УДК 004.423 **Информационные технологии**

Халевин Тимофей Анатольевич, *студент факультета информатики и вычислительной техники, Хакасский государственный университет имени Н.Ф. Катанова, г. Абакан, Россия*

Голубничий Артем Александрович, *научный руководитель, старший преподаватель кафедры ПОВТиАС, Хакасский государственный университет имени Н.Ф. Катанова, г. Абакан, Россия*

МОДУЛЬ COLLECTIONS В ЯЗЫКЕ ПРОГРАММИРОВАНИЯ PYTHON. NAMEDTUPLE И DEQUE

Аннотация: В статье рассматриваются контейнерные типы данных namedtuple и deque модуля collection языка программирования Python.

Ключевые слова: Python, collections, типы данных, namedtuple, deque.

Annotation: This article discusses the container data types namedtuple and deque of the collection module of the Python programming language.

Keywords: Python, collections, data types, namedtuple, deque.

Большинство решений в современных языках программирования созданы для того, чтобы упростить жизнь программисту своим функционалом. Модуль collections в языке Python реализует специализированные контейнерные типы данных, обеспечивающие альтернативы встроенным контейнерам общего назначения, таким как dict, list, set и tuple.

В модуле collections языка программирования Python встречаются 9 типов данных: namedtuple, deque. ChainMap, Counter, OrderedDict, defaultdict, UserDict, UserList и UserString. В статье детально рассмотрены два из них: namedtuple и deque.

Тип данных namedtuple – это функция, для создания подклассов кортежей с именованными полями. Если говорить проще, то этот тип данных ведет себя как кортеж, но каждому элементу присваивается имя, по которому можно обращаться к элементу кортежа.

Разберем пример работы с namedtuple, представленный на рисунке 1.

```
>>> from collections import namedtuple
>>> pythagoras_triangle = namedtuple('pythagoras_triangle', ['AB', 'BC', 'AC'])
>>> pt = pythagoras_triangle(12, 13, 5)
>>> type(pt)
<class '__main__.pythagoras_triangle'>
>>> pt
pythagoras_triangle(AB=12, BC=13, AC=5)
>>> pt[0]
12
>>> pt.AB
12
>>> pt.AB**2 + pt.AC**2 == pt.BC**2
True
>>> pt = pt._replace(AC=6) # замена значения
>>> pt
pythagoras_triangle(AB=12, BC=13, AC=6)
>>> pt.AB**2 + pt.AC**2 == pt.BC**2
False
```

Рисунок 1 – Пример работы с namedtuple

Для начала импортируем namedtuple из модуля collections. Далее создается объект namedtuple, где первым аргументом передается название типа, а вторым поля нашего типа. Для примера создаем тип pythagoras_triangle со сторонами треугольника. При вызове функции type будет получен ответ сообщающий, что переменная pt хранит тип данных pythagoras_triangle. Так же есть возможность обращаться к данным двумя способами: как к обычному tuple, через квадратные скобки явно указывая индекс элемента, или через обращение к переменной с помощью точки. Второй вариант идентичен обращению к переменным класса.

Также есть возможность изменять значение переменной в namedtuple. Но напрямую это сделать не получится, будет выведена ошибка. Для изменения значения необходимо воспользоваться методом _replace в который передаются аргументы, которые необходимо изменить. Важно также отметить, что данный

метод не изменяет исходные данные, он создает новую копию с уже измененными значениями, поэтому необходимо переписать измененный объект в переменную или положить его в новую переменную.

Namedtuple содержит методы, рассмотрим их на рисунке 2.

```
>>> from collections import namedtuple
>>> Person = namedtuple('Person', ['name', 'surname', 'age', 'gender'])
>>> man = Person(name='Alex', surname='Benti', age=24, gender='male')
>>> info = ['Anny', 'Tad', 16, 'female']
>>> woman = Person._make(info)
>>> woman
Person(name='Anny', surname='Tad', age=16, gender='female')
>>> man._fields
('name', 'surname', 'age', 'gender')
>>> woman._asdict()
{'name': 'Anny', 'surname': 'Tad', 'age': 16, 'gender': 'female'}
>>> Person = namedtuple('Person', ['name', 'surname', 'age', 'gender'], defaults=[None, None, None, 'male'])
>>> unknown = Person(name='Vadle', surname='Castr')
>>> unknown
Person(name='Vadle', surname='Castr', age=None, gender='male')
```

Рисунок 2 – Методы namedtuple

- _make(iterable) принимает в качестве аргумента итерируемый объект и создает из данных этого объекта новый экземпляр.
- _fields выводит кортеж строк со списком имен полей.
- _asdict() не принимает значений и возвращает в качестве результата своей работы словарь, ключами которого будут являться имена полей объекта.

