**КОРТЕЖИ**

**Кортеж (tuple) – ещё один вид последовательностей в Python.**

По своей природе они очень схожи со списками, но, в отличие от последних, являются неизменяемыми.

*# кортеж*

immutable\_tuple = (4, 5, 6)

immutable\_tuple[0] = 404

>

Traceback (most recent call last):

immutable\_tuple[0] = 404

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

*# список*

mutable\_list = [7, 8, 9]

mutable\_list[0] = 1

print(mutable\_list)

\*

> [1, 8, 9]

Как видно, в литеральной форме кортеж python 3 записывается в виде последовательности элементов в круглых скобках, в то время как для списков характерны квадратные.

👉 Некоторые особенности кортежей:

* они упорядочены по позициям;
* tuple могут хранить и содержать внутри себя объекты любых типов (и даже составных);
* доступ к элементам происходит по смещению, а не по ключу;
* в рамках этой структуры данных определены все операции, основанные на применении смещения (индексирование, срез);
* кортежи поддерживают неограниченное количество уровней вложенности;
* кортежи хранят указатели на другие объекты, а значит их можно представлять, как массивы ссылок;
* они позволяют очень просто менять местами значения двух переменных.

Примеры кортежей

*# пустой кортеж*

empty\_tuple = ()

*# кортеж из 4-х элементов разных типов*

four\_el\_tuple = (36.6, 'Normal', None, False)

*# пример tuple, что содержит вложенные элементы*

nested\_elem\_tuple = (('one', 'two'), ['three', 'four'], {'five': 'six'}, (('seven', 'eight'), ('nine', 'ten')))

print(nested\_elem\_tuple)

> (('one', 'two'), ['three', 'four'], {'five': 'six'}, (('seven', 'eight'), ('nine', 'ten')))

Преимущества использования кортежей

Тем, кто уже успел познакомиться со списками в Python, может показаться не очевидным смысл использования кортежей. Ведь фактически, списки могут делать всё то же самое и даже больше. Это вполне естественный вопрос, но, разумеется, у создателей языка найдётся на него ответ:

* Неизменяемость – именно это свойство кортежей, порой, может выгодно отличать их от списков.
* Скорость – кортежи быстрее работают. По причине неизменяемости кортежи хранятся в памяти особым образом, поэтому операции с их элементами выполняются заведомо быстрее, чем с компонентами списка.
* Безопасность – неизменяемость также позволяет им быть идеальными кандидатами на роль констант. Константы, заданные кортежами, позволяют сделать код более читаемым и безопасным.
* Использование tuple в других структурах данных – кортежи применимы в отдельных структурах данных, от которых python требует неизменяемых значений. Например ключи словарей (dicts) должны состоять исключительно из данных immutable-типа.

💡 Кроме того, кортежи удобно использовать, когда необходимо вернуть из функции несколько значений:

**def** **get\_status**(service\_name):

**return** None, f"service {service\_name} is OK!"

print(type(get\_status('nginx')))

> <**class** '**tuple**'>

**error**, **message** = **get\_status**('nginx')

**print**(error)

**print**(message)

> **None**

> **service** **nginx** **is** **OK**!

Работа с кортежами

Создание

Как и другие коллекции языка Питон, кортеж можно создать двумя способами.

Способ №1: Литеральное объявление:

literal\_creation = ('any', 'object')

print(literal\_creation)

> ('any', 'object')

print(type(literal\_creation))

> <**class** '**tuple**'>

Способ №2: Через функцию tuple():

tuple\_creation = tuple('any iterable object')

print(tuple\_creation)

> ('a', 'n', 'y', ' ', 'i', 't', 'e', 'r', 'a', 'b', 'l', 'e', ' ', 'o', 'b', 'j', 'e', 'c', 't')

print(type(tuple\_creation))

> <**class** '**tuple**'>

💁‍♀️ Важно, чтобы аргумент, передаваемый в tuple() был итерируемым объектом:

incorrect\_creation = tuple(777)

>

Traceback (most recent call last):

incorrect\_creation = tuple(777)

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

Упаковка

Упаковкой кортежа называют присваивание его какой-то переменной, что, по сути, совпадает с операцией объявления.

