Лабораторная работа №2

Тема: Работа с файловой системой и процессами в Python.

Цель: Научиться программировать на языке Python скрипты, взаимодействующие с файлами и процессами операционной системы.

Темы для предварительной проработки [УСТНО]:

1. Чтение/запись текстовых файлов.
2. Модули os, sys.
3. Модули shutil, glob, pathlib.
4. Модуль subprocess.
5. Регулярные выражения.

Индивидуальные задания [КОД]:

1. Напишите скрипт, который читает текстовый файл и выводит символы в порядке убывания частоты встречаемости в тексте. Регистр символа не имеет значения. Программа должна учитывать только буквенные символы (символы пунктуации, цифры и служебные символы слудет игнорировать). Проверьте работу скрипта на нескольких файлах с текстом на английском и русском языках, сравните результаты с таблицами, приведенными в wikipedia.org/wiki/Letter_frequencies.
2. Напишите скрипт, позволяющий искать в заданной директории и в ее подпапках файлы-дубликаты на основе сравнения контрольных сумм (MD5). Файлы могут иметь одинаковое содержимое, но отличаться именами. Скрипт должен вывести группы имен обнаруженных файлов-дубликатов.
3. Задан путь к директории с музыкальными файлами (в названии которых нет номеров, а только названия песен) и текстовый файл, хранящий полный список песен с номерами и названиями в виде строк формата «01. Freefall [6:12]». Напишите скрипт, который корректирует имена файлов в директории на основе текста списка песен.
4. Напишите скрипт, который позволяет ввести с клавиатуры имя текстового файла, найти в нем с помощью регулярных выражений все подстроки определенного вида, в соответствии с вариантом. Например, для варианта № 1 скрипт должен вывести на экран следующее:

```
Строка 3, позиция 10 : найдено '11-05-2014'
Строка 12, позиция 2 : найдено '23-11-2014'
Строка 12, позиция 17 : найдено '23-11-2014'
```

   Вариант 1: найдите все даты – подстроки вида «11-05-2014».
   Вариант 2: найдите все значения времени – подстроки вида «23:15:59».

Вариант 3: найдите все IPv4-адреса – подстроки вида «192.168.5.48».
Вариант 4: найдите все строки вида «*type x = value*», где *type* – это тип (может принимать значение int, short или byte), *x* – любое слово, *value* – любое число.
Вариант 5: найдите все номера телефонов – подстроки вида «(000)1112233» или «(000)111-22-33».
Вариант 6: найдите все строки вида «*x: type [N]*», где *type* – это тип (может принимать значение int, short или byte), *x* – любое слово, *N* – любое положительное целое число.
Вариант 7: найдите все «смайлы» – подстроки вида «:)», «:-)», «)))» (количество скобок может быть любым, начиная с 1).
Вариант 8: найдите все логические выражения – подстроки вида «x&&y», «x&y», где *x* и *y* – любые слова. Количество пробелов может быть также любым.
Вариант 9: найдите все донецкие почтовые индексы – подстроки вида «83000, Донецк» (первые 2 символа строго закреплены: «83»).
Вариант 10: Найдите все полные имена директорий Windows – подстроки вида «C:\Dir\SubDir3».

5. Введите с клавиатуры текст. Программно найдите в нем и выведите отдельно все слова, которые начинаются с большого латинского символа (от A до Z) и заканчиваются 2 или 4 цифрами, например *«Petr93», «Johnny70», «Service2002»*. Используйте регулярные выражения.
6. Напишите скрипт reorganize.py, который в директории --source создает две директории: Archive и Small. В первую директорию помещаются файлы с датой изменения, отличающейся от текущей даты на количество дней более параметра --days (т.е. относительно старые файлы). Во вторую – все файлы размером меньше параметра --size байт. Каждая директория должна создаваться только в случае, если найден хотя бы один файл, который должен быть в нее помещен. Пример вызова:

```
reorganize --source "C:\TestDir" --days 2 --size 4096
```

7. Написать скрипт trackmix.py, который формирует обзорный трек-микс альбома (попурри из коротких фрагментов mp3-файлов в пользовательской директории). Для манипуляций со звуковыми файлами можно использовать сторонние утилиты, например, FFmpeg.
Пример вызова и работы скрипта:

```
trackmix --source "C:\Muz\Album" --count 5 --frame 15 -l -e

  --- processing file 1: 01 - Intro.mp3
  --- processing file 2: 02 - Outro.mp3
  --- done!
```

