



КОРТЕЖИ В PYTHON

ПОНЯТИ

Е

Кортеж (tuple) — ещё один вид последовательностей в Python.

По своей природе они очень схожи со списками, но, в отличие от последних, являются неизменяемыми. Кортежи являются немутуирующими последовательностями. Это означает, что после того как кортеж создан, его невозможно изменить.

ПРИМЕР ЗАДАНИЯ

```
# 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]
```

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

Кортеж записывается в виде последовательности элементов в круглых скобках, в то время как для списков характерны квадратные.

Так как кортеж является неизменяемой (immutable) структурой, у нас не получится изменить его содержимое

```
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_25416\3461984885.py in <module>
      1 data = ["Иван", 37, "Яндекс"]
      2 t = tuple(data)
----> 3 t[0] = "Андрей"
```

TypeError: 'tuple' object does not support item assignment

Ошибка при попытке изменить содержимое кортежа

Однако, при необходимости, мы можем разложить кортеж на отдельные переменные для дальнейшей работы с ними [1]:

```
data = ["Иван", 37, "Яндекс"]
name, age, company = tuple(data)
print(name) # Иван
print(age) #37
print(company) # Яндекс
```

Применение цикла for для выполнения последовательного перебора элементов кортежа:

```
>>> names = ('Холли', 'Уоррен', 'Эшли')   
>>> for n in names:   
    print(n)    
Холли  
Уоррен  
Эшли  
>>>
```

Подобно спискам кортежи поддерживают индексацию, как показано в приведенном ниже сеансе:

```
>>> names = ('Холли', 'Уоррен', 'Эшли')   
>>> for i in range(len(names)):   
    print(names[i])    
Холли  
Уоррен  
Эшли  
>>>
```

Особенности кортежей:

Кортежи поддерживают все те же самые операции, что и списки, за исключением тех, которые изменяют содержимое списка. Кортежи поддерживают следующие операции:

- доступ к элементу по индексу (только для получения значений элементов);
- методы, в частности `index ()` ;
- встроенные функции, в частности `len`, `min` и `max`;
- выражения среза;
- оператор `in`;
- операторы `+` и `*`.

Кортежи не поддерживают такие методы, как `append ()`, `remove ()`, `insert ()`, `reverse ()` и `sort ()`.

ОСОБЕННОСТИ КОРТЕЖЕЙ:

- они упорядочены по позициям;
- кортежи могут хранить и содержать внутри себя объекты любых типов (и даже составных);
- в рамках элемента структуры данных определены все операции, применяемые к элементу (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.

ОСОБЕННОСТИ КОРТЕЖЕЙ:

Если необходимо создать кортеж всего с одним элементом, то после значения элемента следует написать замыкающую запятую:

```
my_tuple = (1,) # Создает кортеж всего с одним элементом.
```

Если запятая будет пропущена, то кортеж создан не будет. Например, приведенная инструкция просто присваивает переменной `value` целочисленное значение 1:

```
value = (1)      # Создает целочисленное значение.
```

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')))
```

ЗАЧЕМ ИСПОЛЬЗОВАТЬ КОРТЕЖ ВМЕСТО СПИСКА?

- ❑ **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.

РАБОТА С

КОРТЕЖАМИ

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))
```

Создание

tuple= ()

УПАКОВКА КОРТЕЖА

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

Стоит обратить внимание 2 момента:

- ❑ Выражения `some_tuple = (11, 12, 13)` и `some_tuple = 11, 12, 13` тождественны.
- ❑ Для объявления кортежа, включающего единственный элемент, нужно использовать завершающую запятую.

