Instrukcja 5

JAVASCRIPT

Język JavaScript posiada bardzo szerokie możliwości zastosowań. Pisane w nim skrypty mogą być wykorzystywane na stronach internetowych (jego główne zadanie), mogą stać się pełną aplikacją systemową (dzięki odpowiednim bibliotekom systemowym) oraz umożliwia tworzenie skryptów systemowych (np. w Windows jako JScript). JavaScript obecny jest także w Adobe Flash, jednak w zmodyfikowanej postaci jako ActionScript.
Pierwotnym zastosowaniem były właśnie strony internetowe. Język został opracowany przez firmę Netscape pod nazwą LiveScritpt. Ideą było „ożywienie" statycznych stron internetowych – dodanie reakcji na pewne zdarzenia, budowanie elementów nawigacyjnych strony czy też sprawdzanie poprawności wprowadzanych danych do formularzy.
Istotną cechą JavaScript dla WWW jest to, że jego funkcje wykonują się po stronie klienta (czyli ich interpretacją zajmuje się przeglądarka internetowa). Aktualnie możliwe jest wykorzystanie technologii JS do generowania strony WWW po stronie serwera, jednak w tym celu użwa się najczęściej innych, wydajniejszych języków skryptowych (np. Python, Java, Ruby, PHP i inne).
O ile w starszych edycjach HTML (do 4.01 włącznie) w dobrym tonie było projektować strony internetowe, które działały także bez języka JavaScript (użytkownik, w obawie o swoje bezpieczeństwo może wyłączyć obsługę JavaScript), o tyle w standardzie HTML5 JavaScript staje się niejako integralną częścią języka znaczników – niektóre elementy nie będą bez niego działać (np. przestrzeń canvas, niektóre pola typu input, geolokalizacja, obsługa gniazd sieciowych, przeciąganie i upuszczanie treści). Dlatego dobrze jest stworzyć rzetelną informację dla wszystkich użytkowników, że np. strona bez włączonej obsługi skryptów będzie działać niepoprawnie/zostanie wyświetlona tylko jej część.
Jak to zostało już wcześniej wspomniane, dzięki językowi JS możliwe staje się obsłużenie wszelkich zdarzeń na stronie w czasie rzeczywistym. Stronę, owszem, można generować przy użyciu języków po stronie serwera, jednak należy pamiętać, że każda zmiana na stronie wymaga odświeżenia (aby skrypt po stronie serwera mógł otrzymać dane np. z formularza i dopasować do nich treść strony). JS natomiast potrafi zmienić daną część strony „w locie", nie zmuszając serwera do ponownego przetworzenia danych Takie rozwiązania mogą być co prawda wolniejsze (gdyż działania JS zależą w głównej mierze od konfiguracji sprzętu, na którym są wykonywane) jednak stanowią bardzo dobrą alternatywę dla zapchanych łącz oraz nadmiernej eksploatacji serwera WWW (który może spowalniać przegadanie stron WWW).

1. Podstawowa składnia. System komentarzy

Jeden z najprostszych przykładów:

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8"/>
<script>//<![CDATA[
  function first() {
    document.write("<p onclick='showAlert();'>Nowy akapit tworzony za pomocą JavaScript</p>");
  }

