TABLICE w JS
przydatne strony

http://kursjs.pl/kurs/events.html

http://www.w3schools.com/jsref/jsref_obj_array.asp

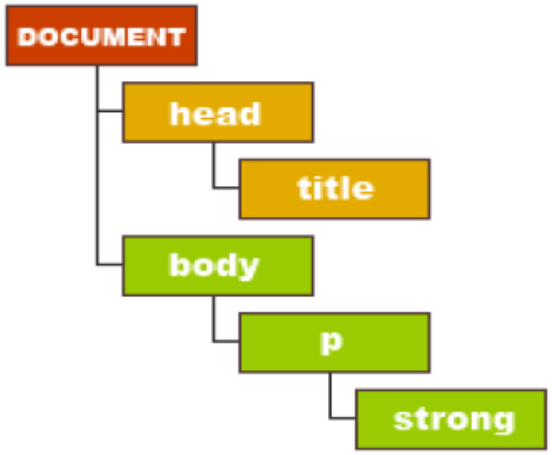
Model DOM- Document Object Model

Każda strona HTML składa się z elementów.

Na samej górze jest okno przeglądarki - **window**, które zawiera w sobie wszystkie obiekty, funkcje i właściwości. W tym oknie znajduje się obiekt **document** (czyli nasza strona). W dokumencie znajduje się mnóstwo różnych obiektów.

Do odzwierciedlenia ułożenia elementów JS korzysta z DOM. DOM czyli Document Object Model to stworzony przez W3C model ułożenia elementów na stronie - czyli hierarchia. DOM nie tylko opisuje ułożenie elementów, ale udostępnia mnóstwo metod, które ułatwiają poruszanie się po tych elementach i manipulowanie nimi.

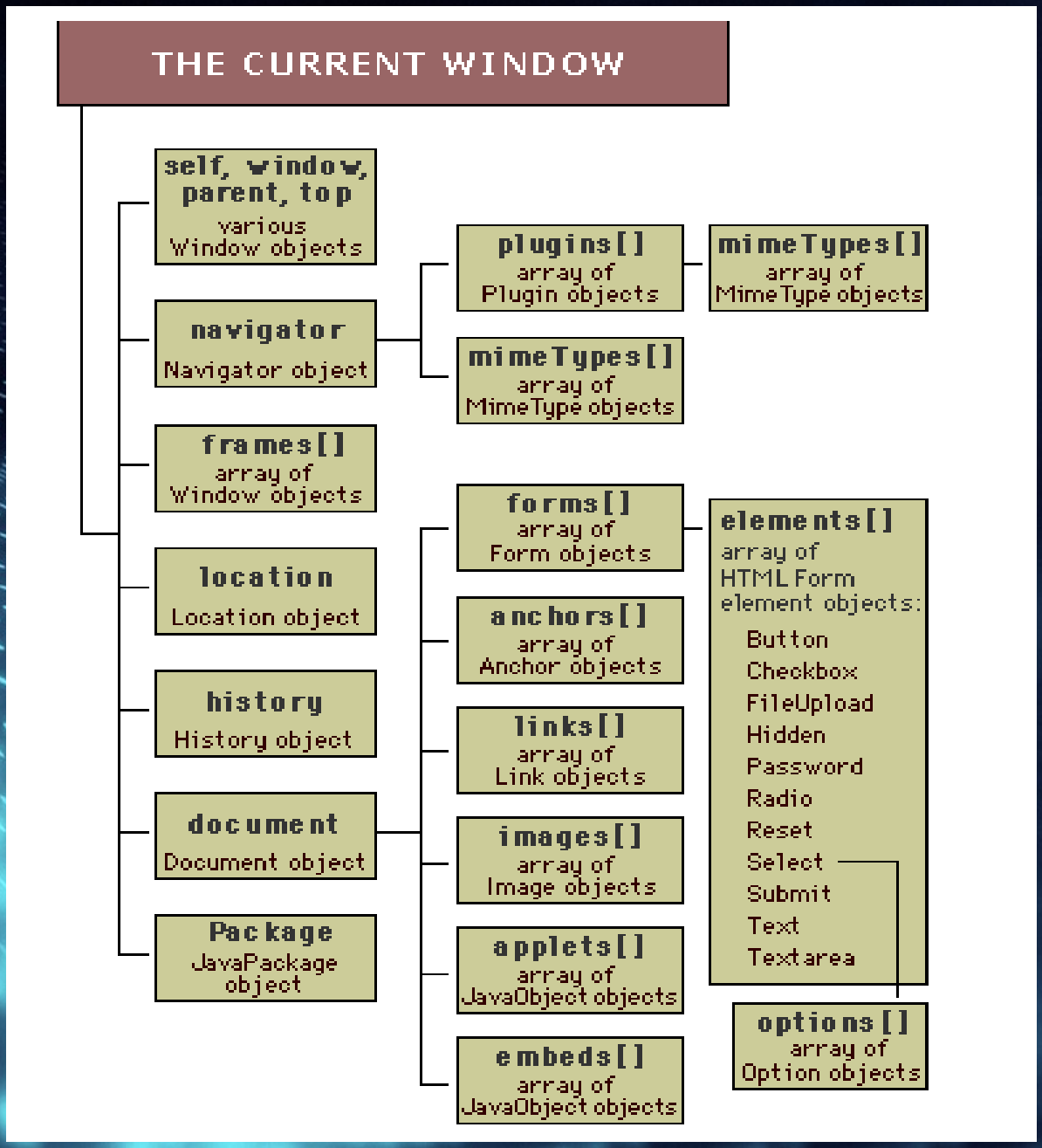
Nasz dokument możemy rozrysować jako hierarchiczne drzewo. Na samej górze jest **document** HTML, a tuż pod nim znajdują się jego "dzieci" (inna nazwa to korzenie - nody) - czyli elementy znajdujące się w document.




```
<html><head>
    <title>To jest tytuł strony</title>
</head>
<body>
    <p>Ten napis zawiera <strong>pogrubiony tekst</strong></p>
</body>
</html>
```

