Перебор ключей словаря в определенном порядке

Начиная с Python версии 3.7, перебор содержимого словаря возвращает элементы в том порядке, в каком они вставлялись. Тем не менее иногда требуется перебрать элементы словаря в другом порядке.

Один из способов получения элементов в определенном порядке основан на сортировке ключей, возвращаемых циклом `for`. Для получения упорядоченной копии ключей можно воспользоваться функцией `sorted()`:

```
favorite_languages = {
    'jen': 'python',
    'sarah': 'c',
    'edward': 'ruby',
    'phil': 'python',
    }

for name in sorted(favorite_languages.keys()):
    print(f"{name.title()}, thank you for taking the poll.")
```

Эта команда `for` не отличается от других команд `for`, если не считать того, что метод `dictionary.keys()` заключен в вызов функции `sorted()`. Эта конструкция приказывает Python выдать список всех ключей в словаре и отсортировать его перед тем, как перебирать элементы. В выводе перечислены все пользователи, участвовавшие в опросе, а их имена упорядочены по алфавиту:

```
Edward, thank you for taking the poll.
Jen, thank you for taking the poll.
Phil, thank you for taking the poll.
Sarah, thank you for taking the poll.
```

Перебор всех значений в словаре

Если вас прежде всего интересуют значения, содержащиеся в словаре, используйте метод `values()` для получения списка значений без ключей. Допустим, вы хотите просто получить список всех языков, выбранных в опросе, и вас не интересуют имена людей, выбравших каждый язык:

```
favorite_languages = {
    'jen': 'python',
    'sarah': 'c',
    'edward': 'ruby',
    'phil': 'python',
    }

print("The following languages have been mentioned:")
for language in favorite_languages.values():
    print(language.title())
```

Команда `for` читает каждое значение из словаря и сохраняет его в переменной `language`. При выводе этих значений будет получен список всех выбранных языков:

```
The following languages have been mentioned:
Python
C
Python
Ruby
```

Значения извлекаются из словаря без проверки на возможные повторения. Для небольших словарей это может быть приемлемо, но в опросах с большим количеством респондентов список будет содержать слишком много дубликатов. Чтобы получить список выбранных языков без повторений, можно воспользоваться *множеством* (set). Множество в целом похоже на список, но все его элементы должны быть уникальными:

```
  favorite_languages = {
      ...
      }

  print("The following languages have been mentioned:")
❶ for language in set(favorite_languages.values()):
      print(language.title())
```

Когда список, содержащий дубликаты, заключается в вызов `set()`, Python находит уникальные элементы списка и строит множество из этих элементов. В точке ❶ `set()` используется для извлечения уникальных языков из `favorite_languages.values()`.

В результате создается не содержащий дубликатов список языков программирования, упомянутых участниками опроса:

```
The following languages have been mentioned:
Python
C
Ruby
```

В ходе дальнейшего изучения Python вы часто будете обнаруживать встроенные возможности языка, которые помогают сделать с данными именно то, что вам требуется.

ПРИМЕЧАНИЕ Множество можно построить прямо в фигурных скобках с разделением элементов запятыми:

```
>>> languages = {'python', 'ruby', 'python', 'c'}
>>> languages
{'ruby', 'python', 'c'}
```

Словари легко перепутать с множествами, потому что обе структуры заключаются в фигурные скобки. Когда вы видите фигурные скобки без пар «ключ-значение», скорее всего, перед вами множество. В отличие от списков и словарей, элементы множеств не хранятся в каком-либо определенном порядке.

УПРАЖНЕНИЯ

6.4. Глоссарий 2: теперь, когда вы знаете, как перебрать элементы словаря, упростите код из упражнения 6.3, заменив серию команд `print` циклом, перебирающим ключи и значения словаря. Когда вы будете уверены в том, что цикл работает, добавьте в глоссарий еще пять терминов Python. При повторном запуске программы новые слова и значения должны быть автоматически включены в вывод.

