Przemierzanie ogromnych danych za pomocą słowników

Słowniki danych, w niektórych językach nazywane także tablicami asocjacyjnymi, przypominają listy. Jednak zamiast identyfikowania każdego elementu na podstawie jego pozycji na liście, każdy element jest jednoznacznie identyfikowany za pomocą klucza. Możesz samodzielnie zdefiniować klucz, którym może być ciąg znaków lub liczba. Liczy się tylko to, aby był unikalny dla każdego elementu w słowniku. Aby zrozumieć, dlaczego wyjątkowość ma znaczenie, pomyśl o numerach telefonów, adresach e-mail i numerach ubezpieczenia społecznego. Gdyby dwie lub więcej osób miało ten sam numer telefonu, to za każdym razem, gdy ktoś dzwoniłby na ten numer, wszystkie te osoby otrzymywałyby połączenie. Gdyby dwie lub więcej osób miało ten sam adres e-mail, wszystkie te osoby otrzymywałyby te same wiadomości e-mail. Jeśli dwie lub więcej osób miało ten sam numer ubezpieczenia społecznego, a jedna z tych osób zalegała z podatkami o milion dolarów, lepiej miej nadzieję, że uda ci się przekonać podatników, że to nie ty jesteś winien milion dolarów, mimo że twój Numer ubezpieczenia społecznego jest na zaległym rachunku. Klucz w słowniku reprezentuje jedną unikalną rzecz i możesz powiązać wartość z tym kluczem. Wartością może być liczba, ciąg znaków, lista, krotka — właściwie wszystko. Możesz więc myśleć o słowniku danych jako o czymś w rodzaju tabeli, w której pierwsza kolumna zawiera pojedynczy element informacji, który jest unikalny dla tego elementu, a druga kolumna, wartość, zawiera informacje, które są istotne i być może unikalne dla tego elementu klucz. W przykładzie na rysunku lewa kolumna zawiera klucz, który jest unikalny dla każdego wiersza. Druga kolumna to wartość przypisana do każdego klucza.

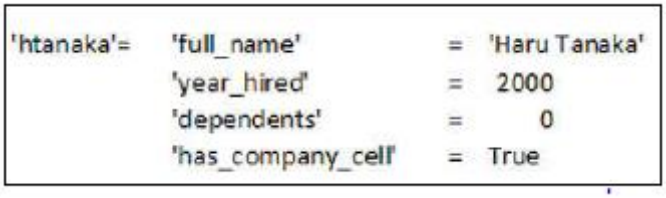
Key		Value
"htanaka"	=	"Haru Tanaka"
"ppatel"	=	"Priya Patel"
"bagarcia"	=	"Benjamin Alberto Garcia"
"zmin"	=	"Zhang Min"
"farooqi"	=	"Ayesha Farooqi"
"hajackson"	=	"Hanna Jackson"
"papatel"	=	"Pratyush Aarav Patel"
"hrjackson"	=	"Henry Jackson"

W lewej kolumnie znajduje się skrót nazwy osoby. Niektóre firmy używają takich nazw, aby przydzielać swoim pracownikom konta użytkowników i adresy e-mail. Wartość klucza nie musi być ciągiem znaków ani liczbą całkowitą. Może to być lista, krotka lub zestaw. Na przykład w słowniku przedstawionym na rysunku

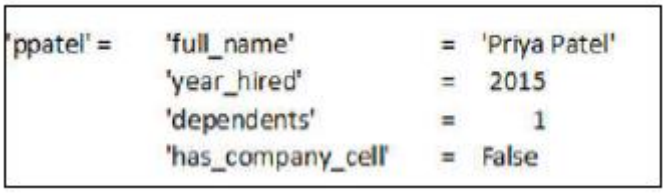
Key		Value
"htanaka"	=	["Haru Tanaka", 2000, 0, True]
"ppatel"	=	["Priya Patel", 2015, 1, False]
"bagarcia"	=	["Benjamin Alberto Garcia", 1999, 2, True]
"zmin"	=	["Zhang Min", 2017, 0, False]
"farooqi"	=	["Ayesha Farooqi", 2001, 1, True]
"hajackson"	=	["Hanna Jackson", 1998, 0, False]
"papatel"	=	["Pratyush Aarav Patel", 2011, 2, True]
"hrjackson"	=	["Henry Jackson", 2016, 0, False]

wartość każdego klucza obejmuje imię i nazwisko, rok (być może rok zatrudnienia lub rok urodzenia), liczbę (która może być liczbą osób na utrzymaniu, na których dana osoba rozlicza się z podatków), oraz

wartość logiczną True lub False (która może wskazywać, czy mają służbowy telefon komórkowy). Na razie nie ma znaczenia, co reprezentuje każdy element danych. Liczy się to, że dla każdego klucza masz listę (ujętą w nawiasy kwadratowe), która zawiera cztery informacje o każdym kluczu. Słownik może również składać się z kilku różnych kluczy, z których każdy reprezentuje fragment danych. Na przykład, zamiast mieć wiersz dla każdej pozycji z unikalnym kluczem, możesz stworzyć każdemu pracownikowi własny mały słownik, a następnie możesz przypisać nazwę klucza do każdej jednostki informacji. Słownik htanaka może więc wyglądać tak, jak na rysunku.



Słownik dla innego pracownika może mieć wszystkie te same nazwy kluczy, full_name, year_hired, ependents, i has_company_cell , ale różne wartości dla każdego z tych kluczy.



Każdy słownik posiadający wiele kluczy jest powszechny w Pythonie, ponieważ język ten ułatwia wyizolowanie konkretny element danych, który chcesz, używając składni object.key, takiej jak ta:

ppatel.full_name = 'Priya Patel'

ppatel.year_hired= 2015

ppatel,dependents = 1

ppatel.has_company_cell=True

Nazwa klucza jest bardziej opisowa niż użycie indeksu opartego na pozycji, jak widać w poniższym przykładzie.

