Spis treści

Część II. Funkcjonalności

Wstęp

Moja przygoda z TypeScriptem zaczęła się w nieoczywisty sposób. Programować zacząłem w szkole, najpierw w języku Java, a potem w C++, i jak wielu początkujących programistów, którzy wychowali się na językach typowanych statycznie, postrzegałem JavaScript jako „tylko" niechlujny język skryptowy, który umieszcza się w witrynach internetowych.

Moim pierwszym poważniejszym projektem napisanym w tym języku była naiwna przeróbka gry wideo *Super Mario Bros.* Napisałem ją korzystając tylko z technologii HTML5, CSS i JavaScript. Jak to zwykle bywa z wieloma pierwszymi projektami, był to absolutny bałagan. Na początku projektu instynktownie poczułem niechęć do dziwnej elastyczności JavaScriptu i braku ograniczeń. Dopiero pod koniec zacząłem dostrzegać potencjał funkcjonalności i dziwactw JavaScriptu: jego elastyczności jako języka, możliwości łączenia małych funkcji oraz tego, że *po prostu działał* w przeglądarkach od razu po wczytaniu strony.

Zanim ukończyłem pracę nad projektem, zakochałem się w języku JavaScript.

Narzędzia do analizy statycznej (które analizują kod bez jego uruchamiania), takie jak TypeScript, również początkowo przyprawiły mnie o mdłości. *JavaScript jest tak lekki i płynny*, myślałem, *czemu zatem ograniczać się sztywnymi strukturami i typami?* Czy wracałem do świata języków Java i C++, który zostawiłem za sobą?

Gdy wróciłem do swoich dawnych projektów, przez całe 10 minut zmagałem się z czytaniem swojego dawnego, zagmatwanego kodu JavaScript, próbując zrozumieć, jaki bałagan może wyniknąć z braku analizy statycznej. Sprzątając kod, odkryłem wszystkie miejsca, w których przydałaby się pewna struktura. Wtedy to zdecydowałem się wprowadzić analizę statyczną do swoich projektów na tyle, na ile mogłem.

Upłynęła niemal dekada, odkąd zacząłem używać języka TypeScript, a nadal korzystam z niego z ogromną przyjemnością. Język ten nadal się rozwija, pojawiają się nowe funkcjonalności. Coraz bardziej przydaje się w zapewnianiu *bezpieczeństwa* i *struktury* językowi JavaScript.

Mam nadzieję, że po przeczytaniu książki *Poznaj TypeScript* będziesz traktował TypeScript tak samo jak ja: nie tylko jako narzędzie do znajdowania błędów i literówek – a z pewnością nie jako znaczącą zmianę we wzorcach kodu języka JavaScript – ale jako JavaScript *z typami*: piękny system służący do deklarowania sposobu, w jaki powinien działać język JavaScript i ułatwiający przestrzeganie tej konwencji.

Kto powinien przeczytać tę książkę

Powinieneś przeczytać tę książkę, jeśli umiesz pisać kod JavaScript, wiesz, jak wykonać podstawowe polecenia w terminalu i chcesz się nauczyć języka TypeScript.

Być może słyszałeś, że TypeScript ułatwia pisanie dużej ilości kodu JavaScript zawierającego mniejszą liczbę błędów *(to prawda!)* lub tworzenie dobrej dokumentacji kodu na użytek przyszłych czytelników *(również prawda!)*. Może zauważyłeś, że TypeScript pojawia się w wielu ogłoszeniach o pracę lub jest wymagany na nowym stanowisku pracy.

Niezależnie od powodu, jeśli znasz podstawy języka JavaScript – zmienne, funkcje, domknięcia, zakresy i klasy – ta książka przeprowadzi Cię od poziomu zupełnie początkującego użytkownika języka TypeScript do poziomu, na którym opanujesz podstawy i najważniejsze funkcje tego języka. Po przeczytaniu tej książki będziesz się mógł pochwalić znajomością następujących zagadnień:

- Historii i okoliczności, które sprawiły, że TypeScript stał się przydatnym dodatkiem do czystego języka JavaScript
- Jak system typów modeluje kod
- Jak narzędzie do sprawdzania typów analizuje kod
- Jak używać adnotacji typów podczas prac programistycznych, aby zapewnić informacje systemowi typów
- Jak TypeScript współpracuje ze środowiskami IDE (Integrated Development Environments) w celu dostarczenia narzędzi do eksploracji i refaktoryzacji kodu

Będziesz też wiedział, jak:

- Opisać korzyści wynikające z używania języka TypeScript oraz ogólne cechy jego systemu typów.
- Dodawać adnotacje typów tam, gdzie są potrzebne w kodzie.
- Reprezentować średnio skomplikowane typy za pomocą wbudowanych mechanizmów wnioskowania i nowej składni języka TypeScript.
- Używać narzędzi TypeScript w lokalnym środowisku podczas refaktoryzacji kodu.

Dlaczego napisałem tę książkę

TypeScript jest niesłychanie popularnym językiem zarówno w firmach, jak i w społeczności open source:

- Według raportu State of the Octoverses z lat 2021 i 2020, publikowanego przez GitHub, był czwartym najpopularniejszym językiem w używanym w tym portalu. Wcześniej, w latach 2019 i 2018 zajmował siódmą pozycję, a w 2017 – dziesiątą.
- Zgodnie z raportem Developer Survey platformy StackOverflow z 2021, był to trzeci najbardziej lubiany język na świecie (72.73% użytkowników).
- Raport State of JS Survey z 2020 pokazuje, że TypeScript cieszy się stałym uznaniem i jest powszechnie wykorzystywany jako narzędzie do budowania i jako odmiana języka JavaScript.

Programiści frontendu cenią TypeScript za dobrą obsługę we wszystkich najważniejszych bibliotekach i platformach interfejsu użytkownika, włącznie z frameworkiem Angular, który promuje używanie języka TypeScript. Jest obsługiwany również przez frameworki Gatsby, Next.js, React, Svelte i Vue. Programiści backendu doceniają TypeScript za możliwość generowania kodu JavaScript, działającego natywnie w platformie Node.js; w Deno, podobnym środowisku uruchomieniowym, opracowanym przez twórcę Node, podkreśla się bezpośrednią obsługę plików TypeScript.

Mimo olbrzymiego wsparcia w wielu popularnych projektach, w czasie, gdy zacząłem poznawać TypeScript, z rozczarowaniem stwierdziłem, że brakuje dobrych materiałów online, wprowadzających w tajniki tego języka. W wielu materiałach dokumentacyjnych online brakowało porządnego wyjaśnienia, czym jest „system typów" i jak go używać. Zwykle zakładano, że czytelnik zna dobrze zarówno JavaScript, jak i silnie typowane języki. W innych źródłach podawano jedynie powierzchowne przykłady kodu.

