Po prostu Internet.
Techniki zaawansowane

Autor: Tomasz Trejderowski
ISBN: 83-7197-670-4
Format: B5, stron: 122

Doskonały podręcznik dla początkujących użytkowników Internetu, którzy chcieliby poznać techniki stosowane przy tworzeniu stron WWW. Już wkrótce Kaskadowe arkusze stylów, JavaScript, dynamiczny HTML nie będą stanowiły tajemnicy. Dzięki tej książce nauczysz się tworzyć doskonałe strony WWW, które podbiją internetowe rankingi popularności.

Książka ta jest kontynuacją przewodnika „Po prostu Internet". Rozszerza informacje dotyczące tworzenia stron internetowych na komputerach klasy PC o zagadnienie Kaskadowych Arkuszy Stylów (CSS), tworzenie skryptów w języku JavaScript itworzenie formularzy na stronach WWW. Nie zabrakło również wprowadzenia do DHTML.

Niniejsza pozycja przeznaczona jest głównie dla osób, które pragną rozszerzyć swoją wiedzę zdobytą podczas pracy z książką „Po prostu Internet", jednakże nie występują w niej odwołania do tego przewodnika, więc pozycję tą można spokojnie polecić także dla osób, które wiedzę z zakresu podstaw Internetu i tworzenia stron internetowych zdobyły z zupełnie innego źródła.

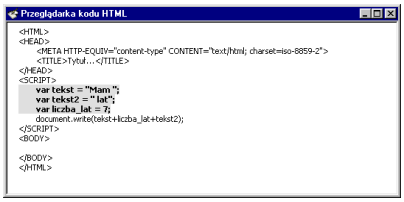
Książka, którą trzymasz w ręku, jest wyjątkowa pod wieloma względami:

- Każde omawiane zagadnienie jest przedstawiane krok po kroku, tak by nikt nie miał problemów z powtórzeniem danego rozwiązania na własnym komputerze. Każde omawiane zagadnienie jest także bardzo bogato ilustrowane – książka zwiera ponad dwieście ilustracji!
- Omawiane problemy są wzbogacane o zestawienia tabelaryczne najczęściej wykorzystywanych funkcji, obiektów czy stylów.
- Autor skupia się na zgodności z obowiązującym od kilku lat standardem HTML 4.0 –czytelnik w trakcie lektury już teraz nabywa wiedzę z zakresu, który będzie obowiązywał w przyszłości, od momentu, gdy wiodące przeglądarki internetowe wprowadzą kolejne rozwiązania zdefiniowane przez twórców HTML 4.0
- Autor książki udostępnił swój prywatny adres e-mail. Możesz wykorzystać go do wyrażenia własnych opinii na temat tej książki, ale przede wszystkim, by uzyskać odpowiedź, której być może nie udało Ci się odnaleźć w książce.

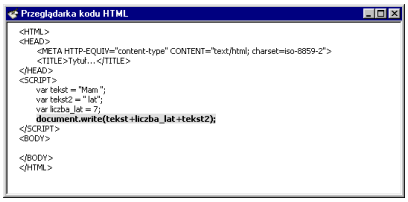
Wydawnictwo Helion
ul. Chopina 6
44-100 Gliwice
tel. (32)230-98-63
e-mail: helion@helion.pl

Spis treści

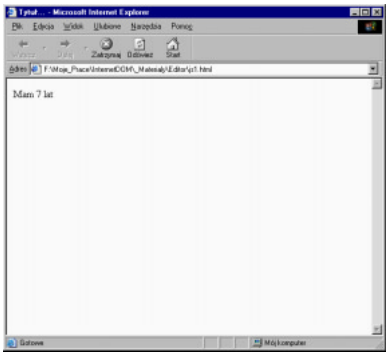
Podstawy JavaScript 5



Rysunek 5.1. *Różne rodzaje zmiennych*



Rysunek 5.2. *Przykład mieszania różnych typów zmiennych w jednym poleceniu języka JavaScript*



Rysunek 5.3. *Efekt działania powyższego kodu w przeglądarce*

Zmienne

Najprościej mówiąc, zmienna to zasobnik (ang. *container*), jednoznacznie określony unikalną nazwą, przechowujący w sobie jakąś wartość. Język *JavaScript* rozróżnia cztery podstawowe typy zmiennych:

- liczby (różne formaty zapisu),
- łańcuchy tekstowe,
- wartości logiczne,
- *null*.

Poniżej pokrótce omówiona została każdy z powyższych typów zmiennych.

Należy zwrócić uwagę na fakt, który nie występuje w większości popularnych języków programowania. Otóż, w *JavaScript* wprowadzone zostały operatory dodawania do siebie zmiennych bez względu na ich typ. To znaczy możliwe jest dodanie do zmiennej typu, np. łańcuch tekstowy innej zmiennej typu np. liczba, bez konieczności zamiany któregoś z typów zmiennej na ten sam jak druga zmienna (rysunki 5.1 – 5.3).

Liczby

JavaScript nie rozróżnia typu liczby — zmienna liczbowa może przechowywać wszystkie liczby zmiennoprzecinkowe i całkowite, bez względu na to czy są to liczby rzeczywiste, dodatnie, ujemne, a nawet bez względu na to czy są one zapisane w systemie dziesiętnym, ósemkowym czy szesnastkowym. Dokładniej obrazuje to poniższa tabela.

Typ zapisu liczby:	Przykłady:	Uwagi:
Liczby zmiennoprzecinkowe (zapis standardowy)	3.1415	—
Liczby zmiennoprzecinkowe (zapis naukowy)	314e-2, 78E17	Jest to tak zwana „eksponenta”.
liczby całkowite	77	—
Liczby w systemie ósemkowym	013	Liczbę poprzedzamy cyfrą 0 (zero), by odróżnić ją od tej samej zapisanej w systemie dziesiętnym.
Liczby w systemie szesnastkowym	0xFF, 0XE3	j.w. przy czym liczbę poprzedza się kombinacją 0x (zero-iks); wielkość znaku x nie ma znaczenia.

Należy pamiętać, że *JavaScript* korzysta ze standardów amerykańskich, gdzie do oddzielenia części ułamkowej używa się znaku kropki (.), a nie przecinka (,)!

Jeżeli nie znasz różnic ani metod przeliczania liczb zapisanych w systemach dziesiętnym, szesnastkowym i ósemkowym — poszukaj w dowolnym podręczniku podstaw informatyki, tam znajdziesz dokładne przedstawienie tego zagadnienia.

