Tworzenie pierwszej aplikacji w Pythonie

Więc chcesz zbudować aplikację w Pythonie, co? Niezależnie od tego, czy chcesz zakodować stronę internetową, przeanalizować dane, czy stworzyć skrypt do automatyzacji, ta częśćł zawiera wszystkie podstawy potrzebne do rozpoczęcia podróży. Większość ludzi używa języków programowania, takich jak Python, do tworzenia programów użytkowych, które są często określane jako po prostu aplikacje lub po prostu programy, a nawet aplikacje w skrócie. Aby tworzyć aplikacje, musisz wiedzieć, jak pisać kod w edytorze kodu. Musisz także zacząć uczyć się języka, w którym będziesz tworzyć te aplikacje. Jak każdy język, musisz zrozumieć poszczególne słowa, aby móc zacząć tworzyć zdania, a na koniec bloki kodu, które umożliwią działanie Twojej aplikacji. Najpierw przeprowadzimy Cię przez proces tworzenia pliku aplikacji, w którym tworzysz swój kod. Następnie dowiesz się wszystkiego o różnych typach danych, operatorach i zmiennych, które są słowami języka Python, a następnie o składni Pythona. Po drodze zobaczysz, jak zapisać swoją aplikację, wyłapać błędy z lintingiem, a nawet jak skomentować swój kod, aby Ty i inni mogli zrozumieć, jak ją zbudowałeś i dlaczego. Jesteś gotowy?

Otwórz plik aplikacji Pythona

Jak wspomnieliśmy w poprzednich częściach, będziemy używać popularnego edytora Visual Studio Code (VS Code) do nauki języka Python i tworzenia aplikacji w języku Python. Zakładamy, że masz już skonfigurowane środowisko uczenia się/rozwoju, i wiesz, jak otworzyć główne narzędzia, Anaconda Navigator i VS Code. Aby kontynuować ćwiczenia praktyczne, zacznij od następujących kroków:

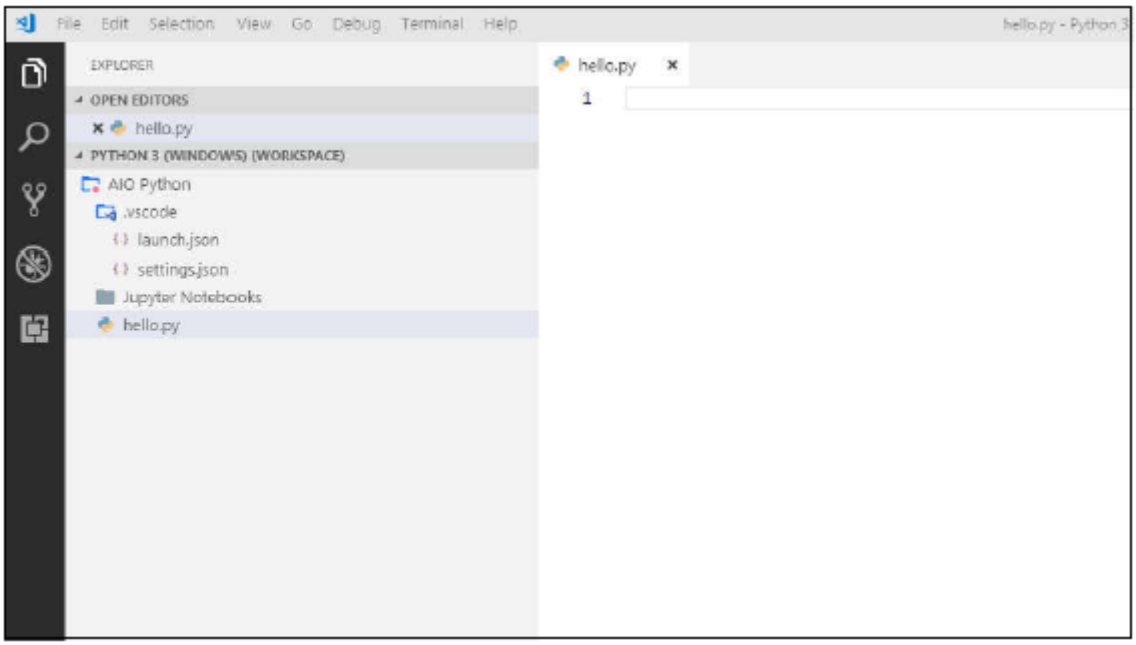
1. Otwórz Anaconda Navigator i stamtąd uruchom VS Code.

2. Jeśli przestrzeń robocza Pythona 3 nie otwiera się automatycznie, wybierz Plik -> Otwórz obszar roboczy z menu VS Code i otwórz obszar roboczy Python 3 utworzony w Części 2.

3. Kliknij plik hello.py utworzony w poprzedniej części.

4. Zaznacz cały tekst w pierwszym wierszu i usuń go, abyśmy mogli zacząć od zera tutaj.

W tym momencie powinieneś otworzyć hello.py w edytorze, jak pokazano na rysunku 4-1. Jeśli nadal masz otwarte inne karty, zamknij je teraz, klikając X w każdej z nich.



Pisanie i używanie komentarzy Pythona

Zanim napiszesz jakiś kod, zacznijmy od komentarza programisty. Komentarz programisty (zwykle nazywany w skrócie komentarzem) to tekst w programie, który nic nie robi. Co rodzi pytanie . . ”. Jeśli nic nie robi, to po co to wpisywać?” Jako osoba ucząca się możesz używać komentarzy w kodzie jako notatek do siebie na temat tego, co robi kod. Mogą one bardzo pomóc, gdy zaczynasz się uczyć. Komentarze w kodzie nie są przeznaczone wyłącznie dla początkujących. Podczas pracy w zespołach profesjonaliści często używają komentarzy, aby wyjaśnić członkom zespołu, co robi ich kod. Deweloperzy będą również umieszczać komentarze w swoim kodzie jako notatki do siebie, więc jeśli w przyszłości przejrzą kod, będą mogli odwołać się do własnych notatek, aby przypomnieć sobie, dlaczego coś zrobili w kodzie. Ponieważ komentarz nie jest kodem, Twoje sformułowanie może być dowolne. Aby jednak zostać zidentyfikowanym jako komentarz, tekst musi:

* Zaczynać się od znaku funta

* Lub być ujęty w potrójne cudzysłowy

Jeśli jest to krótki komentarz (jeśli nie rozciąga się na dwie lub więcej linii), wystarczy wiodący znak krzyżyka. Często zobaczysz ten znak krzyżyka, po którym następuje spacja, jak w poniższym przykładzie, ale ta spacja jest opcjonalna:

To jest komentarz Pythona

Aby wpisać komentarz Pythona do własnego kodu

1. Kliknij obok 1 w VS Code pod zakładką hello.py i wpisz # To jest komentarz Pythona w mojej pierwszej aplikacji Pythona.