Упаковка

```
tuple = (■ ■,)
```

```
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 КОРТЕЖА

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

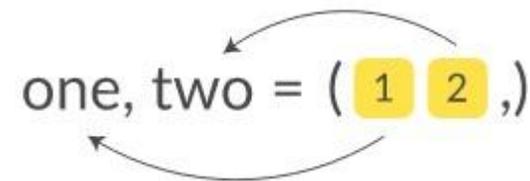
```
notes = ('Do', 'Re', 'Mi', 'Fa', 'Sol', 'La', 'Si')
```

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

```
print(mi)
```

```
> Mi
```

Распаковка



Количество переменных должно совпадать с числом элементов
tuple

Обращение к элементам

Обращение к элементам в кортеже происходит также, как и в списке, по индексу. Индексация начинается также с нуля при получении элементов с начала списка и с -1 при получении элементов с конца списка:

```
1 tom = ("Tom", 37, "Google", "software developer")
2 print(tom[0])      # Tom
3 print(tom[1])      # 37
4 print(tom[-1])     # software developer
```

Но так как кортеж - неизменяемый тип (immutable), то мы не сможем изменить его элементы. То есть следующая запись работать не будет:

```
1 tom[1] = "Tim"
```

При необходимости мы можем разложить кортеж на отдельные переменные:

```
1 name, age, company, position = ("Tom", 37, "Google", "software developer")
2 print(name)      # Tom
3 print(age)       # 37
4 print(position)  # software developer
5 print(company)   # Google
```

ОБРАЩЕНИЕ К ЭЛЕМЕНТУ И ПОИСК В КОРТЕЖЕ

tuple = ()

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
```

ОБРАЩЕНИЕ К ЭЛЕМЕНТУ И ПОИСК В КОРТЕЖЕ

tuple = (■ ■ ■ ■)
 ↑

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
```

ОБРАЩЕНИЕ К ЭЛЕМЕНТУ И ПОИСК В КОРТЕЖЕ

tuple = (■ ■ ■ ■)
↑

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
```

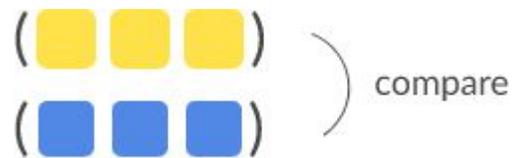
```
> False
```

Проверка наличия значения

Как для списка с помощью выражения элемент `in` кортеж можно проверить наличие элемента в кортеже:

```
1 user = ("Tom", 22, "Google")
2 name = "Tom"
3 if name in user:
4     print("Пользователя зовут Tom")
5 else:
6     print("Пользователь имеет другое имя")
```

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
```

Перебор кортежей

Для перебора кортежа можно использовать стандартные циклы **for** и **while**. С помощью цикла for:

```
1 tom = ("Tom", 22, "Google")
2 for item in tom:
3     print(item)
```

С помощью цикла while:

```
1 tom = ("Tom", 22, "Google")
2
3 i = 0
4 while i < len(tom):
5     print(tom[i])
6     i += 1
```

ПЕРЕБ

ОР

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

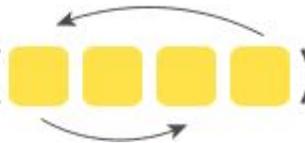
tuple = ()


```
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
```

СОРТИРОВ

tuple = ()

Нет ничего проще, чем отсортировать готовый кортеж, для этого 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)
```

УДАЛЕН

tuple = ( )

ИЕ

Добавить или удалить элемент содержащийся в **tuple** нельзя, по причине всё той же неизменяемости. Однако сам кортеж удалить возможно

```
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
```

Преобразование между списками и

Встроенная функция `list()` может применяться для преобразования кортежа в список, а встроенная функция `tuple()` — для преобразования списка в кортеж. Приведенный ниже интерактивный сеанс это демонстрирует:

```
1 >>> number_tuple = (1, 2, 3)   
2 >>> number_list = list(number_tuple)   
3 >>> print(number_list)   
4 [1, 2, 3]  
5 >>> str_list = ['один', 'два', 'три']   
6 >>> str_tuple = tuple(str_list)   
7 >>> print(str_tuple)   
8 ('один', 'два', 'три')  
9 >>>
```

Вот краткое описание инструкций.