The hierarchy of JavaScript Objects

http://sislands.com/coin70/week1/dom.htm

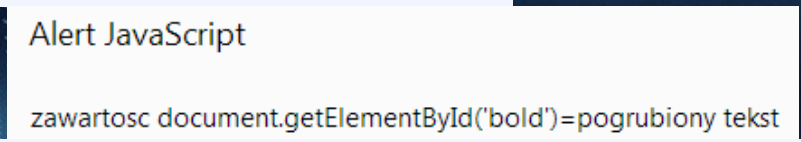


Model DOM-
Document Object Model

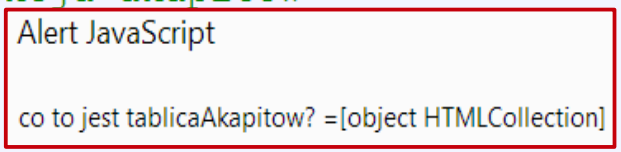
Aby odwołać się do elementów na stronie, skorzystamy z metod getElementById() , getElementsByTagName(), getElementsByClassName() lub getElementsByName() . Tą pierwszą użyjemy wtedy, gdy nasz element ma atrybut id. Trzy pozostałe służą do pobrania kolekcji zawierającej elementy danego typu lub o danej nazwie.

```
<p id="paragraf">Ten napis zawiera <strong id="bold">pogrubiony tekst</strong></p>
<p>Lorem ipsum</p>
```

```
var y=document.getElementById('bold'); //wskazuje na  znacznik strong
alert("zawartosc document.getElementById('bold')="+y.innerHTML);
```




```
var tablicaAkapitow=document.getElementsByTagName('p'); //kolekcja akapitów
alert("co to jest tablicaAkapitow? ="+tablicaAkapitow);
var n=tablicaAkapitow.length;//length zwraca długość tablicy
var s="<br>";
for(i=0;i<n;i++){s+="akapit p["+i+"] o zawartości-- "+tablicaAkapitow[i].innerHTML+"<br>";
}
document.write(s);
```