Łańcuchy tekstowe

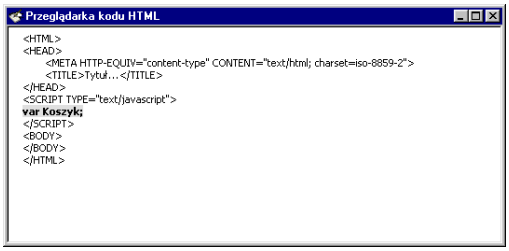
Zmienne typu łańcuch tekstowy (zawierające jeden lub więcej znaków wprowadzonych z klawiatury) odróżnia od innych typów zmiennych znak podwójnego cudzysłowu (") występujący na początku i końcu. Taka zmienna jest traktowana jako tekst bez względu na zawartość, czyli zarówno `"Misio"` jaki i `"56123"` są traktowane jako tekst mimo, że drugi przykład mógłby sugerować coś innego. Kombinacja dwóch cudzysłowów (`""`) — czyli tzw. łańcuch pusty, także jest traktowany jako tekst. W szczególnych przypadkach możliwe jest zastąpienie pary cudzysłowów podwójnych przez parę cudzysłowów pojedynczych (`' '`), ale *W3C* zaleca niedopuszczanie do takich sytuacji.

Wartości logiczne

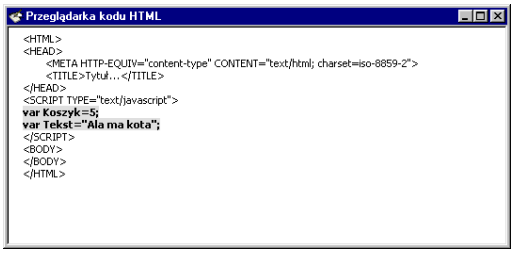
Wartość logiczna to typ pochodny od zmiennej liczbowej, lecz może on przyjmować tylko dwie wartości: prawda (`true`) i fałsz (`false`).

Typ null

To specjalny typ pusty, rzadko wykorzystywany w skryptach. Nie należy mylić go ani z wartością 0 (jest to zmienna typu liczba) ani z `""` (jest to zmienna typu łańcuch tekstowy).



Rysunek 5.4. *Przykład deklarowania zmiennej bez określania jej typu*



Rysunek 5.5. *Deklarowanie zmiennej z jednoczesnym określeniem jej typu oraz nadaniem jej wartości*

Deklarowanie zmiennych

Jeżeli wykorzystujesz zmienną w więcej niż jednym miejscu w skrypcie, musisz ją zadeklarować. Dzięki temu będzie ona „widoczna” w całym kodzie, to znaczy — jej wartość w danej chwili będzie taka sama w każdym miejscu skryptu.

Aby zadeklarować zmienną, poprzedź jej nazwę zastrzeżonym słowem `var`, a po jej nazwie wpisz znak średnika (jest to uniwersalny znak, służący do rozdzielania kolejnych poleceń języka *JavaScript*). Przykład takiej deklaracji jest przedstawiony na rysunku 5.4.

Możesz także od razu nadać zmiennej wartość, a tym samym poinstruować interpreter *JavaScript* z jakim typem zmiennej ma on do czynienia w przypadku konkretnej nazwy. Aby dokonać takiej deklaracji, pomiędzy nazwą zmiennej, a znakiem średnika wstaw znak równości (przypisania) oraz podaj wartość zgodną z typem zmiennej jaki chcesz zadeklarować. Przykłady takich deklaracji znajdują się na rysunku 5.5.

Umieszczanie skryptu JavaScript na stronie WWW

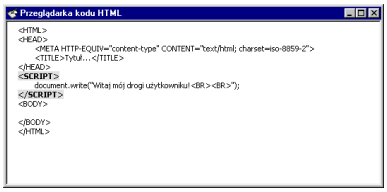
Mówimy teraz o skryptach wyzwalanych automatycznie przez przeglądarkę w czasie ładowania zawartości strony. W odróżnieniu od nich, są jeszcze skrypty wyzwalane jako efekt zajścia jakiegoś zdarzenia (bo jak pamiętasz *JavaScript* jest językiem obiektowo-zdarzeniowym), ale o nich będzie mowa później.

Są dwa sposoby umieszczenia skryptu wyzwalanego automatycznie, w dokumencie *HTML*.

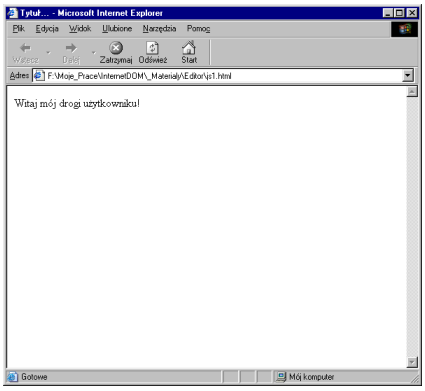
Osadzanie skryptu bezpośrednio w dokumencie HTML.

Pierwszym z nich jest wstawienie kolejnych poleceń pomiędzy znacznikami <SCRIPT> i </SCRIPT>. Są dwie szkoły. Jedna z nich twierdzi, że polecenia języka *JavaScript* wraz z powyższymi znacznikami należy umieszczać pomiędzy sekcjami <HEAD> oraz <BODY>. Druga szkoła twierdzi, że skrypty należy umieszczać w sekcji <BODY>. Wybór należy do Ciebie — w obu przypadkach przeglądarka powinna wykonać skrypt bez błędów (rysunki 5.6 i 5.7).

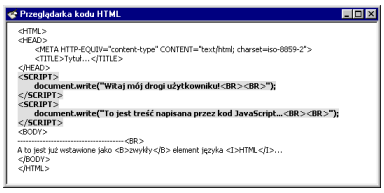
Możesz mieć dowolną ilość skryptów osadzonych w treści dokumentu *HTML*, zostaną one wykonane po kolei, tak jak kolejność ich umieszczania w dokumencie (rysunki 5.8 i 5.9).



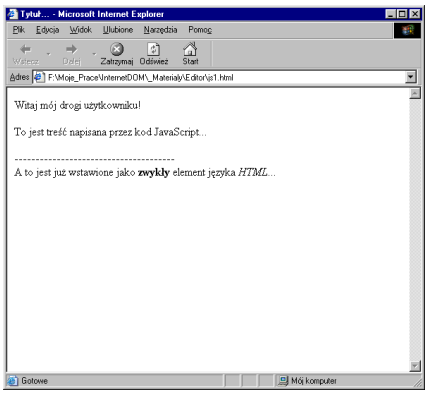
Rysunek 5.6. *Sposób umieszczenia najprostszego skryptu w treści dokumentu HTML...*



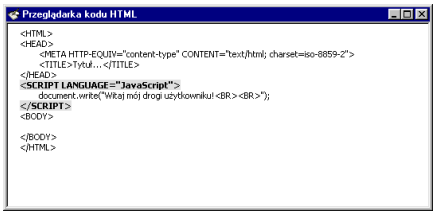
Rysunek 5.7. *...oraz efekt takiego działania, zaprezentowany w przeglądarce*



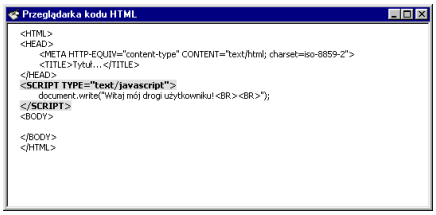
Rysunek 5.8. *Możesz mieć dowolną ilość skryptów osadzonych w dokumencie HTML*