- ◆ Строка 1 создает кортеж и присваивает его переменной `number_tuple`.
- ◆ Строка 2 передает `number_tuple` в функцию `list()`. Эта функция возвращает список, содержащий те же значения, что и в `number_tuple`, и присваивает его переменной `number_list`.
- ◆ Строка 3 передает список `number_list` в функцию `print`. Результат функции выводится в строке 4.
- ◆ Строка 5 создает список строковых значений и присваивает его переменной `str_list`.
- ◆ Строка 6 передает список `str_list` в функцию `tuple()`. Эта функция возвращает кортеж, содержащий те же значения, что и в `str_list`, и присваивает его переменной `str_tuple`.
- ◆ Строка 7 передает кортеж `str_tuple` в функцию `print`. Результат функции выводится в строке 8.

Преобразование между списками и кортежами

Для создания кортежа из другого набора элементов, например, из списка, можно передать список в функцию `tuple()`, которая возвратит кортеж:

```
1 data = ["Tom", 37, "Google"]
2 tom = tuple(data)
3 print(tom)      # ("Tom", 37, "Google")
```

С помощью встроенной функции `len()` можно получить длину кортежа:

```
1 tom = ("Tom", 37, "Google")
2 print(len(tom))  # 3
```

Преобразование между списками и кортежами

Пример:

```
>>> a = (10, 2.13, "square", 89, 'C')
>>> b = [1, 2, 3]
>>> c = list(a)
>>> d = tuple(b)
>>> c
[10, 2.13, 'square', 89, 'C']
>>> d
(1, 2, 3)
```

Получение подкортежей

Как и в списках, можно получить часть кортежа в виде другого кортежа

```
1 tom = ("Tom", 37, "Google", "software developer")
2
3 # получем подкортеж с 1 по 3 элемента (не включая)
4 print(tom[1:3])      # (37, "Google")
5
6 # получем подкортеж с 0 по 3 элемента (не включая)
7 print(tom[:3])      # ("Tom", 37, "Google")
8
9 # получем подкортеж с 1 по последний элемент
10 print(tom[1:])      # (37, "Google", "software developer")
```

Кортеж как параметр и результат функций

Особенно удобно использовать кортежи, когда необходимо вернуть из функции сразу несколько значений. Когда функция возвращает несколько значений, фактически она возвращает в кортеж:

```
1 def get_user():
2     name = "Tom"
3     age = 22
4     company = "Google"
5     return name, age, company
6
7
8 user = get_user()
9 print(user)      # ("Tom", 37, "Google")
```

При передаче кортежа в функцию с помощью оператора * его можно разложить на отдельные значения, которые передаются параметрам функции:

```
1 def print_person(name, age, company):
2     print(f"Name: {name} Age: {age} Company: {company}")
3
4 tom = ("Tom", 22)
5 print_person(*tom, "Microsoft")      # Name: Tom Age: 22 Company: Microsoft
6
7 bob = ("Bob", 41, "Apple")
8 print_person(*bob)                   # Name: Bob Age: 41 Company: Apple
```

Примеры использования создания кортежа и преобразования объектов к типу

```
# Создание кортежа тип tuple
>>> ()
# ()
>>> tuple()
# ()
>>> 105,
# (105,)
>>> 1, 'a', 3, 'b'
# (1, 'a', 3, 'b')

# Преобразование строки str в кортеж тип tuple
>>> tuple('abc')
# ('a', ' b', ' c')

# Преобразование списка list в кортеж тип tuple
>>> tuple([1, 2, 3])
# (1, 2, 3)

# Преобразование множества set в кортеж тип tuple
>>> tuple({1, 2, 3})
# (1, 2, 3)

# Преобразование генератора в кортеж тип tuple
>>> tuple(range(5))
# (0, 1, 2, 3, 4)
```