O'REILLY®

Helion

GitHub Copilot
Wprowadzenie
Efektywne programowanie z asystentem AI

Brent Laster

Tytuł oryginału: Learning GitHub Copilot: Multiplying Your Productivity
With an AI Pair Programmer

Tłumaczenie: Ksawery Sosnowski

ISBN: 978-83-289-3474-0

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

Helion S.A.
ul. Kościuszki 1c, 44-100 Gliwice
tel. 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *helion.pl* (księgarnia internetowa, katalog książek)

Printed in Poland.

Spis treści

ROZDZIAŁ 2.

Programowanie z Copilotem

Teraz gdy poznałeś już podstawy działania Copilota, możemy przejść do kluczowego aspektu: jak z niego korzystać. Copilot ma funkcjonować jako asystent AI, dlatego musisz nauczyć się z nim pracować i komunikować. W praktyce sprowadza się to do nauki obsługi integracji w środowisku programistycznym. Opanowanie tych umiejętności, omówionych w tym i następnym rozdziale, przygotuje cię na pozostałą część książki. Odpowiem też na następujące pytania:

- Jak zadawać pytania asystentowi i udzielać mu wskazówek dotyczących tego, czego potrzebujesz?
- Jak kontynuować rozmowę na podstawie otrzymanych odpowiedzi lub wybierać spośród wielu sugestii?
- Które sposoby komunikacji są najlepsze w przypadku szybkich sugestii lub dłuższych wyjaśnień, gdy są potrzebne?
- Jak najlepiej wykorzystać AI do pomocy w zadaniach programistycznych?

IDE = Visual Studio Code

GitHub Copilot można zainstalować i wykorzystywać w wielu środowiskach programistycznych, w tym VS Code, IDE firmy JetBrains, Eclipse i NeoVim. Nie jesteśmy w stanie omówić w tej książce wszystkich dostępnych narzędzi, dlatego skupimy się na przykładach z VS Code i GitHub Codespaces. W razie potrzeby zapoznaj się z dokumentacją swojego środowiska programistycznego, aby poznać ewentualne różnice w pracy z Copilotem.

W ramach środowiska IDE Copilot oferuje dwa główne tryby interakcji. Pierwszy to integracja bezpośrednio w edytorze oraz poprzez menu kontekstowe i powiązane elementy. Dla uproszczenia będę nazywał to **trybem inline** (podpowiedzi w kodzie, ang. *inline mode*), ponieważ najczęściej jest używany bezpośrednio z poziomu edytora lub powiązanych list menu. Niniejszy rozdział skupia się właśnie na tym trybie.

Drugi tryb interakcji to osobny **interfejs konwersacyjny** (ang. *chat interface*), dostarczany wraz z Copilotem. To podejście zapewnia rozszerzone możliwości interakcji i uzyskiwania informacji zwrotnych. Stanowi również bramę do bardziej zaawansowanych funkcji Copilota, w tym **trybu**

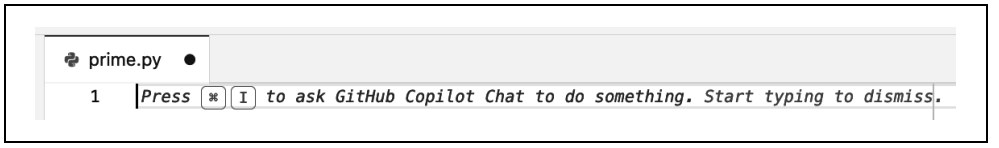
agentowego (ang. *agent mode*) i **trybu zaawansowanej edycji** (ang. *advanced editing mode*). Gdy już omówimy w tym rozdziale podstawowe przypadki użycia trybu inline, w rozdziałach 3. i 4. zajmiemy się funkcjami konwersacyjnymi.

Większość użytkowników prawdopodobnie rozpocznie pracę z Copilotem za pośrednictwem edytora w swoim środowisku programistycznym. W trakcie programowania zainstalowane rozszerzenie Copilota zbiera kontekst i integruje się z komponentami zdalnymi, aby dostarczać sugestie. Omówiliśmy ogólnie ten mechanizm w rozdziale 1. Teraz przyjrzyjmy się mu bliżej.

Copilot wykorzystuje różne kombinacje klawiszy jako skróty do wywoływania rozmaitych funkcji. Te kombinacje zazwyczaj składają się z klawisza modyfikującego (*Command*, *Ctrl* lub *Option*) oraz litery, cyfry lub symbolu. Klawisz modyfikujący może się różnić w zależności od używanego systemu operacyjnego i typu klawiatury. Zamiast wymieniać każdą kombinację osobno, w tej książce będę używać zapisu *Meta+<znak>*, na przykład *Meta+I* zamiast *Option+I*. Pełną listę skrótów klawiszowych dla swojej platformy znajdziesz w dokumentacji Copilota (*https://oreil.ly/aAZ8S*).

Sugestie w trybie inline

Załóżmy, że chcesz stworzyć w Pythonie plik do pracy z liczbami pierwszymi. Możesz zacząć od utworzenia pliku o nazwie *prime.py*. Pierwszą rzeczą, którą prawdopodobnie zauważysz, będzie możliwość interakcji z *czatem Copilota* za pomocą kombinacji klawiszy *Meta+I*. To przypomnienie pojawi się za każdym razem, gdy będziesz edytować plik, jeśli Copilot jest aktywny i funkcja czatu jest włączona. Widać to na rysunku 2.1.



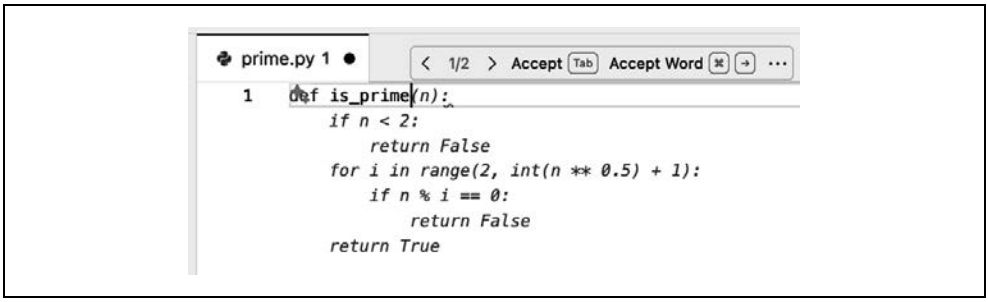
Rysunek 2.1. Opcja uruchamiania czatu Copilota w oknie edytora

Omówimy ten interfejs szczegółowo w rozdziale 3., po przedstawieniu bardziej podstawowych interakcji, ponieważ wtedy będzie to miało większy sens. Chociaż domyślnie ta opcja jest dostępna, możesz ją na razie zignorować w edytorze i zacząć pisać.

Aby zaprezentować przykład, stwórzmy w pliku *prime.py* funkcję sprawdzającą, czy dana liczba jest pierwsza. Zacznij wpisywać:

```
def is_prime
```

W tym momencie Copilot zacznie sugerować możliwe uzupełnienia. Może zaproponować opcję podobną do tej przedstawionej na rysunku 2.2. Proponowany przez Copilota tekst jest wyświetlany jako **ukryty tekst** (ang. *ghost text*). Oznacza to, że dopóki nie zostanie zaakceptowany, sugerowany tekst będzie wyświetlany kursywą z użyciem jaśniejszej czcionki, a nie zwykłą, ciemną czcionką.



Rysunek 2.2. Pierwsza sugestia

W twoim przypadku sugestie mogą wyglądać inaczej

Podczas czytania tego rozdziału oraz innych fragmentów książki zawierających sugestie lub wyniki generowane przez Copilota pamiętaj, że mamy do czynienia z generatywną sztuczną inteligencją. Oznacza to, że sugestie mogą się znacznie różnić nawet w przypadku, gdy kontekst jest ten sam.

Sugestia Copilota to jedna z możliwych opcji rozbudowy twojej funkcji. Na rysunku 2.2 widoczny jest szary pasek nad sugerowanym tekstem. Pasek ten pojawia się, gdy Copilot proponuje tekst w wierszu kodu. Zawiera on przyciski umożliwiające zaakceptowanie sugestii w całości lub części. Inne elementy sterujące pozwalają na przeglądanie wielu propozycji, jeśli Copilot wygenerował kilka wariantów.

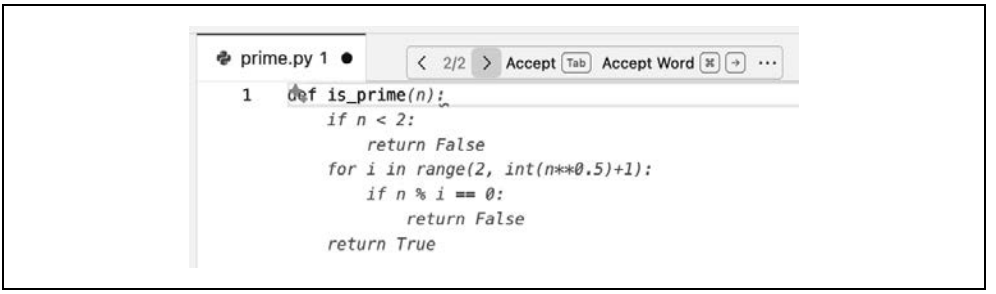
Twoje środowisko programistyczne może wyglądać inaczej

W całej książce korzystamy ze środowisk programistycznych (IDE) w stylu VS Code. Jeśli używasz innego IDE (na przykład firmy JetBrains), interfejs i elementy sterujące mogą się różnić, choć powinny być dostępne podobne funkcje.

Tekst na pasku przypomina również o odpowiednich skrótach klawiszowych i opcjach. Dostępne opcje obejmują zaakceptowanie całej sugestii (za pomocą klawisza *Tab*), akceptowanie słowo po słowie (za pomocą kombinacji klawiszy pokazanej na pasku i w dokumentacji) oraz opcje dostępne w menu dodatkowych czynności, oznaczonym trzema kropkami (więcej o tym później).

