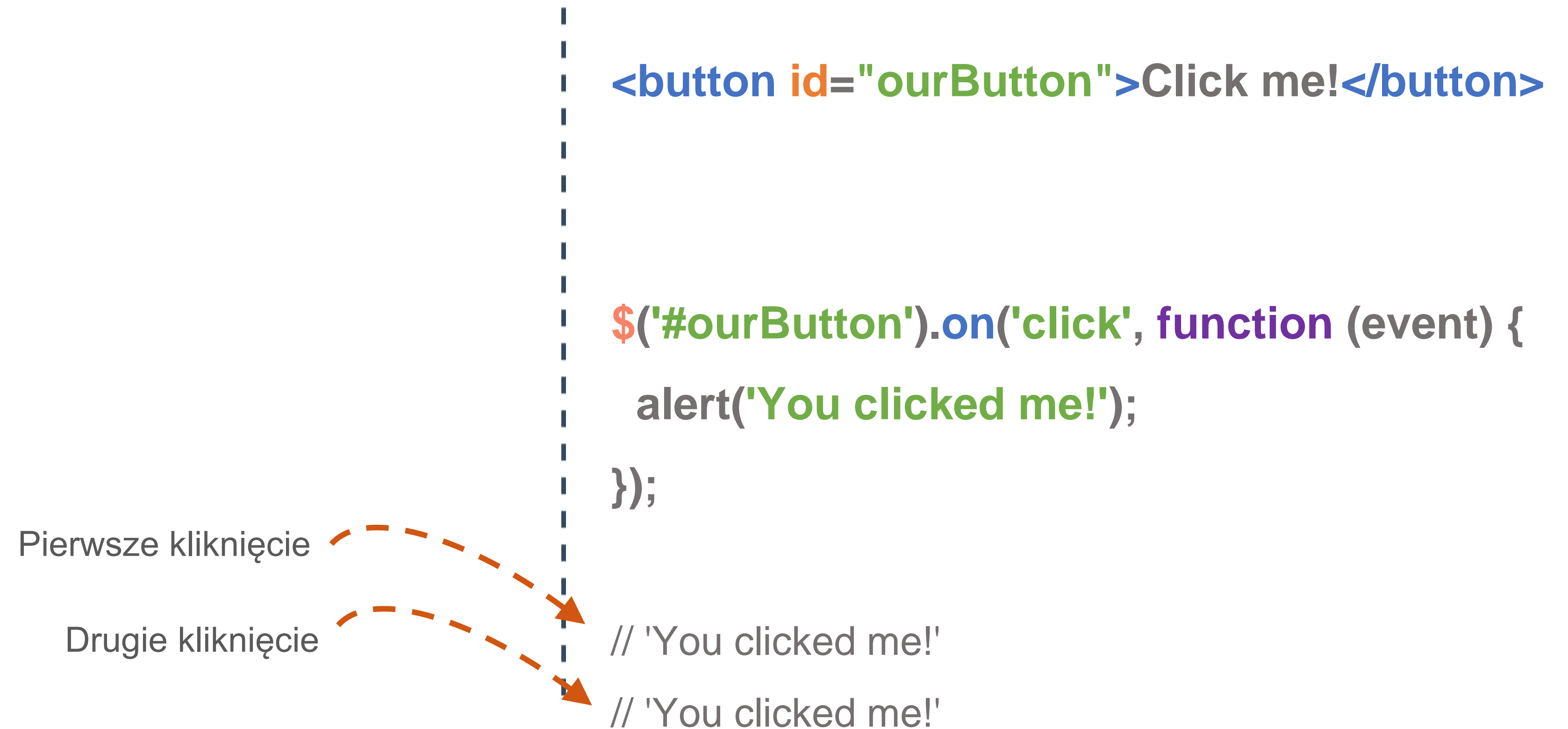
Pierwsze kliknięcie

Drugie kliknięcie

// 'You clicked me!'

