*перевод*

10 Секретов и Уловок в Работе с Python, которые Вам Необходимо Выучить Прямо Сейчас

Короткие советы для Python, способные улучшить Вашу повседневную работу



Фотография принадлежит Rifqi Ali Ridho на Unsplash

. . .

В соответствии с данными сайта Stack Overflow, на сегодняшний день Python является самым быстрорастущим языком программирования. Последний отчет Forbs заявляет, что доля Python на рынке выросла на 456 процентов за прошлый год. Python используют Netflix, IBM и ещё сотни других компаний. Не стоит также забывать о Dropbox, написанном на Python. Исследования Dice показывают, что знание Python является весьма полезным навыком, а сам Python – это самый популярный язык программирования на сегодня исходя из оценки Popularity of Programming Language Index.

Основные преимущества Python перед конкурентами:

1. Совместимость с основными платформами и операционными сетями
2. Множество open-source фреймворков и инструментов
3. Легко читаемый и поддерживаемый код
4. Продвинутая стандартная библиотека
5. Стандартная тестовая разработка

. . .

Секреты и Уловки Python

В этой главе я представляю 10 полезных советов по работе с кодом, которые могут пригодиться Вам в повседневной работе. Что ж, без лишних слов, приступим.

1. Объединение строк

Когда Вам требуется объединить список из строк, можно использовать цикл for и складывать элемент за элементом. Однако это сделает Вашу программу неэффективной, особенно если список элементов достаточно длинный. В Python строки являются неизменными, и поэтому левые и правые строки придётся копировать в новую строку за каждую пару для объединения.

Поэтому наилучшим решением будет использовать функцию join(), как показано ниже:

characters = ['p', 'y', 't', 'h', 'o', 'n']  
word = "".join(characters)  
print(word) **# python**

2. List Comprehension

Функции List Comprehension (или Генератор списков) используют для создания новых списков из повторяющихся объектов. Так как List Comprehensions возвращают списки, они состоят из квадратных скобок, содержащих выражение, которое выполняется для каждого элемента на ряду с циклом for для итерации по каждому элементу. Генератор списков выполняется быстрее, поскольку он оптимизирован для интерпретатора Python и настроен определять возможный паттерн во время цикла.

К примеру, давайте найдем квадраты первых пяти целых чисел, используя генератор списков.

m = [x \*\* 2 for x in range(5)]  
print(m) **# [0, 1, 4, 9, 16]**

А теперь давайте найдем повторяющиеся числа в двух списках, используя генератор списков.

3. Итерации с enumerate()

Метод Enumerate() добавляет счетчик в итерацию и возвращает его в виде enumerate объекта.

Давайте решим проблему под названием Fizz Buzz - классический вопрос на любом собеседовании.

Напишите программу, которая выводит на экран числа из списка. При этом вместо чисел, кратных трем, программа должна выводить слово Fizz, а вместо чисел, кратных пяти — слово Buzz. Если число кратно и тому, и тому , то программа должна выводить слово FizzBuzz.

numbers = [30, 42, 28, 50, 15]  
for i, num in enumerate(numbers):  
if num % 3 == 0 and num % 5 == 0:  
numbers[i] = 'fizzbuzz'  
elif num % 3 == 0:  
numbers[i] = 'fizz'  
elif num % 5 == 0:  
numbers[i] = 'buzz'  
print(numbers) **# ['fizzbuzz', 'fizz', 28, 'buzz', 'fizzbuzz']**

. . .

**4. Использование ZIP в работе со списками**

Предположим, вы получили задание соединить несколько списков одинаковой длины и вывести результат. Ниже представлен общий способ решения, где используется метод zip():

countries = ['France', 'Germany', 'Canada']  
capitals = ['Paris', 'Berlin', 'Ottawa']  
for country, capital in zip(countries,capitals):  
print(country, capital) **# France Paris   
Germany Berlin  
Canada Ottawa**

. . .

1. Использование Itertools

В Python Модуль Itertools — это коллекция инструментов для обработки циклов. Itertools включает в себя много инструментов для создания повторяющихся последовательностей входящих данных. В данном примере я буду использовать метод itertools.combinations (), используемый для выстраивания комбинаций. Им так же можно группировать входящие значения.

Для прояснения вышесказанного давайте посмотрим на реальный пример из жизни.

Представьте четыре команды, учавствующие в турнире. В лиге каждая команда соревнуется с остальными. Ваша задача сгенерировать все возможные команды, способные конкурировать друг с другом.