Kilka lat temu szukałem książki wydawnictwa O'Reilly ze ślicznym zwierzęciem na okładce, zawierającej wprowadzenie do języka TypeScript, ale się zawiodłem. Zanim napisałem tę książkę, pojawiły się inne książki o języku TypeScript, wydane między innymi przez O'Reilly, jednak nie udało mi się znaleźć takiej, która opisywałaby podstawy języka, tak jak chciałem: dlaczego działa w określony sposób i jak współpracują ze sobą jego główne funkcjonalności. Brakowało mi książki, która zaczynałaby się od opisu podstawowych zasad rządzących językiem, a dopiero później opisywałaby po kolei następne funkcjonalności. Jestem zadowolony, że udało mi się napisać jasne, wyczerpujące wprowadzenie do podstaw języka TypeScript, przeznaczone dla czytelników, którzy nie znają jeszcze rządzących nim reguł.

Nawigacja w książce

Poznaj TypeScript służy do dwóch celów:

- Pierwszy raz należy ją przeczytać, aby zrozumieć całościowo język TypeScript.
- Później można do niej wracać, traktując ją jako praktyczny przewodnik po języku TypeScript.

Ta książka zawiera trzy części, w których prowadzi czytelników od podstawowych koncepcji po praktyczne sposoby użycia:

- Część I, „Koncepcje": Historia języka JavaScript, co wnosi do niego TypeScript oraz zasady działania *systemu typów* w języku TypeScript.
- Część II, „ Funkcjonalności": Opisuje, jak system typów współpracuje z najważniejszymi funkcjami języka JavaScript, z których korzysta się podczas pisania kodu TypeScript.
- Część III, „Użycie": Po zapoznaniu się z funkcjonalnościami języka TypeScript, pokazano, jak z niego skorzystać w rzeczywistych sytuacjach, aby poszerzyć umiejętności czytania i edycji kodu.

Na końcu książki znajduje się część IV, „Dodatkowe punkty", która opisuje rzadziej używane, ale czasem przydatne funkcjonalności języka TypeScript. Nie trzeba ich dobrze znać, aby być programistą TypeScript. Są to jednak przydatne koncepcje, z którymi prawdopodobnie każdy zetknie się pracując nad rzeczywistymi projektami w języku TypeScript. Po zapoznaniu się z pierwszymi trzema częściami, naprawdę warto przeczytać tę ostatnią, dodatkową część.

Każdy rozdział zaczyna się wierszem haiku, który wprowadza w nastrój potrzebny do zrozumienia opisanej tematyki. Społeczność programistów aplikacji internetowych jako całość oraz należąca do niej społeczność języka TypeScript słynie z poczucia humoru i przyjaznego nastawienia do początkujących programistów. Chciałem, aby ta książka była atrakcyjna dla czytelników, którzy tak jak ja nie lubią długich, suchych wywodów.

Przykłady i projekty

W przeciwieństwie do wielu innych zasobów zawierających wprowadzenie do języka TypeScript, ta książka celowo koncentruje się na omawianiu funkcjonalności języka z użyciem samodzielnych przykładów, które ilustrują tylko nowe koncepcje, a nie rozwija średnich, ani dużych projektów. Wolę ten sposób nauczania, ponieważ w ten sposób można się skupić przede wszystkim na języku TypeScript. Język ten jest używany w tylu frameworkach i platformach – z których wiele otrzymuje regularne aktualizacje API – że nie chciałem uzależniać tej książki od żadnego frameworka i platformy.

Biorąc to pod uwagę, podczas nauki języka programowania niezwykle ważne jest przećwiczenie poznanych koncepcji od razu po ich poznaniu. Gorąco zalecam przerwę

po przeczytaniu każdego rozdziału, którą warto poświęcić na przećwiczenie poznanych koncepcji. Każdy rozdział kończy się sugestią odwiedzin w odpowiedniej sekcji witryny *https://learningtypescript.com* w celu wykonania wymienionych tam ćwiczeń i projektów.

Konwencje stosowane w książce

W tej książce stosowane są następujące konwencje typograficzne:

Kursywa
: Oznacza nową terminologię, adresy URL i e-mail, nazwy i rozszerzenia plików.

`Znaki o stałej szerokości`
: Używane w listingach programów oraz w akapitach do oznaczenia elementów programu, takich jak nazwy zmiennych lub funkcji, typów danych, instrukcji i słów kluczowych.

Ten element jest wskazówką lub sugestią.

Ten element jest ogólną uwagą.

Ten element jest ostrzeżeniem.

Wykorzystywanie przykładowego kodu

Z witryny *https://learningtypescript.com* można pobrać dodatkowe materiały (przykładowy kod, ćwiczenia, itp.).

Pytania techniczne lub problemy związane z przykładami kodu należy przesyłać na adres *bookquestions@oreilly.com*.

Celem tej książki jest pomoc czytelnikom w wykonywaniu swojej pracy. Przykładowy kod z tej książki można wykorzystać w swoich programach i dokumentacji. Nie trzeba nas prosić o zezwolenie, chyba że ktoś zamierza powielić znaczną część kodu. Przykładowo, jeśli ktoś pisze program wykorzystujący kilka fragmentów kodu z tej książki, nie potrzebuje zezwolenia. Sprzedaż lub dystrybucja przykładów z książki wydanej przez O'Reilly wymaga zezwolenia. Odpowiedź na pytanie poprzez cytowanie tej książki oraz przykładowego

kodu nie wymaga zezwolenia. Umieszczenie znacznej części przykładowego kodu z tej książki w dokumentacji swojego produktu wymaga zezwolenia.

Doceniamy, ale nie wymagamy podawania źródeł. Zwykle wymaga to podania tytułu, autora, wydawnictwo i numer ISBN. Np.: „*Poznaj Typescript* autorstwa Josha Goldberga (Promise). Copyright 2022 Josh Goldberg, 978-1-098-11033-8 ". Jeśli uważasz, że zamierzasz wykorzystać fragmenty kodu w sposób wykraczający poza wymienione wcześniej ramy, napisz do nas na adres *permissions@oreilly.com*.

Podziękowania

Ta książka jest wynikiem pracy całego zespołu i chciałbym podziękować wszystkim, którzy przyczynili się do jej wydania. Przede wszystkim chcę podziękować swojej redaktor prowadzącej, Ricie Fernando, za niesłychaną cierpliwość i doskonałe wskazówki podczas pracy. Pragnę również podziękować pozostałym pracownikom wydawnictwa O'Reilly: Kristen Brown, Suzanne Huston, Clare Jensen, Carol Keller, Elizabeth Kelly, Cheryl Lenser, Elizabeth Oliver i Amandzie Quinn. Jesteście wspaniali!

