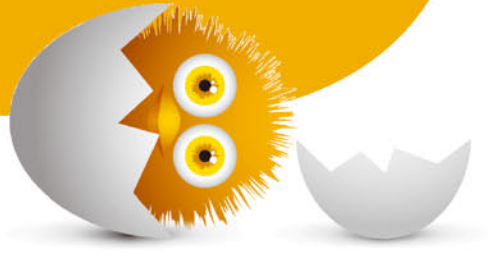
Helion

W KOLORZE

JavaScript

PRZEWODNIK DLA ABSOLUTNIE POCZĄTKUJĄCYCH



que

Kirupa Chinnathambi

Tytuł oryginału: JavaScript Absolute Beginner's Guide

Tłumaczenie: Lech Lachowski

ISBN: 978-83-283-3180-8

Authorized translation from the English language edition, entitled: JAVASCRIPT ABSOLUTE BEGINNER'S GUIDE, First Edition; ISBN 0789758067; by Kirupa Chinnathambi; published by Pearson Education, Inc, publishing as QUE Publishing.

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)

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:
ftp://ftp.helion.pl/przyklady/javpap.zip

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

Printed in Poland.

Spis treści

Część IV. Obsługa zdarzeń

W TYM ROZDZIALE

- nauczysz się, jak wielokrotnie uruchamiać jakiś kod;
- popracujesz z pętlami `for`, `while` oraz `do...while`.

5

POZNAJ PĘTLE: FOR, WHILE ORAZ DO...WHILE!

Gdy coś kodujemy, czasem chcemy powtórzyć pewną akcję lub uruchomić jakiś kod kilka razy. Załóżmy, że mamy na przykład funkcję o nazwie `saySomething` (powiedz coś), którą chcemy wywołać 10 razy.

Jednym ze sposobów, żeby to zrobić, jest po prostu wywołanie tej funkcji 10 razy przy użyciu kopiowania i wklejania:

```
saySomething();
saySomething();
saySomething();
saySomething();
saySomething();
saySomething();
saySomething();
saySomething();
saySomething();
saySomething();
```

Ten sposób działa i realizuje to, co zaplanowaliśmy, ale nie należy robić czegoś takiego. Powielanie kodu nigdy nie jest dobrym pomysłem.

Nawet jeśli zdecydujesz się kilka razy ręcznie powielić jakiś kod, takie podejście w praktyce się nie sprawdza. Często nie będziesz wiedzieć, ile razy trzeba uruchomić kod. Liczba uruchomień kodu będzie zależeć od pewnego czynnika zewnętrznego, takiego jak liczba elementów w kolekcji danych, wynik otrzymany z wywołania usługi internetowej, liczba liter w słowie, oraz od różnych innych rzeczy, które ciągle się zmieniają. Nie zawsze będzie to stała liczba, taka jak 10. Jest więcej niż prawdopodobne, że liczba powtórzeń jakiegoś kodu może być bardzo, BARDZO duża. Nie chcemy kopiować i wklejać czegoś kilkaset lub kilka tysięcy razy, aby powtórzyć jakiś kod. To byłoby straszne.

Potrzebujemy ogólnego rozwiązania dla powtarzania kodu, umożliwiającego kontrolę nad tym, jak wiele razy kod jest powtarzany. W języku JavaScript takie rozwiązanie jest dostarczane w formie zwanej **pętlą**. Możemy tworzyć trzy rodzaje pętli:

- pętle `for`;
- pętle `while`;
- pętle `do...while`.

Każda z tych trzech odmian pętli umożliwia określenie kodu, który chcemy powtórzyć (czyli zapętlić) oraz sposobu zatrzymywania powtarzania, gdy spełniony zostanie pewien warunek. W kolejnych podrozdziałach dowiesz się, jak z nich korzystać.

Do dzieła!

Pętla for

Jednym z najbardziej powszechnych sposobów tworzenia pętli jest użycie instrukcji `for` w celu utworzenia tzw. pętli `for`. Pętla `for` pozwala na wielokrotne uruchamianie jakiegoś kodu, dopóki określone przez nas wyrażenie nie zwróci `false`. Prawdopodobnie na razie nie ma to dla Ciebie zbyt wiele sensu, więc spróbujmy wyjaśnić tę definicję, analizując przykład.

