1. Cel ćwiczenia

Mając gotowy szkielet strony WWW należy dostosować go do dzisiejszych standardów. Obecnie strona, która nie reaguje na działania odwiedzającego staje się przeważnie mniej atrakcyjna od pozostałych. Dlatego celem zajęć będzie dołączenie do istniejącego szkieletu języka JavaScript.

2. Teoria.

Język HTML nie jest językiem programowania. Brak w nim instrukcji warunkowych, pętli, zmiennych (chociaż tutaj niekiedy taką rolę pełnią atrybuty) czy funkcji. Pierwotnie HTML nie posiadał nawet zdarzeń (onclick, onchange itp.).

Nie ingerując w standard HTML firma Netscape stworzyła niezależny język programowania umożliwiający wykonanie reakcji na konkretne zdarzenia występujące w dokumencie HTML. Język ten, pierwotnie nazywany LiveScript, stał się podwaliną wysokopoziomowych parsowanych języków skryptowych ECMAScript. Obecnie najpopularniejszym przedstawicielem tej rodziny jest JavaScript – język, który dzięki swojej prostocie jest niezwykle popularny. Więcej informacji o podstawach języka JavaScript można znaleźć w materiałach podanych na końcu niniejszej instrukcji. W materiałach też omówione są podstawy składni języka, takie jak zmienne, funkcje czy zdarzenia.

INFORMACJA: W materiałach najczęściej podawane jest, że skrypty JavaScript dla WWW powinny być zamykane w dodatkowy komentarz, tj.:

```
<script>//<![CDATA[
/*tutaj dodaje się JavaScript */
//]]</script>
```

Rozwiązanie to pozwala na wyeliminowanie błędów czytania znaków zastrzeżonych dla JavaScript w parserach XML (należy pamiętać, że niektóre programy czytające dokumenty HTML mogą wykorzystywać parser XML zamiast SGML/HTML). Obecnie jednak większość z takich parserów bez problemu rozpoznaje znacznik <script> i nie traktuje znaków językowych jak znaków specjalnych XML/HTML (co miało miejsce jeszcze jakiś czas temu). Stąd komentarz CDATA nie jest potrzebny.

UWAGA! Na niektórych stronach można znaleźć przykłady dodawania kodu JavaScript,w których znacznik rozpoczynający ma dodatkowy atrybut:

```
<script language=’javascript></script>
```

lub

```
<script language=’text/javascript’></script>
```

Oba rozwiązania związane są ze starymi wersjami standardu HTML. Pierwsze wynika z pierwotnych założeń skryptów w HTML (wymagało dodatkowych pól komentarzy dla wielu przeglądarek ,które nie umiały interpretować JavaScript). Drugie pochodzi ze standardu HTML 4 (najczęściej było ono pomijane). Atrybut ten generalnie dopuszczał możliwosc wykorzystywania innych języków skryptowych (należy pamiętać, że JavaScript stał się integralną częścią HTML dopiero w wersji 5). Dlatego obecnie atrybutu language w ogóle nie trzeba używać – wszystkie przeglądarki wiedzą, jakiego języka się spodziewać. Oczywiście w przypadku użytkowania specyficznych parserów, czytających inne języki, atrybut ten nadal będzie użyteczny.

2.1 Obsługa zdarzeń w JavaScript.

Podstawowym atutem JavaScript w HTML jest możliwość obsługi zdarzeń myszy oraz klawiatury użytkownika odwiedzającego naszą witrynę. Pierwotnie zdarzenia dotyczyły jedynie myszy, do tego tylko podstawowych czynności (kliknięcia, najazdu na element, opuszczenia strefy elementu itp.).

Obecnie zdarzenia mogą dotyczyć także zmiany stanu elementu (niekoniecznie wywołanych przez użytkownika). Zdarzenia mogą być wywoływane czasowo (dzięki setTimeout oraz setInterval), przy zmianie wielkości elementów (np. okna przeglądarki), przy usuwaniu/wyłączaniu widoczności elementu/strony (onblur), przy kopiowaniu oraz przeciąganiu, a także w wersji HTML5, podczas odtwarzania audio, wideo bądź animacji (płótno JavaScript). Zdarzenia posiadają swoją pamięć, tj. każde wywołane zdarzenie tworzy obiekt, do którego zapisywane są parametry, dla których zdarzenie miało miejsce. Przykładowo jeżeli zdarzenie wywoła mysz to zapisane zostaną takie informacje jak jej położenie, kliknięty klawisz/klawisze, wykrycie wciśnięcia klawisza ALT czy element, z którym zdarzenie weszło w interakcję. Z kolei zdarzenie klawiatury zapisuje informacje o wciśnięciu klawisza (znakowego bądź id z klawiatury), czy klawisz był wciśnięty w kombinacji oraz jaki stan spowodował uruchomienie zdarzenia (wciśnięcie, odciśnięcie bądź przytrzymanie).