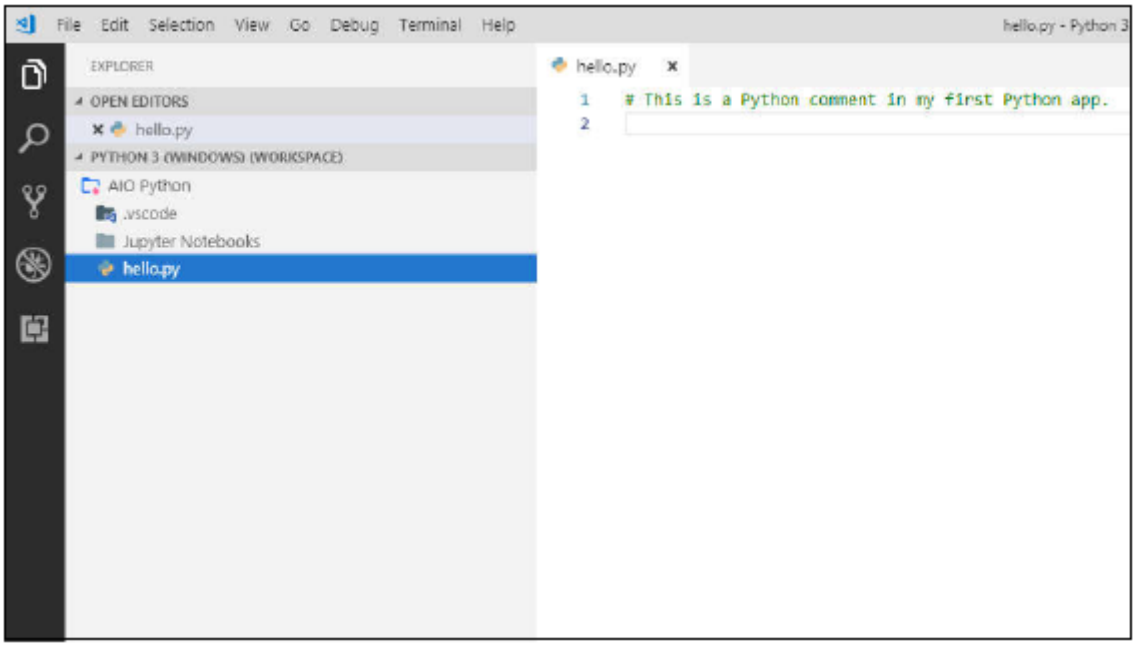
2. Naciśnij Enter.

Wpisany komentarz pojawi się w wierszu 1, jak na rysunku . Tekst komentarza będzie zielony, jeśli używasz domyślnego motywu kolorystycznego. Zauważysz, że migający kursor jest teraz w wierszu 2. Chociaż nie będziesz jeszcze używać komentarzy wielowierszowych, pamiętaj, że możesz pisać dłuższe komentarze wielowierszowe w Pythonie, umieszczając je w potrójnych cudzysłowach. Te większe komentarze są czasami nazywane ciągami dokumentacyjnymi i często pojawiają się na początku modułu Pythona, klasy funkcji lub definicji metody, które są blokami do budowania aplikacji, o których dowiesz się nieco później w tej książce. Nie trzeba go teraz wpisywać, ale oto przykład tego, jak może wyglądać w kodzie Pythona:

"""To jest wielowierszowy komentarz w Pythonie

Ten rodzaj komentarza jest czasami nazywany docstringiem.

Docstring zaczyna się od trzech podwójnych cudzysłowów ("""), a także kończy się trzema podwójnymi cudzysłowami.



W VS Code komentarze są zwykle kolorowane inaczej niż kod. Krótkie komentarze zaczynające się od # są zielone, a ciągi dokumentów są brązowe, aby pomóc im odróżnić się od rzeczywistego kodu Pythona, który jest uruchamiany po uruchomieniu aplikacji. Jeśli się zastanawiasz, nie ma limitu liczby komentarzy, które możesz umieścić w swoim kodzie. Jeśli czekasz, aż coś się wydarzy po wpisaniu komentarza . . . nie. Kiedy pracujesz w takim edytorze, kod nie robi nic, dopóki go nie uruchomisz. W tej chwili mamy tylko komentarz, więc nawet gdybyśmy to zrobili uruchom ten kod, nic by się nie stało, ponieważ kod jest przeznaczony dla ludzi, a nie komputerów. Zanim przejdziesz do praktyki i zaczniesz pisać rzeczywisty kod, musimy zacząć od absolutnych podstaw, czyli . . .

Zrozumienie typów danych Pythona

Cały czas masz do czynienia z informacją pisemną i prawdopodobnie nie myślisz o różnicy między liczbami a tekstem (czyli literami i słowami). Ale w komputerze jest duża różnica między liczbami a tekstem, ponieważ z liczbami można wykonywać operacje arytmetyczne (dodawać, odejmować, wielokrotne, dzielić). Na przykład wszyscy wiedzą, że 1 + 1 = 2. To samo nie dotyczy liter i słów. A A niekoniecznie musi być równe B, AA czy cokolwiek innego, ponieważ w przeciwieństwie do liczb, litery i słowa nie są ilościami. W sklepie można kupić 12 jabłek, ponieważ 12 to ilość, liczba - wartość skalarna w żargonie programistycznym. Tak naprawdę nie możesz kupić jabłek do fajki, ponieważ fajka to rzecz, a nie ilość, liczba ani wartość skalarna.

Liczby

Liczby w Pythonie muszą zaczynać się od cyfry (0-9); kropki (dot), która jest kropką dziesiętną; lub myślnik (-) używany jako znak ujemny dla liczb ujemnych. Liczba może zawierać tylko jeden punkt dziesiętny. Nie powinien zawierać liter, spacji, znaków dolara ani niczego innego, co nie jest częścią normalnej liczby. Tabela pokazuje przykład dobrych i złych liczb Pythona.

Liczba : Dobrze? : Powód

1: Dobrze : Całkowita (liczba całkowita)

1.1 : Dobrze : Liczba z kropką dziesiętną

1234567.89 : Dobrze : Duża liczba z przecinkiem i bez przecinków

-2 : Dobrze : Liczba ujemna, na co wskazuje początkowy myślnik

.99 : Dobrze : Liczba, która zaczyna się od kropką dziesiętną, ponieważ jest mniejsza niż jeden.

1,99 USD : Źle : Zawiera $

12 345,67 : Źle : Zawiera przecinek

1101 3232 : Źle : Zawiera spację

91740-3384 : Źle : Zawiera myślnik

123-45-6789: Źle : Zawiera dwa myślniki

123 Oak Tree Lane : Źle : Zawiera spacje i słowa

(267)555-1234 : Źle : Zawiera nawiasy i łączniki

127.0.0.1 : Źle : Dozwolony jest tylko jeden przecinek dziesiętny

Jeśli martwisz się, że zasady dotyczące liczb nie pozwolą Ci pracować z kwotami w dolarach, kodami pocztowymi, adresami lub czymkolwiek innym, przestań się martwić. Możesz przechowywać i pracować z różnymi rodzajami informacji, jak zobaczysz wkrótce.