Стоит обратить внимание 2 момента:

1. Выражения some\_tuple = (11, 12, 13) и some\_tuple = 11, 12, 13 тождественны.
2. Для объявления кортежа, включающего один единственный элемент, нужно использовать завершающую запятую:

is\_tuple = ('a',)

is\_tuple\_too = 'b',

not\_a\_tuple = 'c'

print(type(is\_tuple))

print(type(is\_tuple\_too))

print(type(not\_a\_tuple))

> <**class** '**tuple**'>

> <**class** '**tuple**'>

> <**class** '**str**'>

Распаковка

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

notes = ('Do', 'Re', 'Mi', 'Fa', 'Sol', 'La', 'Si')

do, re, mi, fa, sol, la, si = notes

print(mi)

> Mi

Количество переменных должно совпадать с числом элементов tuple

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

night\_sky = 'Moon', 'Stars'

moon, \_ = night\_sky

print(moon)

> Moon

Обращение к элементу и поиск в кортеже

Обратиться к элементу кортежа можно по номеру его позиции. Причём как с начала, так и с конца:

*# Mike - [0], Leo - [1], Don - [2], Raph - [3]*

turtles = ('Mike', 'Leo', 'Don', 'Raph')

*# Mike - [-4], Leo - [-3], Don - [-2], Raph - [-1]*

print(turtles[1])

print(turtles[-2])

print(turtles[2] == turtles[-2])

> Leo

> Don

> True

Если элемент кортежа есть вложенный кортеж, то применяются дополнительные квадратные скобки (в зависимости от уровня вложенности). Например, чтобы обратиться ко второму элементу второго элемента, следует поступить так:

input\_box = ('firstbox', (15, 150))

*# помним про индексацию, ведущую своё начало с 0*

print(input\_box[1][1])

> 150

Узнать, присутствует ли объект среди элементов кортежа, можно с помощью оператора in:

song = ('Roses', 'are', 'Red')

print('Red' **in** song)

print('Violet' **in** song)

> True

> False

Сравнение

tuple\_A = 2 \* 2,

tuple\_B = 2 \* 2 \* 2,

tuple\_C = 'a',

tuple\_D = 'z',

tuple\_E = (42, 'maximum')

tuple\_F = (42, 'minimum')

tuple\_Z = 999,

*# при сравнении кортежей, числа сравниваются по значению*

print(tuple\_A < tuple\_B)

> True

*# строки в лексикографическом порядке*

print(tuple\_C < tuple\_D)

> True

*# при равенстве элементов на одинаковых позициях, сравниваются элементы на следующих*

print(tuple\_E < tuple\_F)

> True

*# сравнение элементов продолжается до первого неравенства*

print(tuple\_Z < tuple\_F)

> False

Перебор

Наиболее простым и очевидным способом перебрать элементы кортежа является обход его в цикле for:

my\_tuple = ('Wise', 'men', 'say', 'only', 'fools', 'rush', 'in')

*# Вывести все элементы кортежа*

**for** word **in** my\_tuple:

print(word)

>

Wise

men

say

only

fools

rush

**in**

Сортировка

Нет ничего проще, чем отсортировать готовый кортеж. В этом используется функция sorted():

not\_sorted\_tuple = (10\*\*5, 10\*\*2, 10\*\*1, 10\*\*4, 10\*\*0, 10\*\*3)

print(not\_sorted\_tuple)

> (100000, 100, 10, 10000, 1, 1000)

sorted\_tuple = tuple(sorted(not\_sorted\_tuple))

print(sorted\_tuple)

> (1, 10, 100, 1000, 10000, 100000)

Удаление

Добавить или удалить элемент, содержащийся в tuple нельзя, в связи с неизменяемостью.

