16 Полезных односложных выражений Python для упрощения обычных задач

15.06.2023

Хотите писать элегантный и питонический код? Вот список полезных однострочных фраз Python для выполнения простых задач. Если вы начинающий программист Python, вы потратите время на понимание основных структур данных, таких как списки и строки. А некоторые операции над этими структурами данных можно выполнить с помощью лаконичных однострочных фрагментов кода. Как программист, вы должны отдавать предпочтение читабельности и удобству сопровождения, а не сокращению кода. Но в Python легко придумать однострочные фрагменты, которые соответствуют хорошей практике кодирования. В этой статье мы сосредоточимся на однострочниках для простых задач обработки списков и строк в Python. Давайте приступим!

Генерация списка чисел

Самый простой способ создания списка чисел – это использование функции range(). Функция range() возвращает объект range, который можно преобразовать в список. Использование range(num) даст последовательность 0, 1, 2,…, num-1.

```
>>> nums = list(range(10))
>>> nums
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Подробнее об использовании функции range()

*Вы также можете использовать функцию range() вместе с необязательным значением шага. Так, range (start, end, step) даст последовательность start, start + step, start + 2*step и так далее. Последним значением будет start + k*step такое, что (start + k*step) < end и (start + (k+1)*step) > end.*

Поиск максимального и минимального значений в списке

Вы можете использовать встроенные функции max и min для вычисления максимального и минимального элементов в списке, соответственно.

```
>>> min_elt, max_elt = min(nums), max(nums)
>>> min_elt
0
>>> max_elt
9
```

> ***Замечание о множественном присваивании***
>
> *Обратите внимание, что мы присвоили значения min_elt и max_elt в одном операторе присваивания. Python поддерживает такое множественное присваивание. Это может быть полезно при распаковке итераций и присвоении значений нескольким переменным одновременно.*

Удаление дубликатов из списка

Еще одна распространенная операция – удаление дубликатов из списков Python. Это необходимо, когда нужно работать только с уникальными значениями. Самый простой способ сделать это – преобразовать список в множество. Множество – это встроенная структура данных, все элементы которой уникальны и хэшируемы.

```
>>> nums1 = [2,4,7,9,7,10]
```

В nums1 элемент 7 встречается дважды. Преобразование в множество удалит дубликат (здесь 7), оставив нам список уникальных значений. Поскольку нам все еще нужно работать со списком, мы преобразуем множество обратно в список. Эту операцию можно выполнить с помощью следующей строки кода:

```
>>> nums1 = list(set(nums1))
>>> nums1
[2, 4, 7, 9, 10]
```

Подсчет повторений в списке

Чтобы подсчитать, сколько раз элемент встречается в списке, можно использовать встроенный метод count(). list.count(elt) возвращает количество раз, когда elt встречается в списке.

```
>>> nums
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Здесь 7 встречается один раз в списке nums, поэтому метод count() возвращает 1.

```
>>> nums.count(7)
1
```

Проверьте, все ли элементы списка удовлетворяют условию

Чтобы проверить, все ли элементы списка удовлетворяют условию, можно использовать встроенную в Python функцию all(). Функция all() принимает в качестве аргумента итерабельную переменную и

возвращает True, если все элементы итерабельной переменной имеют значение True (или являются истинными). Здесь мы хотим проверить, все ли элементы в списке nums2 нечетные.

```
>>> nums2 = [3,4,7,11,21,67,12]
```

Мы можем использовать понимание списка для построения списка булевых чисел и передать этот список в качестве аргумента функции all(). Здесь num%2!=0 будет False для элементов 4 и 12, которые являются четными. Поэтому список булевых чисел, построенный с помощью выражения list comprehension, содержит False (и all(list) возвращает False).

```
>>> all([num%2!=0 for num in nums2])
False
```

> *Важно отметить, что all([]) (all(any-empty-iterable) возвращает True.*

Проверьте, удовлетворяет ли условию любой элемент в списке

Чтобы проверить, удовлетворяет ли условию любой элемент списка, вы можете использовать функцию any(). any(some-list) возвращает True, если хотя бы один элемент оценивается как True.

```
>>> nums2 = [3,4,7,11,21,67,12]
```

Как и в предыдущем примере, мы используем понимание списка, чтобы получить список булевых чисел. Список nums содержит четные числа. Поэтому функция any() возвращает True.

```
>>> any([num%2 for num in nums2])
True
```

Обратное преобразование строки

В Python строки неизменяемы, поэтому, когда вы хотите обратить строку, вы можете получить только обратную копию строки. Есть два распространенных подхода – оба могут быть написаны в виде однострочных команд Python – которые используют нарезку строк и встроенные функции.

