

Кортежи.

Кортеж (tuple) – это неизменяемая структура данных, которая по своему подобию очень похожа на список. Кортежи могут состоять из элементов разных типов, перечисленных через запятую. Кортежи заключаются в круглые скобки. Изменять его элементы нельзя (TypeError).

Преимущества кортежей:

1. Позволяют обезопасить данные от случайного изменения.
2. Экономия места - кортежи в памяти занимают меньший объем по сравнению со списками
3. Прирост производительности, который связан с тем, что кортежи работают быстрее, чем списки (т.е. на операции перебора элементов и т.п. будет тратиться меньше времени).
4. Можно использовать в качестве ключа у словаря

Для tuple определены операции конкатенации и повторения:

Раздаточный материал № 75

```
storm_1 = ('Lightning')
```

```
Union = (' and ')
```

```
storm_2 = ('Thunder')
```

```
print(storm_1 + Union + storm_2)
```

Результат: Lightning and Thunder

```
dog_do = ('woof!',)
```

```
print(dog_do * 3)
```

Результат: ('woof!', 'woof!', 'woof!')

Из кортежа можно извлекать элементы и брать срезы:

Раздаточный материал № 76

```
>>> a[3]
```

```
89
```

```
>>> a[1:3]
```

```
(2.13, 'square')
```

Создание пустого кортежа

Раздаточный материал № 77

```
>>> a = ()
```

```
>>> print(type(a))
```

```
<class 'tuple'>
```

Кортеж с заданным содержимым:

Раздаточный материал № 78

```
>>> a = (1, 2, 3, 4, 5)
```

```
>>> print(type(a))
```

```
<class 'tuple'>
```

```
>>> print(a)
```

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

```
>>> print(*a)
```

```
1 2 3 4 5
```

```
>>> a = tuple('hello, world!')
>>> a
('h', 'e', 'l', 'l', 'o', ',', ' ', 'w', 'o', 'r', 'l', 'd', '!')
```

Можно воспользоваться функцией tuple():

Раздаточный материал № 79

```
>>> a = tuple((1, 2, 3, 4))
>>> print(a)
(1, 2, 3, 4)
```

Доступ к элементам кортежа осуществляется через указание индекса.

Раздаточный материал № 80

```
>>> a = (1, 2, 3, 4, 5)
>>> print(a[0])
1
>>> print(a[1:3])
(2, 3)
```

Удалить отдельные элементы из кортежа невозможно. Но можно удалить кортеж целиком:

Раздаточный материал № 81

```
>>> del a
>>> print(a)
Traceback (most recent call last):
File "<pyshell#28>", line 1, in <module>
print(a)
NameError: name 'a' is not defined
```

На базе кортежа можно создать список, верно и обратное утверждение.

Раздаточный материал № 82

```
>>> lst = [1, 2, 3, 4, 5]
>>> print(type(lst))
<class 'list'>
>>> print(lst)
[1, 2, 3, 4, 5]
>>> tpl = tuple(lst)
>>> print(type(tpl))
<class 'tuple'>
>>> print(tpl)
(1, 2, 3, 4, 5)
Обратная операция также является корректной:
>>> tpl = (2, 4, 6, 8, 10)
>>> print(type(tpl))
<class 'tuple'>
>>> print(tpl)
(2, 4, 6, 8, 10)
>>> lst = list(tpl)
>>> print(type(lst))
<class 'list'>
>>> print(lst)
[2, 4, 6, 8, 10]
```

Кортежи могут содержать списки, также как списки могут быть вложенными в другие списки.

Раздаточный материал № 83

```
>>> nested = (1, "do", ["param", 10, 20])
```

Список внутри кортежа изменить можно

Раздаточный материал № 84

```
>>> nested[2][1] = 15
```

```
>>> nested
```

```
(1, 'do', ['param', 15, 20])
```

Выражения типа `nested[2][1]` используются для обращения к вложенным объектам. Первый индекс указывает на позицию вложенного объекта, второй – индекс элемента внутри вложенного объекта.