Zwyczajowo zdarzenia dodawane są jako atrybuty do elementów HTML. Rozwiązanie to jest o tyle wygodne, że działa we wszystkich obecnych przeglądarkach (łącznie z Internet Explorer/Edge). Jednak rozwiązanie to, przy nowych zdarzeniach, nie jest najlepszym rozwiązaniem chociażby niejako ze zmuszenia przywiązania zdarzenia do konkretnego elementu HTML. Ponadto przywiązywanie większej ilości zdarzeń do znaczników jest po prostu mało modyfikowalne (w przypadku zmiany zdarzenia trzeba przepatrywać cały dokument HTML za jego wystąpieniem), a wspomnieć należy także o czystości kodu (dodatkowe atrybuty, z potencjalnie długimi nazwami funkcji, nie wpływają pozytywnie na odnajdywanie się w kodzie). Dlatego zdarzenia można także dowiązywać poprzez odpowiednią metodę – addEventListener. W ten sposób zdarzenia można dowiązywać do każdego elementu drzewa DOM lub BOM, np.

```
window.addEventListener('resize', function() {alert("Zmieniono rozmiar okna!");});

document.addEventListener('click', function() {alert("Kliknięto w dokument!");});
```

Oczywiście zdarzenie można przywiązać do konkretnego elementu na stronie, np.

```
<p>Kliknij mnie</p>
<script>
document.getElementsByTagName('p')[0].addEventListener('click', function() {alert("Kliknięto na paragraf!");});
</script>
```

lub

```
<p>Kliknij mnie</p>
<script>
document.getElementsByTagName('p')[0].addEventListener('click', function(e) {alert("Zawartość kliknięgo paragrafu: " + e.target.innerHTML);}, true);
</script>
```

W drugim przypadku został przechwycony obiekt zdarzenia pod parametrem e. Pozwolił on na wyświetlenie zawartości elementu p.

W ostatnim przykładzie uwagę powinien zwrócić trzeci parametr. Może on przyjąć dwie wartości true bądź false (domyślnie false). Jego ustawienie pozwala na kontrolę działania zdarzenia w przypadku zagnieżdżonych elementów. Pierwotnym zamysłem twórców JavaScript było przechwytywanie tzw. bąblujące (bubbling). Charakteryzuje się ono rozchodzeniem się wywołanego zdarzenia od elementu najgłębiej osadzonego do elementów najwyższych. Obecnie dodano nowy typ obsług, który działa w odwrotną stronę – od elementu zewnętrznego do elementu najgłębiej osadzonego. Więcej informacji na ten temat można znaleźć w materiałach dodatkowych.

2.2 Funkcje i metody w JavaScript

Większość informacji o funkcjach i metodach można znaleźć w materiałach dodatkowych. W niniejszej instrukcji szczególną uwagę autor pragnie zwrócić na tzw. funkcje nienazwane (anonimowe). W poprzednich przykładach były one wykorzystane do wywołania określonego kodu w ramach dodanego zdarzenia. Generalnie funkcje te zwykło się wykorzystywać wszędzie tam, gdzie kod ich będzie wykonany tylko w jednym, konkretnym miejscu (np. jako parametr innej funkcji). Rozwiązanie to pozwala bowiem na wykonanie większej liczby linii kodu, np. z dodatkowymi parametrami (normalnie nie ma możliwości przekazać funkcji z parametrami jako parametru innej funkcji.

Ponieważ opis ten może być mało czytelny, rozważmy następujący przykład:

```
setTimeout(function() {alert('Minął określony czas!');}, 1000);
```

W powyższym przykładzie użyta została metoda okna przeglądarki, której definicja w uproszczonej dokumentacji wygląda następująco:

```
setTimeout(executeFunction, timeInMillis);
```

Jak widać pod pierwszy parametr podkładana jest funkcja/metoda, której kod ma zostać wykonany, zaś drugi parametr to czas w milisekundach, po jakim nastąpić ma wykonanie kodu funkcji.

Inną możliwością wywołania powyższej metody:

```
setTimeout(alert, 1000);
```

jednak w tym wypadku wyskakujące okienko nie będzie zawierać żadnego tekstu (wyświetli się puste). Z kolei gdybyśmy chcieli wywołać następujący kod:

```
setTimeout(alert('Minął określony czas!'), 1000);
```

to spowoduje on błąd – parser nie będzie wiedział co zrobić z kolejnym otwartym nawiasem w parametrze.

