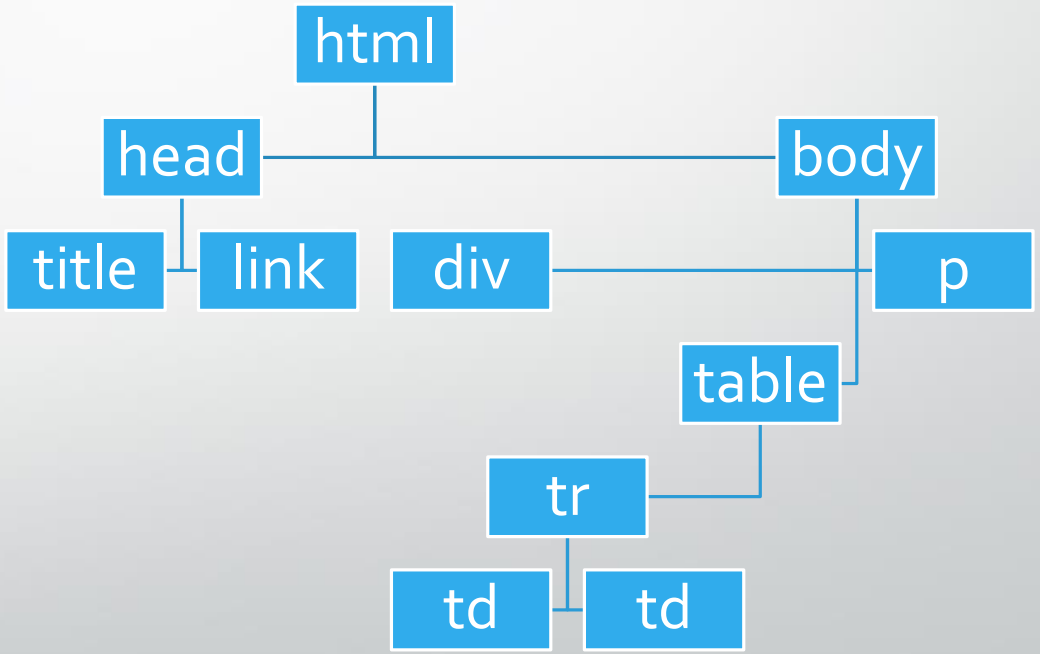
IT-Academy.BG

DOM, CSS, DOM & CSS манипулации

Димитър Митев

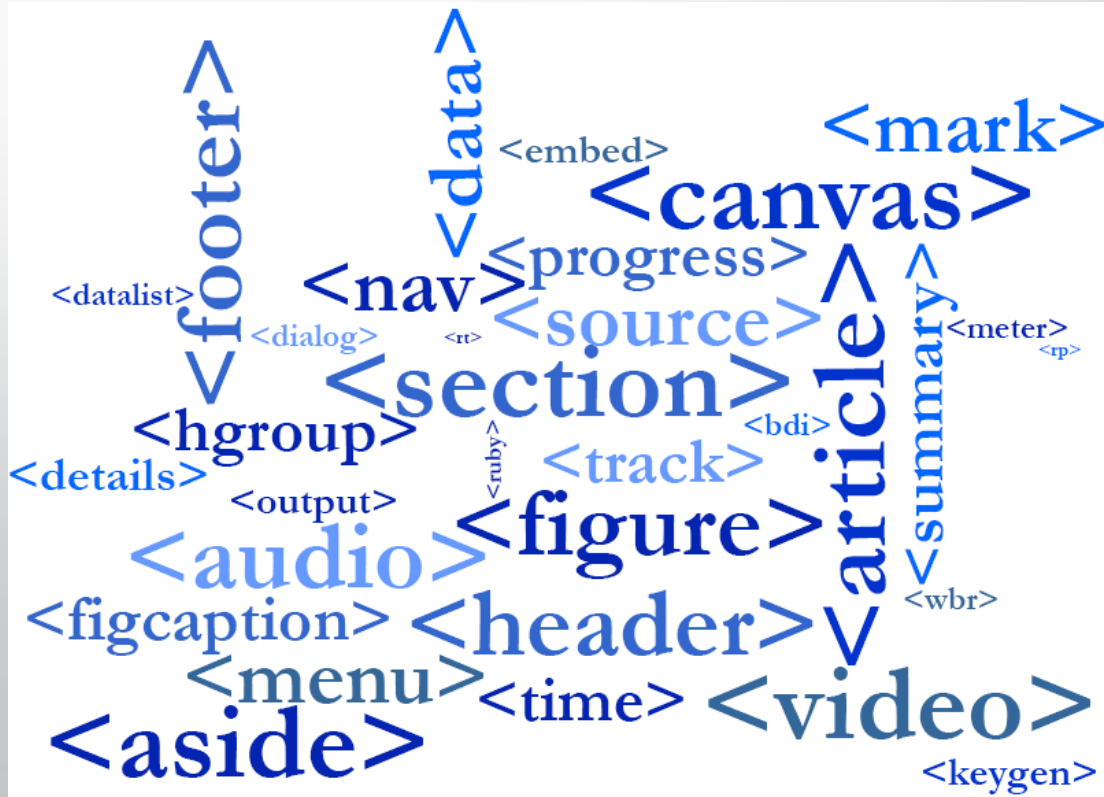
Какво е DOM

- DOM /document object model/ представлява дървовидна структура от елементите, които описват нашето съдържание /markup/
- Чрез DOM може да бъде достъпвано и променяно както съдържанието така и UI-а
- DOM е съвкупност от обекти,с които можем да манипулираме web страница
- Можем да използваме JS, за да достъпим DOM обекта и да работим с него