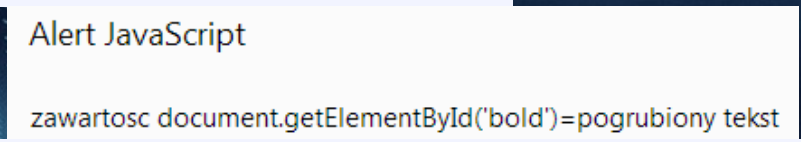
akapit p[0] o zawartości-- Ten napis zawiera **pogrubiony tekst**
akapit p[1] o zawartości-- Lorem ipsum

Model DOM-
Document Object Model

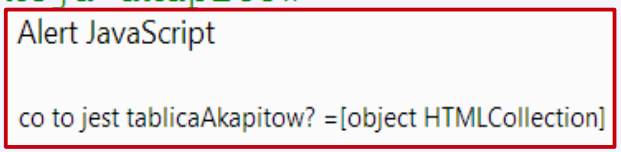
Aby odwołać się do elementów na stronie, skorzystamy z metod getElementById() , getElementsByTagName(), getElementsByClassName() lub getElementsByName() . Tą pierwszą użyjemy wtedy, gdy nasz element ma atrybut id. Trzy pozostałe służą do pobrania kolekcji zawierającej elementy danego typu lub o danej nazwie.

```
<p id="paragraf">Ten napis zawiera <strong id="bold">pogrubiony tekst</strong></p>
<p>Lorem ipsum</p>
```

```
var y=document.getElementById('bold'); //wskazuje na  znacznik strong
alert("zawartosc document.getElementById('bold')="+y.innerHTML);
```




```
var tablicaAkapitow=document.getElementsByTagName('p'); //kolekcja akapitów
alert("co to jest tablicaAkapitow? ="+tablicaAkapitow);
var n=tablicaAkapitow.length;//length zwraca długość tablicy
var s="<br>";
for(i=0;i<n;i++){s+="akapit p["+i+"] o zawartości-- "+tablicaAkapitow[i].innerHTML+"<br>";
}
document.write(s);
```



akapit p[0] o zawartości-- Ten napis zawiera **pogrubiony tekst**
akapit p[1] o zawartości-- Lorem ipsum

Model DOM-
Document Object Model

```
<p id="paragraf">Ten napis zawiera <strong id="bold">pogrubiony tekst</strong></p>
<p>Lorem ipsum</p>
```

```
//dostęp do zawartości akapitów uzyskamy również w następujący sposób:
s="<br>";
for(i=0;i<n;i++){s+=document.getElementsByTagName("p")[i].innerHTML+"<br>";
}
document.write(s);
```

Ten napis zawiera **pogrubiony tekst**
Lorem ipsum

ChildNodes - lista (array) dzieci danego obiektu (child nodes)

```
//sprawdzimy nody w akapicie
s="<br>";
for(i=0;i<n;i++){
for(j=0;j<tablicaAkapitow[i].childNodes.length;j++){
s+="("+i+" ;"+j+")  "+ tablicaAkapitow[i].childNodes[j].nodeName+"<br>";
}
}
document.write(s);
```

(0 ;0) #text
(0 ;1) STRONG
(1 ;0) #text

Model DOM-
Dostęp do elementów formularza

```
<script >
function wyslijFormularz() {
var n=document.forms[0].length;
alert("n="+n);
if(document.forms[0].elements[0].value != '' && document.forms[0].elements[1].value != ''  &&
document.forms[0].elements[2].value != ''   &&  document.forms[0].elements[3].value != ''){
alert("dane zostaną wysłane");
return true;
}

 else { if(document.forms[0].elements[0].value == '') {
 alert("podaj dane");
 document.forms[0].elements[0].select();
document.forms[0].elements[0].focus();
 }
 else  if(document.forms[0].elements[1].value == '') {
 alert("podaj dane");
 document.forms[0].elements[1].select();
document.forms[0].elements[1].focus();
 }
else  if(document.forms[0].elements[2].value == '') {
 alert("podaj dane");
 document.forms[0].elements[2].select();
document.forms[0].elements[2].focus();
 }
  else  if(document.forms[0].elements[3].value == '') {
 alert("podaj dane");
 document.forms[0].elements[3].select();
document.forms[0].elements[3].focus();
 }
return false;
}
}
</script>
```

