Przyspieszenie wraz z listami i krotkami

Czasami w kodzie pracujesz z jednym elementem danych na raz, takim jak imię osoby, cena jednostkowa lub nazwa użytkownika. Często też pracujesz z większymi zestawami danych, takimi jak lista nazwisk osób lub lista produktów i ich cen. Te zestawy danych są często nazywane listami lub tablicami w większości języków programowania. Python ma wiele naprawdę świetnych sposobów radzenia sobie z wszelkiego rodzaju kolekcjami danych w łatwy, szybki i skuteczny sposób. Dowiesz się o nich w tym rozdziale. Jak zawsze, zachęcam do bezpośredniego śledzenia w notatniku Jupyter lub pliku .py podczas lektury tego rozdziału. „Praca" naprawdę pomaga w części „zrozumienie" i pomaga lepiej to wszystko zatopić.

Definiowanie i używanie list

Najprostszym sposobem zbierania danych w Pythonie jest lista. W poprzedniej części przedstawiliśmy ich przykłady. Lista to dowolna lista elementów danych, oddzielonych przecinkami, w nawiasach kwadratowych. Zazwyczaj przypisuje się nazwę do listy za pomocą znaku =, tak jak w przypadku zmiennych. Jeśli lista zawiera liczby, nie używaj cudzysłowu znaki wokół nich. Na przykład, oto lista wyników testów:

wyniki = [88, 92, 78, 90, 98, 84]

Jeśli lista zawiera napisy, to jak zawsze powinny one być ujęte w pojedyncze lub podwójne cudzysłowy, jak w tym przykładzie:

studenci = ["Mark", "Bursztyn", "Todd", "Anita", "Sandy"]

Aby wyświetlić zawartość listy na ekranie, możesz ją wydrukować tak samo, jak każdą zwykłą zmienną. Na przykład wykonanie print(students) w kodzie po zdefiniowaniu tej listy pokazuje to na ekranie.

['Mark', 'Amber', 'Todd', 'Anita', 'Sandy']

To może nie być dokładnie to, o czym myślałeś. Ale nie martw się, Python oferuje wiele świetnych sposobów na dostęp do danych na listach i wyświetlanie ich w dowolny sposób.

NAPRAWDĘ, NAPRAWDĘ DŁUGIE LISTY

Wszystkie listy w tym rozdziale są krótkie, aby przykłady były łatwe i łatwe w zarządzaniu. Jednak w prawdziwym życiu możesz mieć listy zawierające setki, a nawet tysiące elementów, które często się zmieniają. Tego rodzaju listy nie wpisujesz bezpośrednio do kodu, ponieważ utrudnia to pracę z kodem. Bardziej prawdopodobne jest, że będziesz przechowywać takie listy w zewnętrznym pliku lub zewnętrznej bazie danych, gdzie wszystko jest łatwiejsze do zarządzania. Wszystkie techniki, których nauczysz się w tym rozdziale, dotyczą list przechowywanych w plikach zewnętrznych. Jedyną różnicą jest to, że najpierw musisz napisać kod, aby pobrać dane na listę. Jest to o wiele łatwiejsze niż próba wpisywania każdej listy bezpośrednio do kodu. Ale zanim zaczniesz zajmować się tymi naprawdę ogromnymi listami, musisz znać wszystkie techniki pracy z nimi. Więc trzymaj się tego rozdziału, zanim spróbujesz przejść do zarządzania danymi zewnętrznymi. Będziesz zadowolony, że to zrobiłeś.

Odwoływanie się do elementów listy według pozycji

Każda pozycja na liście ma numer pozycji, zaczynając od zera, nawet jeśli nie widzisz żadnych liczb. Możesz odwołać się do dowolnej pozycji na liście, używając jej numeru, używając nazwy listy, po której następuje liczba w nawiasach kwadratowych. Innymi słowy, użyj tej składni:

listame[x]

Zastąp listname nazwą listy, do której uzyskujesz dostęp, i zastąp x numerem pozycji żądanego elementu. Pamiętaj, że pierwsza pozycja to zawsze zero, a nie jeden. Na przykład w pierwszym wierszu poniżej definiuję listę o nazwie studenci, a następnie wypisuję numer pozycji zero z tej listy. Rezultatem wykonywania kodu jest wyświetlenie nazwy Mark.

```
students = ["Mark", "Amber", "Todd", "Anita", "Sandy"]

print(students[0])
```

