Кортежи. Преобразование коллекций

На занятии 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 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('Достоинствокарты:', 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, c =3,2,1
b, a, c = c, a, b
print(b, c, a)
```

С использованием кортежей многие алгоритмы приобретают волшебную краткость. Например, вычислениечиселФибоначчи:

```
n =int(input())
f1, f2 =0,1
```

```
foriinrange(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 inrange(n):# считываем элементы списка
a.append(int(input()))
# Сортировка пузырьком:
foriinrange(n -1):
for j inrange(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 =('ЛевТолстой',1827)
a =list(writer)# a == ['ЛевТолстой', 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))
```

```
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.
```

Пример 1. Произведение

Ограничение времени	1 секунда
Ограничение памяти	64Mb
Ввод	стандартный ввод или input.txt
Вывод	стандартный вывод или output.txt

Напишите программу, которая определяет, является ли данное число произведением двух чисел из данного набора, и выводит «ДА» или «НЕТ» в зависимости от этого. Если число в наборе такое одно, само на себя умножиться оно не может, т. е. два множителя должны иметь разные номера в наборе.

Формат ввода

На первой строке вводится количество чисел в наборе. Далее идут целые числа, составляющие набор (могут повторяться). Затем следует целое число, которое является или не является произведением двух каких-то чисел из набора.

Формат вывода

ДА или НЕТ

Пример

Ввод	Вывод
4 37 3 99 55	ДА

Ввод Вывод

111

```
x = int(input())
k = []
v = False
for i in range(x):
    k.append(int(input()))
z = int(input())
for i in range(x - 1):
    for j in range(i + 1, x):
        if k[i] * k[j] == z:
            v = True
            break
if v:
    print("ДА")
else:
    print("НЕТ")
```

Пример 2. Отбор

Ограничение времени	1 секунда
Ограничение памяти	64Mb
Ввод	стандартный ввод или input.txt
Вывод	стандартный вывод или output.txt

Имеется страница классного журнала с оценками за контрольную работу по информатике. Чтобы учителю удобнее было выбрать тех школьников, которым она предложит отправиться на олимпиаду по программированию, отобразите только отличников и хорошистов.

Формат ввода

Вводится натуральное число N — количество школьников. Далее следует N строк, каждая строка состоит из фамилии школьника, символа табуляции и его оценки (натуральное число от 1 до 5).

Формат вывода

Сначала выводятся все введённые строки с фамилиями и оценками учеников в том же порядке. Затем следует пустая строка. Затем выводятся только те строки из введённых, где указана оценка 4 или 5 (в том же порядке).

Пример

Ввод	Вывод
5 Кузнецов 5 Круглов 4 Федин 4 Тарасов 2 Словецкий 3	Кузнецов 5 Круглов 4 Федин 4 Тарасов 2 Словецкий 3 Кузнецов 5 Круглов 4 Федин 4

```python
x = int(input())
k = []
k1 = []
v = 0
z = ''
for i in range(x):
    z = input()
    z1 = z[-1]
    k.append(z)
    print(k[i])
    if (z1 == "4") or (z1 == "5"):
        k1.append(z)
print()
for i in range(len(k1)):
    print(k1[i])
```

Пример 3. Сортировка по длине

Ограничение времени	1 секунда
Ограничение памяти	64Mb
Ввод	стандартный ввод или input.txt
Вывод	стандартный вывод или output.txt

Отсортируйте введённые строки по возрастанию их длины, строки одинаковой длины при этом должны быть отсортированы по возрастанию в алфавитном порядке.

Формат ввода

На первой строке вводится натуральное число N — количество строк. Далее следуют N строк, которые надо будет отсортировать.

Формат вывода

Выводятся те же N строк, но не в том порядке, в котором они вводились, а в порядке возрастания их длины. Строки одинаковой длины должны быть отсортированы в алфавитном порядке по возрастанию.

Пример

Ввод	Вывод
4 три четыре пять шесть	три пять шесть четыре

