Rysunek 2.1. Konsola przeglądarki Chrome

W oknie konsoli można także wykonywać dowolny program napisany w języku JS. Program to zestaw komend, które uruchamiają się jedna po drugiej tylko wtedy, gdy wykona się poprzednia komenda.

Dzięki lekturze tego i kolejnych rozdziałów tej książki będziesz coraz lepiej rozumieć, jak uruchamiają się poszczególne komendy oraz cała aplikacja.

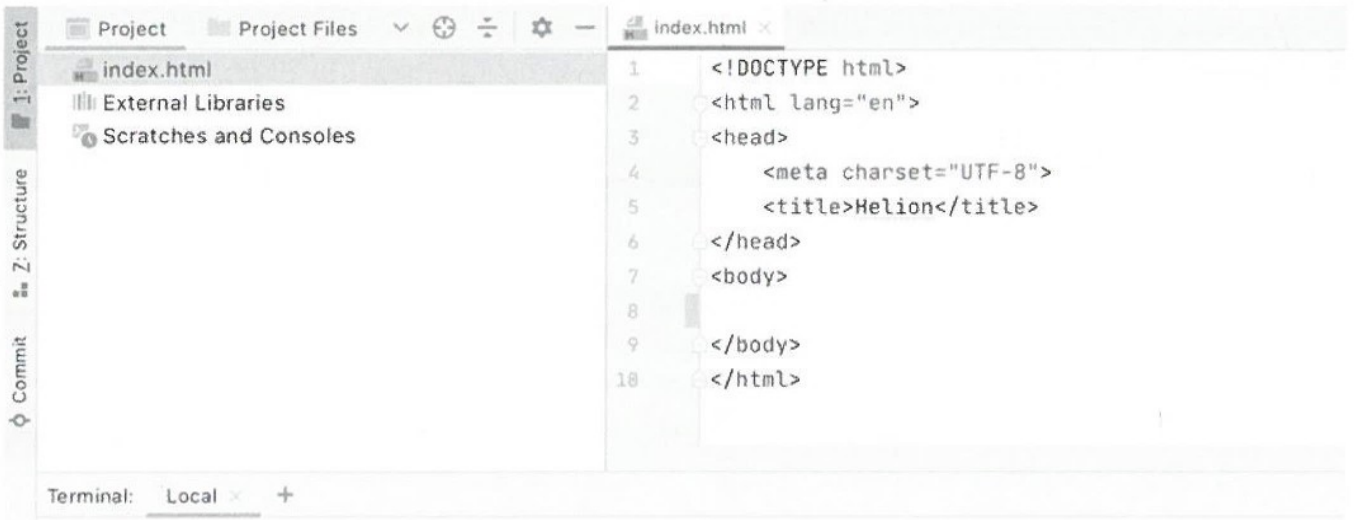
2.2. Składnia

2.2.1. Tagi

Elementem, który jest nam potrzebny do zrozumienia istoty języka JavaScript (a w gruncie rzeczy także HTML), jest **tag**. To pewien znaczący ciąg znaków rozpoczynający się otwierającym nawiasem ostrokątnym (<). Każdy tag musi być ponadto zamknięty. Służy do tego sekwencja znaków ukośnika i zamykającego nawiasu ostrokątnego (/>).

Tagi `<body></body>`, odpowiednio, otwierają i zamykają tzw. ciało strony internetowej. Tagi `<script></script>` wydzielają w obrębie strony skrypt (program) napisany w JS. Tagi mogą być zagnieżdżone. Jak pokazano na poniższym rysunku, tagi `<head></head>` oraz `<body></body>` zawierają się wewnątrz `<html></html>`.

Ponadto, jak widać na rysunku 2.2, w tagu *head* zawarliśmy również dodatkowe informacje o tytule strony i jej kodowaniu.



