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 ИУ5-33Б
Некрасов С. А.

Проверил:
Канев А.И.

2021 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 выдаёт 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  # если есть аргументы
    if len(args) == 1:  # если аргумент один
        for vocabulary in items:
            text = vocabulary.get(args[0])  # получаем значение ключа
            if text is not None:  # если значение не нулевое
                yield text  # yield возвращает генератор text
    else:  # если аргументов несколько
        for j in items:
            vocabulary = dict()  # создали словарь с помощью dict
            for key in args:  # перебираем ключи аргументов
                text = j.get(key)  # получаем значение ключа
                if text is not None:  # если значение не нулевое
                    vocabulary[key] = text  # добавили значение к ключу
            if len(vocabulary) != 0:  # если vocabulary имеет хотя бы одно
значение
                yield vocabulary  # yield возвращает генератор vocabulary


if __name__ == '__main__':
    goods = [
        {'title': 'Ковер', 'price': 2000, 'color': 'green'},
        {'title': 'Диван для отдыха', 'price': 5300, 'color': None},
        {'title': 'Стол', 'price': None, 'color': 'white'},
        {'title': None, 'price': None, 'color': 'pink'}
```

```
    ]
    data1 = list()  # объявили пременную как список
    data2 = list()  # объявили пременную как список

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

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

Результат выполнения:

```
['Ковер', 'Диван для отдыха', 'Стол']
[{'title': 'Ковер', 'price': 2000}, {'title': 'Диван для отдыха', 'price': 5300}, {'title': 'Стол'}]
```

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):
        yield random.randint(begin, end)


if __name__ == '__main__':
    print(list(gen_random(6, 1, 5)))
```

Результат выполнения:

```
[1, 2, 1, 5, 1, 4]
```

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):  # kwargs для переменной длины
аргументов
        self.data = iter(items)  # метод iter() используется для получения
итератора
        self.word = set()  # создание пустого множества
        if kwargs:  # пока есть элементы
            self.app = kwargs['ignore_case']
        else:  # если элементы закончатся
            self.app = False

    def __iter__(self):  # обязательно надо вернуть объект итератора
        return self

    def __next__(self):  # вернуть следующий элемент в последовательности
        while True:
            x = next(self.data)  # перебираются элементы
            if self.app == True and type(x) != int:  # если это буквенные
символы
                x = x.lower()  # делает все элементы нижнего регистра
            if x not in self.word:
                self.word.add(x)  # добавляет элемент если его ещё не было
                return x


if __name__ == "__main__":
    data1 = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]
    data2 = gen_random(10, 1, 3)
    data3 = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']
    data4 = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']

    print('1 case')
    print(list(Unique(data1)))
    print('2 case')
    print(list(Unique(data2)))
    print('3 case')
    print(list(Unique(data3, ignore_case=False)))  # игнорируем регистр
    print('4 case')
    print(list(Unique(data4, ignore_case=True)))  # не игнорируем регистр
```

Результат выполнения:

```
1 case
[1, 2]
2 case
[1, 2, 3]
3 case
['a', 'A', 'b', 'B']
4 case
['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, key=abs, reverse=True)  # abs берёт числа по модулю
    print(result)

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

Результат выполнения:

```
[123, 100, -100, -30, 4, -4, 1, -1, 0]
[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_to_decorate):
    def decorated_func(*args, **kwargs):
        incoming = func_to_decorate(*args, **kwargs)
        print(func_to_decorate.__name__)
        if isinstance(incoming, list):
            # Функцией isinstance() можно проверить класс, кортеж с классами,
либо рекурсивный кортеж кортежей.
            for i in incoming:
                print(i)
        elif isinstance(incoming, dict):
            for i in incoming:
                print(str(i) + ", " + str(incoming[i]))
        else:
            print(incoming)
        return incoming

    return decorated_func


@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__':
    test_1()
    test_2()
    test_3()
    test_4()
```

Результат 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:

```
import time
from contextlib import contextmanager


class cm_timer1:  # реализации на основе класса

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

    def __enter__(self):  # этот метод создает и возвращает объект связи базы
данных
        self.startTime = time.time()

    def __exit__(self, type, value, traceback):  # этот метод закрывает файл
        if type is not None:
            print(type, value, traceback)
        else:
            print("time: {}".format(time.time() - self.startTime))


@contextmanager
def cm_timer2():  # реализация с использованием библиотеки contextlib
    startTime = time.time()
    yield
    print("time: {}".format(time.time() - startTime))


if __name__ == '__main__':
```

```
    with cm_timer1():  # with вызывает метод __enter__
        time.sleep(5.5)
    with cm_timer2():
        time.sleep(5.5)
```

Результат выполнения:

```
time: 5.5142529010772705
time: 5.51325535774231
```

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:

```python
import json
from print_result import print_result
from cm_timer import cm_timer1
from unique import Unique
from field import field
from gen_random import gen_random

# Сделаем другие необходимые импорты

path = r'C:\Users\79508\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 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

@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(str.lower(string),
'программист'), arg))


@print_result
def f3(arg):
    return list(map(lambda s: s + " 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), 1000000, 2000000))))


if __name__ == '__main__':
    with cm_timer1():
        f4(f3(f2(f1(data))))
```

Результат выполнения:

f1 представлена не полностью, так как встречается много неповторяющихся профессий.

```
f1
администратор на телефоне
медицинская сестра
охранник сутки-день-ночь-вахта
врач анестезиолог реаниматолог
теплотехник
разнорабочий
электро-газосварщик
водитель gett/гетт и yandex/яндекс такси на личном автомобиле
монолитные работы
организатор – тренер
помощник руководителя
автоэлектрик
врач ультразвуковой диагностики в детскую поликлинику
менеджер по продажам ит услуг (b2b)
менеджер по персоналу
аналитик
воспитатель группы продленного дня
```

```
f2
программист
программист c++/c#/java
программист 1с
программистр-разработчик  информационных систем
программист c++
программист/ junior developer
программист / senior developer
программист/ технический специалист
программист c#
```

```
f3
программист с опытом Python
программист с++/c#/java с опытом Python
программист 1с с опытом Python
программистр-разработчик  информационных систем с опытом Python
программист с++ с опытом Python
программист/ junior developer с опытом Python
программист / senior developer с опытом Python
программист/ технический специалист с опытом Python
программист с# с опытом Python
```

```
f4
программист с опытом Python, зарплата 170591 рублей
программист с++/c#/java с опытом Python, зарплата 116676 рублей
программист 1с с опытом Python, зарплата 133135 рублей
программистр-разработчик  информационных систем с опытом Python, зарплата 143679 рублей
программист с++ с опытом Python, зарплата 113529 рублей
программист/ junior developer с опытом Python, зарплата 198255 рублей
программист / senior developer с опытом Python, зарплата 107312 рублей
программист/ технический специалист с опытом Python, зарплата 184843 рублей
программист с# с опытом Python, зарплата 138232 рублей
time: 0.04089021682739258
```