ppatel[0] = 'Priya Patel'

ppatel[1]= 2015

ppatel[2] = 1

ppatel[3]=True

Tworzenie słownika danych

Kod do tworzenia słownika danych jest zgodny z podstawową składnią:

name = {key:value, key:value, key:value, key:value, ...}

Nazwa to wymyślona przez Ciebie nazwa, która ogólnie opisuje, do kogo lub do czego odnoszą się pary klucz-wartość. Pary klucz:wartość są ujęte w nawiasy klamrowe. Kluczowe wartości to zwykle łańcuchy ujęte w cudzysłowy, ale jeśli chcesz, możesz zamiast nich użyć liczb całkowitych. Każdy dwukropek (:)

oddziela nazwę klucza od przypisanej mu wartości. Wartością jest to, co chcesz zapisać dla tej nazwy klucza i może to być liczba, ciąg, lista. . . prawie wszystko. Wielokropek (...) oznacza po prostu, że możesz mieć dowolną liczbę par klucz-wartość. Pamiętaj tylko, aby oddzielić pary klucz-wartość przecinkami, jak pokazano w przykładzie ze składnią. Aby kod był bardziej czytelny, programiści często umieszczają każdą parę klucz-wartość w osobnej linii. Ale składnia jest wciąż taka sama. Jedyna różnica polega na tym, że po każdym przecinku następuje podział wiersza, jak w poniższym przykładzie:

```
name = {

key:value,

key:value,

key:value,

key:value,

...

}
```

Jeśli chcesz wypróbować to w praktyce, otwórz notatnik Jupyter, plik .py lub monit Pythona i wpisz następujący kod. Zauważ, że stworzyliśmy słownik o nazwie people. Zawiera wiele par klucz-wartość, z których każda jest oddzielona przecinkiem. Klucze i wartości są łańcuchami, więc są ujęte w cudzysłowy, a każdy klucz jest oddzielony od wartości dwukropkiem. Ważne jest, aby zachować to wszystko prosto, w przeciwnym razie kod nie zadziała - tak, nawet jeden brakujący lub źle umieszczony lub błędnie wpisany cudzysłów, dwukropek, przecinek lub nawias klamrowy może zepsuć całość.

```
people = {

'htanaka': 'Haru Tanaka',

'ppatel': 'Priya Patel',

'bagarcia': 'Benjamin Alberto Garcia',

'zmin': 'Zhang Min',

'afarooqi': 'Ayesha Farooqi',

'hajackson': 'Hanna Jackson',

'papatel': 'Pratyush Aarav Patel',

'hrjackson': 'Henry Jackson'

}
```

Dostęp do danych słownika

Po dodaniu danych można z nimi pracować na wiele sposobów. Używając funkcji print(people) - czyli funkcji print() z nazwą słownika w nawiasach — otrzymujesz kopię całego słownika w następujący sposób:

```
print(people)

{'htanaka': 'Haru Tanaka', 'ppatel': 'Priya Patel', 'bagarcia': 'Benjamin
```

Alberto Garcia', 'zmin': 'Zhang Min', 'afarooqi': 'Ayesha Farooqi',

'hajackson': 'Hanna Jackson', 'papatel': 'Pratyush Aarav Patel',

'hrjackson': 'Henry Jackson'}

Zwykle nie jest to to, czego chcesz. Częściej szukasz jednego konkretnego elementu w słowniku. W takim przypadku użyj tej składni:

dictionaryname[key] nazwa słownika [klucz]

gdzie nazwa_słownika to nazwa słownika, a klucz to szukana wartość klucza. Na przykład, jeśli chcesz poznać wartość klucza zmin, wpisz

print(people['zmin'])

Pomyśl o tym wierszu jako mówiącym print people sub zmin, gdzie sub oznacza po prostu określony klucz. Gdy to zrobisz, Python zwróci wartość dla tej jednej osoby. . . pełna nazwa zmin w tym przykładzie. Rysunek przedstawia dane wyjściowe po uruchomieniu kodu w komórce notesu Jupyter.

Zauważ, że w kodzie zmin jest w cudzysłowie. Są one wymagane, ponieważ zmin jest ciągiem znaków. Zamiast tego możesz użyć nazwy zmiennej, o ile ta zmienna zawiera łańcuch. Rozważmy na przykład następujące dwa wiersze kodu. Pierwszy tworzy zmienną o nazwie osoba i umieszcza w niej ciąg „zmin”. Następna linia, print(people[person]) (print people sub person) nie wymaga cudzysłowów, ponieważ person jest nazwą zmiennej.

person = 'zmin'

print(people[person])

Jak myślisz, co by się stało, gdybyś wykonał poniższy kod?

person = 'hrjackson'

print(people[person])

Oczywiście zobaczyłbyś Henry'ego Jacksona, nazwisko (wartość) towarzyszące kluczowi „hrjackson”. A gdybyś uruchomił ten fragment kodu?

person = 'schmeedledorp'

print(people[person])

Rysunek

pokazuje, co by się stało. Pojawia się błąd, ponieważ nic w słowniku osób nie ma wartości klucza „schmeedledorp”.

Pobieranie długości słownika

Liczba pozycji w słowniku jest uważana za jego długość. Podobnie jak w przypadku list, możesz użyć instrukcji len() do określenia długości słownika. Składnia to:

len(dictionaryname)len(nazwa słownika)

Jak zawsze, zamień nazwę słownika na nazwę sprawdzanego słownika. Na przykład poniższy kod tworzy słownik, a następnie przechowuje jego długość w zmiennej o nazwie howmany:

people = {

'htanaka': 'Haru Tanaka',

'ppatel': 'Priya Patel',

'bagarcia': 'Benjamin Alberto Garcia',

zmin': 'Zhang Min',

'afarooqi': 'Ayesha Farooqi',

'hajackson': 'Hanna Jackson',

'papatel': 'Pratyush Aarav Patel',

'hrjackson': 'Henry Jackson'

}

Count the number of key:value pairs and put in a variable.

howmany = len(people)

Show how many.

print(howmany)

Po wykonaniu instrukcja print pokazuje 8, wartość zmiennej hominy, określoną przez liczbę par klucz-wartość w słowniku. Jak można się domyślić, pusty słownik, który nie zawiera par klucz-wartość, ma długość równą zero.

Sprawdzanie, czy klucz istnieje w słowniku

Możesz użyć słowa kluczowego in, aby sprawdzić, czy klucz istnieje. Jeśli klucz istnieje, to in zwraca True. Jeśli klucz nie istnieje, in zwraca False. Rysunek

```
# Make a data dictionary named people
people = {
    'htanaka': 'Haru Tanaka',
    'ppatel': 'Priya Patel',
    'bagarcia': 'Benjamin Alberto Garcia',
    'zmin': 'Zhang Min',
    'afarooqi': 'Ayesha Farooqi',
    'hajackson': 'Hanna Jackson',
    'papatel': 'Pratyush Aarav Patel',
    'hrjackson': 'Henry Jackson'
    }

# Is there an hajackson in the people dictionary?
print('hajackson' in people)

# Is there an schmeedledorp in the people dictionary?
print('schmeedledorp' in people)
```

```
True
False
```

przedstawia prosty przykład z dwiema instrukcjami print(). Pierwszy sprawdza, czy hajackson istnieje w słowniku. Drugi sprawdza, czy schmeedledorp istnieje w słowniku.