Rysunek 2.2. Kod domyślnie wygenerowanej strony index.html

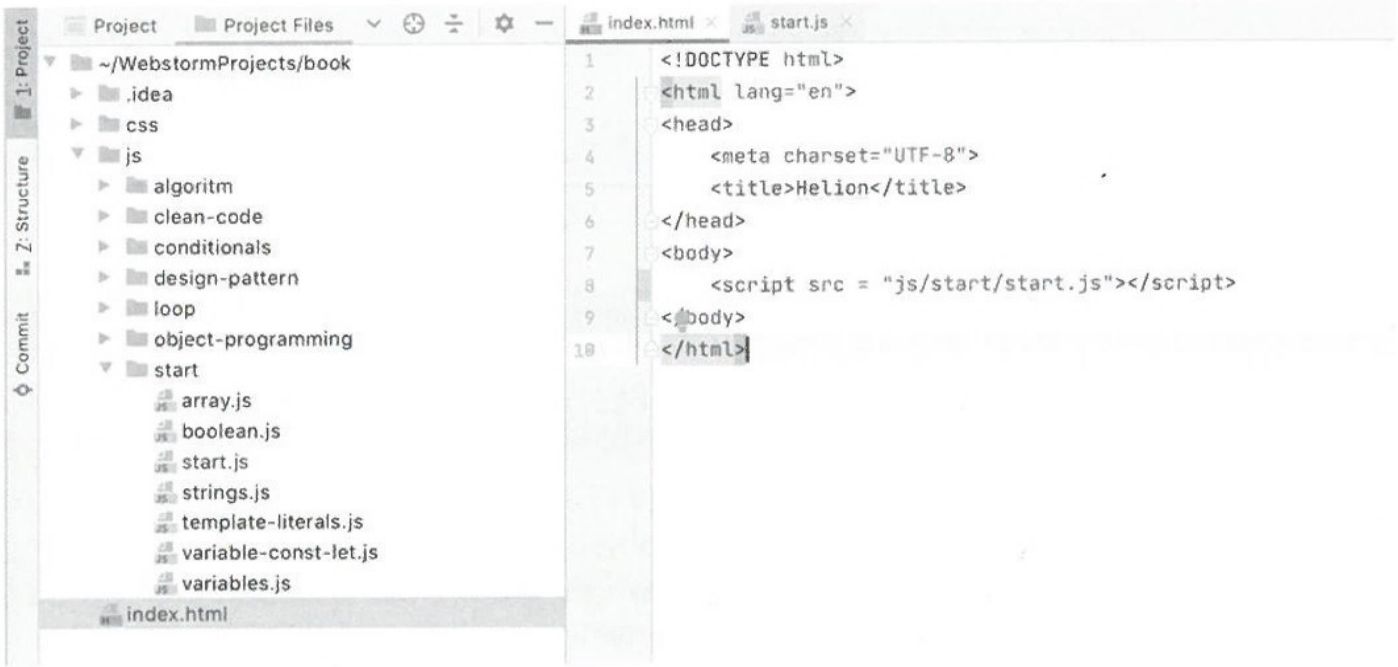
UWAGA

Nie musisz znać dokładnie właściwości wszystkich tagów. Jeśli chcesz się z nimi zapoznać, skorzystaj z ich zestawienia, które możesz znaleźć np. pod adresem *https://www.w3schools.com/tags/tag_html.asp*.

Twoja znajomość tagów będzie się pogłębiać z każdą napisaną przez Ciebie stroną. Im więcej ich napiszesz, tym łatwiejsze będzie kodowanie kolejnych.

2.2.2. Skrypt JS a plik HTML

Teraz do domyślnie wygenerowanej strony wstawimy nasz kod javascriptowy. Na razie zamieścimy pusty plik *start.js* (rysunek 2.3).

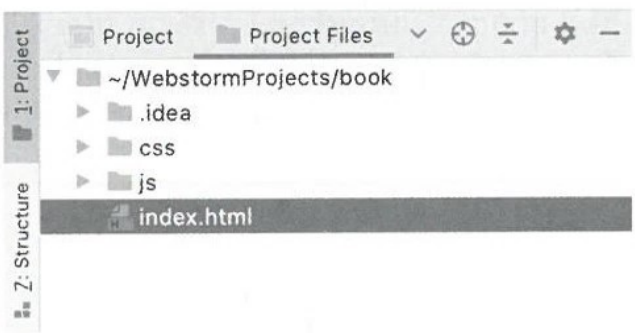


Rysunek 2.3. Skrypt start.js zamieszczony wewnątrz kodu strony index.html

UWAGA

Kody — zarówno HTML, jak i CSS, a przede wszystkim skryptów JS — znaleźć można w serwisie GitHub, pod adresem *https://github.com/weronikakortas/book*.

Aplikacje webowe, jak już wspomniano, składają się z trzech elementów. Każdy z nich dla większej przejrzystości kodu najlepiej jest oddzielać od pozostałych. W projekcie aplikacji będziemy mieli zatem trzy pliki (rysunek 2.4).



