DaneSoul 12 января 2017 в 10:06

Python: коллекции, часть 2/4: индексирование, срезы, сортировка

Python, Программирование

Tutorial

Часть 1	Часть 2	Часть 3	Часть 4

Последовательность	a	b	c	d	e	f	g	Результат слайсинга
Индексы	0 (-7)	1 (-6)	2 (-5)	3 (-4)	4 (-3)	5 (-2)	6 (-1)	
[:] (→)	+	+	+	+	+	+	+	abcdefg
[::-1] (←)	+	+	+	+	+	+	+	gfedcba
[::2] (→)	+		+		+		+	aceg
[1::2] (→)		+		+		+		bdf
[:1]	+							a
[-1:]							+	g
[3:4]				+				d
[-3:] (→)					+	+	+	efg
[-3:1:-1] (←)				+	+	+		edc
[2:5] (→)				+	+	+		cde

Данная статья является продолжением моей статьи "Python: коллекции, часть 1: классификация, общие подходы и методы, конвертация".

В данной статье мы продолжим изучать общие принципы работы со стандартными коллекциями (модуль collections в ней не рассматривается) Python.

Для кого: для изучающих Python и уже имеющих начальное представление о коллекциях и работе с ними, желающих систематизировать и углубить свои знания, сложить их в целостную картину.

ОГЛАВЛЕНИЕ:

1. Индексирование

1.1 Индексированные коллекции

Рассмотрим индексированные коллекции (их еще называют последовательности — sequences) — список (list), кортеж (tuple), строку (string).

Под индексированностью имеется ввиду, что элементы коллекции располагаются в определённом порядке, каждый элемент имеет свой индекс от 0 (то есть первый по счёту элемент имеет индекс не 1, а 0) до индекса на единицу меньшего длины коллекции (т.е. len(mycollection)-1).

1.2 Получение значения по индексу

Для всех индексированных коллекций можно получить значение элемента по его индексу в квадратных скобках. Причем, можно задавать отрицательный индекс, это значит, что будем находить элемент с конца считая обратном порядке.

При задании отрицательного индекса, последний элемент имеет индекс -1, предпоследний -2 и так далее до первого элемента индекс которого равен значению длины коллекции с отрицательным знаком, то есть (-len(mycollection).

элементы	a	b	c	d	e
индексы	0 (-5)	1 (-4)	2 (-3)	3 (-2)	4 (-1)

```
my_str = "abcde"
print(my_str[0])               # a - первый элемент
print(my_str[-1])              # e - последний элемент
print(my_str[len(my_str)-1])   # e - так тоже можно взять последний элемент
print(my_str[-2])              # d - предпоследний элемент
```

Наши коллекции могут иметь несколько уровней вложенности, как список списков в примере ниже. Для перехода на уровень глубже ставится вторая пара квадратных скобок и так далее.

```
my_2lvl_list = [[1, 2, 3], ['a', 'b', 'c']]
print(my_2lvl_list[0])      # [1, 2, 3] - первый элемент — первый вложенный список
print(my_2lvl_list[0][0])   # 1 — первый элемент первого вложенного списка
print(my_2lvl_list[1][-1])  # c — последний элемент второго вложенного списка
```

1.3 Изменение элемента списка по индексу

Поскольку кортежи и строки у нас неизменяемые коллекции, то по индексу мы можем только брать элементы, но не менять их:

```
my_tuple = (1, 2, 3, 4, 5)
print(my_tuple[0])        # 1
my_tuple[0] = 100         # TypeError: 'tuple' object does not support item assignment
```

А вот для списка, если взятие элемента по индексу располагается в левой части выражения, а далее идёт оператор присваивания =, то мы задаём новое значение элементу с этим индексом.

```
my_list = [1, 2, 3, [4, 5]]
my_list[0] = 10
my_list[-1][0] = 40
print(my_list)          # [10, 2, 3, [40, 5]]
```

UPD: Примечание: Для такого присвоения, элемент уже должен существовать в списке, нельзя таким образом добавить элемент на несуществующий индекс.

```
my_list = [1, 2, 3, 4, 5]
my_list[5] = 6      # IndexError: list assignment index out of range
```

2 Срезы

2.1 Синтаксис среза

Очень часто, надо получить не один какой-то элемент, а некоторый их набор ограниченный определенными простыми правилами — например первые 5 или последние три, или каждый второй элемент — в таких задачах, вместо перебора в цикле намного удобнее использовать так называемый срез (slice, slicing).

