Praca z plikami zewnętrznymi

Prawie wszystko, co jest przechowywane na twoim komputerze, czy to dokument, program, film, zdjęcie. . . cokolwiek, jest przechowywane w pliku. Większość plików to uporządkowane foldery informacyjne (zwane także katalogami). Na komputerze Mac możesz używać Findera do przeglądania folderów i plików. W systemie Windows używasz Eksploratora plików lub Eksploratora Windows (ale nie Internet Explorera) do przeglądania folderów i plików. Python oferuje wiele narzędzi do tworzenia, odczytywania i zapisywania w wielu różnych rodzajach plików. W tym rozdziale nauczysz się wszystkich najważniejszych umiejętności pracy z plikami przy użyciu kodu Pythona.

Zrozumienie plików tekstowych i binarnych

Istnieją zasadniczo dwa rodzaje plików:

Pliki tekstowe: Pliki tekstowe zawierają zwykłe znaki tekstowe. Gdy otworzysz je w edytorze tekstu, pokażą treść czytelną dla człowieka. Tekst może nie być w języku, który znasz lub rozumiesz, ale zobaczysz głównie normalne znaki, które możesz wpisać na dowolnej klawiaturze.

Pliki binarne: plik binarny przechowuje informacje w bajtach, które nie są tak czytelne dla człowieka. Nie zalecamy tego, ale jeśli otworzysz plik binarny w edytorze tekstu, to, co zobaczysz, może przypominać rysunek

Jeśli kiedykolwiek otworzysz plik binarny w edytorze tekstu i zobaczysz ten bełkot, nie panikuj. Po prostu zamknij plik lub program i wybierz opcję Nie, jeśli zostaniesz poproszony o zapisanie. Plik będzie w porządku, o ile go nie zapiszesz. Podobnie jak w przypadku każdego kodu w języku Python, do napisania kodu w języku Python można użyć notatnika Jupyter, VS Code lub praktycznie dowolnego edytora kodu. W tym rozdziale używamy VS Code po prostu dlatego, że jego pasek Eksploratora (po lewej stronie, gdy jest otwarty) wyświetla zawartość folderu, w którym aktualnie pracujesz.

Otwieranie i zamykanie plików

Aby otworzyć plik z poziomu aplikacji Pythona, użyj składni:

open(filename.ext[,mode])

Zamień filename.ext na nazwę pliku, który chcesz otworzyć. Jeśli plik nie znajduje się w tym samym katalogu co kod Pythona, musisz określić ścieżkę do pliku. Używaj ukośników, nawet jeśli pracujesz w systemie Windows. Na przykład, jeśli plik, który chcesz otworzyć, to foo.txt na pulpicie, a nazwa Twojego konta użytkownika to Alan, użyjesz ścieżki C:/Users/Alan/Desktop/foo.txt zamiast bardziej powszechnej Składnia systemu Windows z ukośnikami odwrotnymi, na przykład C:\Users\Alan\Desktop\foo.txt.

[,mode] jest opcjonalny (jak wskazano w nawiasach kwadratowych). Użyj go, aby określić, jaki rodzaj dostępu ma mieć Twoja aplikacja, używając następujących jednoznakowych skrótów:

r: (odczyt): umożliwia Pythonowi otwieranie pliku, ale nie wprowadzanie żadnych zmian. Jest to ustawienie domyślne, jeśli nie określisz trybu. Jeśli plik nie istnieje, Python zgłasza wyjątek FileNotFoundError.

r+: (odczyt/zapis): umożliwia Pythonowi odczytywanie i zapisywanie w pliku.

a: (Dołącz): Otwiera plik i umożliwia Pythonowi dodawanie nowej zawartości na końcu pliku, ale nie zmienia istniejącej zawartości. Jeśli plik nie istnieje, ten tryb tworzy plik.

w: (Zapis): otwiera plik i umożliwia Pythonowi wprowadzanie zmian w pliku. Tworzy plik, jeśli nie istnieje.

x: (Utwórz): Tworzy plik, jeśli jeszcze nie istnieje. Jeśli plik istnieje, zgłasza wyjątek FileExistsError.

Możesz także określić typ pliku, który otwierasz (lub tworzysz). Jeśli już określiłeś jeden z powyższych trybów, po prostu dodaj go jako kolejną literę. Jeśli użyjesz tylko jednej z poniższych liter, plik otworzy się w trybie odczytu.

t: (Tekst): Otwórz jako plik tekstowy, czytaj i pisz tekst.

b: (binarny): Otwórz jako plik binarny, odczytuj i zapisuj bajty.

Zasadniczo istnieją dwa sposoby korzystania z metody otwartej. Za pomocą jednej składni przypisujesz nazwę zmiennej do pliku i używasz tej nazwy zmiennej w późniejszym kodzie, aby odnieść się do pliku. Ta składnia to:

var = open(nazwa pliku.ext[,tryb])

Zamień var na wybraną przez siebie nazwę (chociaż w Pythonie bardzo często używa się tylko litery f jako nazwy). Chociaż ta metoda działa, nie jest idealna, ponieważ po otwarciu plik pozostaje otwarty, dopóki nie zamkniesz go za pomocą metody close(). Zapomnienie o zamknięciu plików może spowodować problem z otwarciem zbyt wielu plików w tym samym czasie, co może spowodować uszkodzenie zawartości pliku lub spowodować zgłaszanie wyjątków i awarię aplikacji. Po otwarciu pliku istnieje kilka sposobów uzyskiwania dostępu do jego zawartości, co omówimy w dalszej części tego rozdziału. Na razie po prostu kopiujemy wszystko, co jest w pliku do zmiennej o nazwie filecontents w Pythonie, a następnie wyświetlamy tę zawartość za pomocą prostej funkcji print(). Aby więc otworzyć plik quotes.txt, przeczytać całą jego zawartość i wyświetlić ją na ekranie, użyj tego kodu:

f = open('cytaty.txt')

zawartość pliku = f.odczyt()

print (zawartość pliku)

Dzięki tej metodzie plik pozostaje otwarty, dopóki go nie zamkniesz, używając nazwy zmiennej pliku i metody .close() w następujący sposób:

f.close()

Ważne jest, aby Twoje aplikacje zamykały wszystkie pliki, których nie potrzebują już otwierać. Niezastosowanie się do tego umożliwia gromadzenie się programów obsługi otwartych plików, co może ostatecznie spowodować zgłoszenie wyjątku i awarię aplikacji, a być może nawet uszkodzenie niektórych otwartych plików po drodze. Wracając jednak do samego otwierania pliku: innym sposobem na to jest użycie menedżera kontekstu lub kodowanie kontekstowe. Ta metoda zaczyna się od słowa z. Nadal przypisujesz nazwę zmiennej. Ale robisz to pod koniec linii. Ostatnią rzeczą w wierszu jest dwukropek, który oznacza początek bloku with. Cały poniższy kod z wcięciem, który z założenia jest odpowiedni dla kontekstu otwartego pliku (jak kod z wcięciem wewnątrz pętli). Na koniec nie musisz specjalnie zamykać pliku; Python robi to automatycznie:

```
# ---------------- Contextual syntax

with open('quotes.txt') as f:

filecontents = f.read()

print(filecontents)

# The unindented line below is outside the with... block;

print('File is closed: ', f.closed)
```

Poniższy kod przedstawia pojedynczą aplikację, która otwiera quotes.txt, odczytuje i wyświetla jego zawartość, a następnie zamyka plik. W pierwszej metodzie musisz specjalnie użyć .close() do zamknięcia pliku. W drugim przypadku plik zamyka się automatycznie, więc nie jest wymagana funkcja .close():

```
# - Basic syntax to open, read, and display file contents.

f = open('quotes.txt')

filecontents = f.read()

print(filecontents)

# Returns True if the file is closed, otherwise alse.

print('File is closed: ', f.closed)

# Closes the file.

f.close() #Close the file.

print() # Print a blank line.

# ---------------- Contextual syntax

with open('quotes.txt') as f:

        filecontents = f.read()

        print(filecontents)
```

```
# The unindented line below is outside the with... block;

print('File is closed: ', f.closed)
```

Dane wyjściowe tej aplikacji są następujące. Na końcu pierwszego wyjścia .closed ma wartość False, ponieważ jest testowane przed zamknięciem pliku przez metodę close(). Na końcu drugiego wyjścia .closed ma wartość True, bez wykonywania .close(), ponieważ pozostawienie kodu wciętego pod linią with: powoduje automatyczne zamknięcie pliku.

Miałem wspaniały wieczór, ale to nie było to.

Groucho Marx

Różnica między głupotą a geniuszem polega na tym, że geniusz ma swoje granice.

Albert Einstein

Wszyscy jesteśmy tu na ziemi, aby pomagać innym; po co tu są inni,

nie mam pojęcia.

W.H. Auden

Kończenie zdania przyimkiem to coś, czego nie będę umieszczał.

Winston Churchill

Plik jest zamknięty: Fałsz

Miałem wspaniały wieczór, ale to nie było to.

Groucho Marx

Różnica między głupotą a geniuszem polega na tym, że geniusz ma swoje granice.

Albert Einstein

Wszyscy jesteśmy tu na ziemi, aby pomagać innym; po co tu są inni,

Nie mam pojęcia.

W.H. Auden

Kończenie zdania przyimkiem to coś, czego nie będę umieszczał.