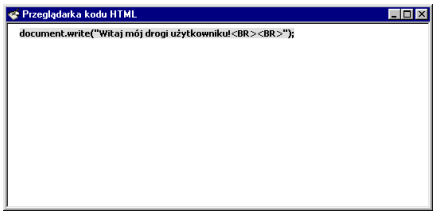
Rysunek 5.9. *Zostaną one wykonane po kolei tak jak były umieszczone przez twórcę*



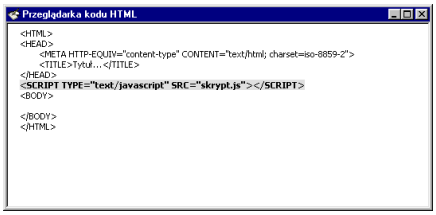
Rysunek 5.10. *Poprzednia metoda deklaracji typu zastosowanego języka skryptowego*



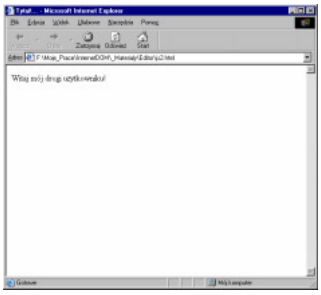
Rysunek 5.11. *Metoda deklaracji wprowadzona w HTML 4.0*



Rysunek 5.12. *Kod JavaScript zapisany w osobnym pliku o rozszerzeniu *.js*



Rysunek 5.13. *Odwołanie do zewnętrznego skryptu w treści dokumentu HTML*



Rysunek 5.14. *Efekt powyższych operacji w przeglądarce*

Jak widać na rysunkach 5.6 – 5.9 skrypty zostały wykonane bezbłędnie bez informowania przeglądarki z jakim językiem programowania ma do czynienia (bo *JavaScript* nie jest jedynym językiem, który można umieszczać na stronach *WWW*). Mimo to *W3C* zaleca by umieszczać w znaczniku `<SCRIPT>` informację o tym. Możesz to zrobić albo według starej metody, poprzez użycie atrybutu `LANGUAGE` tegoż znacznika (tak jak na rysunku 5.10). Zalecane jest oczywiście stosowanie nowej metody (wprowadzonej wraz z *HTML 4.0*), która została przedstawiona na ilustracji 5.11.

Wywoływanie zewnętrznego pliku

Jeżeli na jednej stronie *WWW* wykorzystujesz większą ilość skryptów *JavaScript*, ich umieszczenie bezpośrednio w treści dokumentu może spowodować niezły bałagan. Warto wtedy pomyśleć o „wyrzuceniu" ich do zewnętrznego pliku. Rozwiązanie takie jest również bardzo dobre, jeżeli na kilku różnych stronach wykorzystujesz te same skrypty. To dokładnie ta sama sytuacja jak z omawianymi wcześniej stylami — jeżeli będziesz chciał zmienić fragment kodu, zmienisz go w zewnętrznym pliku, a wszystkie strony odwołujące się do tego pliku automatycznie uwzględnią te zmiany. Stosując poprzednią metodę musiałbyś aktualizować każdy plik zawierający kod *JavaScript*.

Aby skorzystać z tej metody, zapisz cały kod w osobnym pliku. Nadaj mu dowolną nazwę oraz rozszerzenie *.js. Następnie w znaczniku `<SCRIPT>` dodaj nowy atrybut `SRC`, a jako jego wartość nazwę pliku, w którym przed chwilą zapisałeś cały kod *JavaScript*. Zostało to przedstawione na rysunkach 5.12 – 5.14.

Problematyka ścieżek dostępu (dla przypadku, kiedy plik zawierający skrypt nie znajduje się w tym samym katalogu co dokument *HTML*) została omówiona w mojej poprzedniej książce zatytułowanej „Po prostu Internet", która ukazała się nakładem wydawnictwa Helion w grudniu 2001 roku.

Funkcje

Deklarowanie funkcji

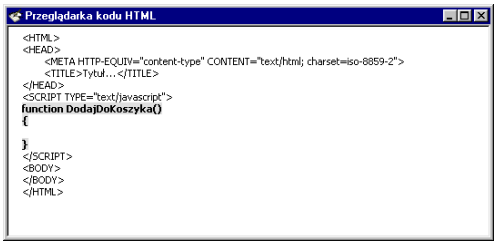
Jak każdy język programowania, także *JavaScript* umożliwia programiście tworzenie funkcji. Funkcje to wyraźnie ograniczone bloki kodu, które są wywoływane w innych częściach tego kodu. Mogą one być wywoływane „tak jak są" lub z określoną listą parametrów, których wartość można odczytać wewnątrz takiej funkcji. Sama funkcja może też zwrócić jakąś wartość.

Dzielenie kodu na funkcje jest niezbędne, jeżeli planujesz wywoływać różne jego elementy jako reakcję na określone zdarzenia (patrz dalej).

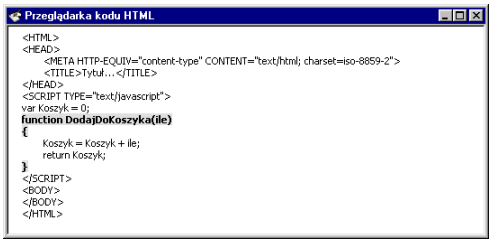
Aby dodać funkcję do kodu *JavaScript*, w dowolnym jego miejscu użyj zastrzeżonego słowa `function`, po nim wstaw znak odstępu i podaj nazwę funkcji (nie można ona zawierać odstępów ani znaków specjalnych), a na koniec kombinację dwóch nawiasów `()`. Następnie w nowej linijce wstaw nawias klamrowy lewy ({). Teraz możesz (albo w jednej linijce, albo w kilku) wpisać cały kod tej procedury. Definicję procedury zakańczasz wstawiając nawias klamrowy prawy (}). Przykład takiej deklaracji znajduje się na rysunku 5.15.

Zgodnie z tym co napisałem wcześniej, w *JavaScript* możesz każdej funkcji przyporządkować jeden lub więcej argumentów, a następnie korzystać z nich wewnątrz tej funkcji tak jak korzysta się z każdej innej zmiennej. Pamiętaj, że jeżeli nie zadeklarujesz zmiennej na początku skryptu to jej wartość wewnątrz funkcji będzie inna niż poza funkcją. Jeżeli używasz więcej niż jednego argumentu — i w deklaracji funkcji i w ewentualnym odwołaniu do niej oddziel je znakiem przecinka (,).

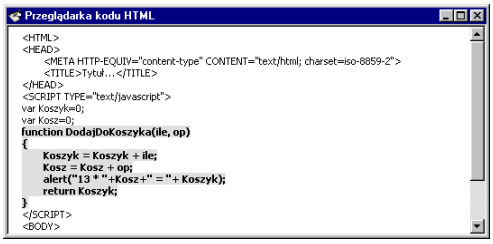
Użyj zastrzeżonego słowa `return`, aby określić jaką wartość dana funkcja ma zwrócić. Taką zwróconą wartość możesz później (traktując ją jak zwykłą zmienną) wykorzystać w innej części skryptu — np. w innej funkcji (rysunki 5.16 – 5.18).