Следует помнить, что взяв элемент по индексу или срезом (slice) мы не как не меняем исходную коллекцию, мы просто скопировали ее часть для дальнейшего использования (например добавления в другую коллекцию, вывода на печать, каких-то вычислений). Поскольку сама коллекция не меняется — это применимо как к изменяемым (список) так и к неизменяемым (строка, кортеж) последовательностям.

Синтаксис среза похож на таковой для индексации, но в квадратных скобках вместо одного значения указывается 2-3 через двоеточие:

```
my_collection[start:stop:step]  # старт, стоп и шаг
```

Особенности среза:

- Отрицательные значения старта и стопа означают, что считать надо не с начала, а с конца коллекции.
- Отрицательное значение шага — перебор ведём в обратном порядке справа налево.

- Если не указан старт **[:stop:step]**— начинаем с самого края коллекции, то есть с первого элемента (включая его), если шаг положительный или с последнего (включая его), если шаг отрицательный (и соответственно перебор идет от конца к началу).
- Если не указан стоп **[start:: step]** — идем до самого края коллекции, то есть до последнего элемента (включая его), если шаг положительный или до первого элемента (включая его), если шаг отрицательный (и соответственно перебор идет от конца к началу).
- step = 1, то есть последовательный перебор слева направо указывать не обязательно — это значение шага по умолчанию. В таком случае достаточно указать **[start:stop]**
- Можно сделать даже так **[:]** — это значит взять коллекцию целиком
- **ВАЖНО**: ***При срезе, первый индекс входит в выборку, а второй нет!***
  То есть от старта включительно, до стопа, где стоп не включается в результат. Математически это можно было бы записать как [start, stop) или пояснить вот таким правилом:

  ```
  [ <первый включаемый> : <первый НЕ включаемый> : <шаг> ]
  ```

  Поэтому, например, mylist[::-1] не идентично mylist[:0:-1], так как в первом случае мы включим все элементы, а во втором дойдем до 0 индекса, но не включим его!

Примеры срезов в виде таблицы:

Последовательность	a	b	c	d	e	f	g	**Результат слайсинга**
Индексы	0 (-7)	1 (-6)	2 (-5)	3 (-4)	4 (-3)	5 (-2)	6 (-1)	
[:] (→)	+	+	+	+	+	+	+	abcdefg
[::-1] (←)	+	+	+	+	+	+	+	gfedcba
[::2] (→)	+		+		+		+	aceg
[1::2] (→)		+		+		+		bdf
[:1]	+							a
[-1:]							+	g
[3:4]				+				d
[-3:] (→)					+	+	+	efg
[-3:1:-1] (←)			+	+	+			edc
[2:5] (→)			+	+	+			cde

Код примеров из таблицы

2.2. Именованные срезы

Чтобы избавится от «магических констант», особенно в случае, когда один и тот же срез надо применять многократно, можно задать константы с именованными срезами с пользованием специальной функции **slice()**()

Примечание: None соответствует опущенному значению по-умолчанию. То есть [:2] становится slice(None, 2), а [1::2] становится slice(1, None, 2).

```
person = ('Alex', 'Smith', "May", 10, 1980)
NAME, BIRTHDAY = slice(None, 2), slice(2, None)
        # задаем константам именованные срезы
        # данные константы в квадратных скобках заменятся соответствующими срезами
print(person[NAME])      # ('Alex', 'Smith')
print(person[BIRTHDAY])  # ('May', 10, 1980)
```

```
my_list = [1, 2, 3, 4, 5, 6, 7]
EVEN = slice(1, None, 2)
print(my_list[EVEN])     # [2, 4, 6]
```

2.3 Изменение списка срезом

Важный момент, на котором не всегда заостряется внимание — с помощью среза можно не только получать копию коллекции, но в случае списка можно также менять значения элементов, удалять и добавлять новые.

