**Кортежи. Преобразование коллекций**

1. [Кортежи](https://lyceum.yandex.ru/courses/275/groups/2142/lessons/1682/materials/3261#1)
2. [Присваивание кортежей](https://lyceum.yandex.ru/courses/275/groups/2142/lessons/1682/materials/3261#2)
3. [Сортировка пузырьком](https://lyceum.yandex.ru/courses/275/groups/2142/lessons/1682/materials/3261#3)
4. [Преобразования между коллекциями](https://lyceum.yandex.ru/courses/275/groups/2142/lessons/1682/materials/3261#4)

**Аннотация**

*В уроке вводится еще один контейнер — кортеж (tuple). Более подробно рассматривается операция присваивания кортежей, знакомая нам по конструкции a, b = b, a, и применение этой операции в реализации классического алгоритма — сортировки пузырьком. Рассматриваются и вопросы преобразования одной коллекции в другую.*

**Кортежи**

Мы уже знаем такие коллекции, как списки, множества и строки. Сегодня мы рассмотрим еще один тип данных, являющийся коллекцией, который называется tuple (читается «тюпл» или «тьюпл», а переводится как «кортеж»).

**Кортежи**

Кортежи очень похожи на списки, они тоже являются индексированной коллекцией, только вместо квадратных в них используются круглые скобки (причем их часто можно пропускать):

# кортеж из двух элементов; тип элементов может быть любой

card = ('7', 'пик')

# пустой кортеж (из 0 элементов)

empty = ()

# кортеж из 1 элемента - запятая нужна, чтобы отличить от обычных скобок

t = (18,)

# длина, значение отдельного элемента, сложение - как у списков

print(len(card), card[0], card + t)

Кортежи можно сравнивать между собой:

(1, 2) == (1, 3) # False

(1, 2) < (1, 3) # True

(1, 2) < (5,) # True

('7', 'червей') < ('7', 'треф') # False

# А вот так сравнивать нельзя: элементы кортежей разных типов

(1, 2) < ('7', 'пик')

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. Поэтому сравнивать ('7', 'червей') и ('7', 'треф') можно, а вот кортежи (1, 2) и ('7', 'пик') нельзя — интерпретатор Python выдаст ошибку. При этом сравнение происходит последовательно элемент за элементом, а если элементы равны — просматривается следующий элемент.

**Неизменяемость**

Важнейшее техническое отличие кортежей от списков — неизменяемость. Как и к строке, к кортежу нельзя добавить элемент методом append, а существующий элемент нельзя изменить, обратившись к нему по индексу. Это выглядит недостатком, но в дальнейшем мы поймем, что у кортежей есть и преимущества.

Есть и семантическое, то есть смысловое, отличие. Если списки предназначены скорее для объединения неопределенного количества однородных сущностей, то кортеж — быстрый способ объединить под одним именем несколько разнородных объектов, имеющих различный смысл.

Так, в примере выше кортеж card состоит из двух элементов, означающих достоинство карты и ее масть.

Еще одним приятным отличием кортежей от списков является то, что они могут быть элементами множества:

a = {('7', 'червей'), ('7', 'треф')}

print(a) # -> {('7', 'треф'), ('7', 'червей')}

**Присваивание кортежей**

Кортежи можно присваивать друг другу. Именно благодаря этому работает красивая особенность Python — уже знакомая нам конструкция вида a, b = b, a.

Как известно, по левую сторону от знака присваивания = должно стоять имя переменной либо имя списка с индексом или несколькими индексами. Они указывают, куда можно «положить» значение, записанное справа от знака присваивания. Однако слева от знака присваивания можно записать еще и кортеж из таких обозначений (грубо говоря, имен переменных), а справа — кортеж из значений, которые следует в них поместить. Значения справа указываются в том же порядке, что и переменные слева (здесь скобки вокруг кортежа необязательны):

n, s = 10, 'hello'

# то же самое, что

n = 10

s = 'hello'

В примере выше мы изготовили кортеж, стоящий справа от =, прямо на этой же строчке. Но можно заготовить его и заранее:

cards = [('7', 'пик'), ('Д', 'треф'), ('Т', 'пик')]

value, suit = cards[0]

print('Достоинство карты:', value)

print('Масть карты:', suit)

Самое приятное: сначала вычисляются все значения справа, и лишь затем они кладутся в левую часть оператора присваивания. Поэтому можно, например, поменять местами значения переменных a и b, написав: a, b = b, a.

a, b = 1, 2 # теперь a == 1 and b == 2

a, b = b, a # теперь a == 2 and b == 1

Пример ниже выведет «1 2 3». Убедитесь, что вы понимаете, почему так.