```python
x = int(input())
s = []
for i in range(x):
    s.append((input()))
for i in range(x):
    for j in range(i + 1, x):
        if len(s[i]) > len(s[j]):
            s[i], s[j] = s[j], s[i]
        elif len(s[i]) == len(s[j]):
            if s[i] > s[j]:
                s[i], s[j] = s[j], s[i]
for i in s:
    print(i)
```

Пример 4. Сортировка по алфавиту

Ограничение времени	1 секунда
Ограничение памяти	64Mb

Ввод	стандартный ввод или input.txt
Вывод	стандартный вывод или output.txt

Отсортируйте введённые строки по возрастанию в алфавитном порядке. В этой задаче (и других задачах к этому занятию) под алфавитным порядком подразумевается, что каждая следующая строка должна быть больше предыдущей с точки зрения оператора сравнения «>» в Питоне; в частности, все строчные буквы идут после всех заглавных.

Формат ввода

На первой строке вводится натуральное число N — количество строк. Далее следуют N строк, которые надо будет отсортировать.

Формат вывода

Выводятся те же N строк, но не в том порядке, в котором они вводились, а в алфавитном порядке по возрастанию.

Пример

Ввод	Вывод
4 три четыре пять шесть	пять три четыре шесть

```
x = int(input())
s = []
for i in range(x):
    s.append((input()))
for i in range(x):
    for j in range(i + 1, x):
        if s[i] > s[j]:
            s[i], s[j] = s[j], s[i]
for i in s:
    print(i)
```

Пример 5. Напёрстки

Ограничение времени	1 секунда
Ограничение памяти	64Mb
Ввод	стандартный ввод или input.txt
Вывод	стандартный вывод или output.txt

Напёрсточник кладёт под каждый из напёрстков какую-нибудь мелочь, несколько раз переставляет напёрстки на столе, при этом некоторые напёрстки он убирает со стола. Определите, что под напёрстками, оставшимися в итоге на столе.

Формат ввода

На первой строке вводится натуральное число n_0 — изначальное количество напёрстков.

Далее следуют n текстовых строк, описывающих, что положено под напёрстки с 1-го по n_0-й.

На следующей строке вводится натуральное число k — количество перестановок напёрстков.

Далее следуют k групп строк, описывающих перестановки. Каждая группа устроена следующим образом. Пусть после предыдущей перестановки на столе осталось n_i напёрстков в определённом порядке. Пронумеруем их с 1-го по n_i-й (эта нумерация может не совпадать с исходной). Сначала на отдельной строке указывается количество напёрстков, которое останется на столе после данной перестановки — n_{i+1} (гарантируется, что $n_{i+1} \leq n_i$). Затем следует n_{i+1} строк, содержащих различные

номера напёрстков от 1 до n_i; эти строки показывают, какие напёрстки и в каком порядке окажутся на столе после перестановки.

Например, в приведённом примере производится две перестановки: после первой на столе остаются все три исходных напёрстка, но в порядке 3, 2, 1, то есть: жук, монета, стеклянный шарик; после второй перестановки на столе остаются только два напёрстка, первый и второй.

Формат вывода

Выводится список предметов под напёрстками, оставшимися на столе, в том порядке, в каком лежат напёрстки.

Пример

Ввод	Вывод
3 стеклянный шарик монета жук 2 3 3 2 1 2 1 2	жук монета

```
x = int(input())
s = [''] * x
for i in range(x):
    s[i] = input()
y = int(input())
for i in range(y):
    z = int(input())
    s1 = [''] * z
    for k in range(z):
        s1[k] = s[int(input()) - 1]
    s = s1
for i in s1:
    print(i)

```

Пример 6. Сортировка в обратном порядке

Ограничение времени	1 секунда
Ограничение памяти	64Mb
Ввод	стандартный ввод или input.txt
Вывод	стандартный вывод или output.txt

Отсортируйте введённые числа по убыванию.

Формат ввода

На первой строке вводится натуральное число N — количество чисел. Далее следуют N целых чисел, которые надо будет отсортировать, каждое на отдельной строке.

Формат вывода

Выводятся те же N чисел, но не в том порядке, в котором они вводились, а в порядке убывания.

Пример

Ввод	Вывод
4 34 243 43 292	292 243 43 34

```
x = int(input())
s = []
for i in range(x):
    s.append((int(input())))
for i in range(x):
    for j in range(i + 1, x):
        if s[i] < s[j]:
            s[i], s[j] = s[j], s[i]
for i in s:
    print(i)

```

Кортеж в Питоне: свойства и особенности

1. Кортежи не поддерживают добавление и удаление элементов, но допускают расширение и дополнение для тех элементов, которые относятся к **изменяемым** типам (списки, словари), а также любые операции с элементами элементов:

```
>>> numbers = ([1, 2, 3, 4], [5, 4, 5])
>>>numbers[1].extend([3, 5, 9])
>>>numbers[0].append([7, 7, 8])
>>>numbers[1][1] += 5
>>>print(numbers)
([1, 2, 3, 4, [7, 7, 8]], [5, 9, 5, 3, 5, 9])
```

2. Кортежи поддерживают **неограниченный** уровень вложенности:

```
numbers = ((4, 5, 8, (5, 1, (5, 3))), (7, 2, (4, 5, (3, 1, 1))))
```

3. Кортежи работают немного быстрее, чем списки – это связано с особенностями хранения tuple в памяти:

```
>>> from timeit import timeit
>>> times = 1000000
>>> t1 = timeit("list(['груша', 'виноград','яблоко', 'банан', 'апельсин'])", number=times)
>>> t2 = timeit("tuple(('груша', 'виноград','яблоко', 'банан', 'апельсин'))", number=times)
>>> diff = "{:.0%}".format((t2 - t1)/t1)
>>>print(f'Время копирования списка {times} раз: {t1}')
Время копирования списка 1000000 раз: 0.5761922669999961
>>>print(f'Время копирования кортежа {times} раз: {t2}')
Время копирования кортежа 1000000 раз: 0.22475764600000048
>>>print(f'Разница: {diff}')
Разница: -61%
```

4. Кортежи занимают меньше места в памяти:

```
>>> from sys import getsizeof
>>> numbers1 = ((1, 2, 3, 4), (5, 4, 5))
>>> numbers2 = [[1, 2, 3, 4], [5, 4, 5]]
>>> print(getsizeof(numbers1))
36
>>> print(getsizeof(numbers2))
44
```

5. Кортежи поддерживают конкатенацию + и повторение * n:

```
>>> num1 = (1, 2, 3)
>>> num2 = (4, 5, 6)
>>>print(num1 + num2)
(1, 2, 3, 4, 5, 6)
>>>print(num1 * 3)
(1, 2, 3, 1, 2, 3, 1, 2, 3)
```

6. Элементы tupleможно использовать в качестве значений переменных; этот прием называют распаковкой кортежа:

```
>>>seasons = ('весна', 'лето', 'осень', 'зима')
>>> s1, s2, s3, s4 = seasons
>>>print(s1, s3)
весна осень
```

Создание tuple в Питоне

Пустой кортеж можно создать двумя способами – с помощью круглых скобок () и с использованием функции tuple():

```
my_tuple = ()
my_tuple2 = tuple()
```

При создании кортежа с **одним** элементом после этого элемента необходимо ставить **запятую**, иначе Python не определяет конструкцию как кортеж:

```
>>>my_tuple = (5)
>>> print(type(my_tuple))
<class 'int'>
>>>my_tuple = (5,)
>>> print(type(my_tuple))
<class 'tuple'>
```

Создать кортеж с несколькими элементами можно следующими способами.

Способ 1: Перечисление элементов

Как и в случае со списками и словарями, кортеж с данными можно создать вручную. Кортеж может содержать данные различных типов:

```
info = ('Егор', 'разработчик', 350000, 28)
```

Элементы кортежа **необязательно** перечислять в круглых скобках – когда Python получает более одного значения для переменной, создание ("упаковка") кортежа происходит автоматически:

```
>>> letters = 'a', 'b', 'c', 'd'
>>> print(letters)
('a', 'b', 'c', 'd')
```

Способ 2: Преобразование других типов данных

С помощью встроенной функции tuple() можно создать кортеж из списка, строки или множества:

```
>>>my_lst = [4, 6, 2, 8]
>>> print(tuple(my_lst))
(4, 6, 2, 8)
>>>my_str = 'Яизучаю Python'
>>> print(tuple(my_str))
('Я', ' ', 'и', 'з', 'у', 'ч', 'а', 'ю', ' ', 'P', 'y', 't', 'h', 'o', 'n')
>>>my_set = {2, 5, 6, 7}
>>> print(tuple(my_set))
(2, 5, 6, 7)
```

Однако при подобном преобразовании словаря в кортеж останутся только ключи:

```
>>>my_dict = {'яблоки': 52, 'апельсины': 35}
>>> print(tuple(my_dict))
('яблоки', 'апельсины')
```

Число напрямую преобразовать в кортеж нельзя:

```
>>>num = 178394
>>> print(tuple(num))
Traceback (most recent call last):
  File "<pyshell>", line 1, in <module>
TypeError: 'int' object is not iterable
```

Но если конвертировать число в строку, преобразование пройдет без ошибок:

```
>>>num = 31232534
>>> print(tuple(str(num)))
('3', '1', '2', '3', '2', '5', '3', '4')
```

В ходе преобразования строки в кортеж с помощью одной лишь функции tuple() строка разбивается на отдельные символы. Если нужно разбить строку по **одному** разделителю, подойдет метод partition():