W lewej części paska widnieje *< 1/2 >*. Ten tekst oznacza, że Copilot wygenerował dwie możliwe sugestie, a ty obecnie oglądasz pierwszą możliwość. Aby zobaczyć inne opcje i przełączać się między potencjalnymi wyborami, kliknij symbole > i < na pasku lub użyj odpowiednich klawiszy na klawiaturze. Na przykład, jeśli chcesz zobaczyć alternatywną sugestię, możesz kliknąć symbol > (patrz rysunek 2.3).

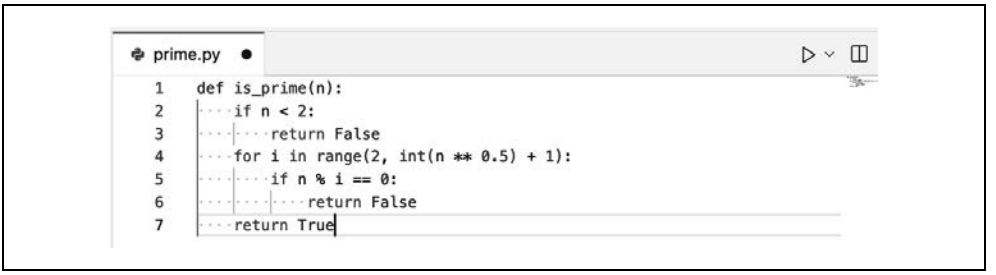
Aby zaakceptować jedną z sugestii Copilota, możesz przełączać się między opcjami, a następnie nacisnąć klawisz *Tab*, by wybrać tę, która ci odpowiada. Jeśli żadna z propozycji ci nie pasuje, możesz po prostu kontynuować pisanie.



Rysunek 2.3. Druga sugestia

W naszym przypadku różnice między obydwiema sugestiami są subtelne i dotyczą głównie odstępów. Mimo to Copilot przedstawia je jako dwie odrębne propozycje. Niezależnie od tego, którą opcję wybierzesz, otrzymasz coś, co wygląda na poprawną, wygenerowaną automatycznie treść funkcji.

Jeśli zdecydujesz się na pierwszą sugestię, jej pełna treść zostanie po naciśnięciu klawisza *Tab* wstawiona do twojego pliku. Ten tekst przestanie być „ukrywany" i będzie wyglądał tak samo, jak gdybyś wpisał go własnoręcznie (rysunek 2.4).



Rysunek 2.4. Wybrano pierwszą sugestię

Po kilkukrotnym użyciu interfejs staje się bardziej przewidywalny i pomocny. Od tego momentu typowy cykl pracy podczas pisania kodu wygląda następująco:

1. Zaakceptuj lub odrzuć sugestię Copilota.
2. W razie potrzeby — albo jeśli tak wolisz — wprowadź dodatkowy kod ręcznie.
3. Jeśli to konieczne, zrób krótką przerwę, aby dać Copilotowi czas na wygenerowanie opcji.
4. Copilot przedstawia sugestie.
5. Powtarzaj kroki od 1. do 4. aż do ukończenia kodu.

Kompletność i *trafność* sugestii Copilota mogą się znacznie różnić. W zależności od dostępnego kontekstu może on oferować propozycje uzupełnień od pojedynczego komentarza przez jeden wiersz aż po wiele wierszy tworzących kompletne rozwiązanie. Podczas pisania możesz otrzymać sugestię pełnej definicji funkcji, a innym razem tylko wiersz lub dwa. Schemat działania opisany w powyższych punktach jest taki sam niezależnie od tego, czy pierwsza sugestia jest kompletną propozycją, czy też musisz przejść przez kilka rund sugestii wiersz po wierszu.

Chwila przerwy na otrzymanie sugestii

Być może zwróciłeś uwagę na krok 3. z poprzedniej listy, który wspomina o zrobieniu przerwy (w razie potrzeby), aby dać Copilotowi czas na wygenerowanie opcji. Podczas opracowywania sugestii zachodzi wiele procesów i odbywa się intensywna komunikacja. Jak omówiono w rozdziale 1., aby Copilot mógł wygenerować sugestię kodu, wykonuje w tle wiele operacji. Zbiera kontekst, syntetyzuje go, przesyła przez sieć, przekazuje do modelu językowego, sprawdza pod kątem potencjalnych luk w zabezpieczeniach, ewentualnie porównuje z publicznie dostępnym kodem, a następnie udostępnia opcje.

Dzieje się to niezwykle szybko. Jednak niekiedy Copilot po naciśnięciu klawisza *Enter* może potrzebować chwili na zwrócenie wyniku. Z tego powodu, jeśli masz zwyczaj naciskania *Enter*, a zaraz potem *Tab*, aby zaakceptować kolejne sugestie, możesz czasem wyprzedzić Copilota. W takiej sytuacji będziesz musiał się wstrzymać przed próbą zaakceptowania następnego wiersza. Warto też pamiętać, że Copilot może czasem jako część wyniku wygenerować pusty wiersz.

Warto zwrócić uwagę na kilka szczegółów dotyczących tej prostej interakcji z Copilotem:

- Copilot był w stanie zasugerować definicje funkcji odpowiednie do kontekstu.
- Proponowane opcje różniły się tylko nieznacznie, w sposób **nieistotny technicznie**, ale były prezentowane inaczej.
- Skąd Copilot wziął konkretny kontekst dla swoich sugestii? Wykorzystał nazwę pliku i nazwę, którą wpisaliśmy dla funkcji.
- Gdybyś powtórzył to samo ćwiczenie, nie miałbyś gwarancji, że otrzymasz takie same sugestie, ani nawet więcej niż jedną. Generatywna sztuczna inteligencja nie zawsze zwróci te same wyniki, nawet jeśli kontekst jest identyczny.

Odnosząc się do ostatniego punktu: chociaż sugestie wyświetlane przez Copilota często będą dla ciebie wystarczające, może się zdarzyć, że będziesz chciał zobaczyć dodatkowe opcje. Wiedz zatem, że Copilot ma wbudowany mechanizm, który pozwala poprosić o dodatkowe **alternatywne sugestie**. Omówimy to w następnej kolejności.

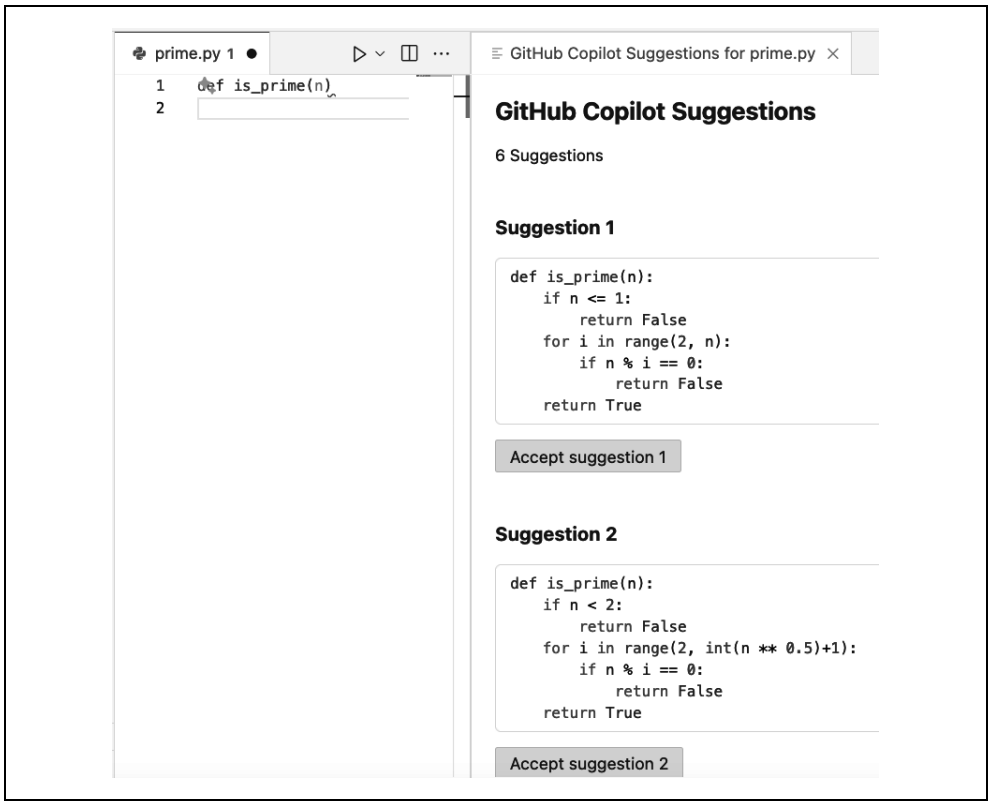
Uzyskiwanie dodatkowych sugestii

Jeśli początkowe sugestie Copilota nie wydają się wystarczające, możesz poprosić go o wygenerowanie kolejnych. Aby uzyskać bardziej rozbudowaną listę alternatywnych propozycji, użyj kombinacji klawiszy *Meta+Enter*.

Wady i zalety korzystania z funkcji dodatkowych sugestii

Chociaż proszenie Copilota o dodatkowe sugestie może wygenerować więcej opcji uzupełniania, wiąże się to z kompromisem w postaci wydłużenia czasu tworzenia i prezentowania uzupełnień w porównaniu z natychmiastowymi opcjami wyświetlanymi w trybie inline. Ponadto alternatywny kod może składać się z częściowych implementacji lub zestawów komentarzy, co czyni go mniej użytecznym.

Najlepszym momentem na skorzystanie z tej funkcji jest czas przed zaakceptowaniem jednej z sugestii wyświetlanych w trybie inline. Wróćmy do chwili, w której wpisywaliśmy nazwę funkcji, ale jeszcze nie zaakceptowaliśmy żadnej sugestii. W tym momencie możesz nacisnąć *Meta+Enter*, a Copilot wygeneruje osobny panel z alternatywnymi uzupełnieniami, jak pokazano na rysunku 2.5.