Использование нарезки строк

Нарезка строк с отрицательными значениями шага возвращает фрагмент строки, начиная с конца. Синтаксис: string[start:stop:step]. Что же возвращает установка step равным -1 и игнорирование индексов start и stop? Возвращается копия строки, начиная с конца строки, включая каждый символ.

```
>>> str1[::-1]
'olleh'
```

Использование функции reversed()

Встроенная функция reversed() возвращает обратный итератор по последовательности.

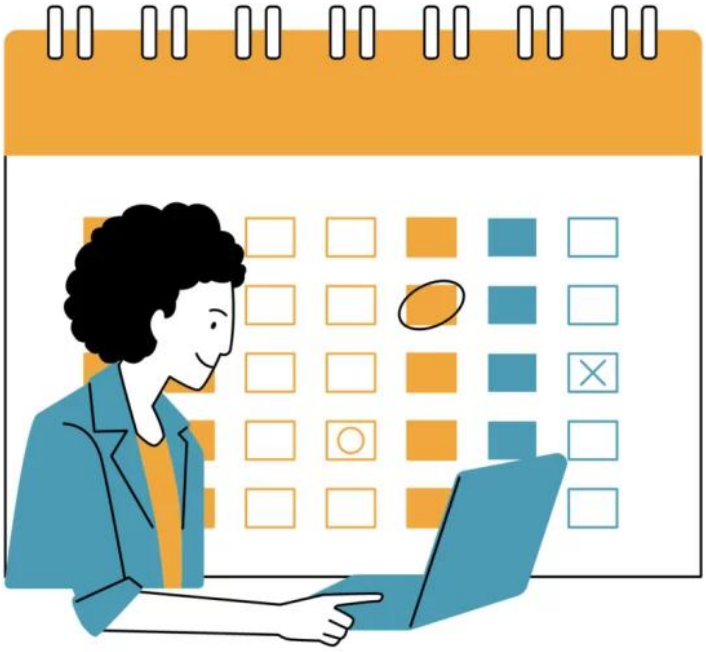
```
>>> reversed(str1)
<reversed object at 0x008BAF70>
>>> for char in str1:
...     print(char)
...
h
e
l
l


o
```

Вы можете использовать его в сочетании с методом join(), как показано на рисунке:

```
>>> ''.join(reversed(str1))
```

```
'olleh'
```

Преобразование строки в список символов



Предположим, мы хотим разделить строку на список символов. Мы можем сделать это с помощью выражения list comprehension.

```
>>> str1 = 'hello'
```

Понимание списков — один из самых мощных однолинейных инструментов в Python. Мы проходим по строке и собираем каждый символ.

```
>>> chars = [char for char in str1]
>>> chars
['h', 'e', 'l', 'l', 'o']
```

Чтобы разбить строку на список символов, можно также использовать конструктор list():

```
>>> str1 = 'hello'
>>> chars = list(str1)
>>> chars
['h', 'e', 'l', 'l', 'o']
```

Чтобы разделить длинную строку на список строк при каждом появлении пробела, можно использовать метод split().

```
>>> str2 = 'hello world'
>>> str2.split()

['hello', 'world']
```

Извлечение цифр из строки

Мы уже видели, как использовать понимание списка для разбиения строки на список символов. В том примере мы собрали все символы, выполнив цикл по строке. Здесь нам нужно пройтись по строке и собрать только цифры. Как же нам это сделать?

- Мы можем задать условие фильтрации в выражении для понимания списка с помощью метода isdigit().
- c.isdigit() возвращает True, если c является цифрой; в противном случае возвращается False.

```
>>> str3 = 'python3'
>>> digits = [c for c in str3 if c.isdigit()]
>>> digits

['3']
```

Проверка, начинается ли строка с определенной подстроки

Чтобы проверить, начинается ли строка с определенной подстроки, вы можете использовать метод startswith() string. str1.startswith(substring) возвращает True, если str1 начинается с подстроки. В противном случае возвращается False.

