УДК 004.423 **Информационные технологии**

Халевин Тимофей Анатольевич, *студент направления подготовки информатика и вычислительная техника, 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. COUNTER И ORDEREDDICT

Аннотация: В статье рассматриваются контейнерные типы данных Counter и OrderedDict модуля collection языка программирования Python.

Ключевые слова: Python, collections, типы данных, Counter, OrderedDict.

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

Keywords: Python, collections, data types, Counter, OrderedDict.

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 [1].

В модуле collections языка программирования python встречаются 9 типов данных: namedtuple, deque. ChainMap, Counter, OrderedDict, defaultdict, UserDict, UserList и UserString. В данной статье подробно будут рассмотрены Counter и

OrderedDict.

Тип данных Counter – это функция, для создания подклассов словарей с подсчетом элементов. Этот тип данных идентичен словарю, но также имеет дополнительные методы, которые будут разобраны далее.

Разберем пример работы с Counter, представленный на рисунке 1.

```
>>> from collections import Counter
>>> string = 'Counter - функция модуля collections языка программирования Python'
>>> string = string.replace(' ', '')
>>> string
'Counter-функциямодуляcollectionsязыкапрограммированияPython'
>>> symbols = Counter(string)
>>> symbols
Counter({'о': 4, 'я': 4, 'n': 3, 't': 3, 'и': 3, 'м': 3, 'о': 3, 'а': 3, 'р': 3, 'е': 2, 'у': 2, 'н': 2, 'к':
2, 'с': 2, 'l': 2, 'C': 1, 'u': 1, 'r': 1, '-': 1, 'ф': 1, 'ц': 1, 'д': 1, 'л': 1, 'і': 1, 'з': 1, 'э': 1, 'ы'
: 1, 'п': 1, 'г': 1, 'в': 1, 'P': 1, 'y': 1, 'h': 1})
```

Рисунок 1 – Пример работы с Counter

Для начала импортируем Counter из модуля collections. После создадим строку, которая будет хранить предложение «Counter – функция модуля collections языка программирования Python».

Предположим, что необходимо узнать сколько раз каждая буква встречается в предложении. Пробелы в строке нам не нужны, поэтому избавляемся от них с помощью метода replace (pattern, value, count), где pattern – это строка, которую мы хотим изменить на value. Количество изменений определяются параметром count, который по стандарту равен -1, что означает замену всех встреченных паттернов.

Теперь вызываем Counter(string), где аргументом будет наша строка. На выходе получаем аналог словаря где ключом будут являться символы, а значением их количество в строке.

Также есть возможность итерироваться по данному типу данных как по словарю. Пример итерации представлен на рисунке 2.

```
>>> cnt = Counter('тест')
>>> for key, value in cnt.items():
...     print(f' Символ {key} встречается в строке {value} раз')
...
...
 Символ т встречается в строке 2 раз
 Символ е встречается в строке 1 раз
 Символ с встречается в строке 1 раз
```

Рисунок 2 – Итерация по Counter

Так же у Counter есть методы, рассмотрим их на рисунке 3.

```
>>> from collections import Counter
>>> cnt = Counter('abracadabra')
>>> list(cnt.elements())
['a', 'a', 'a', 'a', 'a', 'b', 'b', 'r', 'r', 'c', 'd']
>>> cnt.most_common(2)
[('a', 5), ('b', 2)]
>>> cnt.total()
11
>>> cnt.update(['a','r'])
>>> cnt.most_common(2)
[('a', 6), ('r', 3)]
>>> sb = Counter(a=1, r=1, k=2)
>>> cnt - sb
Counter({'a': 5, 'b': 2, 'r': 2, 'c': 1, 'd': 1})
>>> cnt + sb
Counter({'a': 7, 'r': 4, 'b': 2, 'k': 2, 'c': 1, 'd': 1})
>>> cnt & sb
Counter({'a': 1, 'r': 1})
>>> cnt | sb
Counter({'a': 6, 'r': 3, 'b': 2, 'k': 2, 'c': 1, 'd': 1})
>>> cnt == sb
False
>>> cnt.update(k = -5)
>>> +cnt
Counter({'a': 6, 'r': 3, 'b': 2, 'c': 1, 'd': 1})
>>> -cnt
Counter({'k': 5})
>>> cnt['f']
0
```

Рисунок 3 – Методы Counter

Метод elements() возвращает объект со всеми элементами Counter. Для получения элементов, например, в виде списка, необходимо обернуть метод в list().