  function showAlert() {
    alert("kliknąłeś akapit!");
  }
//]]></script>
```

```
</head>
<body >
<script>//<![CDATA[
first();
document.write("<p style='color: red;'>Dodany akapit</p>");
//]]></script>
</body>
</html>
```

Jak łatwo wywnioskować cały kod JavaScript musi być wpisywany pomiędzy specjalnymi znacznikami <script></script>. W prezentowanym kodzie znaczniki te znajdują się w dwóch miejscach – w elemencie <head> oraz w znaczniku <body>. Tak naprawdę mogą wystąpić w dowolnym miejscu i dowolną ilość razy. HTML, a w szczególności jego 5 wersja, nie narzuca żadnych limitów co do umieszczania JS.
W znaczniku <head> zwykło się deklarować funkcje pisane w JS, które później będą wykorzystywane bądź to do obsługi zdarzeń na stronie, bądź to do wykorzystania w dowolnym miejscu strony WWW.
Po pierwsze na przykładzie zostało zaprezentowane samodzielne tworzenie funkcji. Czym jest funkcja? Funkcja jest najczęściej zbiorem poleceń języka programistycznego, który w późniejszym terminie można wykorzystać w dowolnym miejscu strony dowolną ilość razy.
Pierwsza z funkcji ma za zadanie wydrukować w dokumencie HTML akapit (znacznik paragrafu) z napisem 'Nowy akapit tworzony za pomocą JavaScript'. Proszę zwrócić uwagę, że paragraf ten posiada obsługę zdarzenia onclick. Oznacza to, że jeżeli użytkownik strony kliknie na tekst mieszczący się w tym paragrafie (dowolne miejsce w obrębie paragrafu) to wykona się funkcja przypisana do tego zdarzenia (w tym wypadku showAlert).
Funkcja showAlert() z kolei ma za zadanie wyświetlić powszechnie znane okienko powiadomienia z treścią „kliknąłeś akapit".
Aby wszystko zadziałało poprawnie w dowolnym miejscu strony należy wywołać funkcję dodającą akapit – w przytoczonym przykładzie to po prostu otwarcie znacznika skryptów i wywołanie funkcji first() poprzez wypisanie jej nazwy (jedna z możliwości wywoływania funkcji JS).
Dodatkowo w znaczniku dodana została druga linijka (pod wywołaniem funkcji first), która tworzy nowy paragraf 'Dodany akapit' z czerwonym kolorem czcionki.
Jeżeli zostałaby usunięta linijka wywołująca funkcję first() to kod zawarty w tejże funkcji NIE WYKONA SIĘ. Dzieje się tak dlatego, że funkcje są zbiorami poleceń językowych, jednak jako takie nie mają możliwości istnienia w kodzie programu. Szczegółowy opis działania funkcji zostanie poruszone na następnych zajęciach.
Element CDATA (Character Data – dane znakowe) jest nowym sposobem na przesyłanie danych binarnych/surowych. Dla JavaScript jest to istotne, ponieważ jego kod posiada przeważnie dużą ilość znaków typu <, &, [,] i innych, zastrzeżonych przez specyfikację HTML. Dzięki umieszczeniu kodu skryptu w bloku:

```
<![CDATA[
//kod javaScript
]]
```

użyte w nim zastrzeżone znaki zostaną zignorowane przez parser HTML.
Trzeba mieć na uwadze, że element CDATA działa JEDYNIE w dokumencie HTML. W dokumentach XHTML będzie traktowany jak zwyczajny ciąg znakowy. W związku z powyższym, aby w programach parsujących HTML jako XHTML (bądź w dokumentach zgodnych z XHTML) blok CDATA można otoczyć dodatkowymi znakami komentarza JS:

```
<script>//<![CDATA[
        //kod javaScript
```

//]]></script>

Pomiędzy znacznikami <script></script> sekwencja znaków // oznacza komentarz w kodzie. Dlatego w przypadku gdy dokument będzie traktowany jako XHTML linia //<![CDATA[oraz //]] zostanie przez parser zignorowana (przez co uniknie się wyświetlenia tych linii na ekranie).

INFORMACJA HISTORYCZNA!!!!!
Wszystkie przeglądarki wydane po 2000 roku potrafią poprawnie rozpoznać znaczniki skryptu i ewentualnie zignorować kod JS (jeżeli zostanie wyłączony przez użytkownika). Jeżeli natomiast ktoś używał by starszej przeglądarki (co niestety potrafi się jeszcze zdarzyć) kod JS, aby nie został wyświetlony na stronie jako zwykły tekst, można otoczyć komentarzem HTML:

```
<script><!--//<![CDATA[
        //kod javaScript
//]]--></script>
```

Tym sposobem gdy przeglądarka nie jest w stanie obsłużyć kodu JS po prostu go zignoruje. Jednak każda przeglądarka zgodna z HTML5 potrafi bez najmniejszych problemów rozpoznać znaczniki <script></script>.

Skrypty JavaScript, podobnie jak znaczniki HTML i style CSS, posiadają własny system komentarzy. Kod można komentować na dwa sposobny. Jednym z nich jest wstawianie komentarzy jednoliniowych:

```
 function first() {
        //jakis komentarz
   document.write("<p onclick='showAlert();'>Nowy akapit tworzony za pomocą JavaScript</p>");
        //kolejny komentarz
   alert('ALARM'); //komentarz w linii z kodem
  }
```

Innym sposobem jest użycie komentarza wieloliniowego:

```
 function first() {
        /*początek komentarza wieloliniowego
   document.write("<p onclick='showAlert();'>Nowy akapit tworzony za pomocą JavaScript</p>");
        powyższa instrukcja NIE WYKONA SIĘ – jest w komentarza
        //kolejny komentarz
        powyższy komentarz też jest w komentarzu wieloliniowym – sekwencja // zostanie
zignorowana
        tutaj zakończy się komentarz wieloliniowy */
   alert('ALARM'); //komentarz w linii z kodem
  }
```

Jak widać komentarz ten może rozciągnąć się na dowolną ilość linii kodu.

2. Zmienne oraz typy danych.

JS, jak każdy język skryptowy wysokiego poziomu, posiada możliwość deklaracji i używania zmiennych. Zmienne to nic innego jak dane (wartości) skryptu, reprezentowane pod ustaloną przez programistę nazwą, które mogą zostać zmienione w trakcie wykonywania aplikacji. Przykładowo

jeżeli chcielibyśmy, by w zależności od kliknięty elementu wyświetlały się różne komunikaty dla użytkownika, można wykonać następujący przykład:

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8"/>
<script>//<![CDATA[


  function showAlert(variable) {
    alert("Komunikat: " + variable);
  }
//]]></script>
</head>
<body >
<p onclick="showAlert('Pierwszy element');">Pierwszy akapit</p>
<p onclick="showAlert('Drugi element');">Drugi akapit</p>
<p onclick="showAlert('Trzeci element');">Trzeci akapit</p>

</body>
</html>
```

Jeżeli użytkownik kliknie drugi akapit to w okienku informacyjnym przeglądarki pojawi się komunikat: „Komunikat: Drugi element”. Dzieje się tak dlatego, że funkcja wypisuje tekst ze zmiennej variable. Wartość zmiennej przypisywana jest jako parametr przy wywoływaniu funkcji (szczegółowe informacje na temat funkcji zostaną omówione na następnych zajęciach).

BARDZO WAŻNE – JavaScript posiada pewną ilość słów zastrzeżonych, których nie można wykorzystać jako nazwy zmiennych. Tak samo nie można rozpocząć nazwy od liczby, znaku &, %, / i innych znaków specjalnych (z wyłączeniem znaku _). Nazwa zmiennej może natomiast posiadać duże i małe litery, liczby i znaki specjalne (chociaż nie jest to zalecane). Pewną konwencją w programowaniu jest pisanie nazwy zmiennej małymi literami; jeżeli nazwa jest wieloczłonowa to w celu zwiększenia jej czytelności przyjęło się pisanie kolejnego członu nazwy dużą literą, np.:

```
var toJestWieloczlonowaNazwa;
```

WAŻNE: pomimo, iż w nazwach można używać polskich liter to jest to SILNIE NIEZALECANE – niektóre przeglądarki mogą źle zinterpretować te znaki i kod naszej aplikacji może nie zadziałać.

Oczywiście zmienne można deklarować w samym kodzie funkcji (a także zmieniać ich wartość):

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8"/>
<script>//<![CDATA[


  function showAlert() {
    var zmienna1 = 34;
    var zmienna2 = 2;
```

```
   alert ("Wynik dzielenia: " + (zmienna1 / zmienna2));
  }
//]]</script>
</head>
<body >
<p onclick="showAlert();">Kliknij na mnie!</p>

</body>
</html>
```

Jak można łatwo zauważyć, deklaracja zmiennej wymaga użycia zastrzeżonego słowa var. Zmienne da się przypisywać jedną do innej, np.

