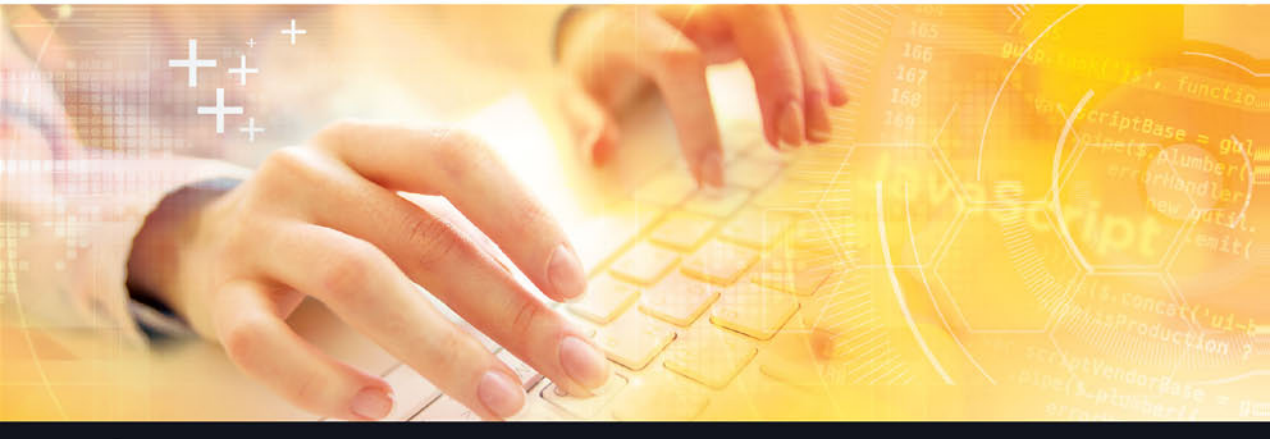
DO NOWEJ PODSTAWY PROGRAMOWEJ

Kwalifikacje E.14 i EE.09
Podstawy programowania w języku JavaScript



Ćwiczenia praktyczne do nauki zawodu
technik informatyk

Piotr Siewniak

Helion

Redaktor prowadzący: Joanna Zaręba

Projekt okładki: Jan Paluch

Fotografia na okładce została wykorzystana za zgodą Shutterstock.

Wydawnictwo HELION
ul. Kościuszki 1c, 44-100 GLIWICE
tel. 32 231 22 19, 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *http://helion.pl* (księgarnia internetowa, katalog książek)

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
http://helion.pl/user/opinie?ke14e9
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

ISBN: 978-83-283-3569-1

Printed in Poland.

Spis treści

Od autora

Jak korzystać z tej książki?

Uczniowie zazwyczaj korzystają z książek, podręczników lub innych materiałów wybiórczo — selektywnie. Często bywa tak, że nie czytają chronologicznie jednego rozdziału po drugim, a jedynie te tematy, które są im w danym momencie potrzebne. Wskazuje na to jednoznacznie moje wieloletnie doświadczenie zawodowe. Na przykład zdarza się, że uczniowie uczą się podstaw programowania obiektowego — opisanego np. w rozdziale 6. określonej książki — ponieważ jest im to potrzebne do zapowiedzianego sprawdzianu w szkole. Jednocześnie, z różnych przyczyn, nie zapoznali się wcześniej z tematyką zawartą w rozdziałach np. od 4. do 5. To po pierwsze.

Po drugie, z informacji przekazywanych mi wielokrotnie przez uczniów wynika, że najchętniej i najskuteczniej uczą się oni, korzystając z gotowych przykładów praktycznych. Z tym że przykłady te powinny być odpowiednio obszernie (wyczerpująco) skomentowane i wyjaśnione.

Biorąc pod uwagę wymienione czynniki, **każdy rozdział niniejszej książki z osobna i książka traktowana jako całość spełniają kilka kryteriów**.

Po pierwsze, każdy rozdział w książce (oprócz rozdziału 12.) został napisany w sposób identyczny organizacyjnie. Każdy z tych rozdziałów składa się z trzech podpunktów: teorii, przykładów praktycznych oraz zadań do wykonania w szkole i (lub) w domu.

Po drugie, każdy przykład praktyczny został bardzo obszernie opisany i wyjaśniony przy wykorzystaniu komentarzy zawartych bezpośrednio w aplikacji. Komentarze te są szczegółowe i nie dotyczą aplikacji traktowanej jako całość, ale odnoszą się do konkretnych instrukcji w niej zawartych. Takie podejście pozwala na szybkie i skuteczne opanowanie przez ucznia prezentowanego materiału. Doświadczenie nauczyło mnie, że większość uczniów preferuje takie pomoce dydaktyczne, które są konkretne, jednoznaczne i dobrze opisane (ale niezbyt obszerne).

Po trzecie, każdy rozdział tworzy integralną całość. Dlatego w komentarzach zawartych w przykładowych aplikacjach (jak również pod nimi) niejednokrotnie pojawiają się powtórzenia. Jest to w pełni świadome i celowe działanie autora. Cele takiego podejścia są dwa. Pierwszy z nich wynika z intencji, aby — mówiąc metaforycznie — uczeń wsiadł do (zatankowanego) samochodu i od razu powoli nim jechał, a nie — bliżej lub dalej — szukał stacji benzynowej. Ujmując to bez metafor, można powiedzieć, że zamiarem autora jest to, aby uczeń otrzymał niezbędny, kompletny opis aplikacji i nie wytracał niepotrzebnie tempa nauki ani nie rozpraszał się poszukiwaniem brakujących mu informacji. Drugi cel wynika z zastosowania zasady polegającej na tym, że powtarzanie materiału stanowi jedną z najskuteczniejszych metod uczenia się.

Po czwarte, książka — traktowana jako całość — obejmuje jedynie podstawy programowania w języku JavaScript. Uczeń nie znajdzie tutaj przykładów (i opisów) żadnych zaawansowanych narzędzi, technik czy też metod programistycznych. Takie podejście mogłoby go zniechęcić do nauki lub wywołać w nim niepokój. Książka ta ma stanowić jedynie zestaw podstawowych „znaków nakazu, zakazu oraz tablic informacyjnych" niezbędnych na „mapie drogowej" ucznia, aby w możliwie jak najkrótszym czasie i jak najskuteczniej opanował ideę programowania w JavaScripcie i zaczął myśleć kategoriami tego języka. Dla uczniów bardziej zainteresowanych tematyką dotyczącą programowania w JavaScripcie przeznaczone są inne książki, np. te, które wymieniono w bibliografii.

A zatem jak korzystać z tej książki? Odpowiedź brzmi następująco: z tej książki można korzystać w sposób dowolny — pod kierunkiem nauczyciela. Lektura tej książki pozwoli na pierwsze „jazdy samochodem na placu manewrowym" JavaScriptu pod okiem instruktora, a następnie na samodzielną, spokojną „jazdę" w środowisku aplikacji internetowych, w tym aplikacji dynamicznych z dynamiką programowaną po stronie przeglądarki.

Przedmowa

Niniejsza książka stanowi uzupełnienie podręcznika do nauki zawodu technik informatyk napisanego przez Jolantę Pokorską i zatytułowanego *Kwalifikacja E.14. Tworzenie aplikacji internetowych*, który został wydany przez wydawnictwo Helion w 2014 r., przy czym uzupełnienie dotyczy jedynie rozdziału 3. wspomnianego podręcznika, noszącego tytuł *Skrypty po stronie klienta — JavaScript*.

Książka jest przeznaczona dla uczniów technikum informatycznego kształcących się w zawodzie technik informatyk. Materiał w niej zawarty jest zgodny z podstawą programową kształcenia w zawodzie technik informatyk. W szczególności dotyczy to kwalifikacji *E.14. Tworzenie aplikacji internetowych i baz danych oraz administrowanie bazami*, jak również kwalifikacji *EE.09. Programowanie, tworzenie i administrowanie stronami internetowymi i bazami danych*.

W niniejszej książce przedstawiono kilkadziesiąt przykładów stron (aplikacji) internetowych, których celem jest zaprezentowanie podstaw programowania w JavaScripcie. Za pomocą wspomnianych przykładów zilustrowano zwłaszcza:

- deklarowanie zmiennych, korzystanie z różnych typów danych, operatorów, wyrażeń;
- definiowanie funkcji zwykłych i funkcji anonimowych oraz ich zastosowanie;
- wykorzystanie instrukcji warunkowych i pętli programowych;
- mechanizm tworzenia obiektów i tablic oraz korzystania z nich;
- wykorzystanie różnych sposobów i narzędzi w celu uzyskania dostępu do elementów HTML z poziomu kodu języka JavaScript;
- obsługę formularzy HTML;
- zagadnienia dotyczące rejestracji i obsługi zdarzeń z poziomu kodu języka JavaScript;
- walidację danych wejściowych użytkownika zawartych w formularzu HTML.