6.5. Реки: создайте словарь с названиями трех больших рек и стран, по которым протекает каждая река. Одна из возможных пар «ключ-значение» — `'nile': 'egypt'`.

- Используйте цикл для вывода сообщения с упоминанием реки и страны — например, «The Nile runs through Egypt».
- Используйте цикл для вывода названия каждой реки, включенной в словарь.
- Используйте цикл для вывода названия каждой страны, включенной в словарь.

6.6. Опрос: возьмите за основу код favorite_languages.py (с. 115).

- Создайте список людей, которые должны участвовать в опросе по поводу любимого языка программирования. Включите некоторые имена, которые уже присутствуют в списке, и некоторые имена, которых в списке еще нет.
- Переберите список людей, которые должны участвовать в опросе. Если они уже прошли опрос, выведите сообщение с благодарностью за участие. Если они еще не проходили опрос, выведите сообщение с предложением принять участие.

Вложение

Иногда бывает нужно сохранить множество словарей в списке или сохранить список как значение элемента словаря. Создание сложных структур такого рода называется *вложением*. Вы можете вложить множество словарей в список, список элементов в словарь или даже словарь внутрь другого словаря. Как наглядно показывают следующие примеры, вложение — чрезвычайно мощный механизм.

Список словарей

Словарь `alien_0` содержит разнообразную информацию об одном пришельце, но в нем нет места для хранения информации о втором пришельце, не говоря уже о целом экране, забитом пришельцами. Как смоделировать флот вторжения? Например, можно создать список пришельцев, в котором каждый элемент представляет собой словарь с информацией о пришельце. Например, следующий код строит список из трех пришельцев:

aliens.py

```
alien_0 = {'color': 'green', 'points': 5}
alien_1 = {'color': 'yellow', 'points': 10}
alien_2 = {'color': 'red', 'points': 15}
```

```
❶ aliens = [alien_0, alien_1, alien_2]

  for alien in aliens:
      print(alien)
```

Сначала создаются три словаря, каждый из которых представляет отдельного пришельца. В точке ❶ каждый словарь заносится в список с именем `aliens`. Наконец, программа перебирает список и выводит каждого пришельца:

```
{'color': 'green', 'points': 5}
{'color': 'yellow', 'points': 10}
{'color': 'red', 'points': 15}
```

Конечно, в реалистичном примере будут использоваться более трех пришельцев, которые будут генерироваться автоматически. В следующем примере функция `range()` создает флот из 30 пришельцев:

```
# Создание пустого списка для хранения пришельцев.
aliens = []

# Создание 30 зеленых пришельцев.
❶ for alien_number in range(30):
❷     new_alien = {'color': 'green', 'points': 5, 'speed': 'slow'}
❸     aliens.append(new_alien)

  # Вывод первых 5 пришельцев:
❹ for alien in aliens[:5]:
      print(alien)
  print("...")

  # Вывод количества созданных пришельцев.
❺ print(f"Total number of aliens: {len(aliens)}")
```

В начале примера список для хранения всех пришельцев, которые будут созданы, пуст. В точке ❶ функция `range()` возвращает множество чисел, которое просто сообщает Python, сколько раз должен повторяться цикл. При каждом выполнении цикла создается новый пришелец ❷, который затем добавляется в список `aliens` ❸. В точке ❹ сегмент используется для вывода первых пяти пришельцев, а в точке ❺ выводится длина списка (для демонстрации того, что программа действительно сгенерировала весь флот из 30 пришельцев):

```
{'speed': 'slow', 'color': 'green', 'points': 5}
{'speed': 'slow', 'color': 'green', 'points': 5}
{'speed': 'slow', 'color': 'green', 'points': 5}
{'speed': 'slow', 'color': 'green', 'points': 5}
{'speed': 'slow', 'color': 'green', 'points': 5}
...

Total number of aliens: 30
```

Все пришельцы обладают одинаковыми характеристиками, но Python рассматривает каждого пришельца как отдельный объект, что позволяет изменять атрибуты каждого владельца по отдельности.