DOM обекти

- Всеки HTML елемент има кореспондиращ DOM обект
  - *HTMLLIElement* represents *<li>*
  - *HTMLAudioElement* represents *<audio>*
- Всеки обект има съответните характеристики
  - *HTMLAnchorElement* има *href* характеристика
  - *HTMLImageElement* има *src* характеристика
- Обектът *document* е специалиен обект
  - Идва от браузъра
  - Входящата точка към DOM API



HTML елементи

- HTML елементите имат характеристики, които отговарят на техните атрибути
  - Напр. *id*, *className*, *draggable*, *style*, *onclick* ….
- Различни HTML елементи имат специфични за тях характерситики
  - *HTMLImageElement* има характеристика *src*
  - *HTMLInputElement* има характеристика *value*
  - *HTMLAnchorElement* има характерситика *href*

HTML елементи

- HTML елементите имат характеристики, които позволяват да достъпваме тяхното съдържание
  - *innerHTML* -> връща като *string* съдържанието на елемента без самия елемент
  - *outerHTML* -> връща като *string* както самия елемент, така и неговото съдържание
  - *innerText*/*textContent* -> връща като *string* текстовото съдържанние на елемента

Селектиране на HTML елементи

- HTML елементите могат да бъдат намирани и достъпвани посредством DOM API
  - Селектиране на конкретен елемент
  - Селектиране на колекция от елементи
  - Селектиране посредством предефинирани колекции

```
var header = document.getElementById('header');
var nav = document.querySelector('#main-nav');

var inputs = document.getElementsByTagName('li');
var radiosGroup = document.getElementsByName('genders[]');
var header = document.querySelectorAll('#main-nav li');

var links = document.links;
var forms = document.forms;
```

Използване на *getElementsByXXX()* за селектиране на елементи

- *getElementById(id)* -> връща 1 елемент или *null*
  - *var header = document.getElementById('header');*
- *getElementsByClassName(className)* -> връща колекция /масив/ от елементи или *null*
  - *var sales = document.getElementsByClassName('sale-item');*
- *getElementsByTagName(tagName)* -> връща колекция /масив/ от елементи или *null*
  - *var divs= document.getElementsByTagName('div');*
- *getElementsByName(name)* -> връща колекция /масив/ от елементи или *null*
  - *var astrologyGroup= document.getElementsByName('astrology[]');*

Използване на *querySelector()* метода

- DOM API позволява да бъдат използвани CSS селектори, за да бъдат намирани елементи
  - *querySelector(selector)* връща 1вия елемент, който отговаря на дадения селектор
  - *querySelectorAll(selector)* връща колекция от елементи, които отговарят на дадения селектор
  - Методите получават *string* като параметър, който представлява CSS селектор