// Nic się nie wyświetla, ponieważ użyliśmy metody **one**

Metoda on – przykład



Metoda off – przykład

```
<button id="ourButton">Click me!</button>
```

```
$('#ourButton').on('click', function (event) {  
    alert('You clicked me!');  
});
```

Pierwsze kliknięcie



```
$('#ourButton').off('click');
```

```
// Nic się nie wyświetla, ponieważ odpięliśmy event.
```

Najpopularniejsze eventy

Mouse Events

- **click** – kliknięcie
- **dblclick** – podwójne kliknięcie
- **mouseenter** – najechanie
- **mouseleave** – zjechanie

KeyBoard Events

- **keypress** – wciśnięty klawisz (klawisze specjalne)
- **keydown** – wciśnięty klawisz
- **keyup** – zwolniony klawisz

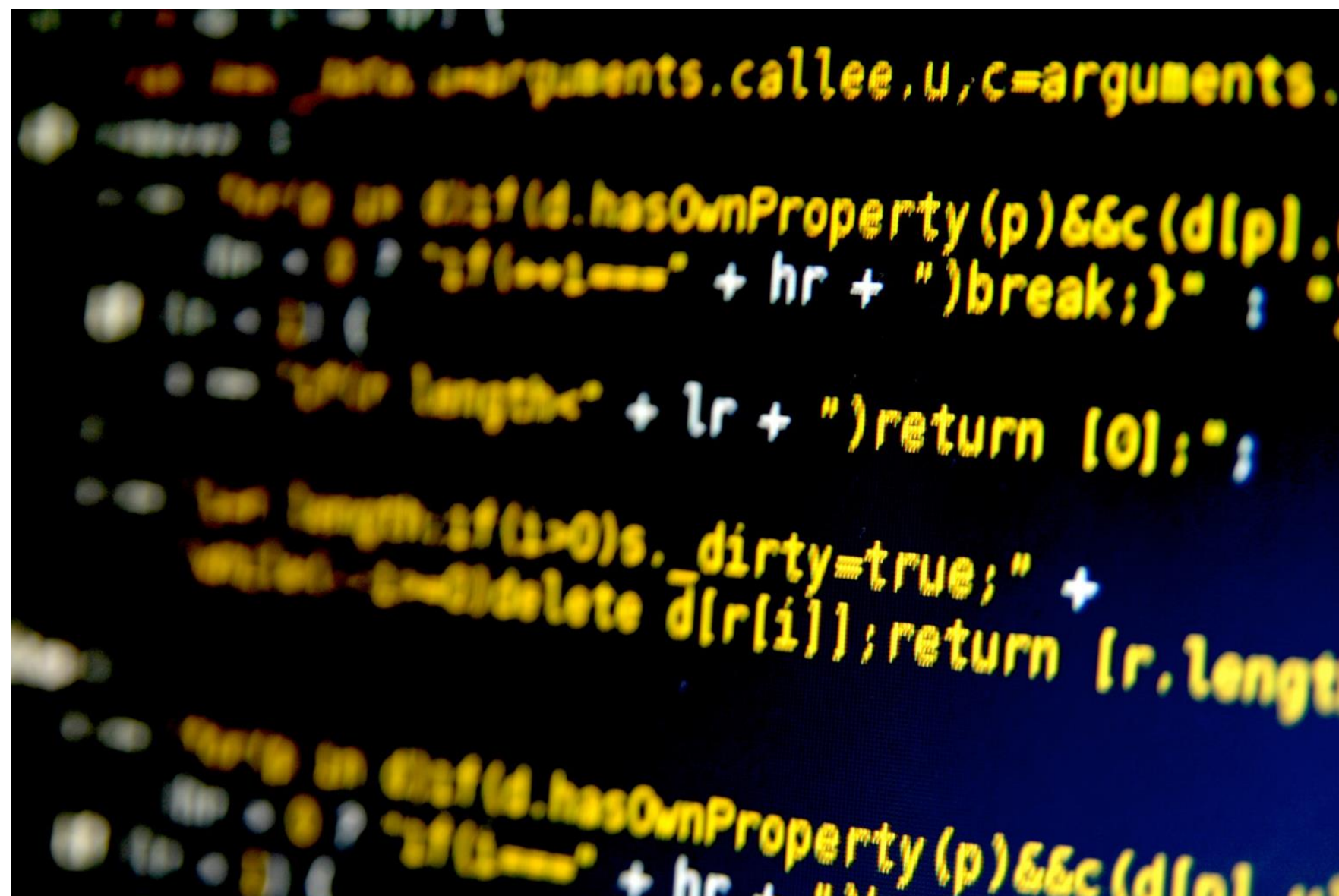
Form Events

- **submit** – kliknięty submit
- **change** – zmiana elementu
- **focus** – focus na elemencie
- **blur** – utrata eventu focus

Document/Window Events

- **load** – ładowanie dokumentu
- **resize** – zmiana wielkości okna
- **unload** – event po opuszczeniu przez użytkownika strony (blokowany przez niektóre przeglądarki, możesz użyć `onbeforeunload`)
- **scroll** – scrollowanie