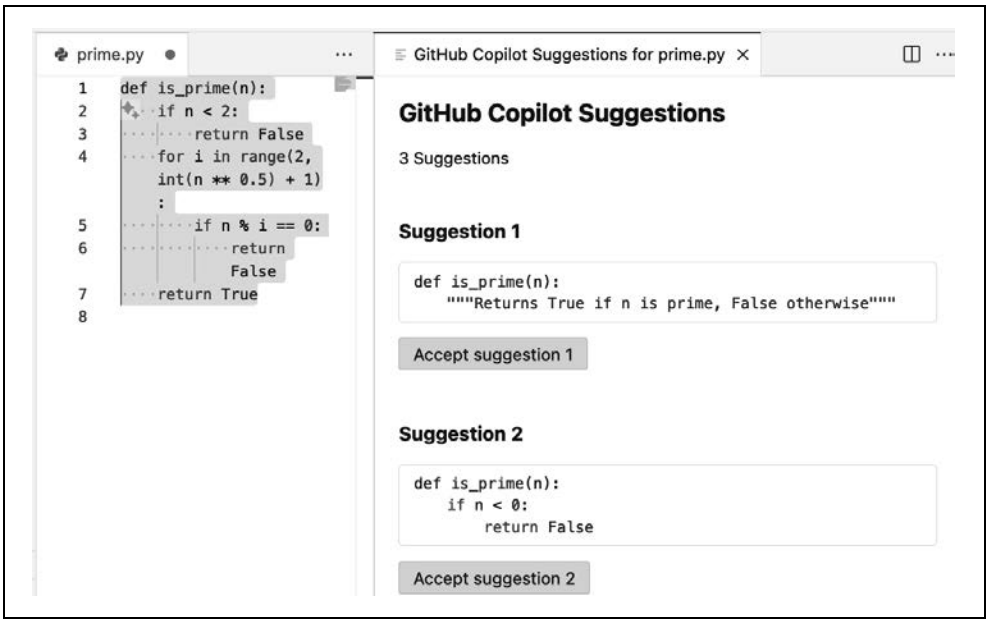
Rysunek 2.5. Alternatywne sugestie przed uzupełnieniem kodu

Copilot zaproponował sześć alternatywnych opcji, z których prawdopodobnie wszystkie są możliwe do zastosowania, przy czym niektóre implementacje różnią się tylko nieznacznie.

Zakładka GitHub Copilot Suggestions działa w trybie tylko do odczytu. Możesz przewijać dostępne sugestie, aby zobaczyć alternatywne uzupełnienia. Jeśli znajdziesz taką, która ci odpowiada, możesz wstawić ją do swojego pliku przez kliknięcie pod nią przycisku *Accept suggestion #* (gdzie # oznacza numer sugestii).

Jeśli chcesz skorzystać z funkcji alternatywnych sugestii dla napisanego już przez siebie kodu, wyniki mogą być jeszcze bardziej zróżnicowane. W tym przypadku koniecznie się upewnij, że zaznaczyłeś kod, dla którego Copilot ma wygenerować sugestie. W przeciwnym razie możesz nie otrzymać żadnych wyników.

Prosząc Copilota o alternatywne sugestie dla istniejącego kodu, możemy zauważyć różnice w jakości i użyteczności proponowanych rozwiązań. Na rysunku 2.6 pokazany jest zestaw takich sugestii wygenerowanych na podstawie istniejącego kodu.



Rysunek 2.6. Prośba o alternatywne propozycje dla istniejącego kodu

Zwróć uwagę, że w zakładce sugestii znajdują się tylko trzy propozycje, z których żadna nie jest kompletną implementacją. Dlatego jeśli masz tylko minimalny kontekst, najlepszym rozwiązaniem jest poprosić o alternatywne sugestie jeszcze przed dodaniem kodu.

Jak wiesz z rozdziału 1., Copilot opiera swoje sugestie nie tylko na faktycznym kodzie, ale także na powiązanych elementach, takich jak komentarze. Chociaż zazwyczaj myślimy o dodawaniu komentarzy tylko w celu wyjaśnienia kodu, możemy również wykorzystać je do sterowania zachowaniem Copilota na inne sposoby.

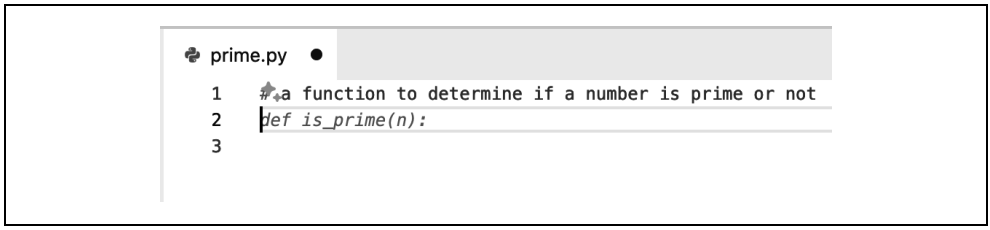
Korzystanie z komentarzy

Komentarze mogą odgrywać istotną rolę w kontekście działania Copilota. Wyobraź sobie, że wprowadzasz pojedynczy komentarz jako jedyny wiersz w nowym pliku:

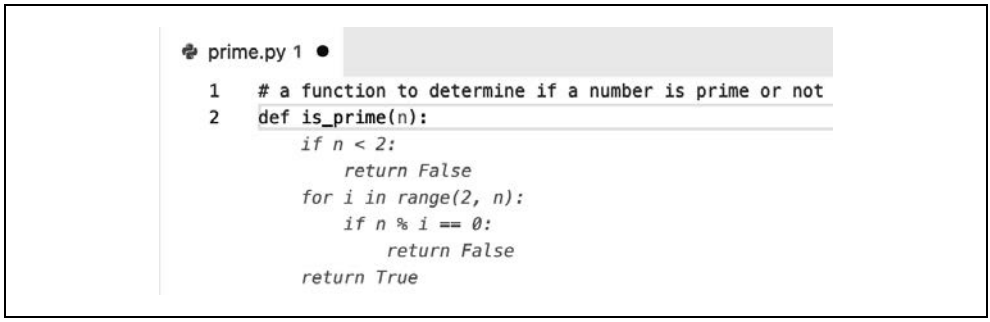
```
# a function to determine if a number is prime or not
```

Copilot może zasugerować uzupełnienie, jak pokazano na rysunku 2.7.

Po zaakceptowaniu go zostanie zasugerowany kod dla reszty funkcji (patrz rysunek 2.8).



Rysunek 2.7. Sugestia wygenerowana na podstawie komentarza



Rysunek 2.8. Uzupełnienie wygenerowane na podstawie komentarza

Poza pasywnym kontekstem, który Copilot może uzyskać ze standardowych komentarzy, możesz również aktywnie wykorzystywać komentarze, aby powiedzieć Copilotowi, co ma zrobić. Różnica polega na tym, że zamiast pisać komentarze jako kontekst, wydajesz instrukcje Copilotowi. Wiąże się to z formułowaniem komentarza bardziej jako dyrektywy.

Komentarze dyrektywne

Możesz przekazywać Copilotowi instrukcje za pomocą komentarzy. Załóżmy, że zamiast ręcznego wpisywania nagłówka funkcji, jak to miało miejsce w poprzednim rozdziale:

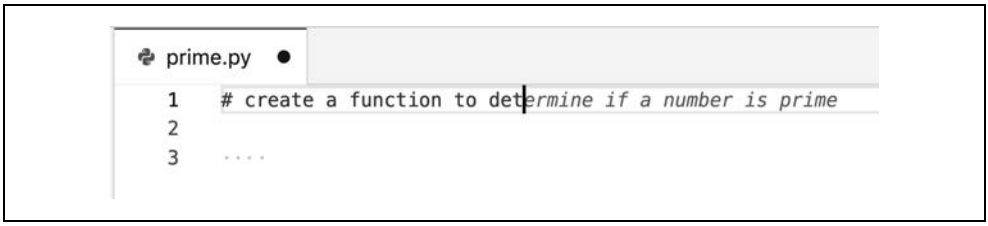
```
def is_prime
```

wpisujesz komentarz opisujący, co chcesz osiągnąć. Przykład w naszej funkcji sprawdzającej liczby pierwsze mógłby wyglądać tak:

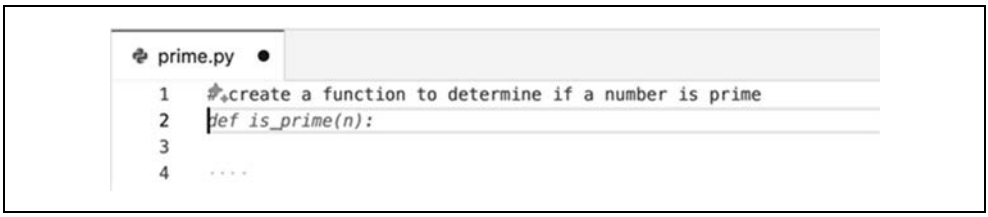
```
# create a function to determine if a number is prime
```

W zależności od dostępnego kontekstu Copilot może zasugerować odpowiednie uzupełnienie twojego komentarza, jak pokazano na rysunku 2.9. Proponowane uzupełnienie może być przydatne lub nie.

Ten jeden wiersz wystarczy, aby rozpocząć generowanie sugerowanego kodu funkcji. Przykład znajdziesz na rysunku 2.10.

