Министерство науки и высшего образования Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования «Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)» (МГТУ им. Н.Э. Баумана)

Факультет «Информатика и системы управления»

Кафедра ИУ5 «Системы обработки информации и управления»

Лабораторная работа №3

по дисциплине «Базовые компоненты интернет-технологий»

Выполнил:
студент группы ИУ5-32Б
Поддубный М.Н.

Проверил:
Канев А.И.

2021 г.

Описание задания

Задание лабораторной работы состоит из решения нескольких задач.

Файлы, содержащие решения отдельных задач, должны располагаться в пакете lab_python_fp. Решение каждой задачи должно раполагаться 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. В качестве первого аргумента генератор принимает список словарей, дальше через *args генератор принимает неограниченное количествово 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.

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`

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.

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.

Необходимо решить задачу двумя способами:

1. С использованием lambda-функции.
2. Без использования lambda-функции.

Задача 5 (файл print_result.py)

Необходимо реализовать декоратор print_result, который выводит на экран результат выполнения функции.

- Декоратор должен принимать на вход функцию, вызывать её, печатать в консоль имя функции и результат выполнения, после чего возвращать результат выполнения.
- Если функция вернула список (list), то значения элементов списка должны выводиться в столбик.
- Если функция вернула словарь (dict), то ключи и значения должны выводить в столбик через знак равенства.

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.
После завершения блока кода в консоль должно вывестись `time: 5.5` (реальное время может несколько отличаться).
`cm_timer_1` и `cm_timer_2` реализуют одинаковую функциональность, но должны быть реализованы двумя различными способами (на основе класса и с использованием библиотеки contextlib).

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.

Текст программы

field.py

```
# 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}

"""goods3 = [
   {'title': 'Ручка', 'color': 'blue', 'place': 'pencil box'},
   {'title': 'Карандаш', 'place': 'table'}
]"""


def field(items, *args):
    assert len(args) > 0
```

```
    # Необходимо реализовать генератор
    q = 0
    r = len(args)
    m = []
    for z in items:
        m.append(z.get(args[q]))
        while q + 1 < r:
            q = q+1
            m.append(z.get(args[q]))
        q = 0
    return m


print(*field(goods, 'title'))
print()
print(*field(goods, 'title', 'price'))
"""print()
print(*field(goods3, 'title', 'place'))"""
```

gen_random.py

```
# Пример:
# gen_random(5, 1, 3) должен выдать выдать 5 случайных чисел
# в диапазоне от 1 до 3, например 2, 2, 3, 2, 1
# Hint: типовая реализация занимает 2 строки
import random


def gen_random(num_count, begin, end):
    mas = []
    for i in range(0, num_count):
        mas.append(random.randint(begin, end))
    return mas

    # Необходимо реализовать генератор


print(gen_random(8, 1, 7))
```

unique.py

```
from lab3.gen_random import gen_random


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 = True, Абв и АБВ - разные строки
        #           ignore_case = False, Абв и АБВ - одинаковые строки, одна
из которых удалится
        # По-умолчанию ignore_case = False
        i = 0
        for k, v in kwargs.items():
            if v == True:
                for u in range(0, len(items)):
                    items[u] = items[u].lower()
                i = i + 1
        r = len(items)
```

```
        i = 0
        j = -1
        print(items[0], end=' ')
        while i < r:
            while j >= 0:
                if i > j:
                    if items[i] == items[j]:
                        break
                if j == 0:
                    print(items[i], end=' ')
                j = j - 1
            i = i + 1
            j = i - 1


    def __next__(self):
        # Нужно реализовать __next__
        num = self.num
        self.num += 1
        return num

    def __iter__(self):
        return self


data1 = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']

data2 = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]

data3 = gen_random(10, 1, 3)


print(data1)
Unique(data1)
print()
print(data1)
Unique(data1, ignore_case = True)
print()
print(data2)
Unique(data2)
print()
print(data3)
Unique(data3)
```

sort.py

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

if __name__ == '__main__':
    result = sorted(data, reverse=True, key=abs)
    print('Без lambda функции', result)

    result_with_lambda = sorted(data, key=lambda c: abs(c), reverse=True)
    print('С помощью lambda функции', result_with_lambda)
```

print_result.py

```
# Здесь должна быть реализация декоратора

def print_result(funct):
    def tt(*args, **kwargs):
        print(funct.__name__)
        if type(funct(*args, **kwargs)) == list:
            for value in funct(*args, **kwargs):
```

