Кортежи (tuple)

По аналогии со списками кортежи в Python — это стандартный тип, позволяющий хранить значения в виде последовательности.

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.

Как вы могли заметить, кортежи очень похожи на списки. По сути, они являются неизменяемыми списками. Это значит, что после создания кортежа хранимые в нем значения нельзя удалять или менять. Добавлять новые также нельзя

- Нельзя добавлять в них новые элементы. У этого типа нет методов append() или extend()
- Удалять элементы тоже нельзя, также из-за неизменяемости. Методов remove() и pop() здесь нет
- Искать элементы в кортеже можно, потому что этот процесс его не меняет
- Разрешено использовать оператор in для проверки наличия элемента в кортеже

Так что если вы планируете использовать постоянный набор значений для перебора, используйте кортеж вместо списка. Он будет работать быстрее. Плюс, это безопаснее, ведь такой тип данных защищен от записи.

```
import timeit

print(timeit.timeit("x = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)", number = 1000000))

print(timeit.timeit("x = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]", number = 1000000))
```

```
0.034131127635760095
0.11737610517116082
```

timeit Библиотека позволяет нам измерять прошедшее время в секундах. Мы можем ясно видеть, что инициализация кортежа выполняется быстрее, чем инициализация списка.

```
>>> lst = [10, 20, 30]
>>> tpl = (10, 20, 30)
>>> print(lst.__sizeof__())
32
>>> print(tpl.__sizeof__())
24
```

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
```

Перебор

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

```
my_tuple = ('Wise', 'men', 'say', 'only', 'fools', 'rush')
# Вывести все элементы кортежа
for word in my_tuple:
    print(word)
```

```
>
```

```
Wise

men

say

only

fools

rush
```

Сортировка

Нет ничего проще, чем отсортировать готовый кортеж. В этом наш друг и помощник — прекрасная 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, по причине всё той же неизменяемости. Однако сам кортеж стереть с цифрового лица Земли возможно. Оператор `del` к нашим услугам:

```
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
```

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

Представляем вашему вниманию лёгкий 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

Тут всё ещё проще. Для такой конвертации необходимо всего лишь 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]
```

Именованные кортежи

.

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

Покажем на примере:

```
# 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. В общем-то одно из применений namedtuple как раз связано с ситуациями, когда нужно передать несколько свойств объекта одним куском.

Если использовать много вложенных кортежей, то легко возникает путаница с индексами элементов и работа с такими кортежами теряет осмысленность:

```
>>> t = (('mod_1', 8.71, (-1.32, 23.87)), ('mod_2', 5.12, (-0.41, 19.86)))
```

С одной стороны, можно смириться с этим и пользоваться обычными целочисленными индексами:

```
>>> t[0][2][1]

23.87
```

С другой стороны, можно попытаться придать индексам какие-то осмысленные обозначения:

```
>>> model_1, model_2 = 0, 1

>>> name, mean, min_max = 0, 1, 2

>>> minimum, maximum = 0, 1
```

Теперь обращение к элементам `t` может выглядеть чуть логичнее:

```
>>> t[model_1][min_max][maximum]

23.87
```

Но самый простой способ это использование встроенного модуля `collections` из стандартной библиотеки. Благодаря ему "очень легко" создать структуру кортежа из примера выше:

```
>>> import collections

>>>

>>> models = collections.namedtuple('models', 'model_1 model_2')
```

```
>>> params = collections.namedtuple('params', 'name mean min_max')

>>> limit = collections.namedtuple('limit', 'minimum maximum')
```

Потом, точно также "очень легко" создать сам именованный кортеж:

```
>>> Models = models(params('mod_1', 8.71, limit(-1.32, 23.87)),...
params('mod_2', 5.12, limit(-0.41, 19.86)))
```

А вот извлекать элементы из такого кортежа, действительно легко:

```
>>> Models.model_1.min_max.maximum

23.87
```

Зачем же может понадобиться такая сложная штука? Дело в том, что иногда, вечером вы можете не понять код, который написали утром. Комментарии, строки документирования, и любые другие возможности, включая именованные кортежи призваны сделать ваш код легким для понимания. Да, постучать по клавиатуре, придется чуть дольше, но это лучше и не так обидно, чем начинать все сначала, после того как спустя неделю смысл кода окончательно забылся.