

Московский государственный технический
университет им. Н.Э. Баумана

Факультет «Информатика и системы управления»
Кафедра ИУ5 «Системы обработки информации и управления»

Курс «Разработка интернет-приложений»

Отчет по лабораторной работе №3

Выполнил:

Студент группы ИУ5-53Б

Балабас Анна

Руководители: Гапанюк Ю.Е.

Дата: 24.10.21

Москва, 2021 г.

Задание:

Задание лабораторной работы состоит из решения нескольких задач.

Файлы, содержащие решения отдельных задач, должны располагаться в пакете `lab_python_fr`. Решение каждой задачи должно располагаться в отдельном файле.

При запуске каждого файла выдаются тестовые результаты выполнения соответствующего задания.

Задача 1 (файл `field.py`)

Необходимо реализовать генератор `field`. Генератор `field` последовательно выдает значения ключей словаря. Пример:

```
goods = [  
    {'title': 'Ковер', 'price': 2000, 'color': 'green'},  
    {'title': 'Диван для отдыха', 'color': 'black'}  
]
```

`field(goods, 'title')` должен выдавать 'Ковер', 'Диван для отдыха'

`field(goods, 'title', 'price')` должен выдавать `{'title': 'Ковер', 'price': 2000}`, `{'title': 'Диван для отдыха'}`

В качестве первого аргумента генератор принимает список словарей, дальше через `*args` генератор принимает неограниченное количество аргументов.

Если передан один аргумент, генератор последовательно выдает только значения полей, если значение поля равно `None`, то элемент пропускается.

Если передано несколько аргументов, то последовательно выдаются словари, содержащие данные элементы. Если поле равно None, то оно пропускается. Если все поля содержат значения None, то пропускается элемент целиком.

Шаблон для реализации генератора:

Пример:

```
# goods = [  
#   {'title': 'Ковер', 'price': 2000, 'color': 'green'},  
#   {'title': 'Диван для отдыха', 'price': 5300, 'color': 'black'}  
# ]  
  
# field(goods, 'title') должен выдавать 'Ковер', 'Диван для отдыха'  
# field(goods, 'title', 'price') должен выдавать {'title': 'Ковер', 'price': 2000},  
# {'title': 'Диван для отдыха', 'price': 5300}
```

```
def field(items, *args):
```

```
    assert len(args) > 0
```

```
    # Необходимо реализовать генератор
```

```
goods = [  
    {'title': 'Кресло', 'price': 5700, 'color': 'pink'},  
    {'title': 'Диван раскладной', 'price': 26700, 'color': 'cream'}] #создание списка  
с аргументами в виде словарей  
  
def field(items:list[dict], *args): # генератор field  качестве первого  
аргумента генератор принимает список словарей,  
# дальше через *args генератор принимает неограниченное количество  
аргументов  
    assert len(args) > 0 # если данное утверждение верно  
    for i in items: # перебираем весь список  
        if(len(args)==1): # если передан только один аргумент, то генератор  
выводит только значение поля  
            if(i.get(args[0])): #метод get() возвращает значение для данного ключа  
                yield i [args[0]]
```

```

else: #если передано несколько аргументов
    res={} # пустой словарь
    for a in args:
        if(i.get(a)):
            res[a]=i[a]
    if(len(res.items())!=0):
        yield res

field_gen=field(goods, 'title')
field_gen1=field(goods, 'title', 'price')
for i in field_gen:
    print(i)
for i in field_gen1:
    print(i)

"D:\Бомонка\3 лр\venv\Scripts\python.exe" "D:/Бомонка/3 лр/lab_python_fp/field.py"
Кресло
Диван раскладной
{'title': 'Кресло', 'price': 5700}
{'title': 'Диван раскладной', 'price': 26700}

Process finished with exit code 0

```

Задача 2 (файл gen_random.py)

Необходимо реализовать генератор gen_random(количество, минимум, максимум), который последовательно выдает заданное количество случайных чисел в заданном диапазоне от минимума до максимума, включая границы диапазона. Пример:

gen_random(5, 1, 3) должен выдать 5 случайных чисел в диапазоне от 1 до 3, например 2, 2, 3, 2, 1

Шаблон для реализации генератора:

Пример:

gen_random(5, 1, 3) должен выдать 5 случайных чисел

в диапазоне от 1 до 3, например 2, 2, 3, 2, 1

Hint: типовая реализация занимает 2 строки

```
def gen_random(num_count, begin, end):
```