Czas na zadania



Wykonajcie zadania z działu:

3. jQuery

Katalog

3. Eventy

Manipulacja DOM-em

Atrybuty i własności

- **attributes** (atrybuty) – występują w tekstowym dokumencie HTML,
- **properties** (własności) – są dostępne tylko przez JavaScript.

attributes

```
<!DOCTYPE html>
<html itemscope itemtype="http://schema.org/WebPage" lang="pl">
  <head>
    <meta content="/images/branding/googleg/1x/googleg_standard_color_128dp.png" itemprop="image">
    <link href="/images/branding/product/ico/googleg_lodp.ico" rel="shortcut icon">
    <meta content="origin" id="mref" name="referrer">
    <title>Google</title>
```

properties

 `console.log(document.body)`

```
<body class="hp vasq" onload="try{if(!google.j.b)
{document.f&&document.f.q.focus();document.gbqf&&document.gbqf.q.focus();}
}catch(e){}if(document.images)new Image().src='/images/nav_logo242.png'"
id="gsr">
  <div class="ctr-p" id="viewport">...</div>
  <script src="/xjs/_/js/k=xjs.s.pl.U3zptU63adU.O/
m=sy30,sy36,em3,em2,sy38,em0,sy268,...73wQCBfCESE1UMHi/rt=j/d=0/t=zcms/
rs=ACT90oGA1JdCoN0o_6rhzWu4gLMTU_AeLg" gapi_processed="true"></script>
  <iframe src="https://clients5.google.com/pagead/drt/dn/" aria-hidden=
"true" style="display: none;">...</iframe>
  <div class="jfk-bubble chw-oc" role="alertdialog" aria-describedby=
"bubble-5" style="display: none;">...</div>
</body>
```


Atrybuty

Możemy łatwo wczytywać lub modyfikować atrybuty elementów dzięki następującym metodom:

- **attr(name, newValue)** – pobierz lub ustaw atrybut elementu,
- **removeAttr(name)** – usuń atrybut elementu,

Każda z tych metod może nastawiać atrybut (jeżeli podamy nową wartość) **attr(name)** lub zwracać jego wartość (jeżeli podamy tylko nazwę).

HTML

```
<div class='footer' id='plan'>O nas</div>
```

JavaScript

```
var elementID = $('.footer').attr('id');
```

id='plan'



JavaScript

```
var elementID = $('.footer').removeAttr('id');
```

HTML

```
<div class='footer'>O nas</div>
```

Metoda prop()

- **prop** – również sprawdza własności (properties) elementu.
- Jest używany podczas pobierania atrybutów boolean oraz własności nieistniejących w dokumencie HTML.
- Wszystkie inne atrybuty powinno się pobierać za pomocą **attr()**.

Przykład zastosowania tej metody:

- <http://jsfiddle.net/bipen/54nLM/>

HTML

```
<input type="checkbox" value="test"
id="test"/>
```

JavaScript

```
$('#test').prop('checked');
```


Atrybut data

Możemy nastawiać lub odczytywać atrybut data za pomocą metody **data(dataSet, value)**.

```
<div data-role="page" data-last-value="43"  
data-hidden="true">  
</div>
```

```
$("div").data("role") ; //"page"
```

```
$("div").data("lastValue"); //43
```

```
$("div").data("hidden"); // true
```

```
$("div").data("options", "new option" );
```

Pobieranie i wstawianie tekstu do elementu

- **html()** – wstawia/ustawia tekst lub HTML (zrenderowany)
- **text()** – wstawia/ustawia tekst lub HTML (jako string, np. `Tekst`)

Zobacz różnicę:

- <http://jsfiddle.net/hossain/sUTVg/>

Modyfikowanie elementów

Tworzenie nowych elementów

- Tworzenie nowych elementów w jQuery staje się bardzo proste.
- Wystarczy, że string z kodem HTML danego elementu otoczymy `$()` i dostaniemy wrapper z tym elementem.
- Musimy pamiętać, że tak jak w przypadku czystego JavaScript, element ten nie będzie jeszcze podczepiony do DOM-u.

```
var newDiv = $("

");  
//<div></div>


```

```
var newDiv = $("

Tekst, który  
wyświetlamy</div>");  
// <div>Text który wyświetlamy</div>


```

```
var newDiv = $("id='newDiv'>");  
//<div class="class1 foo" id="newDiv"></div>
```

```
var newDiv = $("

", {id: "myId", class:  
"class1 class2"});  
//<div id="myId" class="class1 class2"></div>


```

Dodawanie elementów do DOM-u

- Tak samo jak w przypadku czystego JavaScript po utworzeniu elementu należy go jeszcze podpiąć do DOM-u.
- jQuery udostępnia nam bardzo dużo metod, dzięki którym możemy łatwo podpiąć element w wybranym miejscu.

