**DOM – podstawy**

**Document:**

Obiekt zawierający wszystkie elementy w sobie. ***document.{...}***

**Funkcje:**

**Zaznaczanie elementów:   
document.getElementById( „{id}” )** – wyszukuje i zwraca element o podanym id

**document.getElementsByTagName( „{tag}” )** – wyszukuje i zwraca listę elementów o danym typie (‘img’,’h1’ etc)

**document.getElementsByClassName ( „{class}” )** – wyszukuje i zwraca listę elementów o danej klasie

**Lepsze zaznaczanie:**

**document.querySelector( ‘{css name}’ )** – wyszukuje i zaznacza pierwszy element na podstawie argumentu. Jak w css: bez niczego **html**; z **# id**; z **. klasa**; **nth-of-type**

**document.querySelectorAll( ‘{css name}’ )** - wyszukuje i zaznacza wszystkie elementy na podstawie argumentu. Jak w css.

**Text:**

**.innerText** – wszystko widoczne przez użytkownika

**.textContent** – wszystko co jest zapisane w tekście według hmtl’a (nowe linie etc.)

**.innerHTML** – zwraca wszystko włącznie z typami (<h1>, <b> etc). Używane do updatowania elementów.   
**temp.innerHTML = ‘<div = „.temp”>Tekst</div>’**

**Style:   
.style** – tyczy sie elementów zapisanych w html’u. Średnio optymalne

**.getComputedStyle( {element} ).{prop}** – zwraca zastosowaną wartość właściwości

**ClassList:**

**{element}.classList** – funkcje pozwalajace operować na klasach   
**.add( {name} )** – dodaje podaną klasę

**.remove( {name} )** – usuwa podaną klasę

**.toggle( {name} )** – przełącza między włączeniem a wyłączeniem klasy. Przydatne do eventów

**Parents:   
{element}** – start

**.parentElement** – zwraca element będący rodzicem   
**.children** – zwraca listę dzieci

**.nextElementSibling** – zwraca następny element tego samego poziomu

**.previousSibling** – zwraca poprzedni element tego samego poziomu

**Tworzenie elementów:   
.createElement(‘{type}’}** – zwraca nowy element  
**.body.appendChild( ‘{new element}’ )** – dodaje nowy element na koniec {czegoś} u nas body jako pod element   
**.{}.append( ‘{new element}’ )** – dodaje element na koniec tego samego bytu

**{}.prepend( ‘{}’ )** – dodaje element na początek tego samego bytu

**Lepszy sposób insertowania elementów:   
{target}.insertAdjacentElement( {pos}, {el} )** – wrzuca element w zależności od tego czym jest ***pos***:   
**beforebegin** – bliźniak przed targetem

**afterbegin** – dziecko na górze

**beforeend** – dziecko na dole

**afterend** – bliźniak za targetem

**Remove:   
{element}.remove();** - usuwa dany element

**DOM – Events**

**Jak dodać eventy?   
{element}.onclick = function** – najprostszy sposób. Ale tylko jedna funkcja danego typu

**{element}.addEventListner( ‘{type}’, {function} )** – najlepszy sposób. Można dodawać parę funkcji

**Rodzaje eventów jako wartości:   
.onclick** – naciśnięcie elementu

**.onmouseenter** – nakierowanie myszką na element

**Rodzaje eventów do Listner’a:**

**click** - naciśnięcie

**dbclick** – podwójne naciśnięcie

**keydown** – naciśnięcie przycisku na klawiaturze

**keyup** – podnieniesie przycisku

**submit** – przechwycenie submitowania elementu

**change** – tyczy się zmiany ***input:t*** ale dopiero po wyjściu z okna

**input** – wpisywanie/usuwanie elementów w ***input:t***

**This:   
 this.** – odwołuje się do elementu na którym wywołujemy ***.addEventListener***

**Event object:**Wywołując funkcję z eventLisner’a funkcja ta odbiera argument. Można go przechwyić tworząc jawny argument

**.addEventListner( ‘keydown’, function (e) { /// } )**

**e.key** – nazwa klawisza

**e.code** – pełny kod klawisza

**e.preventDefault()** – wyłącza typowe działanie form (nie wysyła ich)

**e.stopPropagation()** – wyłącza kotłowanie się eventów nadrzędnych

**e.target** – zwraca element dziecka wewnątrz obiektu na którym została wywołana funkcja

**Forms:   
{input name}.value** – ma zapisaną wartość wejścia inputu