Управляющие конструкции и коллекции (продолжение)

Рассмотрим следующие структуры:

- словари (dict)
- множества (set)
- кортежи (tuple)

Кроме того, затронем тему list comprehension.

СЛОЖНОСТЬ АЛГОРИТМОВ

Поиск по списку можно осуществлять разными способами:

- простой перебор
- бинарный (двоичный) поиск - подобен телефонному справочнику, где есть изначальная сортировка по алфавиту
- по хэш-таблице (подробнее по ссылке в ноутбуке) - в Python используется в множествах и словарях.
- иные способы

Сложность алгоритма перебора зависит от количества элементов. От этого зависит и скорость работы. При бинарном поиске зависимость остаётся, но ослабевает. При поиске по хэш-таблице в большинстве случаев сложность алгоритма не будет зависеть от количества элементов.

Для определения времени в Jupyter Notebook используется команда %%time. Нотация %% сообщает, что это не часть программы, а особая команда.

При построении алгоритма версия на основе списков на несколько порядков проигрывает версии на основе множеств. Объясняется это удалением дублей в множестве, более оптимальным методом поиска. Кроме того, во втором случае алгоритм не зависит от объёма данных, поскольку время поиска по множеству не зависит от количества элементов ввиду использования хэш-таблиц.

LIST COMPREHENSION

Это синтаксический сахар, который позволяет более кратко писать код в определённых случаях.

Синтаксис: [выражение цикл условие отбора]

Пусть имеется некоторая последовательность sequence (любая). Например, [1, 2, 3, 4, 5]
Пример цикла без list comprehension:

```
for x in sequence:
    if x % 5 == 0:
        print x
```

То же самое с list comprehension:
`[x for x in sequence if x % 5 == 0]`

Вывод одинаковый: {5}

Рекомендуется не делать вложенные циклы с помощью list comprehension, поскольку их тяжело читать.

СЛОВАРИ

Представляют собой структуру с парами "Ключ: значение". Это позволяет удобно обращаться к элементам словаря. В качестве ключа могут быть числа, строки, иные типы данных (за некоторыми исключениями). Напоминают формат JSON, могут быть переведены в этот формат и обратно. По умолчанию элементы словаря не отсортированы. Поэтому при некоторых операциях элементы могут менять порядок.

Важно: при обращении к словарю ключ должен существовать. По этой причине нужно использовать проверку на существование ключа, задание значения по умолчанию или безопасное обращение (возврат некоторого значения, которое можно задать, если нет ключа).

Базовые операции:

- обращение к элементу
- перебор ключей (.keys())
- перебор значений (.values())
- перебор ключей и значений (.items())
- получение количества ключей (len())
- значение по умолчанию (.setdefault())
- безопасное обращение (.get())

СЛОВАРИ И СПИСКИ

Операции:

- совмещение 2 списков с использованием zip(). Применение dict(zip()) приведёт 2 списка к одному словарю
- приведение списка к словарю с использованием аналога list comprehension: `{x: f(x) for x in list_}`, где f(x) - функция от x, либо `{x: y for x, y in list_}`, если в списке лежали пары значений
- сортировка вложенных словарей (sorted() с дополнительными параметрами)
- группировка значений: если имеется список списков, то можно сделать n-ное значение каждого вложенного списка ключом словаря. Полученный словарь будет аналогом сгруппированного по n-ому значению списка.

КОРТЕЖИ

Очень похожи на списки, но есть отличия

- неизменяемый
- занимают меньше места (можно проверить с помощью .__sizeof__())
- может быть ключом для словаря (что позволяет намного ускорить поиск по нему, поскольку остальные элементы можно поместить в кортеж)

Важно: для задания кортежа из 1 элемента необходима запятая: tuple = ('1',)

Операции над кортежем аналогичны операциям над списком, которые не меняют его.

МНОЖЕСТВА

Также похожи на списки, но состоят из неповторяющихся элементов. Операции также аналогичны операциям над списком.
Множества удобны для решения определённых задач, которые требуют определения уникальных элементов. Например, для пересечения списков.

Все вышеописанные структуры можно удобно записывать в Excel при подключении библиотеки Workbook.