```
>>>st = 'Яизучаю***Python'
>>> print(st.partition('***'))
('Я изучаю', '***', 'Python')
```

Чтобы разбить строку по всем разделителям, вместе с tuple() используют split():

```
>>>st = tuple(input().split())
```

Коднаязыке Python
>>> print(st)
('Код', 'на', 'языке', 'Python')

Способ 3: Генератор кортежей

Генераторы кортежей, в отличие от списков и словарей, не слишком удобно использовать для решения практических задач:

>>>my_tuple = (i for i in range(5))
>>> print(my_tuple)
<generator object <genexpr> at 0x023B5830>

Но если генератор кортежа все-таки необходимо использовать, это можно сделать двумя способами:

1. Распаковать сгенерированный объект при выводе:

>>> numbers = (i for i in range(10))
>>>print(*numbers)
0 1 2 3 4 5 6 7 8 9

2. Либо использовать в генераторе функцию tuple():

>>>my_tuple = tuple(i ** 2 for i in range(1, 12, 2))
>>>print(my_tuple)
(1, 9, 25, 49, 81, 121)

Способ 4: Распаковка строк и списков

Оператор распаковки * и запятая после имени переменной автоматически превращают строки и списки в кортежи:

>>>st = 'Яизучаю Python'
>>>sp = ['Python', 'HTML5', 'CSS', 'JavaScript']
>>> tuple1 = (*st,)
>>> tuple2 = (*sp,)
>>> print(tuple1)
('Я', ' ', 'и', 'з', 'у', 'ч', 'а', 'ю', ' ', 'P', 'y', 't', 'h', 'o', 'n')
>>> print(tuple2)
('Python', 'HTML5', 'CSS', 'JavaScript')

Методы кортежей Python

Кортежи поддерживают большинство методов списков, за исключением удаления элементов и присваивания им новых значений:

>>>my_tuple = (4, 5, 6)
>>> del my_tuple[1]
Traceback (most recent call last):
  File "<pyshell>", line 1, in <module>
TypeError: 'tuple' object doesn't support item deletion
>>>my_tuple[2] = 9
Traceback (most recent call last):
  File "<pyshell>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment

Индексация и срезы

Индексация и срезы в кортежах работают так же, как и в списках:

```
>>>my_tuple = ('банан', 'груша', 'манго')
>>> print(my_tuple[0], my_tuple[-1])
бананманго
>>> print(my_tuple[1:])
('груша', 'манго')
>>> print(my_tuple[::-1])
('манго', 'груша', 'банан')
```

Для возвращения индекса элемента используется index():

```
>>>nutr = ('белки', 'жиры', 'углеводы')
>>>print(nutr.index('белки'))
0
```

Длина, сумма, минимальный и максимальный элементы

Эти методы тоже аналогичны списочным:

```
>>>my_tuple = ('карандаш', 'ручка', 'шар')
>>> print(len(my_tuple))
3
>>>print(min(my_tuple, key=len))
шар
>>>print(max(my_tuple, key=len))
карандаш
>>>nums = (4, 5, 2, 1, 6)
>>> print(sum(nums))
18
```

Принадлежность

С помощью операторовin и notin можно убедиться в наличии (отсутствии) определенного значения в кортеже:

```
>>>nums = (1, 3, 5, 6, 7, 8)
>>> (5 in nums)
True
>>> (25 not in nums)
True
```

Подсчет элементов

Для подсчета числа вхождений какого-либо значения в кортеже используется count():

```
>>>cities = ('Москва', 'Ростов', 'Москва', 'Рязань', 'Саратов', 'Москва')
>>>print(cities.count('Москва'))
3
```

Сортировка

Кортежи поддерживают сортировку, однако результатом будет список:

```
>>>nums = (4, 1, 7, 2, 0, 9, 5)
>>>print(sorted(nums))
[0, 1, 2, 4, 5, 7, 9]
```

Если результат должен сохраняться в виде кортежа, нужно использовать tuple():

```
>>>nums = (4, 1, 7, 2, 0, 9, 5)
>>>print(tuple(sorted(nums, reverse=True)))
(9, 7, 5, 4, 2, 1, 0)
```

Преобразование кортежей в другие типы данных

Кортеж можно преобразовать в строку:

```
>>> letters = ('P', 'y', 't', 'h', 'o', 'n')
>>> print('*'.join(letters))
P*y*t*h*o*n
```

В список:

```
>>>sp = (2, 7, 5, 8, 1)
>>>print(list(sp))
[2, 7, 5, 8, 1]
```

В словарь (если кортеж вложенный и состоит из пар значений):

```
>>> info = (('фрукты', 5), ('овощи', 15), ('конфеты', 3))
>>> print(dict(info))
{'фрукты': 5, 'овощи': 15, 'конфеты': 3}
```

Во множество:

```
>>> numbers = (3, 2, 1, 6, 7, 2, 2, 9)
>>> print(set(numbers))
{1, 2, 3, 6, 7, 9}
```

Сравнение кортежей

Как и списки, кортежи с однородными данными можно сравнивать между собой с помощью операторов >, >=, <, <=, ==, !=. Если элементы кортежей принадлежат к разным типам данных, поддерживаются только операторы == и !=.