Głębokie podziękowania kieruję ku recenzentom technicznym, którzy czuwali nad warstwą pedagogiczną oraz służyli ekspercką wiedzą o języku TypeScript. Byli to: Mike Boyle, Ryan Cavanaugh, Sara Gallagher, Michael Hoffman, Adam Reineke i Dan Vanderkam. Bez Waszej pomocy ta książka nie byłaby taka sama. Mam nadzieję, że udało mi się z powodzeniem uwzględnić Wasze wskazówki!

Dodatkowe podziękowania kieruję ku tym, którzy zrecenzowali tę książkę i pomogli mi w uzyskaniu lepszej jakości technicznej i prozatorskiej. Byli to: Robert Blake, Andrew Branch, James Henry, Adam Kaczmarek, Loren Sands-Ramshaw, Nik Stern i Lenz Weber-Tronic. Każda wskazówka się liczy!

Na koniec pragnę podziękować swojej rodzinie, za miłość i wsparcie przez wszystkie lata. Moim rodzicom, Frances i Markowi oraz bratu Danny'emu dziękuję za czas spędzony na układaniu klocków Lego, czytaniu książek i grze w gry wideo. Mojej żonie Mariah Goldberg dziękuję za cierpliwość podczas moich długich godzin poświęconych redakcji i pisaniu. Moim kotom Luci, Tiny i Jerry'emu dziękuję za wyjątkową miękkość i za dotrzymywanie mi towarzystwa.

Część I

Koncepcje

Rozdział 1

Od języka JavaScript do TypeScript

Dzisiejszy JavaScript
obsługuje przeglądarki sprzed dziesięcioleci
Piękno internetu

Zanim zaczniesz się uczyć języka TypeScript, powinieneś poznać jego korzenie, czyli język JavaScript!

Historia języka JavaScript

Język JavaScript został opracowany w ciągu 10 dni przez Brendana Eicha z firmy Netscape w 1995 r. Miał być prosty oraz łatwy w użyciu i służyć do tworzenia witryn internetowych. Od tego czasu programiści nie ustają w wyśmiewaniu jego dziwactw i mankamentów. Niektóre z nich omówię w następnym podrozdziale.

Jednak od 1995 roku JavaScript przeszedł ogromną ewolucję! Komitet sterujący jego rozwojem, zwany TC39, od 2015 roku wydaje co roku nową wersję ECMAScriptu – specyfikacji języka, na której opiera się JavaScript – wzbogaconą o nowe funkcjonalności, które dostosowują go do innych nowoczesnych języków. Na uwagę zasługuje fakt, że regularnie wydawane nowe wersje języka JavaScript przez dziesięciolecia zachowały zgodność wsteczną z różnymi środowiskami, takimi jak przeglądarki, aplikacje wbudowane i środowiska uruchomieniowe na serwerach.

Obecnie JavaScript jest niesłychanie elastycznym językiem, który ma wiele mocnych stron. Warto docenić, że chociaż ma swoje dziwactwa, dzięki niemu możliwy był niesłychany rozwój aplikacji WWW i internetu.

> Pokaż mi doskonały język programowania, a pokażę Ci język, którego nikt nie używa.
>
> *– Anders Hejlsberg, TSConf 2019*

Problemy z czystym językiem JavaScript

Programiści zwykle określają język JavaScript bez poważniejszych rozszerzeń lub frameworków jako „vanilla", czyli czysty. Jest to oryginalna wersja tego języka. Niebawem wyjaśnię, dlaczego TypeScript jest dokładnie tym dodatkiem, który rozwiązuje opisane tu najważniejsze problemy. Warto jednak zrozumieć ich istotę. Wszystkie te słabe strony dają się we znaki tym dotkliwiej, im większy i dłuższy jest projekt.

Kosztowna wolność

Największą bolączką wielu programistów JavaScriptu jest niestety jedna z głównych cech tego języka: w JavaScripcie nie ma właściwie żadnych ograniczeń dotyczących struktury kodu. Dzięki temu rozpoczynanie nowego projektu w JavaScripcie jest świetną zabawą!

Jednak w miarę tworzenia dodatkowych plików, da się zauważyć, jak szkodliwa może się okazać ta wolność. Przeanalizujmy poniższy, wyrwany z kontekstu, fragment pewnej fikcyjnej aplikacji do rysowania:

```
function paintPainting(painter, painting) {
  return painter
    .prepare()
    .paint(painting, painter.ownMaterials)
    .finish();
}
```

Widząc ten kod bez żadnego kontekstu można mieć tylko ogólny pomysł na wywołanie funkcji `paintPainting`. Jeśli ktoś pracował nad kodem tej aplikacji, może sobie przypomnieć, że `painter` jest zwracany przez inną funkcję `getPainter`. Być może uda się zgadnąć, że `painting` jest ciągiem tekstowym.

Nawet jeśli to prawda, późniejsze zmiany w kodzie mogą unieważnić początkowe założenia. Być może `painting` nie będzie już ciągiem tekstowym, ale innym typem danych, a może nazwa jednej lub kilku metod obiektu `painter` ulegnie zmianie.

Aplikacje napisane w innych językach mogą się nie uruchomić, jeśli kompilator uzna, że grozi im awaria. Nie dotyczy to języków typowanych dynamicznie – których kod zostanie uruchomiony bez sprawdzania ryzyka wystąpienia awarii – takich jak JavaScript.

Wolność, dzięki której pisanie kodu w języku JavaScript jest tak atrakcyjne, może się okazać ogromnym problemem, jeśli zależy nam na bezpieczeństwie działania kodu.

Luźna dokumentacja

W specyfikacji języka JavaScript brakuje formalnych zasad opisywania przeznaczenia parametrów funkcji, wartości zwracanych, zmiennych i innych konstrukcji kodu. Wielu programistów kieruje się standardem JSDoc, według którego funkcje i zmienne opisuje się za pomocą komentarzy blokowych. Standard JSDoc dotyczy tworzenia komentarzy

dokumentacji tuż nad konstrukcjami, takimi jak funkcje i zmienne, oraz ich formatowania. Oto przykład, również wyjęty z kontekstu:

```
/**
 * Sprawia, że painter maluje określony obraz (painting).
 *
 * @param {Painting} painter
 * @param {string} painting
 * @returns {boolean} Czy painter namalował obraz (painting).
 */
function paintPainting(painter, painting) { /* ... */ }
```

Standard JSDoc ma pewne niedogodności, które utrudniają jego użycie w dużych projektach:

- Nic nie wymusza poprawnego opisu kodu w dokumentacji JSDoc.
- Nawet, jeśli opisy JSDoc były wcześniej poprawne, podczas refaktoryzacji kodu można napotkać na trudności w znalezieniu wszystkich nieważnych komentarzy JSDoc związanych z wprowadzanymi zmianami.
- Opisywanie złożonych obiektów jest nieefektywne i rozwlekłe. Wymaga wielu osobnych komentarzy definiujących typy i relacje między nimi.