```
                if isinstance(value, tuple) == True:
                    print('{0}, зарплата {1} руб'.format(value[0], value[1]))
                else:
                    print(value)
        elif type(funct(*args, **kwargs)) == dict:
            for x in funct(*args, **kwargs):
                print(x, '=', funct()[x])
        else:
            print(funct(*args, **kwargs))
        return funct(*args, **kwargs)
    return tt


@print_result
def test_1():
    return 1


@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]


if __name__ == '__main__':
    print('!!!!!!!!!')
    test_1()
    test_2()
    test_3()
    test_4()
```

cm_timer.py

```
import time
from contextlib import contextmanager
from time import sleep


@contextmanager
def cm_timer_2():
    try:
        begin = time.perf_counter()
        yield begin
    finally:
        print(time.perf_counter() - begin)


class cm_timer_1():

    def start(self):
        self.begin = time.perf_counter()

    def stop(self):
        work = time.perf_counter() - self.begin
        print(work)
```

```
    def __enter__(self):
        self.start()

    def __exit__(self, *args):
        self.stop()


with cm_timer_1():
    sleep(0.5)


with cm_timer_2():
    sleep(0.5)
```

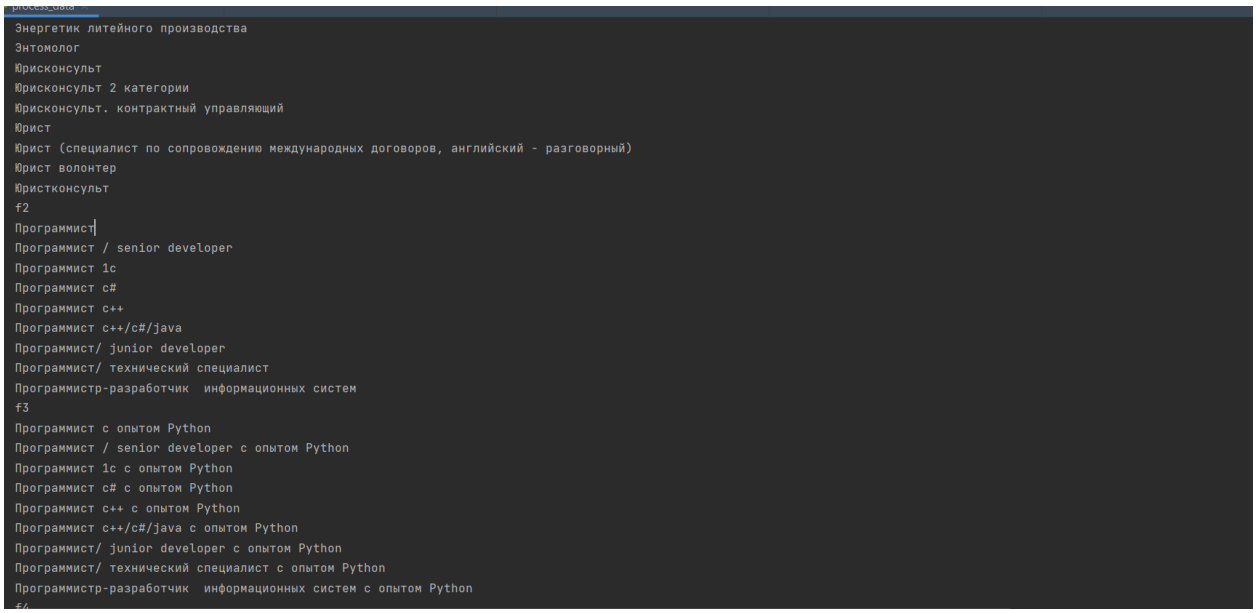
process_data.py

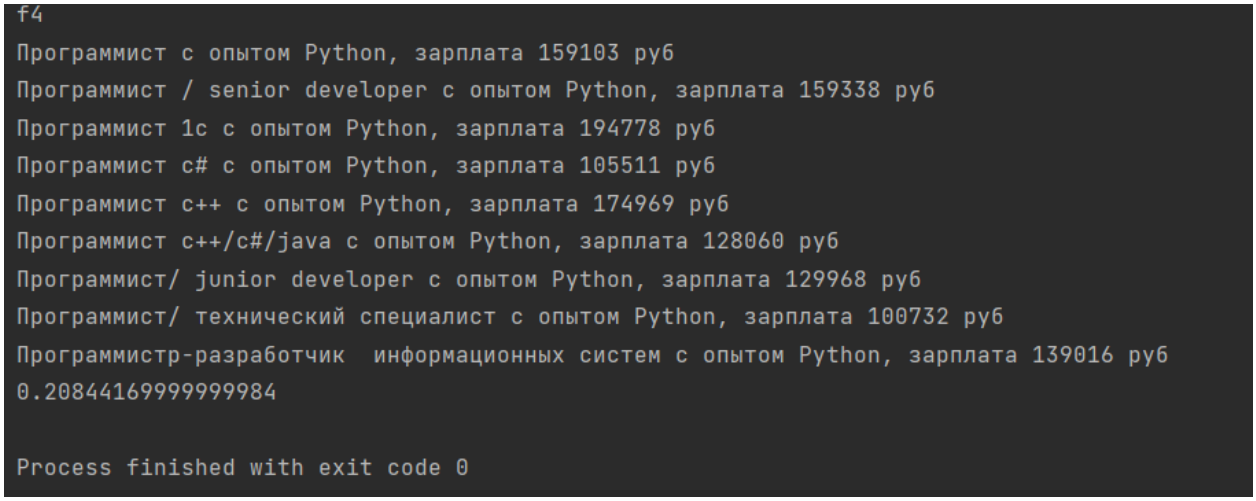
```
import json
# Сделаем другие необходимые импорты
from lab3.cm_timer import cm_timer_1
from lab3.field import field
from lab3.print_result import print_result
from lab3.gen_random import gen_random
from re import search