Как работать с таким множеством? Представьте, что в этой игре некоторые пришельцы изменяют цвет и начинают двигаться быстрее. Когда приходит время смены цветов, мы можем воспользоваться циклом `for` и командой `if` для изменения цвета. Например, чтобы превратить первых трех пришельцев в желтых, двигающихся со средней скоростью и приносящих игроку по 10 очков, можно действовать так:

```
# Создание пустого списка для хранения пришельцев.
aliens = []

# Создание 30 зеленых пришельцев.
for alien_number in range (0,30):
    new_alien = {'color': 'green', 'points': 5, 'speed': 'slow'}
    aliens.append(new_alien)

for alien in aliens[0:3]:
    if alien['color'] == 'green':
        alien['color'] = 'yellow'
        alien['speed'] = 'medium'
        alien['points'] = 10

# Вывод первых 5 пришельцев:
for alien in aliens[0:5]:
    print(alien)
print("...")
```

Чтобы изменить первых трех пришельцев, мы перебираем элементы сегмента, включающего только первых трех пришельцев. В данный момент все пришельцы зеленые (`'green'`), но так будет не всегда, поэтому мы пишем команду `if`, которая гарантирует, что изменяться будут только зеленые пришельцы. Если пришелец зеленый, то его цвет меняется на желтый (`'yellow'`), скорость — на среднюю (`'medium'`), а награда увеличивается до 10 очков:

```
{'speed': 'medium', 'color': 'yellow', 'points': 10}
{'speed': 'medium', 'color': 'yellow', 'points': 10}
{'speed': 'medium', 'color': 'yellow', 'points': 10}
{'speed': 'slow', 'color': 'green', 'points': 5}
{'speed': 'slow', 'color': 'green', 'points': 5}
...
```

Цикл можно расширить, добавив блок `elif` для превращения желтых пришельцев в красных — быстрых и приносящих игроку по 15 очков. Мы не станем приводить весь код, а цикл выглядит так:

```
for alien in aliens[0:3]:
    if alien['color'] == 'green':
        alien['color'] = 'yellow'
        alien['speed'] = 'medium'
        alien['points'] = 10
```

```
    elif alien['color'] == 'yellow':
        alien['color'] = 'red'
        alien['speed'] = 'fast'
        alien['points'] = 15
```

Решение с хранением словарей в списке достаточно часто встречается тогда, когда каждый словарь содержит разные атрибуты одного объекта. Например, вы можете создать словарь для каждого пользователя сайта, как это было сделано в программе user.py на с. 114, и сохранить отдельные словари в списке с именем `users`. Все словари в списке должны иметь одинаковую структуру, чтобы вы могли перебрать список и выполнить с каждым объектом словаря одни и те же операции.

Список в словаре

Вместо того чтобы помещать словарь в список, иногда бывает удобно поместить список в словарь. Представьте, как бы вы описали в программе заказанную пиццу. Если ограничиться только списком, сохранить удастся разве что список топпингов к пицце. При использовании словаря список топпингов может быть всего лишь одним аспектом описания пиццы.

В следующем примере для каждой пиццы сохраняются два вида информации: основа и список топпингов. Список топпингов представляет собой значение, связанное с ключом `'toppings'`. Чтобы использовать элементы в списке, нужно указать имя словаря и ключ `'toppings'`, как и для любого другого значения в словаре. Вместо одного значения будет получен список топпингов:

pizza.py

```
  # Сохранение информации о заказанной пицце.
❶ pizza = {
      'crust': 'thick',
      'toppings': ['mushrooms', 'extra cheese'],
      }

  # Описание заказа.
❷ print(f"You ordered a {pizza['crust']}-crust pizza "
      "with the following toppings:")

❸ for topping in pizza['toppings']:
      print("\t" + topping)
```