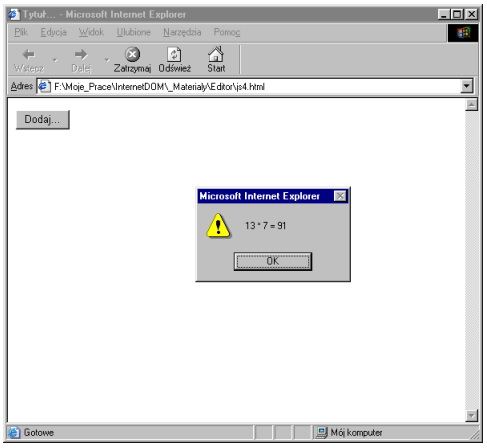
Rysunek 5.15. *Przykład podstawowej deklaracji funkcji języka JavaScript*



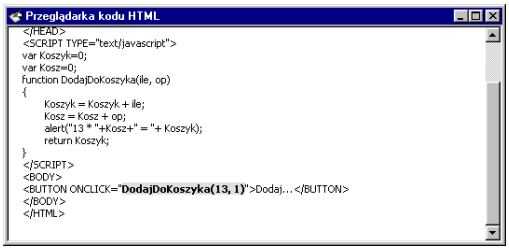
Rysunek 5.16. *Przykład funkcji z jednym parametrem, która zwraca określoną wartość*



Rysunek 5.17. *Funkcja korzystająca z dwóch argumentów*



Rysunek 5.18. *Efekt działania powyższego skryptu*



Rysunek 5.19. *Przykład wywołania zadeklarowanej wcześniej funkcji*

Wywoływanie zadeklarowanej funkcji

To proste! Po prostu w miejscu, w którym tego potrzebujesz napisz nazwę funkcji, a w nawiasach wartości jakie chcesz nadać jej konkretnym parametrom (o ile dana funkcja posiada jakieś parametry). Najczęściej wywoływanie określonych funkcji stosuje się w przypadku zdarzeń, o czym dalej (rysunek 5.19).

Zdarzenia

Podstawy

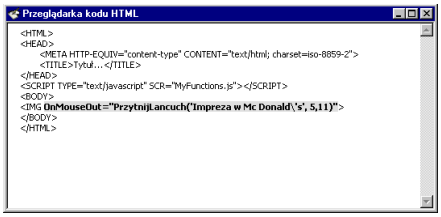
Pewna grupa znaczników umożliwia obsługę zdarzeń. Zdarzenia to pewne procesy zachodzące w związku z zachowaniem się użytkownika na Twojej stronie internetowej — ruchem myszki po ekranie, klikaniem na różnych elementach czy wypełnianiem pól formularzy. Aby dodać do swojej strony internetowej obsługę jakiegoś zdarzenia, po pierwsze zdecyduj się jaki obiekt (znacznik) ma to zdarzenie obsługiwać. Następnie dodaj mu atrybut, który będzie nazywał się tak jak jedna z wymienionych poniżej, zastrzeżonych nazw funkcji, dodaj znak równości, a następnie w cudzysłowie wpisz nazwę zadeklarowanej wcześniej funkcji wraz ze wszystkimi argumentami. (rysunki 5.20 – 5.22)

A co zrobić w sytuacji, gdy parametr wywoływanej funkcji musi być podany w cudzysłowach, gdyż zmienna jest typu łańcuch tekstowy? Otóż w tej sytuacji możesz zastosować cudzysłowy pojedyncze (' i '), by podać wartości zmiennych typu łańcuch tekstowy. Zaś samo wywołanie zdarzenia wraz z nazwą funkcji (wartość określonego atrybutu) bez zmian podajesz w cudzysłowach podwójnych (" i "). (rysunek 5.23)

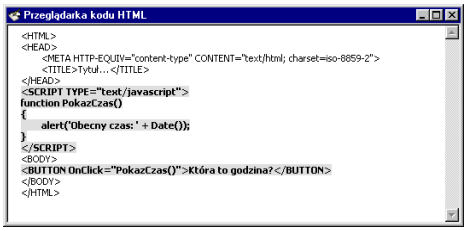
Jeżeli natomiast musisz użyć znaku apostrofu (cudzysłowu pojedynczego) jako elementu argumentu typu łańcuch tekstowy, poprzedź go znacznikiem odwrotnego ukośnika. Dzięki temu unikniesz błędów przy wykonywaniu skryptu. (rysunek 5.24)

Lista zdarzeń

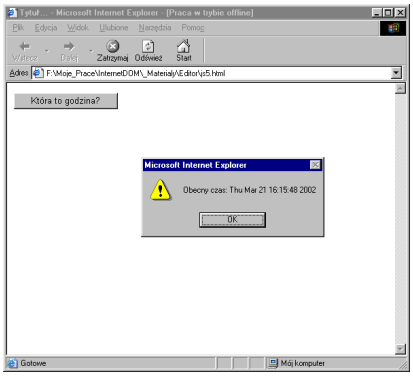
W *JavaScript* możesz pisać skrypty, których funkcje będą reagowały na jedną spośród 18 niżej opisanych funkcji.



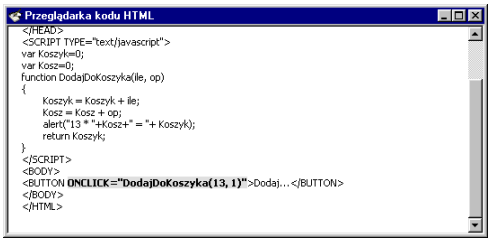
Rysunek 5.24. *Sposób radzenia sobie w przypadku, gdy wewnątrz łańcucha tekstowego niezbędne jest użycie apostrofa*



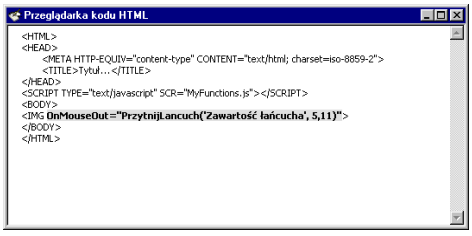
Rysunek 5.20. *Przykładowy skrypt obrazujący wykorzystanie zdarzeń*



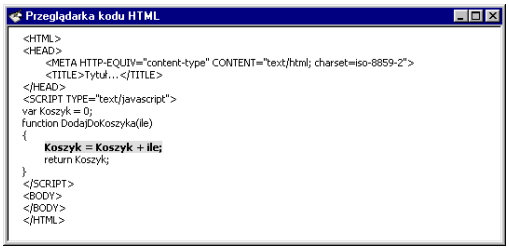
Rysunek 5.21. *Efekt działania powyższego skryptu w przeglądarce*



Rysunek 5.22. *Znacznik z zadeklarowanym zdarzeniem (kliknięcie); w przypadku jego zajścia zostanie wywołana funkcja JavaScript zadeklarowana we wcześniejszym bloku <SCRIPT> </SCRIPT>*



Rysunek 5.23. *Wywołanie funkcji (jako reakcji na zajście zdarzenia) z argumentami typu łańcuch tekstowy — wykorzystano pojedyncze i podwójne cudzysłowy*



Rysunek 5.25. *Przykłady operatorów w JavaScript, pierwszy jest operatorem przypisania, drugi operatorem matematycznym*

Operatory

Opisane wcześniej zmienne mogą przyjmować określoną wartość, ale aby można było tę wartość im przypisać lub zmodyfikować, niezbędne staje się wykorzystanie operatorów — sekwencji zastrzeżonych znaków, których zaistnienie w dowolnym miejscu kodu powoduje zmianę wartości zmiennej. Operatory w *JavaScript* zostały podzielone na kilka grup (rysunek 5.25).

