



Python. Подготовка к собеседованию

## Урок 5

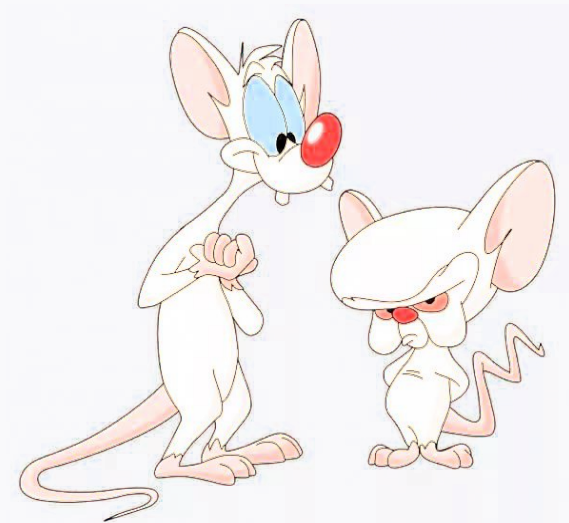
# Django - AJAX, JavaScript, jQuery

Особенности использования фреймворка в привязке к технологии AJAX, языку программирования JavaScript и библиотеки jQuery.

# Цели урока

Повторить принципы реализации технологии AJAX, а также использования JavaScript и библиотеки jQuery в Django-проектах.

- Что означает каждое из понятий: AJAX, JavaScript, jQuery?
- Преимущества jQuery, преимущества и недостатки AJAX.
- Подключение jQuery и JavaScript к шаблону Django-проекта.
- Селекторы в jQuery, основные конструкции.
- Выполнение AJAX-запроса с помощью jQuery.
- Привязка обработчиков к событиям в JavaScript.
- Объектная модель документа, отслеживание ошибок.

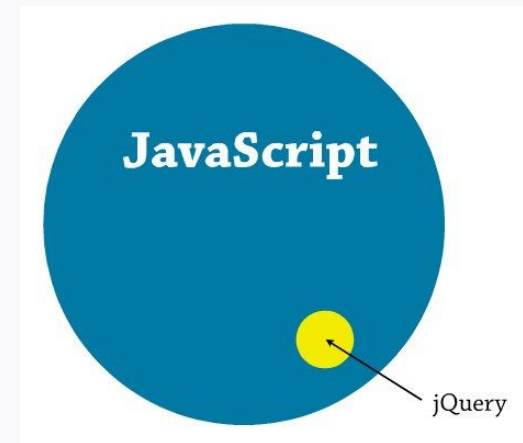


# AJAX, JavaScript, jQuery



# Преимущества jQuery

- Простота освоения по сравнению с чистым JavaScript, упрощение работы с объектной моделью документа.
- Упрощение реализации проектов на базе технологии AJAX.
- Решение проблем кроссбраузерной совместимости приложений.
- Наличие мощной библиотеки плагинов.
- Простота применения, обширная документация, развитое интернет-сообщество.



# Преимущества и недостатки AJAX



Экономия трафика.  
Снижение нагрузки на сервер.  
Ускорение ответа интерфейса.  
Широкие возможности для интерактивной обработки.



Требование наличия включенного JavaScript.  
Отсутствие взаимосвязи со стандартными возможностями браузера.  
Проблема фиксации просмотра новых страниц.  
Сложность отображения нестандартных кодировок.  
Замедление обновления страницы.



# Подключение jQuery и JavaScript к шаблону Django-проекта

Подключаем статику

```
{% load staticfiles %}
```

```
{% block additional_js %}
```

Подключаем jQuery-библиотеку

```
<script src="{% static 'js/jquery.js' %}" type="text/javascript"></script>
```

Подключаем JavaScript-сценарий

```
<script src="{% static 'js/script.js' %}" type="text/javascript" %}"></script>
```

```
{% endblock %}
```



# Выполнение кода с помощью jQuery

```
$(document).ready(function() {  
    // Код обработчика для .ready()  
});
```

```
$(document).ready(function() {  
    // Код обработчика для .ready()  
});
```



# Селекторы в jQuery

**#id** – получаем единственный элемент веб-страницы по его идентификатору.

**element** – получаем доступ ко всем элементам с данным именем.

**.class** – получаем доступ ко всем элементам данного класса.


**.class.class** – получаем доступ ко всем элементам перечисленных классов.

**\*** – получаем доступ ко всем элементам.





# Конструкции window.onload, document.ready(), \$(function(){...})

<code>\$(document).ready()</code>	<code>window.onload</code>
<ul style="list-style-type: none"><li>• It's a jQuery method</li><li>• check DOM elements if all loaded</li></ul> <p><code>&lt;html&gt;</code> <code>&lt;/html&gt;</code></p>	<ul style="list-style-type: none"><li>• It's a pure javascript method</li><li>• check everything including DOM elements and images loaded</li></ul>  <p><code>&lt;html&gt;</code> <code>&lt;/html&gt;</code></p>



