

Функции max/min/sorted и использование ключа сортировки

Рассмотрим еще один полезный специальный синтаксис в Python, позволяющий избавиться от промежуточных итераторов, которые исходно нам не даны и не нужны в итоговом результате. Так мы сможем сократить число неуклюжих конструкций, в которых сначала создается сложная структура, а потом эта структура упрощается обратно.

Параметр key

У функций вроде min/max/sorted есть опциональный (необязательный) параметр key. Параметр key принимает функцию, по значению которой будут сравниваться элементы.

Например, пусть у нас есть набор слов, который мы хотим отсортировать:

```
words = ['мир', 'и', 'война']
```

Отсортировать слова можно различными способами. Если мы применим функцию sorted без аргумента key, слова будут отсортированы как в словаре (это называется лексикографически):

```
sorted(words) # => ['война', 'и', 'мир']
```

Теперь давайте вызовем функцию sorted следующим образом:

```
sorted(words, key=lambda s: len(s))  
# => ['и', 'мир', 'война']
```

Мы указали, что в качестве ключа для сортировки должны использоваться не сами строки (встроенное в Python сравнение строк — лексикографическое), а их длины. Таким образом, мы получаем список, отсортированный по возрастанию длины слова.

Очень удобно использовать ключ сортировки, если нам надо отсортировать список упорядоченных коллекций (списков, кортежей, строк). Например, у нас есть список, элементами которого тоже являются списки, которые содержат название фильма, его возрастное ограничение и рейтинг по отзывам критиков. И мы хотим отсортировать его сначала по возрастному ограничению, затем по оценке критиков (по убыванию) и только в конце по названию. В этом случае нам поможет вот такой код:

```
li = [  
    ['Crawl', 'R', 61],  
    ['Stuber', 'R', 42],  
    ['Midsommar', 'R', 73],  
    ['Yesterday', 'PG-13', 56],
```

```

['Annabelle Comes Home', 'R', 53],
["Child's Play", 'R', 48],
['Anna', 'R', 40],
['Toy Story 4', 'G', 84],
['Shaft', 'R', 40],
['Men in Black: International', 'PG-13', 38]
]
print(*sorted(li, key=lambda x: (x[1], -x[2], x[0])), sep='\n')
['Toy Story 4', 'G', 84]
['Yesterday', 'PG-13', 56]
['Men in Black: International', 'PG-13', 38]
['Midsommar', 'R', 73]
['Crawl', 'R', 61]
['Annabelle Comes Home', 'R', 53]
["Child's Play", 'R', 48]
['Stuber', 'R', 42]
['Anna', 'R', 40]
['Shaft', 'R', 40]

```

Помимо функции `sorted`, параметр `key` принимают функции `max` и `min`. Вызов `max(values, key)` позволяет найти значение из набора `values`, наибольшее по ключу `key`.

Проверка коллекций: `all`, `any`

При работе с коллекциями часто приходится определять, выполняется ли некоторое условие одновременно для всех элементов коллекции или хотя бы для одного.

`all` и `any`

Для этих целей существуют две встроенные функции: `all` и `any`. Первая проверяет, что все элементы переданного ей итерируемого набора значений истинны (приводятся к `True`). Вторая проверяет, что есть хотя бы один такой элемент. В терминах математической логики эти функции — кванторы общности и существования.

В качестве единственного аргумента `all` и `any` принимают что-нибудь перечисляемое — например, список, кортеж или итератор.

Итак, `all` вернет `True` в том случае, если все элементы аргумента `True` или приводятся к `True` (или если коллекция пустая):

```

print(all([1, 2, 3, 4, 5, 6, 7, 8, 9]))
# True - так как все элементы ненулевые

print(all([1, 2, 3, 4, 5, 6, 7, 8, 0]))
# False - есть ноль

print(all([1, 2, 3, 4, 5, 6, [], set()]))
# False - есть пустые вложенные коллекции

```

```
print(all([])) # True
```

Функция `any` вернет True, если истинен хотя бы один элемент аргумента. `any` возвращает False для пустых коллекций:

```
print(any((set(), [], {}, 0, True)))  
# True - есть True среди элементов
```

```
print(any([set(), [], {}, 0, [1, 2, 3]]))  
# True - непустой список приводится к True
```

```
print(any([set(), [], {}, 0, False]))  
# False - все элементы приводятся к False
```

```
print(any([])) # False
```

