Programowanie obiektowe

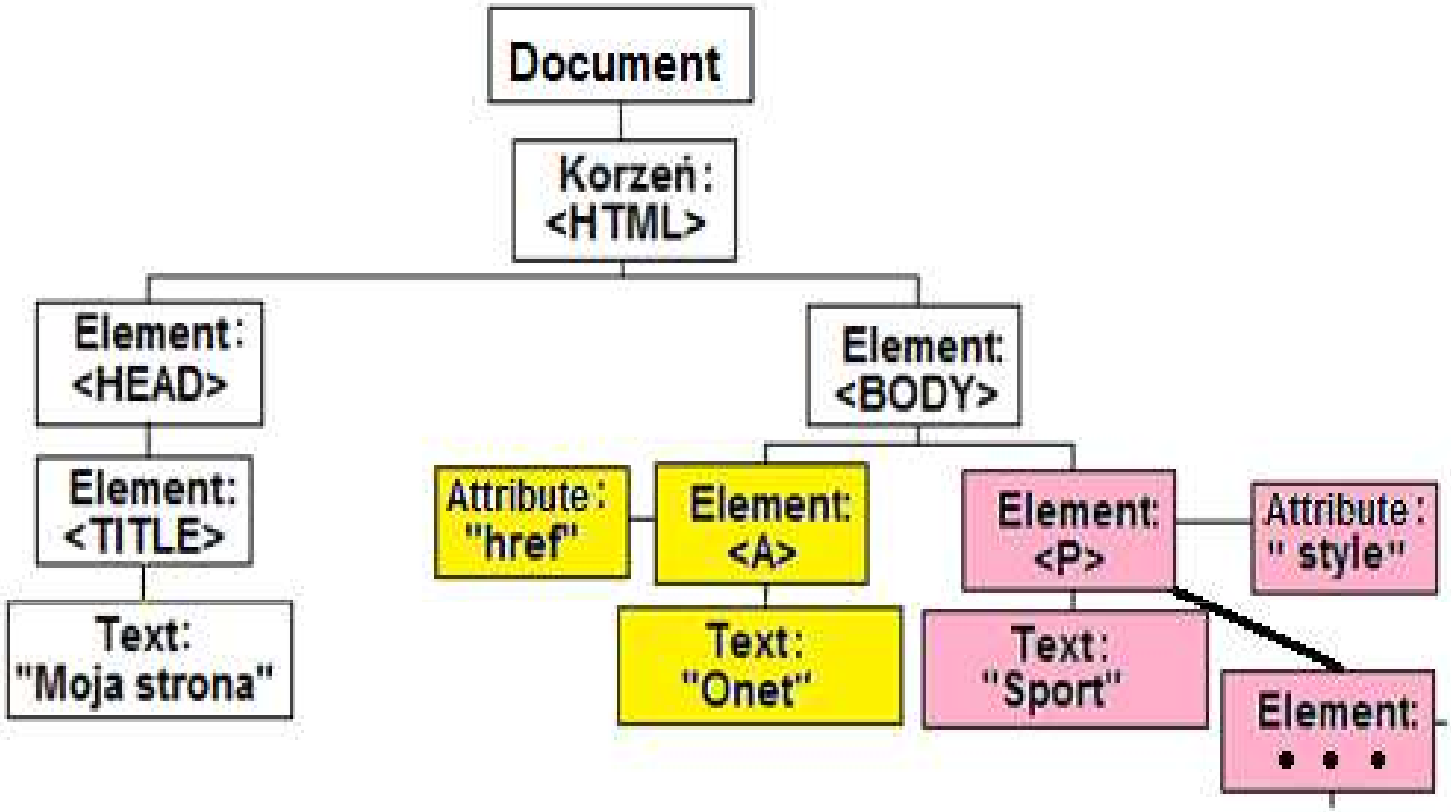
Wykład 3

DOM – model dokumentu HTML

Dokument HTML składa się z obiektów (standardowych i utworzonych przez użytkownika).

Głównym obiektem jest **document**, posiada strukturę drzewiastą składowych (obiektów wewnętrznych-znaczników na stronie) – tworzących tzw. **DOM** (*Document Object Model*).

Wykorzystamy do wizualizacji funkcjonalności obiektowych



Jak widać elementy mają hierarchię, mogą się zagnieżdżać ("rodzice" i "dzieci").