```
  function dodawanie() {
    var zmienna1 = 34;
    var zmienna2 = 2;
    var suma = zmienna1 + zmienna2;
  }
```

Jak widać zmienne można np. ze sobą dodawać (operator +), odejmować (operator -), mnożyć (operator *), dzielić (operator /) bądź uzyskać resztę z ich dzielenia (modulo, operator %). Zmienne w językach programowania posiadają swój typ. Maszynie (komputerowi) trzeba „powiedzieć" czy dany znak jest liczbą czy może zwykłym znakiem tekstowym. W przypadku JavaScript typy zmiennych są automatycznie dopasowywane przez interpreter – jeżeli z kontekstu skryptu wynika, iż zmienne powinny być traktowane jako zmienne liczbowe to tak zostaną potraktowane. Jeżeli interpreter uzna, że zmienna przechowuje typ znakowy to tak zostanie ona przez niego potraktowana.
Ogólnie JavaScript rozpoznaje następujące typy danych:

a) liczbowe – wszystkie rodzaje liczb – naturalne, ujemne, całkowite i rzeczywiste (zmiennoprzecinkowe)
Przykłady:

```
var a = 12; //liczba naturalna, liczba całkowita
var b = -3; //liczba ujemna
var c = 14.564; //liczba rzeczywista (zmiennoprzecinkowa)
```

b) typy znakowe – są to zrozumiałe dla człowieka ciągi znaków, które mogą być układane w całe zdania, akapity czy inne zbiory (przykładowo strony). Zmienne znakowe można ze sobą łączyć operatorem +
Przykłady:

```
var a= 'a'; //pojedyczny znak
var b= 'Nazwisko'; //ciąg znaków
var c=  b + ' ' + a; //zmienna c przechowuje wartość „Nazwisko a" (dodano znak białej spacji)
var d= "a"; //w JS nie ma znaczenia cz typy znakowe będą wypisane w apostrofie czy cudzysłowie
var e= "Nazwisko";
```

c) typ boolean – zmienna może przyjąć jedynie 2 wartości – prawdę (true) oraz fałsz (false). W informatyce typ ten jest bardzo często wykorzystywany do określenia zachowania kodu programu. Zmienna tego typu zajmuje najmniej pamięci operacyjnej – zapisywana jest na 1 bicie (0 to fałsz, 1 to prawda).
Przykład:

```
var a = True; //(wartość prawdziwa)
```

var b = False; //nieprawda

JavaScript dopuszcza deklarację wielu zmiennych w jednym poleceniu:

var a=12; b='45', c='Przykładowy tekst';

Istnieje możliwość samodzielnego wskazania interpreterowi jakiego typu zmienne będą przechowywane pod daną zmienną poprzez odpowiednią deklarację typu:

var a = new String; //zmienna typu znakowego (znak, ciąg znaków)
var b = new Number; //zmienna typu liczbowego (całkowita/zmiennoprzecinkowa)
var c = new Boolean; //zmienna logiczna

Ostatnim rodzajem zmiennych, bardzo pożytecznym, jest typ null. Oznacza on dosłownie pustą (nieistniejącą) wartość. JavaScript posiada dwa określenia dla tego typu: undefined oraz właśnie null. Zastosowanie typu pustego zostanie wyjaśnione wraz ze szczegółowym omówieniem roli funkcji.

3. Obiekty i ich zastosowanie.

WAŻNE! Przy tworzeniu zmiennych obiektowych używany jest atrybut new (poznany w poprzednim podpunkcie przy tworzeniu zmiennych o określonym typie). Jego zadaniem jest utworzenie zupełnie nowego obiektu i potencjalne usunięcie poprzedniego (o ile istniał pod daną zmienną).

Język JavaScript każdy typ zmiennych traktuje jako obiekt. Nawet jeżeli zmienną zadeklarujemy jako podstawowy (prmitywny) typ, to JavaScript zamieni ją w razie potrzeby na obiekt. Obiekt, jak to już zostało wcześniej wyjaśnione, to abstrakcyjny byt, który ma jak najdokładniej opisywać część świata rzeczywistego. Tak więc deklarując np. zmienną znakową:

var a = 'Tekst znakowy';

tworzymy w rzeczywistości obiekt znakowy. Dzięki temu możemy np. poznać długość zapisanego ciągu znakowego, spróbować go podzielić bądź też znaleźć interesującą nas jego część, np.:

var pos = a.indexOf('k'); //znajdzie pierwszą pozycję wystąpienia litery k i zwróci numer pozycji do zmiennej pos
var len = a.length; //zwróci długość znaków (policzy wszystkie znaki w zmiennej)
var arr = a.split(' '); //podzieli ciąg znakowy po znaku pustej spacji i każdy tak utrzowony podciąg znakowy umieści w tablic zmiennych (tablice zostaną omówione w późniejszm terminie)

Podobne właściwości posiadają pozostałe typy obiektowe. Prócz predefiniowanych typów obiektów JS oferuje możliwość tworzenia własnych obiektów. Składnia tworzonego obiektu wygląda następująco:

var ownObject = new Object();
ownObject.name = 'Moja nazwa';
ownObject.length = '56';
ownObject.favourite = True;

Powyżej został utworzony nowy obiekt o nazwie ownObject. Następnie dodane zostały do niego dodatkowe zmienne, takie jak name, lenght oraz favourite. Zmienne te będą do niego przypisane i

można się do nich odwołać w taki sam sposób jak przy ich deklaracji – z wykorzystaniem operatora . (kropka). W JavaScript do wszystkich zmiennych w obiekcie można się odwołać właśnie poprzez wspomniany operator (tzw. operator wskazania).

Predefiniowane typy obiektowe w JavaScript:
a) Number
1) pola (właściwości/zmienne):
MAX_VALUE, MIN_VALUE, NEGATIVE_ININITY, POSITIVE_INFINITY, NaN, prototpe, constructor
2) metody (funkcje):
toExponential(), toFixed(), toPrecision(), toString(), valueOf().

Przykładowe użycie:
var a = 12.45;
var b = a.MAX_VALUE; //wyświetli maksymalną dostępną wartość typu Number
var c = a.toString(); // zamieni liczbę 12.45 na ciąg znaków '12.45'

b) String
1) pola:
length, prototype, constructor
2) metody:
charAt(), charCodeAt() , concat(), fromCharCode(), indexOf(), lastIndexOf(), localeCompare(), match(), replace(), search(), slice(), split(), substr(), substring(), toLowerCase(), toUpperCase(), toString(), trim, valueOf()

Przykładowe użycie:
var a = 'To jest nowy tekst. To proste.';
var b = a.length; //zmienna przyjmie wartość liczbową odpowiadającą liczbie znaków w ciągu znakowym
var c = a.lastindexOf('To'); // poda pozycję występowania ostatniego wystąpienia wskazanej frazy
var d = a.toUpperCase(); //zamieni w ciągu znakowym wszystkie litery na duże
var e = a.replace('proste','banalne'); //zamieni znalezione ciągi znakowe 'proste' na 'banalne'
(szczegółowy opis obiektu znakowego - http://www.w3schools.com/jsref/jsref_obj_string.asp)

c) Date
1) pola:
prototype, constructor
2) wybrane metody:
getFullYear(), getTime(), setFullYear(), toUTCString(), getDay()

Przykład użycia:
var a = new Date(); //tworzy zmienną, która przechowuje aktualny czas i datę
a.setFullYear(2000,5,1); //ustawia datę na 1 czerwca 2000 roku
var b = a.getTime(); //pobiera czas zapisany w zmiennej
var c = a.getFullYear(); //zwraca rok

(szczegółowy opis daty - http://www.w3schools.com/jsref/jsref_obj_date.asp)

4. Zmienne tablicowe

Zmienne tablicowe pozwalają przechowywać wiele wartości pod postacią jednej zmiennej. Można je przyrównać do wektorów i/lub list/zbiorów znanych z matematyki. Zmienne tego typu można

zagnieżdżać. Oznacza to, że można tworzyć tzw. macierze zmiennych. Przykłady zmiennych tablicowych:

```
var a= [];
a[0] = 1;
a[1] = 2;
a[2] = 3;
a[3] = 4;
```

Zmienna tablicowa a będzie zawierać następujące zmienne: 1,2,3,4. Aby odwołać się do którejkolwiek wartości wystarczy podać nazwę zmiennej (w tym wypadku a) oraz indeks, pod którym wartość się znajduje.
Poniżej zostaną przedstawione inne możliwości utworzenia zmiennej tablicowej:

```
a = [1,2,3,4];
a = new Array(1,2,3,4);
```

Zmienne tablicowe mogą posiadać DOWOLNE nazwy indeksów. Tym samym zamiast kolejnych liczb 0,1,2..,n mogą to być np. litery alfabetu ('a', 'b',...), nazwy ('auto', 'rower', 'owoc'....) czy też mieszane (3, 'x', 45, -2, 'auto'). Nazwa poszczególnych indeksów w dużej mierze zależy od upodobań programisty oraz od potencjalnego zastosowania.
Przykład zmiennej tablicowej z mieszanymi ID:

```
var a = [];
a['x'] = „płatek”;
a[1] = „malina”;
a['auto'] = 'porshe';
a[-2] = 45.6;
```

Jak już zostało wspomniane zmienne tablicowe mogą się zagnieżdżać. Na czym to polega najlepiej zobrazuje przykład:

```
var a  = [];
var b = [3,4,5,6,7];
a[0] = b;
var c = [1,2,3,4];
a[1] = c;
var d = ['auto', 'jablko', 'moneta'];
a[2] = d;
```

W powyższym przykładzie zmienna a jest tablicą trzech zmiennych, z których każda zawiera wcześniej utworzoną inna zmienną tablicową. Tego typu zagnieżdżenia mogą się powtarzać, jednak im „głębsze” zagnieżdżenie tym większe spowolnienie działania skryptu (wzrasta jego potencjalna złożoność).
Jak odwoływać się do zagnieżdżonej tablicy:

```
document.write(a[0][1]);
```

Efektem działania takiej funkcji będzie wyświetlenie wartości 4 (pobierana jest wartość spod indeksu 0 zmiennej a, po czym następuje odwołanie do drugiej wartości b).

PRZYKŁADY:

1. Za pomocą pętli for wyświetlić wszystkie elementy tablicy

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8"/>
<script>//<![CDATA[
function switchTest() {
var tab = [1,2,3,4,5,6,7,8,9,10];
var out = "";
for(var i = 0; i < tab.length; i++)
out += tab[i] + " next ";
var x = document.getElementById("text");
x.innerHTML=out;
}
//]]></script>
</head>
<body >
<p id="text"></p><br/>
<button onclick="switchTest();">Sprawdź</button>
</body>
</html>
```

2. Posegregować wszystkie elementy tablicy w kolejności alfabetycznej

```
<!DOCTYPE html>
<html>
<body>
<p id="demo" onclick="showArray()">Elementy tablicy zostaną posegregowane (kliknij na akapit
by wyświetlić tablice)</p>
<button onclick="myFunction()">Segreguj</button>
<script>//<![CDATA[
var fruits = ["Banana", "Orange", "Strawberry", "Apple", "Mango"];
function showArray() {
var x=document.getElementById("demo");
x.innerHTML=fruits;
}
function myFunction()
{
	for (var i = 0; i < fruits.length; i++) {
		var tmp = fruits[i];
		for (var j = i; j < fruits.length; j++) {
			var cmpNumber = tmp.localeCompare(fruits[j]);
			if (cmpNumber == 1) {
				fruits[i] = fruits[j];
				fruits[j] = tmp;
				tmp = fruits[i];
			}
		}
	}
	var x=document.getElementById("demo");
	x.innerHTML=fruits;
}
//]]></script>
</body>
```

</html>

5. Wybrane funkcje drzewa BOM (Browser Object Model)

a) window – główny obiekt całego modelu BOM. Odwołuje się bezpośrednio do okna przeglądarki, w której wyświetlona została strona. Zawiera odnośniki do wszystkich aktywnych dokumentów, skryptów, zmiennych oraz pozostałych obiektów.
Z obiektu window korzysta się najczęściej do odczytania i/lub zmiany wielkości okna przeglądarki czy też otwarcia/zamknięcia dodatkowych okien (większość przeglądarek blokuje tego typu próby i ogranicza swoje działanie do poinformowania użytkownika o takowej próbie)