Zdecydowana większość liczb, których używasz, prawdopodobnie będzie pasować do jednego z pierwszych czterech przykładów dobrych liczb. Jeśli jednak przyglądasz się kodowi używanemu w bardziej zaawansowanych zastosowaniach naukowych lub matematycznych, od czasu do czasu możesz zobaczyć liczby zawierające literę e lub literę j. To dlatego, że w rzeczywistości Python obsługuje trzy różne typy liczb, jak omówiono w poniższych sekcjach.

Liczby całkowite

Liczba całkowita to dowolna liczba całkowita, dodatnia lub ujemna. Nie ma ograniczeń co do jego wielkości. Liczby takie jak 0, -1 i 999999999999999 są całkowicie prawidłowymi liczbami całkowitymi. Z Twojej perspektywy liczba całkowita to po prostu dowolna poprawna liczba, która nie zawiera kropki dziesiętnej.

Liczby zmiennoprzecinkowe

Numer punktu osadzenia, często nazywany w skrócie float, to po prostu dowolna poprawna liczba zawierająca kropkę dziesiętną. Ponownie, nie ma limitu rozmiaru, 1.1 i -1.1 oraz 123456.789012345 są całkowicie prawidłowymi liczbami zmiennoprzecinkowymi.

Jeśli zdarzy ci się pracować z bardzo dużymi liczbami naukowymi, możesz umieścić e w liczbie, aby wskazać potęgę 10. Na przykład 234e1000 jest prawidłową liczbą i zostanie potraktowana jako zmiennoprzecinkowa, nawet jeśli nie ma przecinka dziesiętnego. Jeśli znasz notację naukową, wiesz, że 234e3 to 234 000 (zastąp e3 3 zerami). Jeśli nie znasz notacji naukowej, nie przejmuj się tym. Jeśli nie używasz go teraz w swojej codziennej pracy, prawdopodobnie nigdy nie będziesz go również potrzebował w Pythonie.

Liczby zespolone

Prawie każdy rodzaj liczby może być wyrażony jako liczba całkowita lub zmiennoprzecinkowa, więc znajomość ich jest wystarczająca dla prawie każdego. Chociaż w trosce o dokładność powinniśmy

zaznaczyć, że Python obsługuje również liczby zespolone. Te dziwaczne małe zaklinacze zawsze kończą się literą j, która jest wyimaginowaną częścią liczby. Jeśli absolutnie nie masz pojęcia, o czym mówimy, to jesteś normalną osobą, ponieważ dbają o to tylko ludzie, którzy naprawdę są „tam" w matematyce. Jeśli nigdy wcześniej o nich nie słyszałeś, prawdopodobnie nie będziesz ich używać w pracy z komputerem lub w programowaniu w Pythonie.

Słowa (ciągi)

Ciągi znaków są przeciwieństwem liczb. Liczby umożliwiają dodawanie, odejmowanie, mnożenie i dzielenie, ponieważ liczby reprezentują ilości. Ciągi służą prawie do wszystkiego innego. Nazwy, adresy i wszystkie inne rodzaje tekstu, które widzisz codziennie, byłyby ciągiem w Pythonie (i ogólnie dla komputerów). Nazywa się to ciągiem, ponieważ jest to ciąg znaków (litery, spacje, znaki interpunkcyjne, może jakieś cyfry). Dla nas ciąg zwykle ma jakieś znaczenie, takie jak imię lub adres osoby, ale komputery nie mają oczu, którymi można by patrzeć, ani mózgów, którymi można by myśleć, ani świadomości, że ludzie w ogóle istnieją, więc jeśli nie jest to coś, na czym można arytmetyka, to tylko ciąg znaków. W przeciwieństwie do liczb, ciąg musi być zawsze ujęty w cudzysłów. Możesz użyć podwójnych cudzysłowów („) lub singli („). Wszystkie poniższe są prawidłowymi ciągami:

"Cześć, jestem ciągiem"

'Witaj świecie'

„123 Aleja Dębów"

"(267)555-1234"

„18901-3384"

Zwróć uwagę, że doskonale jest używać znaków numerycznych (0-9), a także łączników i kropek w ciągach. Każdy jest nadal ciągiem, ponieważ jest ujęty w cudzysłów. Słowo ostrzeżenia. Jeśli ciąg zawiera apostrof (pojedynczy cudzysłów), to cały ciąg powinien być ujęty w podwójny cudzysłów w następujący sposób:

"Mary's dog said Woof"

Znaki podwójnego cudzysłowu są niezbędne, ponieważ nie ma wątpliwości, gdzie zaczyna się i kończy ciąg. Jeśli zamiast tego użyłeś pojedynczych cudzysłowów, na przykład:

‘'Mary's dog said Woof'

. . . komputer byłby zbyt głupi, żeby to zrobić dobrze. Zobaczyłby pierwszy pojedynczy cudzysłów jako początek ciągu, drugi (po Mary) jako koniec ciągu, a potem nie wiedziałby, co zrobić z resztą, a Twoja aplikacja nie działa poprawnie. Podobnie, jeśli ciąg zawiera znaki podwójnego cudzysłowu, całość powinna być ujęta w pojedyncze cudzysłowy, aby uniknąć pomyłek. Na przykład:

'The dog of Mary said "Woof".'

Zatem pierwszy pojedynczy cudzysłów rozpoczyna łańcuch, drugi go kończy, a podwójne cudzysłowy nie powodują zamieszania, ponieważ znajdują się w tym łańcuchu. Co z tego, że mamy łańcuch zawierający zarówno pojedyncze, jak i podwójne cudzysłowy, taki jak ten?:

Mary's dog said "Woof".

To zasługuje na głośne hmm. Na szczęście twórcy Pythona zdali sobie sprawę, że takie rzeczy mogą się zdarzyć, więc wymyślili ucieczkę. W rzeczywistości rzeczy, których używasz, nazywane są znakami

ucieczki, ponieważ w pewnym sensie pozwalają ci uciec (unikać) „specjalnego znaczenia” znaku, takiego jak pojedynczy lub podwójny cudzysłów. Aby zmienić znak, po prostu poprzedź go odwrotnym ukośnikiem (\). Upewnij się, że używasz odwrotnego ukośnika (tego, który odchyla się do poprzedniego znaku, na przykład \) lub nie zadziała prawidłowo. Korzystając z tego ostatniego przykładu, moglibyśmy ująć całość w pojedyncze cudzysłowy, a następnie uniknąć apostrofu (który jest tym samym znakiem), poprzedzając go odwrotnym ukośnikiem, w ten sposób:

'Mary\'s dog said "Woof".'

Lub możemy umieścić całość w podwójnych cudzysłowach i uciec przed cudzysłowami osadzonymi w łańcuchu, w ten sposób:

"Mary's dog said \"Woof\"."

Innym powszechnym zastosowaniem odwrotnego ukośnika jest dodanie łamania wiersza na końcu wiersza na ekranie, na którym użytkownik go przegląda (użytkownikiem jest każdy, kto korzysta z napisanej przez Ciebie aplikacji). Na przykład ten ciąg

"The old pond\nA frog jumped in,\nKerplunk!"