Метод most_common(n) в качестве аргумента принимает число и возвращает

n самых часто встречаемых элементов.

Вычитание двух объектов Counter cnt – sb, оставляет только элементы ключи которых остались положительными после вычитания.

Сложение двух объектов Counter cnt + sb, складывает значения двух ключей.

Пересечение двух объектов Counter cnt & sb, выбирает минимальное значение из двух.

Сравнение двух объектов Counter cnt == sb, возвращает True если объекты идентичны, иначе False.

Добавление знака + перед объектом типа Counter возвращает все элементы, значения которых положительны.

Добавление знака – перед объектом типа Counter возвращает элементы, значения которых отрицательны. Но при этой операции значения элементов на выводе изменятся на положительные, это важно учитывать

Так же мы можем получить значение элемента через обращение по ключу. Если элемента с переданным ключом нет, то мы получим нулевое значение.

Теперь перейдем к такому типу данных как OrderedDict. OrderedDict возвращает экземпляр подкласса dict, который имеет методы, специализированные для изменения порядка словаря [2].

На рисунке 4 представлены методы добавления в OrderedDict элементов.

```
>>> from collections import OrderedDict
>>> od = OrderedDict.fromkeys('example', value = 1)
>>> od
OrderedDict([('e', 1), ('x', 1), ('a', 1), ('m', 1), ('p', 1), ('l', 1)])
>>> od['f'] = 1
>>> od
OrderedDict([('e', 1), ('x', 1), ('a', 1), ('m', 1), ('p', 1), ('l', 1), ('f', 1)])
```

Рисунок 4 – Методы добавления элементов в OrderedDict

OrderedDict имеет дополнительные методы для работы со словарем. Начнем с метода для получения элемента из словаря. Метод popitem(last=True) возвращает

последний элемент словаря. Аргумент last является булевым и означает откуда возьмется элемент словаря, из начала при ложном аргументе или конца при истинном аргументе. На рисунке 5 представлено выполнение данного метода.

```
>>> od.popitem(last=True)
('f', 1)
>>> od
OrderedDict([('e', 1), ('x', 1), ('a', 1), ('m', 1), ('p', 1), ('l', 1)])
>>> od.popitem(last=False)
('e', 1)
>>> od
OrderedDict([('x', 1), ('a', 1), ('m', 1), ('p', 1), ('l', 1)])
```

Рисунок 5 – Выполнение метода popitem

Так же мы можем изменять порядок элементов в словаре с помощью метода move_to_end(element, last=True). Данный метод принимает два аргумента: элемент словаря и булево значение last которое отвечает за то, куда будет перемещен элемент, в конец словаря при истинном значении или в начало при ложном. На рисунке 6 представлено выполнение данного метода.

```
>>> od.move_to_end('x')
>>> od
OrderedDict([('a', 1), ('m', 1), ('p', 1), ('l', 1), ('x', 1)])
>>> od.move_to_end('m', last=False)
>>> od
OrderedDict([('m', 1), ('a', 1), ('p', 1), ('l', 1), ('x', 1)])
```

Рисунок 6 – Выполнение метода move_to_end

Есть возможность так же итерироваться по OrderedDict как по словарю, а также можно итерироваться в обратном порядке. Пример итерирования по OrderedDict в обратном порядке представлен на рисунке 7.

```
>>> for key, value in reversed(od.items()):
...     print(f'{key}: {value}')
...
...
x: 1
l: 1
p: 1
a: 1
m: 1
```

Рисунок 7 – Итерирование в обратном порядке по OrderedDict

Заключение

Подводя итоги можно сказать, что Counter модуля collection предоставляет возможность для подсчета элементов в итерируемом объекте. Это позволяет сократить время для решения некоторых задач. OrderedDict модуля collection представляет собой расширение обычного словаря в языке Python и может пригодиться для специализированных задач в которых важен порядок.

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.