- window.innerWidth – zmienna przechowuje szerokość użytkową dla wskazanego dokumentu
- window.innerHeight = zmienna przechowuje wysokość użytkową dla wskazanego dokumentu

Dla Internet Explorer aż do wersji 9 by wydobyć powyższe informacje (szerokość oraz wysokość użytkową okna) należy posłużyć się drzewem DOM:

- document.documentElement.clientWidth, document.body.clientWidth – dla szerokości
- document.documentElement.clientHeight, document.body. ClientHeight – dla wysokości

-window.open() - funkcja otwiera one okno, którego rodzicem staje się aktualne okno
-window.close() - funkcja zamyka aktywne okno

Przykład użycia funkcji open():

window.open("http://www.w3schools.com","_blank","toolbar=yes, scrollbars=yes, resizable=yes, top=500, left=500, width=400, height=400");

Powyższe wywołanie otworzy nowe okno przeglądarki o szerokości wysokości 400 pikseli, odsunięte od lewej krawędzi ekranu oraz od górnej krawędzi ekranu o 500 pikseli; okno będzie posiadać pasek narzędziowy, suwaki oraz będzie umożliwiało zmianę swojego rozmiaru. W oknie wyświetlona zostanie główna strona podanego adresu WWW.
Więcej o opcjach tworzenia okna:

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

b) screen – poprzez ten obiekt otrzymujemy dostęp do wartości ustawień ekranu, takich jak rozdzielczość (w pikselach), dostępnej przestrzeni całego ekranu czy głębi kolorów. Przykład:

```
<!DOCTYPE html>
<html>
<body>

<h3>Your Screen:</h3>

<script>
document.write("Rozdzielczość ekranu: ");
document.write(screen.width + "*" + screen.height);
document.write("<br>");
document.write("Dostępna szerokość i wysokość ekranu: ");
document.write(screen.availWidth + "*" + screen.availHeight);
document.write("<br>");
```

```
document.write("Głębia kolorów: ");
document.write(screen.colorDepth);
document.write("<br>");
document.write("Głębia pikseli: ");
document.write(screen.pixelDepth);
</script>

</body>
</html>
```

c) systemy komunikacji z uzytkownikiem

-window.alert(„Treść wiadomości”) - generuje standardowy komunikat ze strony WWW (np. okno powitalne)
-window.confirm(„Treść wiadomości”) - generuje okno z przyciskami Ok oraz Anuluj. Przykład użycia:

```
<!DOCTYPE html>
<html>
<body>
<button onclick="alert()">Wciśnij mnie</button>
<p id="wynik"></p>
<script>
function alert()
{
var x;
var r=confirm("Wybierz przycisk!");
if (r==true)
  {
  x="Potwierdziłeś!";
  }
else
  {
  x="Anulowałeś!";
  }
document.getElementById("wynik").innerHTML=x;
}
</script>
</body>
</html>
```

- window.prompt(„Treść wiadomości”, „Domyśla zawartość pola edycji”) - tworzy okno z edytowalnym elementem tekstowym. Przykład użycia:

```
<!DOCTYPE html>
<html>
<body>
<button onclick="alert()">Wciśnij mnie</button>
<p id="wynik"></p>
<script>
function alert()
{
```

```
var x;
var person=prompt("Podaj swoje imię oraz nazwisko","Jan Kowalski");
if (person!=null)
  {
  x="Witaj " + person + " na przykładowej stronie";
  document.getElementById("wynik").innerHTML=x;
  }
}
</script>
</body>
</html>
```

Zadania do realizacji:
1) Proszę wstawić okienko z powiadomieniem (alertem) przy ładowaniu strony, np. „Witaj na stronie!”.
2) Proszę na każdej stronie wstawić datę oraz aktualny czas (pobrany przy ładowaniu strony). Data i czas ma wyświetlać się np. pod tytułem, albo w jednej linii z menu.
3) Proszę przetestować możliwie najwięcej metod obiektu ciągu znakowego (co najmniej 10; test ma polegać na wyświetlaniu na dowolnej stronie wyniku działania metody poprzez np. document.write lub poprzez funkcję alert).