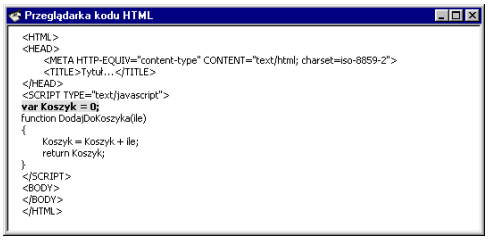
Zdarzenie	Moment zajścia	Przykładowe znaczniki
OnClick	Użytkownik kliknął klawiszem myszki na danym obiekcie.	<BUTTON>, <A>, <IMG>
OnDblClick	j.w. — dwukrotne kliknięcie.	j.w.
OnMouseDown	Użytkownik wskazał dany obiekt wskaźnikiem myszki i nacisnął klawisz, lecz nie zwolnił go.	j.w.
OnMouseUp	Występuje bezpośrednio po poprzednim zdarzeniu — użytkownik zwolnił klawisz myszki.	j.w.
OnMouseOver	Użytkownik wskazał myszką dany obiekt.	j.w.
OnMouseMove	Użytkownik przesuwa myszkę nad danym obiektem.	j.w.
OnMouseOut	Wskaźnik myszki przemieścił się poza obszar danego obiektu.	j.w.
OnLoad	Zachodzi w momencie rozpoczynania rysowania zawartości strony na podstawie jej kodu *HTML*.	Tylko <BODY> i <FRAMESET>
OnUnLoad	Zachodzi w momencie opuszczenia danej strony przez przeglądarkę celem udania się pod nowy adres.	j.w.
OnSelect	Zachodzi w momencie zaznaczenia przez użytkownika jakiegoś fragmentu tekstu w obiekcie, w którym można pisać.	Tylko elementy formularza — znaczniki <INPUT>
OnFocus	Zachodzi w momencie uaktywnienia (poprzez użycie klawisza Tab lub wykorzystanie myszki) jakiegoś elementu formularza.	j.w.
OnBlur	Zachodzi, gdy dany aktywny element formularza przestaje być aktywny, a zaznaczenie (focus) przemieszcza się na inny obiekt.	Tylko wybrane elementy formularza — znaczniki <INPUT>
OnKeyPress	Użytkownik wpisuje coś w elementach formularza, które umożliwiają wprowadzanie tekstu.	j.w.
OnKeyDown	Użytkownik nacisnął klawisz na klawiaturze, ale nie zwolnił go.	j.w.
OnKeyUp	Występuje bezpośrednio po poprzednim zdarzeniu wskutek zwolnienia naciśniętego klawisza.	j.w.
OnChange	Zmiana zawartości elementu umożliwiającego wprowadzanie znaków.	j.w.
OnSubmit	Wysłanie formularza	Tylko znacznik <FORM>
OnReset	Wyczyszczenie zawartości formularza	j.w.

Operatory matematyczne

Przez niektórych bywają nazywane operatorami arytmetycznymi — dzięki nim dokonuje się operacji na liczbach.

Operatory przypisania

Jak nazwa wskazuje, są to operatory przypisujące konkretną wartość (podaną jako argument po prawej stronie operatora) zmiennej (znajdującej się po lewej stronie operatora) (rysunek 5.26).



Rysunek 5.26. *Nawet zwykłe zadeklarowanie zmiennej z jednoczesnym określeniem jej typu i nadaniem początkowej wartości, wiąże się z wykorzystaniem operatora przypisania*

Operator	Działanie	Przykłady
+	dodawanie dwóch liczb	5+11 s=11+b
-	odejmowanie dwóch liczb	7-3
*	mnożenie dwóch liczb	7*3
/	dzielenie dwóch liczb	7/3
%	reszta z dzielenia	7%3
++	zwiększenie wartości o jeden	t++
--	zmniejszenie wartości o jeden	t--

Operator	Działanie	Przykłady	Dotyczy
=	standardowe przypisanie	t=11 ala="ma kota"	wszystkich typów zmiennych
+=	tak jak a=a+b	a+=b	tylko liczb
-=	tak jak a=a-b	a-=b	tylko liczb
*=	tak jak a=a*b	a*=b	tylko liczb
/=	tak jak a=a/b	a/=b	tylko liczb
%=	tak jak a=a%b	a%=b	tylko liczb

Operatory porównania

Różnią się one tym od pozostałych operatorów, że porównują (w określony sposób) dwie wartości (podane z lewej i prawej strony operatora) i zwracają wartość logiczną typu prawda (`true`) lub fałsz (`false`) w zależności od tego, czy zadany warunek porównania jest spełniony czy nie. Operatory te najczęściej wykorzystuje się w instrukcjach warunkowych i pętlach, o czym mowa w dalszej części rozdziału.

Operatory logiczne

W odróżnieniu od poprzednich, te operatory powodują **nadanie** argumentowi znajdującemu się po lewej stronie operatora konkretnej wartości logicznej typu prawda (`true`) lub fałsz (`false`).

Operatory porównania

Operator	Opis	Wynik działania
`!`	negacja, zaprzeczenie	`false`, jeśli było `true` lub odwrotnie.
`&&`	koniunkcja	Wynikiem jest `true`, jeśli oba argumenty są `true`, w każdym innym przypadku wynikiem jest `false`.
`||`	alternatywa	Wynikiem jest `false`, jeżeli oba argumenty są `false`, w każdym innym przypadku wynikiem jest `true`.