Mark

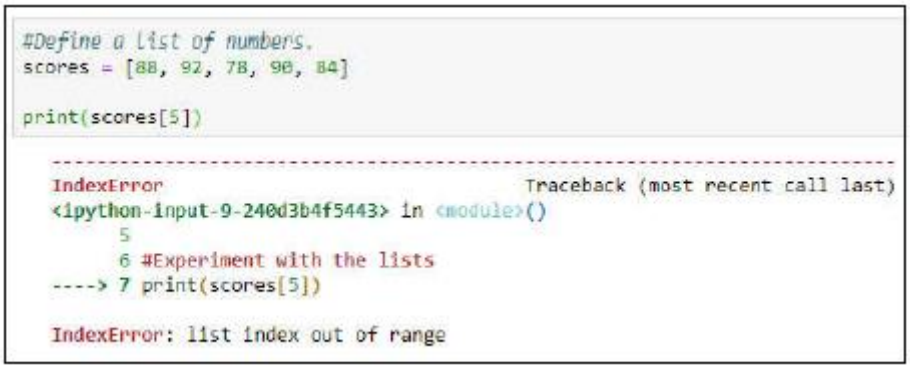
Podczas czytania elementów listy dostępu profesjonaliści używają słowa sub przed liczbą. Na przykład uczniowie[0] będą wypowiadani jako uczniowie poniżej zera. Następny przykład pokazuje listę o nazwie scores. Funkcja print() drukuje numer pozycji ostatniego wyniku na liście, który wynosi 4 (ponieważ pierwsza z nich to zawsze zero).

```
scores = [88, 92, 78, 90, 84]

print(scores[4])
```

84

Jeśli spróbujesz uzyskać dostęp do elementu listy, który nie istnieje, otrzymasz błąd „indeks poza zakresem”. Część indeksowa to odwołanie do numeru w nawiasach kwadratowych. Na przykład Figure pokazuje mały eksperyment w notatniku Jupytera, w którym stworzyliśmy listę wyników, a następnie próbowaliśmy wydrukować partyturę[5]. Nie powiodło się i wygenerowało błąd, ponieważ nie ma punktów[5]. Są tylko scores[0], scores[1], scores[2], scores[3] i scores[4], ponieważ liczenie zawsze zaczyna się od zera od pierwszego na liście.



```
#Define a list of numbers.
scores = [88, 92, 78, 90, 84]

print(scores[5])
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-9-240d3b4f5443> in <module>()
      5
      6 #Experiment with the lists
----> 7 print(scores[5])

IndexError: list index out of range
```

Przeglądanie listy

Aby uzyskać dostęp do każdego elementu na liście, po prostu użyj pętli for o następującej składni:

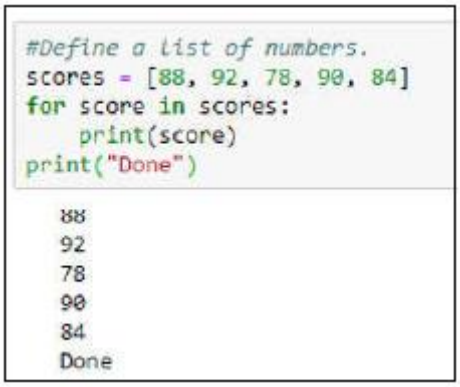
for x in list

Zastąp x wybraną nazwą zmiennej. Zastąp listę nazwą listy. Prostym sposobem na uczynienie kodu czytelnym jest zawsze używanie liczby mnogiej w nazwie listy (np. studenci, wyniki). Następnie możesz użyć nazwy pojedynczej (student, wynik) jako nazwy zmiennej. W tym podejściu nie musisz również używać liczb w indeksie dolnym (liczby w nawiasach kwadratowych). Na przykład następujący kod

for score in scores:

print(score)

Pamiętaj, aby zawsze wcinać kod, który ma zostać wykonany w pętli. Rysunek przedstawia bardziej kompletny przykład, w którym można zobaczyć wynik uruchomienia kodu w notatniku Jupyter.



Sprawdzanie, czy lista zawiera element

Jeśli chcesz, aby Twój kod sprawdzał zawartość listy, aby zobaczyć, czy zawiera już jakiś element, użyj w listname w instrukcji if lub przypisaniu zmiennej. Na przykład kod na rysunku tworzy listę nazw. Następnie dwie zmienne przechowują wyniki wyszukiwania na liście nazwisk Anita i ja Bob. Wydrukowanie zawartości każdej zmiennej pokazuje True dla tej, której imię (Anita) znajduje się na liście. Test sprawdzający, czy Bob jest na liście, dowodzi fałszu

Uzyskiwanie długości listy

Aby określić, ile elementów znajduje się na liście, użyj funkcji len() (skrót od długości). Umieść nazwę listy w nawiasach. Na przykład wpisz następujący kod do notatnika Jupyter, monitu Pythona lub cokolwiek innego:

students = ["Mark", "Amber", "Todd", "Anita", "Sandy"]

print(len(students))

Uruchomienie tego kodu daje następujące dane wyjściowe:

5

Rzeczywiście na liście jest pięć pozycji, chociaż ostatnia jest zawsze o jeden mniejsza od liczby, ponieważ Python zaczyna liczyć od zera. Tak więc ostatnia, Sandy, w rzeczywistości odnosi się do studentów[4], a nie studentów[5].

Dodawanie pozycji na końcu listy

Jeśli chcesz, aby Twój kod dodał nowy element na końcu listy, użyj metody .append() z wartością, którą chcesz dodać w nawiasach. Możesz użyć nazwy zmiennej lub wartości literału w cudzysłowie. Na przykład na rysunku wiersz z napisem students.append("Goober") dodaje nazwę Goober do listy. Wiersz, który czyta studens.append(new_student) dodaje jakąkolwiek nazwę która jest przechowywana w zmiennej o nazwie new_studet na liście. Metoda .append() zawsze dodaje się na końcu listy. Więc kiedy drukujesz listę, widzisz te dwie nowe nazwy na końcu.

