Лабораторная работа №2

Умный rm

Общее описание

Написать консольную утилиту для удаления файлов/директорий с поддержкой “корзины” и возможностью восстановления удаленных объектов. Интерфейс вызова программы должен быть похож на системную команду rm с дополнительными возможностями.

Список базовых операций:

- удаление файла/директории (см. rm)
- удаление всех объектов по регулярному выражению в заданном поддереве (директории)
- просмотр корзины (подумать о возможных проблемах, например, как быть есть, если корзина огромная)
- ручная очистка корзины, автоматическая очистка по различным политикам
- восстановление объектов

Месторасположение корзины и политики её очистки (по максимальному объему, по времени, комбинированное и т.п.) должны задаваться пользователем через конфигурационный файл.

Требования к программе

Режимы работы:

- Вызовом “как rm”.
- Как библиотека, для переиспользования основной логики или вспомогательных функций из другого скрипта.

Задание конфигурации:

- Возможность передачи всех конфигурационных параметров по отдельности через аргументы командной строки.
- Опциональный конфигурационный файл для пользователя по умолчанию.
- Опциональный конфигурационный файл для конкретного запуска через аргументы командной строки.

Внутреннее устройство:

- Работа с аргументами командной строки с помощью модуля argparse.
- Продумать структуру программы и разбить на модули.
- Стараться оформить код так, чтобы отдельные функции можно было бы переиспользовать пользуясь программой как библиотекой.

Защита от ошибок:

- Лимиты и предупреждения на возможные превышения (слишком много файлов, системная директория и т.п.)
- Обнаружение циклов
- Политики разрешения конфликтов и возможность их выбора в конфигурационном файле и в аргументах.
- Учитывать свободное место.
- Режим с подтверждением операций пользователем перед применением.
- Режим dry-run - возможность запустить “имитацию” команды, результатом которой будет описание (с визуализацией) того, что бы произошло в результате её выполнения. При этом никакие изменения не применяются.
- ...

Отчёты:

- Результаты операций (+ silent)
    - Выводить подробные описания о сделанных изменениях и статистику.
    - Уметь отображать прогресс исполняемой операции.
    - Опция для silent-режима, когда программа ничего не пишет в потоки вывода, а просто завершается с некоторым кодом возврата.
- Техническое логирование операций
    - Программа логгирует свои действия с различными уровнями подробности, для того, чтобы разработчику было легче понимать и отлаживать её. Использовать модуль logging.

Форматы данных:

- Для конфигурационных файлов, отчётов и истории изменений поддержать как минимум два формата:
    - Человекочитаемый формат собственного изобретения для максимально удобного чтения при работе с программой.
    - JSON – для удобства использования в других программах.

Тесты:

- Основная функциональность должна быть покрыта юнит-тестами в отдельном файле (файлах). Тесты запускают программу в различных возможных режимах работы и проверяют результаты.
- Примеры фреймворков: unittest, nose, pytest и др. на выбор.

Установка:

- Установка с помощью setup.py.

Необязательные задания

Автодополнение команд для основных шеллов (bash, zsh).

Теория и практика к защите лабораторной

1. Темы рассмотренные на лекциях.
2. Практические задания ниже для самостоятельного решения и разбора: эти задания могут спросить на защите и по ним, и связанным темам, также могут быть вопросы.

Задания для самостоятельной подготовки

1. Статистики по тексту. На вход поступают текстовые данные. Необходимо посчитать и вывести:
    - сколько раз повторяется каждое слово в указанном тексте
    - среднее количество слов в предложении
    - медианное количество слов в предложении
    - top-K самых часто повторяющихся буквенных N-грам (K и N имеют значения по-умолчанию 10 и 4, но должна быть возможность задавать их с клавиатуры)

    При решении использовать контейнер dict() или его аналоги и встроенные операции над строками. Предусмотреть обработку знаков препинания.
2. Сортировки. На вход поступает строка содержащая числа разделённые пробелами. На выходе будут отсортированные по возрастанию числа. Необходимо реализовать алгоритмы:
    - быстрой сортировки

    - сортировки слияниями (при решении воспользоваться механизмом срезов (слайсы, slices))
    - поразрядной сортировки

  Необходимо знать алгоритмическую сложность реализованных алгоритмов.
3. Хранилище уникальных элементов. При запуске программа работает в интерактивном режиме и поддерживает команды:
    - add <key> [<key> …] - добавить один или более элементов в хранилище (если уже содержится, то не добавлять).
    - remove <key> - удалить элемент из хранилища.
    - find <key> [<key> …] - проверить наличие одного или более элементов в хранилище, вывести найденные.
    - list - вывести все элементы в хранилище.
    - grep <regexp> - поиск значения по регулярному выражению.
    - save и load - сохранить хранилище в файл и загрузить хранилище из файла

  При решении использовать контейнер set().
4. Генератор чисел Фибоначчи. Написать генератор возвращающий последовательно числа Фибоначчи начиная с первого.
5. Изучить модуль re и написать регулярные выражения для:
    - валидации email-адреса
    - валидации записи числа с плавающей строчкой
    - получения отдельных частей URL (схема, хост, порт, путь, параметры) с помощью механизма именованных групп
6. 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.
7. 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.
8. 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.

9. 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.
10. 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.
11. Декторатор @cached, который сохраняет значение функции при каждом вызове. Если функция вызвана повторно с теми же аргументами, то возвращается сохраненное значение, а функция не вычисляется.
12. Свой xrange. Реализовать полностью свой xrange с аналогичным встроенному интерфейсом.
13. 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.
14. Свой преобразователь из JSON. Реализовать функцию from_json(text), которая возвращает python-объект соответствующий json-строке. Не использовать стандартные инструменты работы с JSON.
15. 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).
16. 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.

17. 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).