. . . wyglądałby tak po wyświetleniu użytkownikowi:

The old pond

A frog jumped in,

Kerplunk!

. . . ponieważ każdy \n zostałby przekonwertowany na podział wiersza, kończący tam wiersz

Wartości logiczne Prawda/Fałsz

W Pythonie istnieje trzeci typ danych, który nie jest dokładnie liczbą ani łańcuchem. Nazywa się Boolean (od nazwiska matematyka George'a Boole'a) i może być jedną z dwóch wartości: True lub False. Może wydawać się trochę dziwne posiadanie typu danych dla czegoś, co może być tylko prawdą lub fałszem. Jednak ze względu na sposób, w jaki komputery wykonują swoją pracę, efektywne jest ustawienie Prawda/Fałsz jako własnego typu danych. Pojedynczy bit, który jest najmniejszą jednostką pamięci w komputerze, jest wszystkim, czego potrzeba do przechowywania wartości, która może być tylko Prawda lub Fałsz. W kodzie Pythona ludzie przechowują wartości True lub False w zmiennych (elementy zastępcze w kodzie, które omówimy w dalszej części tego rozdziału) przy użyciu formatu podobnego do tego:

x = Prawda

. . . a może to:

X = Fałsz

Wiesz, że prawda i fałsz są tutaj wartościami logicznymi, ponieważ nie są ujęte w cudzysłów (jak w przypadku łańcucha) i nie są liczbami. Wymagany jest również początkowy nasadka. Innymi słowy, pierwsza litera musi być wielka, a wszystkie pozostałe litery muszą być małe.

Praca z operatorami Pythona

Korzystasz z komputerów na kilka sposobów. Jednym ze sposobów jest po prostu przechowywanie informacji, jak szafka na akta, ale bez papieru. Jak omówiliśmy w poprzedniej sekcji, w Pythonie i

ogólnie na komputerach pomaga myśleć o informacjach jako o jednym z następujących typów danych: liczba, ciąg lub Boolean. Korzystasz również z komputerów do operowania tymi informacjami, co oznacza, że wykonujesz wszelkie niezbędne obliczenia, porównania, wyszukiwania lub cokolwiek, aby pomóc Ci znaleźć informacje i zorganizować je w sposób, który ma dla Ciebie sens. Python oferuje wiele różnych operatorów do pracy i porównywania typów. Tutaj po prostu podsumowujemy je wszystkie do wykorzystania w przyszłości, bez wchodzenia w szczegóły. To, czy korzystasz z operatora we własnej pracy, zależy od rodzaju aplikacji, które tworzysz. Na razie wystarczy mieć świadomość, że są dostępne.

Operatory arytmetyczne

Jak sama nazwa wskazuje, operatory arytmetyczne służą do wykonywania arytmetyki; dodawanie, odejmowanie, mnożenie, dzielenie i inne. Tabela zawiera listę operatorów arytmetycznych Pythona

Operator : Opis : Przykład

+ : Dodawanie : 1 + 1 = 2

- : Odejmowanie : 10 - 1 = 9

*: Mnożenie : 3 * 5 = 15

/ : Dzielenie : 10 / 5 = 2

% : Moduł (pozostałość po dzieleniu) : 11 % 5 = 1

** : Wykładnik : 3**2 = 9

// : Floor division : 11 // 5 = 2

Pierwsze cztery pozycje w tabeli są takie same, jak nauczyłeś się w szkole podstawowej. Ostatnie trzy są nieco bardziej zaawansowane, więc wyjaśnimy je tutaj:

* Moduł to reszta po dzieleniu. Na przykład 11% 5 równa się 1, ponieważ jeśli podzielisz 11 przez 2, otrzymasz 5 resztę 1. To 1 to moduł (czasami nazywany modułem).

* Wykładnik to **, ponieważ w kodzie nie można wpisać małej liczby podniesionej. Ale oznacza to po prostu „podniesiony do potęgi" w tym sensie, że 3**2 to 3^2 lub 3 do kwadratu, czyli 3*3 lub 9. I oczywiście 3**4 byłoby 3*3*3*3 lub 81.

* Floor division, wskazany przez //, jest dzieleniem całkowitym w tym sensie, że wszystko po przecinku jest obcinane (ignorowane). Termin skrócony w tym sensie oznacza „odcięty", bez żadnego zaokrąglania. Na przykład w zwykłym dzieleniu 9/5 to 1,8. Ale 9//5 to tylko 1, ponieważ 0,8 jest właśnie odcięte, nawet nie zaokrągla 1 do 2.

Operatory porównania

Komputery mogą podejmować decyzje w ramach swojej pracy. Ale te decyzje nie są decyzjami „wezwania do sądu" ani niczym „ludzkim" w tym rodzaju. Są to decyzje oparte na faktach bezwzględnych opartych na porównaniach. Operatory, które Python oferuje, aby pomóc w pisaniu kodu, który podejmuje decyzje, są wymienione w tabeli.

Operator : Znaczenie

< : Mniej niż

<= : Mniejszy lub równy

> : Większe niż

>= : Większe lub równe

== : Równe

!= : Nie równe

is : Tożsamość obiektu

is not : Zanegowana tożsamość obiektu

Kilka pierwszych nie wymaga wyjaśnień, więc nie będziemy tam wchodzić w szczegóły. Ostatnie dwa są trudne, ponieważ dotyczą obiektów Pythona, o których jeszcze nie rozmawialiśmy. Mówienie teraz o obiektach Pythona byłoby wielką dygresją, więc jeśli w ogóle jesteś zdezorientowany co do jakichkolwiek operatorów, nie martw się tym. Na razie chcemy je tylko pokazać, aby w przypadku, gdy kiedykolwiek spojrzysz na kod innych osób, będziesz miał pewne pojęcie o tym, co mają na myśli.

Operatory logiczne

Operatory Boolean działają z wartościami Boolean (True lub False) i są używane do określenia, czy jedna lub więcej rzeczy jest True lub False. Tabela podsumowuje operatory logiczne.

Operator : Przykład kodu : Co określa

lub : x lub y : Albo x albo y jest prawdziwe

oraz : x i y : Zarówno x, jak i y są prawdziwe

nie : nie x : x nie jest pełne

Przewodnik po stylu Pythona zaleca zawsze umieszczanie białych znaków wokół operatorów. Innymi słowy, chcesz użyć spacji na klawiaturze, aby umieścić spację przed operatorem, następnie wpisać operator, a następnie dodać kolejną spację przed kontynuowaniem wiersza kodu. Oto dość prosty przykład. Wiemy, że jeszcze nie znasz się na kodowaniu, więc nie przejmuj się zbytnio dokładnym znaczeniem kodu. Zamiast tego zwróć uwagę na spacje wokół operatorów = i > (większe niż).

```
num = 10

if num > 0:

print("Positive number")

else:

print("Negative number")
```

Pierwsza linia przechowuje liczbę 10 w zmiennej o nazwie num. Następnie if sprawdza, czy liczba jest większa niż (>) 0. Jeśli tak, program wypisuje liczbę dodatnią. W przeciwnym razie drukuje liczbę ujemną. Załóżmy więc, że zmieniasz pierwszą linię programu na następującą:

```
num = -1
```

Jeśli dokonasz tej zmiany i ponownie uruchomisz program, wypisze liczbę ujemną, ponieważ -1 jest liczbą ujemną. W tym przykładzie użyliśmy num jako przykładowej nazwy zmiennej, aby pokazać kilka operatorów z przestrzenią wokół nich. Oczywiście nie powiedzieliśmy ci, czym są zmienne, więc ta część przykładu mogła sprawić, że podrapałeś się po głowie. Później wyjaśnimy tę część tej sprawy.