Проиллюстрируем это на примерах ниже:

- Даже если хотим добавить один элемент, необходимо передавать итерируемый объект, иначе будет ошибка TypeError: can only assign an iterable

```
my_list = [1, 2, 3, 4, 5]
# my_list[1:2] = 20     # TypeError: can only assign an iterable
my_list[1:2] = [20]     # Вот теперь все работает
print(my_list)          # [1, 20, 3, 4, 5]
```

- Для вставки одиночных элементов можно использовать срез, код примеров есть ниже, но делать так не рекомендую, так как такой синтаксис хуже читать. Лучше использовать методы списка .append() и .insert():

  Срез аналоги .append() и insert()

- Можно менять части последовательности — это применение выглядит наиболее интересным, так как решает задачу просто и наглядно.

```
my_list = [1, 2, 3, 4, 5]
my_list[1:3] = [20, 30]
print(my_list)          # [1, 20, 30, 4, 5]
my_list[1:3] = [0]      # нет проблем заменить два элемента на один
print(my_list)          # [1, 0, 4, 5]
my_list[2:] = [40, 50, 60]   # или два элемента на три
print(my_list)               # [1, 0, 40, 50, 60]
```

- Можно просто удалить часть последовательности

```
my_list = [1, 2, 3, 4, 5]
my_list[:2] = []    # или del my_list[:2]
print(my_list)      # [3, 4, 5]
```

2.4 Выход за границы индекса

Обращение по индексу по сути является частным случаем среза, когда мы обращаемся только к одному элементу, а не диапазону. Но есть очень важное отличие в обработке ситуации с отсутствующим элементом с искомым индексом.

Обращение к несуществующему индексу коллекции вызывает ошибку:

```
my_list = [1, 2, 3, 4, 5]
print(my_list[-10])       # IndexError: list index out of range
print(my_list[10])        # IndexError: list index out of range
```

А в случае выхода границ среза за границы коллекции никакой ошибки не происходит:

```
my_list = [1, 2, 3, 4, 5]
print(my_list[0:10])      # [1, 2, 3, 4, 5] – отработали в пределах коллекции
```

```
print(my_list[10:100])    # [] - таких элементов нет — вернули пустую коллекцию
print(my_list[10:11])     # [] - проверяем 1 отсутствующий элемент - пустая коллекция, без ошибки
```

Примечание: Для тех случаев, когда функционала срезов недостаточно и требуются более сложные выборки, можно воспользоваться синтаксисом выражений-генераторов, рассмотрению которых посвещена 4 статья цикла.

3 Сортировка элементов коллекции

Сортировка элементов коллекции важная и востребованная функция, постоянно встречающаяся в обычных задачах. Тут есть несколько особенностей, на которых не всегда заостряется внимание, но которые очень важны.

3.1 Функция sorted()

Мы может использовать функцию sorted() для вывода списка сортированных элементов любой коллекции для последующее обработки или вывода.

- функция не меняет исходную коллекцию, а возвращает новый список из ее элементов;
- не зависимо от типа исходной коллекции, вернётся список (list) ее элементов;
- поскольку она не меняет исходную коллекцию, ее можно применять к неизменяемым коллекциям;
- Поскольку при сортировке возвращаемых элементов нам не важно, был ли у элемента некий индекс в исходной коллекции, можно применять к неиндексированным коллекциям;
- Имеет дополнительные не обязательные аргументы:
  reverse=True — сортировка в обратном порядке
  key=funcname (начиная с Python 2.4) — сортировка с помощью специальной функции funcname, она может быть как стандартной функцией Python, так и специально написанной вами для данной задачи функцией и лямбдой.

```
my_list = [2, 5, 1, 7, 3]
my_list_sorted = sorted(my_list)
print(my_list_sorted)       # [1, 2, 3, 5, 7]

my_set = {2, 5, 1, 7, 3}
my_set_sorted = sorted(my_set, reverse=True)
print(my_set_sorted)        # [7, 5, 3, 2, 1]
```