Оператор del позволяет удалить кортеж:

some\_useless\_stuff = ('sad', 'bad things', 'trans fats')

**del** some\_useless\_stuff

print(some\_useless\_stuff)

>

Traceback (most recent call last):

print(some\_useless\_stuff)

NameError: name 'some\_useless\_stuff' **is** **not** defined

Методы и особые операции

Срез

Слайсы кортежей похожи на срезы у строк, а выглядят они следующим образом:

tuple[start:fin:step]

Где start – начальный элемент среза (включительно), fin – конечный (не включительно) и step— "шаг" среза.

float\_tuple = (1.1, 0.5, 45.5, 33.33, 9.12, 3.14, 2.73)

print(float\_tuple[0:3])

> (1.1, 0.5, 45.5)

*# тождественная запись*

print(float\_tuple[:3])

> (1.1, 0.5, 45.5)

*# если не указывать конечное значение, будут выведены все элементы до конца*

print(float\_tuple[0:])

> (1.1, 0.5, 45.5, 33.33, 9.12, 3.14, 2.73)

*# не указывать начальное - с начала*

print(float\_tuple[:])

> (1.1, 0.5, 45.5, 33.33, 9.12, 3.14, 2.73)

*# выведем элементы с шагом 2*

print(float\_tuple[-7::2])

> (1.1, 45.5, 9.12, 2.73)

*# отрицательный шаг позволит вывести tuple в обратном порядке*

print(float\_tuple[::-1])

> (2.73, 3.14, 9.12, 33.33, 45.5, 0.5, 1.1)

Длина кортежа

Используя функцию len(), получаем длину/количество элементов:

php = ('p', 'h', 'p')

print(len(php))

> 3

Конкатенация

Для tuple определена операция конкатенации:

storm\_1 = ('Lightning')Union = (' and ')

storm\_2 = ('Thunder')

print(storm\_1 + Union + storm\_2)

> Lightning **and** Thunder

Повторение

Как и в случае с конкатенацией, для кортежей, впрочем, как и для строк, определена операция повторения:

dog\_do = ('woof!',)

print(dog\_do \* 3)

> ('woof!', 'woof!', 'woof!')

Индекс заданного элемента

Метод index() позволяет получить индекс элемента. Достаточно передать нужное значение элемента, как аргумент метода:

rom = ('I', 'II', 'III', 'IV', 'V', 'VI', 'VII', 'VIII', 'IX', 'X')

print(rom.index('X'))

> 9

Число вхождений элемента

Метод count() ведёт подсчет числа вхождений элемента в кортеж.

AT = ('Finn', 'Jake', 'BiMo', 'Marceline', 'Princess Bubblegum', 'BiMo')

print(AT.count('Finn'))

> 1

print(AT.count('BiMo'))

> 2

Преобразование

Tuple to Str

Способ преобразовать кортеж в строку при помощи метода join():

game\_name = ('Breath', ' ', 'of', ' ', 'the', ' ', 'Wild')

game\_name = ''.join(game\_name)

print(game\_name)

> Breath of the Wild

Tuple to List

Для конвертации шаблона в список необходимо передать кортеж, как аргумент функции list():

dig\_tuple = (1111, 2222, 3333)

print(dig\_tuple)

> (1111, 2222, 3333)

dig\_list = list(dig\_tuple)

print(dig\_list)

> [1111, 2222, 3333]

Tuple to Dict

А для преобразования кортежа в словарь нужно применить генератор словарей:

score = (('Eric', 65000), ('Stephany', 87000))

score\_dict = dict((x, y) **for** x, y **in** score)

print(score\_dict)

> {'Eric': 65000, 'Stephany': 87000}

Именованные кортежи

Мощная особенность языка.

Именованный кортеж (или named tuple) позволяет программисту обращаться к элементу кортежа не по индексу, а через удобочитаемый заранее заданный идентификатор.

Пример:

*# для начала импортируем сам модуль*

**from** collections **import** namedtuple

citizen = namedtuple("Citizen", "name age status")

Alex = citizen(name='Alex Mercer', age=27, status='show businessman')

print(Alex.name)

> Alex Mercer

print(Alex.status)

> show businessman

Точечная нотация при обращении к свойству объекта может вызвать невольную ассоциацию с классами. Одно из применений namedtuple связано с ситуациями, когда нужно передать несколько свойств объекта.