Utrzymanie komentarzy JSDoc w kilkunastu plikach nie jest zbyt czasochłonne, ale w przypadku setek lub tysięcy stale aktualizowanych plików może być wyzwaniem.

Słabsze narzędzia programistyczne

Ponieważ JavaScript nie oferuje wbudowanych sposobów na identyfikację typów, a kod łatwo wymyka się opisom z komentarzy JSDoc, automatyzacja wielkich zmian lub uzyskanie wglądu w kod źródłowy może być trudne. Programiści języka JavaScript zwykle są zdumieni cechami języków typowanych, takich jak C# i Java, dzięki którym można zmieniać nazwy członków klas lub przeskakiwać do miejsca deklaracji typu argumentu.

Być może zaprotestujesz twierdząc, że nowoczesne środowiska programistyczne IDE, takie jak VS Code oferują pewne narzędzia programistyczne, np. automatyczną refaktoryzację kodu JavaScript. To prawda, ale w tym celu w tle wykorzystują TypeScript lub jego odpowiednik. Ponadto, w przypadku większości kodu JavaScipt nie są tak solidne jak dobrze napisany kod TypeScript.

TypeScript!

TypeScript został opracowany w firmie Microsoft na początku drugiej dekady XXI w., a w 2012 został opublikowany i udostępniony na zasadach open source. Na czele prac rozwojowych tego języka stoi Anders Hejlsberg, który przewodził również pracom nad popularnymi językami C# i Turbo Pascal. TypeScript jest zwykle opisywany jako „nadzbiór JavaScriptu" lub „typowany JavaScript". Czym jednak *jest* TypeScript?

TypeScript odgrywa następujące cztery role:

Język programowania
: Język obejmujący całą istniejącą składnię języka JavaScript oraz dodatkowo nową składnię specyficzną dla języka TypeScript, służącą do definiowania i używania typów.

Narzędzie do sprawdzania typów
: Program, który przetwarza zbiór plików napisanych w języku JavaScript i (lub) TypeScript, aby zrozumieć wszystkie utworzone konstrukcje (zmienne, funkcje…) i informuje o znalezionych błędach w ich konfiguracji.

Kompilator
: Program, który uruchamia narzędzie do sprawdzania typów, zgłasza znalezione błędy, a następnie zwraca odpowiednik w postaci kodu JavaScript.

Usługa językowa
: Program wykorzystujący narzędzie do sprawdzania typów, używany przez edytory kodu, takie jak VS Code do udostępniania przydatnych funkcji programistom.

Korzystanie z TypeScript Playground

Przeczytałeś już sporo o języku TypeScript. Czas, aby z niego skorzystać!

Na głównej stronie witryny języka TypeScript dostępny jest edytor *https://www.typescript lang.org/play*, pełniący rolę swego rodzaju placu zabaw. W głównym edytorze możesz napisać kod i uzyskać sugestie, które zostałyby wyświetlone w lokalnym pełnym środowisku IDE (Integrated Development Environment) podczas pisania kodu TypeScript.

Większość fragmentów w tej książce celowo jest niewielka i stanowi pewną całość. Dzięki temu można je wpisać we wspomnianym edytorze i przećwiczyć ich działanie.

TypeScript w działaniu

Przeanalizuj poniższy kod:

```
const firstName = "Georgia";
const nameLength = firstName.length();
//                           ~~~~~~
// This expression is not callable (To wyrażenie nie jest wywoływalne).
```

Jest to kod napisany za pomocą zwykłej składni JavaScript – jeszcze nie korzystam ze składni języka TypeScript. Po przeanalizowaniu tego kodu narzędzie do sprawdzania typów języka TypeScript ustali, że właściwość `length` ciągu tekstowego jest liczbą – a nie funkcją – i wyświetli komunikat pokazany w komentarzu[1].

Po wklejeniu tego kodu w edytorze lub w środowisku Playground, usługa językowa podkreśli słowo `length` czerwoną falistą linią, informując, że nie jest to poprawny kod TypeScript. Po przesunięciu myszą nad podkreślonym kodem pojawi się komunikat błędu (rysunek 1-1).

```
const firstName = "Lizzo";
const nameLength = firstName.length();
```

```
(property) String.length: number
Returns the length of a String object.
This expression is not callable.
  Type 'Number' has no call signatures. ts(2349)
View Problem    No quick fixes available
```

Rysunek 1-1 *TypeScript informuje, że właściwość length ciągu tekstowego nie jest wywoływalna*

Możliwość uzyskania informacji o drobnych błędach podczas pisania kodu w edytorze jest znacznie wygodniejsza, niż czekanie, aż wykonanie określonego wiersza kodu spowoduje błąd. Próba uruchomienia tego kodu JavaScript doprowadzi do awarii!

Wolność poprzez ograniczenie

Za pomocą TypeScriptu można określić typy wartości, jakie można przypisać do parametrów i zmiennych. Niektórzy programiści uważają początkowo, że konieczność jawnego określania w kodzie sposobu działania pewnych elementów jest ograniczeniem.

Ale! Według mnie, tego typu „ograniczenie" jest w rzeczywistości pożądane! Nakładając na swój kod ograniczenia dotyczące sposobu jego użycia za pomocą języka TypeScript, mamy pewność, że zmiany w jednej części kodu nie doprowadzą do błędów w kodzie, który z niej korzysta.

Załóżmy, że uległa zmianie liczba parametrów wymaganych przez funkcję. W tym przypadku TypeScript przypomni o konieczności aktualizacji jej wywołania.