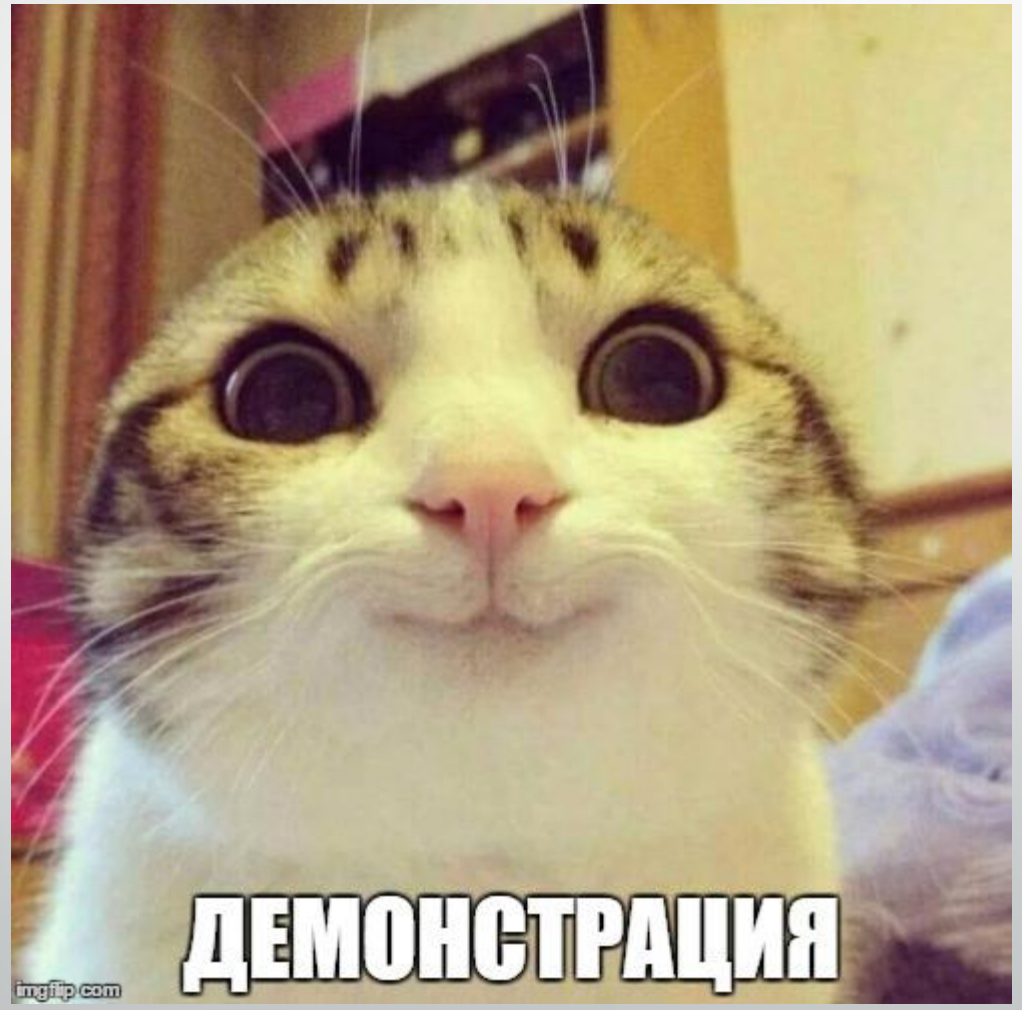
```
//the element with id="header"
var header = document.querySelector('#header');
//li elements contained in element with id=main-nav
var navItems = document.querySelectorAll('#main-nav li');
```

Селектиране на вложени HTML елементи

- Чрез DOM API може да се селектират и вложени HTML елементи
  - Всички методи могат да бъдат използвани върху вече селектирани HTML елементи

```
//the element with id="header"
var header = document.querySelector('#header');
//li elements contained in element with id=main-nav
var navItems = document.querySelectorAll('#main-nav li');
```