Formularz

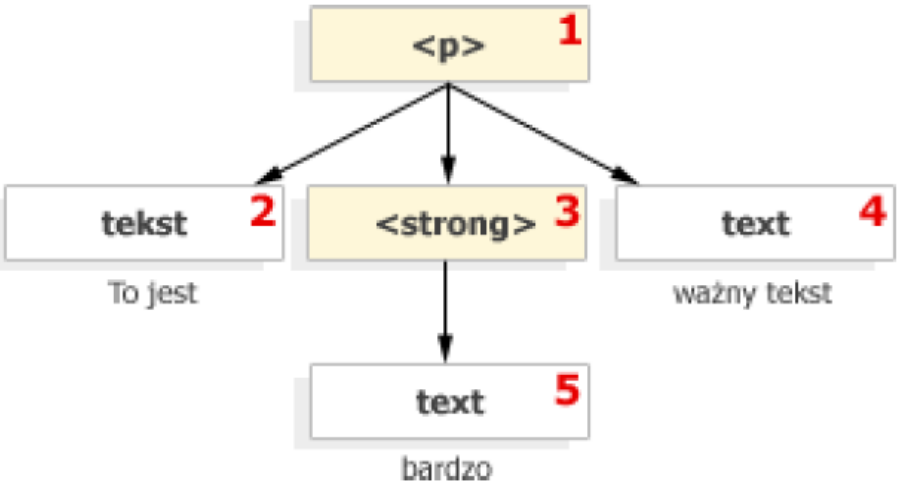
e-mail:	
nazwisko:	
imie:	
miejsce urodzenia:	

Wyślij Wyczyść formularz

```
</head>
<body >
<h1>Formularz </h1>
<form  onsubmit="return wyslijFormularz()">
  <table bgcolor="#7789aa" border="3" cellpadding="2" cellspacing="0">
      <td> e-mail:</td><td><input size="35" maxlength="80" type="text"></td></tr>
<tr> <td> nazwisko:</td><td><input  size="35" maxlength="80" type="text"></td></tr>
<tr> <td> imie:</td><td><input  size="35" maxlength="80" type="text"></td></tr>
<tr> <td> miejsce urodzenia:</td><td><input  size="35" maxlength="80" type="text"></td></tr>
</table>
<br>
    <input  type="submit"  value="  Wyślij">
    <input   type="reset"   value="Wyczyść formularz">
</form>
</body></html>
```

Model DOM-
Relacje między nodami

```
<p id="paragraf">To jest <strong>bardzo</strong> ważny tekst</p>
```



- Node 1jest rodzicem (parent) nodów **2**,**3** i **4**
- Nody 2, 3 i 4 są dziećmi (children) **noda 1**
- Node 2 jest pierwszym dzieckiem (first child) **noda 1**
- Node 4 jest ostatnim dzieckiem (last child) **noda 1**
- Node 3 jest kolejnym elementem po Nodzie 2
- Node 3 jest poprzednim elementem przez Nodem **4**
- Node 5 nie jest dzieckiem (child) **noda 1** - jest za to dzieckiem Noda 3
- Nody 2, 4 i 5są typu tekstowego

Model DOM- Relacje między nodami

```
<p id="paragraf">Ten napis zawiera <strong id="bold">pogrubiony tekst</strong></p>
<p>Lorem ipsum</p>
```

```
document.getElemensByTagName('p')[0]; //wskazuje na pierwszy akapit
document.getElemensByTagName('p')[0].getElementsByTagName('strong')[0];
//pobieramy pierwszy akapit, a w nim pobieramy pierwszy strong
document.getElementById('paragraf').getElementsByTagName('strong');
//kolekcja znaczników strong znajdujących się w akapicie paragraf
```

