Tuples

Введение в кортежи

Кортежи

Мы изучили списки и строки. Списки – изменяемые коллекции, строки – неизменяемые последовательности Unicode символов. В Python имеются и неизменяемые последовательности, содержащие, в отличие от строк, абсолютно произвольные данные. Такие коллекции называются **кортежами** (`tuple`, читается "тюпл").

Рассмотрим следующий программный код:

```
my_list = [1, 2, 3, 4, 5]
```

Мы объявили список чисел и присвоили его переменной `my_list`. Содержимое списка можно изменять.

Следующий программный код:

```
my_list = [1, 2, 3, 4, 5]
my_list[0] = 9
my_list[4] = 7
print(my_list)
```

выведет:

```
[9, 2, 3, 4, 7]
```

Заменив квадратные скобки при объявлении списка на круглые, мы объявляем кортеж:

```
my_tuple = (1, 2, 3, 4, 5)
```

Кортежи по своей природе (задумке) – **неизменяемые аналоги списков**. Поэтому программный код:

```
my_tuple = (1, 2, 3, 4, 5)
my_tuple[0] = 9
my_tuple[4] = 7
print(my_tuple)
```

приводит к ошибке

```
TypeError: 'tuple' object does not support item assignment
```

Кортеж (`tuple`) – ещё один вид коллекций в Python. Они похожи на списки, но являются **неизменяемыми**.

В литеральной форме кортеж записывается в виде последовательности элементов в круглых скобках, а список – в квадратных.

Примеры кортежей

```
empty_tuple = ()                          # пустой кортеж
point = (1.5, 6.0)                        # кортеж из двух чисел
names = ('Timur', 'Ruslan', 'Roman')      # кортеж из трех строк
info = ('Timur', 'Guev', 28, 170, 60, False)   # кортеж из 6 элементов разных типов
nested_tuple = (('one', 'two'), ['three', 'four'])   # кортеж из кортежа и списка
```

- в переменной `empty_tuple` хранится пустой кортеж;
- в переменной `point` хранится кортеж, состоящий из двух вещественных чисел (такой кортеж удобно использовать для представления точки на координатной плоскости);
- в переменной `names` хранится кортеж, содержащий три строковых значения;
- в переменной `info` содержится кортеж, содержащий 6 элементов разного типа (строки, числа, булевы переменные);
- в переменной `nested_tuple` содержится кортеж, содержащий другой кортеж и список.

Кортежи могут хранить и содержать в себе объекты любых типов (даже составных) и поддерживают неограниченное количество уровней вложенности.

Кортеж с одним элементом

Для создания кортежа с единственным элементом после значения элемента ставят замыкающую **запятую**:

```
my_tuple = (1,)
print(type(my_tuple))  # <class 'tuple'>
```

Если запятую пропустить, то кортеж создан не будет. Например, приведенный ниже код просто присваивает переменной `my_tuple` целочисленное значение `1`:

```
my_tuple = (1)
print(type(my_tuple))  # <class 'int'>
```

Зачем использовать кортеж вместо списка?

Списки могут делать то же, что кортежи, и даже больше. Но **неизменяемость кортежей** обеспечивает им особые свойства:

- **скорость** – кортежи быстрее работают, так как из-за неизменяемости хранятся в памяти иначе, и операции с их элементами выполняются заведомо быстрее, чем с компонентами списка. Одна из причин существования кортежей – производительность. Обработка кортежа выполняется быстрее, чем обработка списка, поэтому кортежи удобны для обработки большого объема неизменяемых данных.
- **безопасность** – неизменяемость превращает их в идеальные константы. Заданные кортежами константы делают код более читаемым и безопасным. Кроме того, в кортеже можно безопасно хранить данные, не опасаясь, что они будут случайно или преднамеренно изменены в программе.

В Python существуют операции, требующие применения кортежа. По мере освоения языка Python вы будете чаще сталкиваться с кортежами.

Примечания

Примечание 1. Мы уже сталкивались с кортежами, когда изучали функции, возвращающие несколько значений. Такие функции возвращают именно кортежи.