Winston Churchill

Plik jest zamknięty: Prawda

Przez resztę tej części trzymamy się składni kontekstowej, ponieważ jest to ogólnie preferowana i zalecana składnia oraz dobry nawyk, którego należy się nauczyć od samego początku. Poprzedni przykład działa dobrze, ponieważ cytaty.txt to naprawdę prosty plik tekstowy, który zawiera tylko znaki ASCII - rodzaje liter, cyfr i znaków interpunkcyjnych, które można wpisywać ze standardowej klawiatury języka angielskiego.

```
with open('happy_pickle.jpg') as f:
```

filecontents = f.read()

print(filecontents)

Próba uruchomienia tego kodu powoduje następujący błąd:

UnicodeDecodeError: 'charmap' codec can't decode byte 0x90 in position 40:

character maps to <undefined>

To nie jest najbardziej pomocna wiadomość na świecie. Załóżmy, że próbujesz otworzyć names.txt, który (jak można by przypuszczać) jest plikiem tekstowym, takim jak quotes.txt, używając tego kodu:

with open('names.txt') as f:

filecontents = f.read()

print(filecontents)

Uruchamiasz ten kod i ponownie pojawia się dziwny komunikat o błędzie, taki jak ten:

UnicodeDecodeError: 'charmap' codec can't decode byte 0x81 in position 45:

character maps to <undefined>

Więc co tu się do cholery dzieje?

Pierwszy problem wynika z faktu, że plik jest plikiem .jpg, obrazem graficznym, co oznacza, że jest to plik binarny, a nie plik tekstowy. Więc aby otworzyć ten, potrzebujesz b w trybie. Lub użyj rb, co oznacza odczyt binarny, tak jak poniżej:

with open('happy_pickle.jpg', 'rb') as f:

filecontents = f.read()

print(filecontents)

Uruchomienie tego kodu nie generuje błędu. Ale to, co pokazuje, nie przypomina rzeczywistego obrazu. Dane wyjściowe tego kodu to wiele rzeczy, które wyglądają mniej więcej tak:

\x07~}\xba\xe7\xd2\x8c\x00\x0e|\xbd\xa8\x12l \xca\xf7\xae\xa5\x9e^\x8d\x89

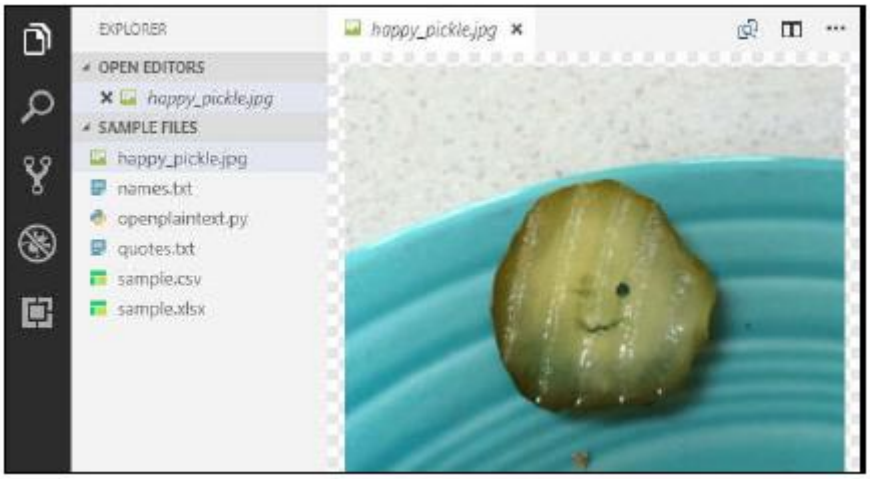
\x7f\xde\xb4f>\x98\xc7\xfc\xcf46d\xcf\x1c\xd0\xa6\x98m$\xb6(U\x8c\xa6\x83

\x19\x17\xa6>\xe6\x94\x96|g\'4\xab\xdd\xb8\xc8=\xa9[\x8b\xcc`\x0e8\xa3

\xb0;\xc6\xe6\xbb(I.\xa3\xda\x91\xb8\xbd\xf2\x97\xdf\xc1\xf4\xefI\xcdy

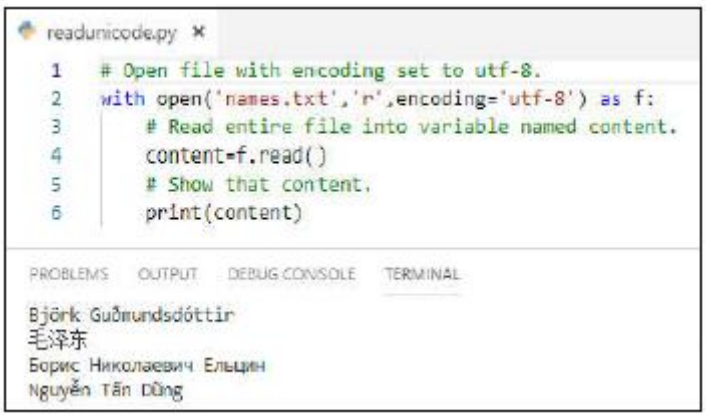
\x97d\x1e`;\xf64\x94\xd7\x03

Jeśli otworzymy happy_pickle.jpg w aplikacji graficznej lub w VS Code, nie będzie to wyglądało jak ten bełkot. Zamiast tego wygląda jak rysunek



Dlaczego więc wygląda to tak pomieszane w Pythonie? To dlatego, że print() pokazuje tylko nieprzetworzone bajty, które składają się na plik. Nie ma wyboru, ponieważ nie jest aplikacją graficzną. To nie jest problem ani problem, po prostu nie jest to dobry sposób na pracę z plikiem .jpg w tej chwili. Problem z plikami names.txt jest inny. Ten plik jest plikiem tekstowym (.txt), podobnie jak quotes.txt. Ale jeśli otworzysz go i spojrzysz na jego zawartość, jak na rysunku , zauważysz, że zawiera wiele niezwykłych znaków, których normalnie nie widzisz w ASCII, liczby z dnia na dzień, litery i znaki interpunkcyjne widoczne na klawiaturze.

Ten plik names.txt jest rzeczywiście plikiem tekstowym, ale wszystkie te fantazyjnie wyglądające znaki wskazują, że nie jest to zwykły plik tekstowy ASCII. Bardziej prawdopodobne jest, że jest to plik UTF-8, który jest w zasadzie plikiem tekstowym, który wykorzystuje więcej niż tylko standardowe znaki tekstowe ASCII. Aby otworzyć ten plik, musisz powiedzieć Pythonowi, aby „oczekiwał” znaków UTF-8, używając encoding='utf-8' w instrukcji open(), jak na rysunku . Rysunek przedstawia wyniki otwarcia pliku names.txt jako pliku tekstowego do odczytu z dodatkiem encoding =. Wyjście z Pythona dokładnie pasuje do tego, co znajduje się w pliku names.txt.



Nie wszystkie okna terminala w programie VS Code poprawnie wyświetlają znaki Unicode, które można zastąpić ogólnymi symbolami znaku zapytania na ekranie. Ale nie martw się o to, w prawdziwym życiu nie będziesz się przejmować danymi wyjściowymi w oknie Terminala. Tutaj liczy się tylko to, że jesteś w stanie otworzyć plik bez zgłaszania wyjątku. Jeśli chodzi o otwieranie plików, należy pamiętać o trzech rzeczach:

- Jeśli jest to zwykły tekst (ASCII), wystarczy użyć jako trybu r lub nic.
- Jeśli jest to plik binarny, musisz określić b w trybie.
- Jeśli jest to plik tekstowy z wymyślnymi znakami, najprawdopodobniej będziesz musiał otworzyć go jako plik tekstowy, ale z kodowaniem ustawionym na utf-8 w instrukcji open().

CZYM JEST UTF-8?

UTF-8 jest skrótem od Unicode Transformation Format, 8-bit i jest standardowym sposobem przedstawiania liter i cyfr na komputerach. Oryginalny zestaw znaków ASCII, który zawiera głównie wielkie i małe litery, cyfry i znaki interpunkcyjne, działał dobrze we wczesnych latach komputerów. Ale kiedy zaczniesz wprowadzać inne języki do miksu, te znaki po prostu nie wystarczą. Od tego czasu zaproponowano i przyjęto wiele różnych standardów postępowania z innymi językami. Spośród nich UTF-8 stale rośnie w użyciu, podczas gdy większość innych spada. Obecnie UTF-8 jest prawie standardem dla wszystkich rzeczy związanych z Internetem, więc jest dobrym wyborem, jeśli kiedykolwiek będziesz musiał wybrać zestaw znaków dla jakiegoś projektu. Jeśli szukasz więcej informacji historycznych lub technicznych na temat UTF-8, spójrz na te strony internetowe:

https://www.w3.org/International/questions/qa-what-is-encoding

https://pythonconquerstheuniverse.wordpress.com/2010/05/30/unicode-beginnuction-for-dummies-ers-introdmade-simple/

https://www.joelonsoftware.com/2003/10/08/the-absolute-minimum-everysoftware-developer-absolutely-positively-must-know-about-unicodeand-character-sets-no-excuses/

Jeśli naprawdę utkniesz podczas próby otwarcia pliku, który powinien być w formacie UTF-8, ale nie współpracuje, Google przekonwertuje plik na kodowanie utf-8. Następnie poszukaj strony internetowej lub aplikacji, która będzie działać z Twoim systemem operacyjnym, aby dokonać konwersji.

Odczytywanie zawartości pliku

Wcześniej w tym rozdziale widziałeś, jak możesz przeczytać całą zawartość otwartego pliku za pomocą .read(). Ale to nie jedyny sposób, aby to zrobić. Właściwie masz trzy możliwości:

read([rozmiar]): odczytuje cały plik, jeśli pozostawisz puste nawiasy. Jeśli określisz rozmiar w nawiasach, odczytana zostanie taka liczba znaków (dla pliku tekstowego) lub tyle bajtów (dla pliku binarnego).

readline(): Odczytuje jedną linię treści z pliku tekstowego (linia kończy się tam, gdzie występuje znak nowej linii).

readlines(): Wczytuje wszystkie wiersze pliku tekstowego do listy.