Взгляните на данный код:

import itertools

friends = ['Team 1', 'Team 2', 'Team 3', 'Team 4']

list(itertools.combinations(friends, r=2)) # [('Team 1', 'Team 2'), ('Team 1', 'Team 3'), ('Team 1', 'Team 4'), ('Team 2', 'Team 3'), ('Team 2', 'Team 4'), ('Team 3', 'Team 4')]

Важно запомнить, что последовательность значений не важна. Потому что ('Team 1', 'Team 2') и ('Team 2', 'Team 1') определяют одну и ту же пару, только одна из них попадет в конечный список. Таким же образом мы можем использовать как itertools.permutations () , так и другие функции из модуля. Для более полной картины взгляните на этот шикарный туториал.

. . .

1. Использование Python коллекций

Коллекции Python — это контейнеры типов данных, а именно списки, сеты, кортежи, словари. Коллекции предоставляют высоко производительные типы данных, которые могут расширить функционал вашего кода, сделать его чище и проще. Они предоставляют большое количество функций. Для примера я использую функцию Counter ().

Функция Counter () принимает на вход повторяющиеся данные, кортежи, например, и возвращает Counter Dictionary (Словарь типа Counter). Ключи в словаре должны быть уникальными элементами, и значения каждого ключа должно быть номером элемента в данной последовательности.

Чтобы создать обьект типа counter поместите последовательность (лист) в функцию Counter (), как показано ниже:

from collections import Countercount = Counter(['a','b','c','d','b','c','d','b'])

print(count) # Counter({'b': 3, 'c': 2, 'd': 2, 'a': 1})

Для более полной картины, взгляните на мой туториал по коллекциям в Python.

. . .

1. Преобразование двух списков в Cловарь

Представим, что у нас есть два списка, один из которых содержит имена учеников, а второй – их оценки. Давайте попробуем преобразовать их в единый словарь. Сделать это можно, используя функцию zip:

students = ["Peter", "Julia", "Alex"]  
marks = [84, 65, 77]  
dictionary = dict(zip(students, marks))  
print(dictionary) **# {'Peter': 84, 'Julia': 65, 'Alex': 77}**

. . .

1. Использование Генераторов Python

Генераторные функции позволяют Вам объявить функцию, которая ведет себя по принципу итератора. Такие функции позволяют программистам быстро, легко и опрятно создавать итераторы. Чтобы было наглядно, возьмем пример:

Представьте, что Вас попросили найти сумму первого 100000000 идеальных квадратов начиная с 1.

Звучит просто, не так ли? Такую задачку можно было бы с лёгкостью решить, используя List Comprehension, однако возникает одна проблема: вводимые данные имеют слишком большой размер. Взгляните на код ниже:

t1 = time.clock()  
sum([i \* i for i in range(1, 100000000)])  
t2 = time.clock()  
time\_diff = t2 - t1  
print(f"It took {time\_diff} Secs to execute this method") **# It took 13.197494000000006 Secs to execute this method**

**При увеличении идеальных чисел, которые нам нужно сложить, мы понимаем, что этот метод неосуществим из-за чрезвычайно длительного времени вычисления. Именно здесь к нам на помощь приходит Генераторы Python. При замене квадратных скобок на круглые мы изменяем List Comprehension на генераторную функцию. А теперь давайте подсчитаем время исполнения программы:**

t1 = time.clock()  
sum((i \* i for i in range(1, 100000000)))  
t2 = time.clock()  
time\_diff = t2 - t1  
print(f"It took {time\_diff} Secs to execute this method") **# It took 9.53867000000001 Secs to execute this method**

Как мы видим, время немного сократилось. Этот эффект будет еще лучше заметен при использовании более крупных входных данных.

. . .

9. Каким образом функция возвращает сразу несколько значений?

В Python у нас есть возможность принимать несколько значений с вызова функции, это то, чего лишены остальные популярные языки программирования. В этом случае возвращаемое значение должно быть списком со значениями, разделенными запятыми. Python выстроит кортеж и вернет его вызывающей функции. Вы можете увидеть это в коде ниже:

def multiplication\_division(num1, num2):

return num1\*num2, num1/num2

product, division = multiplication\_division(15, 3)

print("Product=", product, "Quotient =", division) # Product= 45 Quotient = 5.0

. . .