Параметры скрипта:
--source (-s) – имя рабочей директории с треками, обязателен;
--destination (-d) – имя выходного файла, необязателен (если не указан, то имя выходного файла – mix.mp3 в директории source);
--count (-c) – количество файлов в "нарезке", необязателен (если он не указан, то перебираются все mp3-файлы в директории source);
--frame (-f) – количество секунд на каждый файл, необязателен (если не указан, скрипт вырезает по 10 секунд из произвольного участка каждого файла);
--log (-l) – необязательный ключ (если этот ключ указывается, то скрипт должен выводить на консоль лог процесса обработки файлов, как в примере);
--extended (-e) – необязательный ключ (если этот ключ указывается, то каждый фрагмент попурри начинается и завершается небольшим fade in/fade out).

Контрольные вопросы [ОТЧЕТ]:
1. Как в Python осуществляется чтение/запись текстовых файлов?
2. Как в Python можно получить информацию о содержимом директории?
3. Как в Python можно скопировать, удалить и переименовать файл?
4. Каковы базовые синтаксические элементы регулярных выражений?
5. Какие возможности предоставляет модуль subprocess?

Краткая теоретическая справка.
Чтение/запись файлов в Python осуществляется с помощью функции открытия файла open() и функций непосредственно чтения и записи данных. Пример кода:

```
s = ['Hello ', 'World\n', '(c) Admin\n']

# запись списка строк в текстовый файл demo.txt:
try:
    f = open('demo.txt', 'wt')
    f.writelines(s)
except IOError as err:
    print(err)
except:
    print('Something\'s gone wrong...')
else:
    print('File\'s been updated succesfully')
finally:
    if f:
        f.close()
```

Приведенный код можно сократить с помощью конструкции with, цель которой – предоставить более короткую запись кода вида:

```
# -- setup (начало работы с ресурсом) --
# try:
#     -- do_smth (действия с ресурсом) --
# finally:
#     -- tear down (корректное закрытие ресурса) --
```

Например:

```
# чтение строк из файла (с конструкцией with)
try:
    with open(r'data\demo.txt', 'rt') as f:
        lines = f.readlines()
except IOError as err:
    print(err)
    lines = []

# вывести строки из файла в "сыром" виде
print('Read from file:', lines)

# убрать \n на конце каждой строки с помощью rstrip и map
newlines = list(map(lambda x: x.rstrip(), lines))

# убрать \n на конце каждой строки
# с помощью rstrip и list comprehension
newlines = [x.rstrip() for x in lines]

print('After post-processing:', newlines)
```

Все необходимые функции для работы с файловой системой в Python содержатся в модулях os, shutil, pathlib, glob.
Пример кода для получения списка только папок в текущей директории:

```
folders = [entry for entry in os.listdir(os.getcwd())
                 if os.path.isdir(entry)]
print(folders)
```

Для вычисления контрольной суммы файла существует модуль hashlib. Пример кода с использованием этого модуля приведен ниже:

```
import hashlib

with open(r'data\somefile.txt', 'r') as f:
      content = f.read()

checksum = hashlib.md5(content).hexdigest()
```

Одним из распространенных способов сериализации/десериализации объектов в Python является использование функций модуля pickle. Например:

```
import pickle

data = (1, 3.15, 'Hello', [1, 4, 5])

# сериализация объекта в pkl-файл
pickle.dump(data, open(r'data\data.pkl', 'wb'))

# десериализация (загрузка объекта из pkl-файла)
obj = pickle.load(open(r'data\data.pkl', 'rb'))
print(obj)
```

Для работы с процессами операционной системы в настоящее время предпочтительнее использовать модуль subprocess. Ниже приведен код вызова внешнего процесса (ipconfig.exe) с синхронным получением выходного результата:

```
output = subprocess.check_output(['ipconfig', '/all'])
print(output)
```

Функция check_output() будет ожидать завершения вызванного процесса. Если необходимо отслеживать поток вывода порожденного процесса, можно в простейшем случае организовать цикл опроса потока. Например, утилита ping.exe выводит данные на консоль в определенные промежутки времени. Код скрипта, работающего с данной утилитой, выглядит так:

```
process = subprocess.Popen(['ping', '127.0.0.1'],
stdout=subprocess.PIPE)

# синхронный busy spin, работа с потоком вывода,
# пока процесс не отработает
while True:
    output = process.stdout.readline()

    # когда процесс завершится,
    # poll() вернет код завершения, а не None
    rc = process.poll()

    if rc is not None:
        break
    if output:
        print(output.strip())

print('The process has finished with return code {}\n'.
   format(rc))
```