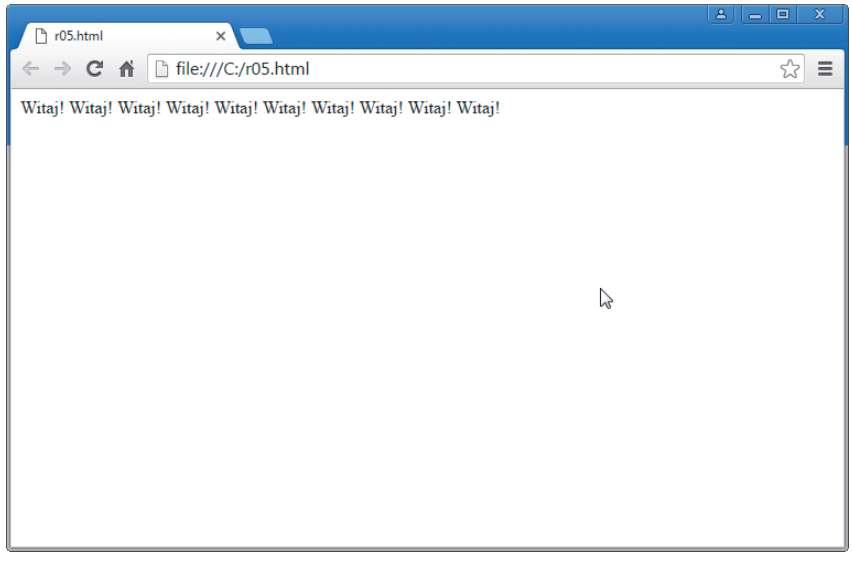
Gdybyśmy mieli przetłumaczyć nasz wcześniejszy przykład funkcji `saySomething`, używając instrukcji `for`, wyglądałoby to następująco:

```
var count = 10;

function saySomething() {
    document.writeln("Witaj!");
}

for (var i = 0; i < count; i++) {
    saySomething();
}
```

Jeśli umieścimy ten kod pomiędzy znacznikami `script` i uruchomimy go w przeglądarce, zobaczymy coś podobnego do tego, co zostało pokazane na rysunku 5.1.



RYSUNEK 5.1.

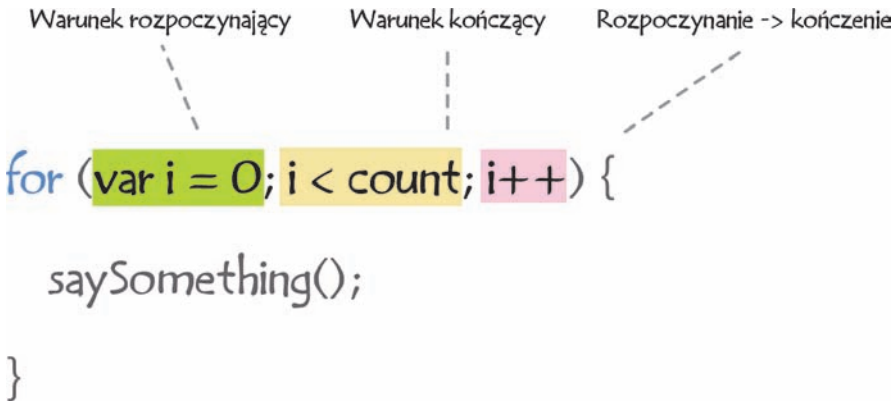
Słowo „Witaj!" zostanie powtórzone dziesięć razy na stronie

UWAGA Słowa są wyświetlane w przeglądarce, ponieważ użyliśmy metody `document.writeln`. Jest to kolejna metoda (poza `alert`), którą będziemy wykorzystywać, aby szybko coś przetestować. Metoda `document.writeln` przyjmuje jako argumenty cokolwiek, co chcemy wypisać, a gdy zostaje uruchomiona, to wszystko, co znajduje się na stronie, zastępuje zawartością wybraną przez nas do wyświetlenia.