Ludzie nie wpisują plików binarnych, więc wszelkie znaki nowej linii, które się tam znajdują, byłyby arbitralne. Dlatego readline() i readlines() są użyteczne tylko dla plików tekstowych. Obie metody read() i readline() odczytują jednocześnie cały plik. Jedyną rzeczywistą różnicą jest to, że read wczytuje to jako jeden duży fragment danych, podczas gdy readlines() odczytuje to po jednym wierszu na raz i zapisuje każdy wiersz jako element na liście. Na przykład poniższy kod otwiera cytaty.txt, wczytuje całą zawartość, a następnie ją wyświetla

with open('quotes.txt') as f:

Read in entire file

content = f.read()

print(content)

Zmienna content kończy się przechowywaniem kopii wszystkiego, co znajduje się w pliku CSV. Wydrukowanie tej zmiennej pokazuje zawartość. Jest podzielony na wiele wierszy dokładnie tak, jak oryginalny plik, ponieważ znak nowej linii na końcu każdego wiersza w pliku również rozpoczyna nowy wiersz na ekranie podczas drukowania zawartości. Oto ten sam kod używający readlines() zamiast read:

with open('quotes.txt') as f:

content = f.readlines()

print(content)

Dane wyjściowe z tego kodu to

[„Miałem wspaniały wieczór, ale to nie było to.\n”, „Groucho Marx\n”, „Różnica między głupotą a geniuszem polega na tym, że geniusz ma swoje granice.\n”, „Albert Einstein \n”, „Wszyscy jesteśmy tu na ziemi, aby pomagać innym; nie mam pojęcia, po co tu są inni.\n”, „W. H. Auden\n”, „Zakończenie zdania przyimkiem to coś, czego nie wstawię.\n”, „Winston Churchill\n”]

Nawiasy kwadratowe otaczające dane wyjściowe informują, że jest to lista. Każda pozycja na liście jest ujęta w cudzysłowy i oddzielona przecinkami. \n na końcu każdego elementu to znak nowej linii, który kończy wiersz w pliku. W przeciwieństwie do readlines() (liczba mnoga), readline() czyta tylko jedną linię z pliku. Wiersz rozciąga się od początku pliku do tuż za pierwszym znakiem nowej linii. Wykonanie kolejnego readline() odczytuje następną linię w pliku i tak dalej. Załóżmy na przykład, że uruchamiasz ten kod:

with open('quotes.txt') as f:

content = f.readline()

print(content)

Wyjście to:

Miałem wspaniały wieczór, ale to nie było to

Wykonanie kolejnego readline() po tym spowoduje odczytanie następnej linii. Jak możesz się domyślić, jeśli chodzi o readline() i readlines(), prawdopodobnie będziesz chciał użyć pętli, aby uzyskać dostęp do wszystkich danych w sposób zapewniający większą kontrolę.

Zapętlanie pliku

Możesz przeglądać plik w pętli, używając funkcji readlines() lub readline(). Metoda readlines() zawsze odczytuje plik jako całość. Co oznacza, że jeśli plik jest bardzo duży, może zabraknąć pamięci (RAM) przed wczytaniem pliku. Ale jeśli znasz rozmiar pliku i jest on stosunkowo mały (może kilkaset wierszy danych lub mniej), readlines() to szybki sposób na uzyskanie wszystkich danych. Te dane będą na liście. Będziesz więc przechodził przez listę, a nie przez plik. Możesz także przeglądać pliki binarne w pętli, ale nie mają one wierszy tekstu, jak pliki tekstowe. Więc te, które czytasz w „kawałkach", jak zobaczysz na końcu tej części.

Zapętlanie z readlines()

Kiedy czytasz plik za pomocą readlines(), czytasz cały plik za jednym zamachem jako listę. Więc tak naprawdę nie przechodzisz przez pętlę po jednym rzędzie na raz. Zamiast tego przeglądasz listę elementów, które readlines() przechowuje w pamięci. Kod, który to zrobi, wygląda następująco:

```
with open('quotes.txt') as f:

# Reads in all lines first, then loops through.

for one_line in f.readlines():

print(one_line)
```

Jeśli uruchomisz ten kod, dane wyjściowe będą z podwójnymi odstępami, ponieważ każdy element listy kończy się znakiem nowej linii, a następnie print zawsze dodaje własną nową linię przy każdym przejściu przez pętlę. Jeśli chcesz zachować pojedyncze odstępy, dodaj end='' do instrukcji print (upewnij się, że używasz dwóch pojedynczych lub podwójnych cudzysłowów bez niczego pomiędzy po znaku =). Oto przykład:

```
with open('quotes.txt') as f:

# Reads in all lines first, then loops through.

for one_line in f.readlines():

print(one_line, end='')
```

Dane wyjściowe z tego kodu to:

Miałem wspaniały wieczór, ale to nie było to.

Groucho Marx

Różnica między głupotą a geniuszem polega na tym, że geniusz ma swoje granice.

Albert Einstein

Wszyscy jesteśmy tu na ziemi, aby pomagać innym; po co tu są inni,

Nie mam pojęcia.

W.H. Auden

Kończenie zdania przyimkiem to coś, czego nie będę umieszczał.

Winston Churchill

Załóżmy, że całkiem nieźle sobie z tym radzisz, z tym wyjątkiem, że jeśli linia do wydrukowania jest nazwą, chcesz wciąć nazwę spacją i umieścić pod nią dodatkową pustą linię. Jak mogłeś to zrobić? Cóż, Python ma wbudowaną funkcję enumerate(), która używana z listą zlicza liczbę przejść przez pętlę, zaczynając od zera. Więc zamiast pętli for: pokazanej w poprzednim przykładzie, zapisujesz ją jak dla one_line w enumerate(f.readlines()):. Przy każdym przejściu przez pętlę one_line[0] zawiera numer tej linii, podczas gdy one_line[1] zawiera jej zawartość. . . tekst wiersza. Przy każdym przejściu przez pętlę możesz zobaczyć, czy licznik jest liczbą parzystą (ta liczba % 2 będzie równa zero dla liczb parzystych, ponieważ % zwraca resztę po dzieleniu). Więc możesz napisać kod w ten sposób:

with open('quotes.txt') as f:

Reads in all lines first, then loops through.

Count each line starting at zero.

for one_line in enumerate(f.readlines()):

If counter is even number, print with no extra newline

if one_line[0] % 2 == 0:

print(one_line[1], end='')

Otherwise print a couple spaces and an extra newline.

else:

print(' ' one_line[1])

Dane wyjściowe z tego będą następujące:

Miałem wspaniały wieczór, ale to nie było to.

Groucho Marx

Różnica między głupotą a geniuszem polega na tym, że geniusz ma swoje granice.

Albert Einstein

Wszyscy jesteśmy tu na ziemi, aby pomagać innym; po co tu są inni,

Nie mam pojęcia.

W.H. Auden

Kończenie zdania przyimkiem to coś, czego nie będę umieszczał.

Winston Churchill

Zapętlanie z readline()

Jeśli nie masz pewności co do rozmiaru czytanego pliku lub ilości pamięci RAM w komputerze, na którym działa Twoja aplikacja, użycie readlines() do wczytania całego pliku może być ryzykowne. Ponieważ jeśli nie ma wystarczającej ilości pamięci do przechowywania całego pliku, aplikacja ulegnie awarii, gdy zabraknie pamięci. Aby zachować bezpieczeństwo, możesz przeglądać plik w pętli, po jednym wierszu na raz, tak aby w danej chwili w pamięci znajdował się tylko jeden wiersz zawartości pliku. Aby skorzystać z tej metody, możesz otworzyć plik, przeczytać jedną linię i umieścić ją w zmiennej. Następnie przeglądaj plik w pętli, dopóki (dopóki) zmienna nie jest pusta. Ponieważ każda linia w pliku zawiera jakiś tekst, zmienna nie będzie pusta, dopóki nie zostanie odczytana ostatnia linia. Oto kod tego podejścia do pętli:

with open('quotes.txt') as f:

one_line = f.readline()

while one_line:

print(one_line, end='')

one_line = f.readline()

W przypadku większych plików byłoby to dobre rozwiązanie, ponieważ w żadnym momencie nie czyta się całego pliku. Jedynym niebezpieczeństwem jest zapomnienie o wykonaniu .readline() wewnątrz pętli, aby przejść do następnego wskaźnika. Niezastosowanie się do tego tworzy nieskończoną pętlę, która w kółko drukuje pierwszą linię. Jeśli kiedykolwiek znajdziesz się w takiej sytuacji, naciśnięcie Ctrl C w oknie terminala, w którym uruchamiamy kod, zatrzyma pętlę. A co jeśli chcesz zrobić coś takiego jak w readlines() gdzie robisz wcięcie i drukujesz dodatkową pustą linię po imionach ludzi? W tym przykładzie naprawdę nie możesz użyć enumerate z pętlą while. Ale możesz sam ustawić prosty licznik, zaczynając od 1, jeśli chcesz, i zwiększać go o 1 przy każdym przejściu przez pętlę. Wcięcie i dodatkowe miejsce na liniach parzystych w ten sposób:

Store a number to use as a loop counter.

counter = 1

Open the file.

with open('quotes.txt') as f:

Read one line from the file.

one_line = f.readline()

As long as there are lines to read...

while one_line:

If the counter is an even number, print a couple spaces.

if counter % 2 == 0:

print(' ' one_line)

Otherwise print with no newline at the end.

```
else:

print(one_line,end='')

# Increment the counter

counter = 1

# Read the next line.

one_line = f.readline()
```

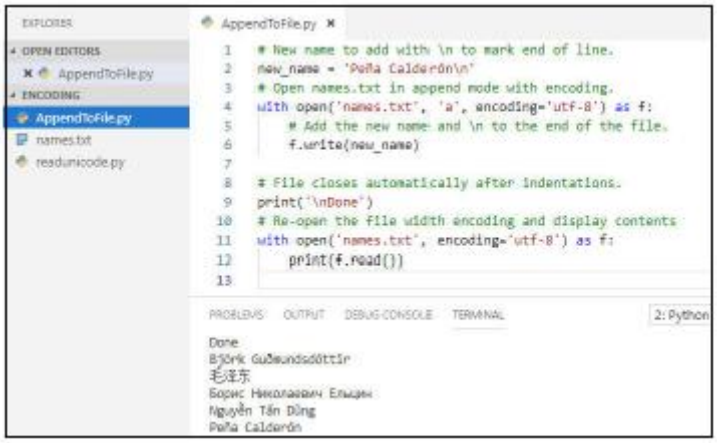
Dane wyjściowe z tej pętli są takie same, jak w przypadku drugiej pętli readlines(), w której nazwisko każdego autora jest wcięte, a po nim następuje dodatkowa pusta linia spowodowana użyciem print() bez end=''.