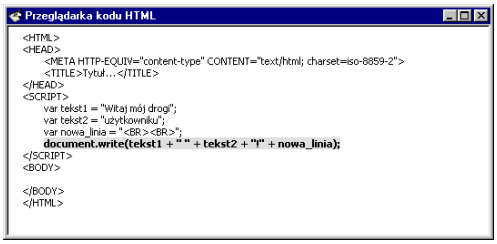
Operatory logiczne

Operator	Sprawdzany warunek	Przykłady
`==`	Czy oba wyrażenia są równe?	`a==b` `r==3`
`!=`	Czy oba wyrażenia nie są równe?	`a!=b`
`<`	Czy lewe wyrażenie jest mniejsze od prawego?	`a<b`
`>`	Czy lewe wyrażenie jest większe od prawego?	`a>b`
`<=`	Czy lewe wyrażenie jest mniejsze lub równe prawemu?	`a<=b`
`>=`	Czy lewe wyrażenie jest większe lub równe prawemu?	`a>=b`

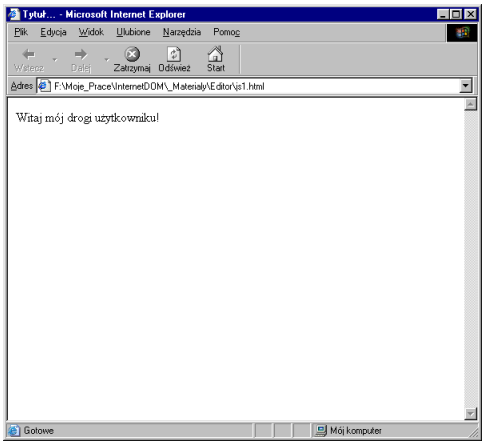
Operator konkatenacji

Konkatenacja to trudne słowo, przez które należy rozumieć po prostu łączenie tekstów. Nie chodzi tu o nowy operator, a jedynie o to, że jeżeli użyjesz operatora dodawania (+) dla dwóch zmiennych typu łańcuch tekstowy, to w efekcie uzyskasz jedną zmienną typu łańcuch tekstowy, której wartość będą stanowiły połączone dwa łańcuchy poddane operacji konkatenacji (rysunki 5.27 i 5.28).

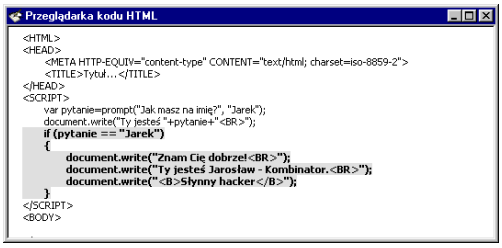
Tak jak wspominałem wcześniej, możesz połączyć przy użyciu operatora + dwie zmienne różnych typów (np. łańcuch tekstowy i liczbę) i nie spowoduje to wygenerowania komunikatu o błędzie, gdyż *JavaScript* automatycznie konwertuje typ zmiennej na pożądany.



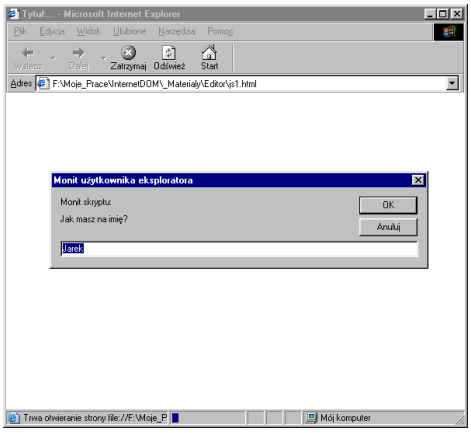
Rysunek 5.27. *Przykład użycia operatora konkatenacji*



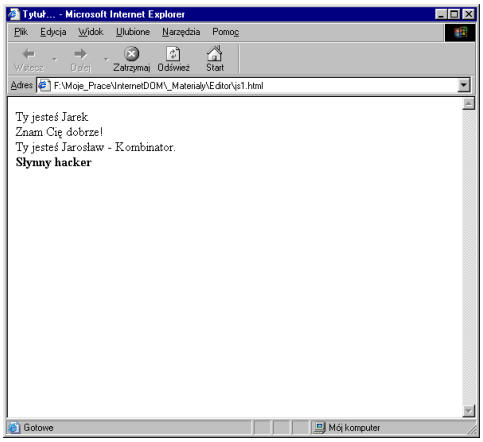
Rysunek 5.28. *Efekt działania powyższego skryptu w przeglądarce*



Rysunek 5.29. *Fragment skryptu zawierającego instrukcję warunkową*



Rysunek 5.30. *Uruchomienie skryptu w przeglądarce powoduje wyświetlenie monitu z prośbą o podanie imienia — oczywiście możesz wpisać co tylko ci się żywnie podoba...*



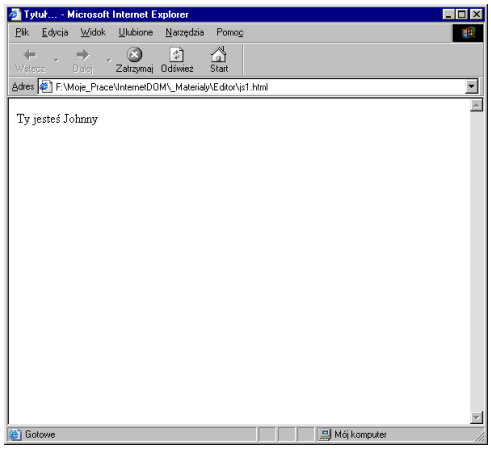
Rysunek 5.31. *Efekt działania skryptu w przypadku, gdy warunek zadany w instrukcji warunkowej if został spełniony*

Instrukcje warunkowe

Jest to jedna z podstaw programowania (pisania skryptów), gdyż umożliwia sterowanie przepływem programu — w zależności od tego czy określony warunek jest spełniony czy nie, wykonywane są odpowiednie partie skryptu. W *JavaScript* możesz korzystać z kilku typów instrukcji warunkowych — najważniejsze z nich zostały pokrótce opisane poniżej.

if

Najprostsza w użyciu i chyba przez to najpopularniejsza instrukcja warunkowa. Przykład wykorzystania tego rodzaju instrukcji warunkowej w skrypcie, został przedstawiony na rysunkach 5.29 – 5.32. Interpreter *JavaScript* sprawdza, czy warunek podany w nawiasach okrągłych zwraca wartość `true` i jeśli tak to wykonuje polecenia zawarte w nawiasach klamrowych ({ i }), po czym kontynuuje wykonywanie skryptu od następnej linii kodu. Jeżeli zadany warunek zwraca wartość `false` wszystkie polecenia w nawiasach klamrowych są ignorowane, a wykonywanie skryptu jest kontynuowane od pierwszej linii kodu po zamykającym nawiasie klamrowym (rysunki 5.29 – 5.32).



Rysunek 5.32. *Efekt działania skryptu w przypadku niespełnienia zadanego warunku*

if... else

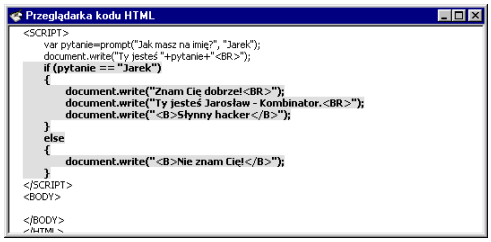
Jest to rozbudowana wersja poprzedniej instrukcji warunkowej. Jedyną różnicą jest dodany element `else`, który oddziela (również nawiasami klamrowymi) instrukcje, które będą wykonane **wyłącznie** w przypadku, gdy zadany warunek nie jest spełniony. Popatrz na rysunki 5.33 i 5.34.