Tworzenie i używanie zmiennych

Zmienne są dużą częścią Pythona i wszystkich języków programowania komputerowego. Zmienna to po prostu miejsce na informacje, które mogą się zmieniać (zmieniać). Na przykład, kiedy dzisiaj przeglądasz Amazon, możesz zobaczyć swoje imię i nazwisko oraz datę „członek od" na ich stronie głównej. Z pewnością nie każdy, kto tego dnia idzie do Amazona, nazywa się Alan i jest członkiem od 1996 roku. Inni ludzie muszą tam widzieć inne rzeczy. Ale Amazon z pewnością nie może stworzyć niestandardowej strony głównej dla każdego z milionów użytkowników. Więc większość tego, co jest na tej stronie, jest prawdopodobnie dosłowne – co oznacza, że każdy, kto ją przegląda, widzi te same rzeczy. Tylko informacje, które zmieniają się w zależności od tego, kto przegląda stronę, są przechowywane jako zmienna (czyli tylko te informacje, które są zmienne). Zmienna w Pythonie jest po prostu miejscem przechowywania informacji, które mogą ulec zmianie w dowolnym momencie. Ale w twoim kodzie jest reprezentowana przez nazwę zmiennej, a nie przez miejsce. Oto inny sposób myślenia o tym. Za każdym razem, gdy kupujesz jeden lub więcej produktów, cena rozszerzona to cena jednostkowa pomnożona przez liczbę zakupionych przedmiotów. Innymi słowy

Ilość * Cena jednostkowa = Cena rozszerzona

Możesz uznać ilość i cenę jednostkową za zmienne, ponieważ bez względu na to, jakie liczby podłączysz do ilości i ceny jednostkowej, otrzymasz prawidłową cenę rozszerzoną. Na przykład, jeśli kupisz 3 turkawki za 1,00 USD za sztukę, Twoja rozszerzona cena wyniesie 3,00 USD (3 * 1,00 USD). Jeśli kupisz dwa tuziny róż po 1,50 USD za sztukę, rozszerzona cena wyniesie 36 USD, ponieważ 1,5 * 24 to 36. Tak więc, jeśli weźmiesz pod uwagę ilość i cenę jednostkową jako zmienne, w których możesz przechowywać liczby, otrzymasz prawidłową rozszerzoną cenę, gdy zwielokrotnisz te dwie liczby bez względu na to, jakich liczb użyjesz do zastąpienia ilości i ceny jednostkowej.

Tworzenie prawidłowych nazw zmiennych

W naszym wyjaśnieniu, czym jest zmienna, użyliśmy nazw takich jak Ilość i Cena jednostkowa, co z pewnością jest dobre dla ogólnego przykładu. To są imiona, które właśnie wymyśliliśmy. W Pythonie możesz także tworzyć własne nazwy zmiennych. Ale zasady są bardziej rygorystyczne niż w przypadku wymyślania ich w prostym języku angielskim. Nazwy zmiennych Pythona muszą być zgodne z pewnymi regułami, aby mogły być rozpoznawane jako nazwy zmiennych. Zasady to:

* Nazwa zmiennej musi zaczynać się od litery lub podkreślenia (_).

* Po pierwszej literze możesz użyć liter, cyfr lub podkreśleń. *W nazwach zmiennych jest rozróżniana wielkość liter, więc po utworzeniu nazwy każde inne odwołanie do tej zmiennej musi zawierać te same wielkie i małe litery, co nazwa, którą pierwotnie wymyśliłeś.

* Nazwy zmiennych nie mogą być ujęte ani zawierać pojedynczych lub podwójnych cudzysłowów.

* Konwencje stylu PEP 8 zalecają używanie tylko małych liter w nazwach zmiennych, użyj podkreślenia do oddzielenia wielu słów w nazwie zmiennej.

PEP 8, o którym wspomnieliśmy w poprzednich częściach, jest przewodnikiem stylu pisania kodu, a nie ścisłymi regułami, których należy przestrzegać. Dlatego często widzisz nazwy zmiennych, które nie są zgodne z tym ostatnim stylem. Formatowanie zapisu wielbłądziego – w którym pierwsza litera jest tworzona małymi literami, a nowe słowa są pisane wielkimi literami zamiast oddzielać je spacjami – jest dość powszechne, nawet w Pythonie. Na przykład ExtendedPrice lub unitPrice to terminy z wielkimi literami, które są technicznie poprawne i nie spowodują niepowodzenia programu. Ale doświadczeni puryści Pythona czasami mają „obrzydzenie" na twarzy, gdy widzą takie nazwy w twoim

kodzie. Woleliby, abyś trzymał się wytycznych dotyczących stylu PEP 8, które zalecają używanie extension_price i unit_price jako nazw zmiennych, ponieważ składnia PEP 8 jest bardziej czytelna dla ludzkich programistów.

Tworzenie zmiennych w kodzie

Aby utworzyć zmienną, użyj składni (kolejności rzeczy) w następujący sposób:

nazwa zmiennej = wartość

Jak wspomniano, nazwa zmiennej to nazwa, którą wymyślasz. Możesz używać x lub y, tak jak ludzie często robią to w matematyce, ale w większych programach dobrym pomysłem jest nadanie zmiennym bardziej znaczących nazw, takich jak ilość lub cena_jednostkowa, podatek_sprzedaży lub nazwa_użytkownika, aby zawsze móc zapamiętać, czym się zajmujesz. przechowywanie w zmiennej. Wartością jest to, co chcesz przechowywać w zmiennej. Może to być liczba, a ciąg znaków lub Boolean True lub False. Znak = w operatorze przypisania jest tak nazwany, ponieważ przypisuje wartość (po prawej) do zmiennej (po lewej). Na przykład tutaj . . .

x = 10

. . . przechowujemy liczbę 10 w zmiennej o nazwie x. Innymi słowy, przypisujemy wartość 10 zmiennej o nazwie x.

Tutaj. . .

user_name = "Alan"

Umieszczamy ciąg Alan w zmiennej o nazwie username.