Пример сортировки списка строк по длине len() каждого элемента:

```
my_files = ['somecat.jpg', 'pc.png', 'apple.bmp', 'mydog.gif']
my_files_sorted = sorted(my_files, key=len)
print(my_files_sorted)      # ['pc.png', 'apple.bmp', 'mydog.gif', 'somecat.jpg']
```

3.2 Функция reversed()

Функция reversed() применяется для последовательностей и работает по другому:

- возвращает генератор списка, а не сам список;
- если нужно получить не генератор, а готовый список, результат можно обернуть в list() или же вместо reversed() воспользоваться срезом [: :-1];
- она **не сортирует** элементы, а возвращает их в обратном порядке, то есть читает с конца списка;
- из предыдущего пункта понятно, что если у нас коллекция неиндексированная — мы не можем вывести её элементы в обратном порядке и эта функция к таким коллекциям не применима — получим «TypeError: argument to reversed() must be a sequence»;

- не позволяет использовать дополнительные аргументы — будет ошибка «TypeError: reversed() does not take keyword arguments».

```
my_list = [2, 5, 1, 7, 3]
my_list_sorted = reversed(my_list)
print(my_list_sorted)           # <listreverseiterator object at 0x7f8982121450>
print(list(my_list_sorted))     # [3, 7, 1, 5, 2]
print(my_list[::-1])            # [3, 7, 1, 5, 2] - тот же результат с помощью среза
```

3.3 Методы списка .sort() и .reverse()

У списка (и только у него) есть особые методы .sort() и .reverse() которые делают тоже самое, что соответствующие функции sorted() и reversed(), но при этом:

- Меняют сам исходный список, а не генерируют новый;
- Возвращают None, а не новый список;
- поддерживают те же дополнительные аргументы;
- в них не надо передавать сам список первым параметром, более того, если это сделать — будет ошибка — не верное количество аргументов.

```
my_list = [2, 5, 1, 7, 3]
my_list.sort()
print(my_list)          # [1, 2, 3, 5, 7]
```

Обратите внимание: Частая ошибка начинающих, которая не является ошибкой для интерпретатора, но приводит не к тому результату, который хотят получить.

```
my_list = [2, 5, 1, 7, 3]
my_list = my_list.sort()
print(my_list)          # None
```

3.4 Особенности сортировки словаря

В сортировке словаря есть свои особенности, вызванные тем, что элемент словаря — это пара ключ: значение.

UPD: Так же, не забываем, что говоря о сортировке словаря, мы имеем ввиду сортировку полученных из словаря данных для вывода или сохранения в индексированную коллекцию. Сохранить данные сортированными в самом стандартном словаре не получится, они в нем, как и других неиндексированных коллекциях находятся в произвольном порядке.

- sorted(my_dict) — когда мы передаем в функцию сортировки словарь без вызова его дополнительных методов — идёт перебор только ключей, сортированный список ключей нам и возвращается;
- sorted(my_dict.**keys**()) — тот же результат, что в предыдущем примере, но прописанный более явно;
- sorted(my_dict.**items**()) — возвращается сортированный список кортежей (ключ, значение), сортированных по ключу;
- sorted(my_dict.**values**()) — возвращается сортированный список значений

```
my_dict = {'a': 1, 'c': 3, 'e': 5, 'f': 6, 'b': 2, 'd': 4}
mysorted = sorted(my_dict)
print(mysorted)           # ['a', 'b', 'c', 'd', 'e', 'f']
mysorted = sorted(my_dict.items())
print(mysorted)           # [('a', 1), ('b', 2), ('c', 3), ('d', 4), ('e', 5), ('f', 6)]
mysorted = sorted(my_dict.values())
print(mysorted)           # [1, 2, 3, 4, 5, 6]
```

Отдельные сложности может вызвать **сортировка словаря** не по ключам, а **по значениям**, если нам не просто нужен список значений, и именно выводить пары в порядке сортировки по значению.