Możesz użyć testu, aby sprawdzić, czy element znajduje się na liście, a następnie dołączyć go tylko wtedy, gdy element jeszcze nie istnieje. Na przykład poniższy kod nie doda nazwy Amber do listy, ponieważ ta nazwa jest już na liście:

student_name = "Amanda"

#Add student_name but only if not already in the list.

if student_name in students:

print (student_name " already in the list")

else:

students.append(student_name)

print (student_name " added to the list")

Wstawianie pozycji do listy

Chociaż metoda append() umożliwia dodanie elementu na końcu listy, metoda insert() umożliwia dodanie elementu do listy w dowolnej pozycji. Składnia insert() to

listname.insert(position, item)

Zastąp listname nazwą listy, position pozycją, w której chcesz wstawić element (na przykład 0, aby uczynić go pierwszym elementem, 1, aby uczynić go drugim elementem itd.). Zastąp element wartością lub nazwą zmiennej zawierającej wartość, którą chcesz umieścić na liście. Na przykład poniższy kod czyni Lupe pierwszą pozycją na liście:

```
#Create a list of strings (names).

students = ["Mark", "Amber", "Todd", "Anita", "Sandy"]

student_name = "Lupe"

# Add student name to front of the list.

students.insert(0,student_name)

#Show me the new list.

print(students)
```

Jeśli uruchomisz kod, print(students) pokaże listę po wstawieniu nowego nazwiska w następujący sposób:

['Lupe', 'Mark', 'Amber', 'Todd', 'Anita', 'Sandy']

Zmiana pozycji na liście

Pozycję na liście można zmienić za pomocą operatora = przypisania, tak jak w przypadku zmiennych. Tylko upewnij się, że wpisujesz numer indeksu w nawiasach kwadratowych elementu, który chcesz zmienić. Składnia to:

listname[index]=newvalue

Zastąp listname nazwą listy; zamień indeks na indeks dolny (numer indeksu) elementu, który chcesz zmienić; i zastąp newvalue tym, co chcesz umieścić w elemencie listy. Na przykład spójrz na ten kod:

```
#Create a list of strings (names).

students = ["Mark", "Amber", "Todd", "Anita", "Sandy"]

students[3] = "Hobart"

print(students)
```

Po uruchomieniu tego kodu wynik jest następujący, ponieważ imię Anity zostało zmienione na Hobart.

['Mark', 'Amber', 'Todd', 'Hobart', 'Sandy']

Łączenie list

Jeśli masz dwie listy, które chcesz połączyć w jedną listę, użyj funkcji extend() ze składnią:

original_list.extend(additional_items_list)

W kodzie zastąp original_list nazwą listy, do której będziesz dodawać nowe elementy listy. Zastąp dodatkowe_elementy_lista nazwą listy zawierającej elementy, które chcesz dodać do pierwszej listy. Oto prosty przykład z użyciem list o nazwach lista1 i lista2. Po wykonaniu list1.extend(list2), pierwsza lista zawiera pozycje z obu list, jak widać na końcu instrukcji print().

```
# Create two lists of Names.

list1 = ["Zara", "Lupe", "Hong", "Alberto", "Jake"]

list2 = ["Huey", "Dewey", "Louie", "Nader", "Bubba"]

# Add list2 names to list1.
```

```
list1.extend(list2)

# Print list 1.

print(list1)

['Zara', 'Lupe', 'Hong', 'Alberto', 'Jake', 'Huey', 'Dewey', 'Louie', 'Nader',

'Bubba']
```

Łatwy Parcheesi, nie?

Usuwanie pozycji z listy

Python oferuje metodę remove(), dzięki której możesz usunąć dowolną wartość z listy. Jeśli element znajduje się na liście wiele razy, usuwane jest tylko pierwsze wystąpienie. Na przykład poniższy kod przedstawia listę liter z literą C powtórzoną kilka razy. Następnie kod używa letters.remove("C"), aby usunąć literę C z listy:

```
#Create a list of strings.

letters = ["A", "B", "C", "D", "C", "E", "C"]

# Remove "C" from the list.

letters.remove("C")

#Show me the new list.

print(letters)
```

Kiedy faktycznie wykonasz ten kod, a następnie wydrukujesz listę, zobaczysz, że tylko pierwsza litera C została usunięta:

```
['A', 'B', 'D', 'C', 'E', 'C']
```

Jeśli chcesz usunąć cały element, możesz użyć pętli while, aby powtórzyć .remove, dopóki element nadal pozostaje na liście. Na przykład ten kod powtarza .remove, dopóki „C” nadal znajduje się na liście.

```
#Create a list of strings.

letters = ["A", "B", "C", "D", "C", "E", "C"]
```

Jeśli chcesz usunąć element na podstawie jego pozycji na liście, użyj pop() z numerem indeksu zamiast remove() z wartością. Jeśli chcesz usunąć ostatni element z listy, użyj pop() bez numeru indeksu. Na przykład poniższy kod tworzy listę, jeden wiersz usuwa pierwszy element (0), a drugi usuwa ostatni element (pop() bez niczego w nawiasach). Drukowanie listy pokazuje ,że te dwa elementy zostały usunięte:

```
#Create a list of strings.

letters = ["A", "B", "C", "D", "E", "F", "G"]

#Remove the first item.

letters.pop(0)

#Remove the last item.
```

letters.pop()

