Błędy omijania

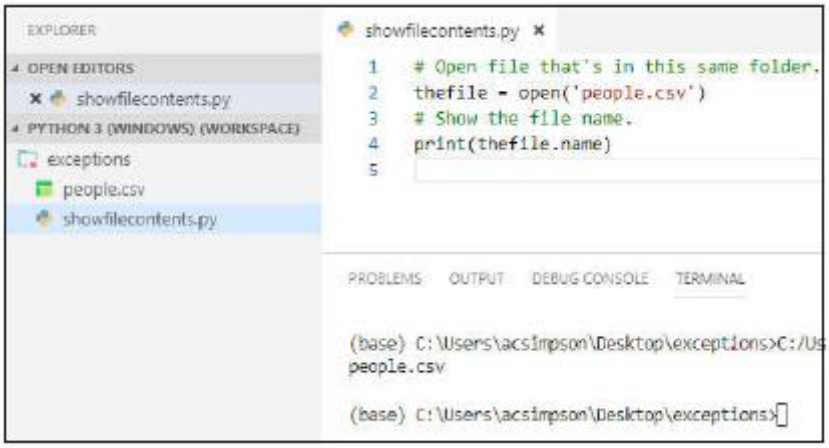
Wszyscy chcemy, aby nasze programy działały idealnie przez cały czas. Ale czasami w prawdziwym świecie zdarzają się sytuacje, które na to nie pozwalają. To nie jest twoja wina ani twój program. Zwykle jest to coś, co osoba korzystająca z programu zrobiła źle. Obsługa błędów polega na próbie przewidzenia, jakie mogą być te problemy, a następnie „wychwyceniu" błędu i poinformowaniu użytkownika o problemie, aby mógł go naprawić. Należy pamiętać, że przedstawione tu techniki nie służą do naprawiania błędów w kodzie. Tego rodzaju błędy musisz naprawić sam. Mówimy ściśle o błędach w środowisku, w którym działa program, nad którym nie masz kontroli. Obsługa błędu to po prostu sposób na zastąpienie technicznego komunikatu o błędzie, który zwykle wyświetla Python, który jest bez znaczenia dla większości ludzi, komunikatem, który mówi im prostym językiem, co jest nie tak, a najlepiej, jak to naprawić. Ponownie użytkownik będzie naprawiał środowisko, w którym działa program. . .nie będą naprawiać twojego kodu.

Zrozumienie wyjątków

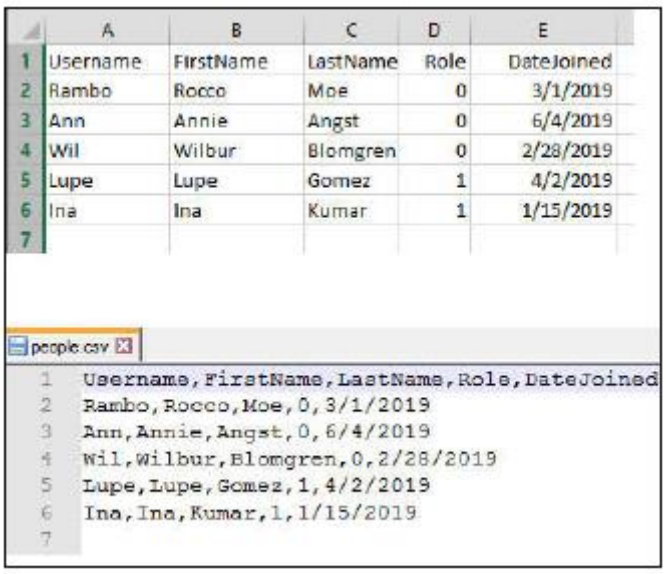
W Pythonie (i wszystkich innych językach programowania) termin wyjątek odnosi się do błędu, który nie jest spowodowany błędem programistycznym. Raczej jest to błąd w prawdziwym świecie, który uniemożliwia prawidłowe działanie programu. Jako prosty przykład niech Twoja aplikacja w Pythonie otworzy plik. Składnia tego jest łatwa. Kod jest po prostu

name = open(filename)

Zamień nazwę na wybraną przez siebie nazwę, taką samą jak nazwa dowolnej zmiennej. Zastąp nazwę pliku nazwą pliku. Jeśli plik znajduje się w tym samym folderze co kod, nie trzeba określać ścieżki do folderu, ponieważ zakładany jest bieżący folder. Rysunek przedstawia przykład.