Tuple index out of range

Такая ошибка может возникнуть, например, при запуске следующего кода:

i\_initiate\_error = ('west', 'north', 'east', 'south')

print(i\_initiate\_error[4])

Причина её возникновения в том, что нельзя получить доступ к индексу кортежа за пределами его длины. Эта ошибка может возникать и при совсем нетривиальных обстоятельствах, однако суть её останется прежней. Начинающим программистам стоит помнить, что индексация кортежей начинается с 0, а не с 1.

Кортежи весьма похожи на списки, но очень сложно спорить с тем, что гибкость и функционал последних намного выше. Поэтому, при написании программ, следует, в основном, пользоваться ими, а не кортежами.

##### Свойства кортежей

Для кортежей выделяют следующие свойства:

1. Кортежи – это упорядоченные коллекции объектов произвольных типов. Коллекции объектов упорядочены слева направо.

2. В кортежах доступ к элементам обеспечивается по смещению. Кортежи поддерживают операции, которые используются по смещению элементов, например, вытягивание среза, индексирование и т.п.

3. Кортежи относятся к категории неизменяемых последовательностей. К кортежам невозможно применить операции непосредственного изменения. Однако, если элементом кортежа есть изменяемый элемент (например, список), то этот элемент можно изменять в кортеже.

4. Кортежи гетерогенны. Термин «гетерогенный» означает, что кортежи могут содержать другие составные объекты, например, списки, строки или другие кортежи.

5. Кортежи поддерживают произвольное количество вложений.

6. Кортежи имеют фиксированную длину. Если нужно изменить длину кортежа, то при этом обязательно создается копия кортежа в новом месте памяти, длина которой иная.

7. Кортежи можно представлять, как массивы ссылок на объекты.

##### Отличия между кортежами и списками

Между кортежами и списками можно выделить следующие отличия:

* кортеж формируется в круглых скобках **()**, список формируется в квадратных скобках **[]**;
* кортежи относятся к неизменяемым последовательностям, списки относятся к изменяемым последовательностям. К кортежам невозможно применить операции, которые непосредственно их изменяют. К спискам такие операции применять можно;
* кортежи имеют фиксированную длину (количество элементов), списки имеют переменную длину. Если нужно увеличить размер кортежа, то нужно создать копию.

##### Что есть общего между кортежами и списками?

Между кортежами и списками можно выделить следующие общие особенности:

* Списки и кортежи являются коллекциями объектов или последовательностями.
* Списки и кортежи обеспечивают приведение в порядок своих объектов слева направо.
* Кортежи и списки могут содержать элементы (объекты) любого типа.
* В кортежах и списках доступ к элементам осуществляется по смещению.
* Кортежи и списки поддерживают одинаковые операции, основанные на использовании смещения. Например, индексирование, получение среза и т.п.
* Кортежи и списки есть гетерогенными. Это значит, что кортежи и списки могут содержать другие составные элементы (строки, списки, кортежи).
* Кортежи и списки поддерживают произвольное количество вложений.
* Кортежи и списки позволяют сохранять массивы ссылок на другие сложные объекты, которые, в свою очередь также могут быть кортежами, списками или строками.

##### Способы (операции) создания кортежа

Создать кортеж можно одним из **4 способов** приведенных ниже.

**1.** С помощью пары пустых скобок ().

В этом случае создается пустой кортеж. Например

() # создается пустой кортеж

**2.** В случае одиночного кортежа с помощью запятой в конце элемента, взятого в круглые скобки.

Например.

(2,)

(True, )

**3.** С помощью использования нескольких элементов разделенных запятой и взятых в круглые скобки.

Например.

(2.5, -11, 3)

(a, True, "Tuples")

**4.** С помощью встроенной функции tuple().

Например.