Jednocześnie należy podkreślić, że materiał zamieszczony w książce z założenia nie obejmuje tematyki dotyczącej biblioteki jQuery, techniki AJAX oraz języka (formatu) wymiany danych JSON.

Zakłada się, że uczeń posiada podstawową wiedzę i umiejętności z zakresu projektowania stron (witryn) internetowych w języku znaczników HTML (HTML5) oraz stylizacji tych stron (witryn) za pomocą kaskadowych arkuszy stylów CSS (CSS3).

Pomocne (ale nie niezbędne) jest posiadanie przez uczniów wiedzy i umiejętności związanych z zasadami programowania obiektowego w wybranym języku programowania, np. C++ lub C#.

UWAGA

Zaleca się, aby korzystanie z tej książki rozpocząć bezwzględnie od rozdziału 1., „Wprowadzenie”. Istnieje kilka przyczyn takiego zalecenia, a najważniejszą z nich jest to, że zapoznanie się z materiałem przedstawionym we wspomnianym rozdziale 1. jest potrzebne do skutecznego opanowania materiału zawartego w kolejnych rozdziałach.

1 Wprowadzenie

W niniejszym rozdziale przedstawiono w skrócie najważniejsze elementy i narzędzia języka JavaScript oraz zasady programowania w tym języku.

Zapoznanie się z materiałem zaprezentowanym w tym rozdziale jest bardzo ważne, ponieważ jest potrzebne do skutecznego opanowania materiału zawartego w kolejnych rozdziałach.

Teoria

Skrypty języka JavaScript

W którym miejscu (gdzie) względem dokumentu HTML umieszczać skrypty (kod) języka JavaScript?

Kod JavaScriptu może zostać zdefiniowany względem dokumentu HTML na trzy sposoby, a mianowicie:

1) wewnątrz dokumentu HTML — **„w linii”** (ang. *inline*) definicji elementu (znacznika) HTML, z którym ten kod jest skojarzony;
2) wewnątrz dokumentu HTML, pomiędzy znacznikami `<script>` oraz `</script>` — **skrypty „osadzone”** (ang. *embedded scripts*);
3) na zewnątrz dokumentu HTML, w osobnym pliku dołączanym do tego dokumentu poprzez użycie znaczników `<script src="nazwapliku">` oraz `</script>` wraz z określeniem źródła (ewentualnie ze ścieżką dostępu) — **skrypty zewnętrzne** (ang. *external scripts*).

W ogólności — biorąc pod uwagę dokument HTML — skrypty języka JavaScript można definiować na wszystkie wymienione powyżej sposoby jednocześnie. Tym samym w danym dokumencie HTML można korzystać np. zarówno ze skryptów osadzonych, jak i ze skryptów zewnętrznych.

Najważniejsze zasady używania skryptów

- Deklaracja użycia JavaScriptu o postaci `<script type="text/javascript">` nie jest konieczna, ponieważ JavaScript stanowi domyślny język skryptowy w dokumentach HTML5.
- Skrypty osadzone można umieszczać w dokumencie HTML zarówno w sekcji `head`, jak i w sekcji `body` albo w obu wymienionych sekcjach jednocześnie.
- W ogólności dany dokument HTML może zawierać dowolną liczbę skryptów osadzonych.
- Plik zewnętrzny zawierający skrypt JavaScriptu posiada zazwyczaj rozszerzenie *.js*. W pliku tym nie używa się znaczników `<script>` oraz `</script>`.
- Kod JavaScriptu jest kompilowany (tłumaczony) w dokumencie HTML dokładnie w tym miejscu, gdzie został zadeklarowany (zdefiniowany) za pomocą znacznika (znaczników) `<script>`, przy czym dotyczy to zarówno skryptów osadzonych, jak i skryptów zewnętrznych.
- Kolejność wykonywania skryptów JavaScriptu wynika z kolejności występowania skryptów osadzonych w dokumencie HTML oraz z miejsca, w którym dołączane są skrypty zewnętrzne. Skrypty są wykonywane sekwencyjnie — od góry do dołu dokumentu HTML — przy uwzględnieniu ewentualnych deklaracji dołączenia do tego dokumentu skryptów zewnętrznych. Z tego wynika, że najpierw wykonywane są np. skrypty [zdefiniowane w dokumencie i (lub) do niego dołączane] w sekcji `head`, a dopiero później skrypty zawarte w sekcji `body` [i (lub) do niej dołączane].

Polecenie (wyrażenie) JavaScriptu zawarte w skrypcie może obejmować kilka (wiele) linii. Zalecanym sposobem podziału długiego wyrażenia na części składowe (zawarte w kilku liniach) jest „złamanie" danej linii w miejscu zaraz po wybranym operatorze, np.:

```
document.getElementById("div1").innerHTML = "Zespół Szkół Elektronicznych" +
    " " + "i Informatycznych w Sosnowcu";
```

Ponadto można łamać linie z kodem JavaScriptu zawierającym długie łańcuchy znaków poprzez podział danego łańcucha za pomocą znaku \ (ang. *backslash*), np.:

```
document.getElementById("div1").innerHTML = "Zespół Szkół Elektronicznych \
    i Informatycznych w Sosnowcu";
```

Definicja i wywołanie funkcji

W ogólności w danym języku programowania — np. C++, C#, JavaScript — za rozwiązanie postawionego problemu w całości odpowiada program (aplikacja). Z kolei funkcja (ang. *function*) albo metoda (ang. *method*) odpowiada za rozwiązanie części (tj. podproblemu składowego) danego problemu.

Określenie **funkcja** oznacza zwykle podprogram zdefiniowany niezależnie od danego obiektu (tj. na zewnątrz obiektu). Z kolei **metoda** odnosi się do podprogramu zdefiniowanego jako integralna część składowa obiektu (wewnątrz obiektu).

UWAGA 1.1

Tematyka związana z metodami nie będzie tutaj omawiana. Szczegółowe informacje dotyczące zasad definiowania i wykorzystania metod w JavaScripcie można znaleźć w rozdziale 6.

UWAGA 1.2

W niniejszym rozdziale omawiane są jedynie takie funkcje, które są definiowane za pomocą deklaracji (ang. *function declaration*). Problematyka związana z innymi rodzajami funkcji, np. z funkcjami anonimowymi (ang. *anonymous functions*), została zaprezentowana w rozdziale 3.

W JavaScripcie funkcje deklaruje się za pomocą słowa kluczowego `function`. Nazwa funkcji oraz ewentualne parametry funkcji (ang. *function parameters*) stanowią interfejs (ang. *interface*), za pomocą którego funkcja może się komunikować ze swoim otoczeniem. W JavaScripcie parametry odgrywają wyłącznie rolę wejścia, poprzez które funkcja otrzymuje dane potrzebne do jej działania. Ta cecha odróżnia język JavaScript od innych języków programowania, np. C++ albo C#, w których można stosować parametry zarówno wejściowe, jak i wyjściowe.

Ewentualne parametry funkcji określa się w **nagłówku funkcji** (ang. *function header*) jako pierwszej, integralnej części składowej definicji funkcji w formie **deklaracji**.

W JavaScripcie nagłówek funkcji zawiera wyłącznie nazwy parametrów. W przeciwieństwie do języków takich jak C++ czy też C# w JavaScripcie nie definiuje się typów parametrów funkcji.

Drugą część składową definicji (deklaracji) funkcji stanowi jej **ciało** (ang. *function body*), inaczej zwane treścią funkcji. Ciało (treść) definicji funkcji to **blok kodu** (ang. *code block*), który jest wykonywany tylko i wyłącznie wtedy, gdy funkcja zostanie wywołana. W ogólności blok kodu stanowi zestaw dowolnych instrukcji ujętych w nawiasy klamrowe (ang. *curly braces*) `{}`. Celem stosowania bloku kodu jest zazwyczaj zgrupowanie kilku lub wielu instrukcji składowych w jedną całość.

Funkcja może zwracać na zewnątrz określoną wartość za pośrednictwem swojej nazwy. Tym samym nazwa funkcji może stanowić element składowy interfejsu, za pomocą którego — o czym wspomniano wcześniej — funkcja ta komunikuje się ze swoim otoczeniem. W takim przypadku w treści funkcji należy użyć słowa kluczowego `return` oraz należy podać treść wyrażenia (ang. *expression*), którego wartość ma być zwrócona na zewnątrz za pośrednictwem nazwy funkcji: `return wyrażenie;`. Identyczna sytuacja występuje w innych językach programowania, np. C++ bądź C#.