W tym przykładzie użyliśmy VS Code, abyś mógł zobaczyć zawartość folderu, w którym pracowaliśmy. Folder zawiera plik o nazwie showfilecontents.py, który jest plikiem zawierającym napisany przez nas kod Pythona. Drugi plik nosi nazwę people.csv. Plik showcontents.py jako jedyny zawiera kod. Plik people.csv zawiera dane, informacje o ludziach. Jego zawartość nie ma teraz większego znaczenia; to, czego się tutaj uczysz, będzie działać w dowolnym pliku zewnętrznym. Ale dla twojej wiadomości, rysunek



	A	B	C	D	E
1	Username	FirstName	LastName	Role	DateJoined
2	Rambo	Rocco	Moe	0	3/1/2019
3	Ann	Annie	Angst	0	6/4/2019
4	Wil	Wilbur	Blomgren	0	2/28/2019
5	Lupe	Lupe	Gomez	1	4/2/2019
6	Ina	Ina	Kumar	1	1/15/2019
7					

people.csv

```
1  Username,FirstName,LastName,Role,DateJoined
2  Rambo,Rocco,Moe,0,3/1/2019
3  Ann,Annie,Angst,0,6/4/2019
4  Wil,Wilbur,Blomgren,0,2/28/2019
5  Lupe,Lupe,Gomez,1,4/2/2019
6  Ina,Ina,Kumar,1,1/15/2019
7
```

pokazuje zawartość tego pliku w Excelu (u góry), dzięki czemu jest łatwy do odczytania, oraz w edytorze tekstu (u dołu), tak jak wygląda to w Pythonie i innych językach. Kod Pythona to tylko dwie linie (bez komentarzy), jak następuje:

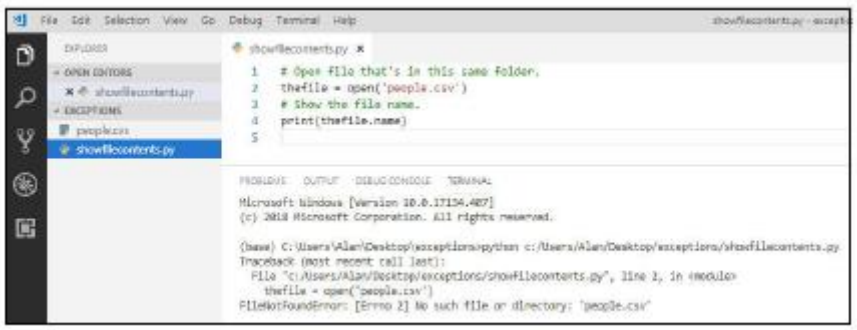
```
# Open file that's in this same folder.

thefile = open('people.csv')

# Show the file name.

print(thefile.name)
```

Więc to proste. Pierwszy wiersz kodu otwiera plik o nazwie people.csv. Drugi wiersz kodu pokazuje nazwę pliku (people.csv) na ekranie. Uruchomienie tej prostej aplikacji showfilecontents.py (klikając prawym przyciskiem myszy jej nazwę w VS Code i wybierając polecenie Uruchom plik Pythona w terminalu) pokazuje na ekranie people.csv — zakładając, że w folderze do otwarcia znajduje się plik o nazwie people.csv. Przy takim założeniu w grę wchodzi obsługa wyjątków. Załóżmy, że z przyczyn niezależnych od Ciebie nie ma tam pliku o nazwie people.csv, ponieważ jakaś osoba lub jakaś zautomatyzowana procedura nie umieściła go tam albo ktoś przypadkowo przekręcił nazwę pliku. Łatwo jest przypadkowo wpisać nazwę pliku, powiedzmy, .cvs zamiast .csv, jak na rysunku.



W takim przypadku uruchomienie aplikacji generuje wyjątek (co w języku angielskim oznacza „wyświetla komunikat o błędzie”), jak widać w terminalu na tym samym obrazie. Wyjątek brzmi:

Traceback (most recent call last):

File "c:/ Users/ acsimpson/ Desktop/ exceptions/ showfilecontents.py", line 2,

in <module>

thefile = open('people.csv')

FileNotFoundError: [Errno 2] No such file or directory: 'people.csv'

Traceback jest odniesieniem do faktu, że gdyby istniało wiele wyjątków, wszystkie zostałyby wymienione, przy czym najnowszy byłby wymieniony jako pierwszy. W tym przypadku jest tylko jeden wyjątek. Część Plik informuje, gdzie wystąpił wyjątek, w wierszu 2 pliku o nazwie showfilecontents.py. Część, która czyta

thefile = open('people.csv')