Dołączanie a nadpisywanie plików

Za każdym razem, gdy pracujesz z plikami, ważne jest, aby zrozumieć różnicę między zapisem a dołączaniem. Jeśli plik zawiera już informacje i otworzysz go w trybie zapisu, a następnie napiszesz w nim więcej, twoja nowa zawartość w rzeczywistości zastąpi wszystko, co już znajduje się w pliku. Nie ma na to możliwości cofnięcia. Więc jeśli zawartość pliku jest ważna, chcesz się upewnić, że nie popełnisz tego błędu. Aby dodać zawartość do pliku, otwórz plik w trybie dołączania (a), a następnie użyj polecenia .write, aby napisać do pliku. Jako działający przykład załóżmy, że chcesz dodać nazwisko Peña Calderón do pliku names.txt użytego w poprzedniej sekcji. Ta nazwa, jak również nazwy, które już znajdują się w tym pliku, zawierają znaki specjalne spoza alfabetu angielskiego, co oznacza, że należy ustawić kodowanie na UTF-8. Ponadto, jeśli chcesz, aby każda nazwa w pliku znajdowała się w osobnym wierszu, powinieneś dodać \n (nowy wiersz) na końcu dodawanej nazwy. Więc twój kod powinien wyglądać tak:

```
# New name to add with \n to mark end of line.

new_name = 'Peña Calderón\n'

# Open names.txt in append mode with encoding.

with open('names.txt', 'a', encoding='utf-8') as f:

f.write(new_name)
```

Aby sprawdzić, czy zadziałało, rozpocznij nowy blok kodu, bez wcięć, a więc bez nazw. txt Plik zamyka się automatycznie. Następnie otwórz ten sam plik w trybie odczytu (r) i przejrzyj jego zawartość. Na rysunku pokazano cały kod służący do dodania nowej nazwy oraz kod do wyświetlenia pliku names.txt po dodaniu tej nazwy.



Wpisywanie znaków specjalnych, takich jak ñ i ó, zwykle wiąże się z przytrzymaniem klawisza Alt i wpisaniem 3 lub 4 cyfr; na przykład Alt 164 dla ñ lub Alt 0243 dla ó. Dokładny sposób, w jaki to zrobisz, zależy od używanego systemu operacyjnego i edytora. Ale z reguły możesz wyszukać w Google wyrażenie takie jak wpisz tyldę n w systemie Windows lub wpisz o z akcentem na komputerze Mac itd., aby dokładnie dowiedzieć się, co należy zrobić, aby wpisać znak specjalny.

Używanie tell() do określenia położenia wskaźnika

Za każdym razem, gdy przeglądasz plik w pętli, jego zawartość jest odczytywana od góry do dołu, od lewej do prawej. Python utrzymuje niewidoczny wskaźnik, aby śledzić, gdzie się on znajduje w pliku. Kiedy czytasz plik tekstowy za pomocą readline(), jest to zawsze pozycja znaku następnego wiersza w pliku. Jeśli wszystko, co do tej pory zrobiłeś, to otwarcie pliku, pozycja znaku wyniesie zero, czyli sam początek pliku. Za każdym razem, gdy wykonujesz readline(), wskaźnik przesuwa się na początek następnego wiersza. Oto kod do zilustrowania;

with open('names.txt', encoding='utf-8') as f:

Read first line to get started.

print(f.tell())

one_line = f.readline()

Keep reading one line at a time until there are no more.

while one_line:

print(one_line[:-1], f.tell())

one_line = f.readline()

0

Björk Guðmundsdóttir 25

问题 36

Борис Николаевич Ельцин 82

Nguyễn Tấn Dũng 104

Peña Calderón 121

Pierwsze zero to pozycja wskaźnika zaraz po otwarciu pliku. 25 na końcu następnego wiersza to pozycja wskaźnika po przeczytaniu tego pierwszego wiersza. 36 na końcu następnego wiersza jest pozycją wskaźnika na końcu drugiego wiersza i tak dalej, aż do 121 na końcu, kiedy wskaźnik znajduje się na samym końcu pliku. Jeśli spróbujesz to zrobić za pomocą readlines(), otrzymasz zupełnie inny wynik. Oto kod:

```
with open('names.txt', encoding='utf-8') as f:

print(f.tell())

# Reads in all lines first, then loops through.

for one_line in f.readlines():

print(one_line[:-1],f.tell())
```

Here is the output:

```
0

Björk Guðmundsdóttir 121

121

Борис Николаевич Ельцин 121

Nguyễn Tấn Dũng 121

Peña Calderón 121
```

Wskaźnik zaczyna się od pozycji zero, zgodnie z oczekiwaniami. Ale każda linia pokazuje 121 na końcu. Dzieje się tak, ponieważ funkcja readlines() podczas wykonywania odczytuje cały plik, pozostawiając wskaźnik na końcu, na pozycji 121. Pętla w rzeczywistości przechodzi przez kopię pliku, który znajduje się w pamięci; nie jest to już czytanie pliku. Jeśli spróbujesz użyć .tell() z superprostą pętlą read() pokazaną tutaj:

```
with open('names.txt', encoding='utf-8') as f:

for one_line in f:

print(one_line, f.tell())
```

. . . nie będzie działać w systemie Windows. Jeśli więc z jakiegoś powodu musisz śledzić, gdzie znajduje się wskaźnik w jakimś zewnętrznym pliku tekstowym, który czytasz, upewnij się, że używasz pętli z readline().

Przesuwanie wskaźnika za pomocą seek()

Chociaż metoda tell() informuje, gdzie znajduje się wskaźnik w pliku zewnętrznym, metoda seek() umożliwia zmianę położenia wskaźnika. Składnia to:

```
file.seek(position[,whence])
```

Zastąp plik nazwą zmiennej otwartego pliku. Zastąp pozycję, aby wskazać miejsce, w którym chcesz umieścić wskaźnik. Na przykład 0, aby przenieść go z powrotem na początek pliku. Skąd jest opcjonalne i można go użyć do wskazania, do którego miejsca w pliku ma być obliczona pozycja. Twoje wybory to:

0: Ustaw pozycję względem początku pliku.

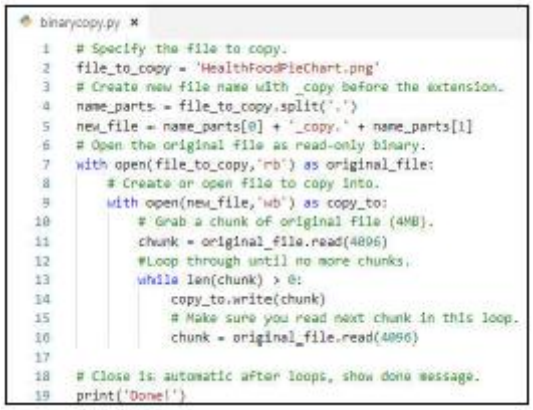
1: Ustaw pozycję względem bieżącej pozycji wskaźnika.

2: Ustaw pozycję względem końca pliku. Użyj liczby ujemnej dla pozycji.

Jeśli pominiesz wartość wherece, wartość domyślna wynosi zero. Zdecydowanie najczęstszym zastosowaniem seek jest po prostu przestawienie wskaźnika z powrotem na górę pliku w celu kolejnego przejścia przez plik. Składnia tego jest po prostu .seek(0).

Odczytywanie i kopiowanie pliku binarnego

Załóżmy, że masz aplikację, która w jakiś sposób zmienia plik binarny i chcesz zawsze pracować z kopią oryginalnego pliku, aby zachować bezpieczeństwo. Pliki binarne mogą być ogromne, więc zamiast otwierać je wszystkie na raz i ryzykować, że zabraknie pamięci, możesz czytać je w porcjach i zapisywać w porcjach. Pliki binarne nie zawierają treści czytelnej dla człowieka. Nie mają też linii tekstu. Więc readline() i readlines() nie są dobrym wyborem do zapętlania plików binarnych. Ale możesz użyć .read() o określonym rozmiarze, aby osiągnąć podobny wynik z plikami binarnymi. Rysunek przedstawia plik o nazwie binarycopy.py, który utworzy kopię dowolnego pliku binarnego. Przeprowadzimy Cię przez to krok po kroku, abyś mógł zrozumieć, jak to działa.