Работа начинается в точке ❶ со словаря с информацией о заказанной пицце. С ключом в словаре `'crust'` связано строковое значение `'thick'`. С другим ключом `'toppings'` связано значение-список, в котором хранятся все заказанные топпинги. В точке ❷ выводится описание заказа перед созданием пиццы. Если вам нужно разбить длинную строку в вызове `print()`, выберите точку для разбиения выводимой строки и закончите строку кавычкой. Снабдите следующую строку отступом, добавьте открывающую кавычку и продолжите строку. Python автоматически объединяет все строки, обнаруженные в круглых скобках. Для вывода дополне-

ний пишется цикл for ❸. Чтобы вывести список топпингов, мы используем ключ 'toppings', а Python берет список топпингов из словаря.

Следующее сообщение описывает пиццу, которую мы собираемся создать:

```
You ordered a thick-crust pizza with the following toppings:
    mushrooms
    extra cheese
```

Вложение списка в словарь может применяться каждый раз, когда с одним ключом словаря должно быть связано более одного значения. Если бы в предыдущем примере с языками программирования ответы сохранялись в списке, один участник опроса мог бы выбрать сразу несколько любимых языков. При переборе словаря значение, связанное с каждым человеком, представляло бы собой список языков (вместо одного языка). В цикле for словаря создается другой цикл для перебора списка языков, связанных с каждым участником:

favorite_languages.py

```
❶ favorite_languages = {
      'jen': ['python', 'ruby'],
      'sarah': ['c'],
      'edward': ['ruby', 'go'],
      'phil': ['python', 'haskell'],
      }

❷ for name, languages in favorite_languages.items():
      print(f"\n{name.title()}'s favorite languages are:")
❸     for language in languages:
          print(f"\t{language.title()}")
```

Вы видите в точке ❶, что значение, связанное с каждым именем, теперь представляет собой список. У некоторых участников один любимый язык программирования, у других таких языков несколько. При переборе словаря в точке ❷ переменная с именем languages используется для хранения каждого значения из словаря, потому что мы знаем, что каждое значение будет представлять собой список. В основном цикле по элементам словаря другой цикл ❸ перебирает элементы списка любимых языков каждого участника. Теперь каждый участник опроса может указать сколько угодно любимых языков программирования:

```
Jen's favorite languages are:
    Python
    Ruby

Sarah's favorite languages are:
    C

Phil's favorite languages are:
    Python
    Haskell
```

```
Edward's favorite languages are:
    Ruby
    Go
```

Чтобы дополнительно усовершенствовать программу, включите в начало цикла `for` словаря команду `if` для проверки того, выбрал ли данный участник более одного языка программирования (проверка основана на значении `len(languages)`). Если у участника только один любимый язык, текст сообщения изменяется для единственного числа (например, «Sarah's favorite language is C»).

ПРИМЕЧАНИЕ Глубина вложения списков и словарей не должна быть слишком большой. Если вам приходится вкладывать элементы на глубину существенно большую, чем в предыдущих примерах, или если вы работаете с чужим кодом со значительной глубиной вложения, скорее всего, у задачи существует более простое решение.

Словарь в словаре

Словарь также можно вложить в другой словарь, но в таких случаях код быстро усложняется. Например, если на сайте есть несколько пользователей с уникальными именами, вы можете использовать имена пользователей как ключи в словаре. Информация о каждом пользователе при этом хранится в словаре, который используется как значение, связанное с именем. В следующем примере о каждом пользователе хранится три вида информации: имя, фамилия и место жительства. Чтобы получить доступ к этой информации, переберите имена пользователей и словарь с информацией, связанной с каждым именем:

many_users.py

```
  users = {
      'aeinstein': {
          'first': 'albert',
          'last': 'einstein',
          'location': 'princeton',
          },

      'mcurie': {
          'first': 'marie',
          'last': 'curie',
          'location': 'paris',
          },
      }

❶ for username, user_info in users.items():
❷     print(f"\nUsername: {username}")
❸     full_name = f"{user_info['first']} {user_info['last']}"
      location = user_info['location']
❹     print(f"\tFull name: {full_name.title()}")
      print(f"\tLocation: {location.title()}")
```

