Лабораторная работа 7.

Создание текстового файла большого объема

Компьютерная математика II, ММФ, БГУ

Лаврова О.А., идея Козак А.В., Кушнеров А.В., апрель 2024

Навыки

1. базовый **Python**: работа с файлами: `open` , методы файловых объектов `close` , `write` , `writelines` ; генераторные выражения и функции; `ord` , `chr` , `len` ; методы строковых объектов `join` , `encode`
2. модуль `time` из стандартной библиотеки модулей: `process_time`
3. модуль `random` из стандартной библиотеки модулей: `randint`

Постановка задачи

Напишите пользовательскую функцию

```
file_gen(file_name, file_size: int, type_symbols: str, number_words:
(int,int), number_symbols: (int,int))
```

для создания текстового файла заданного имени и размера, строки которого будут состоять из случайного количества слов случайной длины и составленных из случайных символов. Слова в строке разделены одним пробелом.

Значения аргументов:

- `file_name` определяет имя текстового файла;
- `file_size` определяет размер текстового файла **в мегабайтах**;
- `type_symbols` может принимать одно из трех значений: `'digits'`, `'latin'`, `'cyrillic'`:
  - если `type_symbols='digits'`, то для создания файла разрешены только цифровые символы;
  - если `type_symbols='latin'`, то для создания файла разрешены только **прописные** латинские символы от `a` до `z`; **стандартное значение** `type_symbols='latin'`;
  - если `type_symbols='cyrillic'`, то для создания файла разрешены только **прописные** кириллические символы;
- `number_words` является кортежем из двух целых чисел для задания количества слов в строке; количество слов в каждой строке файла определяется как случайное число из диапазона, определенного кортежем; **стандартное значение** `number_words=(10,50)`;
- `number_symbols` является кортежем из двух целых чисел для задания количества символов в слове; количество символов в каждом слове файла определяется как случайное число из диапазона, определенного кортежем; **стандартное значение** `number_symbols=(5,9)`.

Используем функцию randint и функцию process_time

Для выполнения заданий лабораторной работы будем использовать функцию генерации целых чисел `randint` из модуля `random` для создания:

- случайного количества слов в строке файла по заданному диапазону `number_words`,
- случайной длины слова в строке файл по заданному диапазону `number_symbols`,
- случайного символа в слове из коллекции `type_symbols`.

In [1]:
```
from random import randint
```

In [2]:
```
?randint
```

```
Signature: randint(a, b)
Docstring:
Return random integer in range [a, b], including both end points.

File:      c:\users\olga\anaconda3\lib\random.py
Type:      method
```

In [3]:
```
randint(0,100)
```

Out[3]:
```
45
```

Для выполнения заданий лабораторной работы будем использовать функцию `process_time` из модуля `time`, которая возвращает значение времени, затраченное процесором на выполнение кода

In [4]:
```
from time import process_time
```

In [5]:
```
?process_time
```

```
Docstring:
process_time() -> float

Process time for profiling: sum of the kernel and user-space CPU time.
Type:      builtin_function_or_method
```

In [6]:
```
start = process_time()
[x**2 for x in range(1_000_000)]
end = process_time()
print(f'{end - start} секунд')
```

```
0.15625 секунд
```

Задание 7.1. Запись данных в текстовый файл

Откроем файл с именем `test.txt` в текстовом режиме для записи

In [7]:
```
file_name = 'test.txt'
f = open(file_name, 'w')
```

Создадим список, состоящий из некоторых строковых объектов, каждый из которых заканчивается символом новой строки `\n`

In [8]:
```
lines_list = [f'{x}\n' for x in 'test']
lines_list
```

Out[8]:
```
['t\n', 'e\n', 's\n', 't\n']
```

Запишем созданные строки в файл с помощью метода `writelines` файлового объекта `f`

In [9]:
```
f.writelines(lines_list)
```

Вызовем метод `close` файлового объекта `f` для прекращения его связи с внешним файлом, освобождения системных ресурсов и сброса буферизованного вывода на диск, если он находится в памяти.

In [10]:
```
f.close()
```

Прочитаем записанные данные. Для этого откроем файл в режиме для чтения `'r'`

In [11]:
```
with open(file_name,'r') as f:
    for line in f:
        print(line, end='')
```

```
t
e
s
t
```

В случае записи в файл больших объемов данных целесообразнее использовать не списки для хранения строк, а генераторные объекты

In [12]:
```
lines_gen_expr = (f'{x}\n' for x in range(10**6))

def lines_gen_fun(number_lines=10**6):
    yield from (f'{x}\n' for x in range(number_lines))

with open(file_name,'w') as f:
    f.writelines(lines_gen_expr)
    f.writelines(lines_gen_fun())
```

В дальнейшем будем создавать данные для записи в файл с помощью генераторной функции.