Wykonanie instrukcji `return` zawartej w treści funkcji kończy działanie tej funkcji.

Wywołanie funkcji (ang. *function call*) powoduje wykonanie instrukcji zawartych w bloku kodu tej funkcji. Wywołanie funkcji wymaga podania listy argumentów tego wywołania (ang. *function arguments*), tj. parametrów bieżących „w chwili" i „w miejscu" wywołania funkcji.

Zmienne globalne i lokalne

W przypadku ogólnym każda zmienna zadeklarowana wewnątrz (w treści definicji) jakiejkolwiek funkcji w dokumencie HTML za pomocą słowa kluczowego `var` jest **zmienną lokalną** (ang. *local variable*). Zmienna lokalna jest widoczna wyłącznie wewnątrz funkcji, w której została zadeklarowana.

W przeciwieństwie do zmiennych lokalnych **zmienne globalne** (ang. *global variables*) deklarowane są w dokumencie HTML na zewnątrz funkcji (poza definicją każdej ze zdefiniowanych funkcji). Mogą być one zadeklarowane zarówno wewnątrz skryptów osadzonych (tj. skryptów zdefiniowanych wewnątrz dokumentu HTML), jak i wewnątrz skryptów zewnętrznych (tj. w plikach zewnętrznych dołączanych do dokumentu HTML). Zmienne globalne są dostępne (widoczne) w całym dokumencie HTML. Są one również dostępne (czyli można z nich korzystać) wewnątrz funkcji, które zostały zdefiniowane w dokumencie.

Deklaracja zmiennej jest często połączona z jej **inicjacją** (ang. *initialization*), tj. nadaniem tej zmiennej wartości początkowej (ang. *initial value*). W ogólności deklaracja zmiennej może być połączona z jej inicjacją lub nie. Jeżeli deklaracja zmiennej jest połączona z jej inicjacją, np. `var ZM = "Visual Studio";`, wówczas typ tej zmiennej jest ustalany w sposób automatyczny, na podstawie typu literału (ang. *literal*) po prawej stronie operatora =. W wyrażeniu powyżej typ zmiennej o nazwie `ZM` zostaje ustalony

(w sposób automatyczny) na typ łańcuchowy `String` na podstawie typu literału `"Visual Studio"`.

Zmienne w JavaScripcie należą do **typów dynamicznych** (ang. *dynamic data types*). Oznacza to, że określona zmienna mogła najpierw należeć np. do typu łańcuchowego `String`, a następnie do typu liczbowego `Number`. Przedstawiony poniżej kod jest poprawny.

```
var ZM = "Visual Studio"; // Typ zmiennej ZM: String.
var ZM = 1000; // Typ zmiennej ZM: Number.
```

UWAGA 1.3

Jeżeli jakakolwiek zmienna została zadeklarowana i jednocześnie zainicjowana — ale bez użycia słowa kluczowego `var` — wówczas niezależnie od tego, czy deklaracja ta występuje wewnątrz, czy na zewnątrz funkcji, taka zmienna jest zmienną globalną.

Model obiektowy DOM dokumentu HTML

W **modelu obiektowym DOM** dokumentu (ang. *document object model*) każdemu znacznikowi HTML, np. `body`, `form`, `input`, `div`, odpowiada **obiekt** (ang. *object*). Dlatego też dany element HTML, traktowany jako obiekt modelu DOM, może posiadać określone **właściwości** (ang. *properties*) oraz **metody** (ang. *methods*).

Właściwość najprościej jest porównać do pewnej cechy danego elementu HTML traktowanego jako obiekt modelu DOM. Może to być np. zawartość HTML elementu `div` — `innerHTML`, wartość wpisana do pola wejściowego `input` — `value`, atrybut `src` elementu `img`, kolor czcionki elementu `p` [jako składnik stylu (ang. *style*) elementu `p`] — `style.color` itp. Używając metafory, można powiedzieć, że właściwości obiektów w języku programowania JavaScript odpowiadają rzeczownikom i (lub) przymiotnikom opisującym pewne obiekty realnego świata w języku polskim.

Z kolei metody obiektów modelu DOM to predefiniowane (ang. *predefined*) funkcje, które operują (działają) na tych obiektach. Jest to np. metoda `focus()`, której zadaniem jest nadanie statusu „fokus" (aktywności) określonemu elementowi HTML. Posługując się ponownie metaforą, można powiedzieć, że metody obiektów w języku JavaScript odpowiadają w języku polskim czasownikom opisującym w realnym świecie określone czynności (działania).

W ogólności JavaScript, jako standardowy język skryptowy dokumentu HTML, umożliwia manipulowanie obiektami modelu DOM za pośrednictwem wspomnianych powyżej właściwości oraz metod. Odpowiedni kod JavaScriptu pozwala m.in. na dynamiczną zmianę zawartości, atrybutów oraz stylu CSS elementów HTML traktowanych jako obiekty modelu DOM. Ponadto model DOM umożliwia obsługę zdarzeń, np. zdarzeń myszy skojarzonych z określonymi elementami HTML w dokumencie.

UWAGA 1.4

Tematyka związana z obsługą zdarzeń zostanie zaprezentowana w dalszej części tego rozdziału.

Model DOM dokumentu HTML jest tworzony w czasie, w którym dokument ten jest ładowany do przeglądarki. Innymi słowy, przeglądarka tworzy model DOM strony WWW (dokumentu HTML) w czasie, w którym jest ona wczytywana do pamięci komputera.

Obiekty wchodzące w skład modelu obiektowego DOM tworzą hierarchiczną strukturę drzewiastą.

UWAGA 1.5

Więcej informacji dotyczących wykorzystania obiektów modelu DOM dokumentu HTML można znaleźć np. w [1], [2] oraz [3]. W dalszej części książki odwołania do określonych źródeł książkowych i (lub) internetowych wymienionych w dodatku „Bibliografia" będą oznaczane za pomocą numeru pozycji w tym spisie, ujętego w nawiasy kwadratowe, np. [1].

Dostęp do elementów HTML z poziomu kodu języka JavaScript

Dostęp do elementów HTML z poziomu kodu języka JavaScript można uzyskać na kilka sposobów. Najprostszy z nich polega na wykorzystaniu predefiniowanej w JavaScripcie metody o nazwie `getElementById()`, przy czym przez pojęcie dostępu rozumie się np. możliwość wykorzystania właściwości oraz metod danego elementu (znacznika) HTML, traktowanego jako obiekt modelu DOM dokumentu.

Zastosowanie metody `getElementById()` w odniesieniu do danego elementu HTML w dokumencie wymaga tego, aby ten element miał nadany unikatowy identyfikator `id`. Identyfikator ten stanowi argument metody `getElementById()`. Na przykład dostęp do właściwości o nazwie `innerHTML` elementu `div` posiadającego identyfikator `id="div1"` można uzyskać za pomocą wyrażenia: `getElementById("div1").innerHTML`.

UWAGA 1.6

Więcej informacji na temat dostępu do elementów HTML z poziomu kodu języka JavaScript można znaleźć w rozdziałach 8. i 9.

Zdarzenia

Z poziomu kodu JavaScriptu można obsługiwać różne zdarzenia (ang. *events*) związane z pracą użytkownika na stronie WWW (obsługą strony WWW), np. zdarzenia myszy (ang. *mouse events*).

Jednym z najczęściej obsługiwanych zdarzeń myszy jest zdarzenie `onClick`, polegające na kliknięciu lewym przyciskiem myszy w obrębie określonego elementu HTML, np. przycisku sterującego. Zadaniem programisty jest wówczas m.in. konstrukcja procedury obsługi tego zdarzenia (ang. *event handler*).

Procedura obsługi zdarzenia określa, jakie działania mają zostać wykonane na stronie WWW po wystąpieniu (zarejestrowaniu) tego zdarzenia. Reakcja na wystąpienie określonego zdarzenia może być różnoraka. Może to być np. wyświetlenie okna dialogowego z informacją dla użytkownika strony WWW, pobranie danych wejściowych z formularza czy też wykonanie zadanych operacji obliczeniowych, a następnie prezentacja uzyskanego wyniku.

Procedury obsługi zdarzeń są zazwyczaj funkcjami (o których była mowa wcześniej). Poza tym procedura obsługi zdarzenia może stanowić zestaw instrukcji JavaScriptu, które nie są zawarte w żadnej funkcji.