# Создание вложенных кортежей с помощью операции =

a = (2.5, -11, 3)

b = (a, True, "Tuples") # b = ((2.5, -11, 3), True, 'Tuples')

c = tuple(b) # c = ((2.5, -11, 3), True, 'Tuples')

### 

##### 6. Пример создания кортежа, содержащего целые случайные числа

# Создать кортеж из последовательности 5 целых чисел,

# которые лежат в диапазоне от 0 до 10.

# Для использования случайных чисел нужно

# подключить модуль random

import random

# 1. Создать список из чисел

lst = [] # сначала пустой список

i = 0

while i < 5:

num = random.randint(0,10)

lst = lst + [num]

i = i+1

# 2. Создать кортеж из элементов списка

a = tuple(lst)

print("a = ", a) # a = (3, 0, 2, 6, 2)

Результат работы программы

a = (4, 1, 4, 5, 7)

##### 7. Пример поиска заданного элемента в кортеже

# Поиск заданного элемента в кортеже

# 1. Заданный кортеж из строк

a = ('ab', 'abcd', 'cde', 'abc', 'def')

# 2. Ввод искомой строки

s = str(input("s = "))

# 3. Цикл обхода кортежа

result = False # результат

i = 0

while i < len(a): # len(a) - количество элементов в кортеже

if (a[i]==s):

result = True

break

i = i+1

if (result):

print("Yes")

else:

print("No")

### 

##### 8. Функция len(). Количество элементов в кортеже

Для определения длины кортежа (количество элементов в кортеже) используется стандартная функция len().

Пример.

# Функция len() - количество элементов в кортеже

a = () # пустой кортеж

l = len(a) # l = 0

b = (1, 3, 7, 13, 'abc', True)

l = len(b) # l = 6

# Вложенный кортеж

c = (a, b, "Hello", 2.33)

l = len(c) # l = 4

### 

##### 9. Представление кортежей с одним элементом

Чтобы указать, что объект есть кортежем, нужно после элемента указать символ **‘ , ‘** (запятая). В противном случае объект будет восприниматься как число.

**Например.**

# Отличие между объектом, взятым в скобки () и кортежем.

# Число num1 имеющее значение 255

num1 = (255)

num2 = num1 + 5 # num2 = 255 + 5 = 260 - суммирование обычных целых чисел

# Кортеж содержащий число 255 - в конце указывается запятая

A = (255,)

# B = A + 5 - запрещено, исключительная ситуация

Примеры.

1. Напишите функцию tpl\_sort, которая сортирует кортеж, состоит из целых чисел, упорядоченных по возрастанию значений, и возвращает его. Если хотя бы один элемент не является целым числом, то функция возвращает исходный кортеж.

**def** **tpl\_sort**(tpl):

**for** element **in** tpl:

**if** **not** isinstance(element, int):

**return** tpl

**return** tuple(sorted(tpl))

*# Тесты*

print(tpl\_sort((5, 5, 3, 1, 9)))

print(tpl\_sort((5, 5, 2.1, '1', 9)))

Результат:

(1, 3, 5, 5, 9)

(5, 5, 2.1, '1', 9)

1. Функция slicer() на вход принимает кортеж и случайный элемент. Требуется вернуть новый кортеж, начинающийся с первого появления элемента в нем и заканчивающийся вторым его появлением включительно.

Если элемента нет вовсе – вернуть пустой кортеж.

Если элемент встречается только один раз, то вернуть кортеж, который начинается с него и идет до конца исходного.

**def** **slicer**(any\_tuple, element):

**if** element **in** any\_tuple:

**if** any\_tuple.count(element) > 1:

first\_index = any\_tuple.index(element)

second\_index = any\_tuple.index(element, first\_index + 1) + 1

**return** any\_tuple[first\_index:second\_index]

**else**:

**return** any\_tuple[any\_tuple.index(element):]

**else**:

**return** ()

**РЕШЕНИЕ 2**

BEGIN\_INDEX = 0

END\_INDEX = -1

**def** **slicer**(any\_tuple, element):

first = second = BEGIN\_INDEX

**if** element **in** any\_tuple:

first = any\_tuple.index(element)

**if** any\_tuple.count(element) > 1:

second = any\_tuple.index(element, first + 1) + 1

**else**:

second = END\_INDEX

**return** any\_tuple[first:second]

*# Тесты*

print(slicer((1, 2, 3), 8))

print(slicer((1, 8, 3, 4, 8, 8, 9, 2), 8))

print(slicer((1, 2, 8, 5, 1, 2, 9), 8))

Результат выполнения:

()

(8, 3, 4, 8)

(8, 5, 1, 2, 9)

1. Перед студентом стоит задача: на вход функции sieve() поступает список целых чисел. В результате выполнения этой функции будет получен кортеж уникальных элементов списка в обратном порядке.