Рассмотрим функцию `get_powers()`, которая принимает в качестве аргумента число и возвращает его 2, 3 и 4 степень.

```
def get_powers(num):
    return num**2, num**3, num**4
```

Результатом выполнения следующего кода:

```
result = get_powers(5)
print(type(result))
print(result)
```

будет:

```
<class 'tuple'>
(25, 125, 625)
```

Примечание 2. Списки предназначены для объединения неопределенного количества однородных сущностей. Кортежи, как правило, объединяют под одним именем несколько разнородных объектов, имеющих различный смысл.

Например, список удобен для хранения нескольких городов:

```
cities = ["Perth", "San Francisco", "Lisbon", "Sochi"]  # список городов
```

А кортеж удобен для хранения данных о людях:

```
person = ("Tony", 21, "Auckland")  # кортеж с данными о человеке: имя, возраст, город
```

Примечание 3. Тот факт, что кортеж является неизменяемым вовсе не означает, что мы не можем поменять содержимое списка в кортеже.

Приведенный ниже код:

```
my_tuple = (1, 'python', [1, 2, 3])
print(my_tuple)
my_tuple[2][0] = 100
my_tuple[2].append(17)
print(my_tuple)
```

выводит:

```
(1, 'python', [1, 2, 3])
(1, 'python', [100, 2, 3, 17])
```

При этом важно понимать: меняется список, а не кортеж. Списки являются ссылочными типами данных, поэтому в кортеже хранится ссылка на список, которая не меняется при изменении самого списка.

Как кортеж хранится в памяти

В Python tuple является неизменяемым (immutable) типом данных, что означает, что после его создания вы не можете изменить его содержимое (в отличие от списков, которые являются изменяемыми). Внутреннее представление tuple в памяти оптимизировано для эффективности и скорости доступа к его элементам. Давайте рассмотрим, как tuple хранится в памяти.

1. **Ссылочная структура**: Как и большинство составных типов данных в Python, tuple хранит ссылки на объекты, которые он содержит, а не сами объекты. Это означает, что при создании кортежа создается структура, которая содержит указатели на объекты, находящиеся в других областях памяти. Это позволяет tuple эффективно хранить элементы различных типов данных.

2. **Фиксированный размер**: Поскольку tuple неизменяем, его размер задается при создании и не может быть изменен. Это означает, что Python выделяет в памяти фиксированный блок для хранения ссылок на содержащиеся в tuple объекты. Фиксированный размер также способствует тому, что доступ к элементам tuple осуществляется быстрее, чем к элементам списка, чья длина может изменяться.

3. **Оптимизация памяти**: Благодаря неизменяемости, tuple может быть оптимизирован для эффективного использования памяти. Например, маленькие кортежи могут быть кэшированы и повторно использованы в различных частях программы без необходимости выделения дополнительной памяти для их создания.

4. **Быстрый доступ**: Структура данных tuple оптимизирована для быстрого доступа к элементам. Доступ к элементу кортежа осуществляется за время O(1), что делает чтение данных из кортежа очень эффективным по сравнению с некоторыми другими структурами данных.

Важно отметить, что, хотя tuple и хранит только ссылки на объекты, эти объекты могут быть изменяемыми. Это означает, что если tuple содержит список, вы не можете изменить сам tuple (например, добавить или удалить из него элемент), но можете изменить содержимое списка, который он содержит. Таким образом, неизменяемость tuple относится к структуре кортежа и его размеру, но не обязательно к изменяемости объектов, на которые он ссылается.

Основы работы с кортежами. Часть 1

Функция tuple()

Встроенная функция list() может применяться для преобразования **кортежа в список**.

Приведенный ниже код:

```
number_tuple = (1, 2, 3, 4, 5)
number_list = list(number_tuple)
print(number_list)
```

выводит:

```
[1, 2, 3, 4, 5]
```

Встроенная функция tuple() может применяться для преобразования **списка в кортеж**.

Приведенный ниже код:

```
str_list = ['один', 'два', 'три']
str_tuple = tuple(str_list)
print(str_tuple)
```

выводит:

```
('один', 'два', 'три')
```

Аналогичным образом мы можем создать кортеж на основании строки.

Приведенный ниже код:

```
text = 'hello python'
str_tuple = tuple(text)
print(str_tuple)
```

выводит:

```
('h', 'e', 'l', 'l', 'o', ' ', 'p', 'y', 't', 'h', 'o', 'n')
```

Обратите внимание, что символ пробела содержится в кортеже str_tuple.