Rysunek 2.4. Struktura aplikacji webowej w widoku projektu w WebStorm

W tej chwili skupimy się wyłącznie na *start.js* i *index.html*, pomijając folder *css*, zawierający kaskadowe arkusze stylów.

To kod HTML odpowiada za to, co znajduje się na stronie. W pliku HTML można umieścić jej treść, np. nagłówki, akapity tekstu, listy wypunktowane, obrazki czy odsyłacze do innych stron. JavaScript odpowiedzialny jest za wykonywanie akcji, np. obliczeń czy złożonych interakcji z użytkownikiem. Aby poinstruować przeglądarkę internetową, że ma wykonać napisany przez nas kod JavaScript, w pliku HTML trzeba zawrzeć informację o tym, że na naszej stronie będziemy używać skryptów, które znajdują się w pliku *start.js*. Czynimy to, zamykając ścieżkę do pliku w znacznikach `<script></script>`, jak na rysunku 2.3.

UWAGA

Jak już wspomniano, plik *index.html* to domyślny nowy plik w formacie HTML, generowany za pomocą środowiska WebStorm. Zwróć jednak uwagę, że stanowi on prostą, ale kompletną i działającą aplikację. Wystarczy kliknąć w prawym górnym rogu przeglądarki, by od razu zobaczyć rezultat. Można też znaleźć plik na dysku i uruchomić go za pomocą Chrome (zgodnie z instrukcją zamieszczoną w pierwszym rozdziale).

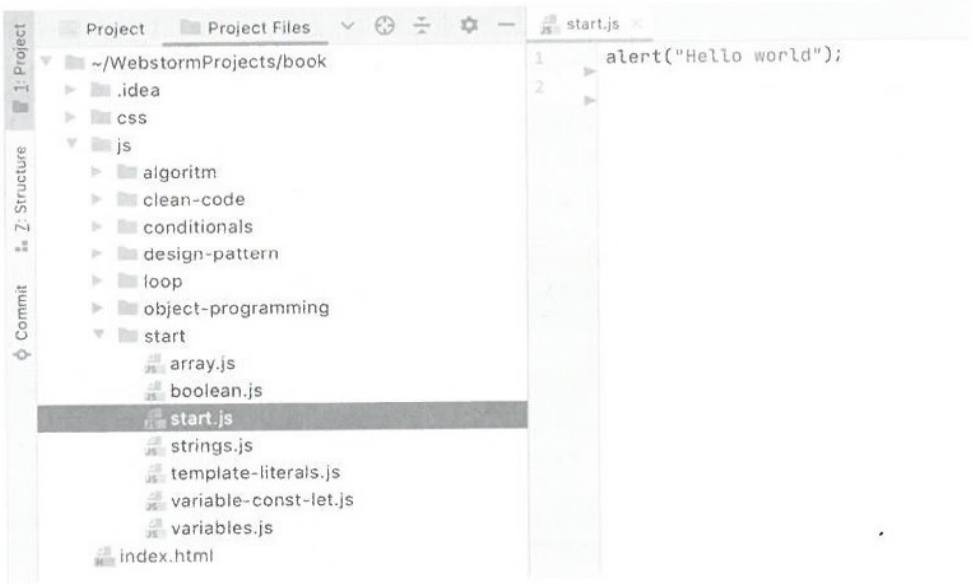
CIEKAWOSTKA

Współpraca między kodem HTML a JavaScriptem nie ogranicza się jedynie do poinstruowania przeglądarki internetowej znacznikiem `<script>` o konieczności wykonania skryptów ze wskazanych plików. Z poziomu skryptów JS możliwe jest odczytywanie treści strony internetowej, wpływanie na jej zawartość i wygląd, jak również reagowanie na działania użytkownika i zdarzenia — np. kliknięcie przycisku lub przewinięcie strony do określonego miejsca. Aby taka współpraca była możliwa, przeglądarki internetowe udostępniają dla JavaScriptu model obiektowy dokumentu (DOM, ang. *Document Object Model*) i model obiektowy przeglądarki (BOM, ang. *Browser Object Model*) — zespoły klas i interfejsów pozwalających na dostęp do zawartości strony internetowej i interakcję z przeglądarką i użytkownikiem z poziomu kodu skryptów.