Obiekt **document**

Obiekt document jest ważnym obiektem *JavaScript*. Zawiera on informacje o bieżącej stronie i dostarcza wielu sposobów wyświetlania strony HTML.

Składnia wykorzystania właściwości i metod:

document.właściwość

lub

document.metoda( )

Wybrane właściwości

Właściwość	Opis
document.linkColor	określa kolor odsyłaczy
document.bgColor	określa kolor tła
document.fgColor	określa kolor tekstu
document.lastModified	zawiera datę i czas ostatniej modyfikacji dokumentu
document.location	zawiera bieżący adres URL dokumentu
document.title	zawiera zawartość znacznika <TITLE>
document.body	zawiera element(obiekt) body
document.links	zawiera kolekcję wszystkich hiperłączy w dokumencie – kolekcja (TABLICA - zawiera właściwość **length -** liczba elementów kolekcji)
document.title	zawiera tytuł dokumentu
document.URL	zawiera pełny URL dokumentu

Metody

document.write() **document.writeln**()
dostęp do elementów
document.getElementById("*id***")**
document.getElementsByTagName("*znacznik***")** kolekcja
document.getElementsByName("*nazwa***")** kolekcja
tworzenie nowego elementu
document.createTextNode("tekst")
document.ceateElement("*znacznik***")**

```
<HTML><HEAD> </HEAD>
<BODY style="font-size:18px">
<P>Przykłady wykorzystania obiektu document</P>
<P> Akapit na stronie</P>
<A href="http://www.youtube.com">Youtube</A>
<A href="http://www.facebook.com">Facebook</A>
<PRE>
<SCRIPT language="JavaScript">
//ustalanie właściwości dokumentu
document.bgColor="yellow";
document.linkColor="red";
document.fgColor="green";
//wyświetlanie właściwości
document.writeln("Data utworzenia:",document.lastModified);
document.writeln("URL:",document.URL);
document.writeln("location:",document.location);
document.writeln("Tytuł:",document.title);
//elementy kolekcji w pętli
for (k=0;k<document.links.length;k++)
     document.writeln("Link ",k,":",document.links[k]); //jak tablica (indeks)

</SCRIPT>
</PRE></BODY></HTML>
```

Wykorzystanie własnej funkcji do ustawienia właściwości

```
<HTML><HEAD>
<TITLE> Przykłady wykorzystania obiektu document</TITLE>
<SCRIPT language="JavaScript">
function czerw()
        {document.bgColor="red";}
function zielo()
        {document.bgColor="green";}
</SCRIPT>
</HEAD>
<BODY style="font-size:18px">

<INPUT type="button" value="czerwone"  onclick="czerw()">
<INPUT type="button" value="zielone"  onclick="zielo()">

</BODY></HTML>
```

.. albo lepiej…

```
<HTML><HEAD>
<TITLE> Przykłady wykorzystania obiektu document</TITLE>
<SCRIPT language="JavaScript">
function setKolor(k)
{
        document.bgColor=k;
}
</SCRIPT>
</HEAD>
<BODY style="font-size:18px">

<INPUT type="button" value="czerwone" onclick="setKolor('red')">
<INPUT type="button" value="zielone" onclick="setKolor('green')">
</BODY></HTML>
```

Dostęp do elementów strony

Tworzenie referencji do istniejącego elementu (obiektu) – zmiennej reprezentującej element na stronie - najwygodniej wykonać instrukcją:

var *element* **= document.getElementById** *("identyfikator")*;

pobierz element dokumentu o pewnym id

albo

var *element*

= document.getElementsByTagName *("znacznikHTML")*;