Преобразование строки в список позволяет получить список символов строки. Это может быть полезно, например, когда надо изменить один символ строки:

```
s = 'симпотичный'
print(s)

a = list(s)
a[4] = 'a'
s = ''.join(a)
print(s)
```

Приведенный выше код выводит:

симпотичный
симпатичный

С этой же целью может потребоваться преобразование кортежа в список:

```
writer = ('Лев Толстой', 1827)
print(writer)

a = list(writer)
a[1] = 1828
writer = tuple(a)
print(writer)
```

Приведенный выше код выводит:

```
('Лев Толстой', 1827)
('Лев Толстой', 1828)
```

Особенности кортежей

Кортежи поддерживают те же операции, что и списки, за исключением изменяющих содержимое.

Кортежи поддерживают:

- доступ к элементу по индексу (только для получения значений элементов);
- методы, в частности index(), count();
- встроенные функции, в частности len(), sum(), min() и max();
- срезы;
- оператор принадлежности in;
- операторы конкатенации (+) и повторения (*).

Функция len()

Длиной кортежа называется количество его элементов. Для того, чтобы посчитать длину кортежа, мы используем встроенную функцию len().

Следующий программный код:

```
numbers = (2, 4, 6, 8, 10)
languages = ('Python', 'C#', 'C++', 'Java')

print(len(numbers))      # выводим длину кортежа numbers
print(len(languages))    # выводим длину кортежа languages

print(len(('apple', 'banana', 'cherry')))   # выводим длину кортежа, состоящего из 3 элементов
```

выведет:

```
5
4
3
```

Оператор принадлежности in

Оператор in позволяет проверить, содержит ли кортеж некоторый элемент.

Рассмотрим следующий код:

```
numbers = (2, 4, 6, 8, 10)

if 2 in numbers:
    print('Кортеж numbers содержит число 2')
else:
    print('Кортеж numbers не содержит число 2')
```

Такой код проверяет, содержит ли кортеж numbers число 2 и выводит соответствующий текст:

Кортеж numbers содержит число 2

Мы можем использовать оператор in вместе с логическим оператором not. Например

```
numbers = (2, 4, 6, 8, 10)

if 0 not in numbers:
    print('Кортеж numbers не содержит нулей')
```

Индексация

При работе со списками (строками) мы использовали **индексацию**, то есть обращение к конкретному элементу списка (строки) по его индексу. Аналогично можно индексировать и элементы кортежей.

Для индексации кортежей в Python используются квадратные скобки [], в которых указывается индекс (номер) нужного элемента в кортеже:

Пусть numbers = (2, 4, 6, 8, 10).

Таблица ниже показывает, как работает индексация:

Выражение	Результат	Пояснение
numbers[0]	2	первый элемент кортежа
numbers[1]	4	второй элемент кортежа
numbers[2]	6	третий элемент кортежа
numbers[3]	8	четвертый элемент кортежа
numbers[4]	10	пятый элемент кортежа

Так же, как и в списках, для нумерации с конца разрешены отрицательные индексы:

Выражение	Результат	Пояснение
numbers[-1]	10	пятый элемент кортежа
numbers[-2]	8	четвертый элемент кортежа
numbers[-3]	6	третий элемент кортежа
numbers[-4]	4	второй элемент кортежа
numbers[-5]	2	первый элемент кортежа

Как и в списках, попытка обратиться к элементу кортежа по несуществующему индексу:

```
print(numbers[17])
```

вызовет ошибку:

```
IndexError: tuple index out of range
```

Срезы

Рассмотрим кортеж numbers = (2, 4, 6, 8, 10).

С помощью среза мы можем получить несколько элементов кортежа, создав диапазон индексов, разделенных двоеточием numbers[x:y].

Следующий программный код:

```
print(numbers[1:3])
print(numbers[2:5])
```

выводит:

```
(4, 6)
(6, 8, 10)
```

При построении среза numbers[x:y] первое число – это то место, где начинается срез (**включительно**), а второе – это место, где заканчивается срез (**невключительно**).