W tym rozdziale poznasz podstawy składni języka JavaScript, jak również jego bardziej złożone elementy, takie jak obiekty i kolekcje, które ułatwią Ci zaznajomienie się z tymi interfejsami. Jeśli chcesz się z nimi zapoznać, zajrzyj np. pod adres *https://www.w3schools.com/js/js_htmldom.asp*, gdzie znajdziesz obszerne informacje na ten temat.

2.2.3. Wyświetlanie komunikatów

Rozbudujmy teraz nasz skrypt. Na początek niech ma w sobie wyłącznie komunikat informujący o tym, że użytkownik odwiedził naszą stronę. Do wyświetlania takich komunikatów służy polecenie `alert` (rysunek 2.5).



Rysunek 2.5. Instrukcja w skrypcie start.js wyświetlająca powitanie

Niemal każdy początkujący programista zaczyna od komunikatu „Hello world". Pójdźmy tym tropem i my.

Teraz czas dodać nasz alert do już wcześniej działającej aplikacji (rezultatem ma być wykonanie kodu strony i wyświetlenie zapisanego w skrypcie komunikatu).

Aby uzyskać ten efekt, należy nasz skrypt podlinkować do strony *index.html*. Pokazywaliśmy to już wcześniej (na rysunku 2.3); tym razem nasz skrypt nie jest jednak pusty, inna jest też jego lokalizacja (rysunek 2.6).

Rysunek 2.6. Plik index.html z „podpiętym" skryptem start.js

Do dodania skryptu znajdującego się w katalogu *js* należy użyć tagu *script* i dodać do niego właściwość informującą o tym, gdzie ten skrypt się znajduje. Musimy zatem użyć składni `<script src = "nazwa_katalogu/nazwa_skryptu.js"></script>`; w naszym przypadku użyliśmy jej w 8. linii domyślnego pliku HTML.

Po uruchomieniu pliku uzyskamy następujący efekt (rysunek 2.7):

Rysunek 2.7. Okno wyświetlone po wykonaniu skryptu start.js

Komunikat został wyświetlony w oknie alertu, który jest widoczny dla użytkownika odwiedzającego naszą stronę. A co, jeśli chcemy uzyskać informację widoczną także dla nas?

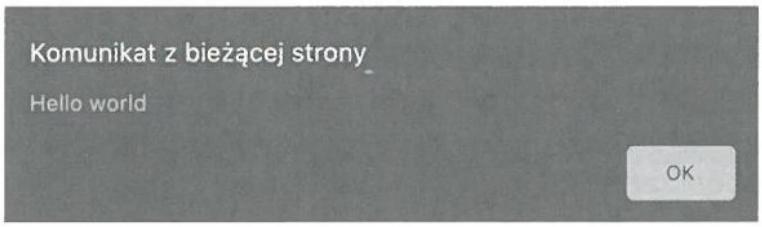
W takim przypadku należy dodać do skryptu wiersz powodujący wysłanie komunikatu do okna konsoli, używając polecenia pokazanego w drugiej linijce na listingu 2.1:

Listing 2.1

Przekierowanie komunikatu do okna konsoli

```
1. alert("Hello world");
2. console.log("Hello world"); // przekierowanie komunikatu do okna konsoli
```

Najpierw pojawi się alert w wyskakującym okienku — to skutek wykonania instrukcji `alert` z pierwszej linijki (rysunek 2.8).



Rysunek 2.8. Okno z komunikatem alertu

Następnie, po kliknięciu *OK*, ten sam komunikat zostanie wyświetlony w konsoli przeglądarki dzięki drugiej instrukcji skryptu (rysunek 2.9).

Rysunek 2.9. Ten sam komunikat wyświetlony w konsoli

2.2.4. Warto korzystać z IDE — mechanizmy autouzupełniania kodu i sygnalizacji błędów

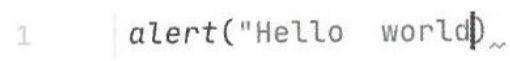
Skoro już wiesz, jak uruchomić nasz przykładowy skrypt, dowiesz się teraz, jak wygląda pisanie poleceń w JavaScripcie: jakich słów należy użyć, kiedy i jakie stawiać punktory (średniki, kropki). Słowem: zapoznasz się z podstawami składni. Oczywiście w tym podręczniku skupimy się tylko na tych jej elementach, które są nam niezbędne. Nie wszystko będzie jasne od razu, więc zalecamy przeczytać poszczególne fragmenty kilkakrotnie, a przede wszystkim od razu pisać je w IDE, np. WebStorm, które można pobrać ze strony jetbrains.com lub z darmowych edytorów, takich jak Atom lub Visual Studio Code. Warto zapisywać sobie elementy składni na kartce w celu jej zapamiętania, ale korzystanie z IDE ma tę zaletę, że udostępnia ono mechanizm uzupełniania składni, a ponadto poprawia (lub przynajmniej sygnalizuje) większość błędów.