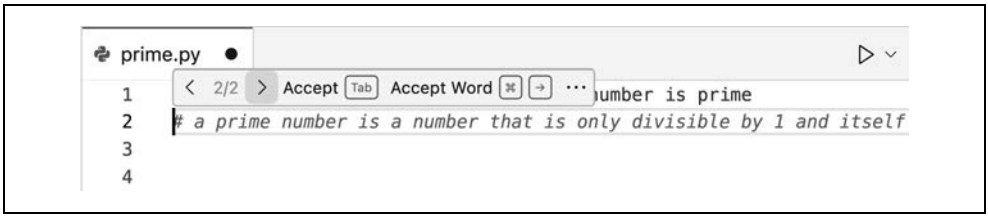


Rysunek 2.9. Autouzupełnianie komentarza



Rysunek 2.10. Wygenerowany pierwszy wiersz funkcji

Proponowana nazwa funkcji jest taka sama jak wcześniej, ale różnica pojawia się w opcjach, które oferuje Copilot. Zamiast sugerować tylko jeden wiersz, Copilot proponuje kilka możliwych uzupełnień. Jedno z nich to nagłówek funkcji, a drugie to kontynuacja komentarza. Te sugestie można przeglądać za pomocą wyskakującego paska lub skrótów klawiszowych. Szczegóły przedstawiono na rysunku 2.11.

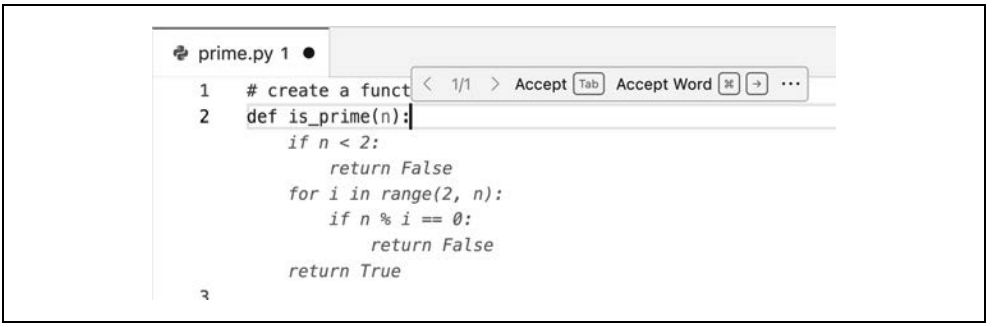


Rysunek 2.11. Druga opcja: kontynuacja komentarza

U ciebie dostępne opcje mogą wyglądać inaczej

Przypominamy, że podczas pracy z generatywną sztuczną inteligencją sugestie otrzymywane przy korzystaniu z Copilota w praktyce mogą różnić się od tych przedstawionych w książce, nawet jeśli używasz tego samego przykładowego kodu.

W tym przypadku Copilot nie ma wystarczającego kontekstu, aby wiedzieć, czy chcesz wygenerować rozszerzony komentarz czy rozpocząć tworzenie kodu funkcji. Dlatego przedstawia obie opcje. Po zaakceptowaniu pierwszej opcji Copilot otrzymuje wystarczający kontekst i proponuje definicję funkcji (rysunek 2.12).

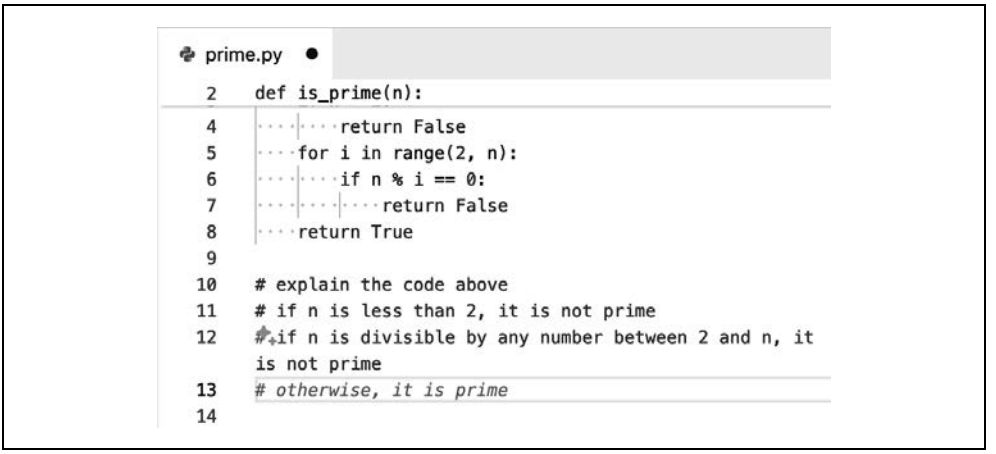


Rysunek 2.12. Pełna sugestia funkcji

Innym przykładem jest możliwość poproszenia Copilota o wyjaśnienie istniejącego kodu przez dodanie komentarza bezpośrednio w trybie inline:

```
# explain the code above
```

W odniesieniu do omawianego przykładu Copilot generuje krótkie, opatrzone komentarzami wiersze wynikowe, które wyjaśniają działanie kodu. Na rysunku 2.13 przedstawiono wygenerowane komentarze. Copilot utworzył każdy z trzech wierszy komentarzy (11. – 13.) po dyrektywie `explain`.



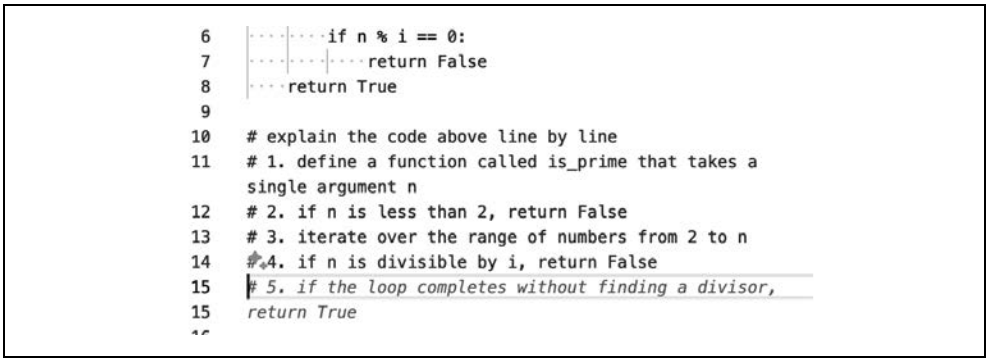
Rysunek 2.13. Rezultat polecenia wyjaśnienia kodu za pomocą komentarzy

Możesz również dodać komentarz, aby uzyskać bardziej szczegółową odpowiedź:

```
# explain the code above line-by-line
```

Efekt zastosowania tego komentarza przedstawiono na rysunku 2.14.

Warto zauważyć, że jeśli używasz komentarzy do *generowania kodu przez Copilota*, to Copilot powinien tworzyć kod na podstawie danego komentarza. W przypadku innych dyrektyw niezwiązanych z programowaniem (takich jak `explain`) wyniki będą zwracane w formie *dodatkowych komentarzy*.



Rysunek 2.14. Wyniki bardziej szczegółowej dyrektywy explain w komentarzu

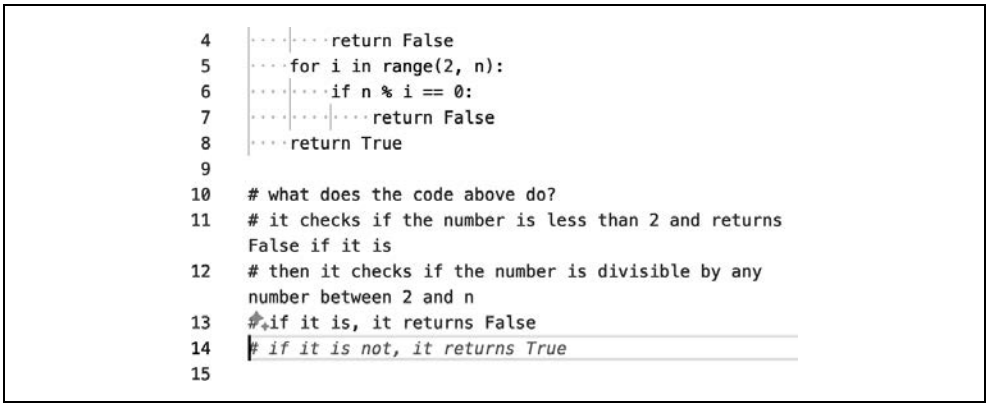
Innym zastosowaniem komentarzy jest zadawanie bezpośrednich pytań Copilotowi.

Pytania poprzez komentarze

Podobnie jak w przypadku przekazywania instrukcji Copilotowi za pomocą komentarzy, możesz również zadawać mu pytania. Na przykład zamiast kazać Copilotowi wyjaśnić kod, możesz zadać mu pytanie w następujący sposób:

```
# What does the code above do?
```

Rezultat jest pokazany na rysunku 2.15.



Rysunek 2.15. Wykorzystanie komentarza do zadania pytania Copilotowi

Możesz również formalnie oznaczyć swoje komentarze jako pytania przez dodanie `q:` po znaku komentarza, ale nie jest to wymagane. Jeśli to zrobisz, Copilot udzieli odpowiedzi zaczynającej się od `a:`.

Po omówieniu sposobów interakcji z Copilotem w edytorze przyjrzyjmy się teraz, jak korzystać z Copilota za pomocą menu środowiska programistycznego.

Kontekst pytań i odpowiedzi

Zadawanie pytań Copilotowi za pomocą komentarzy było wcześniej, zanim Copilot Chat dołączono do planów Copilota, sposobem na uzyskanie odpowiedzi w formie konwersacyjnej. Obecnie to podejście jest zwykle mniej przydatne niż korzystanie z interfejsu konwersacyjnego w edytorze lub głównego okna czatu, ale jeśli chcesz uzyskać krótkie odpowiedzi bezpośrednio w kodzie, nadal warto je rozważyć.