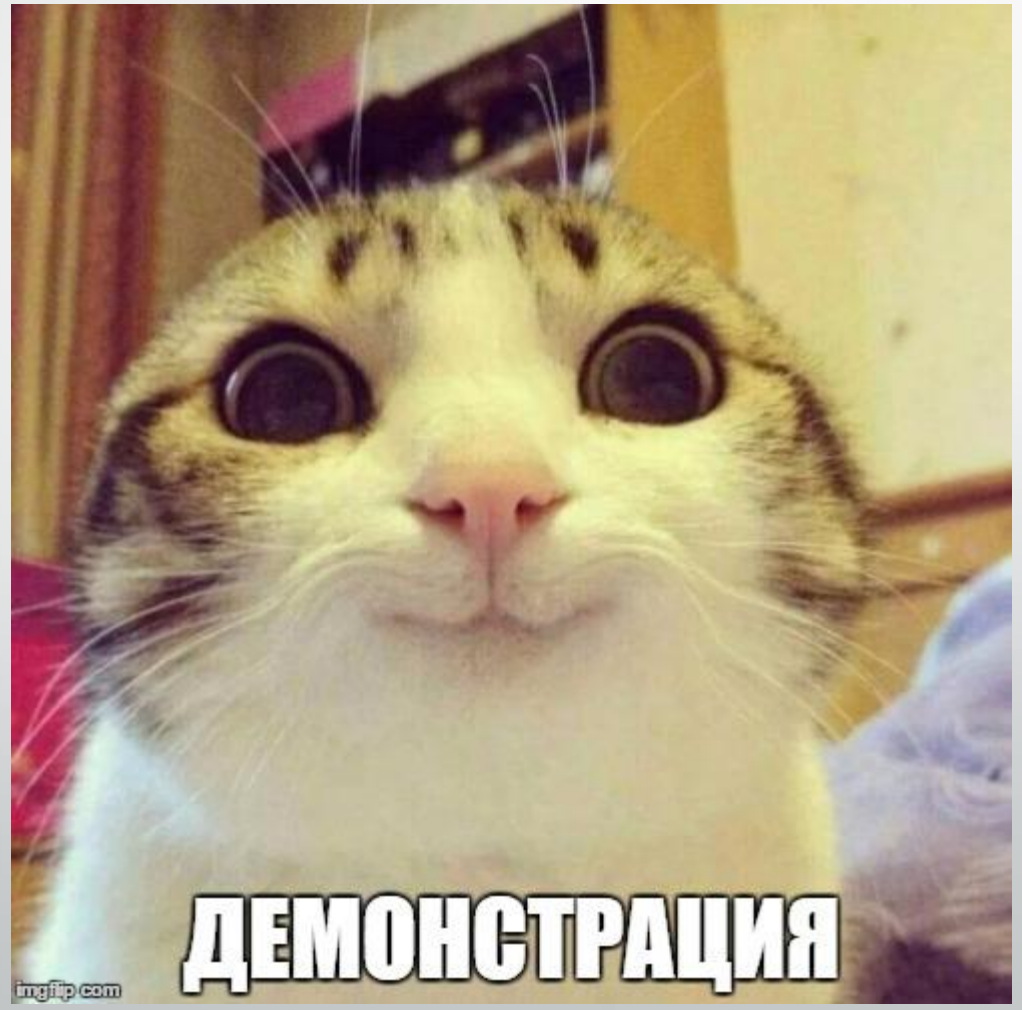
Селектиране на HTML елементи



NodeLists. Статичен NodeList и динамичен NodeList

- NodeList е типа на колекцията, която селекторите на HTML елементите връщат
  - Привидно изглежда масив и има характеристики на масив, в но в действителност е обект, който прилича на масив
    - *Може да обходите резултата с for-in цикъл и ще видите,че не се държи точно като масив*
- Има 2 вида NodeList
  - *getElementsByXXX()* връща т.нар. LiveNodeList
    - LiveNodeList -> следи промените и след изпълнението на метода
    - Значително по-бавен
    - Добре е да се кешира дължината му след изпълнението на метода, за да не се получат аномалии
  - querySelectorAll() връща т.нар. StaticNodeList
    - StaticNodeList -> не следи промените след изпълнението на метода

NodeLists. Статичен NodeList и динамичен NodeList



IT-Academy.BG

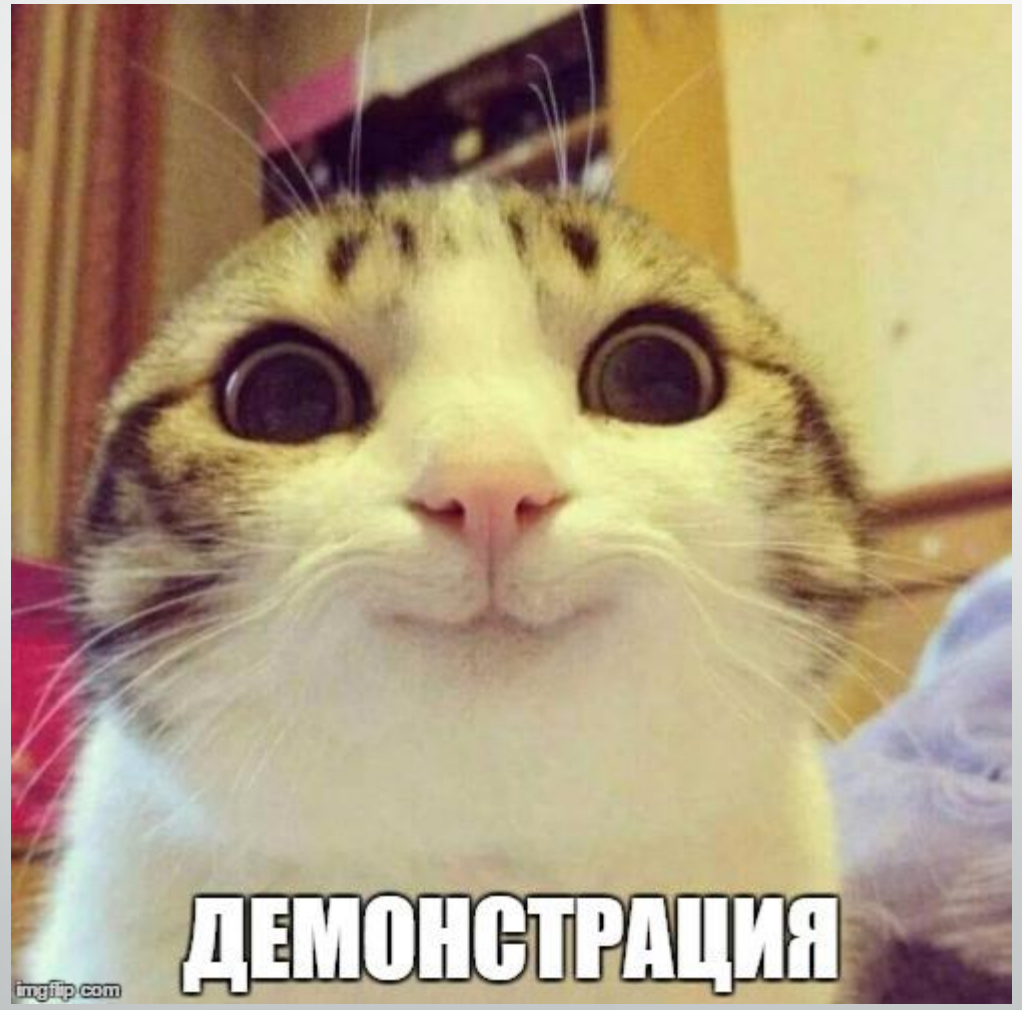
Обхождане на DOM дървото

- DOM елементите имат характеристики относно това къде се намират в DOM дървото
  - Техния родител /parent/
  - Техните наследници /children/
  - Техните съседни /siblings/
    - Точно преди или след елемента
  - Посредством тези характеристики можем да обхождаме DOM дървото