W dokumencie HTML zdarzenie można „przechwycić" (czyli można zarejestrować jego wystąpienie) na kilka sposobów. Najprostszym z nich jest rejestracja i obsługa zdarzenia „w linii" (ang. *inline*) definicji znacznika HTML skojarzonego z tym zdarzeniem. W tym przypadku procedura obsługi zdarzenia [wywołanie pewnej funkcji i (lub) zestawu instrukcji JavaScriptu] jest przypisana do nazwy zdarzenia jako atrybutu (ang. *attribute*) znacznika HTML, np. `<button onclick="oblicz(); return false;">`.

UWAGA 1.7

Rejestrowanie zdarzeń (procedur obsługi zdarzeń) „w linii" nie jest zalecane! Przyczyn takiego stanu rzeczy jest kilka, a najważniejszą z nich jest to, że rejestracja (i obsługa) zdarzeń „w linii" powoduje wymieszanie kodu języka HTML z kodem języka JavaScript.

Jednakże w celu możliwie jak największego uproszczenia procesu nauki programowania w JavaScripcie zakłada się, że we wszystkich przykładach prezentowanych w tej książce — do rozdziału 9. włącznie — zdarzenia będą rejestrowane i obsługiwane „w linii".

UWAGA 1.8

Więcej informacji dotyczących obsługi zdarzeń z poziomu kodu języka JavaScript można znaleźć w rozdziale 10.

Konwencje kodowania w JavaScripcie

Stosowanie zalecanych konwencji i zasad kodowania (ang. *coding conventions and rules*), czyli odpowiedniego stylu programowania (ang. *programming style*) skryptów języka JavaScript, ma na celu uzyskanie możliwie jak najbardziej czytelnego (ang. *readable*), spójnego (ang. *consistent*) oraz przejrzystego (ang. *maintainable*) kodu.

Konsekwentne stosowanie ogólnie przyjętych zasad (stylu) kodowania ułatwia pracę przede wszystkim w zespole programistów. Najważniejszym założeniem jest wówczas to, aby konwencja (styl) programowania stosowana przez wszystkich członków tego zespołu była taka sama. Biorąc z kolei pod uwagę określoną aplikację, przyjęta konwencja kodowania (programowania) powinna obowiązywać konsekwentnie w całej aplikacji.

Do najważniejszych zasad kodowania w JavaScripcie należą np. określone reguły nazewnicze (ang. *naming rules*). Niektóre z nich przedstawiono poniżej.

- Nazwy zmiennych globalnych należy pisać drukowanymi literami (ang. *capital letters, uppercase*), np. `var IM = "Piotr";`.
- Nazwy zmiennych lokalnych powinny rozpoczynać się od małą literą. W praktyce w nazwach składających się z kilku członów najczęściej stosuje się tzw. konwencję *camelCase* (ang. *camel case convention*), np. `var poleProstokata = bok1 * bok2;`. W niektórych skryptach można spotkać także nazwy zmiennych, w których poszczególne człony są oddzielone od siebie kreskami podkreślenia (ang. *underscores*), np. `var pole_prostokata = bok1 * bok2;`.
- W nazewnictwie funkcji (oraz metod) należy stosować wyłącznie konwencję *camelCase*, np. `obliczPole()`.
- Nazwy funkcji typu konstruktor (umożliwiające utworzenie obiektu należącego do określonego typu obiektowego) powinny rozpoczynać się dużą literą, np. `var osoba1 = new Osoba();`.
- Nie należy używać kreski podkreślenia jako pierwszego znaku nazw zmiennych i funkcji.
- Wieloczłonowe identyfikatory i klasy CSS wykorzystywane w kodzie języka JavaScript należy nazywać w taki sposób, aby poszczególne człony były oddzielone od siebie kreską podkreślenia, np. `id="mila_morska"`.

UWAGA 1.9

Więcej informacji o zasadach kodowania w JavaScripcie można znaleźć np. na stronie *https://www.w3schools.com*, w artykule *JavaScript Style Guide and Code Conventions*.

UWAGA 1.10

W różnych językach programowania stosowane (i zalecane) są różne konwencje kodowania aplikacji. Na przykład w języku C++ bardzo często wykorzystywany jest styl kodowania zaproponowany przez twórcę tego języka, Bjarnego Stroustrupa, znacznie odbiegający od konwencji kodowania aplikacji w JavaScripcie.

Przykłady

Przykład 1.1

Słowa kluczowe: ***rejestracja zdarzenia „w linii”; zdarzenie myszy `onClick`***

W aplikacji przedstawionej poniżej, na listingu 1.1.1, zademonstrowano rejestrację i obsługę „w linii” zdarzenia myszy `onClick`. Kod JavaScriptu występuje „w linii” definicji znacznika HTML `button`.

Listing 1.1.1

```
<!DOCTYPE html>
<html lang="pl">
<head>
    <title> JavaScript: rejestracja zdarzeń w linii. </title>
    <meta charset="UTF-8">
</head>
<body>
    <!-- Zdarzenie myszy onClick jest rejestrowane i obsługiwane „w linii” definicji znacznika button
        o identyfikatorze id="button". Rejestracja zdarzenia „w linii” polega na przypisaniu procedury obsługi zdarzenia atrybutowi znacznika (tutaj: onclick) odpowiadającemu temu zdarzeniu. Procedurę obsługi zdarzenia onClick stanowi tutaj metoda alert(). -->
    <button id="button" onclick="alert('Komunikat');"> alert </button>
</body>
</html>
```

Zdarzenie myszy `onClick` jest skojarzone ze znacznikiem HTML `button` o identyfikatorze `id="button"`. Obsługa tego zdarzenia rejestrowana „w linii” polega na wywołaniu predefiniowanej w JavaScripcie metody o nazwie `alert()`. Innymi słowy, procedurę obsługi zdarzenia `onClick`, skojarzonego z elementem HTML `button`, stanowi wywołanie predefiniowanej metody `alert()`, zarejestrowanej „w linii” definicji tego elementu.

W ogólności w przypadku rejestracji danego zdarzenia „w linii” nazwa tego zdarzenia występuje jako atrybut elementu HTML skojarzonego z obsługiwanym zdarzeniem.

UWAGA 1.11

W celu uzyskania możliwie jak największej przejrzystości w książce przyjęto tradycyjną pisownię nazw zdarzeń, np. `onClick`, w odróżnieniu od nazw atrybutów znaczników HTML odpowiadających tym zdarzeniom, np. `onclick`.

Przykład 1.2

Słowa kluczowe: zdarzenia myszy ***onMouseOver*** *oraz* ***onMouseOut****; zmiana stylu elementu HTML*

W aplikacji przedstawionej poniżej, na listingu 1.2.1, zrealizowano obsługę zdarzeń myszy `onMouseOver` oraz `onMouseOut`, skojarzonych ze znacznikiem `p`. Obsługa wymienionych zdarzeń jest realizowana „w linii".

Listing 1.2.1

```
<!DOCTYPE html>
<html lang="pl">
<head>
    <title> Rejestracja zdarzeń myszy. </title>
    <meta charset="UTF-8">
</head>
<body>
    <!-- Zdarzenia: onMouseOver oraz onMouseOut są rejestrowane „w linii" definicji znacznika HTML
         o nazwie p.
         Obsługa zdarzeń onMouseOver oraz onMouseOut polega na:
           – zmianie koloru czcionki (style.color) elementu (znacznika) p;
           – zmianie koloru tła (style.backgroundColor) elementu p. -->
    <p
        onmouseover="style.color = 'red'; style.backgroundColor = 'yellow';"
        onmouseout="style.color = 'black'; style.backgroundColor = 'white';">
        Umieść kursor myszy ponad tym napisem …
    </p>
</body>
</html>
```

Zdarzenie `onMouseOver` zachodzi (jest wyzwalane) wówczas, gdy wskaźnik myszy zostanie przesunięty ponad określony element HTML [albo jego dowolne „dziecko" (ang. *child*) — biorąc pod uwagę model DOM dokumentu HTML]. Z kolei zdarzenie `onMouseOut` występuje wtedy, gdy wskaźnik myszy zostanie zdjęty znad elementu HTML (albo jego dziecka).

W przedstawionej aplikacji kod JavaScriptu mający na celu rejestrację i obsługę zdarzeń myszy `onMouseOver` oraz `onMouseOut` występuje wyłącznie „w linii" definicji

znacznika `p`. Obsługa wymienionych zdarzeń polega na zmianie stylu CSS elementu `p`. Wykorzystano do tego obiekt JavaScriptu o nazwie `style` oraz jego właściwości:

- `color` — w celu zmiany koloru elementu `p`;
- `backgroundColor` — w celu zmiany koloru tła elementu `p`.