Manipulowanie zmiennymi

Wiele programów komputerowych kręci się wokół przechowywania wartości w zmiennych i manipulowania tymi informacjami za pomocą operatorów. Przyjrzyjmy się i wypróbujmy kilka prostych przykładów, aby to zrozumieć. Jeśli nadal masz otwarty program VS Code z widocznym jednym komentarzem, wykonaj następujące kroki w edytorze programu VS Code:

1. Pod wierszem, który brzmi # This is a Python comment w mojej pierwszej aplikacji Pythona, wpisz ten komentarz # Ta zmienna zawiera liczbę całkowitą.

2. Naciśnij Enter, wpisz quatity = 10 (nie zapomnij wstawić spacji przed i po znaku =) i naciśnij Enter.

3. Wpisz # Ta zmienna zawiera zmiennoprzecinkową i ponownie naciśnij Enter.

4. Wpisz unit_price = 1,99 i ponownie naciśnij Enter (nie wpisuj znaku dolara!).

5. Wpisz # Ta zmienna zawiera wynik przemnożenia ilości przez cenę jednostkową i naciśnij Enter.

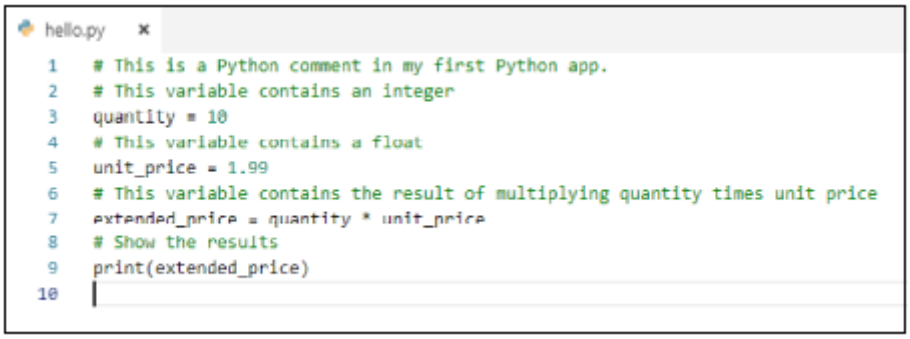
6. Wpisz extend_price = ilość * unit_price (znowu ze spacjami wokół operatorów) i naciśnij Enter.

7. Wpisz # Pokaż wyniki i naciśnij Enter.

8. Wpisz print(extended_price) i naciśnij Enter.

Twoja aplikacja Pythona tworzy pewne zmienne, przechowuje w nich pewną wartość, a nawet oblicza nową wartość, rozszerzoną cenę, na podstawie zawartości zmiennych ilości i ceny jednostkowej. Ostatnia linia wyświetla na ekranie zawartość zmiennej Extendedprice. Pamiętaj, że komentarze w

rzeczywistości nie robią nic w programie podczas jego działania. Komentarze to tylko notatki do siebie o tym, co dzieje się w programie. Rysunek pokazuje, jak wszystko powinno wyglądać teraz.



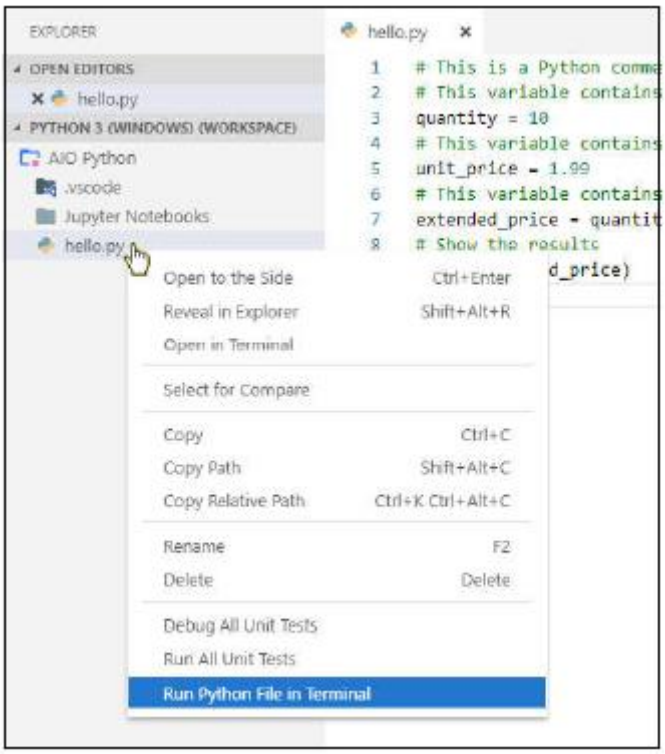
Jeśli popełniłeś jakieś błędy, możesz zobaczyć czerwone faliste linie w pobliżu miejsca błędu lub zielone faliste linie, jeśli jest to tylko błąd stylistyczny, taki jak dodatkowa spacja lub pominięty Enter na końcu wiersza. Jeśli masz błędy, upewnij się, że rozumiesz, że wpisując kod, musisz być dokładny. Nie możesz wpisać czegoś, co wygląda trochę jak to, co powinieneś wpisać. Wysyłając SMS-y do ludzi, możesz popełnić wszelkiego rodzaju błędy typograficzne, a Twój odbiorca może zwykle dowiedzieć się, co miałeś na myśli, na podstawie kontekstu wiadomości. Ale komputery nie mają oczu, mózgu ani żadnej koncepcji „kontekstu", więc generalnie nie będą działać poprawnie, jeśli w kodzie wystąpią błędy. Innymi słowy, jeśli kod jest błędny, nie zadziała po uruchomieniu. To takie proste, bez wyjątku.

Zapisywanie Twojej pracy

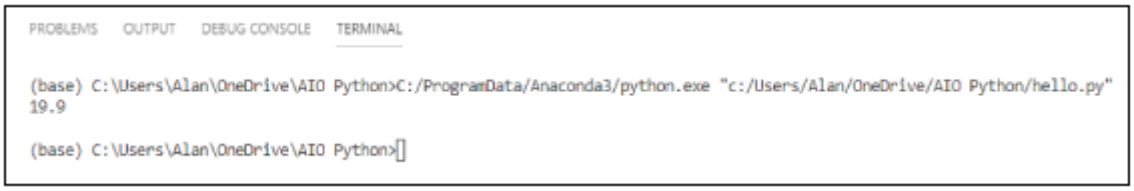
Wpisywanie kodu przypomina wpisywanie innych dokumentów na komputerze. Jeśli nie zapiszesz swojej pracy, możesz jej nie mieć następnym razem, gdy usiądziesz przy komputerze i zaczniesz jej szukać. Jeśli więc nie włączyłeś automatycznego zapisu w menu Plik, jak omówiono to w poprzednich rozdziałach, wybierz Plik, a następnie Zapisz.