Copilot a menu kontekstowe

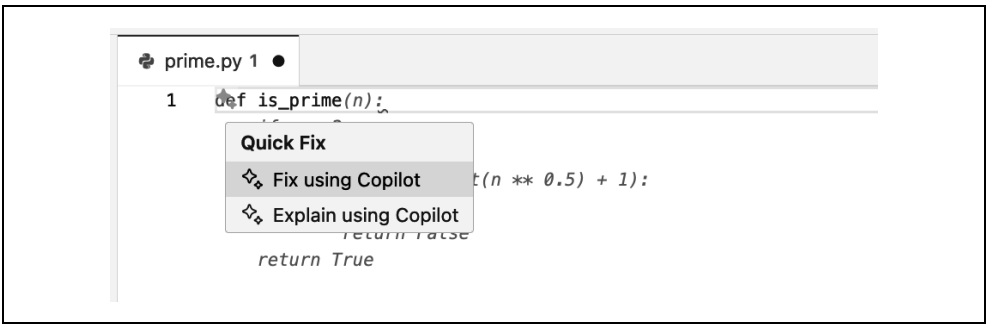
Niektóre funkcje Copilota są dostępne poprzez menu kontekstowe w twoim środowisku programistycznym. Ta integracja umożliwia szybkie uruchamianie zaawansowanych funkcji bez konieczności wydawania poleceń w oknie czatu (interfejsy konwersacyjne są omówione w rozdziale 3.)

Jeszcze o kontekście

Używany w tym podrozdziale termin *kontekst* nie odnosi się do informacji, które Copilot gromadzi w celu proponowania uzupełnień. Dotyczy on natomiast opcji Copilota dostępnych w **menu kontekstowym** twojego środowiska programistycznego.

Na przykład być może zauważyłeś na niektórych wcześniejszych rysunkach w tym rozdziale dwie małe **gwiazdki czteroramienne** na początku wiersza. Te dwie gwiazdki tworzą tak zwaną ikonę **iskry**, która jest ogólnym symbolem (nie tylko w Copilocie) oznaczającym sztuczną inteligencję. W środowisku programistycznym obecność tego symbolu zwykle wskazuje na odnośnik, który możesz kliknąć, aby wejść w interakcję z Copilotem. Kliknięcie go spowoduje z kolei otwarcie menu z kilkoma zaawansowanymi opcjami.

W omawianych przypadkach ikona pojawia się wtedy, gdy definicja funkcji jest niekompletna. Jednak system nie potrafi rozróżnić między funkcją, która jest jeszcze w trakcie pisania, a taką, która jest uszkodzona. Dlatego Copilot zaproponuje uzupełnienie i zaoferuje **naprawę** (ang. *fix*). Kliknięcie ikony iskry otwiera zaprezentowane na rysunku 2.16 menu podręczne (warto zauważyć, że ta funkcja może nie być dostępna w innych środowiskach programistycznych).

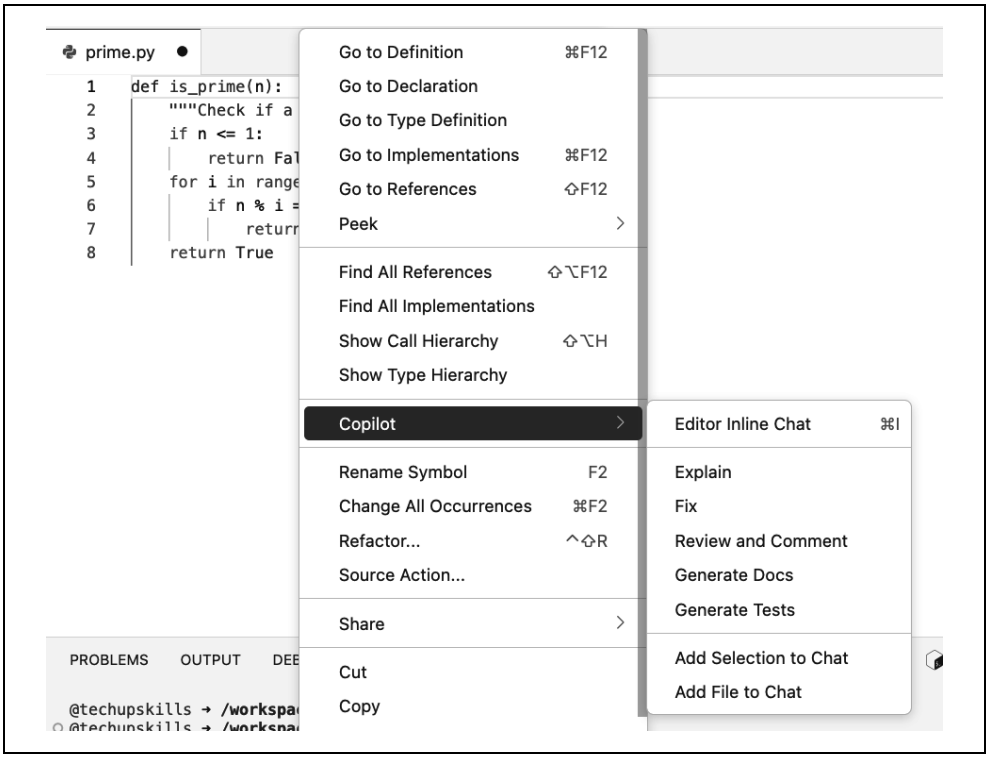


Rysunek 2.16. Opcja szybkiej naprawy z użyciem Copilota

Wykorzystanie poprawek do ukończenia kodu

Opcja *Fix using Copilot* może czasami stanowić alternatywę dla sugerowanych uzupełnień kodu. Jej wybór może spowodować, że Copilot wygeneruje częściowy kod, aby przywrócić działanie systemu. Nie można jednak liczyć na to, że ta funkcja zapewni pełne uzupełnienie kodu.

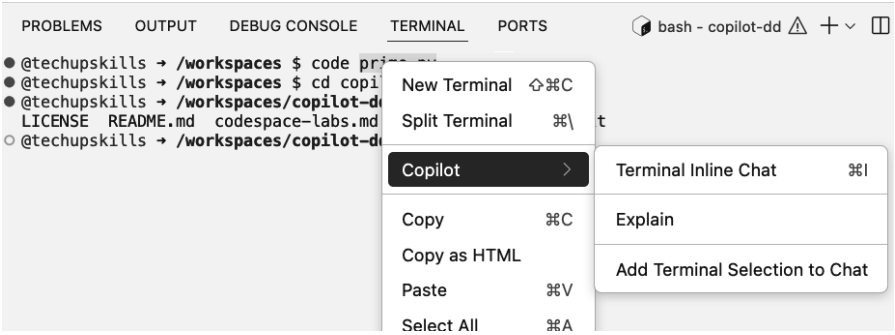
Gdy masz więcej danych, na których Copilot może pracować (na przykład kompletną implementację funkcji), zobaczysz dodatkowe opcje w menu (rysunek 2.17).



Rysunek 2.17. Rozszerzone opcje menu kontekstowego

Opcje dostępne w menu (oraz to, do czego się odnoszą) zmieniają się w zależności od tego, w jakim obszarze środowiska programistycznego właśnie pracujesz. Na przykład, jeśli korzystasz z terminala, opcja wyjaśniania (*Explain*) w menu odnosi się do zawartości terminala, a nie edytora. Jak widać na rysunku 2.18, istnieją również różne opcje dotyczące wyłącznie terminala (są to opcje czatu, dlatego wyjaśnię je w następnym rozdziale).

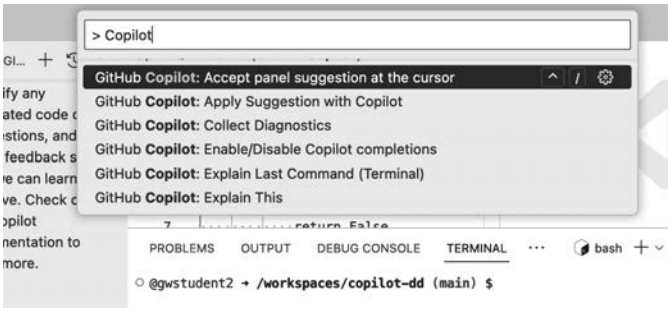
Menu kontekstowe udostępniają podzbiór funkcji Copilota, które są odpowiednie do danego zadania. W programie VS Code i podobnych środowiskach programistycznych możesz także wywoływać polecenia związane z Copilotem za pomocą interfejsu zwanego **paletą poleceń** (ang. *Command Palette*), co omówiono w poniższej ramce.



Rysunek 2.18. Opcje Copilota w terminalu

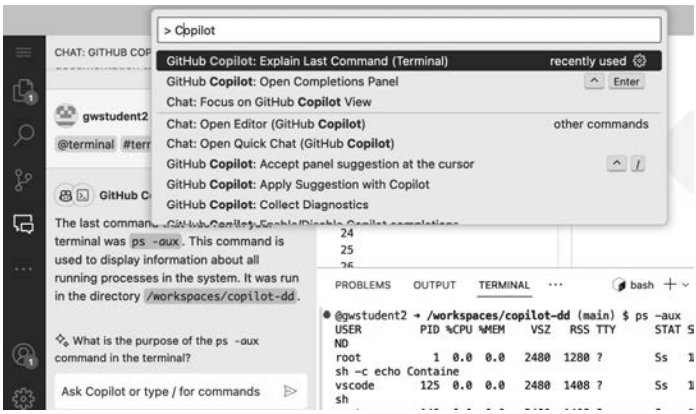
Dostęp do Copilota za pomocą palety poleceń

Środowiska programistyczne typu VS Code mają interfejs umożliwiający szybki dostęp do poleceń związanych z zainstalowanymi rozszerzeniami. Ten interfejs, nazywany paletą poleceń, można wywołać za pomocą skrótu *Meta+P* lub klawisza *F1*. Otwiera się wtedy okno dialogowe, w którym możesz przewijać lub wpisywać nazwy rozszerzeń, poleceń, symbolów i plików. Na przykład, gdy masz zainstalowane rozszerzenie Copilot, możesz uzyskać dostęp do powiązanych z nim poleceń poprzez paletę poleceń przez wpisanie **Copilot**, jak pokazano na rysunku 2.19.