Jak widać, pierwsza instrukcja print() ma wartość True, ponieważ hajackson znajduje się w słowniku. Drugi zwraca False, ponieważ schmeedledorp nie występuje w słowniku.

Pobieranie danych ze słownika za pomocą funkcji get()

Awaria i palenie programu, gdy szukasz czegoś, czego nie ma w słowniku, jest trochę trudne. Bardziej eleganckim sposobem obsługi jest użycie metody .get() ze słownika danych. Składnia to:

dictionaryname.get(key)

Zamień nazwę słownika na nazwę szukanego słownika. Wymień klucz na rzecz, której szukasz. Zauważ, że funkcja get() używa nawiasów okrągłych, a nie kwadratowych. Jeśli szukasz czegoś, co jest w słowniku, na przykład tego:

Look for a person.

person = 'bagarcia'

print(people.get(person))

. . . uzyskasz taki sam wynik, jak przy użyciu nawiasów kwadratowych. Tym, co wyróżnia .get(), jest to, co dzieje się, gdy szukasz nieistniejącej nazwy. Nie pojawia się błąd, a program nie tylko się zawiesza i nie pali. Zamiast tego funkcja get() z wdziękiem zwraca słowo None, aby poinformować, że w słowniku osób nie ma osoby o nazwisku schmeedledorp. Rysunek przedstawia przykład.

W rzeczywistości możesz przekazać dwie wartości funkcji get(), z których druga to wartość, którą chcesz zwrócić, jeśli funkcja get nie znajdzie tego, czego szukasz. Na przykład w poniższym wierszu kodu ponownie wyszukujemy schmeedledorp, ale tym razem, jeśli nie znajdzie tej osoby, nie wyświetla słowa Brak. Zamiast tego wyświetla bardziej pompatyczny komunikat Unbeknownst to this dictionary.

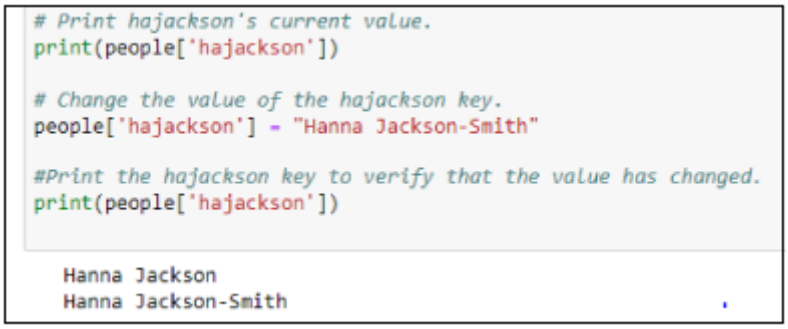
print(people.get('schmeedledorp','Unbeknownst to this dictionary'))

Zmiana wartości klucza

Słowniki są zmienne, co oznacza, że możesz zmienić zawartość słownika z poziomu kodu (nie możesz zamknąć słownika). Składnia jest prosta:

dictionaryname[key] = newvalue

Zastąp Dictionaryname nazwą słownika, key kluczem identyfikującym ten element, a newvalue dowolną nową wartością. Na przykład przypuszczalna Hanna Jackson wychodzi za mąż i zmienia nazwisko na Hanna Jackson-Smith. Chcesz zachować ten sam klucz, po prostu zmień wartość. Linia, która brzmi tak naprawdę people['hajackson'] = "Hanna Jackson-Smith" dokonuje zmiany. Instrukcja print() pod tym wierszem pokazuje wartość hajacksona po wykonaniu tego wiersza kodu. Jak widać, to imię rzeczywiście zostało zmienione na Hana Jackson-Smith.



W prawdziwym życiu dane w słowniku prawdopodobnie byłyby również przechowywane w jakimś zewnętrznym pliku, więc jest to trwałe. Do zapisania zmian w słowniku w tym pliku zewnętrznym wymagany byłby dodatkowy kod. Ale musisz nauczyć się tych podstaw, zanim przejdziesz do tego wszystkiego. Więc na razie posuwajmy się naprzód ze słownikami.

Dodawanie lub zmiana danych słownika

Możesz użyć metody Dictionary update() w celu dodania nowego elementu do słownika lub zmiany wartości bieżącego klucza. Składnia to

dictionaryname.update(key, value)

Zamień dictionaryname nazwę słownika na nazwę dictionary. Zastąp key kluczem elementu, który chcesz dodać lub zmienić. Jeśli podany klucz nie istnieje jeszcze w słowniku, zostanie dodany jako nowy element z określoną przez Ciebie wartością. Jeśli określony klucz istnieje, nic nie zostanie dodane. Wartość klucza zostanie zmieniona na dowolną wartość, którą określisz jako value. Rozważmy na przykład następujący kod Pythona, który tworzy słownik danych o nazwie people i umieszcza w nim imiona dwóch osób:

Make a data dictionary named people.

people = {

'papatel': 'Pratyush Aarav Patel',

'hrjackson': 'Henry Jackson'

}

Change the value of the hrjackson key.

people.update({'hrjackson':'Henrietta Jackson'})

print(people)

Update the dictionary with a new property:value pair.

people.update({'wwiggins':'Wanda Wiggins'})

Pierwsza linia aktualizacji pokazana poniżej. . .

people.update({'hrjackson':'Henrietta Jackson'})

. . . zmienia wartość dla hrjackson z Henry Jackson na 'Henrietta Jackson. Zmienia istniejącą nazwę, ponieważ klucz hrjackson już istnieje w słowniku danych. Druga update() brzmi następująco:

people.update({'wwiggins':'Wanda Wiggins'})

W słowniku nie ma klucza wwiggins. Więc update() nie może zmienić nazwy dla wwiggins. Zamiast tego ta linia dodaje do słownika nową parę klucz-wartość z kluczem wwigins i wartością Wanda Wiggins. Kod nie określa, czy zmienić, czy dodać wartość. Nie musi, ponieważ decyzja jest podejmowana automatycznie. Każdy klucz w słowniku musi być unikalny; nie możesz mieć dwóch lub więcej wierszy z tym samym kluczem. Kiedy więc wykonujesz funkcję update(), najpierw sprawdza ona, czy klucz już istnieje. Jeśli tak, to tylko wartość tego klucza jest modyfikowana; nic nowego nie jest dodawane. Jeśli klucz nie istnieje jeszcze w słowniku, nie ma nic do modyfikacji, więc cała nowa wartość klucza jest dodawana do słownika. Jest to automatyczne, a decyzja o tym, którą czynność wykonać, jest prosta:

* Jeśli klucz już istnieje w słowniku, to jego wartość jest aktualizowana, ponieważ żadne dwa elementy w słowniku nie mogą mieć tego samego klucza.

