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»

Курс «Базовые компоненты интернет-технологий»
Отчет по лабораторной работе №3

«Функциональные возможности языка Python»

Выполнил:
студентка группы ИУ5-33Б
Юрова Е.О.

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)

Описание задачи:

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 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.

Текст программы:

```python
def field(items, *args):
    try:
        assert len(args) > 0
    except AssertionError:
        print('Вы не указали ключи для вывода')
    if len(args) == 1:
        for arg in args:
            for i in range(len(items)):
                for j in items[i]:
                    if j == arg and items[i][j] is not None:
                        yield items[i][j]
    else:
        for i in range(len(items)):
            d = {}
            for arg in args:
                for j in items[i]:
                    if j == arg:
                        d[j] = items[i][j]
            yield d
goods = [
    {'title': 'Ковер','price': 2000, 'color': 'green'},
    {'title': 'Диван для отдыха','color': 'black'}
```

```
]

x = field(goods, 'title')
for i in range (len(goods)):
    if i != len(goods) - 1:
        print(next(x), end = ', ')
    else:
        print(next(x))

x = field(goods, 'title', 'price')
for i in range(len(goods)):
    if i != len(goods) - 1:
        print(next(x), end = ', ')
    else:
        print(next(x))
```

Экранные формы с примерами выполнения программы:

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

Задача 2 (файл gen_random.py)

Описание задачи:

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

Текст программы:

```
import random
def gen_random(count, begin, end):
    for x in range(count):
        print(random.randint(begin, end), end=' ')

gen_random(5,1,3)
```

Экранные формы с примерами выполнения программы:

```
2 1 1 3 2
Process finished with exit code 0
```

Задача 3 (файл unique.py)

Описание задачи:

- 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.

Текст программы:

```
class Unique(object):
    def __init__(self, items, ignore_case=False, **kwargs):
        self.seen = set()
        self.items = items
        self.ic = ignore_case
        self.kwargs = kwargs

    def __next__(self):
        it = iter(self.items)
        while True:
            try:
                current = next(it)
            except StopIteration:
                raise
            else:
                if self.ic == True and isinstance(current, str):
                    temp = current[:]
                    if temp.lower() not in self.seen:
                        self.seen.add(temp.lower())
                        return current
                elif current not in self.seen:
                    self.seen.add(current)
                    return current

    def __iter__(self):
        return self

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

un = Unique(data)
 for i in un:
     print(i, end = ' ')
 print()
```

Экранные формы с примерами выполнения программы:

```
A a k b B K R


Process finished with exit code 0
```

Задача 4 (файл sort.py)

Описание задачи:

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:

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

Текст программы:

```
import lab_python_fp.unique
import lab_python_fp.field

def sort(data):
    ndata = []
    ndata = lab_python_fp.field.field(data, 'job-name')
    ndata = list(lab_python_fp.unique.Unique(ndata, True))
    ndata = sorted(ndata, reverse=True)
    # for i in ndata:
    #     print(i, end=', ')
    # print()
    return list(ndata)


# data = [4, -30, 30, 100, -100, 123, 1, 0, -1, -4]
# result = sorted(data, key = abs, reverse=True)
# print(result)
# resultLambda = sorted(data, key = lambda x: x if x > 0 else -x,
reverse=True)
# print(resultLambda)
```

Экранные формы с примерами выполнения программы:

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

Process finished with exit code 0
```

Задача 5 (файл print_result.py)

Описание задачи:

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.

Текст программы:

```
import lab_python_fp.sort

def print_result(func):
    def f(data):
        print('Имя функции {}'.format(func.__name__))
        print('Результат функции:')
        if type(func(data)) == list:
            for i in func(data):
                print(i)
        elif type(func(data)) == dict:
            for key, value in func(data).items():
                print(key, ' = ', value)
        else:
            print(func(data))
        return func(data)
    return f


@print_result
def test_1():
    return 1


@print_result
def test_2():
    return 1
```

```python
@print_result
def test_3():
    return 1


@print_result
def test_4():
    return 1

test_1()
test_2()
test_3()
test_4()
```

Экранные формы с примерами выполнения программы:

```
Имя функции test_1
Результат функции:
1
Имя функции test_2
Результат функции:
iu5
Имя функции test_3
Результат функции:
a  =  1
b  =  2
Имя функции test_4
Результат функции:
1
2

Process finished with exit code 0
```

Задача 6 (файл cm_timer.py)

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

Текст программы:

```
import time
import contextlib


class cm_timer_1:
    def __enter__(self):
        self.start_time = time.time()

    def __exit__(self, exc_type, exc_val, exc_tb):
        print('time: ', time.time() - self.start_time)


@contextlib.contextmanager
def cm_timer_2():
    start_time = time.time()
    yield
    print('time: ', time.time() - start_time)


with cm_timer_1():
    time.sleep(0)

with cm_timer_2():
     time.sleep(0)
```

Экранные формы с примерами выполнения программы:

```
time:  3.361701965332031e-05
time:  2.86102294921875e-06
```

Задача 7 (файл process_data.py)

Описание задачи:

- 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.

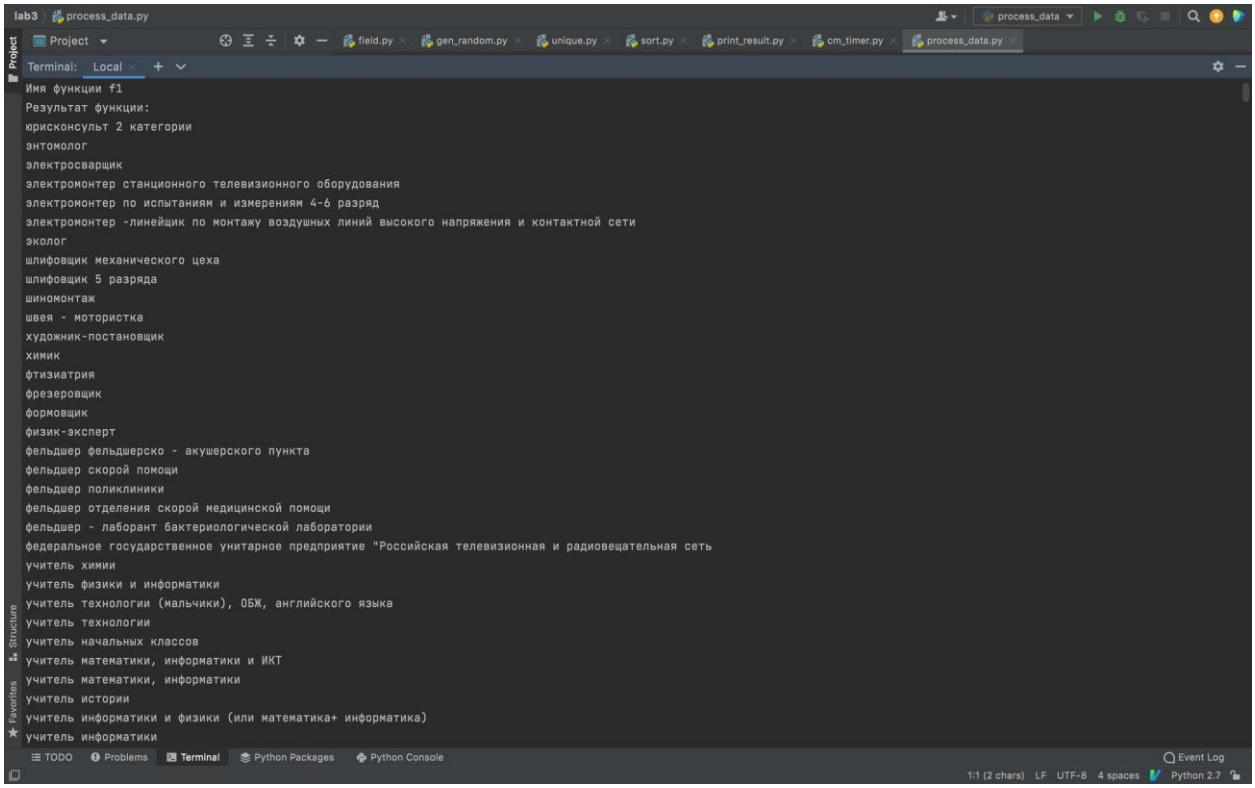
- Функция f4 должна сгенерировать для каждой специальности зарплату от 100 000 до 200 000 рублей и присоединить её к названию специальности. Пример: Программист C# с опытом Python, зарплата 137287 руб. Используйте zip для обработки пары специальность — зарплата.

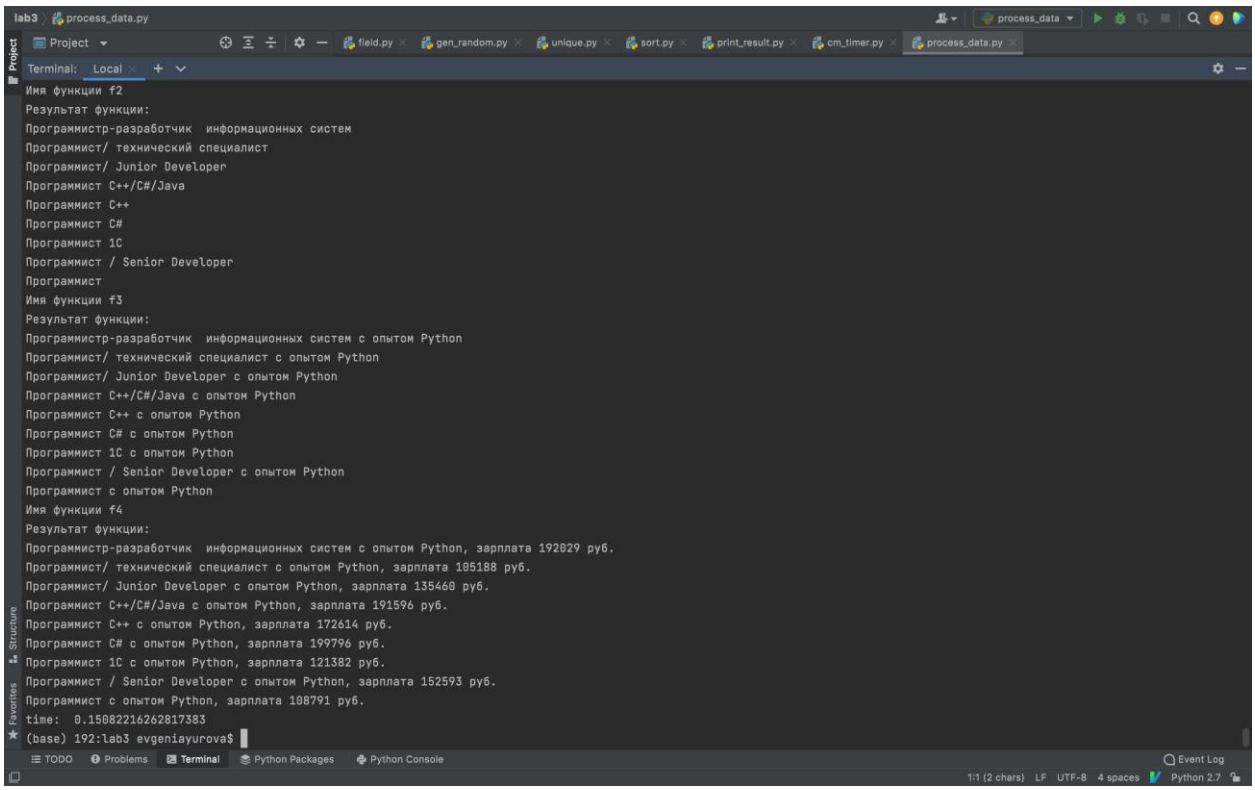
Текст программы:

```
import lab_python_fp.sort
import lab_python_fp.unique
import lab_python_fp.cm_timer
import lab_python_fp.print_result
import lab_python_fp.gen_random
import lab_python_fp.field
import json
import sys
import os


path = r'/Users/evgeniayurova/Desktop/data_light.json'
with open(path, encoding='utf-8') as f:
    data = json.load(f)

@lab_python_fp.print_result.print_result
def f1(arg):
    return lab_python_fp.sort.sort(arg)


@lab_python_fp.print_result.print_result
def f2(arg):
    return list(filter(filtration, arg))


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


@lab_python_fp.print_result.print_result
def f4(arg):
    salary = [i for i in lab_python_fp.gen_random.gen_random(len(arg),
100000, 200000)]
    return ['{}, зарплата {} руб.'.format(job, salary) for job, salary in
zip(arg, salary)]


def filtration(arg):
    if arg[0:11] == 'Программист':
        return True
    else:
```

```
        return False



with lab_python_fp.cm_timer.cm_timer_1():
    f4(f3(f2(f1(data))))
```

Экранные формы с примерами выполнения программы:




lab3 process_data.py
process_data
Project
field.py
gen_random.py
unique.py
sort.py
print_result.py
cm_timer.py
process_data.py
Terminal: Local
Имя функции f2
Результат функции:
Программистр-разработчик информационных систем
Программист/ технический специалист
Программист/ Junior Developer
Программист C++/C#/Java
Программист C++
Программист C#
Программист 1C
Программист / Senior Developer
Программист
Имя функции f3
Результат функции:
Программистр-разработчик информационных систем с опытом Python
Программист/ технический специалист с опытом Python
Программист/ Junior Developer с опытом Python
Программист C++/C#/Java с опытом Python
Программист C++ с опытом Python
Программист C# с опытом Python
Программист 1C с опытом Python
Программист / Senior Developer с опытом Python
Программист с опытом Python
Имя функции f4
Результат функции:
Программистр-разработчик информационных систем с опытом Python, зарплата 192029 руб.
Программист/ технический специалист с опытом Python, зарплата 105188 руб.
Программист/ Junior Developer с опытом Python, зарплата 135468 руб.
Программист C++/C#/Java с опытом Python, зарплата 191596 руб.
Программист C++ с опытом Python, зарплата 172614 руб.
Программист C# с опытом Python, зарплата 199796 руб.
Программист 1C с опытом Python, зарплата 121382 руб.
Программист / Senior Developer с опытом Python, зарплата 152593 руб.
Программист с опытом Python, зарплата 108791 руб.
time: 0.15082216262817383
(base) 192:lab3 evgeniayurova$
TODO
Problems
Terminal
Python Packages
Python Console
Event Log
1:1 (2 chars) LF UTF-8 4 spaces Python 2.7