ChainMap

ChainMap – класс, который является частью модуля collections, дающий возможность объединить несколько сопоставлений вместе таким образом, чтобы они стали единым целым.

ChainMap принимает любое количество словарей и превращает их в единое обновляемое представление.

Доступные методы и свойства:

- *maps* – возвращает обновляемый пользователем список сопоставлений.
- *new_child(m=None)* – возвращает новую ChainMap, дополненную элементом m.
- *parents* – свойство, возвращающее новую ChainMap, содержащую все элементы, кроме первого.

Изменения в версии 3.4:

Для *new_child* добавлен необязательный параметр m.

До версии 3.4:

```
>>> division1 = {'Иванов': 12, 'Петров': 15, 'Сидоров': 11}
>>> division3 = {'Зуев': 22, 'Абрамов': 19, 'Шарапов': 25}
>>> branch1 = ChainMap(division1, division3)
>>> branch1.new_child({'Зубков': 26})
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: new_child() takes 1 positional argument but 2 were given
```

С версии 3.4:

```
>>> division1 = {'Иванов': 12, 'Петров': 15, 'Сидоров': 11}
>>> division3 = {'Зуев': 22, 'Абрамов': 19, 'Шарапов': 25}
>>> branch1 = ChainMap(division1, division3)
>>> branch1.new_child({'Зубков': 26})
ChainMap({'Зубков': 26}, {'Иванов': 12, 'Петров': 15, 'Сидоров': 11}, {'Зуев': 22, 'Абрамов': 19,
'Шарапов': 25})
```

Изменения в версии 3.9:

Добавлена поддержка операторов | и | =, указанных в PEP 584.

До версии 3.9:

```
>>> division1 = {'Иванов': 12, 'Петров': 15, 'Сидоров': 11}
>>> division2 = {'Смирнов': 4, 'Дроздов': 5, 'Носов': 7}
>>> division3 = {'Зуев': 22, 'Абрамов': 19, 'Шарапов': 25}
>>> division4 = {'Павлов': 6, 'Чернов': 3, 'Ильин': 5}
>>> branch1 = ChainMap(division1, division3)
>>> branch2 = ChainMap(division2, division4)
>>> branch1.new_child() | branch2
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for |: 'ChainMap' and 'ChainMap'
```

С версии 3.9:

```
>>> division1 = {'Иванов': 12, 'Петров': 15, 'Сидоров': 11}
>>> division2 = {'Смирнов': 4, 'Дроздов': 5, 'Носов': 7}
>>> division3 = {'Зуев': 22, 'Абрамов': 19, 'Шарапов': 25}
>>> division4 = {'Павлов': 6, 'Чернов': 3, 'Ильин': 5}
>>> branch1 = ChainMap(division1, division3)
>>> branch2 = ChainMap(division2, division4)
>>> branch1.new_child() | branch2
ChainMap({'Павлов': 6, 'Чернов': 3, 'Ильин': 5, 'Смирнов': 4, 'Дроздов': 5, 'Носов': 7}, {'Иванов': 12,
'Петров': 15, 'Сидоров': 11}, {'Зуев': 22, 'Абрамов': 19, 'Шарапов': 25})
```

Counter

Counter – часть модуля collections, вид словаря, который позволяет считать количество неизменяемых объектов.

Доступные методы и свойства:

- elements() – возвращает список элементов в лексикографическом порядке
- most_common([n]) – возвращает n наиболее часто встречающихся элементов, в порядке убывания встречаемости. Если n не указано, возвращаются все элементы
- subtract([iterable-or-mapping]) – вычитание

Изменения в версии 3.7:

Как подкласс dict, Counter унаследовал способность запоминать порядок вставки. Математические операции с объектами Counter также сохраняют порядок. Результаты упорядочиваются в соответствии с тем, как элемент впервые встречается в левом операнде, а затем в порядке, в котором он встречается в правом операнде.