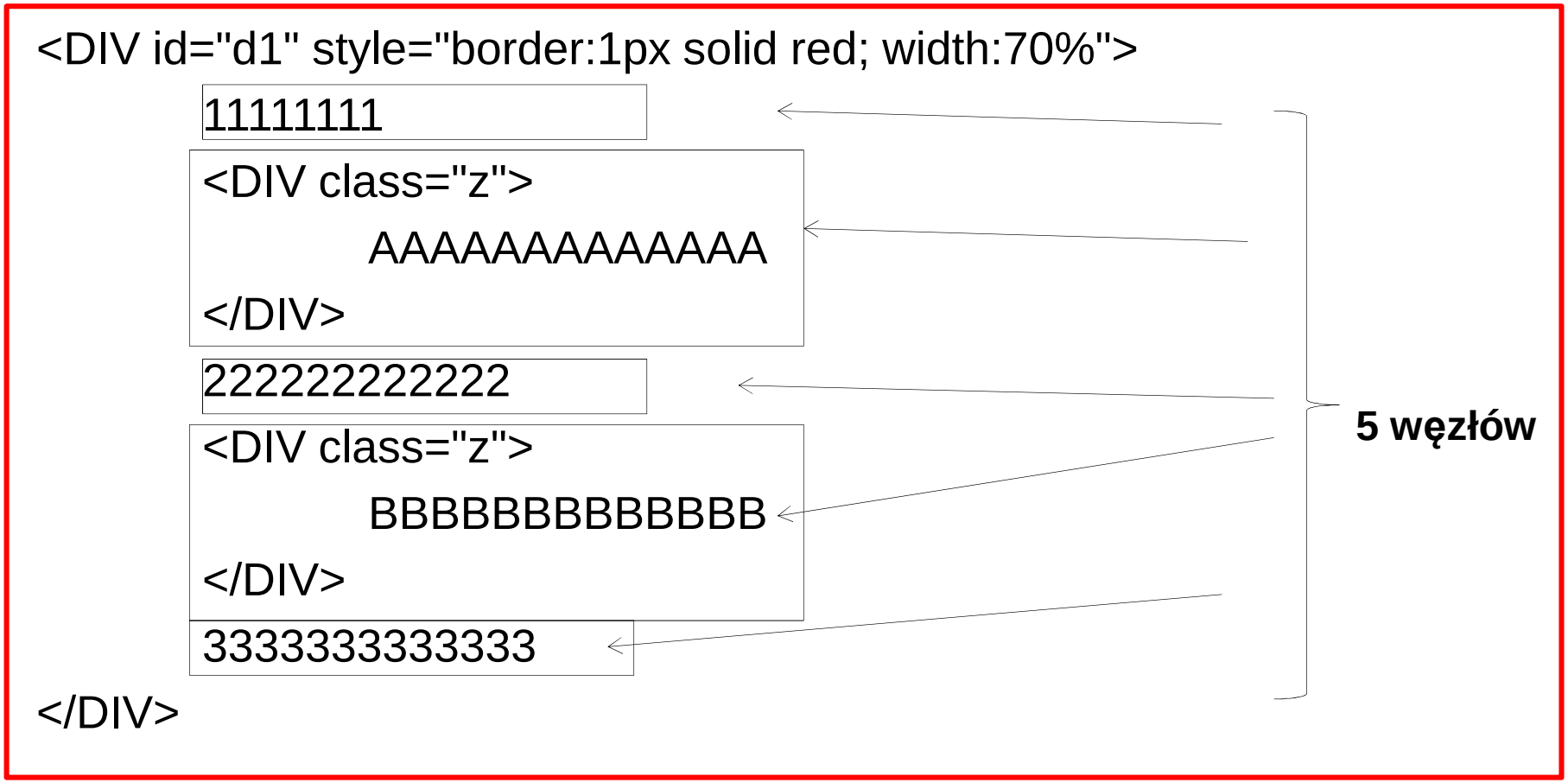
pobierz wszystkie elementy dokumentu o pewnym znaczniku HTML
to będzie kolekcja - TABLICA

Ważniejsze właściwości tak pobranego elementu:

node= węzeł

*element.**childNodes***	kolekcja **węzłów** dla elementu - zawiera atrybut **length** (liczba elementów kolekcji), każdy węzeł ma **nodeName**, **nodeType** i jeśli tekstowy **nodeValue**
*element.**style***	dostęp do ustawiania atrybutu *style* elementu
*element.**attributes***	**kolekcja** atrybutów elementu - mają one właściwości: **length –** liczba atrybutów **[nr].name** – nazwa **[nr].value –** wartość atrybutu
*element.**innerHTML***	wartość węzła tekstowego

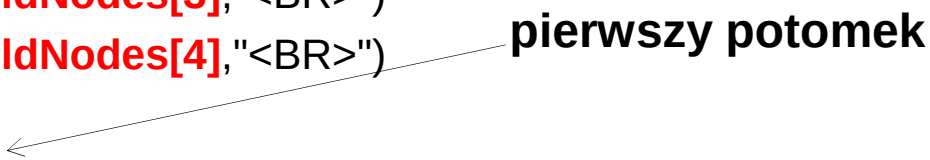
Przykład1 – mamy tak zagnieżdżające się znaczniki