Jest to możliwe dzięki pętli `for`, więc wyrazimy swoje podziękowania, poznając szczegółowo sposób jej działania. Po pierwsze, przyjrzyjmy się naszej gwieździe:

```
for (var i = 0; i < count; i++) {
    saySomething();
}
```

Jest to pętla `for`. Prawdopodobnie bardzo różni się od pozostałych instrukcji, jakie widziałeś do tej pory, a to dlatego, że tak naprawdę jest zupełnie inna. Aby zrozumieć różnice, sprowadźmy pętlę `for` do postaci ogólnej przedstawionej na rysunku 5.2.



RYSUNEK 5.2.

Nasza pętla for i trzy regiony, na których się skupimy

Każdy z tych trzech zaznaczonych na rysunku innym kolorem regionów odgrywa bardzo ważną rolę w działaniu pętli `for`. Aby móc prawidłowo korzystać z pętli `for`, musisz wiedzieć, jakie funkcje pełni każdy region. To prowadzi nas do...

Warunek rozpoczynający

W pierwszym regionie definiujemy **warunek rozpoczynający**. Typowy warunek rozpoczynający zwykle obejmuje deklarowanie i inicjowanie zmiennej. W naszym przykładzie utworzyłem nową zmienną o nazwie `i` oraz zainicjowałem ją do liczby `0`:

```
for (var i = 0; i < count; i++) {
    saySomething();
}
```

Gdybyś się nad tym zastanawiał, to informuję Cię, że nazwą zmiennej w takich przypadkach jest tradycyjnie pojedyncza litera, a najczęściej jest to litera `i`. Wartością, do której inicjowana jest ta zmienna, jest również tradycyjnie `0`. Istnieje jeden naprawdę dobry powód, aby użyć liczby `0`, ale wytłumaczę to nieco później.

Warunek kończący (czyli czy już skończyliśmy?)

Gdy zdefiniujemy nasz punkt początkowy, w kolejnym regionie możemy określić **warunek kończący**. Zasadniczo jest to fantazyjny sposób na powiedzenie tego, jak długo należy wykonywać pętlę. Jest to obsługiwane przez wyrażenie warunkowe (takie jakie widziałeś w poprzednim rozdziale), które zwraca wartość `true` lub `false`. W naszym przykładzie warunek jest taki, że zmienna `i` ma być mniejsza niż wartość `count`, który wynosi od `10`:

```
for (var i = 0; i < count; i++) {
    saySomething();
}
```

Jeśli zmienna `i` jest mniejsza niż 10, wyrażenie ewaluuje do wartości `true` i pętla jest wykonywana dalej. Jeśli zmienna `i` jest równa lub większa niż 10, warunek ma wartość `false`, a pętla zostaje przerwana. Być może zastanawiasz się teraz, co powoduje, że wartość zmiennej `i` faktycznie zmienia się z jej wartości początkowej `0`. To omówimy w kolejnym punkcie.

Docieramy do końca

Do tej pory poznaliśmy punkt początkowy oraz wyrażenie, które musi ewaluować do `false`, jeśli chcemy przerwać pętlę. Brakuje ostatniego regionu, w którym opisujemy, jak przejść *od punktu początkowego do punktu kończącego*:

```
for (var i = 0; i < count; i++) {
    saySomething();
}
```

W naszym przykładzie przy każdej iteracji pętli zwiększamy wartość zmiennej `i` o *1*. Na wypadek, gdybyś nie był zaznajomiony z tą składnią, wyjaśniam, że znaki ++ po `i` oznaczają, iż zwiększamy o *1* bieżącą wartość zmiennej `i` bez względy na to, jaka ona jest. Przy każdym przejściu pętli wykonywane będzie to, co zostało w niej określone. W tym przypadku zwiększana będzie nasza wartość `i`.