. . . pokazuje dokładną linię kodu, która spowodowała błąd. I wreszcie sam wyjątek jest opisany w ten sposób:

FileNotFoundError: [Errno 2] No such file or directory: 'people.csv'

Ogólna nazwa tego typu błędu to FileNotFoundError. Wiele wyjątków ma również powiązany numer, ale zwykle różni się on w zależności od środowiska systemu operacyjnego, więc zwykle nie jest używany do obsługi błędów. W tym przypadku głównym błędem jest FileNotFoundError, a fakt, że jest to ERRNO 2, w którym teraz siedzę, nie ma większego znaczenia. Niektórzy ludzie używają wyrażenia rzuć wyjątek, zamiast zgłaszać wyjątek. Ale to tylko dwa różne sposoby opisania tej samej rzeczy. Nie ma różnicy między zgłoszeniem a zgłoszeniem wyjątku. Ostatnia część mówi dokładnie, co poszło nie tak. Brak takiego pliku lub katalogu: „people.csv”. Innymi słowy, Python nie może wykonać operacji open('people.csv'), ponieważ w bieżącym folderze nie ma pliku o nazwie people.csv o tej nazwie. Możesz rozwiązać ten problem, zmieniając kod, ale .csv jest powszechnym rozszerzeniem plików zawierających wartości oddzielone przecinkami. Bardziej sensowna byłaby zmiana nazwy people.cvs na people.csv, tak aby odpowiadała temu, czego szuka program, a rozszerzenie .csv było dobrze znane. Nie możesz również zmienić nazwy pliku w aplikacji Python, ponieważ nie wiesz, czy inne pliki w tym folderze zawierają dane, których szuka aplikacja Python. Utworzenie pliku people.csv, upewnienie się, że ma on poprawną nazwę i zawiera typ informacji, którego szuka program w Pythonie, należy do człowieka.

Łagodna obsługa błędów

Najlepszym sposobem radzenia sobie z tego rodzaju błędami jest nie pokazywanie tego, co Python zwykle pokazuje dla tego błędu. Zamiast tego najlepiej byłoby zastąpić to czymś, co osoba korzystająca z aplikacji jest bardziej skłonna zrozumieć. Aby to zrobić, możesz zakodować try . . . oprócz bloku przy użyciu tej podstawowej składni:

try:

The things you want the code to do

except Exception:

What to do if it can't do what you want it to do

Oto jak możemy przepisać kod showfilecontents.py, aby obsłużyć błąd brakującego (lub błędnie napisanego) pliku:

try:

Open file and shows its name.

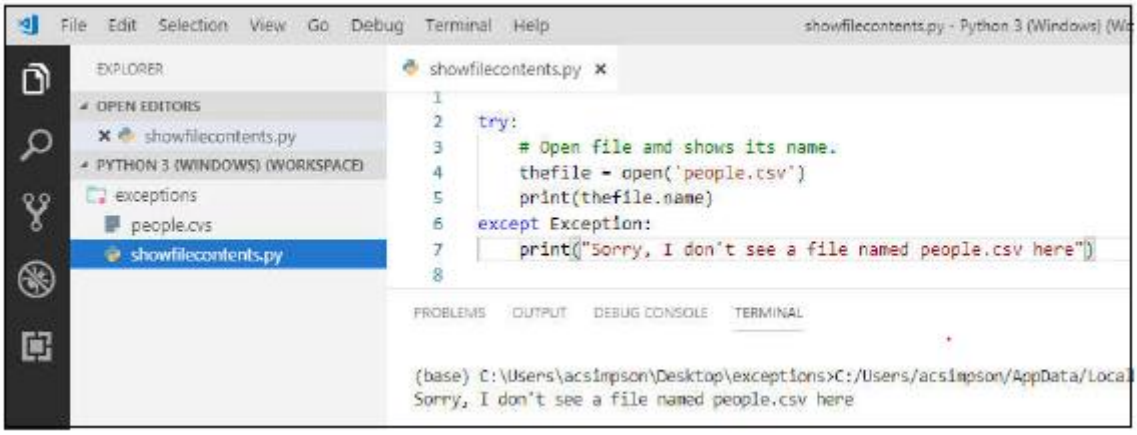
thefile = open('people.csv')

print(thefile.name)

except Exception:

print("Sorry, I don't see a file named people.csv here")