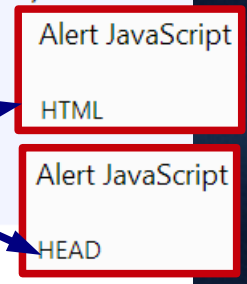
Model DOM- Relacje między nodami

Nie potrzeba korzystać z metody getElementsByTagName do pobrania wszystkich elementów na stronie i używać indeksów. Każdy węzeł (czyli element, fragment tekstu, komentarz), posiada pola wskazujące na jego sąsiednie węzły:

Istnieje kilka właściwości przypisanych do każdego node:

Właściwość	Opis
nodeName	nazwa node
nodeValue	wartość node (tylko dla nodów tekstowych)
nodeType	typ node
parentNode	rodzic (parent), jeżeli istnieje
childNodes	lista (array) dzieci danego obiektu (child nodes)
firstChild	pierwsze dziecko (first child)
lastChild	ostatnie dziecko (last child)
previousSibling	zwraca poprzedni nod na tym samym poziomie
nextSibling	zwraca następny nod na tym samym poziomie

```
<title>Dostęp do elementów HTML</title>
<script >
var obiektHtml = document.documentElement;
var obiekt2 = obiektHtml.firstChild;
function pokaz_wezly()
{
    alert(obiektHtml.nodeName);
    alert(obiekt2.nodeName);
}
</script>
<head>
<body>
    <input type="button" onclick=
    "pokaz_wezly()" value="Pokaż węzły">
</body>
</html>
```



Zdarzenia- Interakcja z użytkownikiem

Zdarzenia to czynności, które użytkownik wykonuje podczas odwiedzania naszej strony. Przykładowymi zdarzeniami mogą być np. przesunięcie kursora na obrazek, kliknięcie jakiegoś linka, wysłanie formularza, zaznaczenie jakiegoś obiektu, naciśnięcie klawisza itp.

Standard DOM zawiera specyfikację zdarzeń. Zdarzenia można ogólnie podzielić na cztery grupy:

- zdarzenia generowane przez mysz: kliknięcie (onClick), podwójne kliknięcie (onDblClick), naciśnięcie elementu (onMouseDown), zwolnienie przycisku myszy (onMouseUp), najechanie myszą (onMouseOver), przesuwanie myszą nad elementem (onMouseMove), opuszczenie myszą obszaru nad elementem (onMouseOut)
- zdarzenia generowane przez klawiaturę: wciśnięcie i zwolnienie klawisza (onKeyPress), wciśnięcie klawisza (onKeyDown), zwolnienie klawisza (onKeyUp)
- zdarzenia związane z obiektami i ramkami: załadowanie (onLoad), przerwanie ładowania (onAbort), wystąpienie błędu ładowania (onError), zmiana rozmiaru okna/ramki (onResize), przesunięcie widocznego okna (onScroll)
- zdarzenia związane z formularzami: zaznaczenie tekstu (onSelect), zmiana wartości pola (onChange), przesłanie formularza (onSubmit), wyczyszczenie formularza (onReset), wejście do elementu (onFocus), wyjście z elementu (onBlur)

Zdarzenia-
Interakcja z użytkownikiem

```
<!DOCTYPE html>
<html lang="pl">
<head>
<meta charset="utf-8">
    <title>Zmiana koloru elementów strony</title>

    <script language="javascript" type="text/javascript">
        function zmianaKoloruTlaElementu(jakiesID)
        {
           // var kolorElementu = new String();

          var kolorElementu= document.getElementById(jakiesID).style.backgroundColor;


            if(kolorElementu.toLowerCase()=='#eeeeee' || kolorElementu.toLowerCase()=='rgb(238, 238, 238)')
            {
                document.getElementById(jakiesID).style.backgroundColor = '#ff0000';
            }
            else
            {
                document.getElementById(jakiesID).style.backgroundColor = '#eeeeee';
            }
        }
    </script>
</head>
<body>
        <div id="div1" style="background-color : #EEEEEE; width: 200px; height: 100px">
            po kliknięciu nastąpi zmiana koloru elementu div
        </div>
        <input type="button" value="zmien klor" onclick="zmianaKoloruTlaElementu('div1')" />
</body>
</html>
```

Zdarzenia- Interakcja z użytkownikiem

Rejestrowanie zdarzenia inline - niepolecany!!!!