Обхождане на DOM дървото

- *element.parentNode*
  - Връща родителя на element
  - Родителя на *document* е *null*
- *element.childNodes*
  - Връща *LiveNodeList* от елементите вложени в element
- *element.firstChild/lastChild / element.firstElementChild/lastElementChild*
  - Връща първото/последното дете/дете-елемент на element
- *element.firstElementChild/lastElementChild*
  - Връща първото/последното дете-елемент на element
- *element.nextSibling/previousSibling / element.nextElementSibling/previousElementSibling*
  - Връща следващия/предишния елемент

Обхождане на DOM дървото



IT-Academy.BG

Манипулации върху DOM дървото

- DOM дървото може да бъде динамично манипулирано посредсвом JS
  - HTML елементи могат да бъдат създавани
  - HTML елементи могат да бъдат изтривани
  - HTML елементи могат да бъдат променяни
    - Може да се променя съдържанието им
    - Стиловете им
    - Атрибутите им

Създаване на DOM елементи

- Обектът *document* има предефиниран метод, чрез който може да се създават HTML елементи
  - document.createElement(elementName)
  - Връща нов обект от съответния тип на HTML елемент

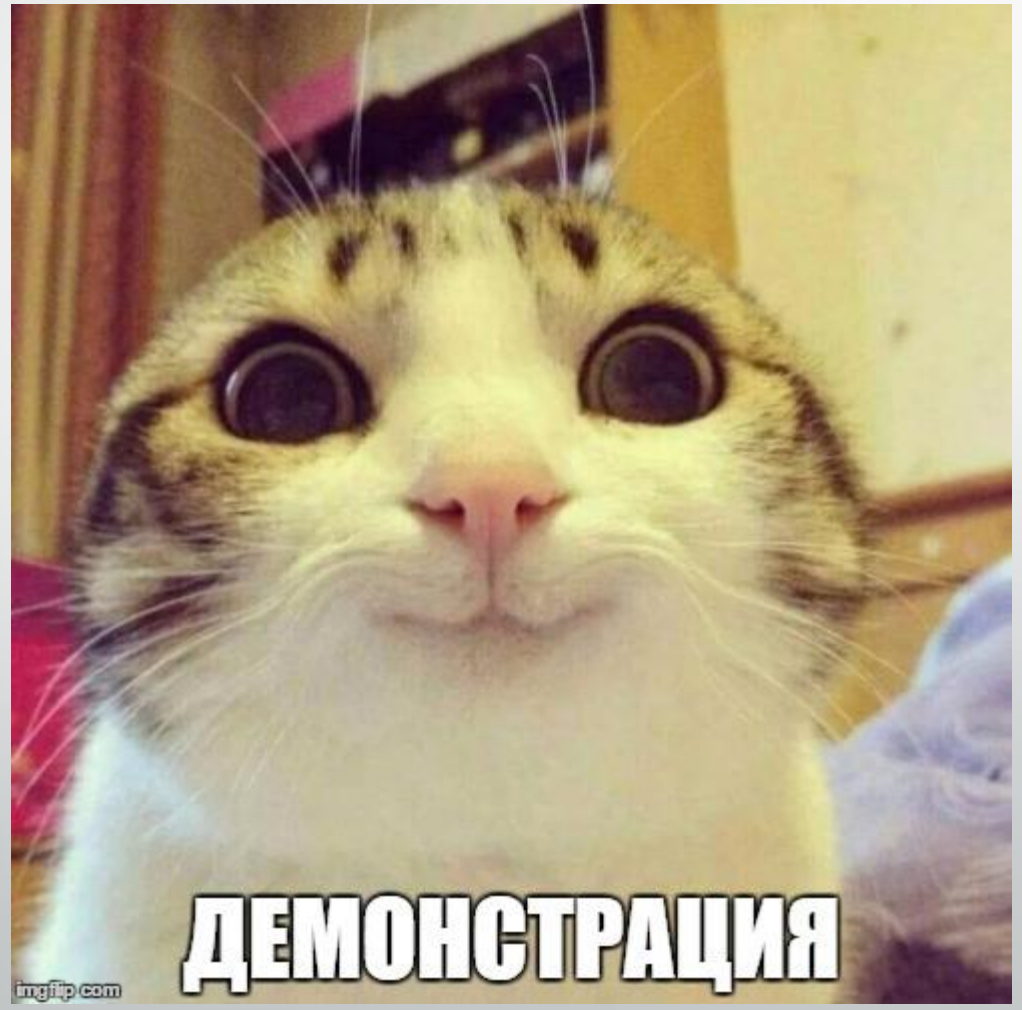
```
var liElement = document.createElement("li");
console.log(liElement instanceof HTMLLIElement); //true
console.log(liElement instanceof HTMLElement); //true
console.log(liElement instanceof HTMLDivElement); //false
```