Zbierając wszystko razem

Skoro przyjrzeliśmy się już każdej części instrukcji `for` w najdrobniejszych szczegółach, przeanalizujmy całość jeszcze raz, aby upewnić się, że wszystko zrozumieliśmy. Nasz kompletny przykład wygląda następująco:

```
var count = 10;

for (var i = 0; i < count; i++) {
    saySomething();
}

function saySomething() {
    document.writeln("Witaj!");
}
```

Gdy kod po raz pierwszy dociera do pętli `for`, zmienna `i` jest tworzona i inicjowana do wartości `0`. Następnie kod sprawdza, czy wartość `i` jest mniejsza od wartości wskazywanej przez zmienną `count`..., która wynosi 10. Na tym etapie wszystko się zgadza i wykonywany jest kod, który znajduje się wewnątrz pętli. W naszym przypadku jest to funkcja `saySomething`. Po wykonaniu poleceń wewnątrz pętli zaczyna działać ostatnia część instrukcji `for`. Zmienna `i` jest zwiększana o *1*.

Teraz pętla zaczyna się od nowa, z tym wyjątkiem, że zmienna `i` nie jest ponownie inicjowana. Jej wartość jest ustawiana na wartość zwiększoną o *1*, a to oznacza, że wciąż przechodzi test `i < count`. Funkcja `saySomething` jest wywoływana ponownie, a wartość `i` jest ponownie zwiększana. Teraz wartość zmiennej `i` wynosi 2.

Cały ten proces powtarza się, aż wartość `i` będzie równa 10. Na tym etapie test `i < count` nie powiedzie się, a pętla zostanie przerwana po pomyślnym wykonaniu funkcji `saySomething` *10* razy.

Kilka przykładów pętli for

W poprzednim podrozdziale zrobiliśmy sekcję prostej pętli `for` i opisaliśmy wszystkie jej wewnętrzne mechanizmy działania. W przypadku pętli `for`, a przede wszystkim w całym języku JavaScript, chodzi o to, że prosty przykład nie zawsze obejmuje wszystko, czego możesz potrzebować. Najlepszym rozwiązaniem jest przeanalizowanie kilku kolejnych przykładów pętli `for` i to właśnie będziemy robić w kilku kolejnych punktach tego podrozdziału.

Wcześniejsze przerywanie pętli

Czasami możemy chcieć zakończyć pętlę, zanim dotrze do końca. Do zakończenia pętli używamy słowa kluczowego `break`. Poniżej znajduje się przykład:

```
for (var i = 0; i < 100; i++) {
    document.writeln(i);

    if (i == 45) {
        break;
    }
}
```

Gdy wartość `i` będzie równa 45, słowo kluczowe `break` powstrzyma dalsze wykonywanie pętli. Chociaż ten przykład jest nieco naciągany, to gdy w prawdziwym kodzie napotkasz przypadek wymagający zakończenia pętli, będziesz wiedział, co robić.

Pomijanie iteracji

Niekiedy może być konieczne, aby pętla pominęła swoją bieżącą iterację i przeszła do następnej. Jest to sprytnie obsługiwane przez słowo kluczowe `continue`:

```
var floors = 28;

for (var i = 1; i <= floors; i++) {
    if (i == 13) {
        // tu nie ma piętra
        continue;
    }

    document.writeln("Piętro: " + i + "<br>");
}
```

W przeciwieństwie do słowa kluczowego `break`, które powoduje po prostu zatrzymanie wykonywania pętli, słowo kluczowe `continue` wskazuje, aby zatrzymać bieżącą iterację i przejść do następnej iteracji. Przekonasz się, że będziesz często używać słowa kluczowego `continue` podczas obsługi błędów, gdzie będziesz chciał, aby pętla po prostu przeszła do kolejnego elementu.

Odliczanie wstecz

Nie ma żadnego powodu, który zmuszałby nas do zaczynania odliczania od 0 i następnie zwiększania wartości:

```
for (var i = 25; i > 0; i--) {
    document.writeln("Witaj");
}
```

Można równie dobrze zacząć od wysokiej wartości, a następnie zmniejszać ją aż do momentu, kiedy warunek pętli zwróci `false`.

Mogłeś słyszeć opinie, że takie postępowanie zwiększa wydajność pętli. Za wcześnie jeszcze na ocenę, czy dekrementacja jest rzeczywiście szybsza niż inkrementacja, ale zachęcam do eksperymentowania i sprawdzania, czy można zaobserwować jakiekolwiek korzyści związane z wydajnością.