Ponieważ wiemy, że jeśli brakuje pliku, który aplikacja ma otworzyć, zaczynamy od try:, a następnie próbujemy otworzyć plik pod tym adresem. Jeśli plik się otworzy, świetnie, instrukcja print() zostanie uruchomiona i wyświetli nazwę pliku. Ale jeśli próba otwarcia pliku spowoduje wyjątek, program nie „bombarduje” i nie wyświetla ogólnego komunikatu o błędzie. Zamiast tego pokazuje wiadomość, która jest lepsza dla przeciętnego użytkownika komputera, jak pokazano na rysunku



Konkretne podejście do wyjątków

Poprzednia składnia z wdziękiem obsłużyła błąd „nie znaleziono pliku”. Ale mogłoby być bardziej zgrabnie. Na przykład, jeśli zmienisz nazwę people.cvs na people.csv i ponownie uruchomisz aplikację, zobaczysz nazwę pliku na ekranie. Żaden błąd. Załóżmy teraz, że dodajesz kolejną linię kodu pod instrukcją print, coś w tym stylu:

try:

Open file and shows its name.

thefile = open('people.csv')

print(thefile.name)

print(thefile.wookems())

except Exception:

print("Sorry, I don't see a file named people.csv here")

Uruchomienie tego kodu powoduje wyświetlenie następującego komunikatu:

people.csv

Sorry, I don't see a file named people.csv here

Musiał znaleźć plik, aby wydrukować nazwę pliku (co robi). Ale potem mówi, że nie może znaleźć pliku. Co więc daje? Problem leży w wierszu z wyjątkiem wyjątku: oznacza to, że „jeśli w tym bloku try zostanie zgłoszony wyjątek, wykonaj kod pod wierszem wyjątku”. Hmmm, to nie jest dobre, ponieważ błąd jest w rzeczywistości spowodowany przez odczytaną linię

print(thefile.wookems())

Ta linia zgłasza wyjątek, ponieważ w Pythonie nie ma właściwości o nazwie wookems(). Aby to wyczyścić, chcesz zastąpić wyjątek: konkretnym wyjątkiem, który chcesz przechwycić. Ale skąd wiesz, czym jest ten konkretny wyjątek? Łatwo. Wyjątek, który jest zgłaszany bez przekazywania wyjątków, to:

FileNotFoundError: [Errno 2] No such file or directory: 'people.csv'

To właśnie pierwsze słowo to nazwa wyjątku, którego możesz użyć zamiast ogólnej nazwy wyjątku, na przykład:

try:

Open file and shows its name.

thefile = open('people.csv')

print(thefile.name)

print(thefile.wookems())

except FileNotFoundError:

print("Sorry, I don't see a file named people.csv here")

To prawda, że nie pomaga to w przypadku złej nazwy metody. Ale zła nazwa metody nie jest tak naprawdę wyjątkiem, to błąd programistyczny, który należy poprawić w kodzie, zastępując .wookems() dowolną nazwą metody, której naprawdę chcesz użyć. Ale przynajmniej komunikat o błędzie, który widzisz, nie jest mylący.

Przepraszamy, ale nie widzę tutaj pliku o nazwie people.csv z błędem. Ten kod po prostu działa, więc zamiast tego pojawia się zwykły błąd — obiekt nie ma atrybutu „wookems”, jak na rysunku.

Ponownie, jeśli myślisz o obsłudze błędu .wookems, nie jest to wyjątek, dla którego napisałbyś procedurę obsługi wyjątków. Wyjątki występują, gdy coś poza programem, od którego program jest zależny, nie jest dostępne. Błędy programistyczne, takie jak nieistniejące nazwy metod, są błędami wewnątrz programu i muszą być poprawiane wewnątrz programu przez programistę, który pisze kod.

Ochrona aplikacji przed awariami

Możesz układać w stos z wyjątkiem: instrukcji w bloku try, aby obsłużyć różne błędy. Pamiętaj tylko, że gdy wystąpi wyjątek, patrzy na każdy z nich, zaczynając od góry. Jeśli znajdzie program obsługi pasujący do wyjątku, zgłasza go. Jeśli wystąpił wyjątek, którego nie obsłużyłeś, otrzymasz standardowy komunikat o błędzie Pythona. Ale na to też jest sposób.