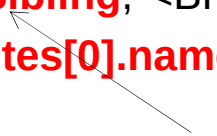


```
<SCRIPT language="JavaScript">
var x=document.getElementById("d1") //pobierz element
document.write("DIV ma "+x.childNodes.length+" węzłów<BR>")
document.write("Czyli: 1text 2DIV 3text 4DIV 5text<BR>")
document.write("DIV childNodes[0] :",x.childNodes[0],"<BR>")
document.write("DIV childNodes[1] :",x.childNodes[1],"<BR>")
document.write("DIV childNodes[2] :",x.childNodes[2],"<BR>")
document.write("DIV childNodes[3] :",x.childNodes[3],"<BR>")
document.write("DIV childNodes[4] :",x.childNodes[4],"<BR>")

document.write("Węzeł 0:<BR>")
document.write("DIV childNodes[0].nodeName :",x.childNodes[0].nodeName, "<BR>")
document.write("DIV childNodes[0].nodeType :",x.childNodes[0].nodeType, "<BR>")
document.write("DIV childNodes[0].nodeValue :",x.childNodes[0].nodeValue, "<BR>")
document.write("i inne.....:<BR>")

document.write("Węzeł 1:<BR>")
document.write("DIV childNodes[1].nodeType :",x.childNodes[1].nodeType,"<BR>")
document.write("DIV childNodes[1].nodeName :",x.childNodes[1].nodeName,"<BR>")
```

```
document.write("Węzeł 2:<BR>")
document.write("DIV childNodes[2].nodeType :",x.childNodes[2].nodeType,"<BR>")
document.write("DIV childNodes[2].nodeName :",x.childNodes[2].nodeName,"<BR>")
document.write("DIV childNodes[2].nodeValue :",x.childNodes[2].nodeValue,"<BR>")

document.write("Węzeł 3:<BR>")
document.write("DIV childNodes[3].nodeType :",x.childNodes[3].nodeType,"<BR>")
document.write("DIV childNodes[3].nodeName :",x.childNodes[3].nodeName,"<BR>")

document.write("Węzeł 4:<BR>")
document.write("DIV childNodes[4].nodeType :",x.childNodes[4].nodeType,"<BR>")
document.write("DIV childNodes[4].nodeName :",x.childNodes[4].nodeName,"<BR>")
document.write("DIV childNodes[4].nodeValue :",x.childNodes[0].nodeValue,"<BR>")
```

```
document.write("Można też tak:<BR>")

document.write("DIV childNodes.item(0) :",x.childNodes.item(0),"<BR>")
document.write("DIV childNodes.item(1) :",x.childNodes.item(1),"<BR>")
document.write("DIV childNodes.item(2) :",x.childNodes.item(2),"<BR>")
document.write("DIV childNodes.item(3) :",x.childNodes.item(3),"<BR>")
document.write("DIV childNodes.item(4) :",x.childNodes.item(4),"<BR>")

document.write("DIV childNodes[0]: ",x.childNodes[0],"<BR>")
document.write("DIV childNodes[1]: ",x.childNodes[1],"<BR>")
document.write("DIV childNodes[2]: ",x.childNodes[2],"<BR>")
document.write("DIV childNodes[3]: ",x.childNodes[3],"<BR>")
document.write("DIV childNodes[4]: ",x.childNodes[4],"<BR>")

document.write("DIV firstChild: ",x.firstChild,"<BR>")
document.write("DIV firstChild.nextSibling: ",x.firstChild.nextSibling,"<BR>")
document.write("Atrybut 0 naszego DIV ma nazwę: ", x.attributes[0].name)
</SCRIPT></BODY></HTML>
```





następny brat (rodzeństwo)

lub pętla…

```
<DIV id="d1" style="border:1px solid red;width:70%">
        11111111
        <DIV class="z">
                AAAAAAAAAAAA
        </DIV>
        222222222222
        <DIV class="z">
                BBBBBBBBBBBB
        </DIV>
        3333333333333
</DIV>
<SCRIPT language="JavaScript">
var x=document.getElementById("d1") //pobierz element
document.write("DIV ma "+x.childNodes.length+" węzłów<BR>")
document.write("Czyli: 1text 2DIV 3text 4DIV 5text<BR>")
for (n=0;n<=4;n++)
        document.write("DIV childNodes[",n,"] :",x.childNodes[n],"<BR>")
</SCRIPT>
```