Uruchamianie aplikacji Pythona w VS Code

Teraz możemy uruchomić aplikację i sprawdzić, czy działa. Prostym sposobem na to jest kliknięcie prawym przyciskiem myszy nazwy pliku hello.py na pasku Eksploratora i wybranie opcji Uruchamianie aplikacji w języku Python w programie VS Code jest łatwe. Wystarczy kliknąć prawym przyciskiem myszy nazwę pliku (hello.py, w tym przykładzie) i wybrać opcję Uruchom plik Python w terminalu, jak pokazano na rysunku



Jeśli twój kod jest wpisany poprawnie, powinieneś zobaczyć wynik 19,9 w terminalu, jak pokazano na rysunku



Wynik działania aplikacji pojawia się w oknie Terminala. Z pewnością nie wygląda jak typowa aplikacja na telefon, komputer Mac lub Windows. To dlatego, że dopiero zaczynasz i dlatego musisz zachować prostotę. Jedyną wskazówką, że aplikacja w ogóle działała, jest liczba 19,9 w wynikach. To jest wynik z print(extendedprice) w kodzie i jest to 19,9, ponieważ ilość (10) razy cena jednostkowa (1,99) wynosi 19,9. Załóżmy, że Twoja aplikacja musi obliczyć koszt 14 elementów, każdy po 26,99 USD. Czy możesz pomyśleć, jak to zrobić? Na pewno nie musiałbyś pisać zupełnie nowej aplikacji. Zamiast tego w kodzie, z którym teraz pracujesz, zmień wartość zmiennej ilościowej z 10 na 14. Zmień wartość zmiennej cena jednostkowa na 26,99 (pamiętaj, że w Twojej liczbie nie ma dolarów). Oto jak wygląda kod z tymi zmianami:

This is a Python comment in my first Python app.

This variable contains an integer

quantity = 14

This variable contains a float

unit_price = 26.99

This variable contains the the result of multiplying quantity times unit price

```
extended_price = quantity * unit_price

# Show some results on the screen.

print(extended_price)
```

Zapisz swoją pracę (chyba że włączyłeś już Autozapis, jak wspomniano wcześniej w tym rozdziale). Następnie uruchom tę aplikację ze zmianami, klikając prawym przyciskiem myszy i ponownie wybierając opcję Uruchom plik Python w terminalu . . . tak jak za pierwszym razem. Wyniki są znowu dość żarłoczne. Ale powinieneś zobaczyć poprawną odpowiedź, 377.85999999999996, w terminalu u dołu okna VS Code. Nie zaokrągla się do grosza i nawet nie wygląda na kwotę w dolarach. Ale zanim nauczysz się skakać o tyczce, musisz nauczyć się czołgać, więc na razie cieszmy się, że uruchomimy nasze aplikacje.

Czym jest składnia i dlaczego ma znaczenie

Jeśli wyszukasz składnię słowa w słowniku, prawdopodobnie zobaczysz, że jest ona zdefiniowana jako „układ słów i fraz w celu utworzenia dobrze sformułowanych zdań w języku". W językach programowania takich jak Python tak naprawdę nie ma czegoś takiego jak dobrze sformułowane zdanie. W przypadku kodu zazwyczaj odnosimy się do każdego wiersza kodu instrukcją lub wierszem (kodu). Ale kolejność słów jest ważna i istnieją „słowa" w tym sensie, że potrzebujesz spacji między każdym słowem, tak jak robisz to podczas pisania zwykłego tekstu w ten sposób. Składnia jest ważna w językach ludzkich, ponieważ kolejność ma duży wpływ na znaczenie. Na przykład porównaj te trzy krótkie zdania:

Mary pocałowała Johna.

John pocałował Mary.

Pocałowałem Mary John.

Wszystkie trzy zdania zawierają dokładnie te same słowa. Ale znaczenia są bardzo różne. Pierwsze dwa wyjaśniają, kto kogo pocałował, a ostatni jest trochę trudny do zinterpretowania. Mimo że wszystkie zawierają dokładnie te same słowa. Właściwa składnia w językach programowania jest tak samo ważna, jak w językach ludzkich - tym bardziej, że pod pewnymi względami, gdy popełnisz błąd mówiąc lub pisząc do kogoś, ta druga osoba może zwykle zrozumieć, co miałeś na myśli przez kontekst twoich słów. Ale komputery nie są aż tak inteligentne. Komputery nie mają mózgów, nie mogą „odgadnąć twojego rzeczywistego znaczenia" na podstawie kontekstu, a w rzeczywistości cała koncepcja „kontekstu" nie istnieje nawet dla komputerów. Tak więc składnia ma jeszcze większe znaczenie podczas pisania kodu, który mówi komputerowi, co ma robić, niż w przypadku ludzkich języków. Patrząc wstecz na nasz najwcześniejszy kod, zauważ, że wszystkie wiersze rzeczywistego kodu (nie komentarze, które zaczynają się od #) są zgodne ze składnią . . .

```
nazwa zmiennej = wartość
```

. . . gdzie nazwa_zmiennej to nazwa, którą wymyśliliśmy, a wartość to coś, co przechowujemy w tej zmiennej. Działa, ponieważ jest to właściwa składnia. Jeśli spróbujesz to zrobić w ten sposób. . .

```
wartość = nazwa zmiennej
```

. . . to nie zadziała. Na przykład jest to poprawny sposób przechowywania wartości 10 w a

zmienna o nazwie x:

```
x = 10
```

Może się wydawać, że możesz to zrobić również w ten sposób. . .

10 = x

. . . ale to nie zadziała w Pythonie. Jeśli uruchomisz aplikację z tą linijką, nic złego się nie stanie. . . niczego nie złamiesz. Ale otrzymasz komunikat o błędzie, który wygląda mniej więcej tak:

File ".../AIO Python/hello.py", line 10

10=x

^

SyntaxError: can't assign to literal

Część SyntaxError mówi, że Python nie wie, co zrobić z tym wierszem kodu, ponieważ nie przestrzegałeś właściwej składni. Aby naprawić błąd, po prostu przepisz linię jako

x = 10

Język Python składa się z wierszy kodu, z których każdy kończy się podziałem wiersza lub średnikiem. Innymi słowy, są to trzy linijki kodu Pythona:

first_name="Alan"

last_name="Simpson"

print(first_name,last_name)

Całkowicie do zaakceptowania byłby dla nas średnik zamiast łamania linii, jak poniżej.

first_name="Alan"; last_name="Simpson"

print(first_name,last_name)

Lub, jeśli wolisz:

first_name="Alan"; last_name="Simpson"; print(first_name,last_name)

Zwróć uwagę, że nie ma to żadnych zalet ani wad, jeśli chodzi o sposób działania kodu. W obu przypadkach kod działa dokładnie tak samo. Średniki są tylko alternatywą dla łamania linii, której możesz użyć, jeśli z jakiegoś powodu chcesz umieścić w edytorze dużo samego kodu w jednej fizycznej linii. Zwróć uwagę, że wszystkie nazwy naszych zmiennych są pisane małymi literami, a słowa są oddzielone podkreśleniem:

first_name

last_name

Jest to konwencja nazewnictwa używana w Pythonie - aby używać wszystkich małych liter w nazwach zmiennych ze słowami oddzielonymi podkreślnikami. Pamiętaj jednak, że konwencja to nie to samo, co reguła składni. Możesz nazwać zmienne jako