* Jeśli klucz jeszcze nie istnieje, dodawana jest para klucz-wartość, ponieważ w słowniku nie ma niczego, co by zawierało ten klucz, więc jedynym wyjściem jest dodanie go.

Po uruchomieniu powyższego kodu słownik zawiera trzy pozycje: paptel, hrjackson (z nową nazwą) i wwiggins. Dodanie następujących wierszy na końcu tego kodu wyświetla wszystko w słowniku:

Show what's in the data dictionary now.

for person in people.keys():

print(person + "=" + people[person])

Jeśli dodasz ten kod i uruchomisz go ponownie, otrzymasz poniższe dane wyjściowe, które pokazują pełną zawartość słownika danych na końcu tego programu:

papatel = Pratyush Aarav Patel

hrjackson = Henrietta Jackson

wwiggins = Wanda Wiggins

Jak zapewne się domyślasz, możesz przechodzić przez słownik w taki sam sposób, jak przeglądasz listy, krotki i zbiory. Ale jest kilka dodatkowych rzeczy, które możesz zrobić ze słownikami, więc spójrzmy na te następne.

Zapętlanie słownika

Możesz przeglądać każdy element w słowniku w taki sam sposób, jak przeglądasz listy i krotki. Ale masz kilka dodatkowych opcji. Jeśli po prostu określisz nazwę słownika w pętli for, otrzymasz wszystkie klucze w następujący sposób:

for person in people:

print(person)

htanaka

ppatel

bagarcia

zmin

afarooqi

hajackson

papatel

hrjackson

Jeśli chcesz zobaczyć wartość każdego elementu, zachowaj tę samą pętlę for, ale wypisz nazwę słownika [klucz], gdzie nazwa słownika to nazwa słownika (ludzie w naszym przykładzie), a klucz to nazwa, której użyjesz zaraz po for w pętli (osoba w poniższym przykładzie).

for person in people:

print(people[person])

Uruchomienie tego kodu w przykładowym słowniku osób powoduje wyświetlenie wszystkich nazwisk w następujący sposób:

Haru Tanaka

Priya Patel

Benjamin Alberto Garcia

Zhang Min

Ayesha Farooqi

Hanna Jackson

Pratyush Aarav Patel

Henry Jackson

Możesz także uzyskać wszystkie nazwy, używając nieco innej składni w pętli for: . . . dodaj .values() do nazwy słownika, jak pokazano poniżej. Następnie możesz po prostu wydrukować nazwę zmiennej (osoba), a nadal będziesz widzieć wartość przed przeglądaniem wartości.

for person in people.values():

print(person)

Wreszcie, możesz przeglądać klucze i wartości w tym samym czasie, używając .items() po nazwie słownika w pętli for. Ale będziesz potrzebować również dwóch zmiennych po for, jednej do odniesienia do klucza, drugiej do odniesienia do wartości. Jeśli chcesz zobaczyć oba podczas przeglądania, musisz użyć tych samych nazw w nawiasach wydruku. Na przykład pętla z rysunku

```
# Make a data dictionary named people
people = {
    'htanaka': 'Haru Tanaka',
    'ppatel': 'Priya Patel',
    'bagarcia': 'Benjamin Alberto Garcia',
    'zmin': 'Zhang Min',
    'afarooqi': 'Ayesha Farooqi',
    'hajackson': 'Hanna Jackson',
    'papatel': 'Pratyush Aarav Patel',
    'hrjackson': 'Henry Jackson'
    }

# Loop through .items to get the key and the value.
for key, value in people.items():
    # Show the key and value with = in between.
    print(key, "=", value)
```

```
htanaka = Haru Tanaka
ppatel = Priya Patel
bagarcia = Benjamin Alberto Garcia
zmin = Zhang Min
afarooqi = Ayesha Farooqi
hajackson = Hanna Jackson
papatel = Pratyush Aarav Patel
hrjackson = Henry Jackson
```

wykorzystuje dwie nazwy zmiennych, klucz i wartość (chociaż mogą to być x i y lub cokolwiek innego), aby przechodzić przez people.items(). Instrukcja print wyświetla zarówno klucz, jak i wartość przy każdym przejściu przez pętlę. W tym przykładzie funkcja print() ma również dosłowny znak równości (ujęty w cudzysłowy), aby oddzielić klucz od wartości. Jak widać na wyjściu, otrzymujesz listę wszystkich kluczy, po których następuje znak równości i wartość przypisana do tego klucza.

Metody słownika danych

Jeśli pilnie śledziłeś rozdział po rozdziale, być może zauważyłeś, że niektóre metody słowników danych wyglądają podobnie do tych stosowanych w przypadku list, krotek i zbiorów. Więc może teraz byłby dobry moment, aby wymienić wszystkie metody oferowane przez słowniki do wykorzystania w przyszłości. Widzieliście już kilka zastosowań w tej części. Do pozostałych docieramy nieco później.

Metoda: Co robi

clear() : Opróżnia słownik, usuwając wszystkie klucze i wartości.

copy() : Zwraca kopię słownika.

fromkeys() : Zwraca nową kopię słownika, ale tylko z określonymi kluczami i wartościami.

get() : Zwraca wartość określonego klucza lub Brak, jeśli nie istnieje.

items() : Zwraca listę elementów jako krotkę dla każdej pary klucz-wartość.

keys() : Zwraca listę wszystkich kluczy w słowniku.

pop() : Usuwa element określony przez klucz ze słownika i zapisuje go w zmiennej.

popitem() : Usuwa ostatnią parę klucz-wartość.

setdefault() : Zwraca wartość określonego klucza. Jeśli klucz nie istnieje: wprowadź klucz o określonej wartości.

update() : Aktualizuje wartość istniejącego klucza lub dodaje nową parę klucz-wartość, jeśli określonego klucza nie ma jeszcze w słowniku.