Rysunek 2.19. Dostęp do funkcji Copilota z poziomu palety poleceń

Pozycje na tej liście można wykonywać jako polecenia. Na rysunku 2.20 pokazano, jak uruchomić funkcję wyjaśniania (*Explain*) z listy. Lista ta jest również pomocna w prezentowaniu wszystkich dostępnych poleceń związanych z Copilotem.



Rysunek 2.20. Przykład wykorzystania palety poleceń

Oprócz omówionych tu *standardowych* funkcji, takich jak wyjaśnianie kodu i naprawianie błędów, możesz również użyć menu kontekstowego, aby zlecić Copilotowi wykonanie bardzo przydatnego i ważnego zadania: przejrzenia twojego kodu i dostarczenia informacji zwrotnej na jego temat. Teraz przyjrzymy się temu mechanizmowi.

Wykorzystanie Copilota do przeglądu kodu

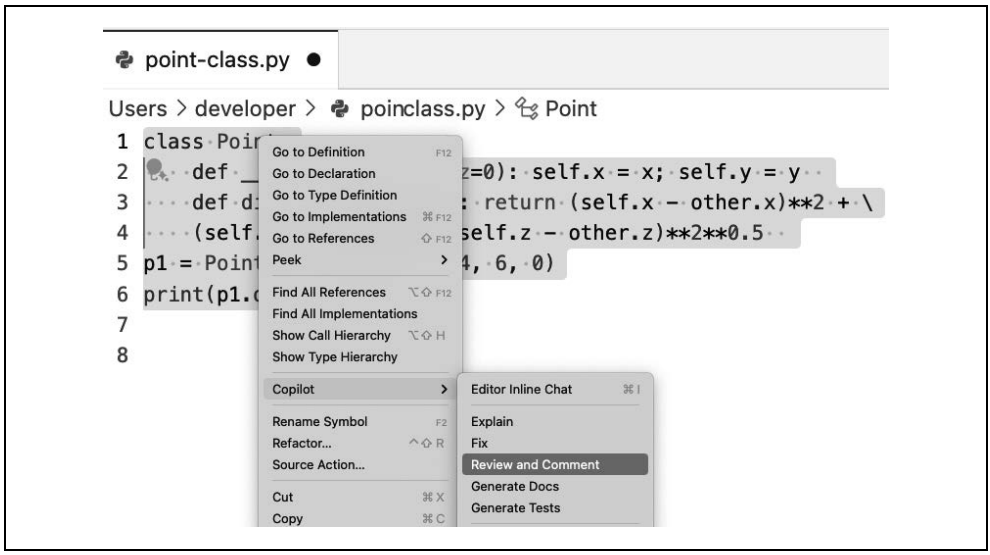
Copilot może odgrywać rolę recenzenta twojego kodu. W serwisie GitHub możesz dodać Copilota jako recenzenta w żądaniach scalenia (patrz rozdział 9.). W środowisku programistycznym Copilot może sprawdzić twój lokalny kod i dostarczyć informacji zwrotnych bezpośrednio w kodzie. Jest to szczególnie przydatne do wykrywania problemów przed wysłaniem kodu do repozytorium. Sposób, w jaki Copilot przeprowadza przegląd, możesz pozostawić w domyślnym ustawieniu albo go dostosować.

Domyślne opcje przeglądu kodu w Copilocie

Przeanalizujmy następujący fragment kodu, który ma na celu zaimplementowanie prostej klasy reprezentującej punkt:

```
class Point:
    def __init__(self, x, y, z=0): self.x = x; self.y = y
    def distance(self, other): return (self.x - other.x)**2 + \
    (self.y - other.y)**2 + (self.z - other.z)**2**0.5
p1 = Point(1, 2); p2 = Point(4, 6, 0)
print(p1.distance(p2))
```

Jeśli jesteś programistą Pythona, prawdopodobnie zauważysz niektóre problemy w tym kodzie. Sprawdźmy, co Copilot ma do powiedzenia na ten temat. Aby to zrobić, zaznaczamy kod w edytorze, klikamy go prawym przyciskiem myszy i wybieramy opcję *Copilot* > *Review and Comment* (jak pokazano na rysunku 2.21).



Rysunek 2.21. Uruchomienie przeglądu kodu z menu

Jeśli Copilot po przetworzeniu zapytania ma jakieś uwagi dotyczące problemów lub sugerowanych zmian (nawet tak prostych jak dodanie dokumentacji), przedstawi je wraz z proponowanymi modyfikacjami bezpośrednio w kodzie. Na rysunku 2.22 jest pokazany przykład elementu przeglądu wygenerowanego przez Copilota.

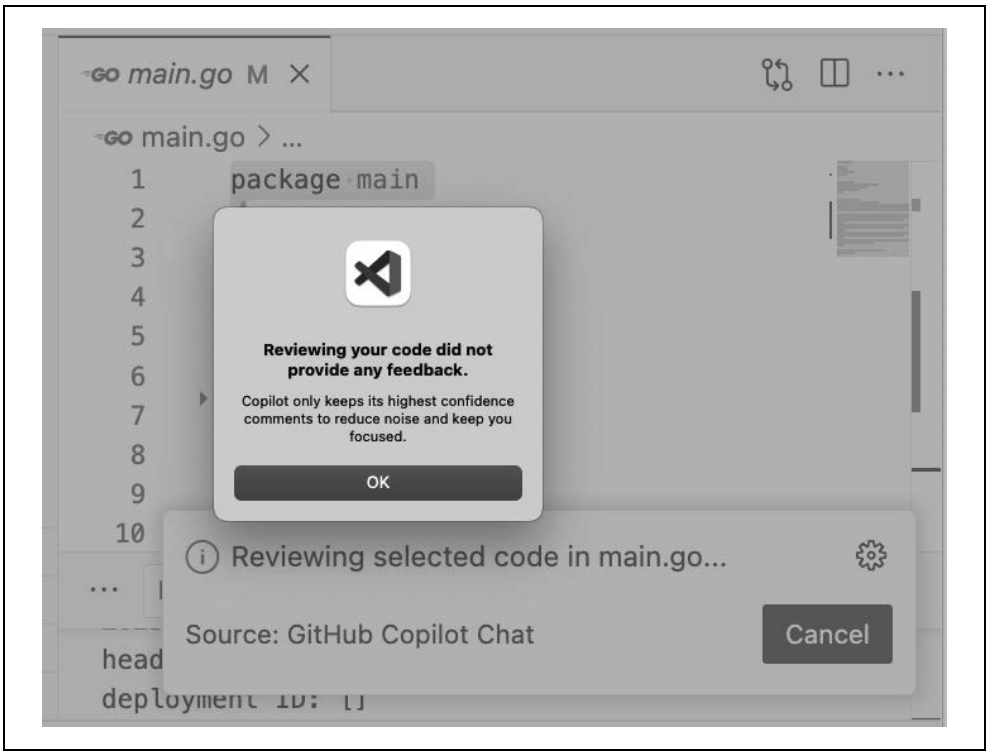
Rysunek 2.22. Informacje zwrotne z przeglądu wykonanego przez Copilota

Okno dialogowe z informacją zwrotną zawiera przyciski ze strzałkami, które umożliwiają przechodzenie między komentarzami przeglądu, a także opcje wyświetlenia uwag w oknie czatu, zamknięcia okna i zwinięcia go. Proponowane zmiany są prezentowane jako aktualny kod w kolorze czerwonym (ze znakiem minus) oraz sugerowany kod w kolorze zielonym (ze znakiem plus). Przyciski na dole okna ułatwiają zastosowanie lub odrzucenie zmiany, a następnie przejście do kolejnego zestawu uwag.

Oprócz domyślnego zachowania podczas przeglądu możesz również określić, na co Copilot ma zwracać uwagę w trakcie przeglądania kodu.

Tworzenie niestandardowych instrukcji przeglądu

Domyślnie Copilot zgłasza tylko problemy, które uważa za istotne. Dlatego nie jest niczym niezwykłym, że nie zgłasza żadnych uwag do kodu zawierającego elementy, które mogą zostać zakwestionowane przez innych recenzentów, takie jak niezgodność z najlepszymi praktykami czy wytycznymi dotyczącymi stylu. Na rysunku 2.23 jest pokazany przykładowy komunikat, który zobaczysz, gdy Copilot nie znajdzie żadnych problemów.



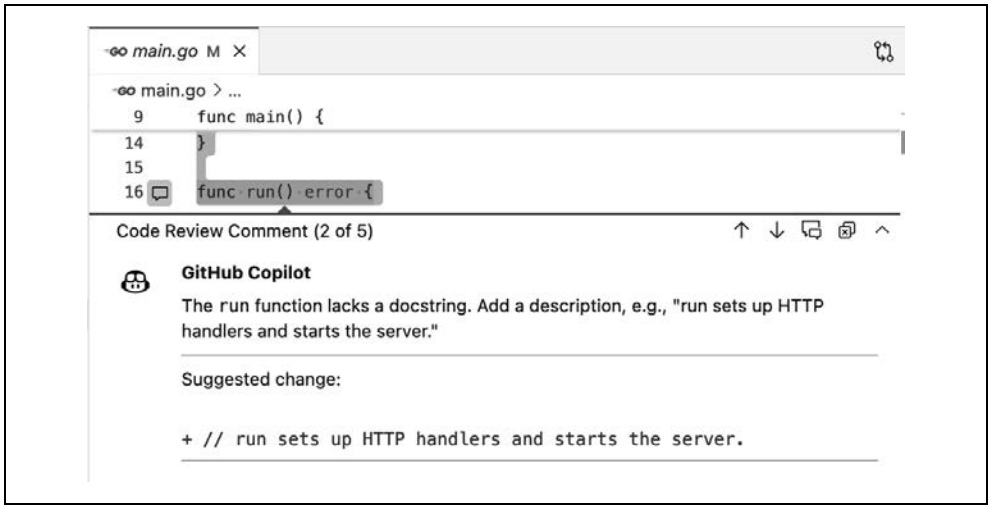
Rysunek 2.23. Brak informacji zwrotnej z przeglądu kodu wykonanego przez Copilota

Jeśli chcesz otrzymać bardziej szczegółowe informacje zwrotne, możesz wprowadzić zestaw własnych instrukcji do przeglądu kodu, z których Copilot skorzysta. Można tego dokonać na dwa sposoby.