Przykład 1.3

Słowa kluczowe: ***skrypt osadzony; właściwości `innerHTML` oraz `value`; wyjście aplikacji***

W aplikacji poniżej (listing 1.3.1) zaprezentowano wykorzystanie elementów HTML `div` oraz `input` jako elementów wyjściowych na stronie WWW. Ponadto pokazano zastosowanie metody `write()`.

Listing 1.3.1

```
<!DOCTYPE html>
<html lang="pl">
<head>
    <title> JavaScript: wyjście aplikacji. </title>
    <meta charset="UTF-8">
</head>
<body>
    <!-- Element div, odgrywający rolę elementu wyjściowego. -->
    <div id="div"> </div> <br />
    <!-- Element input, odgrywający rolę elementu wyjściowego. -->
    <input id="input" type="text"/> <br /> <br />

    <!-- Skrypt języka JavaScript osadzony w dokumencie. -->
    <script>
        // Deklaracja zmiennej globalnej o nazwie Z1 połączona z jej inicjacją.
        var Z1 = "Visual Studio";
        /* Zastosowanie metody getElementById() w celu uzyskania dostępu do elementu div, traktowanego
           jako obiekt modelu DOM dokumentu HTML. Wykorzystanie właściwości innerHTML w celu
           wyświetlenia bieżącej wartości zmiennej Z1 (zdefiniowanej powyżej) jako zawartości elementu
           HTML o identyfikatorze id="div". */
        document.getElementById("div").innerHTML = Z1;
        // Wyświetlenie wartości zmiennej Z1 przy wykorzystaniu właściwości value elementu HTML
        // o identyfikatorze id="input".
        document.getElementById("input").value = Z1;
        /* Wyświetlenie wartości zmiennej Z1 za pomocą predefiniowanej w JavaScripcie metody write(),
           należącej do obiektu globalnego o nazwie document. */
        document.write(Z1);
    </script>
</body>
</html>
```

W przedstawionej aplikacji wyświetlono wartość zmiennej globalnej o nazwie `Z1`. Zrealizowano to na trzy sposoby. Pierwszy z nich polega na wykorzystaniu znacznika `div`, a drugi — znacznika `input`. Dostęp do wymienionych znaczników uzyskano za pomocą metody dostępowej o nazwie `getElementById()`. Trzeci sposób prezentacji wartości zmiennej `Z1` polega na zastosowaniu metody `write()`, należącej do obiektu globalnego o nazwie `document`. Obiekt `document` reprezentuje w modelu DOM cały dokument HTML załadowany przez przeglądarkę.

Przykład 1.4

Słowa kluczowe: ***model DOM dokumentu; zmiana zawartości, atrybutu*** *oraz* ***stylu elementu HTML***

W aplikacji przedstawionej poniżej, na listingu 1.4.1, zademonstrowano wykorzystanie modelu obiektowego DOM dokumentu HTML w celu zmiany (modyfikacji) zawartości, atrybutu oraz stylu obiektu modelu DOM tego dokumentu, skojarzonego ze znacznikiem `p`.

Listing 1.4.1

```
<!DOCTYPE html>
<html lang="pl">
<head>
    <title> JavaScript: model obiektowy DOM dokumentu HTML. </title>
    <meta charset="UTF-8">
</head>
<body>
   <p id="paragraf1"> Początkowa zawartość paragrafu. </p>
   <p id="paragraf2" title="podpowiedź">
      Umieść kursor myszy ponad tym napisem. Zwróć uwagę na podpowiedź …
   </p>
   <p id="paragraf3" style="color: black;">
      Zawartość paragrafu. Jeżeli tekst jest koloru czerwonego, oznacza to,
      że jego styl został zmieniony.
   </p>

    <!-- Skrypt osadzony w dokumencie -->
    <script>
      /* Deklaracja zmiennej globalnej o nazwie P1. Zmiennej P1 nie przypisano żadnej wartości początkowej, a zatem jej typ nie został określony. */
         var P1;
         /* Nadanie wartości zmiennej P1. Wykorzystanie metody dostępowej o nazwie getElementById(). Metoda o nazwie getElementById() jest stosowana w celu uzyskania dostępu do obiektu modelu DOM skojarzonego ze znacznikiem HTML o zadanym (jako argument) identyfikatorze id. */
         P1 = document.getElementById("paragraf1");
```

```
        /* Deklaracje zmiennych globalnych o nazwach: P2 oraz P3 połączone z ich inicjacją (nadaniem
           wartości początkowych). Zmienne P2 oraz P3 są obiektami JavaScriptu, skojarzonymi z elementami HTML o identyfikatorach (odpowiednio): id="paragraf2" oraz id="paragraf3". */
        var P2 = document.getElementById("paragraf2");
        P3 = document.getElementById("paragraf3");


        /* Zmiana zawartości HTML (innerHTML) elementu (znacznika) o identyfikatorze:
           id="paragraf1", traktowanego jako obiekt modelu DOM dokumentu.
           Nazwa tego obiektu to P1. */
        P1.innerHTML = "Zmieniona zawartość paragrafu. ";
        //  Zmiana atrybutu title elementu HTML skojarzonego z obiektem modelu DOM dokumentu,
        // odpowiadającym zmiennej P2.
        P2.title = "Zmieniona podpowiedź";
        // Zmiana stylu elementu HTML skojarzonego z obiektem modelu DOM dokumentu,
        // odpowiadającym zmiennej P3.
        P3.style.color = "red";
    </script>
</body>
</html>
```

W przedstawionej aplikacji zmienne o nazwach `P1`, `P2` oraz `P3` są zmiennymi globalnymi. Zmienne `P1` oraz `P2` zadeklarowano z użyciem słowa kluczowego `var`, a zmienną `P3` — bez słowa kluczowego `var`. W przypadku ogólnym deklaracja zmiennej globalnej bez użycia słowa kluczowego `var` wymaga jej jednoczesnej inicjacji (nadania tej zmiennej wartości początkowej).

Każda ze zmiennych `P1`, `P2` oraz `P3` jako obiekt modelu DOM dokumentu jest skojarzona z określonym elementem `p` zawartym w tym dokumencie. Wspomniane elementy `p` posiadają identyfikatory (odpowiednio) `id="paragraf1"`, `id="paragraf2"` oraz `id="paragraf3"`.

Dostęp do znaczników `p` z poziomu kodu języka JavaScript uzyskano za pomocą metody `getElementById()`.

Przykład 1.5

Słowa kluczowe: ***definicja i wywołanie funkcji; kod JavaScriptu; skrypty osadzony oraz zewnętrzny***

W aplikacji zaprezentowanej na listingu 1.5.1 wykorzystano kod języka JavaScript zdefiniowany:

- „w linii” znacznika HTML;
- w skrypcie osadzonym;
- w skrypcie zewnętrznym.

Listing 1.5.1

```
<!DOCTYPE html>
<html lang="pl">
<head>
    <title> JavaScript: skrypty JavaScriptu. </title>
    <meta charset="UTF-8">

    <!-- Dołączenie do dokumentu HTML skryptu zewnętrznego zawartego w pliku zewnętrznym o nazwie
        skrypt.js. -->
    <script type="text/javascript" src="skrypt.js"> </script>
</head>
<body>
    <!-- Użycie kodu JavaScriptu „w linii” definicji elementu (znacznika) HTML o nazwie button, o identy-
        fikatorze id="button1". Funkcja zmianaNapisu1() stanowi procedurę obsługi zdarzenia onClick sko-
        jarzonego z tym elementem. Definicja funkcji zmianaNapisu1() jest zawarta w skrypcie osadzonym,
        w dolnej części sekcji body. -->
    <button type="button" id="button1" onclick="zmianaNapisu1()">
        Naciśnij w celu zmiany napisu poniżej.
    </button>
    <p id="paragraf1"> Napis początkowy. </p>

    <!-- Użycie kodu języka JavaScript „w linii” przycisku sterującego o identyfikatorze id=button2". Funkcja
        zmianaNapisu2() stanowi procedurę obsługi zdarzenia onClick skojarzonego z tym przyciskiem
        sterującym. Definicja funkcji zmianaNapisu2() zawarta jest w pliku zewnętrznym o nazwie
        skrypt.js. -->
    <button type="button" id="button2" onclick="zmianaNapisu2()">
        Naciśnij w celu zmiany napisu poniżej.
    </button>
    <p id="paragraf2"> Napis początkowy. </p>

    <!-- SKRYPT OSADZONY. -->
    <script>
        function zmianaNapisu1() {
            var x = document.getElementById("paragraf1");
            x.innerHTML = "Informacja z funkcji zdefiniowanej\
                w skrypcie osadzonym …";
        }
    </script>
</body>
</html>
```

Kod JavaScriptu „w linii” definicji znaczników HTML o nazwie `button` ma na celu rejestrację i obsługę zdarzeń myszy `onClick` skojarzonych z tymi znacznikami.