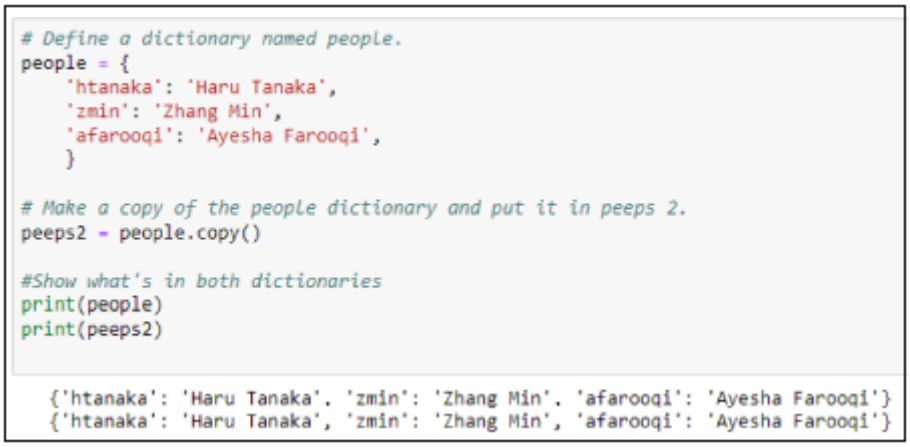
values() : Zwraca listę wszystkich wartości w słowniku.

Kopiowanie słownika

Jeśli musisz utworzyć kopię słownika danych do pracy, użyj następującej składni:

newdictionaryname = dictionaryname.copy()

Zastąp newdictionaryname dowolną nazwą nowego słownika. Zastąp nazwę słownika nazwą istniejącego słownika, który chcesz skopiować. Rysunek



przedstawia prosty przykład, w którym utworzyliśmy słownik o nazwie people. Następnie tworzymy kolejny słownik o nazwie peeps2 jako kopię tego słownika osób. Wydrukowanie zawartości każdego słownika pokazuje, że są one rzeczywiście identyczne.

Usuwanie elementów słownika

Istnieje kilka sposobów usuwania danych ze słowników danych. Słowo kluczowe del (skrót od delete) może usunąć dowolny element na podstawie jego klucza. Składnia jest następująca:

del dictionaryname[key]

Na przykład poniższy kod tworzy słownik o nazwie people. Następnie używa del people["zmin"], aby usunąć element, którego kluczem jest zmin. Późniejsze wydrukowanie zawartości słownika pokazuje, że zmin nie ma już w tym słowniku.

Define a dictionary named people.

people = {

'htanaka': 'Haru Tanaka',

'zmin': 'Zhang Min',

'afarooqi': 'Ayesha Farooqi',

}

Show original people dictionary.

```
print(people)

# Remove zmin from the dictionary.

del people["zmin"]

#Show what's in people now.

print(people)
```

Oto wynik działania tego programu:

```
{'htanaka': 'Haru Tanaka', 'zmin': 'Zhang Min', 'afarooqi': 'Ayesha Farooqi'}

{'htanaka': 'Haru Tanaka', 'afarooqi': 'Ayesha Farooqi'}
```

Jeśli zapomnisz dołączyć określony klucz do słowa kluczowego del i podasz tylko nazwę słownika, to cały słownik zostanie usunięty, nawet jego nazwa. Na przykład, jeśli wykonasz del people zamiast używać del people["zmin"] w poprzednim kodzie, wynik drugiego print(people) będzie błędem, jak poniżej, ponieważ po usunięciu słownika people no już istnieje i nie można wyświetlić jego zawartości.

```
{'htanaka': 'Haru Tanaka', 'zmin': 'Zhang Min', 'afarooqi': 'Ayesha Farooqi'}

---------------------------------------------------------

NameError Traceback (most recent call last)

<ipython-input-32-24401f5e8cf0> in <module>()

13

14 #Show what's in people now.

---> 15 print(people)

NameError: name 'people' is not defined
```

Aby usunąć wszystkie pary klucz-wartość ze słownika bez usuwania całego słownika, użyj metody clear o następującej składni:

```
dictionaryname.clear()
```

Poniższy kod tworzy słownik o nazwie people, umieszcza w nim kilka par klucz-wartość, a następnie drukuje ten słownik, aby można było zobaczyć jego zawartość. Potem ludzie. clear() usuwa wszystkie dane.

```
# Define a dictionary named people.

people = {

'htanaka': 'Haru Tanaka',

'zmin': 'Zhang Min',

'afarooqi': 'Ayesha Farooqi',

}

# Show original people dictionary.
```

print(people)

Remove all data from the dictionary.

people.clear()

#Show what's in people now.

print(people)

Wynik działania tego kodu pokazuje, że początkowo słownik danych osób zawiera trzy pary właściwość:wartość. Po użyciu people.clear() do wyczyszczenia go, wydrukowanie słownika osób wyświetla {}, co jest sposobem Pythona na poinformowanie nas, że słownik jest pusty.

{'htanaka': 'Haru Tanaka', 'zmin': 'Zhang Min', 'afarooqi': 'Ayesha Farooqi'}

{}

Używanie funkcji pop() ze słownikami danych

Metoda pop() oferuje inny sposób usuwania danych ze słownika danych. Właściwie robi dwie rzeczy:

* Jeśli zapiszesz wyniki funkcji pop() w zmiennej, ta zmienna otrzyma wartość wyskakującego klucza.

* Niezależnie od tego, czy przechowujesz wynik pop w zmiennej, określony klucz jest usuwany ze słownika.

Na rysunku

```
# Define a dictionary named people.
people = {
    'htanaka': 'Haru Tanaka',
    'zmin': 'Zhang Min',
    'afarooqi': 'Ayesha Farooqi',
    }
# Show original people dictionary.
print(people)

# Pop zmin from the dictionary, store its value in adios variable.
adios = people.pop("zmin")

# Print the contents of adios and people.
print(adios)
print(people)
```

```
{'htanaka': 'Haru Tanaka', 'zmin': 'Zhang Min', 'afarooqi': 'Ayesha Farooqi'}
Zhang Min
{'htanaka': 'Haru Tanaka', 'afarooqi': 'Ayesha Farooqi'}
```

pokazano przykład, w którym na wyjściu najpierw widzisz cały słownik. Następnie wykonywany jest adios = people.pop("zmin"), umieszczając wartość klucza zmin w zmiennej o nazwie adios. Wydrukowanie zmiennej (adios) pokazuje, że zmienna adios rzeczywiście zawiera Zhang Min, wartość zmiennej zmin. Ponowne wydrukowanie całego słownika ludzi dowodzi, że zmin został usunięty ze słownika.