Pierwszym krokiem jest określenie pliku, który chcesz skopiować. Wybraliśmy happy_pickle.jpg, który, jak widać na rysunku, znajduje się w tym samym folderze, co folder binarycopy.py:

Specify the file to copy.

file_to_copy = 'happy_pickle.jpg'

Aby utworzyć pusty plik do kopiowania, najpierw potrzebujesz nazwy pliku dla pliku. Zajmuje się tym poniższy kod:

Create new file name with _copy before the extension.

name_parts = file_to_copy.split('.')

new_file = name_parts[0] + '_copy.' + name_parts[1]

W celu wykonania kopii można otworzyć oryginalny plik w trybie rb (plik do odczytu, plik binarny). Otwórz plik, do którego chcesz skopiować oryginalny plik w trybie wb (zapis, binarny). Za pomocą zapisu Python tworzy plik o tej nazwie, jeśli plik jeszcze nie istnieje. Jeśli plik istnieje, Python otwiera go ze wskaźnikiem ustawionym na 0, więc wszystko, co zapiszesz w pliku, zastąpi (a nie doda) istniejący plik. W kodzie widać, że użyliśmy pliku_oryginalnego jako nazwy zmiennej, z której należy kopiować, oraz copy_to jako nazwy zmiennej pliku, do którego kopiujemy dane. Wcięcia, jak zawsze, są krytyczne:

Open the original file as read-only binary.

with open(file_to_copy,'rb') as original_file:

Create or open file to copy into.

with open(new_file,'wb') as copy_to:

Jeśli użyjesz metody .read() do wczytania całego pliku binarnego, ryzykujesz, że będzie on tak duży, że przeciąży pamięć RAM komputera i spowoduje awarię programu. Aby tego uniknąć, napisaliśmy ten program tak, aby jednorazowo odczytywał skromne 4 MB (4096 kilobajtów) danych. Ten fragment o wielkości 4 MB jest przechowywany w zmiennej o nazwie fragment:

Grab a chunk of original file (4MB).

chunk = original_file.read(4096)

Następna linia tworzy pętlę, która czyta jeden fragment na raz. Wskaźnik jest automatycznie ustawiany na następny fragment przy każdym przejściu przez pętlę. W końcu trafi na koniec pliku, w którym nie może już czytać. Gdy tak się stanie, fragment będzie pusty, co oznacza, że ma długość równą 0. Tak więc ta pętla przechodzi przez plik aż do końca:

#Loop through until no more chunks.

while len(chunk) > 0:

Wewnątrz pętli pierwsza linia kopiuje ostatnio odczytany fragment do pliku copy_to . Drugi wiersz odczytuje następny fragment o wielkości 4 MB z oryginalnego pliku. I tak, dopóki wszystko z pliku original_file nie zostanie skopiowane do nowego pliku:

copy_to.write(chunk)

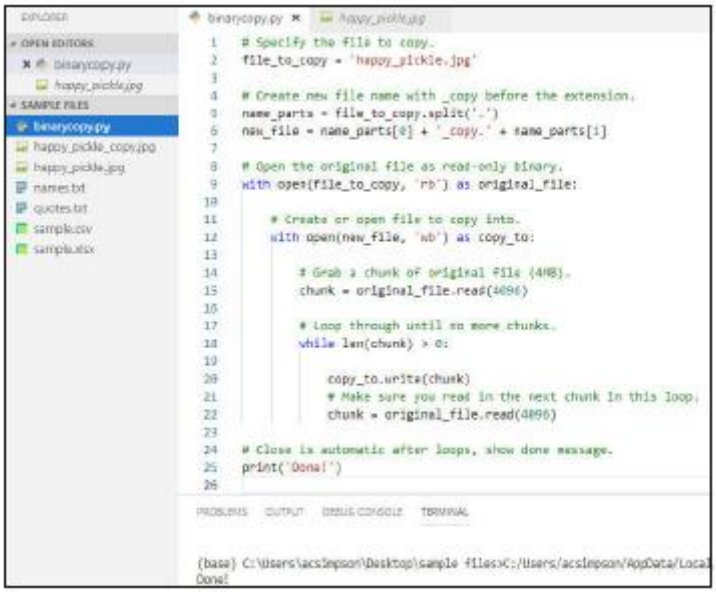
Make sure you read in the next chunk in this loop.

chunk = original_file.read(4096)

Wszystkie wcięcia kończą się po tej linii. Więc kiedy pętla się zakończy, pliki zamykają się automatycznie, a ostatni wiersz pokazuje tylko słowo Gotowe!

print('Done!')

Rysunek przedstawia wyniki działania kodu. Panel terminala po prostu pokazuje Gotowe!. Ale jak widać, w folderze znajduje się teraz plik o nazwie happy_pickle_copy.jpg. Otwarcie tego pliku udowodni, że jest to dokładna kopia oryginalnego pliku.

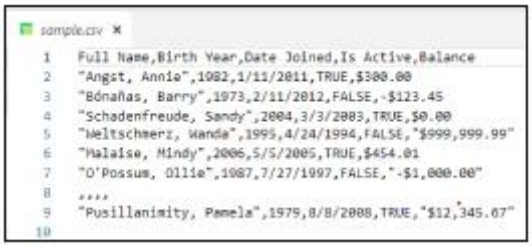


Podbijanie plików CSV

CSV (skrót od Comma Separated Values) to szeroko stosowany format do przechowywania i przesyłania danych tabelarycznych. Tabelaryczny oznacza, że można go ogólnie wyświetlić w formacie tabeli składającym się z wierszy i kolumn. W arkuszach kalkulacyjnych, takich jak Microsoft Excel, Apple Numbers czy Arkusze Google, format tabelaryczny jest dość oczywisty, jak pokazano na rysunku.

	A	B	C	D	E
1	**Full Name**	**Birth Year**	**Date Joined**	**Is Active**	**Balance**
2	Angst, Annie	1982	1/11/2011	TRUE	$300.00
3	Bónañas, Barry	1973	2/11/2012	FALSE	-$123.45
4	Schadenfreude, Sandy	2004	3/3/2003	TRUE	$0.00
5	Weltschmerz, Wanda	1995	4/24/1994	FALSE	$999,999.99
6	Malaise, Mindy	2006	5/5/2005	TRUE	$454.01
7	O'Possum, Ollie	1987	7/27/1997	FALSE	-$1,000.00
8					
9	Pusillanimity, Pamela	1979	8/8/2008	TRUE	$12,345.67

Bez pomocy jakiegoś specjalnego programu, który wyświetla dane w pliku w zgrabnym formacie tabelarycznym, każdy wiersz jest po prostu linią w pliku. A każda unikalna wartość jest oddzielona przecinkiem. Na przykład otwarcie pliku pokazanego na rysunku 1.10 w prostym edytorze tekstu, takim jak Notatnik lub TextEdit, pokazuje, co naprawdę jest zapisane w pliku.



W edytorze tekstu pierwszy wiersz, często nazywany nagłówkiem, zawiera nagłówki kolumn lub nazwy pól, które pojawiają się w pierwszym wierszu arkusza kalkulacyjnego. Jeśli spojrzysz na nazwy w drugim przykładzie, nieprzetworzonym pliku CSV, zauważysz, że wszystkie są ujęte w cudzysłowy, na przykład:

"Angst, Annie"

W rzeczywistości mogą to być pojedyncze cudzysłowy lub podwójne, jak pokazano. Ale tak czy inaczej wskazują, że kropka między cudzysłowami to wszystko jedno. Innymi słowy, przecinek między nazwiskiem a imieniem jest w całości częścią nazwiska. Ten przecinek nie jest początkiem nowej kolumny. Tak więc pierwsze dwie kolumny w tym pierwszym wierszu są

"Angst, Annie", 1982

. . . a nie

Angst, Annie

To samo dotyczy wszystkich innych wierszy: nazwa ujęta w cudzysłowy (wraz z przecinkami) to tylko jedna nazwa, a nie dwie oddzielne kolumny danych. Jeśli którykolwiek z ciągów zawiera apostrof, który jest tym samym znakiem, co pojedynczy cudzysłów, należy użyć podwójnych cudzysłowów wokół ciągu. Bo jeśli zrobisz to tak:

'O'Henry, Harry'

Pierwsza część łańcucha wygląda jak „O”, a potem Python nie będzie wiedział, co zrobić z tekstem po drugim pojedynczym cudzysłowie. Używanie podwójnych cudzysłowów zmniejsza wszelkie nieporozumienia, ponieważ w nazwie nie ma innych podwójnych cudzysłowów:

„O’Henry, Harry”

Rysunek powyższy zawiera również kilka innych problemów, które możesz napotkać podczas samodzielnej pracy z plikami CSV. Na przykład imię Bónañas, Barry zawiera kilka znaków spoza zestawu ASCII. Przedostatni wiersz zawiera tylko kilka przecinków. Jeśli w pliku CSV w komórce brakuje danych, po prostu wstawiasz przecinek kończący tę komórkę bez niczego po jej lewej stronie. Kolumna Saldo zawiera znaki dolara i przecinki w liczbach, które nie działają z typem danych zmiennoprzecinkowych Pythona. Mówimy o tym, jak sobie z tym wszystkim poradzić w kolejnych sekcjach. Chociaż z pewnością dałoby się pracować z plikami CSV przy użyciu tego, czego nauczyłeś się do tej pory, jest to o wiele szybsze i łatwiejsze, jeśli używasz modułu csv, który już masz. Aby go użyć, po prostu umieść to u góry swojego programu:

import csv

Pamiętaj, że to nie przynosi pliku CSV. Po prostu wprowadza wstępnie napisany kod, który ułatwia pracę z plikami CSV we własnym kodzie Pythona.

Otwieranie pliku CSV

Otwarcie pliku CSV tak naprawdę nie różni się od otwierania jakiegokolwiek innego pliku. Pamiętaj tylko, że jeśli plik zawiera znaki specjalne, musisz dołączyć encoding='utf-8', aby uniknąć komunikatu o błędzie. Opcjonalnie, podczas importowania danych prawdopodobnie nie chcesz wczytywać znaku nowej linii na końcu każdego wiersza, więc możesz dodać newline='' do instrukcji open(). Oto jak możesz to skomentować i zakodować, z wyjątkiem zastąpienia pliku sample.csv ścieżką do pliku CSV, który chcesz otworzyć:

Open CSV file with UTF-8 encoding, don't read in newline characters.

with open('sample.csv', encoding='utf-8', newline='') as f:

Aby zapętlić plik CSV, możesz użyć wbudowanej funkcji czytnika, która odczytuje jeden wiersz z wykonaniem. Ponownie, składnia jest dość prosta, jak pokazano w poniższym kodzie. Zamień f na dowolną nazwę, której użyłeś na końcu otwartej instrukcji (bez dwukropka na samym końcu).

reader = csv.reader(f)

Chociaż jest to całkowicie opcjonalne, możesz także liczyć wiersze na bieżąco. Po prostu umieść wszystko na prawo od = w enumerate(), jak pokazano poniżej (gdzie dodaliśmy również komentarz nad kodem):

Create a CVS row counter and row reader.

reader = enumerate(csv.reader(f))