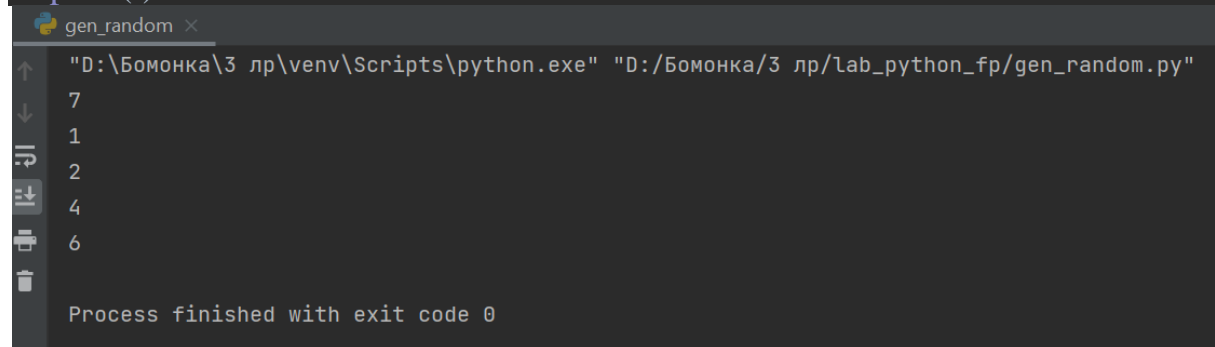
```
    pass
```

```
    # Необходимо реализовать генератор
```

```
import random
#gen_random(5, 1, 3) должен выдать 5 случайных чисел
# в диапазоне от 1 до 3, например 2, 2, 3, 2, 1

def gen_random(num_count, begin, end):
    for i in range(num_count):
        yield random.randint(begin, end)

gen_random1=gen_random(5,1,9)
for i in gen_random1:
    print(i)
```



```
gen_random x
"D:\Бомонка\3 лр\venv\Scripts\python.exe" "D:/Бомонка/3 лр/lab-python_fp/gen_random.py"
7
1
2
4
6
Process finished with exit code 0
```

Задача 3 (файл unique.py)

Необходимо реализовать итератор Unique(данные), который принимает на вход массив или генератор и итерируется по элементам, пропуская дубликаты.

Конструктор итератора также принимает на вход именованный bool-параметр ignore_case, в зависимости от значения которого будут считаться одинаковыми строки в разном регистре. По умолчанию этот параметр равен False.

При реализации необходимо использовать конструкцию ****kwargs**.

Итератор должен поддерживать работу как со списками, так и с генераторами.

Итератор не должен модифицировать возвращаемые значения.

Пример:

```
data = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]
```

Unique(data) будет последовательно возвращать только 1 и 2.

```
data = gen_random(1, 3, 10)
```

Unique(data) будет последовательно возвращать только 1, 2 и 3.

```
data = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']
```

Unique(data) будет последовательно возвращать только a, A, b, B.

Unique(data, ignore_case=True) будет последовательно возвращать только a, b.

Шаблон для реализации класса-итератора:

```
# Итератор для удаления дубликатов
```

```
class Unique(object):
```

```
    def __init__(self, items, **kwargs):
```

```
        # Нужно реализовать конструктор
```

```
        # В качестве ключевого аргумента, конструктор должен принимать  
        bool-параметр ignore_case,
```

```
        # в зависимости от значения которого будут считаться одинаковыми  
        строки в разном регистре
```

```
        # Например: ignore_case = False, Абв и АБВ - разные строки
```

```
# ignore_case = True, Абв и АБВ - одинаковые строки, одна из  
которых удалится
```

```
# По-умолчанию ignore_case = False
```

```
pass
```

```
def __next__(self):
```

```
# Нужно реализовать __next__
```

```
pass
```

```
def __iter__(self):
```

```
return self
```

```
# Итератор для удаления дубликатов
```

```
class Unique:
```

```
def __init__(self, items, **kwargs):
```

```
# Нужно реализовать конструктор
```

```
# В качестве ключевого аргумента, конструктор должен принимать  
bool-параметр ignore_case,
```

```
# в зависимости от значения которого будут считаться одинаковыми  
строки в разном регистре
```

```
# Например: ignore_case = False, Абв и АБВ - разные строки
```

```
# ignore_case = True, Абв и АБВ - одинаковые строки, одна из  
которых удалится
```

```
# По-умолчанию ignore_case = False
```

```
self.used_elements = set() # тут мы будем хранить значения, которые  
уже занесли для вывода как уникальные
```

```
self.data = items
```

```
self.index = 0
```

```
if 'ignore_case' not in kwargs: # ignore_case-ключевое значение
```

```
self.ignore_case=False
```

```
else:
```

```
self.ignore_case=kwargs['ignore_case']
```

```
def __next__(self):
```

```
while True:
```

```
if self.index >= len(self.data):
```

```
raise StopIteration
```

```
else:
```

```

        current = self.data[self.index]
        self.index = self.index + 1
        if self.ignore_case: # если False
            if current.lower() not in self.used_elements:
                self.used_elements.add(current.lower())
                return current
        else:
            if current not in self.used_elements:
                # Добавление в множество производится
                # с помощью метода add
                self.used_elements.add(current)
                return current

    def __iter__(self):
        return self

lst2 = [1,3,2,3,2,1,4,7,3,3]
for i in Unique(lst2):
    print(i)
print("-----")
data=['a','A','c','C','B','b','b']
for a in Unique(data,ignore_case=False):
    print(a)
print("-----")
for a in Unique(data,ignore_case=True):
    print(a)