Słowniki danych oferują odmianę metody pop(), która wykorzystuje następującą składnię:

dictionaryname = popitem()

Ten jest trudny, ponieważ w niektórych wcześniejszych wersjach Pythona usuwałby losowo niektóre elementy. To trochę dziwne, chyba że piszesz grę lub coś w tym stylu i rzeczywiście chcesz losowo usuwać rzeczy. Jednak począwszy od wersji Pythona 3.7 (wersja używana w tej książce) funkcja

popitem() zawsze usuwa ostatnią parę klucz-wartość. Jeśli przechowujesz wyniki popitem w zmiennej, nie otrzymujesz wartości tego elementu, co różni się od sposobu działania pop(). Zamiast tego otrzymujesz zarówno klucz, jak i jego wartość. Słownik nie zawiera już tej pary klucz-wartość. Innymi słowy, jeśli zamienisz adios = people.pop("zmin") w Figure na adios = people. popitem(), wynik jest następujący:

{'htanaka': 'Haru Tanaka', 'zmin': 'Zhang Min', 'afarooqi': 'Ayesha Farooqi'}

('afarooqi', 'Ayesha Farooqi')

{'htanaka': 'Haru Tanaka', 'zmin': 'Zhang Min'

Zabawa ze słownikami wieloklawiszowymi

Do tej pory pracowałeś ze słownikiem, który ma jedną wartość (imię osoby) dla każdego klucza (skrót imienia tej osoby). Jednak nie jest niczym niezwykłym, że słownik ma wiele par klucz-wartość dla jednego elementu danych. Załóżmy na przykład, że sama znajomość pełnego imienia i nazwiska osoby nie wystarczy. Chcesz również wiedzieć, w którym roku został zatrudniony, datę urodzenia i czy pracownikowi wydano firmowego laptopa. Słownik dla dowolnej osoby może wyglądać mniej więcej tak:

employee = {

'name' : 'Haru Tanaka',

'year_hired' : 2005,

'dob' : '11/23/1987',

'has_laptop' : False

}

Załóżmy, że potrzebujesz słownika produktów, które sprzedajesz. Dla każdego produktu, który chcesz znać jego nazwę, jego cenę jednostkową, czy podlega opodatkowaniu i ile masz obecnie w magazynie. Słownik może wyglądać mniej więcej tak (dla jednego produktu):

product = {

'name' : 'Ray-Ban Wayfarer Sunglasses',

'unit_price' : 112.99,

'taxable' : True,

'in_stock' : 10

}

Zwróć uwagę, że w każdym przykładzie nazwa klucza jest zawsze ujęta w cudzysłów. Ujęliśmy nawet datę w dob (data urodzenia) w cudzysłowie. Jeśli tego nie zrobisz, może to być traktowane jako zestaw liczb, jak w przypadku „11 podzielone przez 23 podzielone przez 1987”, co nie jest przydatną informacją. Logiczne są albo Prawdą, albo Fałszem (początkowe wielkie litery) bez nr cudzysłowów. Liczby całkowite (2005, 10) i liczby zmiennoprzecinkowe (112,99) również nie są ujęte w cudzysłowy. Ważne jest, aby upewnić się, że zawsze robisz to poprawnie, jak pokazano w powyższych przykładach. Wartością właściwości może być lista, krotka lub zestaw; nie musi to być pojedyncza wartość. Na

przykład w przypadku okularów przeciwsłonecznych możesz zaoferować dwa modele (kolorowe), czarny i szylkretowy. Możesz dodać kolory lub klucz modelu i wymienić elementy jako listę oddzieloną przecinkami w nawiasach kwadratowych w następujący sposób:

```
product = {

'name' : 'Ray-Ban Wayfarer Sunglasses',

'unit_price' : 112.99,

'taxable' : True,

'in_stock' : 10,

'models' : ['Black', 'Tortoise']

}
```

Następnie przyjrzyjmy się, jak możesz wyświetlić dane słownika. Możesz użyć prostej składni nazwa_słownika[klucz], aby wydrukować tylko wartość każdego klucza. Na przykład, używając tego ostatniego przykładu produktu, dane wyjściowe tego kodu:

```
print(product['name'])

print(product['unit_price'])

print(product['taxable'])

print(product['in_stock'])

print(product['models'])
```

. . . byłoby:

```
Ray-Ban Wayfarer Sunglasses

112.99

True

10

['Black', 'Tortoise']
```

Możesz to wymyślić, dodając opisowy tekst do każdej instrukcji print, po którym następuje przecinek i kod. Możesz także przejść przez listę, aby wydrukować każdy model w osobnej linii. I możesz użyć f-stringu do sformatowania dowolnych danych, jeśli chcesz. Oto przykładowa odmiana instrukcji print() pokazanej powyżej:

```
product = {

'name' : 'Ray-Ban Wayfarer Sunglasses',

'unit_price' : 112.99,

'taxable' : True,

'in_stock' : 10,

'models' : ['Black', 'Tortoise']
```

```
}

print('Name: ', product['name'])

print('Price: ',f"${product['unit_price']:.2f}")

print('Taxable: ',product['taxable'])

print('In Stock:',product['in_stock'])

print('Models:')

for model in product['models']:

print(" " * 10 model)
```

Oto wynik tego kodu:

```
Name: Ray-Ban Wayfarer Sunglasses

Price: $112.99

Taxable: True

In Stock: 10

Models:

Black

Tortoise
```

„ ” * 10 w ostatnim wierszu kodu mówi o wydrukowaniu spacji („ ”) dziesięć razy. Innymi słowy, wcięcie dziesięć spacji. Jeśli nie umieścisz dokładnie jednej spacji między tymi cudzysłowami, nie otrzymasz 10 spacji. Otrzymasz 10 z tego, co jest między cudzysłowami, co oznacza również, że nie dostaniesz nic, jeśli nie umieścisz niczego między cudzysłowami.

Korzystanie z tajemniczych metod fromkeys i setdefault

Słowniki danych w Pythonie oferują dwie metody, nazwane fromkeys() i setdefault(), które są przyczyną wielu problemów wśród uczących się Pythona – i słusznie, ponieważ nie jest łatwo znaleźć praktyczne zastosowania do ich użycia. Ale spróbujemy to zrobić i przynajmniej pokażemy dokładnie, czego możesz się spodziewać, jeśli kiedykolwiek użyjesz tych metod w swoim kodzie. Metoda fromkeys() używa następującej składni:

```
newdictionaryname = dict(iterable[,value])
```

Zastąp nazwę nowego słownika dowolną nazwą nowego słownika. Nie musi to być nazwa ogólna, taka jak produkt. Może to być coś, co jednoznacznie identyfikuje produkt, na przykład UPC (uniwersalny kod produktu) lub SKU (jednostka magazynowa), które są specyficzne dla Twojej firmy. Zamień część iterowalną na dowolną iterowalną - czyli coś, przez co kod może przechodzić w pętli; wystarczy prosta lista. Część wartości jest opcjonalna. Jeśli zostanie pominięty, każdy klucz w słowniku otrzymuje wartość None, co w Pythonie oznacza po prostu, że żadna wartość nie została jeszcze przypisana do tego klucza w tym słowniku.