Następnie możesz skonfigurować pętlę do odczytu jednego wiersza na raz. Ponieważ umieściłeś na nim moduł wyliczający, możesz użyć dwóch nazw zmiennych w swoim for: pierwsza (którą nazwiemy i) będzie śledzić licznik (który zaczyna się od zera i zwiększa o 1 przy każdym przejściu przez pętla). Druga zmienna, wiersz, będzie zawierała cały wiersz danych z pliku CSV:

Loop through one row at a time, i is counter, row is entire row.

for i, row in reader:

Możesz zacząć od tego, a następnie użyć funkcji print(), aby wydrukować wartość i i row przy każdym przejściu przez pętlę, tak jak poniżej:

import csv

Open CSV file with UTF-8 encoding, don't read in newline characters.

with open('sample.csv', encoding='utf-8', newline='') as f:

Create a CVS row counter and row reader.

reader = enumerate(csv.reader(f))

Loop through one row at a time, i is counter, row is entire row.

for i, row in reader:

print(i, row)

print('Done')

Dane wyjściowe z tego, przy użyciu pliku sample.csv opisanego wcześniej jako dane wejściowe, są następujące:

0 ['\ufeffFull Name', 'Birth Year', 'Date Joined', 'Is Active', 'Balance']

1 ['Angst, Annie', '1982', '1/11/2011', 'TRUE', '$300.00']

2 ['Bónañas, Barry', '1973', '2/11/2012', 'FALSE', '-$123.45']

3 ['Schadenfreude, Sandy', '2004', '3/3/2003', 'TRUE', '$0.00']

4 ['Weltschmerz, Wanda', '1995', '4/24/1994', 'FALSE', '$999,999.99']

5 ['Malaise, Mindy', '2006', '5/5/2005', 'TRUE', '$454.01']

6 ["O'Possum, Ollie", '1987', '7/27/1997', 'FALSE', '-$1,000.00']

7 ['', '', '', '', '']

8 ['Pusillanimity, Pamela', '1979', '8/8/2008', 'TRUE', '$12,345.67']

Zwróć uwagę, że wiersz nazw kolumn to wiersz zero. Dziwne \ufeff przed pełną nazwą w tym wierszu nazywa się Byte Order Mark (BOM) i jest to po prostu coś, co Excel tam wtyka. Zazwyczaj nie obchodzi Cię, co jest w pierwszym rzędzie, ponieważ prawdziwe dane zaczynają się dopiero w następnym rzędzie. Więc nie zastanawiaj się nad BOM-em, nie ma on dla ciebie żadnej wartości ani nie wyrządza żadnej szkody. Zwróć uwagę, że każdy wiersz jest w rzeczywistości listą pięciu elementów oddzielonych przecinkami. W swoim kodzie możesz odwoływać się do każdej kolumny według jej pozycji. Na przykład wiersz[0] jest pierwszą kolumną w wierszu (nazwisko osoby). Następnie wiersz [1] to rok urodzenia, wiersz [2] to data połączenia, wiersz [3] to, czy dana osoba jest aktywna, a wiersz [4] to saldo. Wszystkie dane w pliku CSV to ciągi — nawet jeśli nie wyglądają jak ciągi. Jednak wszystko, co pochodzi z pliku CSV, jest ciągiem znaków, ponieważ plik CSV jest typem pliku tekstowego, a plik tekstowy zawiera tylko ciągi znaków (tekst) i nie zawiera liczb całkowitych, dat, wartości logicznych ani liczb owsa. W Twojej aplikacji prawdopodobnie będziesz chciał przekonwertować przychodzące dane na typy danych Pythona, aby móc z nimi pracować bardziej efektywnie, a może nawet przenieść je do bazy danych. W następnych sekcjach przyjrzymy się, jak wykonać konwersję dla każdego typu danych.

Konwersja stringów

Technicznie rzecz biorąc, nie musisz konwertować niczego z pliku CSV na ciąg znaków. Ale możesz chcieć trochę to posiekać lub poradzić sobie w jakiś sposób z pustymi łańcuchami, więc jest kilka rzeczy, które możesz zrobić. Po pierwsze, jak wspomnieliśmy wcześniej, interesują nas tylko dane tutaj, a nie ten pierwszy wiersz. Tak więc wewnątrz pętli możesz zacząć od if, to nic nie da, jeśli bieżący wiersz jest wierszem zerowym. Zastąp print(i,wiersz) w ten sposób:

Row 0 is just column headings, ignore it.

if i > 0:

full_name = row[0].split(',')

last_name=full_name[0].strip()

first_name=full_name[1].strip()

Ten kod mówi: „Dopóki nie patrzymy na pierwszy wiersz, utwórz zmienną o nazwie pełna nazwa i przechowuj w niej to, co znajduje się w pierwszej kolumnie, podzielone na dwie osobne wartości w przecinku”. Po wykonaniu tej linii, full_name[0] zawiera nazwisko osoby, które następnie umieszczamy w zmiennej o nazwie first_name, a full_name[1] zawiera imię osoby, które umieszczamy w zmiennej o nazwie first_name. Ale jeśli uruchomisz kod w ten sposób, zostanie on zbombardowany, ponieważ wiersz 7 nie ma nazwy, a Python nie może podzielić pustego ciągu znaków przecinkiem (ponieważ pusty ciąg znaków nie zawiera przecinka). Aby to obejść, możesz powiedzieć Pythonowi, aby spróbował podzielić nazwę przecinkiem, jeśli to możliwe. Ale jeśli bombarduje podczas próby, po prostu zapisz i pusty ciąg w zmiennych full_name, last_name i first_name. Oto ten kod z kilkoma dodatkowymi komentarzami wyjaśniającymi wszystko, co się dzieje. Zamiast wypisywać i i cały wiersz, kod wypisuje tylko imię i nazwisko (i nic dla wiersza, którego informacji brakuje). Możesz zobaczyć dane wyjściowe poniżej kodu poniżej.

import csv

Open CSV file with UTF-8 encoding, don't read in newline characters.

with open('sample.csv', encoding='utf-8', newline='') as f:

Create a CVS row counter and row reader.

```
reader = enumerate(csv.reader(f))

# Loop through one row at a time, i is counter, row is entire row.

for i, row in reader:

# Row 0 is just column headings, ignore it.

if i > 0:

# Whole name split into two at comma.

try:

full_name = row[0].split(',')

# Last name, strip extra spaces.

last_name=full_name[0].strip()

# First name, strip extra spaces.

first_name=full_name[1].strip()

except IndexError:

full_name = last_name = first_name = ""

print(first_name, last_name)

print('Done!')
```

```
Annie Angst

Barry Bónañas

Sandy Schadenfreude

Wanda Weltschmerz

Mindy Malaise

Ollie O'Possum

Pamela Pusillanimity

Done!
```

Konwersja na liczby całkowite

Druga kolumna w każdym wierszu, wiersz [1], to rok urodzenia. Dopóki łańcuch zawiera coś, co można przekonwertować na liczbę, możesz użyć prostej wbudowanej funkcji int(), aby przekonwertować go na liczbę całkowitą. Mamy jednak problem w rzędzie 7, który jest pusty. Python nie przekonwertuje tego automatycznie na zero, musisz mu trochę pomóc. Oto kod do tego:

```
# Birth year integer, zero for empty string.

birth_year= int(row[1] or 0)
```

Kod wygląda zaskakująco prosto, ale na tym polega piękno Pythona: jest zaskakująco prosty. Ten wiersz kodu mówi: „utwórz zmienną o nazwie rok_urodzenia i umieść w niej wartość drugiej kolumny, jeśli możesz lub jeśli nie ma nic do przekonwertowania na liczbę całkowitą, po prostu wstaw zero”.

Konwersja do daty

Trzecia kolumna w naszym pliku CSV, wiersz[2], to data połączenia i wydaje się, że w każdym wierszu (z wyjątkiem wiersza, w którym brakuje danych) jest odpowiednia data. Aby przekonwertować to na datę, musisz najpierw zaimportować moduł datetime, dodając import datetime jako dt up w górnej części programu. Wtedy prosta konwersja to po prostu:

date_joined = dt.datetime.strptime(row[2],"%m/%d/%Y").date()

Dużo się tam dzieje, więc rozpakujmy to trochę. Najpierw tworzysz zmienną o nazwie date_joined. Struptime oznacza „przetwarzanie łańcucha dla daty i godziny”. [wiersz,2] oznacza trzecią kolumnę (ponieważ pierwszą kolumną jest zawsze kolumna 0). „%m/%d/%Y” mówi strptime, że ciąg data zawiera miesiąc, po którym następuje ukośnik, dzień miesiąca, po którym następuje ukośnik, a następnie czterocyfrowy rok (wielkie litery %Y). Funkcja .date() na samym końcu oznacza „tylko datę; nie ma tu czasu na analizowanie”. Jeden mały problem. Kiedy dojdzie do wiersza, w którym brakuje daty, spowoduje to bombardowanie. Tak więc jeszcze raz użyjemy try…, aby utworzyć datę, a jeśli nie uda się uzyskać daty, wstawimy wartość None, która w Pythonie oznacza pusty obiekt. W Pythonie datetime jest klasą, więc każda utworzona data i godzina jest w rzeczywistości obiektem (typu datetime). Nie używasz „” dla pustego obiektu, „” jest dla pustego ciągu. Python używa słowa None dla pustego obiektu.