При использовании срезов с кортежами мы также можем опускать второй параметр в срезе numbers[x:] (но поставить двоеточие), тогда срез берется до конца кортежа. Аналогично, если опустить первый параметр numbers[:y], то можно взять срез от начала кортежа.

Срез numbers[:] возвращает копию исходного кортежа.

Как и в списках, можно использовать отрицательные индексы в срезах кортежей.

Операция конкатенации + и умножения на число *

Операторы + и * применяют для кортежей, как и для списков.

Следующий программный код:

```
print((1, 2, 3, 4) + (5, 6, 7, 8))
print((7, 8) * 3)
print((0,) * 10)
```

выводит:

```
(1, 2, 3, 4, 5, 6, 7, 8)
(7, 8, 7, 8, 7, 8)
(0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
```

Для генерации кортежей, состоящих строго из повторяющихся элементов, умножение на число — самый короткий и правильный путь.

Расширенные операторы += и *= также можно использовать при работе с кортежами.

Следующий программный код:

```
a = (1, 2, 3, 4)
b = (7, 8)
a += b   # добавляем к кортежу a кортеж b
b *= 5   # повторяем кортеж b 5 раз

print(a)
print(b)
```

выводит:

```
(1, 2, 3, 4, 7, 8)
(7, 8, 7, 8, 7, 8, 7, 8, 7, 8)
```

Встроенные функции sum(), min(), max()

Встроенная функция sum() принимает в качестве параметра кортеж чисел и вычисляет сумму его элементов.

Следующий программный код:

```
numbers = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
print('Сумма всех элементов кортежа =', sum(numbers))
```

выводит:

```
Сумма всех элементов кортежа = 55
```

Встроенные функции min() и max() принимают в качестве параметра кортеж и находят минимальный и максимальный элементы соответственно.

Следующий программный код:

```
numbers = (3, 4, 10, 3333, 12, -7, -5, 4)
print('Минимальный элемент кортежа =', min(numbers))
print('Максимальный элемент кортежа =', max(numbers))
```

выводит:

```
Минимальный элемент кортежа = -7
Максимальный элемент кортежа = 3333
```

Функции min() и max() можно применять только к кортежам с одним типом данных. Если кортеж содержит разные типы данных, скажем, целое число (int) и строку (str), то во время выполнения программы произойдет ошибка.

Метод index()

Метод index() возвращает индекс первого элемента, значение которого равняется переданному в метод значению. Таким образом, в метод передается один параметр:

- value: значение, индекс которого требуется найти.

Если элемент в кортеже не найден, то во время выполнения происходит ошибка.

Следующий программный код:

```
names = ('Gvido', 'Roman' , 'Timur')
position = names.index('Timur')

print(position)
```

выведет:

```
2
```

Следующий программный код:

```
names = ('Gvido', 'Roman' , 'Timur')
position = names.index('Anders')
print(position)
```

приводит к ошибке:

```
ValueError: tuple.index(x): x not in tuple
```

Чтобы избежать таких ошибок, можно использовать метод index() вместе с оператором принадлежности in:

```
names = ('Gvido', 'Roman', 'Timur')

if 'Anders' in names:
    position = names.index('Anders')
    print(position)
else:
    print('Такого значения нет в кортеже')
```

Метод count()

Метод count() возвращает количество элементов в кортеже, значения которых равны переданному в метод значению.

Таким образом, в метод передается один параметр:

- value: значение, количество вхождений которого нужно посчитать.

Если значение в кортеже не найдено, то метод возвращает 0.

Следующий программный код:

```
names = ('Timur', 'Gvido', 'Roman', 'Timur', 'Anders', 'Timur')
cnt1 = names.count('Timur')
cnt2 = names.count('Gvido')
cnt3 = names.count('Josef')

print(cnt1)
print(cnt2)
print(cnt3)
```

выведет:

```
3
1
0
```

Кортежи не поддерживают такие методы, как append(), remove(), pop(), insert(), reverse(), sort(), так как эти методы изменяют содержимое.

Вложенные кортежи

Подобно спискам, мы можем создавать вложенные кортежи.