Jeśli chcesz uniknąć wszystkich komunikatów o błędach Pythona, możesz po prostu utworzyć ostatni z wyjątkiem wyjątku: tak, aby przechwytywał każdy wyjątek, który nie został jeszcze przechwycony przez żaden poprzedzający z wyjątkiem: w kodzie. Na przykład tutaj mamy tylko dwa programy obsługi, jeden dla FileNotFound i jeden dla wszystkiego innego:

Wiem, że nie nauczyłeś się o otwieraniu, czytaniu i zamykaniu, ale nie martw się tym. Wszystko, na czym nam zależy, to obsługa wyjątków, czyli try: iz wyjątkiem: fragmentów kodu — na razie.

try:

Open file and shows its name.

thefile = open('people.csv')

Print a couple blank lines then the first line from the file.

print('\n\n', thefile.readline())

Close the file.

thefile.closed()

except FileNotFoundError:

print("Sorry, I don't see a file named people.csv here")

except Exception:

print("Sorry, something else went wrong")

Uruchomienie tego kodu daje następujące dane wyjściowe:

Username,FirstName,LastName,Role,DateJoined

Sorry, something else went wrong

Pierwszy wiersz przedstawia pierwszy wiersz tekstu z pliku people.csv. Druga linia to dane wyjściowe z drugiej, z wyjątkiem: instrukcji, która brzmi Przepraszamy, coś jeszcze poszło nie tak. Ta wiadomość jest niejasna i tak naprawdę nie pomaga w znalezieniu problemu. Zamiast po prostu drukować ogólny komunikat o nieznanym wyjątku, możesz przechwycić komunikat o błędzie w zmiennej, a następnie wyświetlić zawartość tej zmiennej, aby zobaczyć komunikat. Jak zwykle możesz nadać tej zmiennej dowolną nazwę, chociaż wiele osób używa e lub err jako skrótu oznaczającego błąd. Rozważmy na przykład następujące przepisanie poprzedniego kodu. Ogólny program obsługi, z wyjątkiem wyjątku, ma teraz as e na końcu, co oznacza „jakikolwiek wyjątek zostanie tutaj złapany, umieść komunikat o

błędzie w zmiennej o nazwie e”. Następnie następny wiersz używa print(e) do wyświetlenia wszystkiego, co jest w tej zmiennej e:

```
try:

# Open file and shows its name.

thefile = open('people.csv')

# Print a couple blank lines then the first line from the file.

print('\n\n', thefile.readline())

thefile.wigwam()

except FileNotFoundError:

print("Sorry, I don't see a file named people.csv here")

except Exception as e:

print(e)
```

Uruchomienie tego kodu powoduje wyświetlenie następującego komunikatu:

```
Username,FirstName,LastName,Role,DateJoined

'_io.TextIOWrapper' object has no attribute 'wigwam'
```

Pierwsza linia zawierająca Nazwę użytkownika, Imię itd. to tylko pierwsza linia tekstu z pliku people.csv. W kodzie nie ma błędu, a ten plik tam jest, więc wszystko poszło dobrze. Druga linia jest taka:

```
'_io.TextIOWrapper' object has no attribute 'wigwam'
```

To z pewnością nie jest zwykły angielski. Ale to lepsze niż „Coś innego poszło nie tak”. Przynajmniej część, która czyta obiekt nie ma atrybutu „wigwam”, informuje, że problem ma coś wspólnego ze słowem wigwam. Więc nadal poradziłeś sobie z błędem, a aplikacja nie „zawieszała się”. I przynajmniej uzyskałeś informacje o błędzie, które powinny być dla ciebie pomocne, nawet jeśli mogą nie być tak pomocne dla osób korzystających z aplikacji bez wiedzy o jej wewnętrznym działaniu.

Dodanie innego do miksu

Jeśli spojrzysz na kod napisany przez profesjonalnych programistów Pythona, zwykle nie mają oni zbyt dużo kodu pod try:. Dzieje się tak dlatego, że chcesz, aby bloki catch: wychwytywały określone możliwe błędy i przetwarzały na górze komunikat o błędzie w przypadku wystąpienia błędu. W przeciwnym razie chcesz, aby po prostu kontynuował z resztą kodu (który może również zawierać sytuacje generujące inne typy wyjątków). Bardziej elegancki sposób radzenia sobie z problemem wykorzystuje następującą składnię:

```
try:

The thing that might cause an exception

catch (a common exception):

Explain the problem

catch Exception as e:
```

Show the generic error message

else:

Continue on here only if no exceptions raised

So the logic with this flow is

Try to open the file...

If the file isn't there, tell them and stop.

If the file isn't there, show error and stop

Otherwise...

Go on with the rest of the code.