Są to np.:

- **after,**
- **before,**
- **append,**
- **appendTo,**
- **prepend,**
- **prependTo,**
- **insertAfter,**
- **insertBefore,**
- **wrap.**

Wstawianie elementu przed lub po

- Oto metody, które służą do wstawiania elementów bezpośrednio przed wybranym elementem lub po nim:
 - **before()**
 - **after()**.

```
// <p class="bar">Hello</p>
```

```
var firstOfBar = $(".bar").first();
```

```
var newElement = $("<div class='new'> This is new  
                        element</div>");
```

```
// <p class="bar">Hello</p>
```

```
// <div class="new"> This is new element</div>
```

```
firstOfBar.after(newElement);
```

```
// <div class="new"> This is new element</div>
```

```
// <p class="bar">Hello</p>
```

```
firstOfBar.before(newElement);
```

Wstawianie elementu przed lub po

- Dużo metod w jQuery ma swoje lustrzane odpowiedniki.
- Odpowiedniki **before()** i **after()** są następujące:
 - **insertBefore()**
 - **insertAfter()**

```
// <p class="bar">Hello</p>
```

```
var firstOfBar = $(".bar").first();
```

```
var newElement = $("<div class='new'> This is new  
element</div>");
```

```
// <p class="bar">Hello</p>
```

```
// <div class="new"> This is new element</div>
```

```
newElement.insertAfter(firstOfBar);
```

```
// <div class="new"> This is new element</div>
```

```
// <p class="bar">Hello</p>
```

```
newElement.insertBefore(firstOfBar);
```

after() vs insertAfter()

Różnica polega na tym, że funkcja **insertAfter()** zwraca wszystkie znalezione elementy, natomiast funkcja **after** – nie zwraca nic.

//Obie linie zrobią to samo:

```
$("#p").insertAfter("#foo");
```

```
$("#foo").after("p");
```


Dodawanie elementu do dzieci

Możemy łatwo dodać element do dzieci innego elementu dzięki następującym metodom:

- **append(newElement)** – wstaw nowy element na koniec dzieci już istniejącego elementu,
- **appendTo(oldElement)** – odwrotność (czyli wywołujemy na nowym elemencie i podajemy, gdzie ma się dodać).

- **prepend(newElement)** – wstaw nowy element na początek dzieci już istniejącego elementu,
- **prependTo(oldElement)** – odwrotność.

append vs prepend

Kod HTML

```
<div class="foo" id="foold" style="color: red;">  
  <p class="bar">Hello</p>  
</div>
```

Kod JavaScript

```
var newElement = $("<div class='new'> This is new  
                  element</div>");  
var foo = $("#foold");  
  
foo.append(newElement);  
newElement.appendTo(foo); // obie linie dadzą ten sam wynik
```

Wynik w przeglądarce

```
<div class="foo" id="foold" style="color: red;">  
  <p class="bar">Hello</p>  
  <div class="new"> This is new element</div>  
</div>
```

Kod HTML

```
<div class="foo" id="foold" style="color: red;">  
  <p class="bar">Hello</p>  
</div>
```

Kod JavaScript

```
var newElement = $("<div class='new'> This is new  
                  element</div>");  
var foo = $("#foold");  
  
foo.prepend(newElement);  
newElement.prependTo(foo); // obie linie dadzą ten sam wynik
```

Wynik w przeglądarce

```
<div class="foo" id="foold" style="color: red;">  
  <div class="new"> This is new element</div>  
  <p class="bar">Hello</p>  
</div>
```

Usuwanie elementów z DOM-u

Oto kilka metod ułatwiających usunięcie elementów z DOM-u. Co ważne, jedna z metod pozwala nam odczepić element bez jego niszczenia.