?

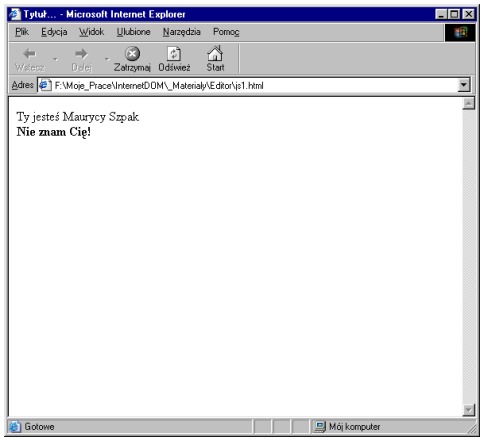
Uproszczona (skrócona) wersja poprzedniej instrukcji warunkowej, przeznaczona dla rozwiązań w których zarówno spełnienie jaki i niespełnienie zadanego warunku spowoduje wykonanie **tylko jednej** (za każdym razem) instrukcji. Ogólna konstrukcja:

```
(warunek) ? instrukcja_jesli_prawda :
instrukcja_jesli_fałsz;
```

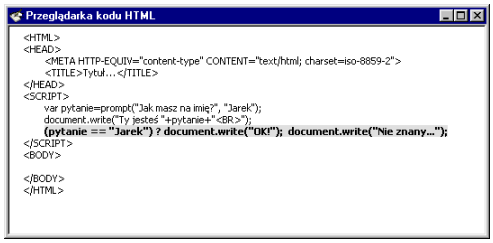
Przykład użycia w skrypcie jest na ilustracji 5.35.



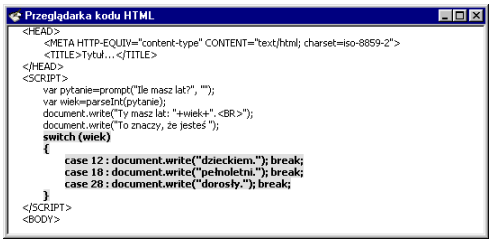
Rysunek 5.33. *Przykład skryptu wykorzystującego instrukcję warunkową if ... else*



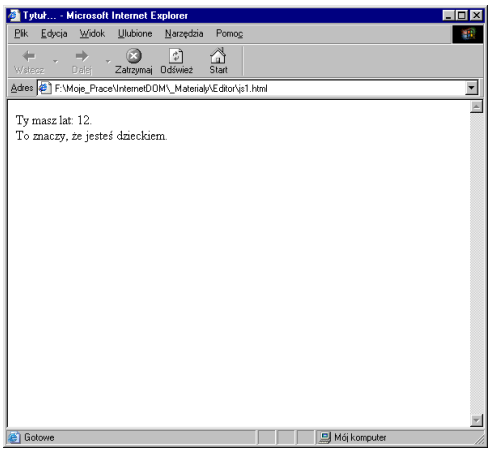
Rysunek 5.34. *Niespełnienie zadanego warunku powoduje wykonanie poleceń, które w żadnym innym przypadku nie zostałyby wykonane*



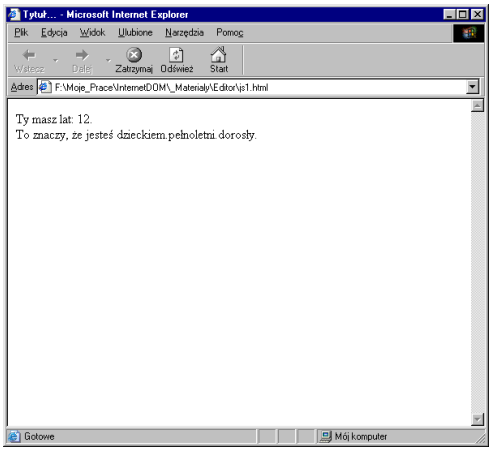
Rysunek 5.35. *Uproszczona wersja instrukcji warunkowej if... else*



Rysunek 5.36. *Wykorzystane instrukcji warunkowej switch*



Rysunek 5.37. *Efekt działania powyższego skryptu w przeglądarce*



Rysunek 5.38. *Efekt działania tego samego skryptu w przypadku niezastosowania polecenia break przerywającego działanie całego bloku switch*

switch

To rozbudowana wersja instrukcji warunkowej, która pozwala na analizę jednocześnie kilku możliwych wartości, zwracanych przez badanie danego warunku. Jak pamiętasz, poprzednie instrukcje warunkowe badały tylko zwrócenie `true` lub `false`. W przypadku tej instrukcji możesz badać np. różne wyniki liczbowe. Przykład zastosowania tej instrukcji warunkowej przedstawiono na ilustracji 5.36. Pamiętać należy, że w przypadku napotkania sekwencji `case` po której zadany warunek jest spełniony, wykonywane są **wszystkie** następujące instrukcje. Aby ograniczyć takie działanie, należy przerywać wykonywanie danego bloku skryptu dyrektywą `break`, gdyż w przeciwnym przypadku efekt końcowy może być inny niż pożądany — przykład na ilustracji 5.38.

Pętle

Pętle bywają czasami nazywane „blokami poleceń zapętlonych”. Jest to element, bez którego budowanie sprawnie działających skryptów byłoby niemożliwe lub bardzo trudne. Pętle to blok poleceń, które są wykonywane określoną bądź nieskończoną ilość razy. Stosowanie pętli zwiększa czytelność i przejrzystość skryptu, nie mówiąc o sytuacjach, w których ich zastosowanie jest wręcz niezbędne. Poznasz dwa rodzaje pętli.

for

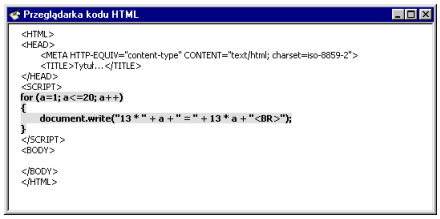
Jest pętlą skończoną. Ogólna postać:

```
for (zmienna, warunki_przerwania,
zmiana_wartosci_zmiennej)
{
polecenia;
}
```

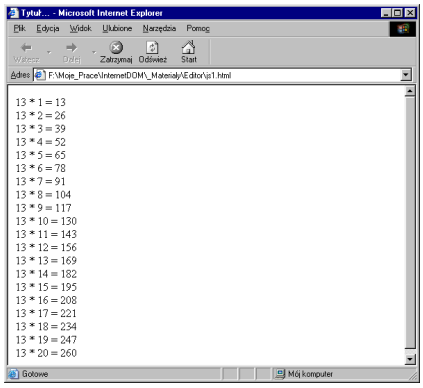
gdzie w miejscu `zmienna` należy podać nazwę zmiennej wraz z jej początkową wartością, jako `warunki_przerwania` należy określić warunek, którego spełnienie będzie oznaczało przerwanie wykonywania pętli i przejście do pierwszego polecenia poza nawiasami klamrowymi zamykającymi pętlę, zaś `zmiana_wartosci_zmiennej` to określenie, jak ma być zmieniana `zmienna` w każdym kolejnym przejściu pętli. Na rysunku 5.39 przedstawiono przykład najprostszej pętli skończonej typu `for`.