Вот несколько примеров:

```
>>> str4 = 'coding'
>>> str4.startswith('co')
```

```
True
```

```
>>> str5 = 'python'
>>> str5.startswith('co')
```

```
False
```

Проверка, заканчивается ли строка определенной подстрокой

Как вы уже догадались, чтобы проверить, заканчивается ли строка заданной подстрокой, вы можете использовать метод endswith().

```
>>> str5 = 'python'
>>> str5.endswith('on')
```

```
True
```

Мы также можем использовать метод string внутри выражения для понимания списка, чтобы получить ends_with, список булевых значений.

```
>>> strs = ['python','neon','nano','silicon']
```

```
>>> ends_with = [str.endswith('on') for str in strs]
>>> ends_with
```

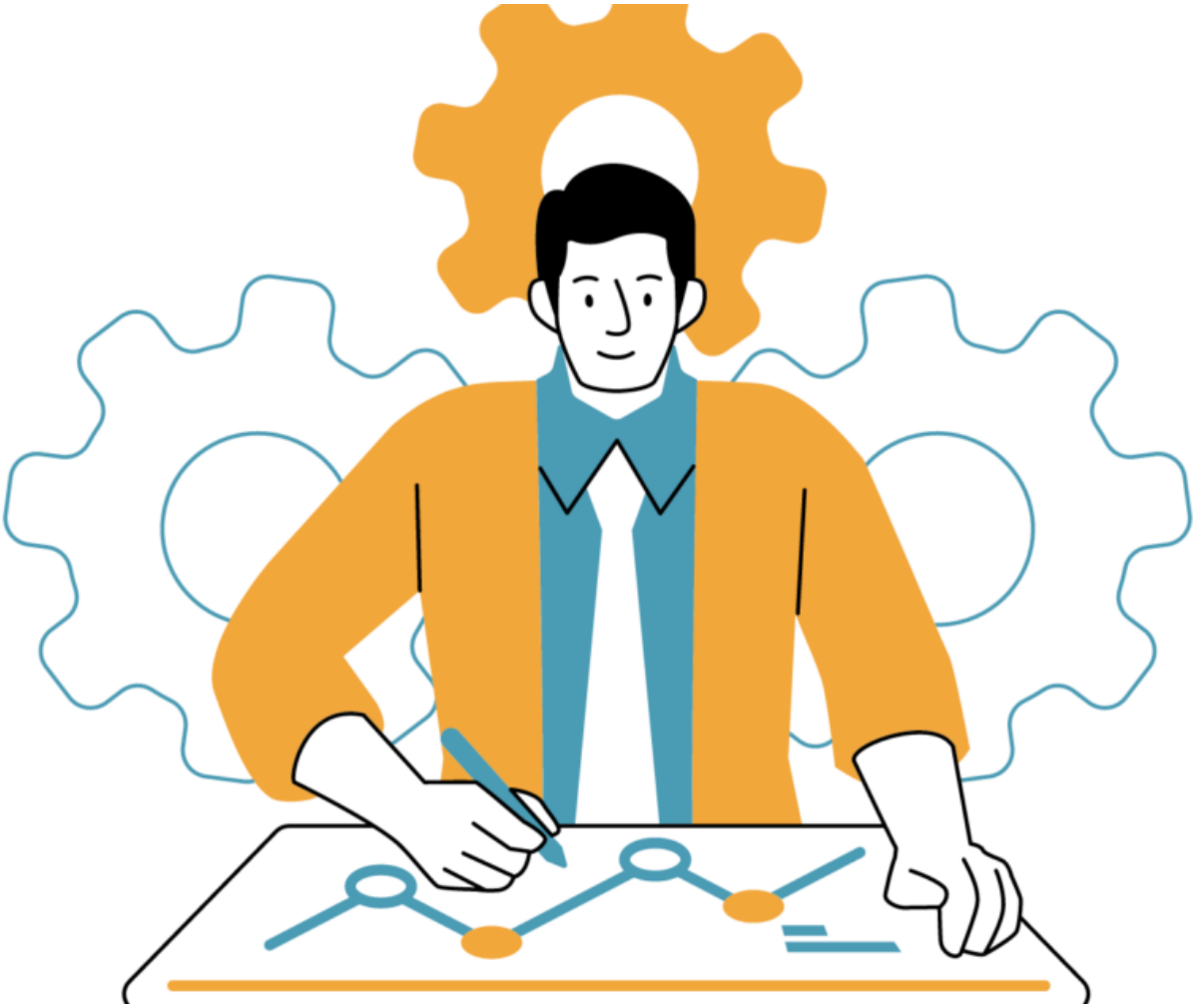
```
[True, True, False, True]
```

Объединение элементов списка в строку

Мы уже видели, как разделить строку на список символов. Теперь как выполнить обратную операцию – объединить элементы списка в строку? Для этого можно использовать строковый метод join() с синтаксисом: separator.join(some-list). Мы хотим объединить только элементы списка в одну строку; нам не нужен никакой разделитель. Поэтому мы задаем разделителем пустую строку.

```
>>> list_1 = ['p','y','t','h','o','n','3']
>>> ''.join(list_1)

'python3'
```

Создание словаря Python



Подобно тому, как понимания списков могут помочь нам построить новые списки из существующих итераций, понимания словарей могут помочь нам построить новые словари из существующих итераций. Постижение словарей в Python — это мощный однострочный инструмент, который может помочь создать словарь “на лету”. Здесь у нас есть имена, которые представляют собой список строк.

```
>>> names = ['Joe','Amy','Jake','Florence']
```

Мы создаем names_d, словарь, содержащий строки имен в качестве ключей и длины строк в качестве значений.

```
>>> names_d = {name:len(name) for name in names}
>>> names_d

{'Joe': 3, 'Amy': 3, 'Jake': 4, 'Florence': 8}
```

Условное присвоение значений переменным

Иногда вам может понадобиться присвоить значения переменным в зависимости от определенного условия. Например, вы можете прочитать в пользовательском вводе, скажем, возраст человека. И в зависимости от введенного значения вы можете решить, разрешено ли ему присутствовать на вечеринке. Чтобы выполнить такое условное присваивание в Python, вы можете написать следующую однострочную фразу с использованием тернарного оператора.