Для объектов кортежей определены два метода `index` и `count`.

Раздаточный материал № 84

```
>>> T = (1, 2, 3, 2, 4, 2) # Методы кортежей в Python 2.6, 3.0
```

```
# и последующих версиях
```

```
>>> T.index(2)
```

```
# Смещение первого появления элемента 2
```

```
1
```

```
>>> T.index(2, 2)
```

```
# Смещение появления элемента 2 после
```

смещения 2

```
3
```

```
>>> T.count(2)
```

```
# Сколько всего элементов 2?
```

```
3
```

Распаковка кортежа требуется для обработки значений кортежа и предполагает, что элементы кортежа будут присвоены отдельным переменным

Раздаточный материал № 85

```
tuplex = (4, 6, 2, 8, 3, 1)
```

```
a, b, *c = tuplex
```

Результат: 4 6 [2, 8, 3, 1, 9]

Переменные `a` и `b` содержат целочисленные переменные, в `c` помещается список. Исходный кортеж `tuplex` остается неизменным.

Словари.

Словари (тип dict) представляют собой изменяемый упорядоченный (начиная с версии 3.6) набор элементов с доступом к ним по ключу. Данные в словаре хранятся в формате "ключ:значение".

Т.к. словарь – это изменяемый тип данных, то он передается в функцию по ссылке.

Создание пустого словаря

Раздаточный материал № 86

```
>>> d1 = dict()
>>> print(type(d1))
<class 'dict'>
```

```
>>> d2 = {}
>>> print(type(d2))
<class 'dict'>
```

Создание словаря с заранее подготовленным набором данных

Раздаточный материал № 87

```
>>> d1 = dict(Ivan="менеджер", Mark="инженер")
>>> print(d1)
{'Mark': 'инженер', 'Ivan': 'менеджер'}
```

```
>>> d2 = {"A1": "123", "A2": "456"}
>>> print(d2)
{'A2': '456', 'A1': '123'}
```

```
>>> a = {'cat': 'кошка', 'dog': 'собака', 'bird': 'птица', 'mouse': 'мышь'}
```

Создание словаря через вложенный список

```
a = [['cat', 'кошка'], ['dog', 'собака'], ['bird', 'птица'], ['mouse', 'мышь']]
s = dict(a)
print(s)
{'cat': 'кошка', 'dog': 'собака', 'bird': 'птица', 'mouse': 'мышь'}
```

В словаре доступ к значениям осуществляется по ключам, которые заключаются в квадратные скобки

Раздаточный материал № 88

```
>>> a['cat']
'кошка'
```

```
>>> a['bird']
'птица'
```

Удаление и добавление элемента (пары "ключ:значение")

Раздаточный материал № 89

```
>>> a['elephant'] = 'бегемот'      # добавляем
>>> a['table'] = 'стол'           # добавляем
>>> a
{'dog': 'собака', 'cat': 'кошка', 'mouse': 'мышь', 'bird': 'птица', 'table': 'стол', 'elephant': 'бегемот'}
```

```
>>> a['elephant'] = 'слон'          # изменяем
>>> del a['table']                  # удаляем
>>> a
{'dog': 'собака', 'cat': 'кошка', 'mouse': 'мышь', 'bird': 'птица', 'elephant': 'слон'}
```

Правила организации словаря:

1. В словаре не может быть двух элементов с одинаковыми ключами. Однако могут быть одинаковые значения у разных ключей.

2. Ключом может быть любой неизменяемый тип данных. Значением – любой тип данных.

Проверка наличия ключа в словаре производится с помощью оператора `in`. Такую проверку желательно производить перед тем как удалить элемент из словаря, при его добавлении в словарь (м.б. он уже есть в словаре).

Раздаточный материал № 90

```
d2 = {"A1": "123", "A2": "456"}
"A1" in d2
True
"A3" in d2
False
```

Значения словарей вполне могут быть структурами, например, другими словарями или списками.