# AJAX-запросы в jQuery

**load()**

**\$.getJSON()**

**\$.getScript()**

**\$.get()**

**\$.post()**

**\$.ajax()**



# Реализация события нажатия кнопки в jQuery

```
$(document).ready(function () {  
    // Обработчик нажатия на кнопку  
    $('button').on('click', function() {  
        alert('Кнопка нажата!');  
    })  
    // Имитация нажатия на кнопку  
    $('button').trigger('click');  
});
```



# Проверка существования элемента средствами jQuery

```
$(document).ready(function () {  
    if ($('.letter').length > 0){  
        alert('Тег span существует');  
    };  
});
```



# Выполнение только одного запуска обработчика в jQuery

```
$(document).ready(function () {  
    $('#test_p').one('click', function() {  
        alert('Выводим сообщение только один раз');  
    });  
});
```



# Способы привязки обработчиков к событиям в JavaScript

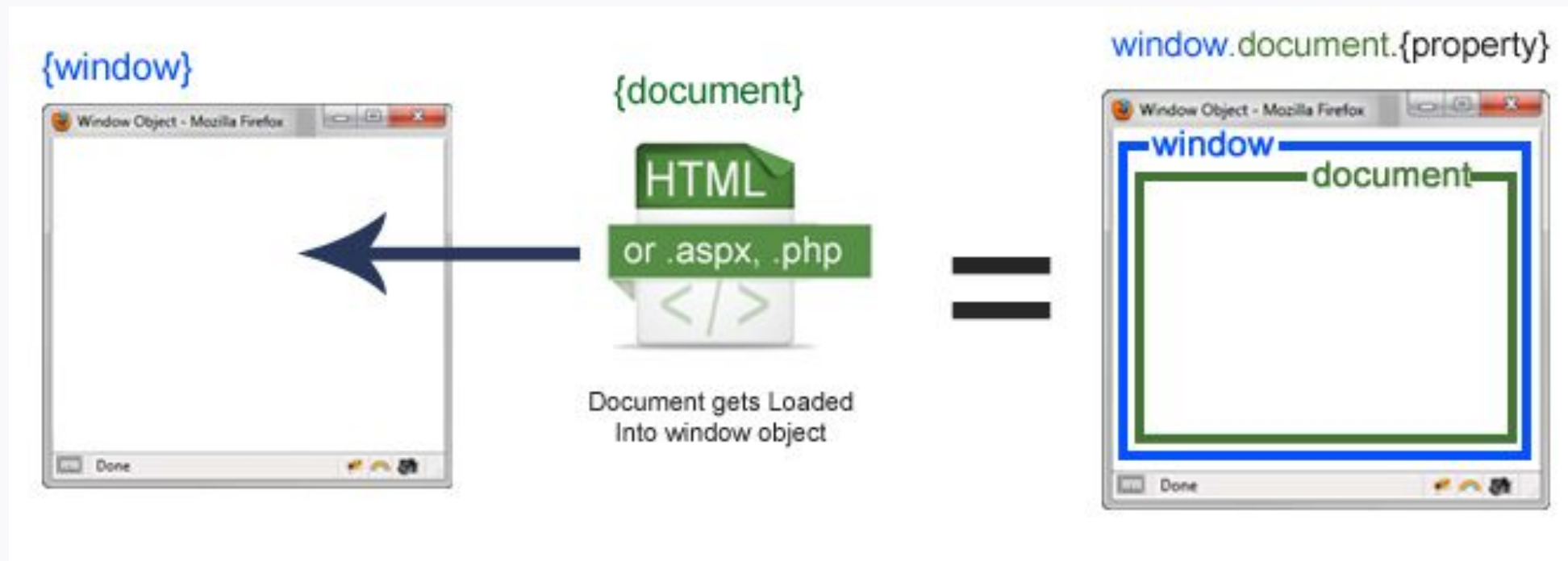
```
target.onclick = function(event) { код обработчика }
```

```
target.addEventListener("click", ...)
```