10. Использование функции sorted ()

Сортировка любой последовательности — это очень простая задача для Python, особенно если использовать встроенный метод sorted (), который сделает всю сложную работу за Вас. Sorted () сортирует любую последовательность (список, кортеж) и всегда возвращает список с отсортированными элементами. Давайте, к примеру, отсортируем список данных по возрастанию:

sorted([3,5,2,1,4]) # [1, 2, 3, 4, 5]

А также по убыванию:

sorted(['france', 'germany', 'canada', 'india', 'china'], reverse=True) # ['india', 'germany', 'france', 'china', 'canada']

. . .

Заключение

В этой статье я показал вам 10 секретов и уловок в Python, которые могут быть использованы в повседневной работе. Надеюсь, Вам понравилась данная статья. Оставайтесь с нами, ведь скоро выйдет еще одна статья - "Советы, как ускорить ваш код на Python".

*оригинал*

# 10 Python Tips and Tricks You Should Learn Today

## Python snippets that can be taken as a reference for your daily work.

According to Stack Overflow, Python is the fastest growing programming language. The latest report from Forbes states that Python showed a 456-percent growth in last year. Netflix uses Python, IBM uses Python, and hundreds of other companies all use Python. Let’s not forget Dropbox. Dropbox is also created in Python. According to research by Dyce Python is also one of the hottest skills to have and also the most popular programming language in the world based on the Popularity of Programming Language Index.

Some of the advantages Python offers when compared to other programming languages are:

1. Compatible with major platforms and operating systems
2. Many open-source frameworks and tools
3. Readable and maintainable code
4. Robust standard library
5. Standard test-driven development

# Python Tips and Tricks

In this piece, I’ll present 10 useful code tips and tricks that can help you in your day-to-day tasks. So without further ado, let’s get started.

# 1. Concatenating Strings

When you need to concatenate a list of strings, you can do this using a for loop by adding each element one by one. However, this would be very inefficient, especially if the list is long. In Python, strings are immutable, and thus the left and right strings would have to be copied into the new string for every pair of concatenation.

A better approach is to use the join() function as shown below:

characters = ['p', 'y', 't', 'h', 'o', 'n']  
word = "".join(characters)  
print(word) **# python**

# 2. Using List Comprehensions

List comprehensions are used for creating new lists from other iterables. As list comprehensions returns lists, they consist of brackets containing the expression, which is executed for each element along with the for loop to iterate over each element. List comprehension is faster because it is optimized for the Python interpreter to spot a predictable pattern during looping.

As an example let’s find the squares of the first five whole numbers using list comprehensions.

m = [x \*\* 2 for x in range(5)]  
print(m) **# [0, 1, 4, 9, 16]**

Now let’s find the common numbers from two list using list comprehension

list\_a = [1, 2, 3, 4]  
list\_b = [2, 3, 4, 5]  
common\_num = [a for a in list\_a for b in list\_b if a == b]  
print(common\_num) **# [2, 3, 4]**

# 3. Iterate With enumerate()

Enumerate() method adds a counter to an iterable and returns it in a form of enumerate object.

Let’s solve the classic coding interview question named popularly as the Fizz Buzz problem.

*Write a program that prints the numbers in a list, for multiples of ‘3’ print “fizz” instead of the number, for the multiples of ‘5’ print “buzz” and for multiples of both 3 and 5 it prints “fizzbuzz”.*

numbers = [30, 42, 28, 50, 15]  
for i, num in enumerate(numbers):  
 if num % 3 == 0 and num % 5 == 0:  
 numbers[i] = 'fizzbuzz'  
 elif num % 3 == 0:  
 numbers[i] = 'fizz'  
 elif num % 5 == 0:  
 numbers[i] = 'buzz'  
print(numbers) **# ['fizzbuzz', 'fizz', 28, 'buzz', 'fizzbuzz']**

# 4. Using ZIP When Working with Lists

Suppose you were given a task to combine several lists with the same length and print out the result? Again, here is a more generic way to get the desired result by utilizing zip()as shown in the code below:

countries = ['France', 'Germany', 'Canada']  
capitals = ['Paris', 'Berlin', 'Ottawa']  
for country, capital in zip(countries,capitals):  
 print(country, capital) **# France Paris   
 Germany Berlin  
 Canada Ottawa**

# 5. Using itertools

The Python itertools module is a collection of tools for handling iterators. itertools has multiple tools for generating iterable sequences of input data. Here I will be using itertools.combinations() as an example. itertools.combinations() is used for building combinations. These are also the possible groupings of the input values.

Let’s take a real world example to make the above point clear.