```

```

unique x
"D:\Бомонка\3 лр\venv\Scripts\python.exe" "D:/Бомонка/3 лр/lab_python_fp/unique.py"
1
3
2
4
7
-----
a
A
c
C
B
b
-----
a
c
B

```


Задача 4 (файл sort.py)

Дан массив 1, содержащий положительные и отрицательные числа. Необходимо одной строкой кода вывести на экран массив 2, которые содержит значения массива 1, отсортированные по модулю в порядке убывания. Сортировку необходимо осуществлять с помощью функции sorted. Пример:

```
data = [4, -30, 30, 100, -100, 123, 1, 0, -1, -4]
```

```
Вывод: [123, 100, -100, -30, 30, 4, -4, 1, -1, 0]
```

Необходимо решить задачу двумя способами:

С использованием lambda-функции.

Без использования lambda-функции.

Шаблон реализации:

```
data = [4, -30, 100, -100, 123, 1, 0, -1, -4]
```

```
if __name__ == '__main__':
```

```
    result = ...
```

```
    print(result)
```

```
    result_with_lambda = ...
```

```
    print(result_with_lambda)
```

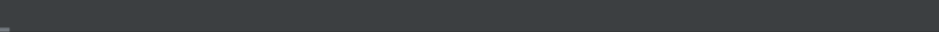
```
data = [4, -30, 100, -100, 123, 1, 0, -1, -4]
```

```
result=sorted(data,key=abs,reverse=True)
```

```
print(result)
```

```
result_with_lambda =sorted(data,key=lambda x: abs(x),reverse=True)
```

```
print(result_with_lambda)
```



```
sort x
"D:\Бомонка\3 лр\venv\Scripts\python.exe" "D:/Бомонка/3 лр/lab_python_fp/sort.py"
[123, 100, -100, -30, 4, -4, 1, -1, 0]
[123, 100, -100, -30, 4, -4, 1, -1, 0]
Process finished with exit code 0
```

```

@print_result
def test_2():
    return 'iu5'

@print_result
def test_3():
    return {'a': 1, 'b': 2}

@print_result
def test_4():
    return [1, 2]

print('!!!!!!!')
test_1()
test_2()
test_3()
test_4()

```

```

print_result x
"D:\Бомонка\3 лр\venv\Scripts\python.exe" "D:/Бомонка/3 лр/lab_python_fp/print_result.py"
!!!!!!!
test_1
1
test_2
iu5
test_3
a = 1
b = 2
test_4
1
2

Process finished with exit code 0

```

Задача 6 (файл cm_timer.py)

Необходимо написать контекстные менеджеры cm_timer_1 и cm_timer_2, которые считают время работы блока кода и выводят его на экран.

```

from contextlib import contextmanager
import time

class Cm_timer_1:
    def __init__(self, before_ms, after_ms):
        self.before_ms = before_ms
        self.after_ms = after_ms
    def __enter__(self):
        print(self.before_ms)

```

```

        self.time=time.time()
        return self.time
    def __exit__(self, exc_type, exc_val, exc_tb):
        if exc_type is not None:
            print(exc_type,exc_val,exc_tb)
        else:
            print("time1:",time.time()-self.time)
            print(self.after_ms)
before_ms = "Сообщение при входе в контекстный менеджер на основе
классса"
after_ms= "Сообщение при выходе из контекстного менеджера на основе
классса"
with Cm_timer_1(before_ms,after_ms) as cm_object:
    time.sleep(5.5)

@contextmanager
def cm_timer_2():
    t=time.time() # начальное время
    yield t
    print ("time2:",time.time()-t)# текущее время- начальное
with cm_timer_2():
    time.sleep(5.5)

```

```

cm_timer x
"D:\Бомонка\3 лр\venv\Scripts\python.exe" "D:/Бомонка/3 лр/lab_python_fp/cm_timer.py"
Сообщение при входе в контекстный менеджер на основе классса
time1: 5.514108657836914
Сообщение при выходе из контекстного менеджера на основе классса
time2: 5.511220455169678
Process finished with exit code 0

```

Задача 7 (файл process_data.py)

В предыдущих задачах были написаны все требуемые инструменты для работы с данными. Применим их на реальном примере.