В программе определяется словарь с именем `users`, содержащий два ключа: для пользователей `'aeinstein'` и `'mcurie'`. Значение, связанное с каждым ключом, представляет собой словарь с именем, фамилией и местом жительства пользователя. В процессе перебора словаря `users` в точке ❶ Python сохраняет каждый ключ в переменной `username`, а словарь, связанный с каждым именем пользователя, сохраняется в переменной `user_info`. Внутри основного цикла в словаре выводится имя пользователя ❷.

В точке ❸ начинается работа с внутренним словарем. Переменная `user_info`, содержащая словарь с информацией о пользователе, содержит три ключа: `'first'`, `'last'` и `'location'`. Каждый ключ используется для построения аккуратно отформатированных данных с полным именем и местом жительства пользователя, с последующим выводом сводки известной информации о пользователе ❹:

```
Username: aeinstein
    Full name: Albert Einstein
    Location: Princeton

Username: mcurie
    Full name: Marie Curie
    Location: Paris
```

Обратите внимание на идентичность структур словарей всех пользователей. Хотя Python этого и не требует, наличие единой структуры упрощает работу с вложенными словарями. Если словари разных пользователей будут содержать разные ключи, то код в цикле `for` заметно усложнится.

УПРАЖНЕНИЯ

6.7. Люди: начните с программы, написанной для упражнения 6.1 (с. 113). Создайте два новых словаря, представляющих разных людей, и сохраните все три словаря в списке с именем `people`. Переберите элементы списка людей. В процессе перебора выведите всю имеющуюся информацию о каждом человеке.

6.8. Домашние животные: создайте несколько словарей, имена которых представляют клички домашних животных. В каждом словаре сохраните информацию о виде животного и имени владельца. Сохраните словари в списке с именем `pets`. Переберите элементы списка. В процессе перебора выведите всю имеющуюся информацию о каждом животном.

6.9. Любимые места: создайте словарь с именем `favorite_places`. Придумайте названия трех мест, которые станут ключами словаря, и сохраните для каждого человека от одного до трех любимых мест. Чтобы задача стала более интересной, опросите нескольких друзей и соберите реальные данные для своей программы. Переберите данные в словаре, выведите имя каждого человека и его любимые места.

6.10. Любимые числа: измените программу из упражнения 6.2 (с. 114), чтобы для каждого человека можно было хранить более одного любимого числа. Выведите имя каждого человека в списке и его любимые числа.

6.11. Города: создайте словарь с именем `cities`. Используйте названия трех городов в качестве ключей словаря. Создайте словарь с информацией о каждом городе; включите в него страну, в которой расположен город, примерную численность населения и один примеча-

тельный факт, относящийся к этому городу. Ключи словаря каждого города должны называться `country`, `population` и `fact`. Выведите название каждого города и всю сохраненную информацию о нем.

6.12. Расширение: примеры, с которыми мы работаем, стали достаточно сложными, и в них можно вносить разного рода усовершенствования. Воспользуйтесь одним из примеров этой главы и расширьте его: добавьте новые ключи и значения, измените контекст программы или улучшите форматирование вывода.

Итоги

В этой главе вы научились определять словари и работать с хранящейся в них информацией. Вы узнали, как обращаться к отдельным элементам словаря и изменять их, как перебрать всю информацию в словаре. Вы научились перебирать пары «ключ-значение», ключи и значения словаря. Также были рассмотрены возможности вложения словарей в список, вложения списков в словари и вложения словарей в другие словари.

В следующей главе будут рассмотрены циклы `while` и получение входных данных от пользователей программ. Эта глава будет особенно интересной, потому что вы наконец-то сможете сделать свои программы интерактивными: они начнут реагировать на действия пользователя.