W poniższym przykładzie liczba parametrów wymaganych przez funkcję `sayMyName` zmienia się z dwóch na jeden, ale nadal jest ona wywoływana z dwoma ciągami tekstowymi. W tej sytuacji TypeScript zgłosi błąd:

```
// Wcześniej: sayMyName(firstName, lastNameName) { ...
```

1 Na użytek polskiego czytelnika komunikaty błędów zostały przetłumaczone przy pierwszym wystąpieniu, tak jak w tym przykładzie (w nawiasach i kursywą). Trzeba jednak mieć na uwadze, że zarówno w środowisku TypeScript Playground, jak i w środowiskach IDE komunikaty te będą dostępne tylko w języku angielskim (przyp. red. wydania polskiego).

```
function sayMyName(fullName) {
  console.log(`Zachowujesz się dziwnie, nie mówiąc do mnie ${fullName}`);
}

sayMyName("Beyoncé", "Knowles");
//                   ~~~~~~~~~
// Expected 1 argument, but got 2. (Oczekiwano 1 argumentu, ale otrzymano 2)
```

W języku JavaScript ten kod się wykona nie wywołując awarii, ale jego wynik będzie inny od oczekiwanego (nie będzie zawierał ciągu `"Knowles"`):

```
Zachowujesz się dziwnie, nie mówiąc do mnie Beyoncé
```

Wywoływanie funkcji z błędną liczbą argumentów jest dokładnie tego typu krótkowzrocznym ułatwieniem w korzystaniu z języka JavaScript, które TypeScript blokuje.

Dokładna dokumentacja

Przyjrzyjmy się wersji poznanej wcześniej funkcji `paintPainting`, tym razem napisanej w języku TypeScript. Chociaż nie omówiono jeszcze składni TypeScript służącej do dokumentowania typów, warto przeanalizować poniższy fragment, pokazujący, jak precyzyjnie można dokumentować kod za pomocą języka TypeScript:

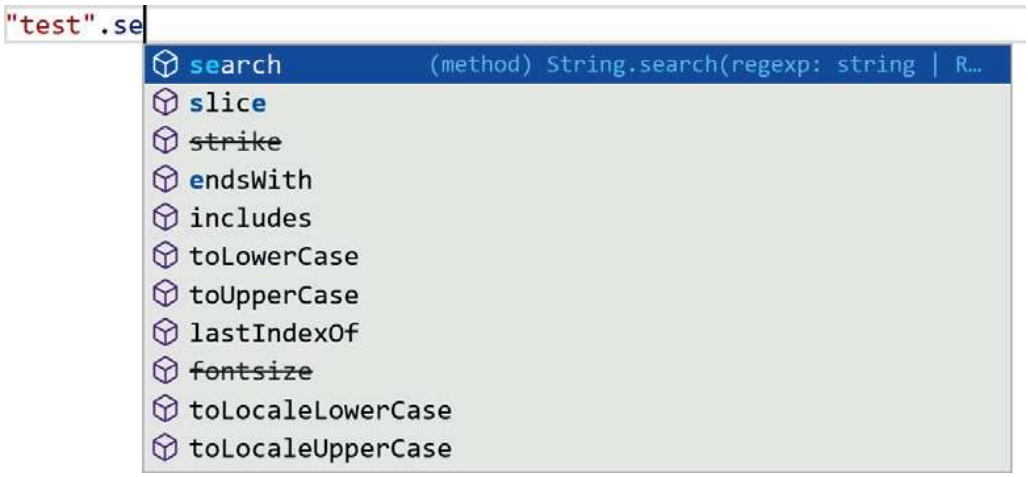
```
interface Painter {
  finish(): boolean;
  ownMaterials: Material[];
  paint(painting: string, materials: Material[]): boolean;
}

function paintPainting(painter: Painter, painting: string): boolean { /* ... */ }
```

Programista języka TypeScript czytający ten kod po raz pierwszy będzie wiedział, że `painter` ma co najmniej trzy właściwości, a dwie z nich są metodami. Dzięki składni opisującej „kształty” obiektów TypeScript oferuje doskonały, wymuszany system opisu obiektów.

Lepsze narzędzia deweloperskie

Dzięki narzędziu typings języka TypeScript edytory, takie jak VS Code, mogą uzyskać znacznie lepszy wgląd w kod. Dzięki temu mogą zaoferować programistom inteligentne sugestie podczas pisania kodu, które mogą się okazać niesłychanie przydatne podczas pracy.

Jeśli ktoś pisał już kod JavaScript w programie VS Code, zapewne zauważył, że dostępna jest funkcja „autouzupełniania” kodu wbudowanymi typami obiektów, np. ciągami tekstowymi. Np. po napisaniu nazwy właściwości, która jest ciągiem tekstowym, TypeScript może wyświetlić listę sugestii z nazwami wszystkich członków typu ciągów tekstowych (rysunek 1-2).



Rysunek 1-2 *Sugestie autouzupełniania oferowane przez TypeScript dla ciągów tekstowych w języku JavaScript*

Po dodaniu narzędzia do sprawdzania typów TypeScriptu można uzyskać takie przydatne sugestie nawet dla już napisanego kodu. Po wpisaniu `painter.` w funkcji `paintPainting` TypeScript wykorzysta wiedzę o tym, że parametr `painter` jest typu `Painter`, w którym zdefiniowane są określone właściwości (rysunek 1-3).

```
interface Painter {
  finish(): boolean;
  ownMaterials: Material[];
  paint(painting: string, materials: Material[]): boolean;
}

function paintPainting(painter: Painter, painting: string): boolean
  painter.
```

finish (method) Painter.finish(): boolean
ownMaterials
paint

Rysunek 1-3 *Sugestie autouzupełniania oferowane przez TypeScript dla zmiennej typu Painter*

Rewelacja! W rozdziale 12., „Używanie funkcji IDE” omówię mnóstwo innych przydatnych funkcji edytora.

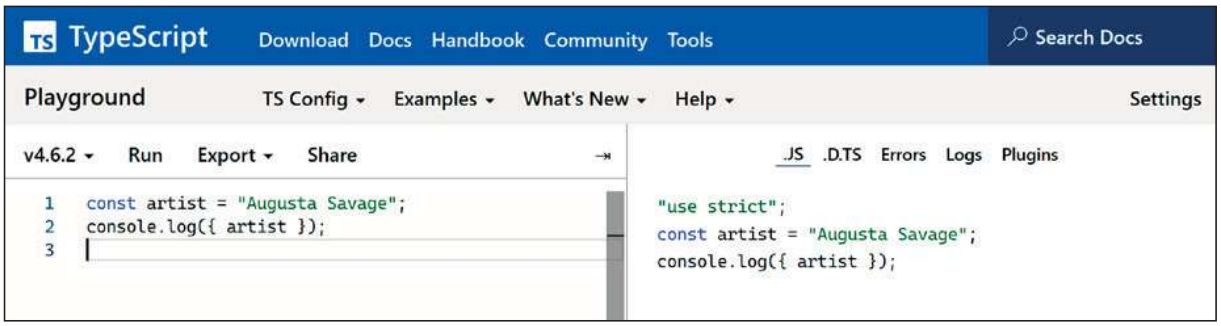
Kompilowanie składni

Kompilator języka TypeScript umożliwia pisanie kodu zgodnego ze składnią tego języka, sprawdza go pod kątem typów, a następnie generuje odpowiednik w języku JavaScript. Dla wygody kompilator może również przekształcić nowoczesną składnię języka JavaScript do starszych wersji standardu ECMAScript.