До версии 3.7:

```
>>> Counter("Троллейбус")
Counter({'л': 2, 'е': 1, 'р': 1, 'б': 1, 'Т': 1, 'у': 1, 'о': 1, 'с': 1, 'й': 1})
>>> Counter("Метро")
Counter({'М': 1, 'е': 1, 'о': 1, 'р': 1, 'т': 1})
```

С версии 3.7:

```
>>> Counter("Троллейбус")
Counter({'л': 2, 'Т': 1, 'р': 1, 'о': 1, 'е': 1, 'й': 1, 'б': 1, 'у': 1, 'с': 1})
>>> Counter("Метро")
Counter({'М': 1, 'е': 1, 'т': 1, 'р': 1, 'о': 1})
```

defaultdict

defaultdict – часть модуля collections, ничем не отличается от обычного словаря за исключением того, что по умолчанию всегда вызывается функция, возвращающая значение.

Изменения в версии 3.9:

Добавлена поддержка операторов | и | =, указанных в PEP 584.

Пример:

До версии 3.9:

```
>>> a = defaultdict(list, {'Десятки тысяч': 1, 'Тысячи': 7, 'Сотни': 8})
>>> b = defaultdict(list, {'Десятки': 4, 'Единицы': 9})
>>> a | b
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for |: 'collections.defaultdict' and 'collections.defaultdict'
```

С версии 3.9:

```
>>> a = defaultdict(list, {'Десятки тысяч': 1, 'Тысячи': 7, 'Сотни': 8})
>>> b = defaultdict(list, {'Десятки': 4, 'Единицы': 9})
>>> a | b
defaultdict(<class 'list'>, {'Десятки тысяч': 1, 'Тысячи': 7, 'Сотни': 8, 'Десятки': 4, 'Единицы': 9})
```

namedtuple

namedtuple – часть модуля collections, класс, который позволяет создать тип данных, ведущий себя как кортеж, с тем дополнением, что каждому элементу присваивается имя, по которому можно в дальнейшем получать доступ.

Доступные методы и свойства:

- _make(iterable) – метод, который создает новый экземпляр из существующей последовательности или из iterable.
- _asdict() – возвращает новый dict, который сопоставляет имена полей с соответствующими значениями.
- _replace(**kwargs) – возвращает новый экземпляр названного кортежа, заменяя указанные поля новыми значениями.
- _fields – возвращает кортеж строк, в котором перечислены имена полей.
- _field_defaults – возвращает словарь, в котором имена полей сопоставлены значениям по умолчанию.

Изменения в версии 3.7:

Удалены параметр *verbose* и атрибут *_source*.

Пример:

До версии 3.7:

```
>>> Number = namedtuple('Number', 'Сотни Десятки Единицы', verbose = True)
from builtins import property as _property, tuple as _tuple
from operator import itemgetter as _itemgetter
from collections import OrderedDict

class Number(tuple):
   'Number(Сотни, Десятки, Единицы)'

   __slots__ = ()

   _fields = ('Сотни', 'Десятки', 'Единицы')

   def __new__(_cls, Сотни, Десятки, Единицы):
      'Create new instance of Number(Сотни, Десятки, Единицы)'
      return _tuple.__new__(_cls, (Сотни, Десятки, Единицы))

   @classmethod
   def _make(cls, iterable, new=tuple.__new__, len=len):
      'Make a new Number object from a sequence or iterable'
      result = new(cls, iterable)
      if len(result) != 3:
         raise TypeError('Expected 3 arguments, got %d' % len(result))
      return result

   def _replace(_self, **kwds):
      'Return a new Number object replacing specified fields with new values'
      result = _self._make(map(kwds.pop, ('Сотни', 'Десятки', 'Единицы'), _self))
      if kwds:
```

```
        raise ValueError('Got unexpected field names: %r' % list(kwds))
    return result

  def __repr__(self):
    'Return a nicely formatted representation string'
    return self.__class__.__name__ + '(Сотни=%r, Десятки=%r, Единицы=%r)' % self

  @property
  def __dict__(self):
    'A new OrderedDict mapping field names to their values'
    return OrderedDict(zip(self._fields, self))

  def _asdict(self):
    '''Return a new OrderedDict which maps field names to their values.
       This method is obsolete.  Use vars(nt) or nt.__dict__ instead.
    '''
    return self.__dict__

  def __getnewargs__(self):
    'Return self as a plain tuple.  Used by copy and pickle.'
    return tuple(self)

  def __getstate__(self):
    'Exclude the OrderedDict from pickling'
    return None

  Сотни = _property(_itemgetter(0), doc='Alias for field number 0')

  Десятки = _property(_itemgetter(1), doc='Alias for field number 1')

  Единицы = _property(_itemgetter(2), doc='Alias for field number 2')


Number._source
"from builtins import property as _property, tuple as _tuple\nfrom operator import itemgetter as
_itemgetter\nfrom collections import OrderedDict\n\nclass Number(tuple):\n    'Number(Сотни,
Десятки, Единицы)'\n\n    __slots__ = ()\n\n    _fields = ('Сотни', 'Десятки', 'Единицы')\n\n    def
__new__(_cls, Сотни, Десятки, Единицы):\n        'Create new instance of Number(Сотни, Десятки,
Единицы)'\n        return _tuple.__new__(_cls, (Сотни, Десятки, Единицы))\n\n    @classmethod\n    def
_make(cls, iterable, new=tuple.__new__, len=len):\n        'Make a new Number object from a sequence
or iterable'\n        result = new(cls, iterable)\n        if len(result) != 3:\n            raise TypeError('Expected 3
arguments, got %d' % len(result))\n        return result\n\n    def _replace(_self, **kwds):\n        'Return a
new Number object replacing specified fields with new values'\n        result =
_self._make(map(kwds.pop, ('Сотни', 'Десятки', 'Единицы'), _self))\n        if kwds:\n            raise
ValueError('Got unexpected field names: %r' % list(kwds))\n        return result\n\n    def __repr__(self):\n
'Return a nicely formatted representation string'\n        return self.__class__.__name__ + '(Сотни=%r,
Десятки=%r, Единицы=%r)' % self\n\n    @property\n    def __dict__(self):\n        'A new OrderedDict
mapping field names to their values'\n        return OrderedDict(zip(self._fields, self))\n\n    def
_asdict(self):\n        '''Return a new OrderedDict which maps field names to their values.\n            This
method is obsolete.  Use vars(nt) or nt.__dict__ instead.\n        '''\n        return self.__dict__\n\n    def
__getnewargs__(self):\n        'Return self as a plain tuple.  Used by copy and pickle.'\n        return
tuple(self)\n\n    def __getstate__(self):\n        'Exclude the OrderedDict from pickling'\n        return
None\n\n    Сотни = _property(_itemgetter(0), doc='Alias for field number 0')\n\n    Десятки =
```

```
_property(_itemgetter(1), doc='Alias for field number 1')\n\n    Единицы = _property(_itemgetter(2), doc='Alias for field number 2')\n\n"
```

С версии 3.7:

```
>>> Number = namedtuple('Number', 'Сотни Десятки Единицы', verbose = True)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: namedtuple() got an unexpected keyword argument 'verbose'
>>> Number.source
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: type object 'Number' has no attribute 'source'
```

Добавлен параметр *defaults* и атрибут *_field_defaults*.

Пример:

До версии 3.7:

```
>>> Number = namedtuple('Number', 'Сотни Десятки Единицы', defaults=[0, 0, 0])
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: namedtuple() got an unexpected keyword argument 'defaults'
```

С версии 3.7:

```
>>> Number = namedtuple('Number', 'Сотни Десятки Единицы', defaults=[0, 0, 0])
>>> Number._field_defaults
{'Сотни': 0, 'Десятки': 0, 'Единицы': 0}
```

OrderedDict

OrderedDict – часть модуля collections, объект, похожий на словарь, который запоминающий порядок, в котором ему были даны ключи.

Доступные методы и свойства:

- popitem(last=True) – удаляет последний элемент если last=True, и первый, если last=False.
- move_to_end(key, last=True) – добавляет ключ в конец если last=True, и в начало, если last=False.

Изменения в версии 3.7:

Добавлены операторы слияния (|) и обновления (| =), указанные в PEP 584.

Пример:

До версии 3.9:

```
>>> a = OrderedDict([('Яблоки', 10), ('Груши', 7), ('Сливы', 16)])
>>> b = OrderedDict([('Бананы', 4), ('Апельсины', 12)])
>>> a | b
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for |: 'collections.OrderedDict' and 'collections.OrderedDict'
```

С версии 3.9:

```
>>> a = OrderedDict([('Яблоки', 10), ('Груши', 7), ('Сливы', 16)])
>>> b = OrderedDict([('Бананы', 4), ('Апельсины', 12)])
>>> a | b
OrderedDict([('Яблоки', 10), ('Груши', 7), ('Сливы', 16), ('Бананы', 4), ('Апельсины', 12)])
```