**def** **sieve**(lst):

unique = []

[unique.append(item) **for** item **in** reversed(lst) **if** item **not** **in** unique]

**return** tuple(unique)

*# Тесты*

print(sieve([1, 2, 3, 3, 2]))

print(sieve([2, 1, 3, 1, 2, 5, 5, 9, 2, 0, 0]))

print(sieve((1, 2, 3, 4, 5, 6, 7)))

**Результат выполнения**

(2, 3, 1)

(0, 2, 9, 5, 1, 3)

(7, 6, 5, 4, 3, 2, 1)

1. Николай знает, что кортежи являются неизменяемыми, но он с этим не готов соглашаться. Ученик решил создать функцию del\_from\_tuple(), которая будет удалять первое появление определенного элемента из кортежа по значению и возвращать кортеж без оного. Попробуйте повторить шедевр не признающего авторитеты начинающего программиста. К слову, Николай не всегда уверен в наличии элемента в кортеже (в этом случае кортеж вернется функцией в исходном виде).

**def** **del\_from\_tuple**(tpl, elem):

**if** elem **in** tpl:

elem\_index = tpl.index(elem)

**return** tpl[:elem\_index] + tpl[elem\_index + 1:]

**return** tpl

*# Тесты*

print(del\_from\_tuple((1, 2, 3), 1))

print(del\_from\_tuple((1, 2, 3, 1, 2, 3, 4, 5, 2, 3, 4, 2, 4, 2), 3))

print(del\_from\_tuple((2, 4, 6, 6, 4, 2), 9))

**РЕШЕНИЕ 2**

**def** **del\_from\_tuple**(tpl, elem):

lst = list(tpl)

**if** elem **in** tpl:

lst.remove(elem)

**return** tuple(lst)

*# Тесты*

print(del\_from\_tuple((1, 2, 3), 1))

print(del\_from\_tuple((1, 2, 3, 1, 2, 3, 4, 5, 2, 3, 4, 2, 4, 2), 3))

print(del\_from\_tuple((2, 4, 6, 6, 4, 2), 9))

**Результат выполнения**

(2, 3)

(1, 2, 1, 2, 3, 4, 5, 2, 3, 4, 2, 4, 2)

(2, 4, 6, 6, 4, 2)

1. Создайте кортеж из 7-ми именованных кортежей учащихся ВУЗов. В именованном кортеже будут присутствовать следующие поля: имя студента, возраст, оценка за семестр, город проживания. Функция good\_students() будет принимать этот кортеж, вычислять среднюю оценку по всем учащимся и выводить на печать следующее сообщение: “Ученики {список имен студентов через запятую} в этом семестре хорошо учатся!”. В список студентов, которые выводятся по результатам работы функции, попадут лишь те, у которых оценка за семестр равна или выше средней по всем учащимся.

Сначала требуется создать объект **namedtuple**с 4-мя обозначенными полями. Потом придумать 7 случайных студентов, которые заносятся в кортеж.  
Далее определяем среднюю отметку за семестр и формируем список учащихся, у которых оценка выше средней.