Po wpisaniu poniższego kodu TypeScript w środowisku Playground:

```
const artist = "Augusta Savage";
console.log({ artist });
```

z prawej strony ekranu pojawi się odpowiednik w języku JavaScript, wygenerowany przez kompilator (rysunek 1-4).



Rysunek 1-4 *Środowisko Playground kompiluje kod języka TypeScript do odpowiednika w języku JavaScript*

TypeScript Playground jest świetnym narzędziem, pokazującym przekształcanie kodu TypeScript w kod JavaScript.

W wielu projektach wykorzystujących JavaScript używa się specjalnych transpilatorów, takich jak Babel (*https://babeljs.io*), pomijając transpilację kodu źródłowego przez TypeScript do działającego kodu JavaScript. Pod adresem *https://learningtypescript.com/starters* znajduje się lista typowych projektów początkowych, na podstawie których można rozpocząć pracę.

Lokalna konfiguracja

Z języka TypeScript można korzystać na komputerze lokalnym po zainstalowaniu platformy Node.js. Aby zainstalować najnowszą wersję TypeScript i udostępnić ją globalnie, należy wykonać poniższe polecenie:

```
npm i -g typescript
```

Teraz będzie można uruchamiać TypeScript w wierszu poleceń, za pomocą polecenia `tsc` (TypeScript Compiler). Aby zweryfikować poprawność konfiguracji, należy wykonać je z opcją `--version`:

```
tsc --version
```

Na ekranie powinien się pojawić napis `Version X.Y.Z` – informujący o bieżącej zainstalowanej wersji języka TypeScript:

```
$ tsc --version
Version 4.7.2
```

Uruchamianie w lokalnym środowisku

Po zainstalowaniu języka TypeScript można utworzyć lokalny folder, w którym będzie uruchamiany TypeScript. W nowym folderze należy wykonać poniższe polecenie, aby utworzyć nowy plik konfiguracyjny *tsconfig.json*:

```
tsc --init
```

W pliku *tsconfig.json* zadeklarowane są ustawienia wykorzystywane przez TypeScript podczas analizy kodu. Większość opcji z tego pliku nie ma znaczenia dla ćwiczeń wykonywanych w tej książce (w językach programowania trzeba uwzględnić mnóstwo nietypowych przypadków brzegowych!). Są one opisane w rozdziale 13., „Opcje konfiguracyjne". Teraz istotna jest możliwość wykonania polecenia `tsc`, za pomocą którego TypeScript skompiluje wszystkie pliki znajdujące się we wspomnianym folderze, z uwzględnieniem opcji konfiguracyjnych z pliku *tsconfig.json*. Utwórz plik o nazwie *index.ts* i wpisz poniższy kod:

```
console.blub("Nic nie jest ważniejsze od śmiechu.");
```

Następnie wykonaj polecenie `tsc` podając nazwę pliku *index.ts*:

```
tsc index.ts
```

Na ekranie powinien zostać wyświetlony błąd podobny do poniższego:

```
index.ts:1:9 - error TS2339: Property 'blub' does not exist on type 'Console'.
                             (Właściwość 'blub' nie istnieje w typie 'Console')
1 console.blub("Nic nie jest ważniejsze od śmiechu.");
          ~~~~
Found 1 error.
```

Rzeczywiście, w obiekcie `console` nie zdefiniowano funkcji `blub`. Co ja sobie myślałem?

Zanim spróbujemy poprawić kod, aby zadowolić TypeScript, musimy zauważyć, że polecenie `tsc` utworzyło plik *index.js* zawierający kod `console.blub`.

Jest to ważna koncepcja: nawet jeśli kod zawiera *błąd typu*, jego *składnia* jest całkowicie poprawna. Kompilator języka TypeScript wygeneruje kod JavaScript na podstawie pliku wejściowego, niezależnie od występujących w nim błędów typu.

Popraw kod w pliku *index.ts* na `console.log` i ponownie wykonaj polecenie `tsc`. W terminalu nie powinno być żadnych błędów, a plik *index.js* powinien zawierać uaktualniony kod wyjściowy:

```
console.log("Nic nie jest ważniejsze od śmiechu.");
```

Zachęcam gorąco do bieżącego wypróbowania kodu przedstawionego w tej książce. Można go wpisać w środowisku Playground lub w edytorze obsługującym TypeScript,

czyli takim, który automatycznie uruchomi usługi języka TypeScript. Pod adresem *https://learningtypescript.com* dostępne są małe, samodzielne ćwiczenia, a także większe projekty, na podstawie których można przećwiczyć poznane koncepcje.

Funkcje edytora

Dzięki utworzeniu pliku *tsconfig.json* edytory, w których otworzymy określony folder, rozpoznają go jako projekt wykorzystujący TypeScript. Jeśli np. otworzymy ten folder w programie VS Code, ustawienia tego programu wykorzystywane podczas analizy kodu TypeScript uwzględnią konfigurację z pliku *tsconfig.json* znajdującego się w tym folderze.

Zachęcam do wpisania kodu z tego rozdziału w edytorze. Na ekranie powinny się pojawić listy z sugestiami uzupełniania wpisywanych nazw, szczególnie w przypadku elementów, takich jak `log` dla obiektu `console`.

To bardzo ekscytujące: używanie usług języka TypeScript, aby uzyskać pomoc w pisaniu kodu! Jesteś na dobrej drodze, aby stać się programistą języka TypeScript!

Program VS Code oferuje doskonałe wsparcie dla języka TypeScript, a ponadto został napisany w tym języku. Nie *musimy* w nim pisać kodu TypeScript – właściwie wszystkie nowoczesne edytory oferują doskonałe wsparcie dla języka TypeScript, zarówno wbudowane lub poprzez wtyczki – jednak zalecam chociaż wypróbowanie programu VS Code podczas czytania tej książki. Jeśli ktoś używa innego edytora, zalecam włączenie obsługi języka TypeScript. Funkcje edytora są dokładniej opisane w rozdziale 12., „Używanie funkcji IDE".

Czym nie jest TypeScript

Wiemy już, jak wspaniały jest TypeScript, a zatem czas na poznanie jego ograniczeń. Każde narzędzie doskonale nadaje się do pewnych zadań, ale zawodzi w przypadku innych.