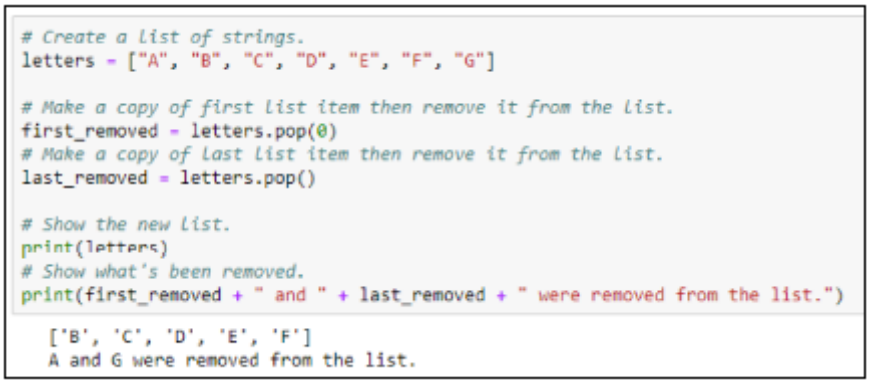
#Show me the new list.

print(letters)

Uruchomienie kodu pokazuje, że popping pierwszego i ostatniego elementu rzeczywiście zadziałał:

['B', 'C', 'D', 'E', 'F']

Kiedy pop() element z listy, możesz przechowywać kopię tej wartości w jakiejś zmiennej. Na przykład Rysunek pokazuje ten sam kod, co powyżej. Jednak przechowuje kopie tego, co zostało usunięte w zmiennych o nazwach first_removed i last_removed. Na koniec wypisuje listę, a także pokazuje, które litery zostały usunięte.



Python oferuje również polecenie del (skrót od usuwania), które usuwa dowolny element z listy na podstawie jego numeru indeksu (pozycji). Ale znowu musisz pamiętać, że pierwsza pozycja to zero. Załóżmy więc, że uruchamiasz następujący kod, aby usunąć pozycję numer 2 z listy:

Create a list of strings.

letters = ["A", "B", "C", "D", "E", "F", "G"]

Remove item sub 2.

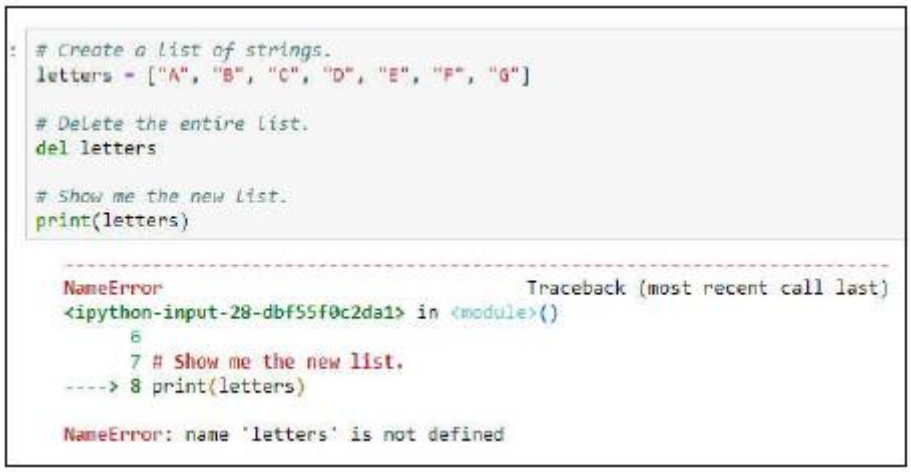
del letters[2]

print(letters)

Uruchomienie tego kodu powoduje ponowne wyświetlenie listy w następujący sposób. Litera C została usunięta, co jest poprawnym elementem do usunięcia, ponieważ litery są ponumerowane 0, 1, 2, 3 i tak dalej.

['A', 'B', 'D', 'E', 'F', 'G']

Możesz także użyć del, aby usunąć całą listę. Po prostu nie używaj nawiasów kwadratowych i numeru indeksu. Na przykład kod na rysunku tworzy listę, a następnie ją usuwa. Próba wydrukowania listy po usunięciu powoduje błąd, ponieważ po wykonaniu instrukcji print() lista już nie istnieje.