FirstName

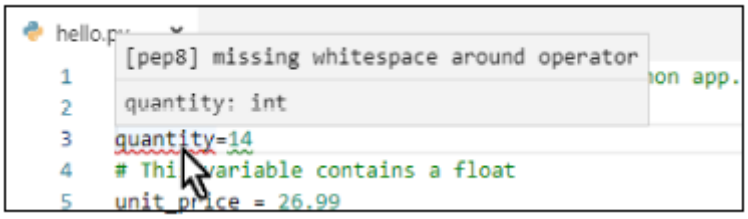
LastName

Jest to całkowicie w porządku w Pythonie i nie łamie żadnych zasad składni. Konwencja nazewnictwa ma na celu skłonienie programistów do przestrzegania pewnych podstawowych zasad stylistycznych,

które czynią kod bardziej czytelnym dla innych programów, co jest szczególnie ważne podczas pracy w zespołach programistycznych lub grupach. Do tej pory patrzyliśmy na wiersze kodu. Istnieją również bloki kodu lub bloki kodu, w których co najmniej dwie linie kodu w jakiś sposób współpracują ze sobą. Oto przykład:

```
x = 10

if x == 0:

print("x is zero")

else:

print("x is ",x)

print("All done")
```

== (dwa znaki równości) oznacza w Pythonie „równe” i służy do porównywania wartości, aby sprawdzić, czy są równe. Różni się to od po prostu = (jeden znak równości), który jest operatorem przypisania do przypisywania zmiennych. Pierwsza linia, x=10, to tylko linia kodu. Następnie if x == 0 sprawdza, czy zmienna x zawiera liczbę zero. Jeśli x zawiera zero, to wcięta linia (print("x to zero")) jest wykonywana i to właśnie widzisz na ekranie. Jeśli jednak x nie zawiera zera, to ta wcięta linia jest pomijana, a instrukcja else: Wcięta linia pod else: print("x to ",x) jest wykonywana, ale tylko jeśli x nie zawiera zera. Ostatnia linia, print("Wszystko zrobione!"), wykonuje się bez względu na wszystko, ponieważ jest bez wcięć. Innymi słowy, wcięcia mają duże znaczenie w Pythonie, ponieważ tylko jeden z wciętych powyżej wierszy zostanie wykonany w zależności od wartości w x. W miarę postępów w kodzie omówimy specyfikę używania wcięć w kodzie książki. Na razie pamiętaj tylko, że składnia i wcięcia są ważne w Pythonie, więc musisz pisać ostrożnie podczas pisania kodu. Następnym razem, gdy uruchomisz aplikację, uruchomi się i nie otrzymasz linii błędu składni. Jeśli masz włączone linting i PEP 8 w ustawieniach Workspace, jak opisano w rozdziale 3, możesz zobaczyć czerwone faliste podkreślenia lub zielone faliste podkreślenia kodu, który wydaje się być w porządku. Dotknięcie wskaźnikiem myszy takiego podkreślenia zwykle powoduje wyświetlenie komunikatu przy wskaźniku myszy wskazującego, na czym polega problem, jak na rysunku



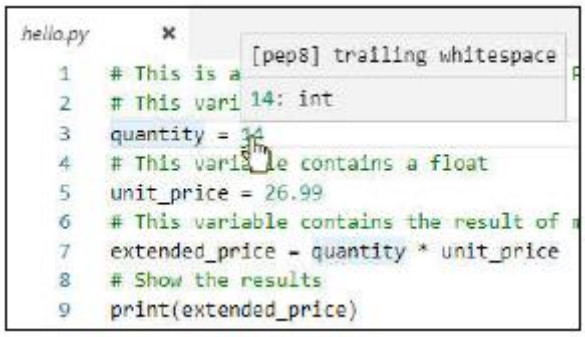
Pierwsza część wiadomości brzmi:

[pep8] brak spacji wokół operatora

To mówi ci, że ten konkretny błąd jest związany ze składnią Pep 8, która mówi, że powinieneś umieścić białe znaki wokół operatorów. Druga część mówi tylko, że zmienna o nazwie ilość zawiera liczbę całkowitą (int), liczbę całkowitą. To nie błąd, tylko informacja.

Aby naprawić błąd, wstaw spację wokół znaku =. Innymi słowy, użyj spacji na klawiaturze, aby umieścić spację przed znakiem =, a następnie kolejną spację po znaku =. Załóżmy teraz, że naprawiłeś ten problem z odstępami, ale nadal widzisz zielone faliste podkreślenie pod 14. O co chodzi? Cóż, aby się

dowiedzieć, po prostu kliknij zieloną falistą linię podkreślenia i pozostaw wskaźnik myszy w tym miejscu, aż zobaczysz wyjaśnienie, jak na rysunku



Tym razem górna wiadomość pokazuje [pep8] końcowe białe znaki na górze i 14: int na dole. Ponownie, dolna część jest tylko informacyjna, mówiąc, że 14 jest przechowywana jako liczba całkowita. Błąd to końcowe białe znaki. Oznacza to, że z jakiegoś powodu po 14 w tym wierszu jest spacja. Nie możesz tego zobaczyć, bo to tylko przestrzeń. Ale jeśli klikniesz koniec tej linii i naciśniesz Backspace, aż kursor znajdzie się dokładnie do 4 na 14, to usuniesz wszystkie końcowe spacje, a to powinno usunąć ten błąd. Pamiętaj, że nie wszystkie czerwone błędy są takie same i nie wszystkie zielone błędy są takie same. Więc nie rób żadnych założeń. Ogólnie rzecz biorąc, czerwone błędy prawdopodobnie uniemożliwiają prawidłowe działanie aplikacji, podczas gdy zielone błędy to tylko błędy stylistyczne. Ale ty nie będziesz wiedział konkretnie, na czym polega błąd, chyba że klikniesz faliste podkreślenie i pozostawisz tam wskaźnik myszy, aż zobaczysz komunikat. A błąd nie zniknie, dopóki nie podejmiesz wszelkich działań wymaganych do naprawienia tego błędu. Jeśli błędy PeP 8 naprawdę doprowadzają Cię do szału podczas próby uczenia się, pamiętaj, że możesz przejść do ustawień obszaru roboczego VS Code i wyłączyć ustawienie Python Linting -> Pep8Enabled, aby nie sprawdzać kodu tak agresywnie.

Składanie kodu razem

Ćwiczenia, które właśnie ukończyłeś, wyjaśniają, jak pisać, zapisywać, uruchamiać, zmieniać i ponownie zapisywać, a po wprowadzeniu zmian ponownie uruchomić aplikację. Wszystkie te różne rzeczy określają, co będziesz robić z dowolnym rodzajem oprogramowania w dowolnym języku. Więc chcesz ćwiczyć je często, aż staną się twoją drugą naturą. Ale nie martw się, to nie znaczy, że musisz powtarzać tą jedną część w kółko, aby to zrozumieć. Będziesz korzystać z tych wszystkich umiejętności, przechodząc od początkującego do wytrawnego programisty Pythona XXI wieku.