Atrybut **style** jest również obiektem, poniżej ważniejsze właściwości:

Właściwość	Opis
backgroundColor	ustalenie lub pobranie koloru tła elementu
border	ustawienie (np. "1px solid red") lub pobranie cech obramowania
borderStyle	ustawienie (np. "solid") lub pobranie stylu obramowania
borderColor	ustawienie (np. "red") lub pobranie koloru obramowania
borderBottom	dolna krawędź obramowania – j.w.
borderBottomColor	dolna krawędź obramowania – kolor
borderBottomStyle	dolna krawędź obramowania – kolor
borderBottomWidth	dolna krawędź obramowania – kolor
borderLeft borderTop borderRight	podobnie – j.w.
margin	margines np. "12px 20mm 2 cm 20px" – góra prawy dolny lewy
marginBottom marginLeft marginRight marginTop	j.w.
color	ustawienie (np. "14px solid red") lub pobranie koloru czcionki
fontFamily	ustawienie (np. "Arial") lub pobranie rodziny czcionki
fontSize	ustawienie (np. "15px") lub pobranie rozmiaru czcionki
fontStyle	ustawienie (np. "italic") lub pobranie stylu czcionki
fontWeight	ustawienie (np. "bold") lub pobranie grubości czcionki
textAlign	ustawienie (np. "center") lub pobranie wyrównania tekstu

Zmiana stylu znacznika

```
<HTML><HEAD> <TITLE> Przykłady obiektowe</TITLE></HEAD>
<BODY style="font-size:18px">
<TABLE style="border:1px solid red;font-size:25px">
<TR><TD>a1</TD> <TD>a2</TD><TD>a3</TD></TR>
<TR><TD>b1</TD> <TD>b2</TD><TD>b3</TD></TR>
</TABLE>
<SCRIPT language="JavaScript">
x=document.getElementsByTagName("TD") //pobierz elementy wg znacznika!
x[0].style.border="6px solid red"
x[1].style.backgroundColor="blue"
</SCRIPT>
</BODY></HTML>
```

Interakcyjna zmiana stylu znacznika

```
<HTML><HEAD> <TITLE> Przykłady obiektowe</TITLE>
<SCRIPT language="JavaScript">
function setRamka(n, kolor){
  var x=document.getElementsByTagName("TD") //pobierz elementy TD!
  x[n].style.border="3px solid "+kolor
}
</SCRIPT>

</HEAD>
<BODY style="font-size:18px">
<TABLE style="border:1px solid red">
<TR><TD>a1</TD> <TD>a2</TD><TD>a3</TD></TR>
<TR><TD>b1</TD> <TD>b2</TD><TD>b3</TD></TR>
</TABLE>
Wpisz numer od 1 do 6<INPUT type="text" name="pole">
<INPUT type="button" value="Ustaw kolor" onclick="setRamka(pole.value-1, 'red')">
</BODY></HTML>
```

uwaga na spację po ***solid***

Proste sterowanie położeniem przez styl margin

```
<HTML><HEAD> <TITLE> Przykłady obiektowe</TITLE>
<SCRIPT language="JavaScript">
function setMargin(m){
   var x=document.getElementsByTagName("DIV") //pobierz elementy wg znacznika!!!!
   x[0].style.marginLeft=m+"px"
}
</SCRIPT>

</HEAD>
<BODY style="font-size:18px">
<DIV  style="border:1px solid red; width:100px;height:100px; margin-
left:200px;border:3px solid red">
</DIV>
<INPUT type="button" value="Prawo" onclick="setMargin(300)">
<INPUT type="button" value="Lewo" onclick="setMargin(100)">
</BODY></HTML>
```