Създаване на HTML елементи

- След създаването си HTML елемента има същите характеристики все едно е бил селектиран от DOM дървото
- Когато създаваме HTML елементи динамично с помощта на JS, те са JS обекти
  - Новосъздадения елемент не е част от DOM дървото
  - Новосъздадения елемент трябва да бъде добавен към DOM дървото

```
var teamsList = document.createElement("ul");
var teamLi = document.createElement("li");
teamsList.appendChild(teamLi);
document.body.appendChild(teamsList);
```

Създаване на HTML елементи



Вмъкване на елементи

- DOM API позволява да се вмъкват елементи преди или след даден елемент
  - *appendChild()* вмъква елемента винаги на края на дадения елемент
  - *parent.insertBefore(newNode, specificElement)* вмъква нов елемент преди зададен конкретен елемент

Премахване на HTML елементи

- Елементи могат да бъдат премахвани от DOM дървото
  - *element.removeChild(elementToRemove)*

```
var teams = document.getElementsByTagName("ul")[0];
var team = teams.getElementsByTagName("li")[0];
teams.removeChild(trainer); //remove a selected element

var selectedElement = //select the element
      selectedElement.parentNode.removeChild(selectedElement);
```

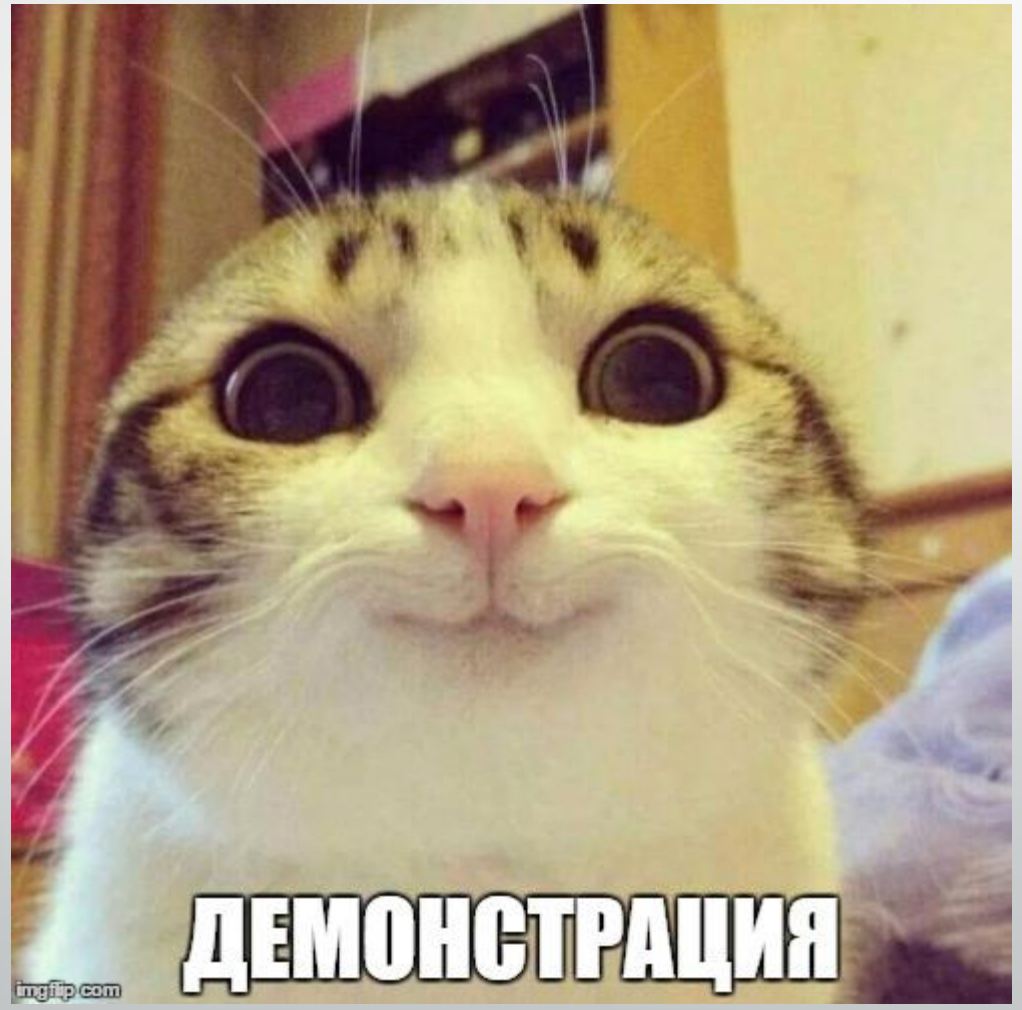
Променяне на HTML елементи

- HTML елементите могат да бъдат променяни динамично чрез JS
- Техните характеристики могат да бъдат достъпвани и променяни

```
var teams = document.getElementsByTagName("ul")[0];
var team = teams.getElementsByTagName("li")[0];
teams.removeChild(trainer); //remove a selected element

var selectedElement = //select the element
      selectedElement.parentNode.removeChild(selectedElement);
```