Czyszczenie listy

Jeśli chcesz usunąć zawartość listy, ale nie samą listę, użyj .clear(). Lista nadal istnieje; jednak nie zawiera żadnych elementów. Innymi słowy, to pusta lista. Poniższy kod pokazuje, jak możesz to przetestować. Uruchomienie kodu wyświetla [] na końcu, co informuje, że lista jest pusta:

Create a list of strings.

letters = ["A", "B", "C", "D", "E", "F", "G"]

Clear the list of all entries.

letters.clear()

Show me the new list.

print(letters)

[]

Zliczanie, ile razy element pojawia się na liście

Możesz użyć metody Python count(), aby policzyć, ile razy element pojawia się na liście. Podobnie jak w przypadku innych metod listowych, składnia jest prosta:

listname.count(x)

Zastąp listname nazwą swojej listy, a x wartością, której szukasz (lub nazwą zmiennej, która zawiera tę wartość). Kod na rysunku zlicza, ile razy litera B pojawia się na liście, używając literału B w nawiasach .count(). Ten sam kod zlicza również liczbę klas C, ale zapisaliśmy tę wartość w zmiennej tylko po to, aby pokazać różnicę w składni. Oba obliczenia zadziałały, jak widać na wyjściu w program na dole. Dodaliśmy również jeden, aby policzyć F, nie używając żadnych zmiennych. Właśnie policzyliśmy literę F w kodzie wyświetlającym wiadomość. Nie ma ocen F, więc to zwraca zero, jak widać na wyjściu.

```
# Create a list of strings.
grades = ["C", "B", "A", "D", "C", "B", "C"]

# Count the B's
b_grades = grades.count("B")

# use a variable for value to count.
look_for = "C"
c_grades = grades.count(look_for)

print("There are " + str(b_grades) + " B grades in the list.")
print("There are " + str(c_grades) + " " + look_for + " grades in the list.")

#Count Fs too.
print("There are " + str(grades.count("F")) + " F grades in the list.")
```

```
There are 2 B grades in the list.
There are 3 C grades in the list.
There are 0 F grades in the list.
```

Kiedy próbujesz połączyć liczby i ciągi w celu utworzenia wiadomości, pamiętaj, że musisz przekonwertować liczby na ciągi za pomocą funkcji str(). W przeciwnym razie otrzymasz błąd, który odczytuje coś takiego jak może tylko łączyć str (nie "int") ze str. W tym komunikacie int jest skrótem od liczby całkowitej, a str jest skrótem od łańcucha.

Znajdowanie indeksu pozycji listy

Python oferuje metodę .index(), która zwraca liczbę wskazującą pozycję, na podstawie numeru indeksu, elementu na liście. Składnia to:

listname.index(x)

Jak zawsze, zastąp listname nazwą listy, którą chcesz przeszukać. Zastąp x tym, czego szukasz (jak zawsze jako literał lub jako nazwę zmiennej). Oczywiście nie ma gwarancji, że przedmiot znajduje się na liście, a nawet jeśli tak, nie ma gwarancji, że przedmiot znajduje się na liście tylko raz. Jeśli pozycji nie ma na liście, wystąpi błąd. Jeśli element znajduje się na liście wiele razy, zwracany jest indeks pierwszego pasującego elementu. Rysunek 3-8 pokazuje przykład, w którym program ulega awarii w wierszu f_index = grades.index(look_for), ponieważ na liście nie ma F.

```
# Create a list of strings.
grades = ["C", "B", "A", "D", "C", "B", "C"]

# Find the index for "B"
b_index = grades.index("B")

#Find the index for F
look_for = "F"
f_index = grades.index(look_for)

# Show the results.
print("The first B is index " + str(b_index))
print("There first "+ look_for + " is at " + str(f_index))
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-38-ee447e55d5c6> in <module>()
      7 #Find the index for F
      8 look_for = "F"
----> 9 f_index = grades.index(look_for)
     10
     11 # Show the results.

ValueError: 'F' is not in list
```

Prostym sposobem na obejście tego problemu jest użycie instrukcji if, aby sprawdzić, czy element znajduje się na liście, zanim spróbujesz uzyskać jego numer indeksu. Jeśli pozycji nie ma na liście, wyświetl komunikat z informacją. W przeciwnym razie uzyskaj numer indeksu i pokaż go w wiadomości. Ten kod wygląda następująco:

Create a list of strings.

grades = ["C", "B", "A", "D", "C", "B", "C"]

Decide what to look for

look_for = "F"

See if the item is in the list.

if look_for in grades:

If it's in the list, get and show the index.

print(str(look_for) " is at index " str(grades.index(look_for)))

else:

If not in the list, don't even try for index number.

print(str(look_for) " isn't in the list.")

Alfabetyzacja i sortowanie list

Python oferuje metodę sort() do sortowania list. W swojej najprostszej formie porządkuje elementy na liście alfabetycznie (jeśli są to ciągi). Jeśli lista zawiera liczby, są one sortowane od najmniejszej do największej. Aby wykonać takie proste sortowanie, po prostu użyj sort() z pustymi nawiasami:

listname.sort()

Zastąp listname nazwą swojej listy. Rysunek przedstawia przykład użycia listy ciągów i listy liczb. Stworzyliśmy nową listę dla każdego z nich, po prostu przypisując każdą posortowaną listę do nowej nazwy listy. Następnie kod wyświetla zawartość każdej posortowanej listy.

```
# Create a list of strings.
names = ["Zara", "Lupe", "Hong", "Alberto", "Jake", "Tyler"]
# Create a list of numbers
numbers = [14, 0, 56, -4, 99, 56, 11.23]

# Sort the names list.
names.sort()
# Sort the numbers list.
numbers.sort()

# Show the results
print(names)
print(numbers)
```

```
['Alberto', 'Hong', 'Jake', 'Lupe', 'Tyler', 'Zara']
[-4, 0, 11.23, 14, 56, 56, 99]
```