**from** collections **import** namedtuple

Student = namedtuple('Student', 'name age mark city')

students = (

Student('Елена', '13', 7.1, 'Москва'),

Student('Ольга', '11', 7.9, 'Иваново'),

Student('Елизавета', '14', 9.1, 'Тверь'),

Student('Дмитрий', '12', 5.2, 'Челябинск'),

Student('Максим', '15', 6.1, 'Самара'),

Student('Николай', '11', 8.7, 'Владивосток'),

Student('Артур', '13', 5.8, 'Екатеринбург')

)

**def** **good\_students**(students):

total\_mark = 0

**for** student **in** students:

total\_mark += student.mark

avg\_mark = total\_mark / len(students)

good\_mark\_students = [student.name **for** student **in** students **if** student.mark >= avg\_mark]

print('Ученики ', ', '.join(good\_mark\_students), ' в этом семестре хорошо учатся!')

good\_students(students)

**Результат выполнения**

Ученики Ольга, Елизавета, Николай в этом семестре хорошо учатся!

#### Соревнования по игре «Тетрис-онлайн»

Соревнования по игре «Тетрис-онлайн» проводятся по следующим правилам.

Каждый участник регистрируется на сайте игры под определённым игровым именем. Имена участников не повторяются.  
Чемпионат проводится в течение определённого времени. В любой момент этого времени любой зарегистрированный участник может зайти на сайт чемпионата и начать зачётную игру. По окончании игры её результат (количество набранных очков) фиксируется и заносится в протокол.

Участники имеют право играть несколько раз. Количество попыток одного участника не ограничивается.  
Окончательный результат участника определяется по одной игре, лучшей для данного участника.  
Более высокое место в соревнованиях занимает участник, показавший лучший результат.  
При равенстве результатов более высокое место занимает участник, раньше показавший лучший результат.

В ходе соревнований заполняется протокол, каждая строка которого описывает одну игру и содержит результат участника и его игровое имя. Протокол формируется в реальном времени по ходу проведения чемпионата, поэтому строки в нём расположены в порядке проведения игр: чем раньше встречается строка в протоколе, тем раньше закончилась соответствующая этой строке игра.  
Напишите эффективную, в том числе по памяти, программу, которая по данным протокола определяет победителя и призёров. Гарантируется, что в чемпионате участвует не менее трёх игроков.  
Перед текстом программы кратко опишите алгоритм решения задачи и укажите используемый язык программирования и его версию.

**Описание входных данных**  
Первая строка содержит число N- общее количество строк протокола. Каждая из следующих N строк содержит записанные через пробел результат участника (целое неотрицательное число, не превышающее 100 миллионов) и игровое имя (имя не может содержать пробелов). Строки исходных данных соответствуют строкам протокола и расположены в том же порядке, что и в протоколе.  
Гарантируется, что количество участников соревнований не меньше 3.

**Описание выходных данных**  
Программа должна вывести имена и результаты трёх лучших игроков по форме, приведённой ниже в примере.

**Пример входных данных:**  
9  
69485 Jack   
95715 qwerty   
95715 Alex   
83647 M  
197128 qwerty   
95715 Jack   
93289 Alex   
95715 Alex   
95715 M

**Пример выходных данных для приведённого выше примера входных данных:**  
1 место. qwerty (197128)  
2 место. Alex (95715)  
3 место. Jack (95715)

#### Решение:

**score\_table = {}**

**N = int(input())**

**for time in range(N):**

**ball, name = input().split()**

**ball = int(ball)**

**if name in score\_table:**

**if ball > score\_table[name][0]:**

**score\_table[name][0] = ball**

**score\_table[name][1] = time**

**else:**

**score\_table[name] = [ball, time]**

**scores = list(score\_table.items())**

**def score\_key(a):**

**return a[1][0]\*100000000 - a[1][1]**

**scores.sort(key=score\_key, reverse = True)**

**for winner\_index in 0, 1, 2:**

**print(winner\_index + 1, 'место.', scores[winner\_index][0], end =' ')**

**print('(', scores[winner\_index][1][0], ')', sep='')**

#### Сдать багаж в камеру хранения

На вход программе подаются сведения о пассажирах, желающих сдать свой багаж в камеру хранения на заранее известное время до полуночи. В первой строке сообщается число пассажиров N, которое не меньше 3, но не превосходит 1000; во второй строке – количество ячеек в камере хранения K, которое не меньше 10, но не превосходит 1000. Каждая из следующих N  
строк имеет следующий формат:  
**<Фамилия> <время сдачи багажа> <время освобождения ячейки>,**  
где <Фамилия> – строка, состоящая не более чем из 20 непробельных символов; <время сдачи багажа> – через двоеточие два целых числа, соответствующие часам (от 00 до 23 – ровно 2 символа) и минутам (от 00 до 59 – ровно 2 символа); <время освобождения ячейки> имеет тот же формат. <Фамилия> и <время сдачи багажа>, а также <время сдачи багажа> и <время освобождения ячейки> разделены одним пробелом. Время освобождения больше времени сдачи.