continue

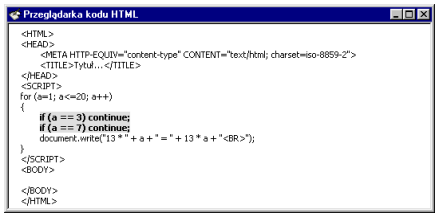
Nie jest to osobny typ pętli, a jedynie polecenie uzupełniające działanie pętli `for`. Jego wystąpienie w dowolnym miejscu bloku wyznaczonego nawiasami klamrowymi (wewnątrz pętli `for`), zmusi interpreter *JavaScript* do zachowania się tak, jakby dany krok przebieg już się skończył — to znaczy do zignorowania wszystkich poleceń następujących po `contiunue` oraz do rozpoczęcia nowego przebiegu pętli, oczywiście wraz ze zwiększeniem wartości `zmienna` w sposób określony przez `zmiana_wartosci_zmiennej` (patrz definicja pętli `for`, powyżej). Przykładem takiego „ominięcia” jednego lub kilku przebiegów pętli jest zmodyfikowana wersja poprzedniego skryptu przedstawiona na rysunku 5.41.



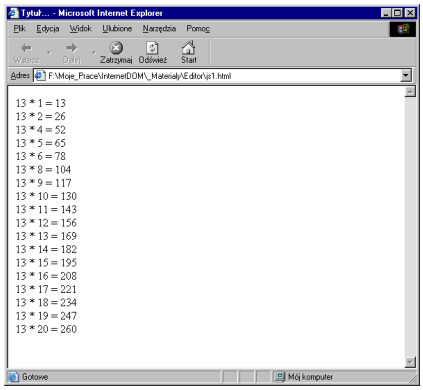
Rysunek 5.39. *Przykład prostej pętli typu for*



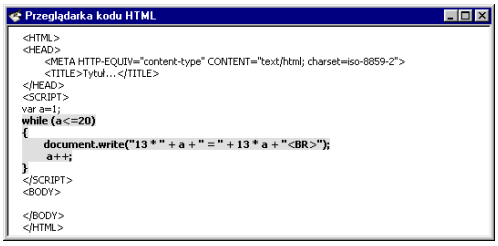
Rysunek 5.40. *Efekt działania powyższego kodu, w przeglądarce*



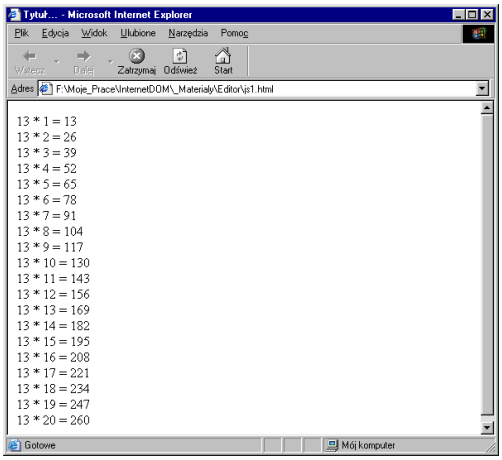
Rysunek 5.41. *Modyfikacja skryptu: „ominięcie” dwóch przebiegów pętli*



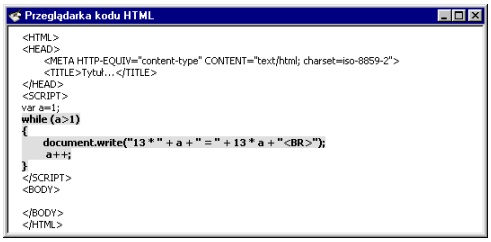
Rysunek 5.42. *Efekt powyższego ominięcia, zaprezentowany w przeglądarce*



Rysunek 5.43. *Prosty przykład wykorzystania pętli while*



Rysunek 5.44. *Efekt działania skryptu — żadnych różnic w stosunku do rysunku 5.40 mimo, że do uzyskania obu efektów wykorzystano dwie różne pętle*



Rysunek 5.45. *Przykład skryptu wykorzystującego pętlę nieskończoną; ze względu na optymalizacje, w przeglądarce nie ujrzysz żadnego efektu działania tego skryptu*

while

Ogólna konstrukcja:

```
while (warunki_przerwania)
{
polecenia;
}
```

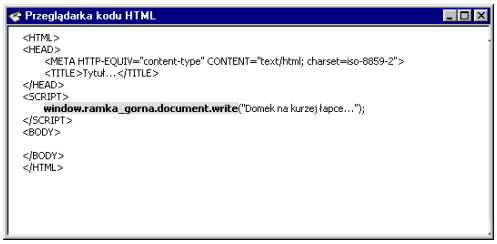
gdzie `warunki_przerwania` mają identyczne znaczenie jak to podane przy omawianiu pętli for. Na rysunku 5.43 przedstawiono skrypt wykonujący dokładnie to samo zadanie, które realizuje skrypt z rysunku 5.39, lecz tym razem z wykorzystaniem pętli `while`.

Pętla `while`, ze względu na swoją konstrukcję pozwala w prosty sposób tworzyć pętle nieskończone (rysunek 5.45), ale ponieważ *JavaScript* ma w założeniu być uzupełnieniem *HTML*, a nie celem samym w sobie, więc stosowanie na stronach *WWW* pętli nieskończonych nie ma chyba większego sensu.

Dziedziczenie w JavaScript

W przypadku języka skryptowego *JavaScript* problem dziedziczenia jest znacznie bardziej rozbudowany niż w przypadku arkuszy stylów *CSS*. Bez zmian, obiekty (wraz z właściwościami i metodami) są ułożone w sposób hierarchiczny. Jednakże w tym przypadku, aby operować na jakimś obiekcie podrzędnym lub jego metodzie czy właściwości, trzeba w poleceniu wskazać wszystkie jego obiekty nadrzędne. Do rozdzielenia kolejnych obiektów oraz ich metod i właściwości służy znak kropki (`.`). Przykład został przedstawiony na rysunku 5.46.

Trzy obiekty *JavaScript* (patrz następny rozdział): `navigator`, `history` oraz `window` nie mają nad sobą obiektów nadrzędnych, wszystkie pozostałe posiadają swojego „rodzica" i to od niego należy rozpoczynać wywołanie określonej metody czy właściwości obiektów podrzędnych.



Rysunek 5.46. *Przykład dziedziczenia obiektów. Aby wypisać tekst w ramce o nazwie „ramka_gorna" (zdefiniowanej gdzieś indziej) należy najpierw odwołać się do obiektu nadrzędnego window, następnie do podrzędnego mu obiektu ramki, dalej do podrzędnego tej ramce obiektu document i dopiero ostatecznie wywołać jej metodę write. Każdy kolejny poziom dziedziczenia został oddzielony kropką.*