Добавим в функцию `lines_gen_fun` подсчет длины записанной информации в мегабайтах. Полагаем, что длина одного символа равна одному байту

In [13]:
```
def lines_gen_fun(number_lines = 10**6):
    file_size = 0
    for x in range(number_lines):
        line = f'{x}\n'
        yield line
        file_size += len(line)
    print(f'{file_size/1024**2} Mb')

with open(file_name,'w') as f:
    f.writelines(lines_gen_fun())
```

```
6.569757461547852 Mb
```

Комментарий к коду. Допущение, что размер символа в строке равен одному байту, не всегда верно. В частности, для кириллических символов это не так. Более правильным решением для подсчета длины записанной информации в байтах будет использование метода `encode` для записываемых строк. В этом случае в коде выше вместо `len(line)` стоит вызывать `len(line.encode('utf8'))` .

In [14]:
```
len('f'), len('ц'), len('f'.encode('utf8')), len('ц'.encode('utf8'))
```

Out[14]:
```
(1, 1, 1, 2)
```

Измеряем время, необходимое для создания файла с помощью функции `lines_gen_fun`

In [15]:
```
with open(file_name,'w') as f:
    start = process_time()
    f.writelines(lines_gen_fun())
    end = process_time()
    print(end - start, 'секунд')
```

```
6.569757461547852 Mb
3.53125 секунд
```

Внесем изменения в генераторную функцию `lines_gen_fun` , чтобы она отображала процент записанных данных в файл, если аргумент `status=True` . Это существенно увеличит время выполнения функции!

In [16]:
```
def lines_gen_fun(number_lines=10**6, status=False):
    file_size = 0
    for x in range(number_lines):
        line = f'{x}\n'
        yield line
        file_size += len(line)
        # отображение статуса записи в файл в процентах
        if status:
            status_number = x/number_lines*10**2
            print(f'\r{int(status_number)}%', end='', flush=True) # \r возврат курс
    print(f'\n {file_size/1024**2} Mb')
```

Запишем данные в файл:

In [17]:
```
# предварительно вызовите в Anaconda Prompt
# jupyter lab --NotebookApp.iopub_data_rate_limit=1.0e10

with open(file_name,'w') as f:
    start = process_time()
    f.writelines(lines_gen_fun(3*10**3, status=True))
    end = process_time()
    print(f'{end - start} секунд')
```

```
99%
 0.013246536254882812 Mb
6.65625 секунд
```

\color{red}\text{Напишите} комментарии для каждой строки кода функции `lines_gen_fun` .

Задание 7.2. Генерация строк

Определим значения для переменных, которые задают параметры создания строки для записи в файл

In [18]:
```
symbols_ord = (ord('a'), ord('z')) # соответствует type_symbols = 'latin'
number_words = (10, 50)
number_symbols = (5, 9)
```

Создадим одно слово случайной длины из интервала `[5,9]`, состоящее из случайно выбранных латинских символов

In [19]:
```
len_word = randint(*number_symbols)
word = ''.join([chr(randint(*symbols_ord)) for i in range(len_word)])
word
```

Out[19]:
```
'gknsywvf'
```

1. Напишите код для генерации строки со случайным количеством слов, случайной длиной слова и случайным набором символов в слове для трех множеств смволов: прописные латинские символы, цифровые символы, прописные кириллические символы. Протестируйте написанный код.
2. Напишите генераторную функцию `lines_gen_fun_v2(file_size, type_symbols, number_words, number_symbols)` для генерации строк, суммарный размер которых в мегабайтах равен `file_size`.

1. \color{red}\text{Напишите} комментарии для каждой строки кода функции `lines_gen_fun_v2`.
2. \color{red}\text{Напишите} строки документации для функции `lines_gen_fun_v2`.
3. \color{red}\text{Протестируйте} функцию `lines_gen_fun_v2` для различных значений аргументов в предположении, что корректность вводимых данных гарантируется.

Задание 7.3. Запись сгенерированных строк в текстовый файл

1. Напишите результирующую функцию

   ```
   file_gen(file_name, file_size: int, type_symbols: str, number_words:
   (int,int), number_symbols: (int,int))
   ```

   Функция `file_gen` должна использовать функцию `lines_gen_fun_v2` из Задания 7.2.

Функция `file_gen` должна выводить на экран:

- процент записанных данных в процессе выполнения функции,
- фактический размер записанных данных в мегабайтах,
- время для выполнения кода функции.

1. \color{red}\text{Напишите} комментарии для каждой строки кода функции `file_gen` .
2. \color{red}\text{Напишите} строки документации для функции `file_gen` .
3. \color{red}\text{Протестируйте} функцию `file_gen` для различных значений аргументов в предположении, что корректность вводимых данных гарантируется.