Oto kod w obecnej postaci z importem do góry dla daty/godziny i try ... except konwersji daty łańcucha na datę Pythona:

```
import csv

import datetime as dt

# Open CSV file with UTF-8 encoding, don't read in newline characters.

with open('sample.csv', encoding='utf-8', newline='') as f:

# # Create a CVS row counter and row reader.

reader = enumerate(csv.reader(f))

# Loop through one row at a time, i is counter, row is entire row.

for i, row in reader:

# Row 0 is just column headings, ignore it.

if i > 0:

# Whole name split into two at comma.

try:

full_name = row[0].split(',')

# Last name, strip extra spaces.

last_name = full_name[0].strip()
```

```
# First name, strip extra spaces.

first_name = full_name[1].strip()

except IndexError:

full_name = last_name = first_name = ""

# Birth year integer, zero for empty string.

birth_year = int(row[1] or 0)

# Date_joined is a date.

try:

date_joined = dt.datetime.strptime(row[2], "%m/%d/%Y").date()

except ValueError:

date_joined = None

print(first_name, last_name, birth_year, date_joined)

print('Done!')
```

Oto wynik tego kodu, który teraz wyświetla imię, nazwisko, rok_urodzenia i datę_dołączenia przy każdym przejściu przez wiersze danych w tabeli:

```
Annie Angst 1982 2011-01-11

Barry Bónañas 1973 2012-02-11

Sandy Schadenfreude 2004 2003-03-03

Wanda Weltschmerz 1995 1994-04-24

Mindy Malaise 2006 2005-05-05

Ollie O'Possum 1987 1997-07-27

0 None

Pamela Pusillanimity 1979 2008-08-08

Done!
```

Konwersja na wartość logiczną

Czwarta kolumna, wiersz [3] w każdym wierszu zawiera PRAWDA lub FAŁSZ. Excel używa wszystkich wielkich liter w ten sposób, co jest automatycznie przenoszone do pliku CSV podczas zapisywania jako CSV w Excelu. Python używa wielkich liter na początku, True i False. Python ma prostą funkcję bool() do wykonywania tej konwersji. I nie wybuchnie, gdy uderzy w pustą komórkę. . . po prostu uważa tę komórkę za Fałsz. Więc ta konwersja może być tak prosta, jak ta:

```
# is_active is a Boolean, automatically False for empty string.

is_active=bool(row[3])
```

Konwersja do float

Piąta kolumna w każdym wierszu zawiera saldo, które jest kwotą wyrażoną w dolarach. W Pythonie chcesz, aby był to owies. Ale od samego początku jest problem. Python oats nie może zawierać znaku dolara ($) ani przecinka (,). Tak więc pierwszą rzeczą, którą musisz zrobić, to usunąć je z łańcucha. Ponadto nie możesz mieć żadnych przypadkowych spacji wiodących lub końcowych. Można je łatwo usunąć za pomocą metody strip(). Ta linia tworzy zmienną o nazwie str_balance (która nadal jest łańcuchem), ale z usuniętym znakiem dolara, przecinkiem i wszelkimi końcowymi spacjami:

Remove $, commas, leading trailing spaces.

str_balance = (row[4].replace('$','').replace(',','')).strip()

Ten drugi wiersz można przeczytać w następujący sposób: „nowy ciąg o nazwie balance składa się z tego, co znajduje się w piątej kolumnie po zastąpieniu jakichkolwiek znaków dolara niczym, przecinków niczym i usunięciu wszystkich początkowych i końcowych spacji”. Poniżej tego wiersza możesz dodać przecinek, a następnie kolejny wiersz, aby utworzyć element zmiennoprzecinkowy o nazwie balance, który wykorzystuje wbudowaną metodę float() do konwersji łańcucha str_balance na element zmiennoprzecinkowy. Podobnie jak int(), float() ma własną wbudowaną procedurę obsługi wyjątków, która, jeśli nie jest w stanie zrozumieć, co próbuje przekonwertować na float, przechowuje zero jako wartość float. Kod na rysunku pokazuje wszystko na swoim miejscu, łącznie z linią print(), która wyświetla wartości wszystkich pięciu kolumn po konwersji.

UŻYWANIE WYRAŻEŃ REGULARNYCH W PYTHONIE

Chociaż zakładamy, że nie znasz jeszcze innych języków programowania, niektórzy czytelnicy nieuchronnie to zrobią, a niektórzy z nich prawdopodobnie zapytają, dlaczego nie użyliśmy wyrażenia regularnego, aby zamiast tego usunąć znak dolara i przecinek z salda metody replace(). Odpowiedź na to brzmiałaby: „Ponieważ nie musisz robić tego w ten sposób, a nie wszyscy czytający tę książkę są świadomi, że coś, co nazywa się wyrażeniami regularnymi, jest dostępne w większości języków programowania”. Ale jeśli zdarzy ci się być osobą, która zastanawiała się nad zadaniem tego pytania,

pierwszą rzeczą, o której musisz wiedzieć, jest to, że wyrażenia regularne nie są wbudowane w Pythona. Więc jeśli chcesz ich użyć, musisz umieścić import re na górze swojego kodu. W tym konkretnym przykładzie, który po prostu wykorzystuje możliwości zastępowania wyrażeń regularnych, potrzebujesz tego w górnej części kodu:

from re import sub

W dalszej części kodu możesz usunąć plik

str_balance = (row[4].replace('$','').replace(',','')).strip()

linię całkowicie i zastąpić ją

str_balance = (sub(r'[\s\$,]','',row[4])).strip()

Ta linia robi dokładnie to samo, co oryginalna linia. Usuwa znak dolara, przecinki oraz wszelkie początkowe i końcowe spacje z wartości w piątej kolumnie.

Od CSV do obiektów i słowników

Widziałeś już, jak wczytywać dane z dowolnego pliku CSV i jak konwertować te dane z domyślnego typu łańcuchowego na odpowiedni typ danych Pythona. Możliwe, że oprócz tego wszystkiego zechcesz uporządkować dane w grupę obiektów, wszystkie wygenerowane z tej samej klasy, lub być może w zestaw słowników w większym słowniku. Cały kod, którego nauczyłeś się do tej pory, będzie przydatny, ponieważ jest niezbędny do wykonania zadania. Aby zmniejszyć bałagan w kodzie w tych przykładach, wzięliśmy różne fragmenty kodu do konwersji danych i umieściliśmy je we własnych funkcjach. Pozwala to na konwersję elementu danych za pomocą nazwy funkcji z wartością do przeliczenia w nawiasach, na przykład: balance(wiersz[4]).

Importowanie CSV do obiektów Pythona

Jeśli chcesz, aby dane z pliku CSV były uporządkowane w postaci listy obiektów, napisz kod w sposób pokazany tutaj:

import datetime as dt

import csv

Use these functions to convert any string to appropriate Python data type.

Get just the first name from full name.

def fname(any):

try:

nm = any.split(',')

return nm[1]

except IndexError:

return ''

Get just the last name from full name.

def lname(any):

```
try:
nm = any.split(',')
return nm[0]
except IndexError:
return ''
# Convert string to integer or zero if no value.
def integer(any):
return int(any or 0)
# Conver mm/dd/yyyy date to date or None if no valid date.
def date(any):
try:
return dt.datetime.strptime(any,"%m/%d/%Y").date()
except ValueError:
return None
# Convert any string to Boolean, False if no value.
def boolean(any):
return bool(any)
# Convert string to float, or to zero if no value.
def floatnum(any):
s_balance = (any.replace('$','').replace(',','')).strip()
return float(s_balance or 0)
# Create an empty list of people.
people = []
# Define a class where each person is an object.
class Person:
def __init__(self, id, first_name, last_name, birth_year, date_joined, is_
active, balance):
self.id = id
self.first_name = first_name
self.last_name = last_name
self.birth_year = birth_year
```

```
self.date_joined = date_joined
self.is_active = is_active
self.balance = balance
# Open CSV file with UTF-8 encoding, don't read in newline characters.
with open('sample.csv', encoding='utf-8', newline='') as f:
# Set up a csv reader with a counter.
reader = enumerate(csv.reader(f))
# Skip the first row, which is column names.
f.readline()
# Loop through remaining rows one at a time, i is counter, row is
entire row.
for i, row in reader:
# From each data row in the CSV file, create a Person object with unique
id and appropriate data types, add to people list.
people.append(Person(i, fname(row[0]), lname(row[0]), integer(row[1]),
date(row[2]), boolean(row[3]), floatnum(row[4])))
# When above loop is done, show all objects in the people list.
for p in people:
print(p.id, p.first_name, p.last_name, p.birth_year, p.date_joined, p.is_
active, p.balance)
```

Oto jak działa ten kod: Pierwsze kilka wierszy to wymagane importy, po których następuje szereg funkcji konwertujących przychodzące dane łańcuchowe na typy danych Pythona. Ten kod jest podobny do poprzednich przykładów w tym rozdziale. Po prostu podzieliliśmy kod konwersji na osobne funkcje, aby wszystko trochę podzielić:

```
import datetime as dt
import csv
# Use these functions to convert any string to appropriate Python data type.
# Get just the first name from full name.
def fname(any):
try:
nm = any.split(',')
return nm[1]
```

```
except IndexError:

return ''

# Get just the last name from full name.

def lname(any):

try:

nm = any.split(',')

return nm[0]

except IndexError:

return '

# Convert string to integer or zero if no value.

def integer(any):

return int(any or 0)

# Conver mm/dd/yyyy date to date or None if no valid date.

def date(any):

try:

return dt.datetime.strptime(any,"%m/%d/%Y").date()

except ValueError:

return None

# Convert any string to Boolean, False if no value.

def boolean(any):

return bool(any)

# Convert string to float, or to zero if no value.

def floatnum(any):

s_balance = (any.replace('$','').replace(',','')).strip()

return float(s_balance or 0)
```

Ten następny wiersz tworzy pustą listę o nazwie people. To po prostu zapewnia miejsce do przechowywania obiektów, które program utworzy z pliku CSV:

```
# Create an empty list of people.

people = []
```

Następnie kod definiuje klasę, która będzie używana do generowania każdego obiektu Person z pliku CSV:

```
# Define a class where each person is an object.
```

```
class Person:

def __init__(self, id, first_name, last_name, birth_year, date_joined, is_

active, balance):

self.id = id

self.first_name = first_name

self.last_name = last_name

self.birth_year = birth_year

self.date_joined = date_joined

self.is_active = is_active

self.balance = balance
```