Daty są nieco trudniejsze, ponieważ nie można ich po prostu wpisać jako ciągów, np. „31.12.2020”. Aby poprawnie posortować, muszą to być dane typu data. Oznacza to użycie modułu datetime i

metody date() do zdefiniowania każdej daty. Możesz dodać daty do listy, tak jak każdą inną listę. Na przykład w poniższym wierszu kod tworzy listę czterech dat, a kod jest idealnie w porządku.

dates = [dt.date(2020,12,31), dt.date(2019,1,31), dt.date(2018,2,28),

dt.date(2020,1,1)]

Komputer z pewnością nie będzie miał nic przeciwko, jeśli utworzysz listę w ten sposób. Ale jeśli chcesz, aby kod był bardziej czytelny dla siebie lub innych programistów, możesz utworzyć i dołączyć każdą datę, pojedynczo, aby trochę łatwiej było zobaczyć, co się dzieje i aby nie mieć radzić sobie z tyloma przecinkami w jednym wierszu kodu. Rysunek pokazuje przykład, w którym utworzyliśmy pustą listę o nazwie datelist:

datelist = []

Następnie dodaliśmy do listy jedną datę naraz, używając składni dt.date(rok,miesiąc,dzień), jak pokazano na rysunku

```
]: # Need this modules for the dates.
   import datetime as dt

   # Create a list of dates, empty for starters
   datelist = []
   # Append dates one at time so code is easier to read.
   datelist.append(dt.date(2020,12,31))
   datelist.append(dt.date(2019,1,31))
   datelist.append(dt.date(2018,2,28))
   datelist.append(dt.date(2020,1,1))

   # Sort the dates (earliest to latest) and show formatted.
   datelist.sort()
   for date in datelist:
       print(f"{date:%m/%d/%Y}")

   02/28/2018
   01/31/2019
   01/01/2020
   12/31/2020
```

Po utworzeniu listy kod używa datelist.sort() do sortowania ich w porządku chronologicznym (od najstarszego do najnowszego). Nie użyliśmy print(datelist) w tym kodzie, ponieważ ta metoda wyświetla daty z dołączonymi informacjami o typie danych, na przykład:

[datetime.date(2018, 2, 28), datetime.date(2019, 1, 31), datetime.date

(2020, 1, 1), datetime.date(2020, 12, 31)]

Nie najłatwiejsza lista do przeczytania. Tak więc, zamiast drukować całą listę za pomocą jednej instrukcji print(), przeszliśmy w pętli przez każdą datę na liście i wydrukowaliśmy każdą z nich sformatowaną za pomocą f-stringu %m/%d/%Y. Spowoduje to wyświetlenie każdej daty w osobnym wierszu w formacie mm/dd/rrrr, jak widać na dole powyższego rysunku. Jeśli chcesz posortować elementy w odwrotnej kolejności, umieść reverse=True w nawiasach sort() (i nie zapomnij, aby pierwsza litera była wielką). Rysunek przedstawia przykłady sortowania wszystkich trzech list w kolejności malejącej (odwrotnej) przy użyciu reverse=True.

```
# Need this modules for the dates.
import datetime as dt

# Create a list of strings.
names = ["Zara", "Lupe", "Hong", "Alberto", "Jake", "Tyler"]

# Create a list of numbers.
numbers = [14, 0, 56, -4, 99, 56, 11.23]

# Create a list of dates, empty for starters because code is long.
datelist = []
datelist.append(dt.date(2020,12,31))
datelist.append(dt.date(2019,1,31))
datelist.append(dt.date(2018,2,28))
datelist.append(dt.date(2020,1,1))

# Sort strings in reverse order (Z to A) and show.
names.sort(reverse=True)
print(names)
print() # This just adds a blank line to the output.

#Sort numbers in reverse order (largest to smallest) and show.
numbers.sort(reverse=True)
print(numbers)
print() # This just adds a blank line to the output.

# Sort the dates in reverse order (Latest to earliest) and show formatted.
datelist.sort(reverse = True)
for date in datelist:
    print(f"{date:%m/%d/%Y}")
```

```
['Zara', 'Tyler', 'Lupe', 'Jake', 'Hong', 'Alberto']

[99, 56, 56, 14, 11.23, 0, -4]

12/31/2020
01/01/2020
01/31/2019
02/28/2018
```

Odwracanie listy

Możesz również odwrócić kolejność elementów na liście za pomocą metody .reverse. To nie to samo, co sortowanie odwrotne, ponieważ podczas sortowania odwrotnego nadal sortujesz: Z–A dla ciągów, od największego do najmniejszego dla liczb, od ostatniego do najwcześniejszego dla dat. Kiedy odwracasz listę, po prostu odwracasz pozycje na liście, bez względu na ich kolejność, bez próby ich sortowania w jakikolwiek sposób. Poniższy kod pokazuje przykład, w którym odwracamy kolejność nazw na liście, a następnie drukujemy listę. Dane wyjściowe pokazują elementy listy odwrócone od ich pierwotnej kolejności:

Create a list of strings.

names = ["Zara", "Lupe", "Hong", "Alberto", "Jake"]

Reverse the list

names.reverse()

Print the list

print(names)