Oto przykład, w którym utworzyliśmy nowy słownik DWC001 (udawaj, że jest to SKU dla jakiegoś produktu w naszym asortymencie). Daliśmy jej listę nazw kluczy ujętą w nawiasy kwadratowe i

oddzieloną przecinkami, co sprawia, że jest to właściwie zdefiniowana lista dla Pythona. Nie daliśmy nic wartościowego. Następnie kod drukuje nowy słownik. Jak widać, ostatnia linia kodu wyświetla słownik, który zawiera określone nazwy kluczy, przy czym każdy klucz ma wartość None.

DWC001 = dict.fromkeys(['name', 'unit_price', 'taxable', 'in_stock', 'models'])

print(DWC001)

{'name': None, 'unit_price': None, 'taxable': None, 'in_stock': None, 'models':

None}

Powiedzmy, że nie chcesz wpisywać wszystkich tych nazw kluczy. Chcesz po prostu używać tych samych kluczy, których używasz w innych słownikach. W takim przypadku możesz użyć Dictionary.keys() dla iterowalnej listy nazw kluczy, o ile słownik odnosi się do innego słownika, który już istnieje w programie. Na przykład w poniższym kodzie utworzyliśmy słownik o nazwie product, który ma kilka nazw kluczy i nie ma nic konkretnego dla wartości. Następnie użyliśmy DWC001 = dict. fromkeys(product.keys()), aby utworzyć nowy słownik o określonej nazwie DWC001, który ma te same klucze, co ogólny słownik produktu. Nie określiliśmy żadnych wartości w wierszu dict.fromkeys(product.keys()), więc każdy z tych kluczy w nowym słowniku będzie miał wartość None.

Create a generic dictionary for products name product.

product = {

'name': '',

'unit_price': 0,

'taxable': True,

'in_stock': 0,

'models': []

}

Create a dictionary names DWC001 that has the same keys as product.

DWC001 = dict.fromkeys(product.keys())

#Show what's in the new dictionary.

print(DWC001)

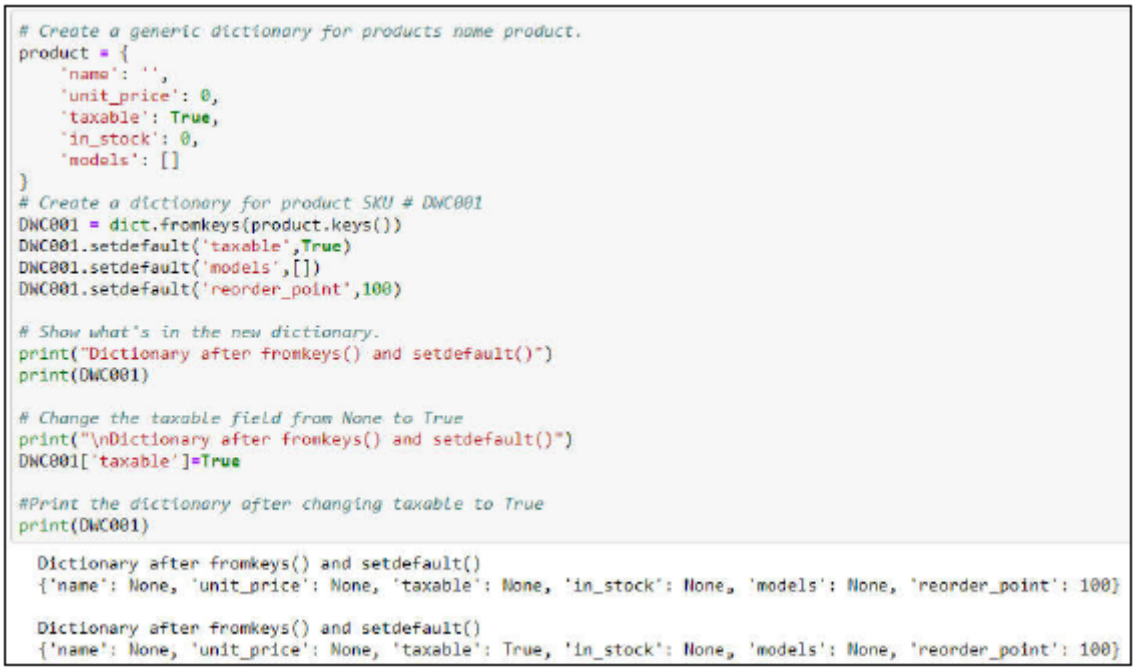
Końcowa instrukcja print() pokazuje zawartość nowego słownika. Możesz zobaczyć, że ma te same klucze, co słownik produktów, z każdą wartością ustawioną na None.

{'name': None, 'unit_price': None, 'taxable': None, 'in_stock': None, 'models':

None}

Wartość .setdefault() pozwala dodać nowy klucz do słownika z predefiniowaną wartością. Dodaje jednak tylko nowy klucz i wartość. Nie zmieni wartości istniejącego klucza, nawet jeśli wartość tego klucza to Brak. Może się więc przydać po fakcie, jeśli zdefiniowałeś słowniki, a później chciałeś dodać kolejną parę właściwość:wartość, ale tylko do słowników, które nie mają jeszcze tej właściwości.

Rysunek



```
# Create a generic dictionary for products name product.
product = {
    'name': '',
    'unit_price': 0,
    'taxable': True,
    'in_stock': 0,
    'models': []
}
# Create a dictionary for product SKU # DWC001
DWC001 = dict.fromkeys(product.keys())
DWC001.setdefault('taxable',True)
DWC001.setdefault('models',[])
DWC001.setdefault('reorder_point',100)

# Show what's in the new dictionary.
print("Dictionary after fromkeys() and setdefault()")
print(DWC001)

# Change the taxable field from None to True
print("\nDictionary after fromkeys() and setdefault()")
DWC001['taxable']=True

#Print the dictionary after changing taxable to True
print(DWC001)
```

```
Dictionary after fromkeys() and setdefault()
{'name': None, 'unit_price': None, 'taxable': None, 'in_stock': None, 'models': None, 'reorder_point': 100}

Dictionary after fromkeys() and setdefault()
{'name': None, 'unit_price': None, 'taxable': True, 'in_stock': None, 'models': None, 'reorder_point': 100}
```