Gdyby nie było funkcji nienazwanych rozwiązanie problemu byłoby następujące:

```
function afterTimeAlert() {
	alert('Minął określony czas!');
}

setTimeout(afterTimeAlert, 1000);
```

Jednak to rozwiązanie zmuszałoby nas do napisania dodatkowej funkcji, która użyta zostałaby dokładnie jeden raz (zajmowałaby w kodzie dodatkowe miejsce, które trzeba następnie przesłać od

serwera do klienta). W związku z tym wykorzystanie funkcji anonimowych nie ma nie tylko kontekst estetyczny lecz także efektywny – zmniejsza bowiem objętość pliku z kodem.

Omawiane funkcje można też dowiązywać do zmiennych/właściwości obiektów JavaScript. Dzięki temu kod takich funkcji można wykonać za każdym razem wywołania wskazanej właściwości. Przykład:

```
<p></p>
<script>
document.getElementsByTagName('p')[0].myAnonim = function(param) {
	if (this.staticVar === undefined) this.staticVar = 0;
	if (param === undefined) param = '';
	this.innerHTML += 'Dodane przez ' + param + ' anonim w wersji ' + (++this.staticVar) +
'<br/>'};

document.getElementsByTagName('p')[0].myAnonim();
document.getElementsByTagName('p')[0].myAnonim();
document.getElementsByTagName('p')[0].myAnonim('Lucjan');
document.getElementsByTagName('p')[0].myAnonim();
</script>
```

Powyższy kod czterokrotnie wpisze do paragrafu dodany tekst + informację o numerze wywołania. To z kolei dzięki zmiennej statycznej. Proszę zauważyć, że zmienna ta została przypisana do pseudoobiektu this, którym w tym przypadku jest sama funkcja nienazwana (funkcje w JavaScript również posiadają postać obiektową!). Z kolei tworzona funkcja anonimowa może przyjąć dodatkowy parametr; jeżeli zostanie on ustawiony podczas wywołania to zostanie dodany do wyświetlanego tekstu.

INFORMACJA: Jeżeli chcemy na szybko wypróbować działanie znalezionego kodu, jednak nie chcemy eksperymentować na własnych plikach/utworzenie ich zajmie dużo czasu, można wykorzystać stronę jsfiddle.net. Pozwala ona na dodawanie kodu HTML, CSS oraz JavaScript i testowanie jego działania. Przykładowo podany powyżej przykład został zapisany pod tym odnośnikiem:

https://jsfiddle.net/81somn4g/1/

Można go otworzyć i wypróbować, a także wprowadzać zmiany, aktualizować zapis i rozsyłać go dalej.

3. Zadania do samodzielnego wykonania

Stworzoną stronę w ramach instrukcji pierwszej należy wzbogacić o rozwiązania opisane w instrukcji drugiej. Autor instrukcji nie narzuca sposobu modyfikacji strony o kod Javascript, jednak absolutnym minimum będzie:

- dodanie powitania do strony – ma ono wyświetlać się na warstwie w dolnym, prawym rogu okna przeglądarki przez 15 sekund (po czym ma zniknąć)
- dołożenie akcji wyłączenia informacji o użytkowaniu ciasteczek
- użycie ciasteczek w JavaScript, pozwalających na zapamiętanie informacji o akceptacji informacji dotyczącej ciasteczek; drugie ciasteczko powinno na godzinę zapamiętywać wizytę użytkownika by nie wyświetlać co odświeżenie strony powitania.

Atutem będzie:
- zmiana wszystkich odnośników na zdarzenia w JavaScript
- dodanie reakcji na najechanie myszą na przycisk (np. zmiana koloru)
- dołożenie funkcji obsługi ciastek na wzór PHP (odnośnik do funkcji obsługi ciastek w PHP w materiałach); konkretnie chodzi o ustawianie, odczyt i usuwanie wartości

Realizacja powyższych punktów będzie wstępem do następnego materiału.

Meteriały:
http://planetatechnika.pl/PAI/javascript.pdf
http://planetatechnika.pl/PAI/funkcje_js.pdf
http://stackoverflow.com/questions/15975786/language-javascript-vs-type-text-javascript
https://www.w3schools.com/jsref/dom_obj_event.asp
http://javascript.info/tutorial/bubbling-and-capturing
https://www.w3schools.com/jsref/met_document_addeventlistener.asp
https://www.w3schools.com/js/js_cookies.asp
https://developer.mozilla.org/en-US/docs/Web/API/Document/cookie
http://www.quirksmode.org/js/cookies.html
http://php.net/manual/en/features.cookies.php