Задание

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. 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.

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.

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. Пример:

```
with cm_timer_1():
    sleep(5.5)
```

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.

Код программы.

field.py

```
def field(goods, *args):
    ans = [{} for i in range(len(goods))]
    i=0; j=0
    for el in goods:
        for key in args:
            if key==None: continue
            if key in el.keys(): ans[i][key] = el[key]
        i+=1
    while j<i:
        if len(ans[j])==0:
            ans.pop(j); i-=1
        else: j+=1
    if len(args)==1:
        return [el[args[0]] for el in ans]
    return ans
```

gen_random.py

```
def gen_random(num_count, begin, end):
    p=4765356337; q=39857473; r=408369587
    ans=[]; seed = num_count**begin+end
    while num_count>0:
        seed = (seed*p + q) % r
        ans.append(begin + seed%(end-begin+1))
        num_count-=1
    return ans
```

unque.py

```
class Unique(object):
    def __init__(self, items, **kwargs):
        ignore_case = kwargs["ignore_case"] if "ignore_case" in kwargs.keys()
else False
        itm = {}
        if ignore_case:
            for el in items:
                if str(el).lower() not in itm.keys(): itm[str(el).lower()] =
el
        else:
            for el in items:
                if str(el) not in itm.keys():itm[str(el)] = el
        self.data = [itm[a] for a in itm]
        self.val = 0
        self.size = len(self.data)

    def __next__(self):
```

```
        if self.val + 1 < self.size:
            self.val += 1
            return self.data[self.val]
        else:
            raise StopIteration

    def __iter__(self):
        self.val = 0
        return self
```

sort.py

```
def sort(data):
    return sorted(data, key=lambda a: -a)
```

print_result.py

```
from .rectangle import Rectangle
def print_result(func):
    def wrapper(*args, **kwargs):
        out = func(*args, **kwargs)
        print(func.__name__)
        if type(out) == type([]): print(*out, sep="\n")
        elif type(out) == type({}): print(*[f"{el} = {out[el]}" for el in
out], sep="\n")
        else: print(out, sep="\n")
        return out
    return wrapper


@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]
```

cm_timer.py

```
import time
from contextlib import contextmanager

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

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

class cm_timer_2:
    @contextmanager
    def maneger(self):
        start_time = time.time()
```

```
        yield
        elapsed_time = time.time() - start_time
        print(elapsed_time)
```

process_data.py

```
from lab_python_fp.field import field
from lab_python_fp.gen_random import gen_random
from lab_python_fp.unique import Unique
from lab_python_fp.print_result import *

path = "lab_python_fp/data_light.json"

@print_result
def f1(arg):
    return sorted(Unique(field(arg, "job-name"), ignore_case=True))

@print_result
def f2(arg):
    return list(filter(lambda x: "программист" in x.lower()[0:12], arg))

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

@print_result
def f4(arg):
    return list(zip(arg, gen_random(len(arg), 100000, 200000)))x
```

main.py

```
import json
from lab_python_fp.sort import sort
from lab_python_fp.cm_timer import *
from lab_python_fp.process_data import *

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

if __name__ == '__main__':
    goods = [
        {'title': 'Ковер', 'price': 2000, 'color': 'green'},
        {'title': 'Диван для отдыха', 'color': 'black'}
    ]
    print("field.py")
    print(field(goods, 'title'))
    print(field(goods, 'title', 'price'))
    print()

    print("gen_random.py")
    print(*gen_random(5, 1, 3))
    print()

    print("unique.py")
    out = gen_random(50, 1, 30)
    print("out:", *out)
    ls = Unique(out, ignore_case=1)
    for el in ls: print(el, end=" ")
    print("\n-")
    for el in ls: print(el, end=" ")
    print("\n")

    print("sort.py")
    result = sort(data)
    print("lambda ", result)
```

```
    result_with_lambda = sorted(data, reverse=True)
    print("reverse", result_with_lambda)
    print()

    print("print_result.py")
    print("test_1 res", test_1())
    print("test_2 res", test_2())
    print("test_3 res", test_3())
    print("test_4 res", test_4())
    print()

    print("cm_timer.py")
    with cm_timer_1():
        time.sleep(1.5)

    with cm_timer_2().maneger():
        time.sleep(2.5)
    print()

    print("process_data.py")
    with open(path) as f:
        data = json.load(f)
    with cm_timer_1():
        f4(f3(f2(f1(data))))
```

Результаты.

```
field.py
['Ковер', 'Диван для отдыха']
[{'title': 'Ковер', 'price': 2000}, {'title': 'Диван для отдыха'}]

gen_random.py
1 3 2 1 1

unique.py
out: 13 26 28 6 16 20 29 30 11 24 21 27 9 13 6 29 7 29 21 28 30 25 8 13 22 22 24 8 11 24 29 4 3 20 14 17 26 30 13 26 7 24 2 21 28 12 11 24 1 13
26 28 6 16 20 29 30 11 24 21 27 9 7 25 8 22 4 3 14 17 2 12 1
-
26 28 6 16 20 29 30 11 24 21 27 9 7 25 8 22 4 3 14 17 2 12 1

sort.py
lambda  [123, 100, 4, 1, 0, -1, -4, -30, -100]
reverse [123, 100, 4, 1, 0, -1, -4, -30, -100]
```

```
print_result.py
----------
test_1
1
----------
test_1 res 1
----------
test_2
iu5
----------
test_2 res iu5
----------
test_3
a = 1
b = 2
----------
test_3 res {'a': 1, 'b': 2}
----------
test_4
1
2
----------
test_4 res [1, 2]

cm_timer.py
1.508753776550293
2.5099451541900635
```

```
----------
f2
Программист
Программист / Senior Developer
Программист 1С
Программист C#
Программист C++
Программист C++/C#/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 C# с опытом Python
Программист C++ с опытом Python
Программист C++/C#/Java с опытом Python
Программист/ Junior Developer с опытом Python
Программист/ технический специалист с опытом Python
Программистр-разработчик  информационных систем с опытом Python
----------
```

```
----------
f4
('Программист с опытом Python', 137067)
('Программист / Senior Developer с опытом Python', 102430)
('Программист 1С с опытом Python', 125735)
('Программист C# с опытом Python', 108182)
('Программист C++ с опытом Python', 184983)
('Программист C++/C#/Java с опытом Python', 198808)
('Программист/ Junior Developer с опытом Python', 143755)
('Программист/ технический специалист с опытом Python', 168679)
('Программистр-разработчик  информационных систем с опытом Python', 165440)
----------
0.058664798736572266
```