przedstawia przykład, w którym utworzyliśmy słownik DWC001 przy użyciu tych samych kluczy co słownik produktu. Po utworzeniu słownika setdefault('taxable',True) dodaje klucz o nazwie taxable i ustawia jego wartość na True — ale tylko wtedy, gdy ten słownik nie ma jeszcze klucza o nazwie taxable. Dodaje również klucz o nazwie reorder_point i ustawia jego wartość na 10, ale znowu tylko wtedy, gdy ten klucz jeszcze nie istnieje. Jak widać w danych wyjściowych kodu, po operacjach fromkeys i setdefault nowy słownik zawiera te same klucze co słownik produktu oraz nową parę klucz-wartość, reorder_point:10, która została dodana przez drugą operację setdefault. Jednak kluczem podlegającym opodatkowaniu w tych danych wyjściowych jest nadal Brak, ponieważ funkcja setdefault nie zmieni wartości istniejącego klucza. Dodaje tylko nowy klucz z wartością domyślną do słownika, który jeszcze nie ma tego klucza. A co by było, gdybyś naprawdę chciał ustawić domyślną wartość podlegającą opodatkowaniu na Prawda, a nie na brak. Prostym rozwiązaniem byłoby użycie standardowej składni, nazwa słownika [klucz] = nowa wartość, aby zmienić wartość istniejącego klucza podlegającego opodatkowaniu z Brak na Prawdziwe. Drugi wynik na rysunku dowodzi, że zmiana wartości klucza w ten sposób zadziałała.

Słowniki zagnieżdżające

Być może już przyszło ci do głowy, że każdy program, który piszesz, może wymagać kilku słowników, z których każdy ma unikalną nazwę. Ale jeśli po prostu zdefiniujesz kilka słowników z nazwami, jak możesz przechodzić przez cały zestaw bez szczegółowego dostępu do każdego słownika według nazwy? Odpowiedź brzmi: uczyń każdy z tych słowników parą klucz-wartość w jakimś zawierającym je słowniku, gdzie klucz jest unikalnym identyfikatorem każdego słownika (na przykład UPC lub SKU dla każdego produktu). Wartość dla każdego klucza byłaby wtedy słownikiem wszystkich par klucz-wartość dla tego słownika. Składnia byłaby więc następująca:

containingdictionaryname = {

key : {dictionary},

key : {dictionary},

key : {dictionary},

...

}

To tylko składnia słownika słowników. Musisz zastąpić wszystkie nazwy zastępcze pisane kursywą w następujący sposób:

* zawierający nazwę słownika: Jest to nazwa przypisana do słownika jako całości. Może to być dowolna nazwa, ale powinna opisywać zawartość słownika.

* klucz: Każda wartość klucza musi być unikatowa, na przykład UPC lub SKU produktu, nazwa użytkownika dla osób, a nawet tylko numer kolejny, o ile nigdy się nie powtarza.

* {słownik} Umieść wszystkie pary klucz-wartość dla tego jednego elementu słownika w nawiasach klamrowych i zakończ to przecinkiem, jeśli następuje inny słownik.

Rysunek

```
# Create a generic dictionary to contain multiple product dictionaries.
products = {
    'RB00111':{'name':'Rayban Sunglasses', 'price':112.98, 'models':['black', 'tortoise']},
    'DWC0317':{'name':'Drone with Camera', 'price':72.95, 'models':['white', 'black']},
    'MTS0540':{'name':'T-Shirt', 'price':2.95, 'models':['small', 'medium', 'large']},
    'ECD2989':{'name':'Echo Dot', 'price':29.99, 'models':[]},
}
```

pokazuje przykład, w którym mamy słownik o nazwie produkty (liczba mnoga, ponieważ zawiera wiele produktów). Ten słownik z kolei zawiera cztery pojedyncze produkty. Każdy z tych produktów ma unikalny klucz, na przykład RB0011, DWC0317 itd., które są prawdopodobnie wewnętrznymi numerami SKU używanymi przez firmę do zarządzania własnymi zapasami. Każdy z tych czterech produktów ma kolejno nazwę, cenę i modele. Złożona składnia ze wszystkimi nawiasami klamrowymi, przecinkami i dwukropkami sprawia, że trudno zobaczyć, co naprawdę się tam dzieje (nie wspominając o tym, że trudno jest pisać). Ale poza Pythonem, gdzie nie trzeba całego kodowania, dokładnie te same dane można przechowywać jako prostą tabelę o nazwie Produkty z nazwami kluczy jako nagłówkami kolumn, jak ta w Tabeli

ID (key)	Name	Price	Models
RB00111	Rayban Sunglasses	112.98	black, tortoise
DWC0317	Drone with Camera	72.95	white, black
MTS0540	T-Shirt	2.95	small, medium, large
ECD2989	Echo Dot	29.99	

Korzystając z kombinacji f-stringów i niektórych pętli, można sprawić, by Python wyświetlał te dane ze słowników danych w zgrabnym, tabelarycznym formacie. Rysunek

```python
# Create a generic dictionary to contain multiple product dictionaries.
products = {
    'RB00111':{'name':'Rayban Sunglasses', 'price':112.98, 'models':['black','tortoise']},
    'DWC0317':{'name':'Drone with Camera', 'price':72.95, 'models':['white', 'black']},
    'MTS0540':{'name':'T-Shirt', 'price':2.95, 'models':['small', 'medium', 'large']},
    'ECD2989':{'name':'Echo Dot', 'price':29.99, 'models':[]}
    }
print(f"{'ID':<6} {' Name':<17} {'Price':>8}  {' Models'}")
print('-' * 60)
# Loop through each dictionary in the products dictionary
for oneproduct in products.keys():
    # Get the id of one product.
    id = oneproduct
    # Get the name of one product.
    name = products[oneproduct]['name']
    # Get the unit price of one product and format with $
    unit_price = '$' + f"{products[oneproduct]['price']:,.2f}"
    # Create and empty string variable named models
    models = ''
    # Loop through the models list and tack onto models
    # one item from the list followed by a comma and a space.
    for m in products[oneproduct]['models']:
        models += m + ', '
    # If the models variable is more than two characters in lenth,
    # Peel off the last two characters (last comma and space)
    if len(models) > 2:
        models = models[:-2]
    else:
        models="<none>"
    # Print all the variables with a neat f-string.
    print(f"{id:<6} {name:<17} {unit_price:>8} {models}")
```

```
ID     Name              Price   Models
------------------------------------------------------------
RB00111 Rayban Sunglasses  $112.08 black, tortoise
DWC0317 Drone with Camera   $72.95 white, black
MTS0540 T-Shirt              $2.95 small, medium, large
ECD2989 Echo Dot            $29.99 <none>
```

przedstawia przykład takiego kodu w notatniku Jupyter, z danymi wyjściowymi tego kodu tuż pod nim.