Nie trzeba używać liczb

Podczas wypełniania pętli `for` nie trzeba używać wyłącznie liczb ani wykonywać tradycyjnej operacji inkrementacji (dekrementacji):

```
for (var i = "a" ; i !="aaaaaaaa" ; i += "a" ) {
    document.writeln("hmm...");
}
```

Można używać, czego tylko się chce, ale pod warunkiem, że pętla ostatecznie osiągnie punkt, w którym będzie mogła się zakończyć. Zwróć uwagę, że w tym przykładzie używam do uruchomienia pętli litery `a` jako mojej waluty. Gdy „dodajemy” do siebie litery (tak jak w tym przykładzie), rezultat jest następujący: `a`, `aa`, `aaa`, `aaaa`, `aaaaa` itd.

Tablica! Tablica! Tablica!

Wiem, że nie przyglądaliśmy się jeszcze szczegółowo tablicom, ale jedna z największych historii miłosnych wszech czasów rozgrywa się pomiędzy strukturą danych zwaną tablicą a pętlą `for`. W większości przypadków będziesz używać pętli do przechodzenia przez wszystkie dane przechowywane w tablicy, rzućmy więc okiem na poniższy kod i objaśnienie. Odświeżymy te informacje w sposób bardziej szczegółowy, kiedy oficjalnie będziemy omawiać tablice w rozdziale 15.

```
var myArray = ["jeden" , "dwa" , "trzy" ];

for (var i = 0; i < myArray.length; i++) {
    document.writeln(myArray[i]);
}
```

Mówiąc w dużym skrócie i bez przynudzania, tablica jest kolekcją elementów. Wyliczanie i uzyskiwanie dostępu do wszystkich elementów w tablicy wymagają pewnego rodzaju pętli, a zazwyczaj wybierana jest w tym celu pętla `for`.

W każdym razie pomyślałem, że zapoznam Cię z tą parą: tablicą i pętlą `for`. Podobnie jak jest w przypadku dwóch pozornie losowych postaci pojawiających się na początku filmu Quentina Tarantino, w tym momencie ich rzeczywisty wkład do historii wydaje się nieistotny.

O nie, nie zrobił tego!

O tak! Zrobiłem. Poszedłem tam, zrobiłem zdjęcie, zamieściłem je na Facebooku i wróciłem:

```
var i = 0;
var yay = true;

for (; yay;) {
    if (i == 10) {
        yay = false;
    }
    i++;
    document.writeln("dziwne");
}
```

Nie trzeba wypełniać wszystkich trzech sekcji pętli `for`, aby ta pętla działała. Dopóki jesteś w stanie ostatecznie spełnić warunek kończący pętlę, możesz robić, co chcesz — tak jak zrobiłem w powyższym przykładzie. Pamiętaj jednak, że chociaż możesz coś zrobić, to nie znaczy jeszcze, że powinieneś. Ten przykład wyraźnie podpada pod kategorię „nie powinieneś tego robić!”.

Pozostałe pętle

Warianty pętli `while` i `do...while` żyją w cieniu ukochanej pętli `for`. Te dwa warianty pętli wyraźnie służą jakiemuś celowi, ale nigdy nie udało mi się go odkryć. Mimo to w trosce o kompletność i w celu zapoznania Cię z kodem, który można spotkać w naturalnym środowisku, przyjrzyjmy się pokrótce obu z nich.

Pętla while

Pętla `while` powtarza pewien kod, dopóki jego warunek testowy (kolejne wyrażenie) nie zwróci `false`:

```
var count = 0;

while (count < 10) {
    document.writeln("zapętlamy!");

    count++;
}
```

W tym przykładzie ten warunek jest reprezentowany przez wyrażenie `count < 10`. Przy każdej iteracji nasza pętla zwiększa wartość `count` o *1*. Gdy wartość `count` będzie równa 10, pętla zatrzyma się, ponieważ wyrażenie `count < 10` zwróci `false`. To wszystko na temat pętli `while`.

Pętla do...while

Przejdźmy teraz do Meg Griffin wariantów pętli. Będzie to pętla `do...while`, której cel jest jeszcze mniej określony niż cel pętli `while`. Podczas gdy w pętli `while` wyrażenie warunkowe znajduje się na początku przed wykonywanym w niej kodem, pętla `do...while` ma swoje wyrażenie warunkowe na końcu.