Ograniczając próbę: do jednej rzeczy, która najprawdopodobniej wywoła wyjątek, możemy zatrzymać działanie kodu, zanim spróbuje przejść dalej. Ale jeśli żaden wyjątek nie zostanie zgłoszony, to kontynuuje normalnie, poniżej else, gdzie poprzednie procedury obsługi wyjątków nie mają już znaczenia. Oto cały kod z komentarzami wyjaśniającymi, co się dzieje:

try:

Open the file name people.csv

thefile = open('people.csv')

Watch for common error and stop program if it happens.

except FileNotFoundError:

print("Sorry, I don't see a file named people.csv here")

Catch any unexpected error and stop program if one happens.

except Exception as err:

print(err)

Otherwise, if nothing bad has happened by now, just keep going.

else:

File must be open by now if we got here.

print('\n') # Print a blank line.

Print each line from the file.

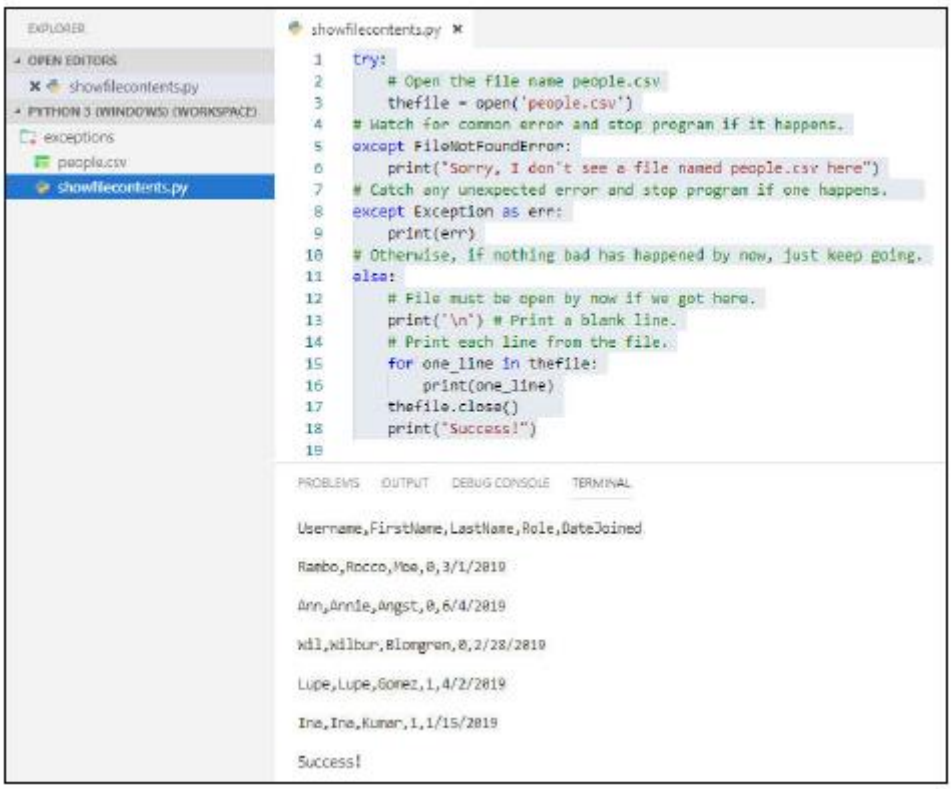
for one_line in thefile:

print(one_line)

thefile.close()

print("Success!")

Jak zawsze w Pythonie, wcięcia mają duże znaczenie. Upewnij się, że tworzysz wcięcia we własnym kodzie, jak pokazano w tym rozdziale, w przeciwnym razie Twój kod może nie działać poprawnie. Rysunek pokazuje również cały kod i wyniki uruchomienia tego kodu w VS Code.



Używając try. . . z except . . . else . . .finally

Jeśli spojrzysz na pełną składnię obsługi wyjątków w Pythonie, zobaczysz, że na końcu jest jeszcze jedna opcja, taka jak ta:

try:

try to do this

except:

if x happens, stop here

except Exception as e:

if something else bad happens, stop here

else:

if no exceptions, continue on normally here

finally:

do this code no matter what happened above

Blok finally:, jeśli jest uwzględniony, to kod, który jest uruchamiany niezależnie od tego, czy wystąpi wyjątek, czy nie. Ten wzorzec jest zwykle używany w bardziej złożonych aplikacjach, takich jak te, w których nowy fragment kodu zależny od dostępności jakiegoś zewnętrznego zasobu jest wywoływany

po przeprowadzeniu serii zdarzeń. Jeśli zasób jest dostępny, kod jest odtwarzany, ale jeśli zasób nie jest dostępny, wykonywany jest inny kod. Aby zilustrować, oto kod, który oczekuje, że zasób zewnętrzny o nazwie people.csv będzie dostępny dla kodu .

```
print('Do this first')

try:

open('people.csv')

except FileNotFoundError:

print('Cannot find file named people.csv')

except Exception as e:

print('e')

else:

print('Show this if no exception.')

finally:

print('This is in the finally block')

print("This is outside the try...except...else...finally")
```