Właściwy odczyt pliku CSV rozpoczyna się w kolejnych liniach. Zwróć uwagę, jak kod otwiera plik sample.csv z kodowaniem. Newline='' po prostu zapobiega przyklejeniu znaku nowej linii, który znajduje się na końcu każdego wiersza, do ostatniego elementu danych w każdym wierszu. Czytnik używa modułu wyliczającego do liczenia podczas czytania wierszy. Funkcja f.readline() odczytuje pierwszy wiersz, który jest po prostu nagłówkami kolumn, więc następująca po nim funkcja for zaczyna się w drugim wierszu. Zmienna i w pętli for to tylko rosnący licznik, a wiersz to cały wiersz danych z pliku CSV:

```
# Open CSV file with UTF-8 encoding, don't read in newline characters.

with open('sample.csv', encoding='utf-8', newline='') as f:

# Set up a csv reader with a counter.

reader = enumerate(csv.reader(f))

# Skip the first row, which is column names.

f.readline()

# Loop through remaining rows one at a time, i is counter, row is

entire row.

for i, row in reader:
```

Z każdym przejściem przez pętlę ta linia tworzy pojedynczy obiekt Person na podstawie rosnącego licznika (i) i dołącza dane w wierszu. Zwróć uwagę, jak wywołaliśmy funkcje zdefiniowane wcześniej w kodzie do konwersji typów danych. Dzięki temu ten kod jest bardziej zwarty i nieco łatwiejszy do odczytania i pracy z nim:

```
# From each data row in the CSV file, create a Person object with unique id

and appropriate data types, add to people list.

people.append(Person(i, fname(row[0]), lname(row[0]), integer(row[1]),

date(row[2]), boolean(row[3]), floatnum(row[4])))
```

Po zakończeniu pętli następny kod po prostu wyświetla każdy obiekt na ekranie, aby sprawdzić, czy kod działa poprawnie:

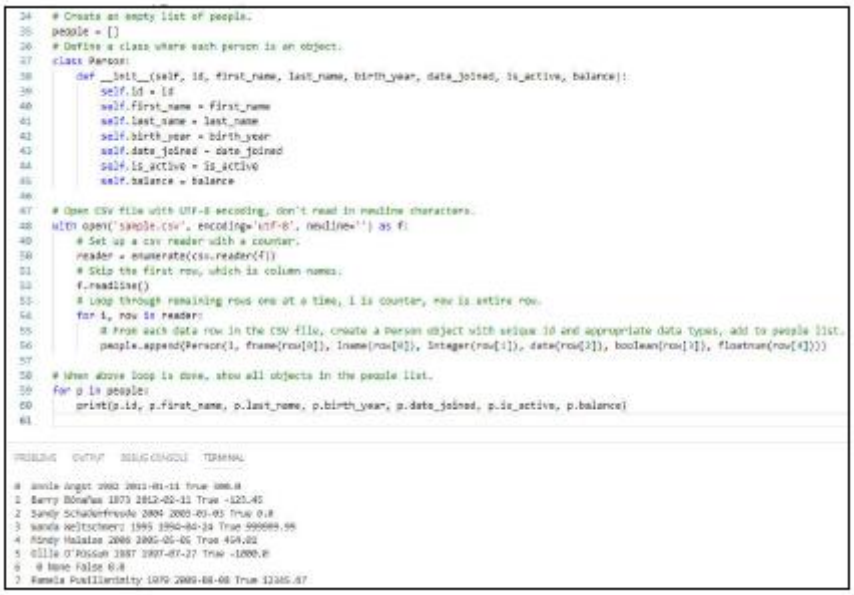
When above loop is done, show all objects in the people list.

for p in people:

print(p.id, p.first_name, p.last_name, p.birth_year, p.date_joined,

p.is_active, p.balance)

Rysunek przedstawia wynik działania tego programu. Oczywiście kolejny kod w programie może zrobić wszystko, co musisz zrobić z każdym obiektem; drukowanie jest tylko po to, aby przetestować i zweryfikować, czy zadziałało.



Importowanie CSV do słowników Pythona

Jeśli wolisz przechowywać każdy wiersz danych z pliku CSV w jego własnym słowniku, możesz użyć kodu podobnego do poprzedniego kodu do tworzenia obiektów. Nie potrzebujesz kodu definicji klasy, ponieważ nie będziesz tutaj tworzyć obiektów. Zamiast tworzyć listę osób, możesz utworzyć pusty słownik osób, w którym będą przechowywane wszystkie indywidualne słowniki „osoby”, na przykład:

Create an empty dictionary of people.

people = {}

Jeśli chodzi o pętlę, ponownie możesz użyć modułu wyliczającego (i) do zliczania wierszy, a także możesz użyć tej unikalnej wartości jako klucza dla każdego tworzonego nowego słownika. Wiersz zaczynający się od newdict= tworzy słownik z danymi z jednego wiersza pliku CSV, używając wbudowanej funkcji dict() Pythona. Następna linia przypisuje każdemu nowo utworzonemu słownikowi wartość i plus jeden (aby zacząć od jedynki zamiast od zera):

Loop through remaining rows one at a time, i is counter, row is entire row.

for i, row in reader:

From each data row in the CSV file, create a Person object with unique

id and appropriate data types, add to people list.

newdict = dict({'first_name': fname(row[0]), 'last_name': lname(row[0]),

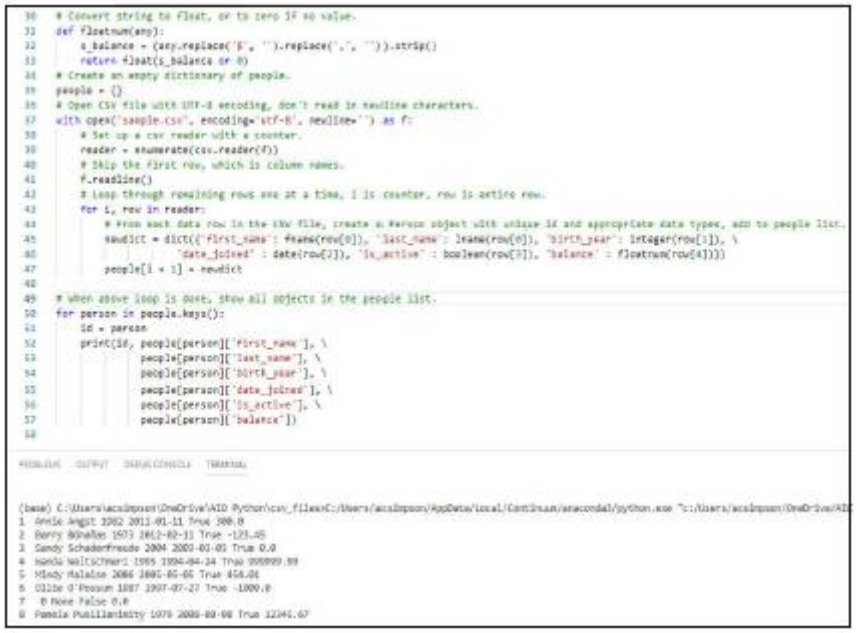
'birth_year': integer(row[1]), \

'date_joined' : date(row[2]), 'is_active' : boolean(row[3]),

'balance' : floatnum(row[4])})

people[i 1] = newdict

Aby sprawdzić, czy kod działa poprawnie, możesz przejść przez słowniki w słowniku osób i wyświetlić parę klucz:wartość dla każdego elementu danych w każdym wierszu. Rysunek 1-13 pokazuje wynik uruchomienia tego kodu w kodzie VS:



Oto cały kod wczytujący dane z plików CSV do słowników:

import datetime as dt

import csv

Use these functions to convert any string to appropriate Python data type.

Get just the first name from full name.

def fname(any):

try:

nm = any.split(',')

return nm[1]

except IndexError:

return ''

```
# Get just the last name from full name.
def lname(any):
try:
nm = any.split(',')
return nm[0]
except IndexError:
return ''
# Convert string to integer or zero if no value.
def integer(any):
return int(any or 0)
# Convert mm/dd/yyyy date to date or None if no valid date.
def date(any):
try:
return dt.datetime.strptime(any, "%m/%d/%Y").date()
except ValueError:
return None
# Convert any string to Boolean, False if no value.
def boolean(any):
return bool(any)
# Convert string to float, or to zero if no value.
def floatnum(any):
s_balance = (any.replace('$', '').replace(',', '')).strip()
return float(s_balance or 0)
# Create an empty dictionary of people.
people = {}
# Open CSV file with UTF-8 encoding, don't read in newline characters.
with open('sample.csv', encoding='utf-8', newline='') as f:
# Set up a csv reader with a counter.
reader = enumerate(csv.reader(f))
# Skip the first row, which is column names.
f.readline()
```

```
# Loop through remaining rows one at a time, i is counter, row is
entire row.
for i, row in reader:
# From each data row in the CSV file, create a Person object with
unique id
# and appropriate data types, add to people dictionary.
newdict = dict({'first_name': fname(row[0]), 'last_name': lname(row[0]),
'birth_year': integer(row[1]), \
'date_joined' : date(row[2]), 'is_active' : boolean(row[3]),
'balance' : floatnum(row[4])})
people[i 1] = newdict
# When above loop is done, show all objects in the people list.
for person in people.keys():
id = person
print(id, people[person]['first_name'], \
people[person]['last_name'], \
people[person]['birth_year'], \
people[person]['date_joined'], \
people[person]['is_active'], \
people[person]['balance'])
```

Pliki CSV są szeroko stosowane, ponieważ eksportowanie danych z arkuszy kalkulacyjnych i tabel baz danych do tego formatu jest łatwe. Pobieranie danych z tych plików może być czasem trudne, ale moduł csv Pythona będzie bardzo pomocny. Zajmuje się wieloma szczegółami, sprawia, że przeglądanie jednego wiersza w pętli jest stosunkowo łatwe i obsługuje dane w sposób, jakiego nie widzisz w swojej aplikacji Pythona. Podobny do CSV do przenoszenia i przechowywania danych w prostym formacie tekstowym jest JSON, który oznacza JavaScript Object Notation. W następnej części dowiesz się wszystkiego o JSON.