Obsługa zdarzenia `onClick` skojarzonego z elementem `button`, dla którego `id="button1"`, polega na wywołaniu funkcji `zmianaNapisu1()`. Funkcja ta została zdefiniowana w skrypcie osadzonym — w dolnej części sekcji `body` dokumentu.

Z kolei obsługa zdarzenia `onClick` skojarzonego z przyciskiem o identyfikatorze `id="button2"` polega na wywołaniu funkcji `zmianaNapisu2()`. Funkcja `zmianaNapisu2()` została zdefiniowana w skrypcie zewnętrznym, zapisanym w pliku o nazwie *skrypt.js*.

Zawartość pliku zewnętrznego *skrypt.js* została zaprezentowana poniżej, na listingu 1.5.2.

Listing 1.5.2

```
// SKRYPT ZEWNĘTRZNY.
function zmianaNapisu2() {
    var x = document.getElementById("paragraf2");
    x.style.color = "red";
    x.innerHTML = "Informacja z funkcji zdefiniowanej w skrypcie zewnętrznym …";
}
```

Podczas analizy przedstawionej aplikacji nasuwa się następujące pytanie: który skrypt zostanie załadowany przez przeglądarkę w pierwszej kolejności — osadzony czy zewnętrzny?

Odpowiedź brzmi: oczywiście najpierw zostanie załadowany skrypt zewnętrzny, zawarty w pliku *skrypt.js*, ponieważ deklaracja jego dołączenia do dokumentu HTML została umieszczona w sekcji `head`, która w dokumencie HTML występuje przed sekcją `body` (w niej został zdefiniowany skrypt osadzony).

Dobrym nawykiem programistycznym jest umieszczanie skryptów osadzonych JavaScriptu na końcu sekcji `body` dokumentu HTML. Dlaczego?

Dlatego, że kompilacja tych skryptów nie wpływa wówczas na szybkość wyświetlania strony (ładowania strony do przeglądarki). Drugim, nie mniej ważnym powodem jest to, że w dolnej części sekcji `body` model DOM dokumentu jest już określony, dlatego można korzystać ze wszystkich obiektów składowych tego modelu.

Przykład 1.6

Słowa kluczowe: ***funkcje; metoda `getElementById()`; model DOM; właściwości `innerHTML`*** *oraz* ***`value`***

Zadanie. Napisać aplikację HTML pozwalającą na przeliczenie zadanej wartości mocy samochodu w kilowatach (kW) na konie mechaniczne (KM). Daną wejściową do aplikacji (moc podaną w kW) wprowadzić z klawiatury za pośrednictwem formularza. Wynik (moc w KM) zaprezentować wewnątrz tego samego formularza. Wykorzystać funkcje.

Rozwiązanie. Kod źródłowy aplikacji stanowiącej rozwiązanie zadania przedstawiono na listingu 1.6.1.

Listing 1.6.1

```
<!DOCTYPE html>
<html lang="pl">
<head>
    <title> JavaScript: funkcje. </title>
    <meta charset="UTF-8">
</head>
<body>
    <form>
        <!-- WEJŚCIE -->
        Podaj liczbę kilowatów: <input type="text" id="input" value="0.736">
        <br /> <br />
        <!-- STEROWANIE I (NIEJAWNE) PRZETWARZANIE -->
        <!-- Jako reakcja na wystąpienie zdarzenia myszy onClick wywoływana jest bezparametrowa
            funkcja wynik(). Instrukcja: return false; zapobiega odświeżeniu strony po naciśnięciu przycisku
            sterującego. -->
        <button onclick="wynik(); return false;">
              przelicz na konie mechaniczne
        </button>
        <!-- WYJŚCIE -->
        <p id="wynik"> </p>
    </form>

    <script>
            /* Definicja funkcji przelicz_kW_na_KM(), której zadaniem jest przeliczenie mocy zadanej
               w kilowatach (kW) na moc w koniach mechanicznych (KM). Parametrem wejściowym
               funkcji jest kW (zadana liczba kilowatów). */
            function przelicz_kW_na_KM(kW) {
             // Funkcja zwraca obliczoną wartość na zewnątrz za pośrednictwem swojej nazwy.
                return kW / 0.736;
            }
            // Definicja bezparametrowej funkcji wynik().
            function wynik() {
            /* Pobranie wartości danej typu String z pola tekstowego input o identyfikatorze id="input"
               i konwersja tej danej (wartości) na typ liczbowy (Number). */
            var x = Number(document.getElementById("input").value);
            /* Wywołanie funkcji przelicz_kW_na_KM() z argumentem (parametrem bieżącym),
               który stanowi zmienna  x, a następnie podstawienie wyniku do zmiennej o nazwie y. */
            var y = przelicz_kW_na_KM(x);
            /* Wyświetlenie wyniku wewnątrz formularza jako zawartości elementu HTML o identyfika-
               torze id="wynik", skojarzonego z obiektem modelu DOM dokumentu. */
            document.getElementById("wynik").innerHTML = y;
            }
    </script>
</body>
</html>
```

W aplikacji powyżej zdefiniowano dwie funkcje: `przelicz_kW_na_KM(kW)` oraz `wynik()`.

Funkcja `przelicz_kW_na_KM(kW)` posiada jeden parametr (wejściowy). Odpowiada on zadanej liczbie kilowatów (kW). Funkcja ta zwraca na zewnątrz — za pośrednictwem swojej nazwy — obliczoną liczbę koni mechanicznych (KM).

Funkcja `wynik()` jest funkcją bezparametrową. Zadania tej funkcji są następujące: pobranie zadanej liczby kilowatów z pola tekstowego `input` zawartego w formularzu; przeliczenie kilowatów na konie mechaniczne; prezentacja wyniku jako zawartości HTML (`innerHTML`) elementu `div`.

Wywołanie funkcji `przelicz_kW_na_KM(x)` następuje wewnątrz (w treści) funkcji `wynik()`. Argumentem tego wywołania jest zmienna o nazwie `x` — zadana liczba kilowatów. Jak wspomniano wcześniej, funkcja `przelicz_kW_na_KM(x)` zwraca na zewnątrz wartość (liczbę koni mechanicznych) za pośrednictwem swojej nazwy.

Wywołanie funkcji `wynik()` następuje jako reakcja na wystąpienie zdarzenia myszy `onClick` skojarzonego z przyciskiem sterującym. Nie jest to jednak jedyne działanie związane z naciśnięciem tego przycisku (czyli wystąpieniem zdarzenia `onClick`). Dodatkowo „w linii" wykonywana jest instrukcja `return false;`, która zapobiega odświeżeniu strony po naciśnięciu przycisku (a dokładniej: przesłaniu formularza na adres URL wskazany za pomocą atrybutu `action` znacznika `form`). Brak atrybutu `action` (jak tutaj) oznacza, że dane z formularza powinny być przesłane do strony, która zawiera ten formularz.

Inny sposób na to, aby strona nie została odświeżona po naciśnięciu przycisku, polega na dodaniu do treści funkcji `wynik()` instrukcji `return false;` na jej końcu. Dodatkowo „w linii", w której zdarzenie `onClick` jest przechwytywane i obsługiwane (tj. „w linii" definicji przycisku sterującego), zamiast instrukcji wywołania funkcji `wynik()` należy użyć instrukcji `return wynik();`. Dzięki temu funkcja `wynik()` przekaże do przeglądarki — za pośrednictwem swojej nazwy — pożądaną wartość `false`.

Odpowiednio zmodyfikowany kod źródłowy aplikacji, uwzględniający uwagi przedstawione powyżej, zaprezentowano na listingu 1.6.2.

Listing 1.6.2

```
<!DOCTYPE html>
<html lang="pl">
<head>
    <title> JavaScript: funkcje. </title>
    <meta charset="UTF-8">
</head>
<body>
    <form>
        Podaj liczbę kilowatów: <input type="text" id="input" value="0.736">
        <br /> <br />
        <!-- Jako reakcja na wystąpienie zdarzenia onClick wywoływana jest funkcja wynik(). Funk-
             cja ta zwraca do przeglądarki wartość logiczną false. Wynika to z tego, że w treści funkcji
```

```
            wynik() – na jej końcu – umieszczono instrukcję: return false;. Jest to potrzebne, aby po
            naciśnięciu przycisku sterującego strona nie została odświeżona. -->
        <button onclick="return wynik();">
             pobierz i przelicz na konie mechaniczne
        </button>
        <p id="wynik"> </p>
    </form>

    <script>
        function kW_na_KM(kW) {
            return kW / 0.736;
        }
        function wynik() {
            var x = Number(document.getElementById("input").value);
            var y = kW_na_KM(x);
            document.getElementById("wynik").innerHTML = y;
            /* Dodanie na końcu funkcji instrukcji: return false; spowoduje, że funkcja ta – za pośredni-
               ctwem swojej nazwy – zwróci na zewnątrz wartość logiczną false. */
            return false;
        }
    </script>
</body>
</html>
```

Przykład 1.7

Słowa kluczowe: ***funkcje; metoda `getElementById()`; zdarzenie `onClick`; zmienne globalne i lokalne***

Zadanie. Napisać aplikację pozwalającą na obliczenie, ile lat pozostało danemu pracownikowi firmy do emerytury. Dane pracownika — imię, nazwisko oraz wiek — wprowadzić z klawiatury za pośrednictwem formularza. Zakłada się, że wiek emerytalny osiągany jest przez pracownika, gdy ukończy on 65 lat. Wynik zaprezentować na ekranie. Wykorzystać zmienne globalne i zmienne lokalne.