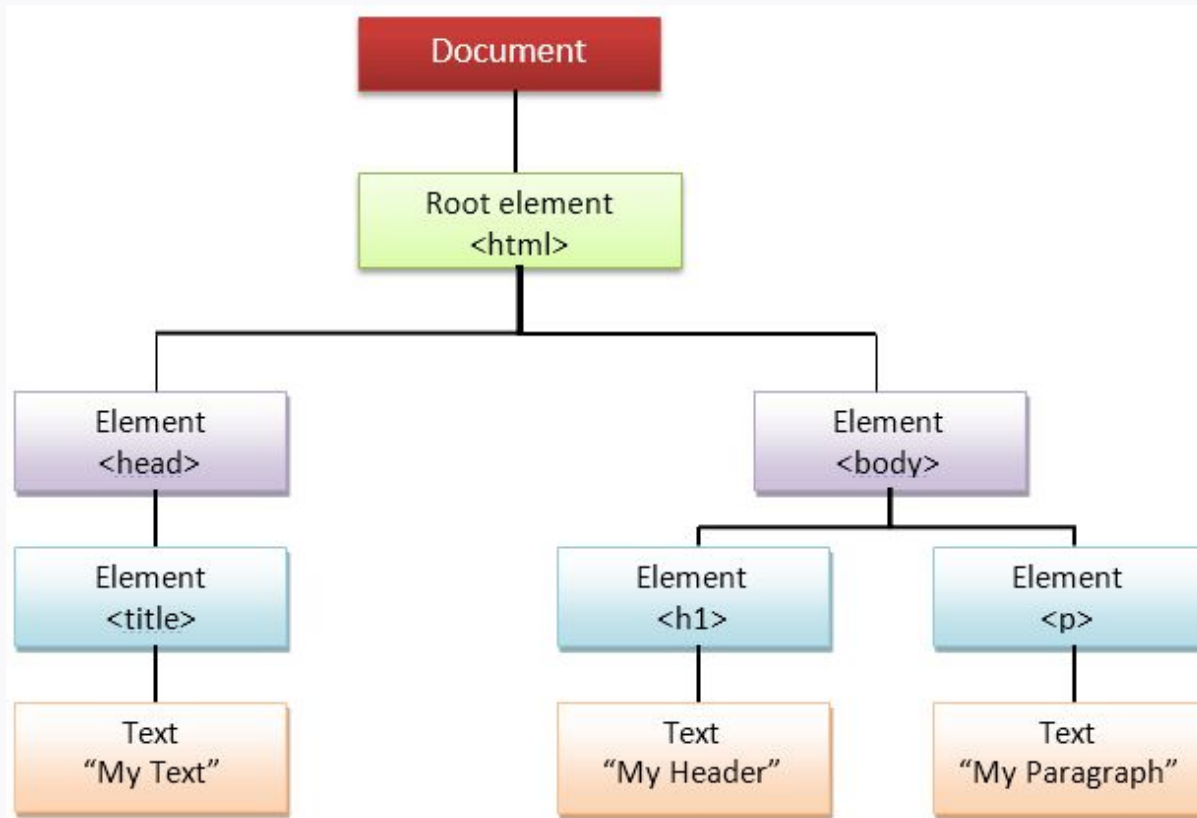
```
target.attachEvent("on"+имя_события, обработчик)
```



# Объекты window и document



# Доступ к элементу из DOM



`getElementById`

`getElementsByClassName`

`getElementsByTagName`

`querySelector`

`querySelectorAll`

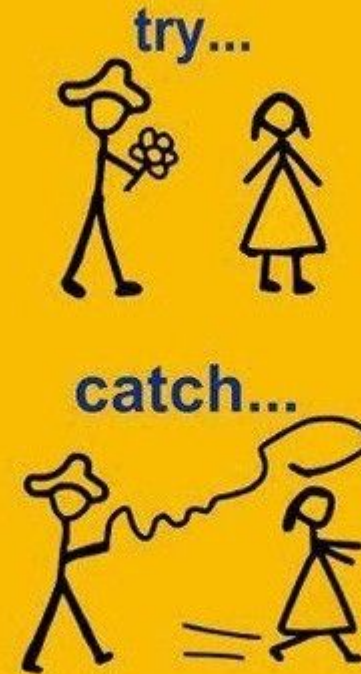
`getElementsByName`





# Отслеживание ошибок в JavaScript

```
try
{
    //do anything
}
catch(error)
{
    //don't forget errors
}
```



# Свойства innerhtml и outerhtml

.innerHTML — *the inside*

```
<div>  
  <p>Foo</p>  
</div>
```

.outerHTML — *the outside*

```
<div>  
  <p>Foo</p>  
</div>
```



# Практическое задание



# Практическое задание

Продолжим работать над каталогом товаров. На данный момент приложение действует в синхронном режиме, т.е. пользователь нажимает кнопку добавления, после чего выполняется переход на соответствующую страницу.

Усовершенствуем работу проекта: сделаем его более похожим на десктопное приложение — без всяких перезагрузок страницы. Форма должна загружаться, как модальное окно, как бы «поверх» содержимого главной страницы. После этого пользователь может указать данные нового товара и сохранить его в базе. При этом модальное окно формы должно закрываться, а изменения, т.е. строка с добавленным товаром, должна отобразиться в таблице на главной странице. Таким образом мы реализуем асинхронную работу приложения.



# Дополнительные материалы

1. <http://qaru.site/questions/101999/what-is-the-difference-between-ajax-with-javascript-and-jquery>.
2. <http://www.cyberforum.ru/javascript-jquery/thread1060896.html>.
3. <https://github.com/h5bp/Front-end-Developer-Interview-Questions/tree/master/Translations/Russian#js>.
4. <http://www.tangowithdjango.com/book17/chapters/jquery.html>.
5. <https://jsfiddle.net/hmq53pqk/>.