```
>>> age = 21
>>> allowed = True if age >= 18 else False
>>> allowed

True
```

Генерировать все перестановки

Перестановка относится к возможному расположению элементов в группе. Если в группе есть n уникальных элементов, то существует n! возможных способов их расположения – следовательно, n! перестановок. Давайте воспользуемся следующим списком букв:

```
>>> letters = ['a','b','c']
```

Вы можете использовать перестановки из модуля itertools для генерации всех возможных перестановок заданной итерации.

```
>>> letters_p = permutations(letters)
>>> letters_p
```

```
<itertools.permutations object at 0x0127AF50>
```

Как видно, использование permutations(iterable) возвращает объект перестановки, который мы можем просмотреть с помощью цикла for:

```
>>> for p in letters_p:
...     print(p)
...
('a', 'b', 'c')
('a', 'c', 'b')
('b', 'a', 'c')
('b', 'c', 'a')
('c', 'a', 'b')

('c', 'b', 'a')
```

Но мы можем переписать его как однострочное выражение, приведя объект перестановки к списку:

```
>>> letters_p = list(permutations(letters))
>>> letters_p
[('a', 'b', 'c'), ('a', 'c', 'b'), ('b', 'a', 'c'), ('b', 'c', 'a'), ('c', 'a', 'b'), ('c', 'b', 'a')]
```

Здесь есть три уникальных элемента, и существует 3!=6 возможных перестановок.

Генерация подмножеств списка

Иногда вам может понадобиться построить все возможные подмножества определенного размера из списка или других итераций. Возьмем список букв и получим все подсписки размера 2. Для этого мы можем использовать комбинации из модуля itertools, например, так:

```
>>> from itertools import combinations

>>> letters_2 = list(combinations(letters,2))
>>> letters_2
[('a', 'b'), ('a', 'c'), ('b', 'c')]
```

Заключение

В этом уроке мы рассмотрели полезные односложные выражения Python для выполнения распространенных операций над списками и строками. Мы также познакомились с такими однострочными операторами, как Python list и dictionary comprehensions, и узнали, как использовать их в сочетании со встроенными функциями для выполнения необходимых задач.