Remedium na kiepski kod

TypeScript ułatwia tworzenie struktury kodu JavaScript, ale oprócz wymuszania bezpieczeństwa typów nie wymusza przestrzegania żadnych zasad dotyczących struktury. Bardzo dobrze!

TypeScript jest językiem, z którego powinien móc skorzystać każdy, a nie dogmatyczną platformą z docelową grupą odbiorców. Można pisać kod korzystając z dowolnych wzorców architektonicznych wykorzystywanych w języku JavaScript, a TypeScript go obsłuży.

Jeśli ktoś będzie twierdził, że TypeScript wymusza używanie klas, utrudnia pisanie dobrego kodu lub narzuca określony styl kodu, warto polecić mu tę książkę. TypeScript nie wymusza żadnego stylu kodu, stosowania klas czy funkcji, ani nie jest powiązany z żadną platformą do tworzenia aplikacji – taką jak Angular, React, itp.

Rozszerzenie języka JavaScriptu (przeważnie)

Cele projektowe języka TypeScript jasno informują, że powinien on:

- Być zgodny z bieżącymi i przyszłymi propozycjami rozwoju standardu ECMAScript
- Zachowywać działanie całego kodu JavaScript w trakcie wykonywania programu

TypeScript nie zmienia sposobu działania kodu JavaScript. Jego twórcy postarali się, aby nie implementować nowych funkcjonalności rozszerzających JavaScript lub będących z nim w konflikcie. Czuwa nad tym komitet techniczny TC39, który czuwa nad samym standardem ECMAScript.

Wiele lat temu dodano do języka TypeScript pewne funkcjonalności, mające odzwierciedlać typowe przypadki użycia w kodzie JavaScript. Większość z tych funkcjonalności obecnie nie ma większego zastosowania lub popadło w zapomnienie. Są one tylko pokrótce opisane w rozdziale 14., „Rozszerzenia składni". Zalecam, aby w większości przypadków trzymać się od nich z daleka.

W 2022 roku TC39 rozważa dodanie składni adnotacji typów do języka JavaScript. Zgodnie z ostatnimi propozycjami, adnotacje miałyby działać jak komentarze, które nie wpływają na działanie kodu w czasie wykonywania i są wykorzystywane tylko podczas prac programistycznych w systemach, takich jak TypeScript. Jednak upłynie wiele lat, zanim komentarze typów lub ich odpowiedniki zostaną dodane do JavaScriptu, a zatem nie są omówione w tej książce.

Wolniejszy niż JavaScript

Czasem w internecie spotyka się opinie programistów skarżących się, że w czasie wykonywania programu TypeScript jest wolniejszy niż JavaScript. Są to nieuzasadnione i mylące oskarżenia. TypeScript wprowadza w kodzie zmiany tylko wtedy, gdy zachodzi konieczność kompilacji kodu do wcześniejszych wersji języka JavaScript w celu obsługi starszych środowisk uruchomieniowych, takich jak Explorer 11. W wielu platformach produkcyjnych wcale nie używa się kompilatora TypeScript, korzystając zamiast niego z innego narzędzia do transpilacji (etapu kompilacji, polegającego na przekształceniu kodu źródłowego z jednego języka programowania do innego), a TypeScript jest wykorzystywany tylko do sprawdzania typów.

TypeScript wydłuża jednak czas budowania kodu. Kod języka TypeScript musi być skompilowany do kodu JavaScript, zanim będzie mógł zostać wykonany w większości środowisk, takich jak przeglądarki i Node.js. Większość procesów budowania konfiguruje się w taki sposób, że wspomniany wpływ na wydajność jest znikomy, a wolniejsze funkcje języka TypeScript, takie jak analizowanie kodu pod kątem możliwych pomyłek, są wykonywane osobno, nie wpływając na generowanie wykonywalnych plików z kodem aplikacji.

Nawet w projektach, które pozornie umożliwiają bezpośrednie wykonywanie kodu TypeScript, np. ts-node i Deno, za kulisami kod TypeScript jest przed wykonaniem przekształcany w JavaScript.

Zakończony rozwój

Sieć nieustannie się rozwija i to samo dotyczy języka TypeScript. W języku tym stale poprawia się błędy i dodaje nowe funkcjonalności, aby nadążyć ze zmieniającymi się wymaganiami społeczności. Podstawowe założenia języka TypeScript opisane w tej książce się nie zmienią, ale komunikaty błędów, bardziej wyrafinowane funkcje i możliwości integracji z edytorami będą z czasem ulepszane.

Chociaż pierwsze wydanie tej książki zostało opublikowane w czasie, gdy dostępny był TypeScript w wersji 4.7.2, w czasie czytania tej książki najprawdopodobniej dostępna będzie nowsza wersja. Niektóre komunikaty błędów języka TypeScript zaprezentowane w tej książce mogą już być nieaktualne!

Podsumowanie

W tym rozdziale omówiono niektóre z głównych słabości języka JavaScript, które rozwiązuje TypeScript, a także opisano, jak zacząć korzystać z języka TypeScript. Oto zagadnienia poruszone w tym rozdziale:

- Krótka historia języka JavaScript
- Słabe strony języka JavaScript: kosztowna wolność, luźna dokumentacja i słabsze narzędzia deweloperskie
- Czym jest TypeScript: językiem programowania, narzędziem do sprawdzania typów, kompilatorem i usługą językową
- Zalety języka TypeScript: wolność poprzez ograniczenia, dokładna dokumentacja i lepsze narzędzia deweloperskie
- Pierwsze kroki w pisaniu kodu TypeScript w środowisku Playground oraz na komputerze
- Czym nie jest TypeScript: remedium na kiepski kod, rozszerzeniem kodu JavaScript (najczęściej), językiem wolniejszym niż JavaScript, oraz językiem o zakończonym rozwoju

Po przeczytaniu tego rozdziału warto przećwiczyć omówione zagadnienia korzystając z zasobów dostępnych na stronie *https://learningtypescript.com/from-javascript-totypescript*.

Rozdział 2

System typów

Moc języka JavaScript
Wynika z elastyczności
Uważaj na nią!

W rozdziale 1., „Od języka JavaScript do TypeScript” wspomniano, że w języku TypeScript istnieje narzędzie do „sprawdzania typów”, które analizuje kod, określa, jak powinien działać i informuje, gdzie mogło dojść do pomyłki. Jak jednak działa to narzędzie?

Czym jest typ?

„Typ” opisuje *kształt*, jaki może mieć wartość zdefiniowana w języku JavaScript. Pisząc „kształt”, mam na myśli właściwości i metody zdefiniowane w tej wartości, a także wynik, jaki dla tej wartości zwróci wbudowany operator `typeof`.