Po uruchomieniu tego kodu z plikiem o nazwie people.csv w folderze otrzymasz następujące dane wyjściowe:

```
Do this first

Show this if no exception.

This is in the finally block

This is outside the try...except...else...finally
```

Żaden kod zgłaszania wyjątków nie został wykonany, ponieważ instrukcja open() była w stanie otworzyć plik o nazwie people.csv. Uruchom ten sam kod bez pliku o nazwie people.csv w tym samym folderze, otrzymasz następujący wynik. Tym razem kod informuje, że nie może ukarać pliku o nazwie people.csv. Ale aplikacja nie „zawiesza się”. Zamiast tego po prostu kontynuuje wykonywanie reszty kodu.

```
Do this first

Cannot find file named people.csv

This is in the finally block

This is outside the try...except...else...finally
```

Te przykłady pokazują, że możesz dokładnie kontrolować, co dzieje się w małej części programu, która jest podatna na błędy użytkownika lub inne „zewnętrzne” wyjątki, jednocześnie pozwalając na normalne działanie innego kodu

Podnoszenie własnych błędów

Python ma wiele wbudowanych wyjątków służących do rozpoznawania i identyfikowania błędów, o czym przekonasz się podczas pisania i testowania kodu, zwłaszcza gdy uczysz się go po raz pierwszy. Jednak nie jesteś do nich ograniczony. Jeśli Twoja aplikacja ma lukę, której nie obejmują wbudowane wyjątki, możesz wymyślić własną. Aby uzyskać szczegółową listę wszystkich wyjątków, które Python może przechwycić, zajrzyj na https://docs.python.org/3/library/exceptions.html w Pythonie.org documentation . Ogólna składnia zgłaszania własnego błędu to:

raise error

Zastąp błąd sformułowaniem znanego błędu, który chcesz zgłosić (na przykład FileNotFoundError). Lub, jeśli błąd nie jest objęty jednym z tych wbudowanych błędów, możesz po prostu zgłosić wyjątek, który wykona wszystko, co znajduje się pod catch Exception: w twoim kodzie. Jako działający przykład, powiedzmy, że chcesz, aby program działał pomyślnie, jeśli spełnione są dwa warunki:

* Plik people.csv musi istnieć, abyś mógł go otworzyć.

* Plik people.csv musi zawierać więcej niż jeden wiersz danych (pierwszy wiersz to tylko nazwy kolumn, a nie dane, ale jeśli zawiera tylko nagłówki kolumn, chcemy go uznać za pusty.

Oto przykład tego, jak możesz poradzić sobie z tą sytuacją, patrząc tylko na część obsługi wyjątków:

try:

Open the file (no error check for this example).

thefile = open('people.csv')

Count the number of lines in file.

line_count = len(thefile.readlines())

If there is fewer than 2 lines, raise exception.

if line_count < 2:

raise Exception

Handles missing file error.

except FileNotFoundError:

print('\nThere is no people.csv file here'

Handles all other exceptions

except Exception as e:

Show the error.

print('\n\nFailed: The error was ' str(e))

Close the file.

thefile.close()

Przejdźmy więc przez to. Pierwsze wiersze próbują otworzyć plik people.csv:

try:

Open the file (no error check for this example).

thefile = open('people.csv')

Wiemy, że jeśli plik people.csv nie istnieje, wykonanie przeskoczy do tego modułu obsługi wyjątków, który informuje użytkownika, że pliku tam nie ma

except FileNotFoundError:

print('\nThere is no people.csv file here')

Zakładając, że tak się nie stało i plik jest teraz otwarty, następna linia liczy, ile linii znajduje się w pliku:

line_count = len(thefile.readlines())

Jeśli plik jest pusty, liczba wierszy wyniesie 0. Jeśli plik zawiera tylko nagłówki kolumn, jak poniżej:

Username,FirstName,LastName,DateJoined

. . . wtedy długość będzie wynosić 1. Chcemy, aby reszta kodu działała tylko wtedy, gdy długość pliku wynosi 2 lub więcej. Więc jeśli liczba linii jest mniejsza niż 2, kod może zgłosić wyjątek. Możesz nie wiedzieć, czym jest ten wyjątek, więc każesz mu zgłosić ogólny wyjątek, taki jak ten:

if line_count < 2:

raise Exception

Procedura obsługi wyjątków w kodzie ogólnych wyjątków wygląda następująco:

Handles all other exceptions

except Exception as e:

Show the error.

print('\n\nFailed: The error was ' str(e))

Close the file.

thefile.close()

e przechwytuje rzeczywisty wyjątek, a następnie następna instrukcja print pokazuje, co to było. Załóżmy więc, że uruchamiasz ten kod, a plik people.csv jest pusty lub niekompletny. Wyjście będzie:

Failed: The error was

Zauważ, że nie ma wyjaśnienia błędu. To dlatego, że znaleziono błąd, który Python może sam rozpoznać. Zamiast tego możesz zgłosić znany wyjątek. Na przykład, zamiast zgłaszać ogólny wyjątek, możesz zgłosić błąd File NotFoundError w następujący sposób:

if line_count < 2:

raise FileNotFoundError

Ale jeśli to zrobisz, zostanie wywołana procedura obsługi FileNotFoundError i wyświetli komunikat Nie ma pliku people.csv, co w tym przypadku nie jest prawdą i nie jest przyczyną problemu. Istnieje plik people.csv; po prostu nie ma żadnych danych do przejścia. Potrzebujesz własnego niestandardowego wyjątku i obsługi tego wyjątku. Wszystkie wyjątki w Pythonie są w rzeczywistości obiektami, instancjami wbudowanej klasy o nazwie Errors in Python. Aby utworzyć własny wyjątek, musisz

najpierw zaimportować klasę Errors, aby użyć jej jako klasy bazowej (podobnie jak klasa Member była klasą bazową dla różnych typów użytkowników). Następnie pod tym definiujesz swój własny błąd jako podklasę tego. Ten kod pojawia się na górze pliku, więc jest wykonywany, zanim jakikolwiek inny kod spróbuje użyć niestandardowego wyjątku:

```
# define Python user-defined exceptions

class Error(Exception):

"""Base class for other exceptions"""

pass

# Your custom error (inherits from Error)

class EmptyFileError(Error):

pass
```

Tak jak poprzednio, słowo pass w każdej klasie mówi Pythonowi: „Wiem, że ta klasa nie zawiera żadnego kodu i tutaj jest to w porządku. Nie musisz zgłaszać wyjątku, żeby mi to powiedzieć. Teraz, gdy istnieje wyjątek o nazwie EmptyFileError, możesz zgłosić ten wyjątek, gdy plik ma niewystarczającą zawartość. Następnie napisz program obsługi, który obsłuży ten wyjątek. Oto ten kod:

```
# If there is fewer than 2 lines, raise exception.

if line_count < 2:

raise EmptyFileError

# Handles my custom error for too few rows.

except EmptyFileError:

print("\nYour people.csv file doesn't have enough stuff.")
```

Oto jak sytuacja się potoczy, gdy kod zostanie uruchomiony. Jeśli w ogóle nie ma pliku people.csv, pojawia się ten błąd

```
There is no people.csv file here
```

Jeśli istnieje plik people.csv, ale jest on pusty lub zawiera tylko nagłówki kolumn, program pokazuje tylko to:

```
Your people.csv file doesn't have enough stuff.
```

Zakładając, że nie wystąpił żaden błąd, kod pod else: działa i pokazuje na ekranie wszystko, co znajduje się w pliku. Jak widać, obsługa wyjątków pozwala z wyprzedzeniem planować błędy spowodowane przez luki w kodzie. Nie zajmujemy się tutaj błędami w twoim kodzie ani błędami w kodowaniu. Ogólnie mówimy o zasobach zewnętrznych, których program potrzebuje do prawidłowego działania. Kiedy brakuje tych zewnętrznych zasobów lub są one niewystarczające, nie musisz pozwolić, aby program po prostu „zawieszał się” i wyświetlał na ekranie komunikat o błędzie nerd-o-rama, aby zbić użytkowników z tropu. Zamiast tego możesz przechwycić wyjątek i pokazać im tekst, który dokładnie mówi im, co jest nie tak, co z kolei pomoże im rozwiązać ten problem i ponownie uruchomić program, tym razem pomyślnie. Na tym polega obsługa wyjątków.