Следующий программный код:

```
colors = ('red', ('green', 'blue'), 'yellow')
numbers = (1, 2, (4, (6, 7, 8, 9)), 10, 11)

print(colors[1][1])
print(numbers[2][1][3])
```

выводит:

```
blue
9
```

Основы работы с кортежами. Часть 2

Перебор кортежей

Перебор элементов кортежа осуществляется точно так же как перебор элементов списка.

Для вывода **каждого** из элементов кортежа **на отдельной строке** можно использовать следующий код:

Вариант 1. Если нужны индексы элементов:

```
numbers = (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10)

for i in range(len(numbers)):
    print(numbers[i])
```

Вариант 2. Если индексы не нужны:

```
numbers = (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10)

for num in numbers:
    print(num)
```

Можно также использовать операцию **распаковки кортежа**.

Приведенный ниже код:

```
numbers = (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
languages = ('Python', 'C++', 'Java')

print(*numbers)
print(*languages, sep='\n')
```

выводит:

```
0 1 2 3 4 5 6 7 8 9 10
Python
C++
Java
```

Сравнение кортежей

Кортежи можно сравнивать между собой.

Приведенный ниже код:

```
print((1, 8) == (1, 8))
print((1, 8) != (1, 10))
print((1, 9) < (1, 2))
print((2, 5) < (6,))
print(('a', 'bc') > ('a', 'de'))
```

выводит:

```
True
True
False
True
False
```

Обратите внимание: операции == и != применимы к любым кортежам, независимо от типов элементов. А вот операции <, >, <=, >= применимы только в том случае, когда соответствующие элементы кортежей имеют один тип.

Приведенный ниже код:

```
print((7, 5) < ('java', 'python'))
```

выводит:

```
TypeError: '<' not supported between instances of 'int' and 'str'
```

Сравнение кортежей происходит последовательно элемент за элементом, а если элементы равны — просматривается следующий элемент.

Сортировка кортежей

Как мы помним, списки имеют метод sort(), который осуществляет сортировку на месте, то есть меняет порядок исходного списка. Поскольку кортежи по своей природе неизменяемы, то встроенного метода sort() они не содержат, тем не менее с помощью встроенной функции sorted() (не путать с списочным методом sort()) мы можем сортировать значения в кортежах.

Приведенный ниже код:

```
not_sorted_tuple = (34, 1, 8, 67, 5, 9, 0, 23)
print(not_sorted_tuple)

sorted_tuple = tuple(sorted(not_sorted_tuple))
print(sorted_tuple)
```

выводит:

```
(34, 1, 8, 67, 5, 9, 0, 23)
(0, 1, 5, 8, 9, 23, 34, 67)
```

Обратите внимание, что функция sorted() возвращает список, но с помощью функции tuple() мы приводим результат сортировки к кортежу.

Для сортировки кортежа можно воспользоваться явным преобразованием в список и использовать метод sort():

```
not_sorted_tuple = ('cc', 'aa', 'dd', 'bb')
tmp = list(not_sorted_tuple)
tmp.sort()

sorted_tuple = tuple(tmp)
print(sorted_tuple)
```

Преобразование кортежа в список и строку

Часто на практике нам приходится преобразовывать кортежи в списки и в строки. Для этого используются функции и методы str(), list(), tuple(), join().

Преобразование кортежа в список и наоборот

Кортеж можно преобразовать в список с помощью функции list().

Приведенный ниже код:

```
tuple1 = (1, 2, 3, 4, 5)
list1 = list(tuple1)
print(list1)
```

выводит:

```
[1, 2, 3, 4, 5]
```

Список можно преобразовать в кортеж с помощью функции tuple().

Приведенный ниже код:

```
list1 = [1, 17.8, 'Python']
tuple1 = tuple(list1)
print(tuple1)
```

выводит:

```
(1, 17.8, 'Python')
```

Преобразование кортежа в строку и наоборот

Кортеж можно преобразовать в строку с помощью строкового метода join().

Приведенный ниже код:

```
notes = ('Do', 'Re', 'Mi', 'Fa', 'Sol', 'La', 'Si')
string1 = ''.join(notes)
string2 = '.'.join(notes)

print(string1)
print(string2)
```

выводит:

```
DoReMiFaSolLaSi
Do.Re.Mi.Fa.Sol.La.Si
```

Обратите внимание, что для применения строкового метода join() кортеж должен содержать именно строковые элементы. Если элементы кортежа отличны от строк, то требуется предварительно их преобразовать.