['Jake', 'Alberto', 'Hong', 'Lupe', 'Zara']

Kopiowanie listy

Jeśli kiedykolwiek będziesz musiał pracować z kopią listy, użyj metody .copy(), aby nie zmieniać oryginalnej listy. Na przykład poniższy kod jest podobny do poprzedniego, z tą różnicą, że zamiast odwracać kolejność oryginalnej listy, tworzymy kopię listy i odwracamy ją. Wydrukowanie zawartości każdej listy pokazuje, że pierwsza lista jest nadal w oryginalnej kolejności, podczas gdy druga jest odwrócona:

```
# Create a list of strings.

names = ["Zara", "Lupe", "Hong", "Alberto", "Jake"]

# Make a copy of the list

backward_names = names.copy()

# Reverse the copy

backward_names.reverse()

# Print the list

print(names)

print(backward_names)
```

```
['Zara', 'Lupe', 'Hong', 'Alberto', 'Jake']

['Jake', 'Alberto', 'Hong', 'Lupe', 'Zara']
```

Tabela zawiera podsumowanie metod, o których do tej pory dowiedziałeś się w tym rozdziale. Jak zobaczysz w kolejnych rozdziałach, te same metody z innymi rodzajami iterowalnych (wymyślna nazwa oznacza dowolną listę lub coś podobnego do listy, przez które możesz przejść pojedynczo).

append() : Dodaje element na koniec listy.

clear() : Usuwa wszystkie pozycje z listy, pozostawiając ją pustą.

copy() : Tworzy kopię listy.

count() : Zlicza ile razy element pojawia się na liście.

extend() : Dołącza elementy z jednej listy na koniec innej listy.

index() : Zwraca numer indeksu (pozycję) elementu na liście.

insert() : Wstawia element na listę w określonej pozycji.

pop() : Usuwa element z listy i udostępnia kopię tego elementu, którą można przechowywać w zmiennej.

remove() : Usuwa jedną pozycję z listy.

reverse() : Odwraca kolejność pozycji na liście.

sort() : Sortuje listę w porządku rosnącym. Umieść reverse=True w nawiasach, aby posortować w kolejności malejącej.

Co to jest krotka i kogo to obchodzi?

Oprócz list Python obsługuje strukturę danych znaną jako krotka. Niektórzy ludzie wymawiają to jak dwa pociągnięcia. Niektórzy ludzie wymawiają to jak krotka (rymy z parą). Ale nie jest to krotka czy tuple, więc naszym najlepszym przypuszczeniem jest to, że jest wymawiane jak dwa pociągnięcia. (Do diabła, z tego, co wiemy, może nawet nie istnieć „poprawny" sposób jego wymówienia. To jednak prawdopodobnie nie powstrzymuje ludzi przed długimi kłótniami na temat „poprawnej" wymowy.) W każdym razie, pomimo dziwacznej nazwy, krotka jest po prostu niezmienną listą (tak jak to wiele mówi). Innymi słowy, krotka jest listą, ale po jej zdefiniowaniu nie można jej zmienić. Dlaczego więc chcesz umieścić w aplikacji niezmienne, niezmienne dane? Rozważ Amazon. Gdybyśmy wszyscy mogli wejść do Amazon i zmieniać rzeczy do woli, wszystko kosztowałoby pensa i wszyscy mielibyśmy całe domy Amazonek, które kosztują pensa, zamiast domów Amazonek, które kosztują więcej niż pensa. Składnia tworzenia krotki jest taka sama jak składnia tworzenia listy, z wyjątkiem tego, że nie używasz nawiasów kwadratowych. Musisz użyć nawiasów, takich jak:

prices = (29.95, 9.98, 4.95, 79.98, 2.95)

Większość technik i metod, których nauczyłeś się do używania list w powyższej tabeli, nie działa z krotkami, ponieważ są one używane do modyfikowania czegoś na liście, a krotki nie można modyfikować. Możesz jednak uzyskać długość krotki za pomocą len, w następujący sposób:

print(len(prices))

Możesz użyć .count(), aby zobaczyć, ile razy element pojawia się w krotce. Na przykład:

print(prices.count(4.95))

Możesz użyć in, aby sprawdzić, czy wartość istnieje w krotce, jak w poniższym przykładowym kodzie:

print(4.95 in prices)

Zwraca to prawda z krotki zawierającej 4,95. Zwraca False, jeśli tego nie zawiera. Jeśli element istnieje w krotce, możesz uzyskać jego numer indeksu. Otrzymasz jednak błąd, jeśli element nie istnieje na liście. Możesz użyć in first, aby sprawdzić, czy element istnieje przed sprawdzeniem jego numeru indeksu, a następnie możesz zwrócić jakąś nonsensowną wartość, taką jak -1, jeśli nie istnieje, jak w tym kodzie:

look_for = 12345

if look_for in prices:

position = prices.index(look_for)

else:

position=-1

print(position)

Możesz przeglądać elementy w krotce i wyświetlać je w dowolnym formacie, używając ciągów formatujących. Na przykład ten kod wyświetla każdą pozycję z wiodącym znakiem dolara i dwiema cyframi groszy:

#Loop through and display each item in the tuple.

for price in prices:

print(f"${price:.2f}")