Metoda inline przypisywania zdarzeń polega na określeniu zdarzenia wewnątrz znacznika. Na przykład:

```
<a href="jakasStrona.html" onclick="alert(' Kliknąłeś link! ')"> kliknij
</a>
```

Od tej chwili po kliknięciu na link zostanie wywołane zdarzenie **click**, które wyświetli okienko dialogowe Alert.

Zdarzenia- Interakcja z użytkownikiem

Oddzielenie javascript od html - tradycyjny model rejestrowania zdarzenia

Pisanie kodu js wewnątrz znaczników stwarza same problemy. Dlatego właśnie o wiele lepszym pomysłem jest oddzielenie skryptów od html dzięki podpinaniu zdarzeń do elementów bezpośrednio w skryptach.

Ogólna konstrukcja przypisania zdarzenia ma postać:

```
var element = document.getElementById('guzik');

element.onclick = zrobCos
element2.onmouseover = zrobCosInnego
```

Jeżel chcemy usunąć dane zdarzenia dla danego obiektu, wystarczy, że przypiszesz mu wartość **null**:

```
element.onclick = null
```

Zdarzenia- Interakcja z użytkownikiem

Przykładowo przypiszmy guzikowi zdarzenie **onclick**:

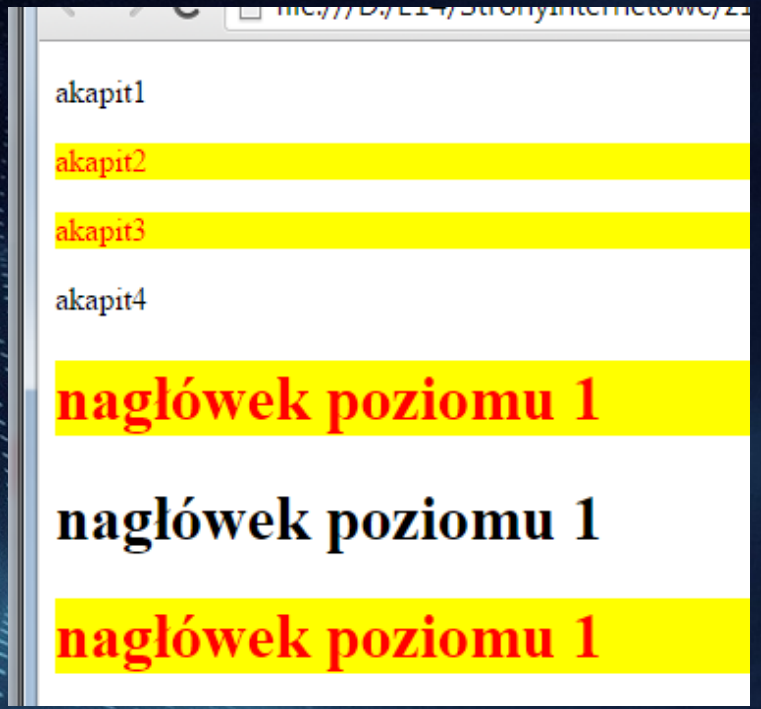
```
1  <input type="button" id="guzik" value="kliknij" />
2
3  ...
4
5  <script type="text/javascript">
6      function wypisz() {
7          alert(' zostałem kliknięty! ');
8      }
9
10     document.getElementById('guzik').onclick = wypisz
11 </script>
12
```