Сведения отсортированы в порядке времени сдачи багажа. Каждому из пассажиров в камере хранения выделяется свободная ячейка с минимальным номером. Если в момент сдачи багажа свободных ячеек нет, то пассажир уходит, не дожидаясь освобождения одной из них.

Требуется написать программу (укажите используемую версию языка программирования, например Borland Pascal 7.0), которая будет выводить на экран для каждого пассажира номер ему предоставленной ячейки (можно сразу после ввода данных очередного пассажира). Если ячейка пассажиру не предоставлена, то его фамилия не печатается.

**Пример входных данных:**  
3  
10  
Иванов 09:45 12:00  
Петров 10:00 11:00  
Сидоров 12:00 13:12

**Результат работы программы на этих входных данных:**  
Иванов 1  
Петров 2  
Сидоров 1

#### Решение:

**N = int(input())**

**K = int(input())**

**time\_outs = [None for x in range(K)]**

**for i in range(N):**

**name, time\_in, time\_out = input().split()**

**for yachejka in range(K):**

**if time\_outs[yachejka] < time\_in or \**

**time\_outs[yachejka] == None:**

**print(name, yachejka)**

**time\_outs[yachejka] = time\_out**

**break**

#### 8. Распознавание чисел, записанных прописью

Вам необходимо написать программу распознавания чисел, записанных прописью. Сначала на вход программе подается обучающий блок, состоящий из 27 строк. Первые 9 строк содержат слова «один», «два», …, «девять», следующие 9 строк – слова «одиннадцать», «двенадцать», … «девятнадцать», следующие 9 строк – слова «десять», «двадцать», …, «девяносто». Все слова записаны маленькими русскими буквами без лишних пробелов в начале и в конце строки.

Затем на вход программе подается значение N – количество записей, которые необходимо обработать. Следующие N строк содержат записанные словами числа. Каждое число записано по-русски, маленькими буквами, без ошибок. Если число состоит из нескольких слов, между словами находится ровно один пробел, лишних пробелов в начале и в конце строк нет.

Напишите эффективную программу, которая определит сумму тех входных чисел, которые находятся в интервале от 1 до 99.  
Размер памяти, которую использует Ваша программа, не должен зависеть от длины исходного списка.  
Перед текстом программы кратко опишите используемый вами алгоритм решения задачи.

**Пример входных данных (обучающий блок показан в примере с сокращениями):**  
один  
два  
три  
четыре  
пять  
шесть  
семь  
восемь  
девять  
одиннадцать  
двенадцать  
тринадцать  
четырнадцать  
пятнадцать  
шестнадцать  
семнадцать  
восемнадцать  
девятнадцать  
десять  
двадцать  
тридцать  
сорок  
пятьдесят  
шестьдесят  
семьдесят  
восемьдесят  
девяносто   
5  
двадцать восемь  
два миллиона  
четырнадцать  
сто двадцать три  
тысяча девятьсот восемьдесят четыре

**Пример выходных данных для приведённого выше примера входных данных:**  
42

#### Решение:

**Number = {}**

**for num in range(1,10):**

**Number[input()] = num**

**for num in range(11,20):**

**Number[input()] = num**

**for num in range(10, 100, 10):**

**Number[input()] = num**

**N = int(input())**

**s = 0**

**for k in range(N):**

**x = 0**

**num = input().split()**

**for written\_num in num:**

**if written\_num in Number:**

**x += Number[written\_num]**

**else:**

**x = 0**

**break**

**s += x**

**print(s)**