Лабораторная работа №2

Основные элементы языка Python

Темы:

1. Типы данных.
2. Строки и форматирование.
3. Ввод-вывод.
4. Модули.
5. Обработка ошибок.
6. Классы и метаклассы.
7. Генераторы и итераторы.
8. Декораторы.
9. Дескрипторы.

Критерии приема задач:

- Решение поставленной задачи.
- Хорошее оформление и читаемость кода.
- Теория по теме задач.

Требования:

- Каждое задание - отдельный модуль, который можно вызывать как файл или импортировать и использовать как библиотеку.
- Там где это нужно, создать свои классы-исключения для сигнализирования о специфических ошибках или использовать уже существующие. Обрабатывать ошибки вызываемых библиотек.
- В заданиях нельзя просто использовать встроенные аналоги запрашиваемых элементов из языка или библиотек - надо реализовывать самостоятельно. Пример: в json-преобразователе нельзя использовать встроенный модуль json.

Основные задания

1. Сортировка большого объема данных.
    - Написать программу для сортировки данных не помещающихся в оперативную память.
    - В качестве входных данных выступает файл с текстовыми данными (или сами данные), в качестве выходных – один сортированный файл (или сами данные).
    - Файлы могут быть очень большие и превышать объем доступной оперативной памяти на порядок.
    - Входные данные представляют собой последовательность ascii строк, каждая из которых оканчивается символом разделителя строк (по-умолчанию \n, но должна быть возможность изменить с помощью аргумента командной строки --line-separator='символ_разделитель').
    - Каждая строка состоит из последовательностей символов разделённых символом разделителя частей строки (по-умолчанию \t, но должна быть возможность изменить с помощью аргумента командной строки -t 'символ_разделитель' или --field-separator='символ_разделитель').
    - С помощью аргумента командной строки (-k 1,2,5; --key 1,2,5) можно перечислить индексы частей строки используемых для сортировки (по-умолчанию используются все позиции).
    - С помощью аргумента командной строки можно указать интерпретировать части строки как числа (-n, --numeric).
    - С помощью аргумента командной строки --reverse можно указать сортировку в обратном порядке.
    - С помощью аргумента командной строки можно указать режим проверки сортированности --check: в этом режиме сортировка не производится, а входные данные просто проверяются на отсортированность согласно правилам указанным в аргументах командной строки.
    - Данные в файле сортируются построчно (меняется порядок строк).
    - Порядок сортировки определяется с помощью частей строки в соответствии с указанными аргументами (или их значениями по-умолчанию).
    - Строки сортируются в лексикографическом порядке на основе частей строки описанных разделителями.
    - Пример сортировки двух строк (спец. символы и кавычки написаны явно для примера):
      "cc\tbb\taa\ncc\taa\tbb\naa\tbb\tcc\n" -> "aa\tbb\tcc\ncc\taa\t\bb\ncc\tbb\taa\n"
    - Промежуточные результаты рекомендуется хранить во временных файлах (обратите внимание на модуль tempfile), а саму сортировку – делать с помощью сортировки слиянием с использованием фиксированного буфера в памяти (размер буфера можно указать в качестве аргумента командной строки --buffer-size=число_байт).

- Интерфейс использования:

cat in.txt | python sort.py > out.txt # читаем данные из стандартного ввода и выводим на стандартный вывод

python sort.py --input in.txt --output out.txt # читаем из файла и пишем результат в файл

