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 заданиямх 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 слиянием во внешней памяти.
   На вход поступает файл с числами в произвольном формате. Размер файла должен быть достаточно большим, чтобы на обычную сортировку слиянием не хватало оперативной памяти. На выходе получается файл с этими числами, упорядоченными по возрастанию. Промежуточные результаты хранить во временных файлах (рекомендуется модуль tempfile). Временные файлы после использования удаляются.
   Создать большой файл с числами можно таким скриптом
   ```
   import random
   with open('numbers.txt', 'w') as f:
       f.writelines('{}\n'.format(random.randint(-1000000, 1000000)) for _ in range(500000000))
   ```
2. Свой преобразователь в JSON. Реализовать функцию to_json(obj), которая на вход получает python-объект, а на выходе у неё строка в формате JSON.
3. 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 равествно. Кроме этого должны быть операции вычисления длины, получение элемента по индексу, а также строковое представление.
4. Класс “линейная функция”. Нужны следующие операции:
   - вычисление в точке (с помощью скобок ())
   - сложение с другой линейной функцией
   - умножение на константу
   - композиция с другой линейной функцией (с помощью скобок () или умножения)
   - строковое представление (как можно более похожее на математическую форму записи, можно взять за пример latex-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. У метакласса должен быть конструктор принимающий на вход имя файла.

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:
    a. Создать объект на основе произвольного iterable объекта.
    b. Итерирование (__iter__) по элементам (не истощаемое).
    c. Отфильтровать последовательность с помощью некоторой функции и вернуть новую расширенную 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).

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. (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)