Sterowanie na płaszczyźnie

```
<HTML><HEAD> <TITLE> Przykłady obiektowe</TITLE>
<STYLE type="text/css">
div.x{border:1px solid red; width:100px;height:100px;background-color:red; margin-left:0px;}
div.z{
border:1px solid red; width:300px;height:300px;border:1px solid green; margin-left:200px;}
</STYLE>
<SCRIPT language="JavaScript">
function setMargin(m){
   var x=document.getElementById("d1") //pobierz element o id=d1
   x.style.marginLeft=m+"px"
}
function setMargin2(m){
   var x=document.getElementById("d1") //pobierz element o id=d1
   x.style.marginTop=m+"px"
}
</SCRIPT>
</HEAD>
```

cd .....

```
<BODY style="font-size:18px">
    <INPUT type="button" value="Prawo" onclick="setMargin(200)">
    <INPUT type="button" value="Lewo" onclick="setMargin(0)">
    <INPUT type="button" value="Dół" onclick="setMargin2(200)">
    <INPUT type="button" value="Góra" onclick="setMargin2(0)">

    <DIV class="z">
        <DIV id="d1" class="x">
        </DIV>
    </DIV>
</BODY></HTML>
```

Przydatne metody dla zmian struktury obiektów

document.**createElement**(typ)	utworzenie elementu o typie znacznika
element.**appendChild**(*element_dołączany*)	dołączanie elementu do innego elementu (na końcu)
element.**cloneNode**(true)	powielenie elementu
element.**removeChild**(element_dziecko);	usunięcie elementu podrzędnego
element.**replaceChild**(el_nowy,el_stary);	podmiana elementu podrzędnego

append = dołącz

create = utwórz

clone = klonuj, kopiuj

remove = usuń

replace= zamień

Tworzenie nowego elementu na końcu strony

```
<HTML>
<HEAD> </HEAD>
<BODY style="font-size:18px">
<!--  Istniejący DIV    -->
<DIV  id="div1">Mechatronika </DIV>
<SCRIPT language="javascript">
//dostęp do elementu
d1 = document.getElementById("div1");
//ustawianie parametrów stylu istniejącego DIV
d1.style.color="red";
d1.style.fontSize="28px";
d1.style.border="4px solid blue";
d1.style.color="yellow";
d1.style.width="600px";
d1.style.marginLeft="200px";
d1.style.background="green";
d1.style.textAlign="center";
```

```
//tworzenie elementu  - tworzymy drugi DIV
d2=document.createElement("DIV");
//ustalamy style dla nowego DIV'a
d2.style.border="4px solid red";
d2.style.margin="0px  100px  0px 100px";
d2.style.textAlign="center";
d2.style.fontSize="32px";
//tworzymy węzeł tekstowy
var t2=document.createTextNode("DRUGI DIV");
//dołaczamy  węzeł tekstowy  do div'a
d2.appendChild(t2);
//dołączamy DIV na końcu strony – dołączenie do BODY
document.body.appendChild(d2);
</SCRIPT>
</BODY></HTML>
```

Tworzenie elementu potomnego do istniejącego elementu
....uzupełnić poprzedni kod o funkcję...

```
function f2( )
{
      //tworzymy akapit P
      p1=document.createElement("P");
      p1.style.background="pink";
      p1.style.marginLeft="40px";
      p1.style.marginRight="40px";
      // i tworzymy jego węzeł tekstowy
      var tp1=document.createTextNode("Akapit potomny do DIV – dziedziczy ");
      //trzeba dołączyć węzeł tekstowy do akapitu
      p1.appendChild(tp1);
      //... i dołączyć akapit do div1
      d1.appendChild(p1);
}
```


robot

Tworzenie własnych obiektów

Słowo kluczowe this

W *JavaScript* można tworzyć własne obiekty. Wykorzystuje się tu zapis utworzonej funkcji o nazwie takiej samej jak klasa, dla której definiujemy **właściwości** (zmienne proste lub obiektowe) i **metody** (funkcje wewnętrzne).

Nadając i wykorzystując identyfikatory właściwości i metod korzystamy ze słowa kluczowego **this**. Rozumiane jest ono jako obiekt bieżący, właściciel metody lub właściwości. Poniżej prosty przykład:

Przykład

```
<HTML>
<HEAD> </HEAD>
<BODY style="font-size:18px">
<P> AKAPIT</P>
<PRE>
<SCRIPT language="JavaScript">
function kolor1( ) {
  this.document.bgColor = "red";
}
function kolor2(e) {
  e.style.color = "yellow";
}
</SCRIPT>
</PRE>
<INPUT type="button" value="Kolor dla tła okna"   onclick="kolor1( )">
<INPUT type="button" value="Kolor dla napisu na przycisku" onclick="kolor2(this)">
</BODY>
</HTML>
```

W powyższym przykładzie mamy dwie własne funkcje.