Oto przykład:

```
var count = 0;

do {
    count++;

    document.writeln("Nie wiem, co ja tu robię!");
} while (count < 10);
```

Główną różnicą między pętlami `while` oraz `do...while` jest to, że zawartość pętli `while` nigdy nie zostanie wykonana, gdy jej wyrażenie warunkowe od samego początku będzie zwracać `false`:

```
while (false) {
    document.writeln("Tego nie tkniesz!");
}
```

Zawartość pętli `do...while` zawsze zostanie uruchomiona co najmniej raz, ponieważ wyrażenie warunkowe jest ewaluowane dopiero po jednej iteracji:

```
do {
    document.writeln("Ten kod zostanie raz uruchomiony!");
} while (false);
```

Aaa... (ziewnięcie). W każdym razie, zanim przejdziemy dalej, muszę Ci przekazać jeszcze jedną informację. Instrukcje `break` i `continue`, które widzieliśmy wcześniej jako część niesamowitej pętli `for`, działają również podobnie, gdy są stosowane wewnątrz wariantów pętli `while` i `do...while`.

ABSOLUTNE MINIMUM

Proszę bardzo — oto opis pętli `for` i sposobów jej używania oraz bardzo podstawowe omówienie pętli `while` i `do...while`. Na razie prawdopodobnie nie będziesz jeszcze zbyt często używać pętli. Gdy przejdziemy do bardziej skomplikowanych sytuacji obejmujących m.in. kolekcje danych, elementy HTML w dokumencie, manipulacje tekstem, pętle staną się jednym z naturalnych elementów, na których często będziesz polegać.

Skorowidz

M

N

O

P

R

S

T

JavaScript:
dzięki niemu strony WWW stają się piękniejsze, lepsze i bardziej imponujące!

Naukę języka programowania można porównać do nauki języka obcego: jedno i drugie wymaga sporo wysiłku i systematyczności. Trzeba najpierw przyswoić sobie podstawy, potem przejść do bardziej złożonych zagadnień i jak najwięcej ćwiczyć. JavaScript szczególnie dobrze nadaje się do nauki programowania, a przy tym jest językiem elastycznym i wciąż rozwijanym. W połączeniu z CSS3 i HTML5 pozwala na tworzenie wszechstronnych aplikacji internetowych i mobilnych.

Książka, którą trzymasz w dłoni, jest przystępnie napisanym, praktycznym podręcznikiem programowania w JavaScripcie. Zawarte tu wyjaśnienia, wskazówki i proste instrukcje pozwolą Ci na bardzo szybkie przyswojenie sobie podstaw tego języka, nawet jeśli dopiero zaczynasz przygodę z kodowaniem. Po przedstawieniu podstaw programowania wyjaśniono nieco bardziej złożone koncepcje, posługując się licznymi – często bardzo zabawnymi – przykładami. Lektura książki jest przy tym niezwykle interesująca i przyjemna. Po zakończeniu ostatniego rozdziału będziesz przygotowany, by sprostać większości wyzwań związanych z językiem JavaScript.

- Jak zacząć pisać kod i gdzie go umieścić
- Wykorzystanie zmiennych, funkcji i pętli
- Co to jest zakres globalny i lokalny
- Czym są domknięcia
- Programowanie obiektowe w JavaScripcie
- Operacje na danych tekstowych i listy danych

Kirupa Chinnathambi jest menedżerem programowania w Microsofcie. Od dzieciństwa pisze kod i animacje komputerowe, jednak jego prawdziwą pasją jest tworzenie stron WWW. Od kilkunastu lat uczy innych tej sztuki: publikuje poradniki na swojej stronie *kirupa.com*, pisze artykuły i książki, nagrał również sporo filmików, które można znaleźć na stronach serwisu YouTube. W wolnych chwilach grywa w gry wideo, chodzi na koncerty, ogląda filmy i spędza czas z przyjaciółmi.

Helion

księgarnia internetowa
http://helion.pl
zamówienia telefoniczne
0 801 339900
0 601 339900

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 978-83-283-3180-8

cena: 59,00 zł

Informatyka w najlepszym wydaniu

ALWAYS LEARNING
PEARSON