Po pierwsze, możesz edytować plik *settings.json* i dodać własne instrukcje w sekcji `github.copilot.chat.codeReview.instructions`. Oto przykład reguły, którą możesz dodać:

```
"github.copilot.chat.reviewSelection.instructions": [
    {
        "text": "Ensure all functions have proper docstrings"
    }
]
```

Po dodaniu reguły w pliku ustawień użytkownika możesz poprosić Copilota o ponowne przeanalizowanie tego samego kodu. Tym razem Copilot oznaczy warunek, który określiłeś w pliku ustawień, co pokazano na rysunku 2.24.



Rysunek 2.24. Oznaczony problem w pliku ustawień

Ewentualnie możesz również dodać swoją regułę (i do pięciu dalszych) w formacie Markdown w lokalnym pliku i zapisać go w określonej ścieżce *.github/copilot-review-guidelines.md*, a Copilot automatycznie go znajdzie i wykorzysta.

Jeśli wolisz przechowywać instrukcje w innym pliku, możesz go stworzyć, a następnie wskazać ten lokalny plik w pliku *settings.json* z użyciem klucza `file:`. Oto przykład, jak może to wyglądać w ustawieniach:

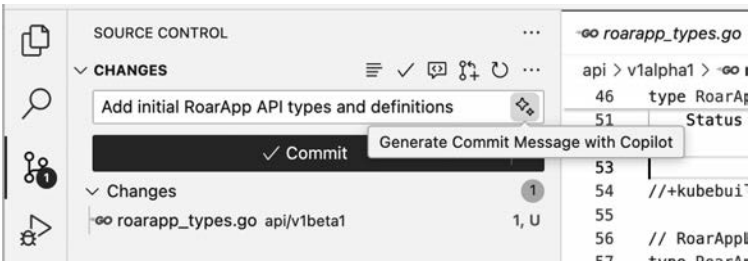
```
"github.copilot.chat.reviewSelection.instructions": [
    {
        "file": "./docs/review-guide.md"
    }
]
```

Zakładamy, że w katalogu *docs* twojego projektu znajduje się lokalny plik o nazwie *review-guide.md*, zawierający następującą treść:

```
- Ensure all functions have proper docstrings.
```

Generowanie automatycznych komunikatów zatwierdzeń za pomocą Copilota

Po zakończeniu przeglądu kodu, gdy jesteś gotowy do zatwierdzenia swoich zmian, Copilot może ci w tym pomóc. Jak pokazano na rysunku 2.25, korzystając z panelu kontroli źródła (*Source Control*), zamiast wpisywać własną wiadomość zatwierdzającą (ang. *commit message*), możesz poprosić Copilota o zasugerowanie jej na podstawie wprowadzonych zmian. Aby to zrobić, wystarczy kliknąć ikonę iskierki na końcu pola wiadomości zatwierdzającej.



Rysunek 2.25. Sugerowanie opisów zatwierdzeń przez Copilota

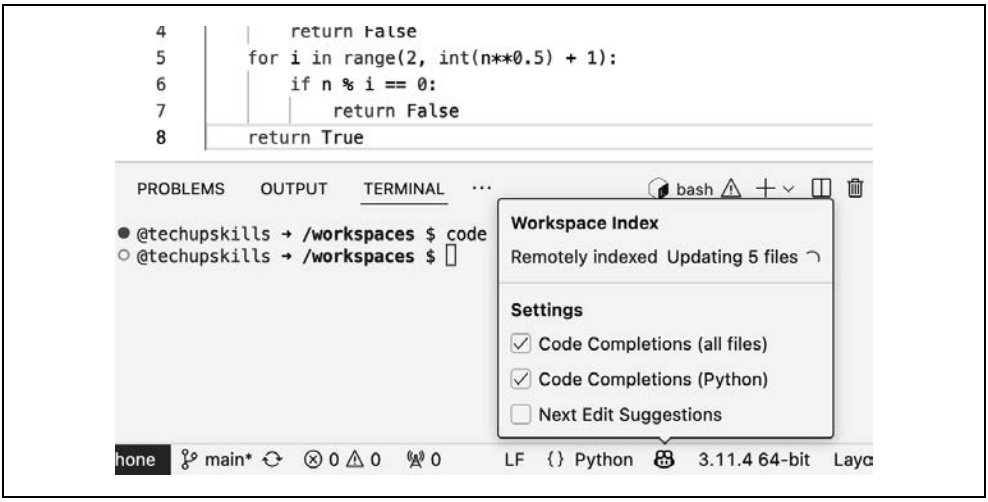
Możesz nawet skonfigurować własne instrukcje do generowania opisów zatwierdzeń. Szczegółowe informacje na ten temat znajdziesz w dokumentacji VS Code.

Na zakończenie naszej dyskusji o programowaniu z Copilotem w środowisku IDE przyjrzyjmy się pokrótce kilku dodatkowym opcjom sterowania i ustawieniom konfiguracyjnym, które pomogą ci dostosować pracę z Copilotem do własnych preferencji.

Konfiguracja Copilota w środowisku programistycznym

W środowisku VS Code po zainstalowaniu rozszerzenia Copilot (i ewentualnym uwierzytelnieniu) możesz korzystać z funkcji Copilota za pomocą kilku elementów sterujących. Jeden z nich znajduje się na pasku stanu u dołu środowiska IDE. Kliknięcie go umożliwia szybkie wyłączenie sugestii automatycznego uzupełniania kodu dla wszystkich plików lub tylko dla typu pliku, nad którym właśnie pracujesz. Na rysunku 2.26 pokazany jest przykład wybranego elementu sterującego, który oferuje opcję wyłączenia sugestii automatycznego uzupełniania kodu dla plików Python, ponieważ jest to obecnie aktywny typ pliku.

Opcja *Next Edit Suggestions* w oknie dialogowym odnosi się do funkcji automatycznej **edycji wsadowej** (ang. *batch editing*), która jest omówiona w rozdziale 4.

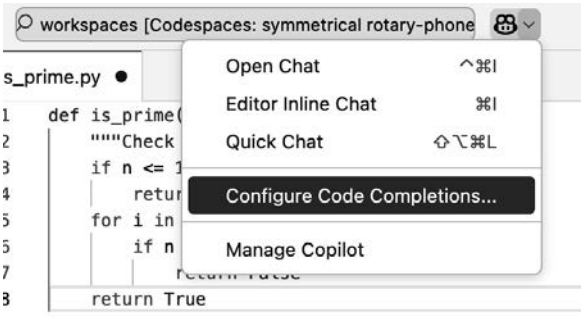


Rysunek 2.26. Dolne elementy sterujące Copilota

Dlaczego wyłączamy uzupełnianie?

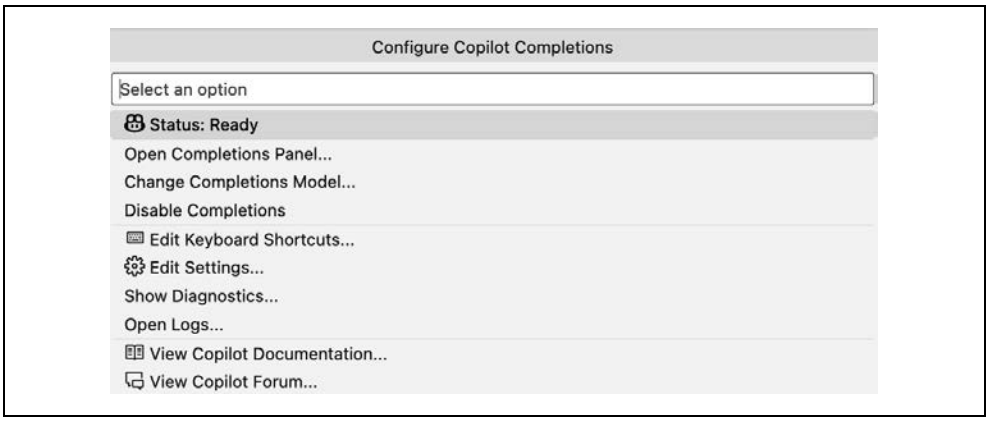
Możesz się zastanawiać, dlaczego istnieje opcja wyłączenia podpowiedzi kodu, skoro jest to jedna z podstawowych funkcji Copilota. Odpowiedź jest prosta: czasami te sugestie mogą być irytujące lub przeszkadzać. Na przykład, jeśli próbujesz szybko wprowadzić dużo kodu lub skupiasz się na tworzeniu kodu, który już znasz, możesz nie chcieć dodatkowego rozproszenia w postaci pojawiających się podpowiedzi Copilota.

Na górze IDE znajduje się kolejna ikona Copilota. Umożliwia ona otwarcie różnych interfejsów czatu, konfigurowanie podpowiedzi kodu oraz zarządzanie ustawieniami Copilota w serwisie GitHub (rysunek 2.27).



Rysunek 2.27. Górne elementy sterujące

W rozdziale 3. omówimy opcje czatu. Opcja *Manage Copilot* przeniesie cię na stronę ustawień (*https://oreil.ly/OlrYN*) twojego konta Copilot w serwisie GitHub (jeśli jesteś zalogowany). Opcja *Configure Code Completions* wyświetli zaś zestaw ustawień (rysunek 2.28) pozwalających dostosować sposób działania funkcji uzupełniania kodu.



Rysunek 2.28. Konfiguracja autouzupełniania w Copilocie

Niektóre z tych opcji są oczywiste, ale nie wszystkie. Oto krótkie omówienie trzech z nich:

Open Completions Panel

Otwiera listę alternatywnych propozycji uzupełnienia dla edytowanego kodu. Jest to ta sama funkcja, którą omówiliśmy wcześniej, wywoływana skrótem *Meta+Enter*.