Przykład 2.1

Zacznijmy teraz pisać wywołanie `alert` (rysunek 2.10).

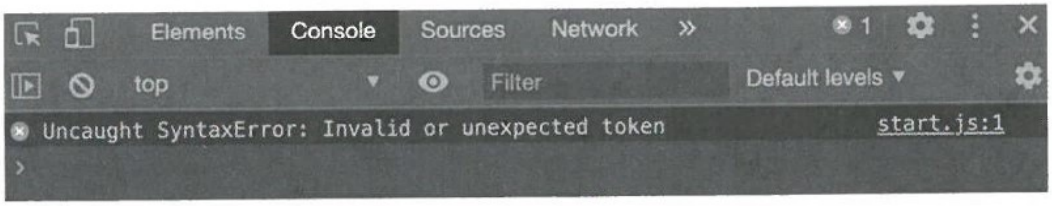
Rysunek 2.10. Mechanizm autouzupełniania poleceń

Po wpisaniu kilku liter pojawiła się podpowiedź sugerująca pełną nazwę funkcji. Jak się można domyślić, `alert` przyjmuje i wyświetla dowolny napis. Napis w JavaScripcie można umieścić w cudzysłowie, przy czym trzeba pamiętać, aby podać zarówno cudzysłów otwierający, jak i zamykający (rysunek 2.11).



Rysunek 2.11. Mechanizm poprawiania składni wskazuje błędy we wpisywanych poleceniach

Jak widać, nasze IDE już podpowiada nam, że coś jest nie tak. My możemy jednak tego nie zauważyć i mimo to uruchomić nasz skrypt. Próba wykonania kodu skończy się wówczas błędem (rysunek 2.12).



Rysunek 2.12. Wyświetlony w konsoli komunikat o błędzie

Akurat w tym przypadku zabrakło cudzysłowu zamykającego.

UWAGA

Liczba cudzysłowów w skrypcie musi być parzysta — każdy cudzysłów musi mieć zakończenie.

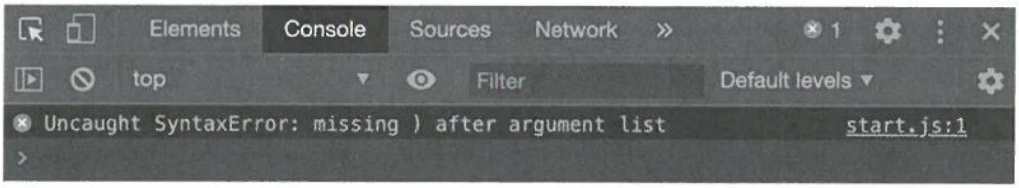
Przykład 2.2

Spróbujmy teraz uruchomić skrypt napisany w taki oto sposób (rysunek 2.13):

```
1    alert("Hello world"
```

Rysunek 2.13. Mechanizm poprawiania składni wskazuje błędy we wpisywanych poleceniach (cd.)

Tym razem wpisaliśmy cudzysłów zamykający. Jednak, jak widać, IDE podkreśliło miejsce za ostatnim wpisanym znakiem — właśnie cudzysłowem. To sugeruje, że i tym razem popełniliśmy błąd. Aby się o tym przekonać, znów uruchomimy skrypt i zobaczymy, co pojawi się w konsoli (rysunek 2.14).



Rysunek 2.14. Komunikat o błędzie informuje, że na końcu polecenia zabrakło nawiasu zamykającego

UWAGA

Każdy nawias (czy to kwadratowy, czy półokrągły, czy klamrowy) musi być domknięty. Podobnie jak w przypadku cudzysłowów, liczba nawiasów musi być zatem parzysta. Ponadto nawiasy muszą być zamykane w kolejności odwrotnej do tej, w której były otwierane: `{ ( [ ] ) }`.