Функции `all` и `any` могут быть особенно полезны в комбинации с функцией `map`, которая для каждого элемента коллекции проверит некоторое условие и вернет итератор, в котором будут перечисляться результаты этих проверок. Так, например, можно проверить, все ли числа в списке четные:

```
data = [1, 2, 3, 4, 5]  
print(all(x % 2 == 0 for x in data))  
False
```

А так — узнать, есть ли среди слов хотя бы одно, длиной 5 букв или более:

```
words = "Ехал грека через реку".split()  
print(any(map(lambda w: len(w) >= 5, words)))  
True
```

Потоковый ввод `stdin`

В Python есть очень полезный встроенный итерируемый объект: `sys.stdin`. Это — итератор так называемого потока ввода.

Поток ввода (`stdin`) — специальный объект в программе, куда попадает весь текст, который ввел пользователь. Поток его называют потому, что данные хранятся там до тех пор, пока программа их не считала. Данные поступают в программу и временно «складируются» в потоке ввода, а программа может «забрать» их оттуда, например, при помощи функции `input()`. В момент прочтения они пропадают из потока ввода: он хранит данные «до востребования».

sys.stdin

`sys.stdin` — пример итератора, который невозможно перезапустить. Как и любой итератор, он может двигаться только вперед. Но если для списка можно сделать второй итератор, который начнет чтение с начала списка, то с потоком ввода такое не пройдет. Как только данные прочитаны, они удаляются из потока ввода безвозвратно.

Элементы, которые выдает этот итератор, — строки, введенные пользователем. Если пользовательский ввод закончен, итератор тоже прекращает работу. Пока пользователь не ввел последнюю строку, мы не знаем, сколько элементов в итераторе.

Хочется обратить ваше внимание на один интересный факт: допустим, вы написали программу, которая дважды вызывает функцию `input()`, и отправили ее на проверку в тестовую систему. Но тестовая система передает лишь одну строку. В этом случае выполнение программы завершится с ошибкой, поскольку функция `input()` не смогла ничего прочитать.

Поэтому, если вы не знаете, в какой момент надо прекратить ввод, воспользоваться функцией `input()` не удастся. В таких случаях остается только работать с `sys.stdin`.

Чтобы работать с `sys.stdin`, прежде всего необходимо подключить модуль `sys` командой `import sys`. Напишем небольшую программу, которая печатает каждую введенную пользователем строку:

```
import sys
for line in sys.stdin:
    # rstrip('\n') "отрезает" от строки line идущий справа символ
    # перевода строки, ведь print сам переводит строку
    print(line.rstrip('\n'))
```

Что происходит?

Пока есть данные в потоке `sys.stdin` (то есть пока пользователь их вводит), программа будет получать вводимые строки в переменную `line`, убирать справа символы перевода строки и выводить их на печать.

Но если вы запустите эту программу, она будет работать вечно. Чтобы показать, что ввод закончен, пользователю недостаточно нажать Enter — компьютер не знает, завершил пользователь работу или будет еще что-то вводить (при этом Enter превратится в пустую строку). Вместо этого вы должны нажать Ctrl + D (если работаете в консоли Linux или IDE PyCharm) либо Ctrl + Z, затем Enter (если работаете в консоли Windows).

Если вы работаете в IDE Wing, кликните правой кнопкой мыши и выберите Send EOF, затем нажмите Enter. Это запишет в поток ввода специальный символ EOF (end of file), который отмечает конец ввода.

Мы обещали показать, что функция `input` выдает ошибку, если не получает ввод. Напишите простую программу:

```
x, y = input(), input()
```

Запустите программу и введите одну строку (не забудьте нажать Enter). Вместо второй строки введите EOF тем способом, которым это делается в вашей системе. Вы увидите ошибку EOFError — это означает, что input пытается считать данные из потока, который закончился.

Ввод в одну строку

С помощью sys.stdin можно в одну строку прочитать весь ввод (о количестве строк которого мы ничего не знаем) в список. Реализуется это, например, так:

```
data = list(map(str.strip, sys.stdin))
```

Кроме того, можно считать все строки (с сохранением символов перевода строки) в список вот таким образом:

```
data = sys.stdin.readlines()
```

А считать многострочный текст из стандартного потока ввода в текстовую переменную можно вот так:

```
str_data = sys.stdin.read()
```