```
>>> tuple1 = (1, 2, 3)
>>> tuple2 = (4, 5, 6)
>>>print(tuple1 < tuple2)
True
>>>print(tuple1 != tuple2)
True
```

Подведем итоги

Кортежи во многом похожи на списки. Выбор в пользу кортежей стоит делать тогда, когда необходимо защитить данные от случайного изменения. Кроме того, многие операции с кортежами выполняются быстрее и занимают меньше памяти. В следующей статье будем изучать множества.

Задания

Задание 1. Напишите программу, которая:

- Создает кортежи из положительных и отрицательных **целых** чисел на основе полученной от пользователя строки.
- Выводит количество положительных и отрицательных чисел в этих кортежах.

Пример ввода:

45 -6 -9 43 23 5 2 -9 -1 6 3

Вывод:

Кортеж (45, 43, 23, 5, 2, 6, 3) состоит из 7 положительных чисел
Кортеж (-6, -9, -9, -1) состоит из 4 положительных чисел

Задание 2. Напишите программу, которая:

- Создает кортеж из полученной от пользователя строки, состоящей из **вещественных** чисел, разделенных пробелами.
- Выводит минимальный и максимальный элементы кортежа, а также их сумму.

Пример ввода:

3.45 6.78 8.99 1.45 4.32 19.04 0.55

Вывод:

Минимальное число: 0.55
Максимальное число: 19.04
Сумма min и max: 19.59

Задание 3. Имеется кортеж списков, в которых перечислены названия фруктов и калорийность:

fruit = (['яблоки', 46], ['персики', 49], ['лимоны', 36], ['виноград', 190])

Калорийность винограда указана ошибочно. Напишите программу, которая исправит калорийность на 75, и добавит в третий элемент кортежа новое значение ['айва', 42].

Результат должен выглядеть так:

fruit = (['яблоки', 46], ['персики', 49], ['лимоны', 36, 'айва', 42], ['виноград', 75])

Задание4. Имеется вложенный кортеж:

numbers = ((5, 4, 5, 4), (3, 3, 4, 6), (8, 9, 5, 4), (12, 4, 5, 1), (9, 3, 5, 1))

Напишите программу, которая формирует новый кортеж, состоящий из средних арифметических значений элементов numbers.

Результат выводится в следующем виде:

4.5 4.0 6.5 5.5 4.5

Задание 5. Имеется вложенный кортеж:
nested_tuple = ((12, 3, 1), (5, 11), (15, 7, 8, 9), (10, 6, 4))

Напишите программу для преобразования **nested_tuple** в обычный кортеж, упорядоченный по возрастанию:
(1, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 15)

Задание 6. Напишите программу, которая на основе исходного кортежа создает новый кортеж, из которого исключены все пользователи с номерами телефонов с региональным кодом **+56**.

Исходный кортеж:

```
info = (('Евгений Романов', 25, '+56(983)354-67-21'),
        ('Марина Дятлова', 22, '+56(190)251-45-79'),
        ('Кирилл Кудрявцев', 34, '+7(890)456-12-42'),
        ('Сергей Дятлов', 24, '+56(190)156-42-99'),
        ('Юлия Степанова', 21, '+16(398)355-33-09'),
        ('Тимофей Иванов', 34, '+7(918)222-52-77'))
```

Ожидаемый результат:

(('Кирилл Кудрявцев', 34, '+7(890)456-12-42'), ('Юлия Степанова', 21, '+16(398)355-33-09'), ('Тимофей Иванов', 34, '+7(918)222-52-77'))

Задание 7. Имеется кортеж списков:
numbers = ([4, 5], [4, 5], [1, 6], [7, 3], [3, 3], [2, 4], [9, 5], [1, 1])

Напишите программу, которая добавит цифру 5 в конец каждого списка.

Ожидаемый результат:
numbers = ([4, 5, 5], [4, 5, 5], [1, 6, 5], [7, 3, 5], [3, 3, 5], [2, 4, 5], [9, 5, 5], [1, 1, 5])