Rozwiązanie. Kod źródłowy aplikacji stanowiącej rozwiązanie zadania przedstawiono poniżej, na listingu 1.7.1.

Listing 1.7.1

```
<!DOCTYPE html>
<html lang="pl">
    <head>
        <title> JavaScript: zmienne globalne i lokalne. </title>
        <meta charset="UTF-8">
    </head>
    <body>
```

```
<!-- Dane wejściowe są wprowadzane za pośrednictwem pól tekstowych input zawartych
     w formularzu. -->
<form>
    Imię: <input type="text" id="imie" value="PIOTR" /> <br />
    Nazwisko: <input type="text" id="nazwisko" value="SIEWNIAK" />
    <br />
    Wiek: <input type="text" id="wiek" value="50" />
</form> <br /> <br />
<p> Ile lat pozostało do emerytury?</p>
<!-- Reakcję na wystąpienie zdarzenia myszy onClick stanowi wywołanie funkcji wynik().
     Zdarzenie jest rejestrowane i obsługiwane „w linii” przycisku sterującego
     (znacznika button). -->
<button onclick="wynik()"> oblicz </button> <br /> <br />
<div id="wynik"> </div>


<script>
    /* Deklaracja zmiennych globalnych o nazwach: IM, NAZW, W oraz WYN, które są dostępne
       w całym dokumencie. Tym samym zmienne te są dostępne również wewnątrz (w treści)
       funkcji zdefiniowanych w tym dokumencie. Zmienne globalne: IM, NAZW, W oraz WYN
       odpowiadają obiektom modelu DOM dokumentu, skojarzonym ze znacznikami HTML
       o określonych identyfikatorach. Wykorzystanie metody getElementById(). */
    var IM = document.getElementById("imie");
    var NAZW = document.getElementById("nazwisko");
    var W = document.getElementById("wiek");
    var WYN = document.getElementById("wynik");


    /* Zadaniem funkcji oblicz() jest obliczenie, ile lat pozostało pracownikowi do emerytury,
       przy założeniu, że wiek emerytalny osiągany jest przez pracownika, gdy ukończy
       on 65 lat. */
    function oblicz() {
        // Deklaracja zmiennej lokalnej o nazwie wn, która jest dostępna wyłącznie
        // wewnątrz funkcji oblicz().
        var wn = Number(W.value);
        // Funkcja oblicz() zwraca obliczoną wartość na zewnątrz za pośrednictwem
        // swojej nazwy.
        return 65-wn;
    }
    /* Zadaniem funkcji wynik() jest wyświetlenie informacji, ile lat pozostało pracownikowi
       do emerytury. W treści funkcji zadeklarowano i zainicjowano dwie zmienne lokalne:
       komunikat oraz pom. */
    function wynik() {
        // Deklaracja zmiennych lokalnych. Zmienne te są dostępne wyłącznie wewnątrz
        // funkcji wynik().
        var komunikat = "do emerytury pozostało lat: ";
        var pom = IM.value + " " + NAZW.value;
        /* W celu prezentacji wyniku wykorzystuje się właściwość innerHTML zmiennej global-
           nej WYN, skojarzonej ze znacznikiem div o identyfikatorze id="wynik". */
```

```
                WYN.innerHTML = pom + " - " + komunikat + oblicz();
            }
        </script>
    </body>
</html>
```

W aplikacji zilustrowano wykorzystanie zmiennych globalnych (`IM`, `NAZW`, `W`, `WYN`) oraz zmiennych lokalnych (`wn` oraz `komunikat` i `pom`). Dla każdej ze zmiennych deklaracja została połączona z jej inicjacją, tj. nadaniem zmiennej wartości początkowej. Tym samym typ tych zmiennych został ustalony podczas ich deklaracji w sposób automatyczny.

Ponadto w przedstawionym przykładzie zaprezentowano zastosowanie funkcji. W treści funkcji można korzystać ze zdefiniowanych tam zmiennych lokalnych oraz ze zmiennych globalnych.

Przykład 1.8

Słowa kluczowe: funkcje; metoda ***`getElementById()`****; model DOM; zdarzenie* ***`onClick`****; zmienne lokalne*

Zadanie. Napisać aplikację pozwalającą na obliczenie liczby płytek ceramicznych potrzebnych do wykonania remontu łazienki. Zakłada się, że wszystkie ściany boczne w łazience zostaną pokryte nowymi płytkami. Łączna powierzchnia ścian bocznych w łazience jest podawana z klawiatury jako dana wejściowa. Pozostałe dane wejściowe to wymiary dostępnego modelu płytek: szerokość oraz wysokość. Dane te są również podawane z klawiatury. Wynik należy zaprezentować na tej samej stronie WWW.

Rozwiązanie. Kod źródłowy aplikacji stanowiącej rozwiązanie zadania przedstawiono poniżej, na listingu 1.8.1.

Listing 1.8.1

```
<!DOCTYPE html>
<html lang="pl">
<head>
    <title> JavaScript: funkcje. </title>
    <meta charset="UTF-8">
</head>
<body>
    <form name="formularz">
        Podaj powierzchnię ścian w łazience (łącznie) w metrach kwadratowych:
        <br />
        <input id="cala_pow" name="cala_pow" type="text" /> <br />
        Podaj wymiary płytki w centymetrach: <br />
        Szerokość: <input id="szer_plytki" name="szer_plytki" type="text"/>
        <br />
        Wysokość: <input id="wys_plytki" name="wys_plytki" type="text" />
        <br />
```

```
    </form>
      <!-- Rejestracja i obsługa „w linii” zdarzenia myszy onClick skojarzonego z przyciskiem sterującym
           (button). Reakcję na wystąpienie zdarzenia onClick stanowi wywołanie bezparametrowej funkcji
           wynik(). -->
    <button onclick="wynik();"> Oblicz liczbę płytek </button> <br /> <br />


    <div id="wynik"> </div>


    <script>
        /* Zadaniem funkcji powierzchniaPlytki() jest obliczenie powierzchni płytki dla zadanych parame-
           trów: szerokości (szer) i wysokości (wys) płytki. */
        function powierzchniaPlytki(szer, wys) {
            // Deklaracja zmiennej lokalnej o nazwie: pow_w_ccm połączona z jej inicjacją,
            // czyli nadaniem wartości początkowej.
            var pow_w_ccm = szer * wys;
            /* Funkcja zwraca na zewnątrz wartość wyrażenia po prawej stronie instrukcji return.
               Wartość ta jest zwracana za pośrednictwem nazwy funkcji. */
            return (0.01 * 0.01) * pow_w_ccm;
        }


        /* Zadaniem funkcji obliczIlePlytek() jest obliczenie liczby płytek na podstawie zależności: liczba
           płytek = łączna powierzchnia ścian w łazience / powierzchnia jednej płytki. */
        function obliczIlePlytek() {
            /* Deklaracje i inicjacje trzech zmiennych lokalnych: szer_plytki, wys_plytki oraz cala_pow.
               Dostęp do poszczególnych pól tekstowych input formularza, traktowanych jako obiekty
               modelu DOM dokumentu, uzyskuje się przy wykorzystaniu zapisu obiektowego oraz właś-
               ciwości value tych obiektów. Wywołanie metody Number() jest potrzebne w celu dokonania
               konwersji typu danych z typu String na typ Number. */
            var szer_plytki = Number(document.formularz.szer_plytki.value);
            var wys_plytki = Number(document.formularz.wys_plytki.value);
            var cala_pow = Number(document.formularz.cala_pow.value);
            // Funkcja zwraca na zewnątrz wartość (liczbę potrzebnych płytek) za pośrednictwem
            // swojej nazwy.
            return cala_pow / powierzchniaPlytki(szer_plytki, wys_plytki);
        }
        /* Zadaniem funkcji wynik() jest wyświetlenie obliczonego wyniku (liczby potrzebnych płytek)
           jako zawartości (HTML) elementu div o identyfikatorze id="wynik". Dostęp do tej zawartości
           uzyskuje się za pomocą metody getElementById() oraz właściwości o nazwie innerHTML wspo-
           mnianego powyżej elementu div, traktowanego jako obiekt modelu DOM dokumentu.  */
        function wynik() {
            var wyn = document.getElementById("wynik");
            wyn.innerHTML = "Liczba potrzebnych płytek: " + obliczIlePlytek();
        }
    </script>
</body>
</html>
```

W przedstawionej aplikacji zilustrowano zastosowanie większości elementów języka JavaScript, o których była mowa w punkcie „Teoria” na początku tego rozdziału. W szczególności w aplikacji wykorzystano:

- kod JavaScriptu „w linii” definicji znacznika HTML oraz skrypt osadzony;
- rejestrację i obsługę zdarzenia myszy `onClick`;
- elementy (obiekty) modelu DOM dokumentu HTML;
- metodę `getElementById()`, pozwalającą na uzyskanie dostępu do elementów HTML z poziomu kodu języka JavaScript;
- funkcje zdefiniowane przez programistę;
- zmienne lokalne.