Для решения этой задачи можно в качестве специальной функции сортировки передавать lambda-функцию **lambda x: x[1]** которая из получаемых на каждом этапе кортежей (ключ, значение) будет брать для сортировки второй элемент кортежа.

```
population = {"Shanghai": 24256800, "Karachi": 23500000, "Beijing": 21516000, "Delhi": 16787941}
# отсортируем по возрастанию населения:
population_sorted = sorted(population.items(), key=lambda x: x[1])
print(population_sorted)
# [('Delhi', 16787941), ('Beijing', 21516000), ('Karachi', 23500000), ('Shanghai', 24256800)]
```

UPD от @ShashkovS: 3.5 Дополнительная информация по использованию параметра key при сортировке

Допустим, у нас есть список кортежей названий деталей и их стоимостей.
Нам нужно отсортировать его сначала по названию деталей, а одинаковые детали по убыванию цены.

```
shop = [('каретка', 1200), ('шатун', 1000), ('седло', 300),
        ('педаль', 100), ('седло', 1500), ('рама', 12000),
        ('обод', 2000), ('шатун', 200), ('седло', 2700)]

def prepare_item(item):
    return (item[0], -item[1])

shop.sort(key=prepare_item)
```

Результат сортировки

Перед тем, как сравнивать два элемента списка к ним применялась функция prepare_item, которая меняла знак у стоимости (функция применяется ровно по одному разу к каждому элементу. В результате при одинаковом первом значении сортировка по второму происходила в обратном порядке.

Чтобы не плодить утилитарные функции, вместо использования сторонней функции, того же эффекта можно добиться с использованием лямбда-функции.

```
# Данные скопировать из примера выше
shop.sort(key=lambda x: (x[0], -x[1]))
```

Дополнительные детали и примеры использования параметра key:

- wiki.python.org/moin/HowTo/Sorting#Key_Functions (на английском).
- habrahabr.ru/post/319200/#comment_10011882 — еще один комментарий с детальным примером @ShashkovS к данной статье

UPD от @ShashkovS: 3.6 Устойчивость сортировки

Допустим данные нужно отсортировать сначала по столбцу A по возрастанию, затем по столбцу B по убыванию, и наконец по столбцу C снова по возрастанию.

Если данные в столбце B числовые, то при помощи подходящей функции в key можно поменять знак у элементов B, что приведёт к необходимому результату.
А если все данные текстовые? Тут есть такая возможность.
Дело в том, что сортировка sort в Python устойчивая (начиная с Python 2.2), то есть она не меняет порядок «одинаковых» элементов.

Поэтому можно просто отсортировать три раза по разным ключам:

```
data.sort(key=lambda x: x['C'])
data.sort(key=lambda x: x['B'], reverse=True)
data.sort(key=lambda x: x['A'])
```

Дополнительная информация по устойчивости сортировки и примеры:
wiki.python.org/moin/HowTo/Sorting#Sort_Stability_and_Complex_Sorts (на наглийском).

Часть 1	Часть 2	Часть 3	Часть 4

Приглашаю к обсуждению:

- Если я где-то допустил неточность или не учёл что-то важное — пишите в комментариях, важные комментарии будут позже добавлены в статью с указанием вашего авторства.
- Если какие-то моменты не понятны и требуется уточнение — пишите ваши вопросы в комментариях — или я или другие читатели дадут ответ, а дельные вопросы с ответами будут позже добавлены в статью.

Теги: python, программирование, начинающим, коллекции, структуры данных, индексация, слайсинг, срез, сортировка

Хабы: Python, Программирование

47,5 Карма 1,2 Рейтинг

Александр @DaneSoul

Веб-программирование, Python

ПОХОЖИЕ ПУБЛИКАЦИИ

18 апреля 2020 в 11:15

Теоретические структуры данных и их применение в JavaScript. Ч1. Пары

+2 3,2k 60 8

19 января 2017 в 00:00

Python: коллекции, часть 3/4: объединение коллекций, добавление и удаление элементов

+20 31,8k 214 24

9 января 2017 в 14:22

Python: коллекции, часть 1/4: классификация, общие подходы и методы, конвертация

+38 91,8k 491 27

ВОПРОСЫ И ОТВЕТЫ

Как исправить ошибку AttributeError: __name__ Django?
Django • Простой • 0 ответов

Какими способами можно решить задачу через Python?
Python • Простой • 2 ответа

Получить изображение из буфера обмена?