W pierwszej o nazwie *kolor1* ustalamy kolor tła dla bieżącego obiektu (**this**) - którym jest domyślnie *window*.

Druga funkcja o nazwie *kolor2* zmienia kolor czcionki przekazanego do funkcji obiektu **e** - wywoływana jest kliknięciem w przycisk z przekazaniem odniesienia do tego właśnie przycisku (**this**), a więc zmienia się kolor napisu na przycisku.

Przykład definicji obiektu i instancji

Można utworzyć definicję – tzw. konstruktor obiektu wykorzystując własną funkcję:

```
function Nazwa(argument1,argument2 )
{
  this. właściwość1 = argument1;
  this. właściwość2 = argument2;
        //itd.
}
```

Teraz tworzymy egzemplarze (**instancje**) obiektu *Pies* i przypisujemy do zmiennych referencji *p1* i *p2*:

```
var p1 = new Nazwa( argumenty aktualne);
var p2 = new Nazwa( inne argumenty aktualne);
```

Przykład

```
<PRE>
<SCRIPT language="JavaScript">
//funkcja konstruktor
function Pies(imie, rasa, kolor, plec) {
  this. imie = imie;
  this. rasa = rasa;
  this. kolor = kolor;
  this. plec = plec;
  this.glos ="HAUUUU"; //dla wszystkich psów ten sam
  this.dajGlos = function () {
              document.writeln (this.glos);
              };
};
 //tworzymy instancje
var piesek1 = new Pies( "Misia", "Labrador", "czekoladowa", "suka" );
var piesek2 = new Pies( "Misiek", "Seter", "rudy", "pies" );
//możemy mieć dostęp do właściwości i metod
document.writeln(piesek1.imie); // -> false
piesek1.dajGlos (); // ->"HAUUUU"
//itd.
</SCRIPT>
</PRE>
```

właściwości

metoda

Sposób 2

```
<PRE>
<SCRIPT language="JavaScript">
//funkcja konstruktor
 Pies2 = function (imie, rasa, kolor, plec) {
                //definiowanie obiektu – tu uwaga na interpunkcje!
        var obiektPies = {
                                imie: imie,
                                rasa: rasa,
                                kolor: kolor,
                                plec: plec,
                                glos:"HAUUU",
                                dajGlos:  function ()
        {
        document.writeln(this.glos);
        }
                };
                //funkcja zwraca zdefiniowany obiekt
                return obiektPies;
};
//tworzenie instancji klasy
var piesek3 = new Pies2( "Milka", "Spaniel", "czekoladowa", "suka" );
var piesek4 = new Pies2( "Reksio", "Dog", "czarny", "pies" );
//możemy mieć dostęp do właściwości i metod
document.writeln(piesek1.imie);
piesek1.dajGlos ();
//itd.
</SCRIPT>
</PRE>
```

W obu sposobach mamy ogólne definicje konstruktorów obiektu.

Potem możemy tworzyć wiele egzemplarzy różniących się wartościami właściwości i wykonywać metody (akcje).

Mniej uniwersalny sposób - tworzenie konkretnej instancji (egzemplarza) bez definicji:

```
var piesek1 = {
        imie: "Maja",
        rasa: "kundel",
        kolor: "zółty",
        plec: "suczka",
        glos:"haaaauuuu",
        dajGlos:  function ()
                {
                document.writeln(this.glos);
                }
  };
document.writeln(piesek1.imie);
piesek1.dajGlos ();
```

Ograniczenia dostępu do pól

```
<PRE>
<SCRIPT language=" JavaScript ">
JakisObiekt= function (n1,n2) {
    w_pryw= n1;  //właściwość prywatna
    this.w_pub=n2; //właściwość publiczna
                this.podajNazwe1 = function () {return w_pryw;         };
                this.podajNazwe2 = function () {return this.w_pub; };
};
t = new JakisObiekt("prywatna","publiczna");
document.writeln("met1:",t.podajNazwe1());//tylko metoda podaje właściwość prywatną
document.writeln("met2:",t.podajNazwe2());//metoda podaje też właściwość publiczną
document.writeln("w_pryw:",t.w_pryw);//nie ma dostępu do prywatnej - undefined
document.writeln("w_pub:",t.w_pub);// jest dostęp do właściwości publicznej
</SCRIPT>
</PRE>
```