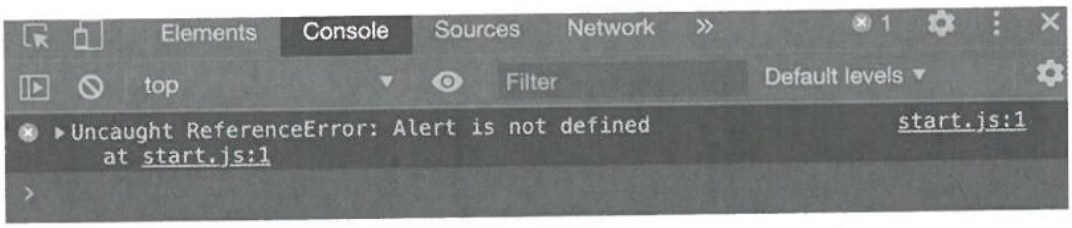
Przykład 2.3

Spróbujmy zapisać słowo `alert`, zaczynając od wielkiej litery (rysunek 2.15).

```
1    Alert("Hello world");
```

Rysunek 2.15. Wielkość liter w JS ma znaczenie

Uzupełniliśmy do pary nawiasy i cudzysłowy, nic nie zostało podkreślone na czerwono. Czy zatem wykonamy kod bez błędu? Zobaczmy (rysunek 2.16).



Rysunek 2.16. Interpreter nie rozpoznaje słowa „Alert" jako nazwy metody

Tym razem wyświetlony został komunikat z informacją „Alert is not defined". Oznacza to, że dane słowo nie występuje wśród dostępnych możliwych wywołań, w szczególności nie zostało rozpoznane jako polecenie `alert`.

UWAGA

Wielkość liter w JS ma znaczenie.

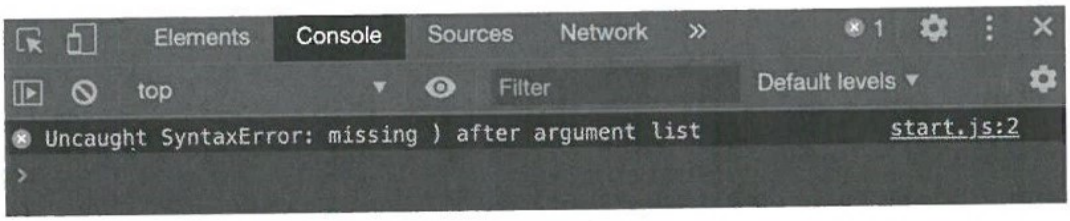
Przykład 2.4

Spróbujmy teraz wypisać „Hello world" i w konsoli, i w oknie alertu. Użyjmy poleceń z rysunku 2.17 (jeśli widzisz w nich błędy, to dobrze; na razie jednak nie przejmujmy się nimi):

```
1  Alert("Hello world");
2  console.log("Hello world"
```

Rysunek 2.17. Jeśli w skrypcie popełnimy dwa błędy...

Po uruchomieniu tak napisanego skryptu uzyskamy w konsoli komunikat pokazany na rysunku 2.18.



Rysunek 2.18. ...po jego uruchomieniu zostanie zgłoszony jeden z nich

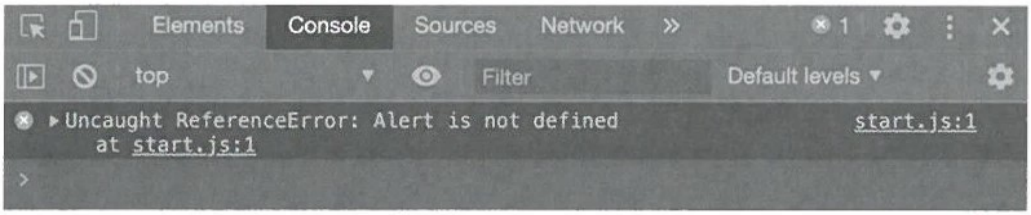
Informuje on nas o tym, że brakuje nawiasu w drugiej linijce (zgodnie z informacją zawartą na końcu linijki — *start.js:2*).

Poprawmy ten błąd i odświeżmy stronę (rysunek 2.19).

```
1  Alert("Hello world");
2  console.log("Hello world");
```

Rysunek 2.19. Po poprawieniu zgłoszonego błędu...

Tym razem w konsoli otrzymaliśmy kolejną informację o błędzie (rysunek 2.20).