В файле data_light.json содержится фрагмент списка вакансий.

Структура данных представляет собой список словарей с множеством полей: название работы, место, уровень зарплаты и т.д.

Необходимо реализовать 4 функции - f1, f2, f3, f4. Каждая функция вызывается, принимая на вход результат работы предыдущей. За счет декоратора `@print_result` печатается результат, а контекстный менеджер `cm_timer_1` выводит время работы цепочки функций.

Предполагается, что функции f1, f2, f3 будут реализованы в одну строку. В реализации функции f4 может быть до 3 строк.

Функция f1 должна вывести отсортированный список профессий без повторений (строки в разном регистре считать равными). Сортировка должна игнорировать регистр. Используйте наработки из предыдущих задач.

Функция f2 должна фильтровать входной массив и возвращать только те элементы, которые начинаются со слова “программист”. Для фильтрации используйте функцию `filter`.

Функция f3 должна модифицировать каждый элемент массива, добавив строку “с опытом Python” (все программисты должны быть знакомы с Python). Пример: Программист C# с опытом Python. Для модификации используйте функцию `map`.

Функция f4 должна сгенерировать для каждой специальности зарплату от 100 000 до 200 000 рублей и присоединить её к названию специальности. Пример: Программист C# с опытом Python, зарплата 137287 руб. Используйте `zip` для обработки пары специальность — зарплата.

```
import json
from gen_random import gen_random
import sys
from lab_python_fp.cm_timer import Cm_timer_1
from lab_python_fp.print_result import print_result
# Сделаем другие необходимые импорты

path = "data_light.json"
```

```

# Необходимо в переменную path сохранить путь к файлу, который был
передан при запуске сценария

with open(path) as f:
    data = json.load(f) # метод считывает файл в формате JSON и возвращает
объекты Python

# Далее необходимо реализовать все функции по заданию, заменив `raise
NotImplemented`
# Предполагается, что функции f1, f2, f3 будут реализованы в одну строку
# В реализации функции f4 может быть до 3 строк

@print_result
def f1(arg):
    return sorted(set([p.lower() for p in arg]),key=str.lower)

@print_result
def f2(arg):
    return list(filter(lambda x: str.startswith(x,'программист'),arg))

@print_result
def f3(arg):
    return list(map(lambda x:x +' с опытом Python',arg))

@print_result
def f4(arg):
    t=list(zip(arg,[" зарплата " + str(e1)+ " руб." for e1 in
list(gen_random(len(arg),100000,200000))]))
    return [e[0]+e[1] for e in t]

if __name__=='__main__':
    before_ms = "Сообщение при входе в контекстный менеджер на основе
класса"
    after_ms = "Сообщение при выходе из контекстного менеджера на основе
класса"
    with Cm_timer_1(before_ms,after_ms) as cm_object:
        f4(f3(f2(f1([el['job-name'] for el in data]))))

```

```
process_data x
↑ "D:\Бомонка\3 лр\venv\Scripts\python.exe" "D:/Бомонка/3 лр/lab_python_fp/process_data.py"
↓ 9
| 5
| 7
| 7
| 2
|
| Сообщение при входе в контекстный менеджер на основе классса
| time1: 5.507345914840698
| Сообщение при выходе из контекстного менеджера на основе классса
| time2: 5.502903461456299
| !!!!!!!
| test_1
| 1
| test_2
| iu5
| test_3
| a = 1
| b = 2
| test_4
| 1
| 2
| Сообщение при входе в контекстный менеджер на основе классса
| f1
| 1с программист
| 2-ой механик
| 3-ий механик
| 4-ый механик
| 4-ый электромеханик
| [химик-эксперт
| asic специалист
| javascript разработчик
```

[химик-эксперт
asic специалист
javascript разработчик
rtl специалист
web-программист
web-разработчик
автожестящик
автоинструктор
автомаляр
автомойщик
автор студенческих работ по различным дисциплинам
автослесарь
автослесарь - моторист
автоэлектрик
агент
агент банка
агент нпф
агент по гос. закупкам недвижимости
агент по недвижимости
агент по недвижимости (стажер)
агент по недвижимости / риэлтор
агент по привлечению юридических лиц