Раздаточный материал № 91

```
nums = {1: 'one', 2: 'two', 3: 'three'}
person = {'name': 'Tom', 1: [30, 15, 16], 2: 2.34, ('ab', 100): 'no'}
```

Элементы словаря перебираются в цикле `for` также, как элементы других сложных объектов.

Раздаточный материал № 92

```
# извлекаются ключи
for i in nums:
    print(i)
```

Результат

```
1
2
3
```

```
# извлекаются значения:
```

```
for i in nums:
    print(nums[i])
```

Результат

```
one
two
three
```

Методы словарей

Раздаточный материал № 93 (справочно)

clear() - удаляет все элементы словаря, но не удаляет сам словарь	<pre>>>> d2 = {"A1": "123", "A2": "456"} >>> print(d2) {'A2': '456', 'A1': '123'} >>> d2.clear() >>> print(d2) {}</pre>
copy() - создает новую копию словаря	<pre>>>> d2 = {"A1": "123", "A2": "456"} >>> d3 = d2.copy() >>> print(d3) {'A1': '123', 'A2': '456'} >>> d3["A1"] = "789" >>> print(d2) {'A2': '456', 'A1': '123'} >>> print(d3) {'A1': '789', 'A2': '456'}</pre>
fromkeys(seq[, value]) - создает новый словарь с ключами из seq и значениями из value. По умолчанию value присваивается значение None.	<pre>t = dict.fromkeys(['a', 'b', 'c'], 15) print(t) {'a': 15, 'b': 15, 'c': 15}</pre>
get(key) - возвращает значение из словаря по ключу key	<pre>>>> d = {"A1": "123", "A2": "456"} >>> d.get("A1") '123'</pre>
items() - возвращает (в виде кортежа) элементы словаря (ключ, значение) в отформатированном виде	<pre>>>> d = {"A1": "123", "A2": "456"} >>> d.items() dict_items([('A2', '456'), ('A1', '123')])</pre>
keys() - возвращает ключи словаря	<pre>>>> d = {"A1": "123", "A2": "456"} >>> d.keys() dict_keys(['A2', 'A1'])</pre>
pop(key[, default]) - если ключ key есть в словаре, то данный элемент удаляется из словаря и возвращается значение по этому ключу, иначе будет возвращено значение default. Если default не указан и запрашиваемый ключ отсутствует в словаре, то будет вызвано исключение KeyError	<pre>>>> d = {"A1": "123", "A2": "456"} >>> d.pop("A1") '123' >>> print(d) {'A2': '456'}</pre>
popitem() - удаляет и возвращает последнюю пару (ключ, значение) из словаря. Если словарь пуст, то будет вызвано исключение KeyError	<pre>>>> d = {"A1": "123", "A2": "456"} >>> d.popitem() ('A2', '456') >>> print(d) {'A1': '123'}</pre>
setdefault(key[, default]) - если ключ key есть в словаре, то возвращается значение по ключу. Если такого ключа нет, то в словарь вставляется элемент с ключом key и значением default, если default не определен, то по умолчанию присваивается None	<pre>>>> d = {"A1": "123", "A2": "456"} >>> d.setdefault("A3", "777") '777' >>> print(d) {'A2': '456', 'A3': '777', 'A1': '123'} >>> d.setdefault("A1")</pre>

	'123' >>> print(d) {'A2': '456', 'A3': '777', 'A1': '123'}
update([other]) - обновляет словарь парами (key/value) из other, если ключи уже существуют, то обновляет их значения	>>> d = {"A1": "123", "A2": "456"} >>> d.update({"A1": "333", "A3": "789"}) >>> print(d) {'A2': '456', 'A3': '789', 'A1': '333'}
values() - возвращает значения элементов словаря	>>> d = {"A1": "123", "A2": "456"} >>> d.values() dict_values(['456', '123'])

В цикле for можно распаковывать кортежи, таким образом сразу извлекая как ключ, так и его значение

Раздаточный материал № 94

```
for key, value in nums.items():
    print(key, 'is', value)
```

Результат

1 is one

2 is two

3 is three