- **remove()** – usuń element,
- **detach()** – wypnij element z drzewa DOM bez usuwania go i zwróć go (np. żebyśmy mogli zapisać go do zmiennej),
- **empty()** – usuń wszystko ze środka elementu

```
// <div class="foo" id="foold" style="color: red;">
//     <p class="bar1">Hello1</p>
//     <p class="bar2">Hello2</p>
//     <p class="bar3">Hello3</p>
// </div>
```

```
$(".bar1").remove();
```

```
// <div class="foo" id="foold" style="color: red;">
//     <p class="bar2">Hello2</p>
//     <p class="bar3">Hello3</p>
// </div>
```

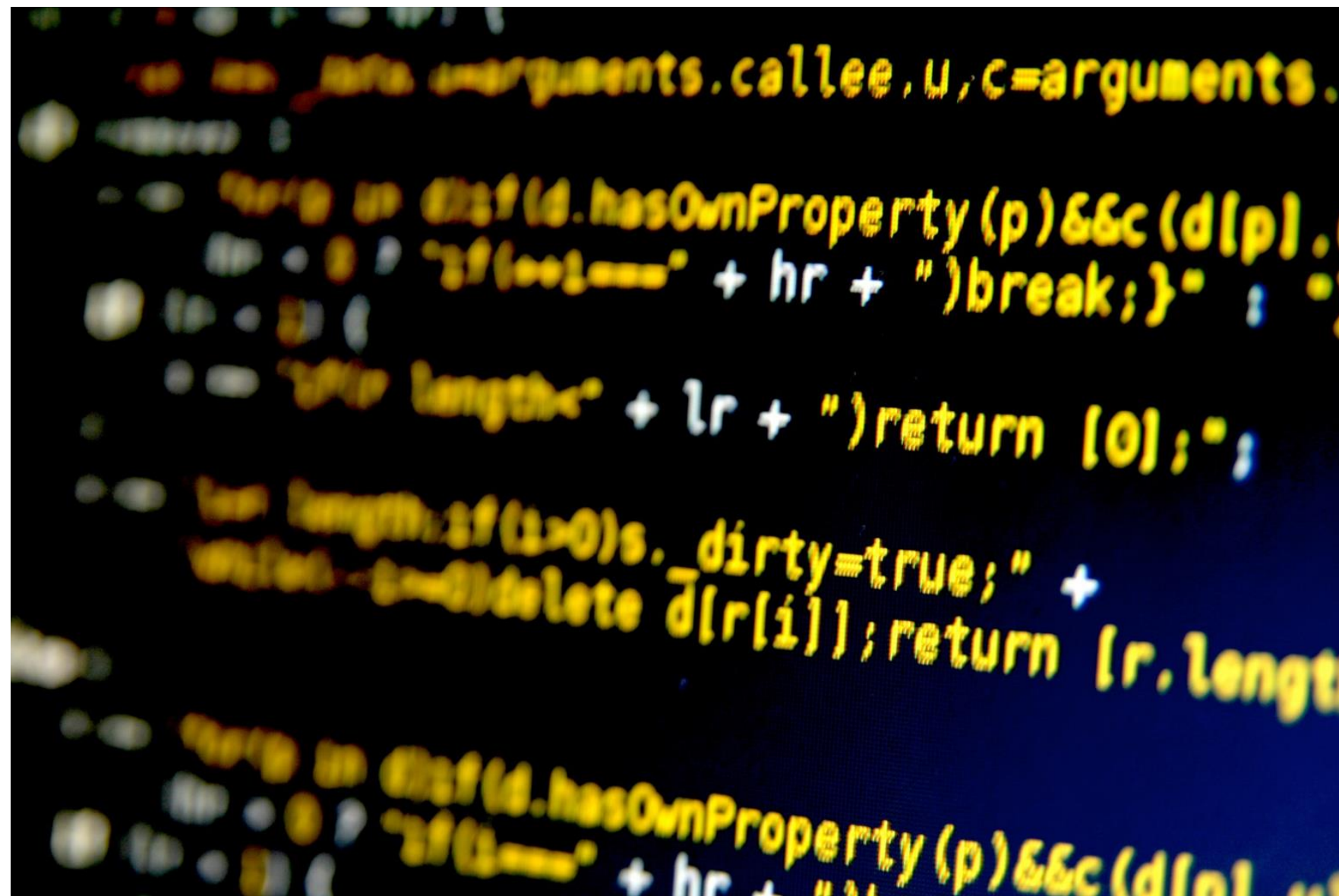
```
var removedBar2 = $(".bar2").detach();
```

```
// możemy zapisać do zmiennej
// <div class="foo" id="foold" style="color: red;">
//     <p class="bar3">Hello3</p>
// </div>
```

```
$("#foold").empty();
```

```
// <div class="foo" id="foold" style="color: red;"></div>
```

Czas na zadania



Wykonajcie zadania z działu:

3. jQuery

Katalog

4. Modyfikowanie elementów