przy podpinaniu funkcji do zdarzeń pomijamy nawiasy

Zdarzenia-
Interakcja z użytkownikiem

```
<body>
        <p>akapit1</p>
         <p>akapit2</p>
          <p>akapit3</p>
           <p>akapit4</p>
        <h1>nagłówek poziomu 1</h1>
         <h1>nagłówek poziomu 1</h1>
          <h1>nagłówek poziomu 1</h1>

    <script>
function zmienKolor(){
          this.style.color = 'red';
          this.style.backgroundColor = 'yellow';
}
var n=document.getElementsByTagName('p').length;
alert(n);
for (i=0; i<n ; i++)
  {
   document.getElementsByTagName('p')[i].onclick=zmienKolor;
  }
var n1=document.getElementsByTagName('h1').length;
alert(n1);
for (i=0; i<n1 ; i++)
  {
   document.getElementsByTagName('h1')[i].onclick=zmienKolor;
  }
    </script>
</body>
```



Uwaga- skrypt na końcu body

Zdarzenia- -funkcja anonimowa

Funkcja anonimowa to funkcja, którą tworzymy bezpośrednio przy deklaracji zdarzenia).

```
    <title>Zmiana koloru elementów strony</title>

</head>
<body>
        <p>akapit1</p>
         <p>akapit2</p>
          <p>akapit3</p>
           <p>akapit4</p>
        <h1>nagłówek poziomu 1</h1>
         <h1>nagłówek poziomu 1</h1>
          <h1>nagłówek poziomu 1</h1>

    <script>

var n=document.getElementsByTagName('p').length;
alert(n);
for (i=0; i<n ; i++)
  {
   document.getElementsByTagName('p')[i].onclick=function (){ this.style.color = 'blue';
        this.style.backgroundColor = 'green'; }
  }
var n1=document.getElementsByTagName('h1').length;
alert(n1);
for (i=0; i<n1 ; i++)
  {
   document.getElementsByTagName('h1')[i].onclick=function (){ this.style.color = 'red';
        this.style.backgroundColor = 'yellow'; }
  }
    </script>
</body>
```

```
function wypisz(tekst) {
    alert(tekst);
}

document.getElementById('guzik').onclick = function() {
    wypisz('Przykładowy tekst')
}

```

akapit1

akapit2

akapit3

akapit4

nagłówek poziomu 1

nagłówek poziomu 1

nagłówek poziomu 1

Nowy model rejestracji zdarzeń

Problem z tradycyjnym modelem polega na tym, że do jednego elementu możemy podpiąć tylko jedną funkcję dla jednego rodzaju zdarzenia. Rejestrując nową funkcję do danego zdarzenia nadpisujemy starą.

Problemów takich nie mamy korzystając z "nowego" modelu rejestrowania zdarzeń opierającego się na metodzie addEventListener().

Przyjmuje ona 3 argumenty: typ zdarzenia, funkcja do wywołania, oraz trzeci(true/false). W praktyce trzeci argument zawsze pozostaje jako false.

Nowy model rejestracji zdarzeń

Problem z tradycyjnym modelem polega na tym, że do jednego elementu możemy podpiąć tylko jedną funkcję dla jednego rodzaju zdarzenia. Rejestrując nową funkcję do danego zdarzenia nadpisujemy starą.

Problemów takich nie mamy korzystając z "nowego" modelu rejestrowania zdarzeń opierającego się na metodzie addEventListener().

Przyjmuje ona 3 argumenty: typ zdarzenia, funkcja do wywołania, oraz trzeci(true/false). W praktyce trzeci argument zawsze pozostaje jako false.

Nowy model rejestracji zdarzeń

```
<body>
        <h1 id="a1"> nagłówek stopnia 1-kliknij</h1>

    <script>
var element=document.getElementById("a1");

function wypiszDate(){
this.innerHTML=Date();
}

element.addEventListener('click', function() {
    this.style.color = 'red';
}, false);

element.addEventListener('click', wypiszDate, false );
element.addEventListener('click',function(){this.style.backgroundColor="blue"}, false );

    </script>
</body>
```

Sat Dec 13 2014 13:11:44 GMT+0100 (Środkowoeuropejski czas stand.)