Строку можно преобразовать в кортеж с помощью функции tuple().

Приведенный ниже код:

```
letters = 'abcdefghijkl'
tpl = tuple(letters)
print(tpl)
```

выводит:

```
('a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l')
```

Обратите внимание, что следующий код:

```
number = 12345
tpl = tuple(number)
print(tpl)
```

приведет к ошибке:

```
TypeError: 'int' object is not iterable
```

Это происходит, поскольку тип данных int не является итерируемым объектом. Для преобразования числа в кортеж сначала нужно число преобразовать в строку и уже только потом использовать функцию tuple().

Основые работы с кортежами. Часть 2

Упаковка кортежей

Упаковкой кортежа называют присваивание его какой-либо переменной.

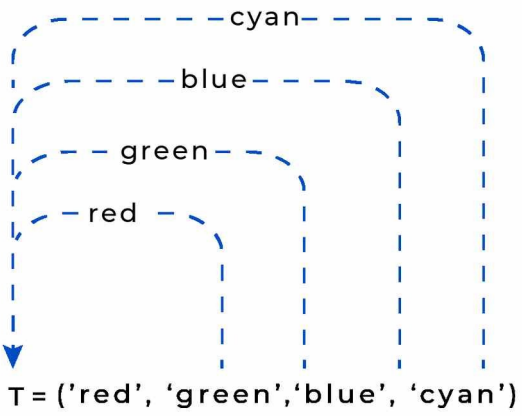
Приведенный ниже код:

```
tuple1 = (1, 2, 3)
tuple2 = ('b',)
tuple3 = ('red', 'green', 'blue', 'cyan')

print(type(tuple1))
print(type(tuple2))
print(type(tuple3))
```

выводит:

```
<class 'tuple'>
<class 'tuple'>
<class 'tuple'>
```



Обратите внимание на то, что упаковка выполняется всегда, когда справа от знака равенства стоит больше одного значения.

Приведенный ниже код автоматически запакует 1, 2, 3 и 'b', в кортежи (1, 2, 3) и ('b',) и присвоит их значения переменным tuple1 и tuple2:

```
tuple1 = 1, 2, 3
tuple2 = 'b',

print(type(tuple1))
print(type(tuple2))
```

выводит:

```
<class 'tuple'>
<class 'tuple'>
```

Распаковка кортежей

Обратная операция, смысл которой в том, чтобы присвоить значения элементов кортежа отдельным переменным, называется **распаковкой кортежа**.

Приведенный ниже код:

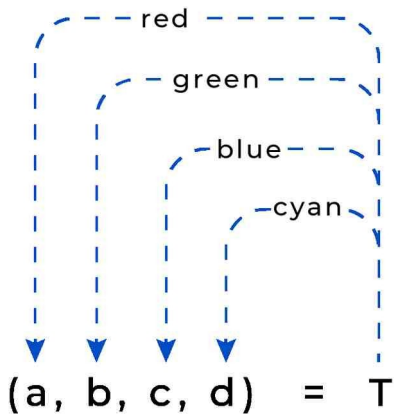
```
colors = ('red', 'green', 'blue', 'cyan')
(a, b, c, d) = colors

print(a)
print(b)
print(c)
print(d)
```

выводит:

```
red
green
blue
cyan
```

В приведенном примере кортеж colors распаковывается в переменные a, b, c, d.



Мы можем опустить скобки слева от знака равенства:

```
colors = ('red', 'green', 'blue', 'cyan')
a, b, c, d = colors
```

Количество переменных должно совпадать с числом элементов в кортеже.

Приведенный ниже код:

```
colors = ('red', 'green', 'blue', 'cyan')
a, b = colors
```

приводит к ошибке:

ValueError: too many values to unpack

Аналогично, приведенный ниже код:

```
colors = ('red', 'green', 'blue')
a, b, c, d = colors
```

приводит к ошибке:

ValueError: not enough values to unpack (expected 4, got 3)

Однако, если необходимо получить лишь какие-то отдельные значения, то в качестве "ненужных" переменных позволено использовать символ нижнего подчеркивания _.

