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»

Выполнила:
студентка группы ИУ5-33Б
Румак Д.П.

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 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.

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):
    assert len(args) > 0
    if len(args) == 1:
        for dictionary in items:
            note = dictionary.get(args[0])
            if note is not None:
                yield note
    else:
        for _dictionary in items:
            d = dict()
            for arg in args:
                note = _dictionary.get(arg)
                if note is not None:
                    d[arg] = note
            if len(d) != 0:
                yield d

if __name__ == '__main__':
    goods = [
        {'title': 'Ковер', 'price': 2000, 'color': 'green'},
        {'title': 'Диван для отдыха', 'price': 5300, 'color': 'black'},
        {'title': None, 'price': 100, 'color': 'black'},
        {'title': 'Кровать', 'price': 15000, 'color': 'yellow'}
    ]
    data1 = list()
    data2 = list()
```

```
    for i in field(goods, 'title'):
        data1.append(i)
    print(data1)

    for i in field(goods, 'title', 'price'):
        data2.append(i)
    print(data2)
```

Экранные формы с примерами выполнения программ

```
C:\Users\User\AppData\Local\Programs\Python\Python39\python.exe C:/Users/User/PycharmProjects/lab3/lab_python_fp/field.py
['Ковер', 'Диван для отдыха', 'Кровать']
[{'title': 'Ковер', 'price': 2000}, {'title': 'Диван для отдыха', 'price': 5300}, {'price': 100}, {'title': 'Кровать', 'price': 15000}]
```

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.

Текст программы

```
import random

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


if __name__ == '__main__':
    data = list()
    for i in gen_random(5, 1, 3):
        data.append(i)
    print(data)
```

Экранные формы с примерами выполнения программ

```
[2, 1, 1, 3, 2]
```

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 gen_random import gen_random

class Unique(object):
    def __init__(self, items, **kwargs):
        self.elements = set()   #пустое множество
        self.data = items
        self.ignore_case = False
        if len(kwargs) > 0:
            self.ignore_case = kwargs['ignore_case']

    def __next__(self):
        it = iter(self.data)
        while True:
            try:
                current = next(it)
            except StopIteration:
                raise StopIteration
            else:
                if self.ignore_case is False and isinstance(current, str):
                    current = current.lower()
                if current not in self.elements:
                    self.elements.add(current)
                    return current

    def __iter__(self):
        return self


if __name__ == '__main__':
    data1 = [1, 1, 3, 1, 1, 2, 2, 2, 2, 2]
    data2 = gen_random(5, 1, 10)
    data3 = ['A', 'a', 'b', 'B', 'a', 'A', 'b', 'B']

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

Экранные формы с примерами выполнения программ

```
[1, 3, 2]
[4, 6, 10, 8]
['A', 'a', 'b', 'B']
['a', '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.

Необходимо решить задачу двумя способами:

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

Текст программы

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

if __name__ == '__main__':
    result = sorted(data, reverse=True, key=abs)
    print("without lambda", result)

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

Экранные формы с примерами выполнения программ

```
without lambda [123, 100, -100, -30, 4, -4, 1, -1, 0]
with lambda [123, 100, -100, -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 decor(*args, **kwargs):
        result = func(*args, **kwargs)
        print(func.__name__)
        if isinstance(result, list):
            for i in result:
                print(i)
        elif isinstance(result, dict):
            for i in result:
```

```
                print(str(i) + " = " + str(result[i]))
        else:
            print(result)
        return result
    return decor


@print_result
def test_1():
    return 1


@print_result
def test_2():
    return 'iu5'


@print_result
def test_3():
    return {'a': 11, 'b': 82}


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


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

Экранные формы с примерами выполнения программ

```
test_1
1
test_2
iu5
test_3
a = 11
b = 82
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.

`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
import time

@contextmanager
def cm_timer_1():
    start = time.time()
    yield
    print("Время работы блока кода: {} секунд".format(time.time() - start))


class cm_timer_2:

    def __init__(self):
        self.start = time.time()

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

    def __exit__(self, ecx_type, exc_value, traceback):
        print("Время работы блока кода: {} секунд".format(time.time() -
self.start))


with cm_timer_1():
    time.sleep(3)
with cm_timer_2():
    time.sleep(3)
```

Экранные формы с примерами выполнения программ

```
Время работы блока кода: 3.0005595684051514 секунд
Время работы блока кода: 3.0005619525909424 секунд
```

Задача 7 (файл process_data.py)

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 print_result import print_result
from cm_timer import cm_timer_1
from unique import Unique
from field import field
from gen_random import gen_random


path = r'C:\Users\User\Downloads\data_light.json'
# Необходимо в переменную path сохранить путь к файлу, который был передан
при запуске сценария

with open(path, encoding='utf-8') as file:
    data = json.load(file)
# 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(list(Unique(field(arg, 'job-name'), ignore_case=True)),
key=str.lower)


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


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

```
@print_result
def f4(arg):
    return dict(zip(arg, list('зарплата {} руб.'.format(val) for val in
gen_random(len(arg), 10000, 200000))))


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

Экранные формы с примерами выполнения программ

```
f1
1С программист
2-ой механик
3-ий механик
4-ый механик
4-ый электромеханик
[химик-эксперт
ASIC специалист
```

… и еще очень много профессий

```
f2
программист
программист 1С
f3
программист с опытом Python
программист 1С с опытом Python
f4
программист с опытом Python = зарплата 160892 руб.
программист 1С с опытом Python = зарплата 101846 руб.
Время работы блока кода: 0.05901503562927246 секунд
```