path = r'C:\Users\vldti\Desktop\data_light.json'

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

with open(path, encoding='utf-8') as f:
    data = json.load(f)

# 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


def Unique(items):
    for x in range(0, len(items)):
        items[x] = items[x].lower()
    temp = []
    temp2 = []
    for x in items:
        if x not in temp:
            temp.append(x)
    for x in temp:
        temp2.append(x.capitalize())
    return temp2


@print_result
def f1(arg):
    p1 = Unique(list(sorted(field(arg, 'job-name'), key=str.casefold)))
    return p1


@print_result
def f2(arg):
    p2 = list(filter(lambda k: search('^Программист', k)!= None, arg))
    return p2
```

```python
@print_result
def f3(arg):
    p3 = list(map(lambda k: k + ' с опытом Python', arg))
    return p3


@print_result
def f4(arg):
    p = len(list(filter(lambda k: search('^Программист', k)!= None, arg)))
    p4 = list(zip(arg, gen_random(p, 100000, 200000)))
    return p4

if __name__ == '__main__':
    with cm_timer_1():
        f4(f3(f2(f1(data))))
```

Экранные формы с результатами выполнения программы

```
f1
1с программист
2-ой механик
3-ий механик
4-ый механик
4-ый электромеханик
[химик-эксперт
Asic специалист
Javascript разработчик
Rtl специалист
Web-программист
Web-разработчик
Автожестянщик
Автоинструктор
Автомаляр
Автомойщик
Автор студенческих работ по различным дисциплинам
Автослесарь
Автослесарь - моторист
Автоэлектрик
Агент
Агент банка
```


process_data
Энергетик литейного производства
Энтомолог
Юрисконсульт
Юрисконсульт 2 категории
Юрисконсульт. контрактный управляющий
Юрист
Юрист (специалист по сопровождению международных договоров, английский - разговорный)
Юрист волонтер
Юристконсульт
f2
Программист
Программист / senior developer
Программист 1с
Программист с#
Программист с++
Программист с++/с#/java
Программист/ junior developer
Программист/ технический специалист
Программистр-разработчик информационных систем
f3
Программист с опытом Python
Программист / senior developer с опытом Python
Программист 1с с опытом Python
Программист с# с опытом Python
Программист с++ с опытом Python
Программист с++/с#/java с опытом Python
Программист/ junior developer с опытом Python
Программист/ технический специалист с опытом Python
Программистр-разработчик информационных систем с опытом Python
f4


f4
Программист с опытом Python, зарплата 159103 руб
Программист / senior developer с опытом Python, зарплата 159338 руб
Программист 1с с опытом Python, зарплата 194778 руб
Программист с# с опытом Python, зарплата 105511 руб
Программист с++ с опытом Python, зарплата 174969 руб
Программист с++/с#/java с опытом Python, зарплата 128060 руб
Программист/ junior developer с опытом Python, зарплата 129968 руб
Программист/ технический специалист с опытом Python, зарплата 100732 руб
Программистр-разработчик информационных систем с опытом Python, зарплата 139016 руб
0.20844169999999984
Process finished with exit code 0