2. Написать программу генерирующую входной файл (произвольного размера) для задачи сортировки больших файлов описанной выше. Файл должен соответствовать спецификации указанной в этой задаче. Должна быть возможность с помощью аргументов командной строки изменить вид генерируемых файлов:
    - Фиксированное или переменное количество полей в одной строке.
    - Числа или произвольные ascii текстовые данные в качестве полей строки.
    - Разделитель полей в строке.
    - Разделитель строк.
    - Количество строк.
3. Свой преобразователь в JSON. Реализовать функцию to_json(obj), которая на вход получает python-объект, а на выходе у неё строка в формате JSON. Создать собственное исключение (унаследовав от подходящего встроенного), которое должно выбрасываться при попытке преобразования неизвестного типа, но только если при вызове to_json был передан опциональный параметр raise_unknown=True (по-умолчанию False). В исключении должна сохраняться информация о типе, которые попытались преобразовать: этот тип должен выводиться при преобразовании исключения в строковое представление.
4. Класс “n-мерный вектор”. У этого класса должны быть определены все естественные для вектора операции – сложение, вычитание, умножение на константу и скалярное произведение, сравнение на равенство. Кроме этого должны быть операции вычисления длины, получение элемента по индексу, а также строковое представление.
5. Класс логгер с возможностью наследования. Класс должен логировать то, какие методы и с какими аргументами у него вызывались и какой был результат этого вызова. Функция str() от этого класса должна отдавать лог вызовов. Должна быть возможность унаследоваться от такого класса, чтобы добавить логгирование вызовов у любого класса. При форматировании строк использовать метод format.
6. Рекурсивный defaultdict. Реализовать свой класс-аналог defaultdict, который позволяет рекурсивно читать и записывать значения в виде d[“a”][“b”] = 1, а при вызове str(d) выводит данные как словарь в текстовом представлении.
7. Метакласс берущий поля класса из файла. Реализовать метакласс, который позволяет при создании класса добавлять к нему произвольные аттрибуты (классу, не экземпляру класса), которые загружаются из файла. В файле должны быть имена аттрибутов и их значения. Нужно уметь передавать путь к файлу как изменяемый параметр.

8. Декоратор @cached, который сохраняет значение функции при каждом вызове. Если функция вызвана повторно с теми же аргументами, то возвращается сохраненное значение, а функция не вычисляется.
9. Свой xrange. Реализовать полностью свой xrange с аналогичным встроенному интерфейсом.
10. Последовательность с фильтрацией. Реализовать класс, соответствующий некоторой последовательности объектов и имеющий следующие методы:
    - Создать объект на основе произвольного iterable объекта.
    - Итерирование (__iter__) по элементам (неистощаемое – можно несколько раз использовать объект в качестве iterable для for).
    - Отфильтровать последовательность с помощью некоторой функции и вернуть новую сокращенную последовательность, в которой присутствуют только элементы, для которых эта функция вернула True.

    Будем считать, что в данной задаче основным приоритетом является экономия памяти. Поэтому количество копирований данных нужно свести к минимуму (возможно, пожертвовав скоростью работы): например, если применим несколько раз по цепочке операции фильтрации, то данные об элементах последовательности не должны дублироваться.

Дополнительные задания

1. Свой преобразователь из JSON. Реализовать функцию from_json(text), которая возвращает python-объект соответствующий json-строке. Не использовать стандартные инструменты работы с JSON. (4 балла)
2. Синглтон. Реализовать шаблон проектирования Singleton, который можно применять на произвольный класс. Разработать самостоятельно, как этот инструмент будет применяться к целевому классу (например, модифицировать исходный класс или изменять способ вызова конструктора). (2 балла)
3. Поддержка параметризованных форматов вывода для класса-логгера. Параметры можно зафиксировать зарание (имя функции, имена аргументов, возвращаемое значение и др.) или придумать как это можно давать задавать пользователю. (1 балла)
4. Метакласс model creator. Реализуйте метакласс ModelCreator, позволяющий объявлять поля класса в следующем виде:

```
class Student ( object ):
    __metaclass__ = ModelCreator
    name = StringField ()
```

   Здесь StringField — некоторый объект, который обозначает, что это поле является текстовым. После такого вызова должен быть создан класс, конструктор которого принимает именованный аргумент name и сохраняет его в соответствующий

атрибут (с возможной проверкой и приведением типа). Таким образом должна быть возможность писать так:

```
s = Student (name =’abc’)
print s.name
```

Постарайтесь добиться того, чтобы создание класса было как можно более гибким: чтобы допускалось наследование и т.п. Обязательно должна быть проверка типов (например, в текстовое поле нельзя записать число). (4 балла)

5. Юниттесты. Использовать любой фреймворк для тестирования (unittest, nose, pytest). Использовать модуль coverage для оценки покрытия кода тестами. На каждое основное задание написать 2+ тестов (корректная работа, некорректная работа). (3 балла)
6. Реализовать свой устанавливаемый (setup.py) пакет со всеми заданиями этой лабораторной, разбитой на модули. Описать зависимости, указать скрипты для запуска заданий из каждого пункта. После установки они должны вызываться как системные команды (`user@host: lab2_task1.py`). (3 балла)