Теперь перейдем к такому типу данных как deque. Deque является двусторонней очередью.

На рисунке 3 представлены методы добавления в deque элементов.

```
>>> from collections import deque
>>> dq = deque()
>>> dq.append('_')
>>> dq
deque(['_'])
>>> dq.appendleft('e')
>>> dq
deque(['e', '_'])
>>> dq.extend('python')
>>> dq
deque(['e', '_', 'p', 'y', 't', 'h', 'o', 'n'])
>>> dq.extendleft('ueuq')
>>> dq
deque(['q', 'u', 'e', 'u', 'e', '_', 'p', 'y', 't', 'h', 'o', 'n'])
```

Рисунок 3 – Методы добавления элементов в deque

Двусторонняя очередь очень похожа на список в языке Python, но в отличие от списка можно добавлять элементы в начало и доставать их оттуда.

У deque есть метод append(element) и appendleft(element), которые принимают на вход элемент и добавляют его в конец или начало двусторонней очереди соответственно.

Также есть методы extend(iterable) и extendleft(iterable). Они принимают итерируемый объект, и вставляют каждый элемент в конец и начало двусторонней очереди соответственно. Причем extendleft добавляет элементы в обратной последовательности.

Есть возможность вращения двусторонней очереди с использованием метода rotate(number) где в качестве аргумента передается целое число, количество элементов, которые нужно повернуть. При передаче целого положительного числа n, n последних элементов перенесутся в начало. Также можно передать отрицательное число, это будет обратный поворот, n первых элементов перейдут в конец. На рисунке 4 представлено выполнение метода rotate(number).

```
>>> dq.rotate(3)
>>> dq
deque(['h', 'o', 'n', 'q', 'u', 'e', 'u', 'e', '_', 'p', 'y', 't'])
>>> dq.rotate(-3)
>>> dq
deque(['q', 'u', 'e', 'u', 'e', '_', 'p', 'y', 't', 'h', 'o', 'n'])
```

Рисунок 4 – Выполнение метода rotate

Есть и другие методы которые могут пригодиться. Вставка по индексу с помощью метода insert(index, element). Удаление первого вхождения элемента с помощью метода remove(element). Метод reverse() для полного переворота очереди. Данные методы показаны на рисунке 5.

```
>>> dq.insert(5, '|')
>>> dq
deque(['q', 'u', 'e', 'u', 'e', '|', '_', 'p', 'y', 't', 'h', 'o', 'n'])
>>> dq.remove('_')
>>> dq
deque(['q', 'u', 'e', 'u', 'e', '|', 'p', 'y', 't', 'h', 'o', 'n'])
>>> dq.reverse()
>>> dq
deque(['n', 'o', 'h', 't', 'y', 'p', '|', 'e', 'u', 'e', 'u', 'q'])
```

Рисунок 5 – Методы вставки, удаления и переворота двусторонней очереди

Конечно имеется возможность удалять элементы из двусторонней очереди с её концов. Для этого используются методы pop() и popleft() для удаления с конца и с начала соответственно. Удаленные элементы возвращаются в качестве значения после использования метода. Методы удаления показаны на рисунке 6.

```
>>> dq.pop()
'n'
>>> dq.popleft()
'q'
>>> dq
deque(['u', 'e', 'u', 'e', '|', 'p', 'y', 't', 'h', 'o'])
```

Рисунок 6 – Методы pop и popleft

Заключение

Подводя итоги можно сказать, что namedtuple модуля collection предоставляет возможность доступа к элементам или по индексу, или по указанному имени. Это увеличивает читаемость кода. Двусторонняя очередь deque модуля collection поддерживает эффективные по памяти операции добавления и извлечения элементов с любой стороны.

Библиографический список:

1. Бэрри П. Изучаем программирование на Python [Текст] / П. Бэрри. – М.: Вильямс, 2014. – 243 с.

2. Гэддис Т. Начинаем программировать на Python [Текст] / Т. Гэддис. – СПб.: БХВ-Петербург, 2021. – 768 с.