Dane wyjściowe z uruchomienia tego kodu z przykładową krotką są następujące:

29,95

9,98 USD

4,95 USD

79,98 zł

2,95 USD

Nie możesz zmienić wartości elementu w krotce przy użyciu tego rodzaju składni:

prices[1] = 234.56

Jeśli spróbujesz tego, otrzymasz komunikat o błędzie, który brzmi TypeError: obiekt „tuple” nie obsługuje przypisywania elementów. Oznacza to, że nie możesz użyć operatora przypisania =, aby zmienić wartość elementu w krotce, ponieważ krotka jest niezmienna, co oznacza, że jej zawartość nie może zostać zmieniona. Każda z metod zmieniających dane, a nawet po prostu kopiujących dane z listy, powoduje błąd podczas próby ich z krotką. Obejmuje to .append(), .clear(), .copy(), .extend(), .insert(), .pop(), .remove(), .reverse() i .sort(). Krótko mówiąc, krotka ma sens, jeśli chcesz pokazać dane użytkownikom, nie dając im żadnych możliwości zmiany jakichkolwiek informacji.

Praca ze zbiorami

Python oferuje również zestawy jako sposób organizowania danych. Różnica między zestawem a listą polega na tym, że elementy w zestawie nie mają określonej kolejności. Nawet jeśli możesz zdefiniować zestaw z elementami w określonej kolejności, żaden z elementów nie otrzyma numerów indeksu, aby zidentyfikować ich pozycje. Aby zdefiniować zestaw, użyj nawiasów klamrowych, w których użyłbyś nawiasów kwadratowych dla listy i nawiasów dla krotki. Na przykład oto zestaw z kilkoma liczbami

sample_set = {1.98, 98.9, 74.95, 2.5, 1, 16.3}

Zestawy są podobne do list i krotek na kilka sposobów. Możesz użyć len(), aby określić, ile elementów znajduje się w zestawie. Użyj in, aby określić, czy element znajduje się w zestawie. Ale nie możesz otrzymać przedmiotu w zestawie na podstawie jego numeru indeksu. Nie możesz też zmienić przedmiotu, który jest już w zestawie. Nie możesz również zmienić kolejności elementów w zestawie. Nie możesz więc użyć .sort() do sortowania zestawu lub .reverse() do odwrócenia jego kolejności. Możesz dodać pojedynczy nowy element do zestawu za pomocą .add(), jak w poniższym przykładzie:

sample_set.add(11.23)

Możesz także dodać wiele elementów do zestawu za pomocą .update(). Ale elementy, które dodajesz, powinny być zdefiniowane jako lista w nawiasach kwadratowych, jak w poniższym przykładzie:

sample_set.update([88, 123.45, 2.98])

Możesz skopiować zestaw. Jednakże, ponieważ zestaw nie ma zdefiniowanej kolejności, podczas wyświetlania kopii jego elementy mogą nie być w tej samej kolejności co oryginalny zestaw, jak pokazano w tym kodzie i jego danych wyjściowych:

Define a set named sample_set.

sample_set = {1.98, 98.9, 74.95, 2.5, 1, 16.3}

Show the whole set

print(sample_set)

Make a copy and shoe the copy.

ss2 = sample_set.copy()

print(ss2)

{1.98, 98.9, 2.5, 1, 74.95, 16.3}

{16.3, 1.98, 98.9, 2.5, 1, 74.95}

Możesz przeglądać zestaw i wyświetlać jego zawartość sformatowaną za pomocą f-strings. Ostatnie kilka wierszy kodu na rysunku pokazuje pętlę, która używa >6.2f do sformatowania wszystkich cen wyrównanych do prawej w obrębie sześciu znaków. Dane wyjściowe z tego kodu są pokazane na dole rysunku. Możesz zobaczyć dane wyjściowe z kilku pierwszych instrukcji print. Lista na końcu jest wyprowadzana z pętli, która wypisuje każdą wartość wyrównaną do prawej. Listy i krotki to dwie najczęściej używane struktury danych Pythona. Wydaje się, że sety nie mają tak dużo zabawy, jak pozostałe dwa, ale dobrze jest o nich wiedzieć.

```
# Define a set named sample_set.
sample_set = {1.98, 98.9, 74.95, 2.5, 1, 16.3}
# Show the whole set
print(sample_set)

# Use len to get the length of a set.
print(len(sample_set))

# Use in to determine if the set contains a value
print(74.95 in sample_set)

# Use add() to add one item to a set.
sample_set.add(11.23)

# Use update() to add a [List] to a set.
sample_set.update([88, 123.45, 2.98])

print("\nSample set after .add() and .update()")
print(sample_set)

# Loop through the set and print each item right-aligned and formatted.
print("\nLoop through set and print each item formatted.")
for price in sample_set:
    print(f"{price:>6.2f}")
```

```
{1.98, 98.9, 2.5, 1, 74.95, 16.3}
6
True

Sample set after .add() and .update()
{1.98, 98.9, 2.5, 1, 2.98, 74.95, 11.23, 16.3, 88, 123.45}

Loop through set and print each item formatted.
  1.98
 98.90
  2.50
  1.00
  2.98
 74.95
 11.23
 16.30
 88.00
123.45
```