Przykładowo, po utworzeniu zmiennej o początkowej wartości `"Aretha"`:

```
let singer = "Aretha";
```

TypeScript ustali, że zmienna `singer` jest *typu* string.

Podstawowe typy w języku TypeScript odpowiadają siedmiu podstawowym typom w języku JavaScript:

- `null`
- `undefined`
- `boolean` – `true` lub `false`
- `string` – `""`, `"Cześć!"`, `"abc123"`, …
- `number` – `0`, `2.1`, `-4`, …
- `bigint` – `0n`, `2n`, `-4n`, …
- `symbol` – `Symbol()`, `Symbol("hi")`, …

Analizując następujące wartości TypeScript uzna, że mają one typ należący do siedmiu podstawowych typów prostych:

- `null; // null`
- `undefined; // undefined`
- `true; // boolean`
- `"Louise"; // string`
- `1337; // number`
- `1337n; // bigint`
- `Symbol("Franklin"); // symbol`

Jeśli komuś zdarzy się zapomnieć nazwy typu prostego, można wpisać `let` *`zmienna`* z wartością w środowisku TypeScript Playground (*https://typescriptlang.org/play*) lub w IDE, a następnie należy przesunąć kursorem nad nazwą zmiennej. W oknie, które się pojawi, widoczna będzie nazwa typu prostego, co widać na poniższym zrzucie ekranu, pokazującym, że zmienna jest typu `string` (rysunek 2-1).

```
2
3
4          let singer: string
5   let singer = "Ella Fitzgerald";
6
```

Rysunek 2-1 *Po umieszczeniu kursora myszy nad nazwą zmiennej TypeScript pokazuje, że jest ona typu string*

TypeScript jest również wystarczająco sprytny, aby ustalić typ zmiennej, której wartość początkowa jest obliczana. W poniższym przykładzie TypeScript wie, że wyrażenie trójargumentowe zawsze będzie typu string, a zatem zmienna `bestSong` jest typu `string`:

```
let bestSong = Math.random() > 0.5
  ? "Chain of Fools"
  : "Respect";
```

Gdy w TypeScript Playground (*https://typescriptlang.org/play*) lub w edytorze IDE przesuniemy kursorem nad zmienną `bestSong`, powinno się pojawić okno z informacją, że zmienna `bestSong` jest typu `string` (rysunek 2-2).

```
    let bestSong: string
let bestSong = Math.random() > 0.5
  ? "Chain of Fools"
  : "Respect";
```

Rysunek 2-2 *TypeScript informuje, że zmienna jest literałem tekstowym, wyznaczonym na podstawie wyrażenia trójargumentowego*

Przypomnijmy sobie różnice między obiektami a typami prostymi w języku JavaScript: klasy, takie jak `Boolean` i `Number` opakowują swoje odpowiedniki w postaci typów prostych. Zgodnie z najlepszą praktyką w języku TypeScript należy się odwoływać się do typów prostych, czyli w tym przypadku do typów `boolean` i `number`.

Systemy typów

System typów jest zbiorem zasad, na podstawie których język programowania rozumie, jakiego typu są konstrukcje w programie.

System typów języka TypeScript działa następująco:

- Odczytuje kod i analizuje wszystkie istniejące typy i wartości
- Sprawdza, jakiego typu jest początkowa deklaracja każdej wartości
- Sprawdza wszystkie sposoby użycia każdej wartości w dalszej części kodu
- Ostrzega użytkownika, jeśli użycie wartości nie jest zgodne z jej typem

Przeanalizujmy szczegółowo ten proces.

W poniższym fragmencie TypeScript zgłasza błąd o właściwości błędnie użytej jako funkcja:

```
let firstName = "Whitney";
firstName.length();
//        ~~~~~~
//  This expression is not callable.
//    Type 'Number' has no call signatures
//    (Typ 'Number' nie ma sygnatur wywołania)
```

Oto przebieg procesu generowania tego błędu przez TypeScript:

1. Odczytanie kodu i ustalenie, że zawiera zmienną o nazwie `firstName`
2. Ustalenie, że zmienna `firstName` jest typu `string`, ponieważ jej początkowa wartość, `"Whitney"`, jest typu `string`
3. Ustalenie, że kod próbuje uzyskać dostęp do właściwości `.length` zmiennej `firstName` i wywołać ją jak funkcję
4. Zgłoszenie informacji, że właściwość `.length` ciągu tekstowego jest liczbą, a nie funkcją *(nie można jej wywołać jak funkcji)*

Zrozumienie systemu typów TypeScriptu jest ważną umiejętnością potrzebną do zrozumienia kodu w języku TypeScript. Fragmenty kodu w tym rozdziale i w pozostałej części książki będą zawierać coraz bardziej złożone typy, które TypeScript będzie mógł wywnioskować na podstawie kodu.

Rodzaje błędów

Podczas pisania kodu TypeScript, najczęściej mamy do czynienia z dwoma następującymi typami „błędów”:

Składniowe
: Blokują przekształcenie kodu TypeScript w kod JavaScript.

Typu
: Narzędzie do sprawdzania typów wykryło pewne niedopasowanie.

Różnice między nimi są ważne.

Błędy składniowe

Błędy składniowe występują wtedy, gdy TypeScript wykryje błędną, niezrozumiałą składnię. Blokuje to generowanie wynikowego kodu JavaScript na podstawie kodu TypeScript. W zależności od narzędzi i ustawień używanych do przekształcania kodu TypeScript w kod JavaScript, czasem można uzyskać pewien kod JavaScript (np. w przypadku domyślnych ustawień narzędzia `tsc`). Jednak nie będzie on zgodny z oczekiwaniami.

W poniższym kodzie TypeScript występuje błąd składniowy dotyczący nieoczekiwanego słowa `let`:

```
let
let wat;
//  ~~~
// Error: ',' expected. (Błąd: Oczekiwano ',')
```

W zależności od wersji kompilatora języka TypeScript, skompilowany wynik w języku JavaScript może wyglądać podobnie do poniższego:

```
let
let, wat;
```

Chociaż TypeScript dołoży wszelkich starań, aby zwrócić kod JavaScript mimo błędów składniowych, uzyskany kod prawdopodobnie nie spełni naszych wymagań. Błędy składniowe należy naprawić, zanim podejmie się próbę wygenerowania kodu JavaScript.

Błędy typów

Błędy typów występują, gdy składnia jest poprawna, ale narzędzie do sprawdzania typów TypeScript wykryje błąd dotyczący typów stosowanych w programie. Nie blokują one konwersji składni TypeScript w kod JavaScript. Jednak zwykle oznaczają, że po uruchomieniu aplikacja ulegnie awarii lub będzie się zachowywać w nieoczekiwany sposób.