Change Completions Model

Pozwala wybrać inny model AI (zamiast domyślnego), jeśli jest dostępny. Warto zauważyć, że zmiana dotyczy tylko modelu stosowanego do uzupełniania kodu, a nie modelu używanego w czacie.

Disable Completions

Podobnie jak wcześniejsza opcja z innego menu Copilota pozwala włączyć lub wyłączyć sugestie uzupełniania.

Opcja *Edit Settings* otwiera dostęp do dalszej personalizacji i konfiguracji Copilota. Kliknięcie jej powoduje otwarcie ustawień w środowisku IDE z wyszukiwaną frazą *GitHub Copilot*. Ten sam efekt można osiągnąć przez kliknięcie ikony koła zębatego oznaczającego ustawienia w głównym interfejsie IDE, a następnie wpisanie *GitHub Copilot* w polu wyszukiwania.

Rysunek 2.29 przedstawia ekran ustawień z pozycjami pasującymi do frazy *GitHub Copilot*. Zwróć uwagę na podkategorie po lewej stronie, dotyczące edytora, funkcji oraz szerszej grupy rozszerzeń. W grupie rozszerzeń znajdują się ustawienia dla wszystkich dodatków związanych ze środowiskiem GitHub Copilot.

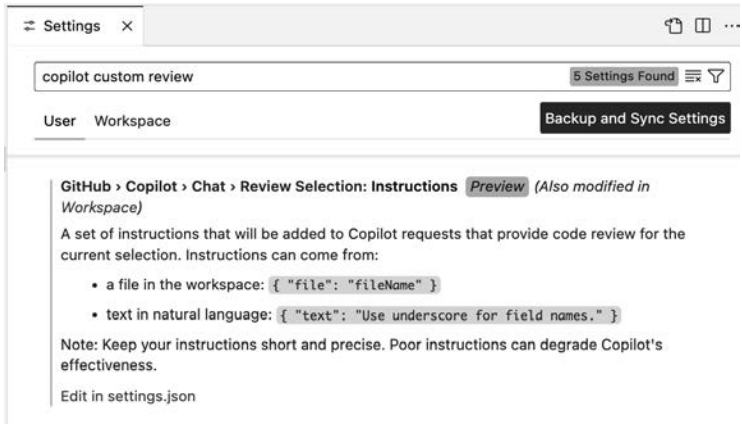
Rysunek 2.29. Ustawienia GitHub Copilot w środowisku programistycznym

W ustawieniach możesz skonfigurować opcje na **poziomie użytkownika** (niezależne od przestrzeni roboczej) oraz ustawienia dla **środowiska przestrzeni roboczej**. Zakładki w lewym górnym rogu pozwalają na przełączanie się między tymi kategoriami.

Podczas przeglądania niektórych obszarów ustawień napotkasz opcje oznaczone jako eksperymentalne (*Experimental*) lub zapoznawcze (*Preview*). Oba te oznaczenia wskazują na funkcje, które nie są jeszcze w pełni gotowe do użycia w środowisku produkcyjnym. **Funkcje w wersji zapoznawczej** są bliższe oficjalnego wydania i służą do publicznych testów oraz zbierania opinii od użytkowników, którzy zdecydują się z nich korzystać. **Funkcje eksperymentalne** to wczesne koncepcje, z których możesz korzystać na własne ryzyko. Mogą one zostać rozwinięte lub znacząco zmienione przed wprowadzeniem do wydania stabilnego, ale istnieje też możliwość, że w ogóle nie zostaną wdrożone.

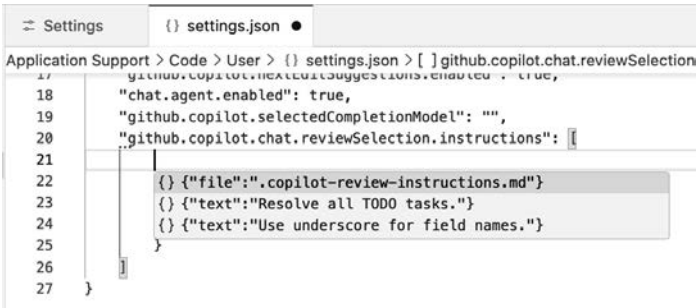
Edycja plików poprzez ustawienia

Jeśli znajdziesz opcję konfiguracyjną, którą można zapisać w pliku (na przykład *settings.json*), zazwyczaj zobaczysz odnośnik do opcji zmodyfikowania tego pliku. Na przykład, jeśli chcesz dodać własne reguły przeglądu (omówione wcześniej w tym rozdziale) i wyszukasz tę opcję, na dole zobaczysz odnośnik umożliwiający bezpośrednią edycję pliku *settings.json* (rysunek 2.30).



Rysunek 2.30. Dostęp do pliku z poziomu ustawień

Kliknięcie tego odnośnika spowoduje otwarcie wskazanego pliku, dzięki czemu możesz go bezpośrednio edytować na podstawie pewnych wstępnych wskazówek (rysunek 2.31).



Rysunek 2.31. Edycja pliku otwartego poprzez ustawienia

Jeśli korzystasz z innego środowiska programistycznego, będzie ono miało podobny sposób dostępu i zmiany ustawień Copilota. Zalecam śledzenie blogów serwisu GitHub dotyczących Copilota (*https://oreil.ly/qdhTv*), aby być na bieżąco z najnowszymi aktualizacjami i dowiedzieć się o nowych lub zmodyfikowanych ustawieniach. Dobrym pomysłem jest również okresowe przeglądanie dostępnych opcji. Czasami możesz odkryć dodatkowe, nieznane ci wcześniej możliwości konfiguracji działania Copilota.

Podsumowanie

W tym rozdziale omówiliśmy najbardziej bezpośrednie sposoby interakcji ze środowiskiem GitHub Copilot podczas pisania kodu. Copilot oferuje szybkie sugestie w trakcie programowania w trybie inline. Należy je oceniać pod kątem kompletności i dopasowania. Choć ich przydatność

może się różnić, sugestie te mogą zaoszczędzić czas i/lub stanowić punkt wyjścia do dalszych prac nad kodem i rozmyślań nad nim.

Jakość sugestii w dużej mierze zależy od ilości kontekstu, do którego Copilot może się odnieść w bieżącym środowisku, czyli nazwy pliku, istniejącego kodu, innych plików, komentarzy i lokalnego indeksu. Komentarze mogą również opisywać zadanie, które zlecasz Copilotowi, oraz służyć do zadawania pytań, na które odpowie on w formie komentarzy.

Copilot może analizować twój kod w poszukiwaniu problemów i sugerować ulepszenia. Możesz określić, na co ma zwracać uwagę podczas analizy, przez wskazanie listy własnych instrukcji.

Chociaż Copilot oferuje przydatne funkcje od razu po instalacji, czasami możesz chcieć dostosować jego działanie lub włączyć/wyłączyć niektóre ustawienia. Możesz wprowadzić ograniczone zmiany za pomocą elementów sterujących Copilota dostępnych w środowisku programistycznym. Możesz też przeszukać ustawienia środowiska w poszukiwaniu poszczególnych opcji do modyfikacji, w tym eksperymentalnych lub zapoznawczych.

Gdy już wiesz, jak pracować z sugestiami Copilota w trybie inline, przejdźmy do najbardziej elastycznego sposobu interakcji z Copilotem i jego bazowym modelem: poprzez interfejs konwersacyjny, który jest tematem rozdziału 3.

Ta książka jest praktyczna, przemyślana i oparta na codziennej pracy programistów z kodem!

— **Andrew Stellman**, programista, kierownik zespołu, instruktor i autor

Idea, by komputer asystował w pisaniu kodu, jeszcze niedawno wydawała się nierealna lub co najmniej odległa. Dziś to codzienność. Wśród dostępnych rozwiązań wyróżnia się GitHub Copilot, który wykorzystuje sztuczną inteligencję, by analizować kontekst, przewidywać intencje programisty i proponować gotowe rozwiązania. To nie tylko narzędzie, ale także dowód na to, jak szybko AI stała się integralną częścią procesu twórczego w świecie technologii.

Dzięki tej książce radykalnie zwiększysz swoją produktywność z pomocą generatywnej sztucznej inteligencji dostępnej w środowisku GitHub Copilot. Nauczysz się szybszego pisania kodu, łatwiejszego przygotowywania testów i opracowywania rzetelnej dokumentacji. Poznasz też zaawansowane możliwości, takie jak autonomiczne dodawanie funkcji i automatyczne przeglądanie żądań pobierania. Poza podstawami zaznajomisz się z trybami Copilot Edits, Agent i Vision. Zaczniesz również tworzyć własne rozszerzenia Copilota i tym samym zyskiwać dodatkowe możliwości — bez względu na to, z którego języka korzystasz.

To najlepszy przewodnik po potężnym narzędziu, jakim jest GitHub Copilot!

— **Tom Taulli**, autor książki *Programowanie wspomagane sztuczną inteligencją*

W książce między innymi:

- zastosowanie AI do analizy i zrozumienia trudniejszych fragmentów kodu i algorytmów
- tryb uzupełniania kodu w edytorze i interfejs konwersacyjny
- przekształcanie promptów w działające funkcje, testy i dokumentację
- optymalizacja wyników generowanych przez AI za pomocą kontekstu i promptów
- opracowywanie funkcji i refaktoryzacja kodu z użyciem AI

Brent Laster jest uznanym w świecie trenerem i autorem książek. Specjalizuje się w tematyce sztucznej inteligencji, DevOps i nowoczesnych metod inżynierii oprogramowania. Wcześniej był programistą i menedżerem w wiodących firmach technologicznych.

Helion
helion.pl
HELION S.A.
ul. Kościuszki 1c
44-100 Gliwice
tel.: 32 230 98 63
helion@helion.pl
KOD KORZYŚCI
Sięgnij po więcej!
ISBN 978-83-289-3474-0
9 788328 934740
Cena: 89,00 zł