UWAGA 1.12

Więcej informacji wstępnych wprowadzających do świata programowania w języku JavaScript można znaleźć we wszystkich pozycjach wymienionych w dodatku „Bibliografia”.

Szczególną uwagę należy zwrócić na tematykę praktycznego wykorzystania funkcji i metod predefiniowanych w JavaScripcie. Najważniejsze różnice pomiędzy nimi przedstawiono w uwagach 6.9 i 6.10 w rozdziale 6.

ZADANIA

Zadanie 1.1

Napisać aplikację pozwalającą na przeliczenie odległości zadanej w kilometrach (km) na odległość podaną w milach morskich INM (ang. *International Nautical Mile*).

Daną wejściową do aplikacji (odległość w km) wprowadzić z klawiatury za pośrednictwem formularza. Wynik (odległość w INM) zaprezentować na ekranie monitora na tej samej stronie WWW.

Wykonać aplikację w dwóch wariantach, w zależności od wybranego elementu wyjściowego. W szczególności zaprezentować wynik:

1. jako zawartość (`innerHTML`) elementu `div` (*wariant I*);
2. jako wartość (`value`) w elemencie `input` (*wariant II*).

Zadanie 1.2

Napisać aplikację pozwalającą na wprowadzenie z klawiatury (za pośrednictwem formularza) następujących danych samochodu osobowego: marki, typu, roku produkcji. Wprowadzone dane wyświetlić kontrolnie na tej samej stronie WWW w wybranym elemencie wyjściowym, np. jako zawartość elementu `div`.

ZADANIA

Wykonać aplikację w dwóch wariantach, w zależności od położenia elementu wyjściowego. Wybrany element wyjściowy umieścić:

1. wewnątrz formularza (*wariant I*);
2. na zewnątrz formularza (*wariant II*).

Zadanie 1.3

Napisać aplikację pozwalającą na obliczenie pola i obwodu kwadratu.

Daną wejściową — długość boku kwadratu — wprowadzić z klawiatury za pośrednictwem formularza. Wyniki zaprezentować na tej samej stronie WWW.

W celu obliczenia obwodu i pola kwadratu zdefiniować dwie osobne funkcje. Każda z nich powinna posiadać jeden parametr wejściowy: zadaną długość boku kwadratu. Ponadto każda z tych funkcji powinna zwracać wynik (pole i obwód) za pośrednictwem swojej nazwy.

Wykonać aplikację w dwóch wariantach, w zależności od tego, gdzie zdefiniowano wspomniane powyżej funkcje. Zdefiniować funkcje mające na celu obliczenie pola oraz obwodu kwadratu:

1. w skrypcie osadzonym (*wariant I*);
2. w skrypcie (pliku) zewnętrznym (*wariant II*).

Zadanie 1.4

Napisać aplikację pozwalającą na zmianę stylu „ramki” (`style.border`) elementu `div` jako reakcję na zdarzenie myszy `onMouseOver` skojarzone z tym elementem.

Z kolei reakcją na zdarzenie myszy `onMouseOut` powinien być powrót do stylu początkowego brzegu. Wymienione zdarzenia zarejestrować „w linii” definicji znacznika `div`.

Skorowidz

L

M

N

O

P

R

S

T

W

Z

Kwalifikacje E.14 i EE.09

Podstawy programowania w języku JavaScript

Ćwiczenia praktyczne do nauki zawodu **technik informatyk**

Technik informatyk nie jest zwykłym użytkownikiem komputerów. Jeśli uczeń wybiera szkołę o takim profilu, z czasem staje się prawdziwym komputerowym specem.

Materiał zawarty w książce ***Kwalifikacje E.14 i EE.09. Podstawy programowania w języku JavaScript. Ćwiczenia praktyczne do nauki zawodu technik informatyk*** jest zgodny z podstawą programową kształcenia w zawodzie technik informatyk — zarówno starą (kwalifikacja E.14), jak i nową (kwalifikacja EE.09).

To kompletna pomoc naukowa — konkretna, jednoznaczna i stworzona bezpośrednio z myślą o potrzebach nastolatków. Ze względu na to, że uczniowie preferują selektywny wybór materiału, każdy z dwunastu rozdziałów może zostać potraktowany niezależnie. Dla ułatwienia wszystkie rozdziały składają się z trzech podpunktów: teorii, przykładów praktycznych oraz zadań do wykonania w szkole i (lub) w domu. Przykłady obszernie opisano i wyjaśniono. Dzięki takiemu układowi deklarowanie zmiennych, korzystanie z różnych typów danych, operatorów czy wyrażeń, a także użycie instrukcji warunkowych i pętli programowych nie powinno sprawiać uczniom żadnych problemów. Ponadto zilustrowano tu sposoby i narzędzia pozwalające na dostęp do elementów HTML z poziomu kodu języka JavaScript oraz na kompleksową obsługę formularzy HTML. Przyszli technicy informatycy poznają też mechanizm tworzenia obiektów i tablic oraz metody ich praktycznego wykorzystywania. Nieobca będzie im również tematyka rejestracji i obsługi zdarzeń z poziomu kodu języka JavaScript. To samo dotyczy walidacji danych wejściowych użytkownika strony WWW.

Technik Informatyk to doskonały, charakteryzujący się wysoką jakością i kompletny zestaw edukacyjny, przygotowany przez dysponującego ogromnym doświadczeniem lidera na rynku książek informatycznych — wydawnictwo Helion. W skład zestawu ***Technik Informatyk*** wchodzą także:

- *Kwalifikacja E.12. Montaż i eksploatacja komputerów osobistych oraz urządzeń peryferyjnych. Podręcznik do nauki zawodu technik informatyk*
- *Kwalifikacja E.13. Projektowanie lokalnych sieci komputerowych i administrowanie sieciami*
- *Kwalifikacja E.14. Część 1. Tworzenie stron internetowych. Podręcznik do nauki zawodu technik informatyk*
- *Kwalifikacja E.14. Część 2. Tworzenie baz danych i administrowanie bazami. Podręcznik do nauki zawodu technik informatyk*
- *Kwalifikacja E.14. Część 3. Tworzenie aplikacji internetowych. Podręcznik do nauki zawodu technik informatyk*

Podręczniki oraz inne pomoce naukowe należące do tej serii zostały opracowane z myślą o wykształceniu kompetentnych techników, którzy bez trudu poradzą sobie z wyzwaniami współczesnej informatyki. Według nowych przepisów, aby otrzymać dyplom w zawodzie technik informatyk, należy zdać szereg egzaminów potwierdzających kolejne kwalifikacje w zawodzie. Ta książka powstała, by ułatwić to zadanie zarówno uczącym się, jak i pedagogom. Zawarta w niej wiedza pomoże zdać egzamin i zyskać wiedzę praktyczną, przydatną w przyszłej pracy.

Helion

księgarnia internetowa
http://helion.pl
zamówienia telefoniczne
0 801 339900
0 601 339900
Informatyka w najlepszym wydaniu

Helion SA
ul. Kościuszki 1c, 44-100 Gliwice
tel.: 32 230 98 63
e-mail: helion@helion.pl
http://helion.pl

Sprawdź najnowsze promocje:
http://helion.pl/promocje
Książki najchętniej czytane:
http://helion.pl/bestsellery
Zamów informacje o nowościach:
http://helion.pl/nowosci

ISBN 978-83-283-3569-1
9 788328 335691
cena: 37,95 zł