Вмъкване, премахване и промяна на HTML елементи

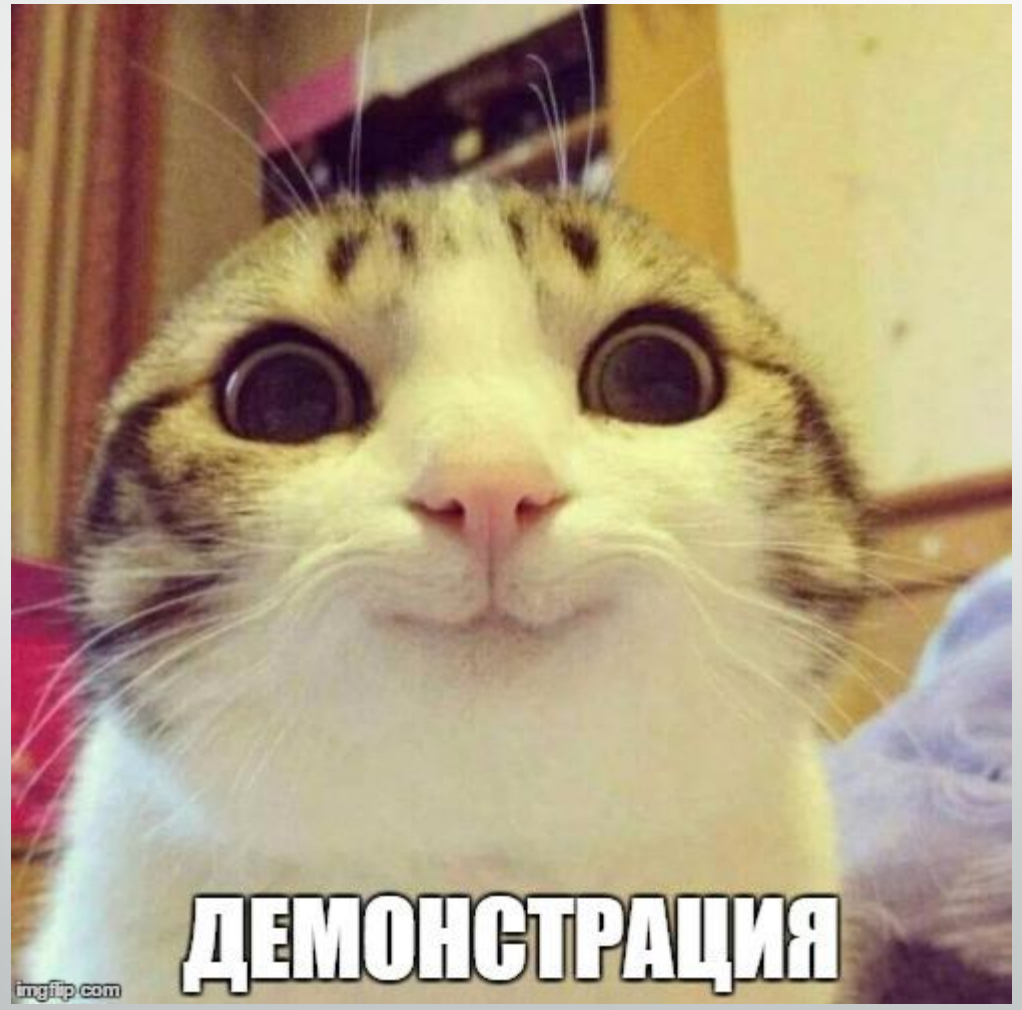


CSS модификации с JS

- Всеки HTML елемент има достъп до своя *style* атрибут
- Стойността на style атрибута може да бъде презаписвана като се достъпват съответните CSS характериситики
- *Това прави т.нар. Inline стилове, които презаписват стиловете вложени в CSS файлове, затова трябва да се внимава*

```
var div = document.getElementById("content");
div.style.display = "block";
div.style.width = "123px";
```

CSS модификации с JS



Оптимизация и производителнност на операции върху HTML елементи

- Създаването на DOM елементи е бавна операция
  - Създаваме елемент
  - Задаваме съдържание
  - Задаваме стилове
  - Задавме атрибути
- Ако трябва да създадем много елементи с обща структура и малко разлика това е проблем
  - Вместо всеки път да създаваме нов елемент, може да използваме *cloneNode(deep)*, което ще ни направи дълбоко копие на елемента
  - *deep* може да бъде *true* или *false*, като ако е *false* то децата /child nodes/ няма да бъдат копирани
  - `var clonedNode = someElement.cloneNode(true);`