Rysunek 2.20. ...zgłoszony zostanie kolejny błąd

Ten błąd także widzieliśmy już wcześniej, wiemy więc, na czym polega. Wpisaliśmy `Alert`, a poprawnym poleceniem jest `alert`.

Po poprawieniu tego błędu wreszcie pojawi się okno alertu, a w konsoli otrzymamy (rysunek 2.21):

Rysunek 2.21. Poprawny wynik skryptu wyświetlony w konsoli

Wynika z tego, że **błędy w konsoli pojawiają się kolejno, zgodnie z występowaniem w kodzie, a zatem nawet po usunięciu jednego błędu może pojawić się drugi.**

Widzieliśmy ponadto, że komendy wykonują się po kolei. W konsoli napis pojawił się dopiero po kliknięciu *OK* w alercie z przeglądarki.

Co więcej, można było zauważyć, że w JS, inaczej niż w wielu innych językach programowania, średnik nie kończy polecenia. Możemy go postawić na końcu instrukcji, ale nie musimy. To już zależy od tego, w jaki sposób łatwiej będzie zrozumieć, gdzie kończy się polecenie. Jeśli wyraźnie widać, że jedno polecenie przypada na jedną linijkę, to możemy nie stosować średnika na końcu, jednak nie powinno się łączyć dwóch podejść i jeśli stosujemy zapis bez średników, to należy go stosować wszędzie.

2.2.5. Komentarze

W celach edukacyjnych będziemy stosowali komentarze, by opisać, co w którym miejscu kodu się dzieje. Ich stosowanie jest dobrą praktyką programistyczną. Ułatwiają orientację w kodzie — zarówno własnym, kiedy wracamy do niego po dłuższej przerwie, jak i (tym bardziej) cudzym. Oczywiście nie ma sensu wyjaśniać w komentarzach każdej linii kodu, nie powinniśmy też tego robić w przypadku tych fragmentów, których przeznaczenie jest dla osób znających składnię języka oczywiste. Trzeba ponadto pamiętać, że w praktyce kod powinien być pisany tak, by komentarze były zbędne. Nazwy zmiennych i metod powinny być na tyle jasne, by osoba czytająca kod nie potrzebowała dodatkowych wyjaśnień.

Wyróżniamy dwa rodzaje komentarzy:

- komentarze jednolinijkowe, które rozpoczynamy dwoma ukośnikami (//); pomiędzy tymi znakami nie może występować spacja ani żaden inny znak (listing 2.2);

Listing 2.2

Komentarz jednolinijkowy

```
// Komentarz jednolinijkowy
```

- komentarze zajmujące wiele linii, które umieszczamy pomiędzy znakami /* a */ (listing 2.3).

Listing 2.3

Komentarz składający się z wielu linii

```
/* Komentarz
wielolinijkowy
*/
```

Jeśli poprawnie udało nam się napisać ten element kodu, to jest on wyszarzony.

Warto być świadomym, że nic spośród tego, co jest komentarzem, nie ma wpływu na działanie programu — jest to podczas jego wykonywania pomijane. Z tego względu dzięki komentarzom możemy w łatwy sposób „wyłączać" fragmenty kodu, których wykonywanie chcielibyśmy chwilowo zatrzymać (np. niedokończone lub potencjalnie błędne).

2.3. Podstawy programowania

Aby móc stworzyć pełną aplikację, trzeba najpierw nauczyć się podstaw programowania. Pewne podstawowe informacje związane ze składnią języka już zaprezentowaliśmy. Ten rozdział stanowi wprowadzenie w tajniki programowania; zasady tu poznane wykorzystamy w ostatnim rozdziale.

Nie jest to pełne kompendium wiedzy dotyczącej programowania w języku JavaScript. Jest to zaledwie niezbędne minimum, pozwalające na stworzenie prostej aplikacji webowej.

2.3.1. Zmienne (var)

UWAGA

W tym i kilku innych punktach rozdziału skupiamy się na opisaniu deklarowania zmiennych i tworzenia obiektów w języku JavaScript. Należy tutaj podkreślić, że w innych językach (choć nie we wszystkich) deklarowanie odbywa się poprzez jawne oznaczenie, jakiego typu zmienną chcemy utworzyć. Język dynamicznie typowany, taki jak JavaScript, nie wymusza na programiście deklaracji typu zmiennej, lecz pozostawia interpretację zmiennej interpreterowi na podstawie jej zawartości.

→