Приведенный ниже код:

```
colors = ('red', 'green', 'blue')
a, b, _ = colors

print(a)
print(b)
```

выводит:

```
red
green
```

Распаковка кортежей очень удобна на практике. По сути мы использовали ее, когда меняли местами значения двух переменных без использования временных переменных.

Приведенный ниже код:

```
a = 7
b = 17
a, b = b, a

print(a, b)
```

выводит:

17 7

Сначала вычисляются все значения справа, и лишь затем они кладутся в левую часть оператора присваивания. Поэтому можно менять местами значения переменных a и b, написав: a, b = b, a.

Приведенный ниже код:

```
a, b, c = 3, 2, 1
b, a, c = c, a, b

print(b, c, a)
```

выводит:

```
1 2 3
```

* при распаковке кортежей

Как мы знаем, если при распаковке кортежа число элементов слева и справа не совпадает, то возникает ошибка времени исполнения. Есть способ собрать сразу **несколько значений в одну переменную**. Это делается при помощи звездочки перед именем переменной.

Рассмотрим программный код:

```
a, b, *tail = 1, 2, 3, 4, 5, 6
```

В этом случае в переменной a будет записана единица, в переменной b — двойка, а в переменной tail — список, состоящий из всех аргументов, которые не попали в предыдущие переменные. В данном случае tail будет равен [3, 4, 5, 6].

Учтите, что tail всегда будет списком, даже когда в него попадает лишь один элемент или даже ноль.

Приведенный ниже код:

```
a, b, *tail = 1, 2, 3

print(tail)
```

выводит:

```
[3]
```

Приведенный ниже код:

```
a, b, *tail = 1, 2
print(tail)
```

выводит:

```
[]
```

Звездочка может быть только у одного аргумента, но необязательно у последнего.

Приведенный ниже код:

```
*names, surname = ('Стефани', 'Джоанн', 'Анджелина', 'Джерманотта')

print(names)
print(surname)
```

выводит:

```
['Стефани', 'Джоанн', 'Анджелина']
Джерманотта
```

Аргумент со звездочкой может стоять и посередине.

```
singer = ('Freddie', 'Bohemian Rhapsody', 'Killer Queen', 'Love of my life', 'Mercury')

name, *songs, surname = singer

print(name)
print(songs)
print(surname)
```

выводит:

```
Freddie
['Bohemian Rhapsody', 'Killer Queen', 'Love of my life']
Mercury
```

Примечания

Примечание 1. Если вы хотите распаковать единственное значение в кортеже, после имени переменной должна идти запятая.

Приведенный ниже код:

```
a = 1,      # не распаковка, а просто присвоение
b, = 1,     # распаковка

print(a)
print(b)
```

выводит:

```
(1,)
1
```

Примечание 2. Распаковывать можно не только кортеж, правая сторона может быть любой последовательностью (кортеж, строка или список).

```
info = ['timur', 'beegeek.org']
user, domain = info     # распаковка списка

print(user)
print(domain)

a, b, c, d = 'math'     # распаковка строки

print(a)
print(b)
print(c)
print(d)
```

выводит:

```
timur
beegeek.org
m
a
```

```
t
h
```

Примечание 3. Помимо метода split() строковый тип данных содержит метод partition(). Метод partition() принимает на вход один аргумент sep, разделяет строку при первом появлении sep и **возвращает кортеж**, состоящий из трех элементов: часть перед разделителем, сам разделитель и часть после разделителя. Если разделитель не найден, то кортеж содержит саму строку, за которой следуют две пустые строки.

Приведенный ниже код:

```
s1 = 'abc-de'.partition('-')
s2 = 'abc-de'.partition('.')
s3 = 'abc-de-fgh'.partition('-')

print(s1)
print(s2)
print(s3)
```

выведет:

```
('abc', '-', 'de')
('abc-de', '', '')
('abc', '-', 'de-fgh')
```

Примечание 4. С использованием кортежей многие алгоритмы приобретают достаточно краткую форму. Например, вычисление чисел Фибоначчи может выглядеть следующим образом:

```
n = int(input())
f1, f2 = 1, 1
for i in range(n):
    print(f1)
    f1, f2 = f2, f1 + f2
```

Примечание 5. Замечательная серия статей о коллекциях (list, tuple, str, set, dict) в Python.