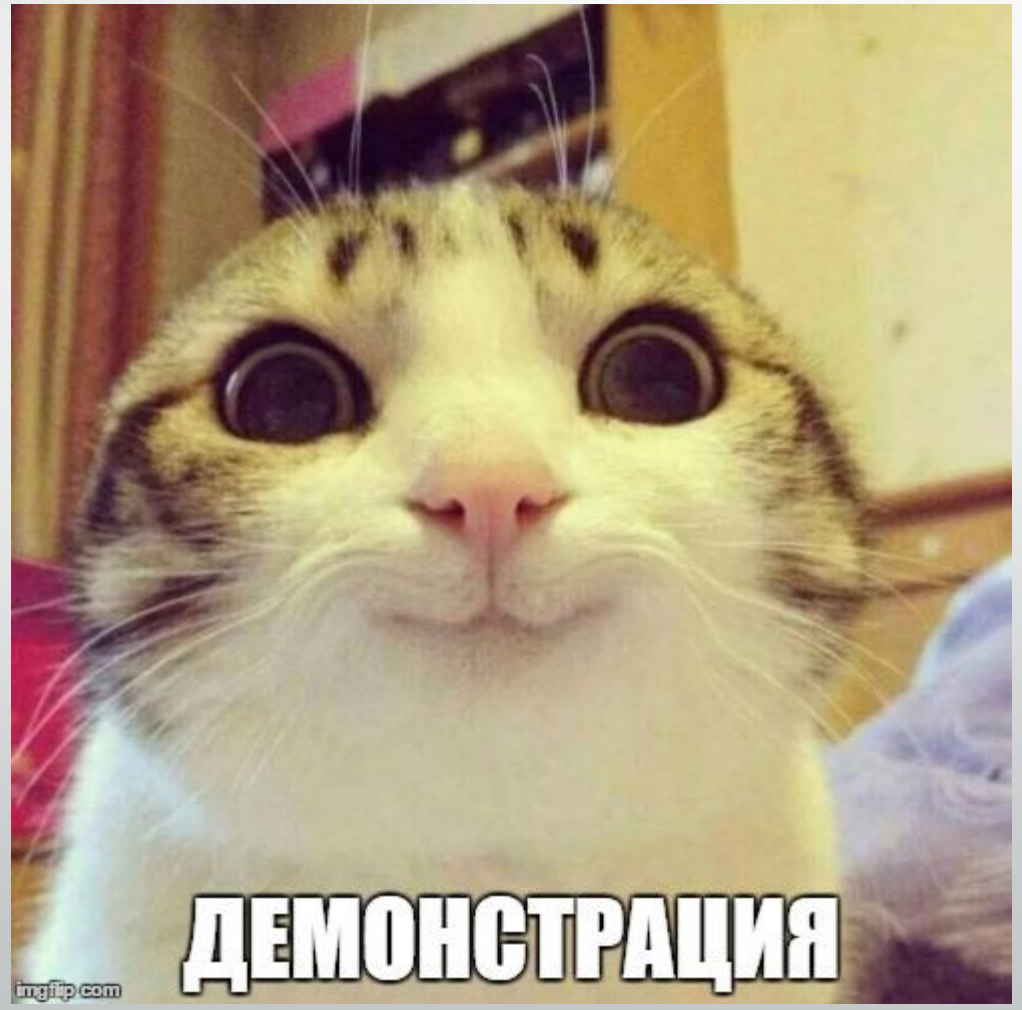
Оптимизация и производителнност на операции върху HTML елементи

- Добавянето на елементи към DOM дървото е доста бавна операция
  - След всяко добавяне DOM дървото се рендерира наново
  - Ако добавим много нови елементи директно към DOM дървото, то те ще се добавят 1 по 1 и след всеки ще се рендерира наново DOM дървото
- Можем да използваме *DocumentFragment*
  - Виртуален DOM елемент, без родител
  - Използва се за да бъдат прикрепени към всички нови HTML елементи, които да се добавят към DOM дървото наведнъж чрез въпросния фрагмент

Оптимизация и производителнност на операции върху HTML елементи

- Добре е да се кешират NodeList-овете, които получаваме , когато използваме *getElementByXXX()* или *querySelector()*
  - *var listItems = document.getElementsByClassName('list-item');*
  - Всички елементи с клас **'list-item'** към момента на изпълнение, ще бъдат кеширани в променливата *listItems*
- Добре е елементите да бъдат търсени с по-общ селектор и да не се използват изключително специфични селектори

Оптимизация и производителнност на операции върху HTML елементи




НЯКОЙ ИМА ЛИ
ВЪПРОСИ?

IT-Academy.BG

Домашна работа

1. Напишете JS код, който по зададено ID на елемент да добавя елементи в него с произволно съдържание
2. Напишете JS функция, която приема като параметри ID на елемент и масив от имена. Към елемента със съответното id да се добави лист, в който са подредени въпросните имена
   - Всеки li елемент да има стилове: цвят на текста, рамка, фон
   - Всеки li елемент да има клас 'list-item'
3. Напишете JS фунцкия, която да взима всички елементи с даден клас и във всеки четен елемент да добавя по 1 още 1 с произволно съдържание