*Suppose there are four teams playing in a tournament. In the league stages every team plays against every other team. Your task is to generate all the possible teams that would compete against each other.*

Let’s take a look at the code below:

import itertools  
friends = ['Team 1', 'Team 2', 'Team 3', 'Team 4']  
list(itertools.combinations(friends, r=2)) **# [('Team 1', 'Team 2'), ('Team 1', 'Team 3'), ('Team 1', 'Team 4'), ('Team 2', 'Team 3'), ('Team 2', 'Team 4'), ('Team 3', 'Team 4')]**

The important thing to notice is that order of the values doesn’t matter. Because ('Team 1', 'Team 2') and ('Team 2', 'Team 1') represent the same pair, only one of them would be included in the output list. Similarly we can use itertools.permutations() as well as other functions from the module.

# 6. Using Python Collections

Python collections are container data types, namely lists, sets, tuples, dictionary. The collections module provides high-performance datatypes that can enhance your code, making things much cleaner and easier. There are a lot of functions provided by the collections module. For this demonstration, I will be using Counter() function.

The Counter() function takes an iterable, such as a list or tuple, and returns a Counter Dictionary. The dictionary’s keys will be the unique elements present in the iterable, and the values for each key will be the count of the elements present in the iterable.

To create a counter object, pass an iterable (list) to Counter() function as shown in the code below.

from collections import Countercount = Counter(['a','b','c','d','b','c','d','b'])  
print(count) **# Counter({'b': 3, 'c': 2, 'd': 2, 'a': 1})**

# 7. Convert Two Lists Into a Dictionary

Let’s say we have two lists, one list contains names of the students and second contains marks scored by them. Let’s see how we can convert those two lists into a single dictionary. Using the zip function, this can be done using the code below:

students = ["Peter", "Julia", "Alex"]  
marks = [84, 65, 77]  
dictionary = dict(zip(students, marks))  
print(dictionary) **# {'Peter': 84, 'Julia': 65, 'Alex': 77}**

# 8. Using Python Generators

Generator functions allow you to declare a function that behaves like an iterator. They allow programmers to make an iterator in a fast, easy, and clean way. Let’s take an example to explain this concept.

*Suppose you’ve been given to find the sum of the first 100000000 perfect squares, starting with 1.*

Looks easy right? This can easily be done using list comprehension but the problem is the large inputs size. As an example let’s take a look at the below code:

t1 = time.clock()  
sum([i \* i for i in range(1, 100000000)])  
t2 = time.clock()  
time\_diff = t2 - t1  
print(f"It took {time\_diff} Secs to execute this method") **# It took 13.197494000000006 Secs to execute this method**

On increasing the perfect numbers we need to sum over, we realize that this method is not feasible due to higher computation time. Here’s where Python Generators come to the rescue. On replacing the brackets with parentheses we change the list comprehension into a generator expression. Now let’s calculate the time taken:

t1 = time.clock()  
sum((i \* i for i in range(1, 100000000)))  
t2 = time.clock()  
time\_diff = t2 - t1  
print(f"It took {time\_diff} Secs to execute this method") **# It took 9.53867000000001 Secs to execute this method**

As we can see, time taken has been reduced quite a bit. This effect will be even more pronounced for larger inputs.

# 9. Return Multiple Values From a Function

Python has the ability to return multiple values from a function call, something missing from many other popular programming languages. In this case the return values should be a comma-separated list of values and Python then constructs a tuple and returns this to the caller. As an example see the code below:

def multiplication\_division(num1, num2):  
 return num1\*num2, num1/num2product, division = multiplication\_division(15, 3)  
print("Product=", product, "Quotient =", division) **# Product= 45 Quotient = 5.0**

# 10. Using sorted() Function

Sorting any sequence is very easy in Python using the built-in method sorted()which does all the hard work for you. sorted()sorts any sequence (list, tuple) and always returns a list with the elements in sorted manner. Let’s take an example to sort a list of numbers in ascending order.

sorted([3,5,2,1,4]) **# [1, 2, 3, 4, 5]**

Taking another example, let’s sort a list of strings in descending order.

sorted(['france', 'germany', 'canada', 'india', 'china'], reverse=True) **# ['india', 'germany', 'france', 'china', 'canada']**

# Conclusions

In this article, I’ve presented 10 Python tips and tricks that can be used as a reference in your day-to-day work. I hope you enjoyed this article. Stay tuned for my next piece, “Tips and Tricks to Speed Up Your Python Code”.