# кручу-верчу

a, b, с = 3, 2, 1

b, a, c = c, a, b

print(b, c, a)

С использованием кортежей многие алгоритмы приобретают волшебную краткость. Например, вычисление чисел Фибоначчи:

n = int(input())

f1, f2 = 0, 1

for i in range(n):

print(f2)

f1, f2 = f2, f1 + f2

**Сортировка пузырьком**

Итак, у нас есть удобный способ поменять местами значения двух переменных. Теперь рассмотрим алгоритм, в котором эта операция играет важную роль.

Часто бывает нужно, чтобы данные не просто содержались в списке, а были отсортированы (например, по возрастанию), то есть чтобы каждый следующий элемент списка был не меньше предыдущего. В качестве данных могут выступать числа или строки. Скажем, отсортированный список [4, 1, 9, 3, 1] примет вид [1, 1, 3, 4, 9]. Конечно, для этого есть стандартные функции и методы, но как они работают?

Классический алгоритм сортировки — **сортировка пузырьком** (по-научному — сортировка обменом). Она называется так потому, что элементы последовательно «всплывают» (отправляются в конец списка), как пузырьки воздуха в воде. Сначала всплывает самый большой элемент, за ним — следующий по старшинству и т. д. Для этого мы сравниваем по очереди все соседние пары и при необходимости меняем элементы местами, ставя больший элемент на более старшее место.

Идею наглядно объясняет венгерский народный танец:

А полный код программы, которая считывает, сортирует и выводит список, выглядит, например, так:

n = int(input()) # количество элементов

a = []

for i in range(n): # считываем элементы списка

a.append(int(input()))

# Сортировка пузырьком:

for i in range(n - 1):

for j in range(n - 1 - i):

if a[j] > a[j + 1]:

a[j], a[j + 1] = a[j + 1], a[j]

print(a)

**Преобразования между коллекциями**

Итак, на данный момент мы знаем уже четыре вида коллекций: строки, списки, множества и кортежи. У вас может возникнуть вопрос: можно ли из одной коллекции сделать другую? Например, преобразовать строку в список или во множество? Конечно, да, для этого можно использовать функции list, set и tuple. Если в качестве аргумента передать этим функциям какую-либо коллекцию, новая коллекция будет создана на ее основе.

Зачем нужно преобразование коллекций?

Преобразование строки в список позволяет получить список символов. В некоторых задачах это может быть полезно: например, если мы хотим изменить один символ строки:

s = 'симпотичный' # Написали с ошибкой

a = list(s) # a == ['с', 'и', 'м', 'п', 'о', 'т', 'и,' 'ч', 'н', 'ы', 'й']

a[4] = 'а' # a == ['с', 'и', 'м', 'п', 'а', 'т', 'и,' 'ч', 'н', 'ы', 'й']

С этой же целью может потребоваться преобразование кортежа в список:

# В кортеже (писатель, дата рождения) допущена ошибка

writer = ('Лев Толстой', 1827)

a = list(writer) # a == ['Лев Толстой', 1827]

a[1] = 1828 # a == ['Лев Толстой', 1828]

Преобразование списка или строки во множество позволяет получить очень интересные результаты. Как вы помните, все элементы множества должны быть уникальны, поэтому при преобразовании списка во множество каждый элемент останется только в одном экземпляре. Таким образом, можно очень легко убрать повторяющиеся элементы и узнать, сколько элементов встречается в списке хотя бы один раз:

a = [1, 2, 1, 1, 2, 2, 3, 3]

print('Количество элементов в списке без повторений: ', len(set(a)))

Таким же образом можно получить все буквы без повторений, которые встречаются в строке:

a = set("Тетрагидропиранилциклопентилтетрагидропиридопиридиновые")

print(a, len(a))

{'л', 'н', 'в', 'о', 'и', 'ц', 'п', 'т', 'Т', 'г', 'ы', 'а', 'д', 'к', 'р', 'е'} 16

Преобразование множества в список тоже возможно, но при этом нужно учитывать, что элементы множества не упорядочены и при преобразовании множества в список порядок элементов в нем предсказать заранее не всегда возможно:

names = {'Иван', 'Петр', 'Сергей', 'Алексей'}

print(list(names))

# Возможные варианты вывода на экран - ['Сергей', 'Алексей', 'Иван', 'Петр'],

# ['Сергей', 'Петр', 'Иван', 'Алексей'], ['Алексей', 'Иван', 'Петр', 'Сергей']

# и так далее.