Федеральное государственное бюджетное образовательное учреждениевысшего образования «Московский государственный технический университетимени Н.Э. Баумана (национальный исследовательский университет)» (МГТУ им. Н.Э. Баумана)

Факультет «Информатика и системы управления»

Кафедра «Системы обработки информации и управления»

Лабораторная работа №3-4 по курсу «Базовые компоненты интернет технологий»

Выполнила:

студент группы № ИУ5-33Б

Балюк А.В

Проверил:

Преподаватель

Гапнюк Ю.Е

2022 г.

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.

- 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:

```
def field(items, *args):
    size = len(args)
    assert size > 0
    buff = {}
    for item in items:
        flag=True
        count = 0
        for arg in args:
           count +=1
           if item.get(arg) is not None:
               buff[arg] = item[arg]
               flag = False
               if size == 1:
                    yield buff[arg]
               elif count==size:
                    yield buff
                    buff.clear()
           elif count==size and flag==False:
               yield buff
               buff.clear()

def main():
    goods = [
        {'title': None, 'price': None, 'color': None},
        {'title': 'Диван для отдыха', 'price': 5300, 'color': 'black'},
        {'name': 'Skip me', 'addText': 'please'},
        {'title': 'Окно', 'color': 'white'},
        {'title': 'Шторы', 'price': int(1e9), 'color': '', 'name': '',
'addText': 'из будущего'}
    ]
    test = field(goods,'title')
    print("-----One field------")
    for i in test:
        print(i)
    print("-----Two fields------")
    test = field(goods, 'title','price')
    for i in test:
```

```
            print(i)
    print("-----Many fields------")
    test = field(goods, 'title', 'price', 'color', 'addText', 'name')
    for i in test:
        print(i)

if __name__ == '__main__':
    main()
```

Пример выполнения:

```
-----One field------
Диван для отдыха
Окно
Шторы
-----Two fields------
{'title': 'Диван для отдыха', 'price': 5300}
{'title': 'Окно'}
{'title': 'Шторы', 'price': 1000000000}
-----Many fields------
{'title': 'Диван для отдыха', 'price': 5300, 'color': 'black'}
{'addText': 'please', 'name': 'Skip me'}
{'title': 'Окно', 'color': 'white'}
{'title': 'Шторы', 'price': 1000000000, 'color': '', 'addText': 'из будущего', 'name': ''}
```

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

Текст программы:

```
from random import randint

def gen_random(num_count, begin, end):
    for i in range(num_count):
        yield randint(begin,end)
def main():
    print(*(gen_random(5,1,3)),sep=', ')

if __name__ == '__main__':
    main()
```

Пример выполнения:

```
2, 1, 3, 3, 1
3, 3, 3, 1, 3
```

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.

Текст программы:

```
from lab_python_fp.gen_random import gen_random

class Unique(object):
    def __init__(self, items, **kwargs):
        self.ignore_case = kwargs.get('ignore_case',False)
        self.iter = set(items)
        self.used = set()

    def __next__(self):
        count=0
        for i in self.iter:
            if i not in self.used:
                if isinstance(i, str):
                    if not self.ignore_case:
                        self.used.add(i)
                        return i
                    elif i.lower() not in self.used:
                        self.used.add(i.lower())
                        return i.lower()
                    else:
                        count+=1
                else:
                    self.used.add(i)
                    return i
            if len(set(self.used)) == len(set(self.iter)):
                raise StopIteration
            if self.ignore_case==True and
len(set(self.used))==(len(set(self.iter))-count):
                raise StopIteration

    def __iter__(self):
        return self

def main():
    data = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]
    for i in Unique(data):
        print(i, end=" ")
    print()

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

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

    data = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']
    for i in Unique(data):
        print(i, end=" ")
    print()

    data = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'b']
    for i in Unique(data, ignore_case=True):
        print(i, end=" ")
    print()


if __name__ == '__main__':
    main()
```

Пример выполнения:

```
1 2
1 2 3
b a A B
b a
```

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.

Текст программы:

```
def main():
    data = [4, -30, 30, 100, -100, 123, 1, 0, -1, -4]
    print(sorted(data, key=(lambda x: abs(x)), reverse=True))

    data = [4, -30, 30, 100, -100, 123, 1, 0, -1, -4]
    print(sorted(data,key = abs,reverse=True))

if __name__ == '__main__':
    main()
```

Пример выполнения:

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

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.

Текст программы:

```
def print_result(func):
    def _wrapper(*args, **kwargs):
        print(func.__name__)
        result = func(*args, **kwargs)
        if isinstance(result,list):
            for i in result:
                print(i)
        elif isinstance(result,dict):
            for i in result.keys():
                print(i ,'=', result[i])
        else:
            print(result)
        return result
    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]

def main():
    test_1()
    test_2()
    test_3()
    test_4()
if __name__ == '__main__':
    main()
```

Пример выполнения:

```
test_1
1
test_2
iu5
test_3
a = 1
b = 2
test_4
1
2
```

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).

Текст программы:

```
from contextlib import contextmanager
from time import sleep,perf_counter

class cm_timer_1:
    def __init__(self):
        self.start = 0
        self.end = 0

    def __enter__(self):
        self.start = perf_counter()

    def __exit__(self, exc_type, exc_val, exc_tb):
        self.end = perf_counter()
        print('Time = {:.4f}'.format(self.end-self.start))

@contextmanager
def cm_timer_2():
    start = perf_counter()
    yield
    end = perf_counter()
    print('Time = {:.4f}'.format(end-start))

def main():
    with cm_timer_1():
```

```
        sleep(5.5)
    with cm_timer_2():
        sleep(5.5)


if __name__ == '__main__':
    main()
```

Пример выполнения:

```
Time = 5.4997
Time = 5.5001
```

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:

```
import json
import sys
from lab_python_fp.gen_random import gen_random
from lab_python_fp.unique import Unique
from lab_python_fp.print_result import print_result
from lab_python_fp.cm_timer import cm_timer_1
from lab_python_fp.field import field

@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: x[:11].lower() == 'программист', arg))

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

@print_result
def f4(arg):
    return [job+', зарплата '+str(salary)+' руб.' for job,salary in
zip(arg,list(gen_random(len(arg),100000,200000)))]

def main():
    path = r'C:\Users\Андрей\Desktop\lab3\data_light.json'
    with open(path, encoding='UTF-8') as f:
        data = json.load(f)

    with cm_timer_1():
        f4(f3(f2(f1(data))))


if __name__ == '__main__':
    main()
```

Пример выполнения(только f2,f3,f4):

```
f2
программист / senior developer
программист 1с
программист c#
программист c++
программист/ junior developer
программист/ технический специалист
программистр-разработчик  информационных систем
f3
программист / senior developer с опытом Python
программист 1с с опытом Python
программист c# с опытом Python
программист c++ с опытом Python
программист/ junior developer с опытом Python
программист/ технический специалист с опытом Python
программистр-разработчик  информационных систем с опытом Python
f4
программист / senior developer с опытом Python, зарплата 115468 руб.
программист 1с с опытом Python, зарплата 133643 руб.
программист c# с опытом Python, зарплата 128327 руб.
программист c++ с опытом Python, зарплата 186686 руб.
программист/ junior developer с опытом Python, зарплата 121276 руб.
программист/ технический специалист с опытом Python, зарплата 141471 руб.
программистр-разработчик  информационных систем с опытом Python, зарплата 122328 руб.
Time = 116.8860
```