**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ**

**Московский государственный технический университет**

**им. Н.Э. Баумана**

**(МГТУ им. Н.Э. Баумана)**

**Кафедра «Системы обработки информации и управления» (ИУ5)**

Отчёт по лабораторной работе № 3-4

По курсу: «Базовые компоненты интернет-технологий»

Выполнил: Строганов Георгий Константинович

студент группы ИУ5-31Б.

Проверил: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_.\_\_\_.2022г.

Подпись: \_\_\_\_\_\_\_\_\_\_\_.

г. Москва 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. 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.

Пример: data = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]

Unique(data) будет последовательно возвращать только 1 и 2.

data = gen\_random(10, 1, 3)

Unique(data) будет последовательно возвращать только 1, 2 и 3.

data = [‘a’, ‘A’, ‘b’, ‘B’, ‘a’, ‘A’, ‘b’, ‘B’]

Unique(data) будет последовательно возвращать только a, A, b, B.

Unique(data, ignore\_case=True) будет последовательно возвращать только 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. 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:

* С использованием lambda-функции.
* Без использования lambda-функции.

**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.

**Текст программы:**

**main.py**

# -\*- coding: cp1251 -\*-

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.sort import sort\_array

from lab\_python\_fp.print\_result import print\_result\_tests

from lab\_python\_fp.cm\_timer import cm\_timer\_1, cm\_timer\_2

import time

def main():

print("Example 1")

goods = [

{'title': 'Kover', 'price': 2000, 'color': 'green'},

{'title': 'Divan dlia otdiha', 'color': 'black'}

]

print(\*field(goods, 'title'))

print(\*field(goods, 'title', 'price'))

print("Example 2")

gen\_random(5, 1, 3)

print("Example 3")

data = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]

print(Unique(data))

data = gen\_random(10, 1, 3)

print(Unique(data))

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

print(Unique(data))

print(Unique(data, ignore\_case = True))

print("Example 4")

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

sort\_array(data, 1)

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

sort\_array(data, 2)

print("Example 5")

print\_result\_tests()

print("Example 6")

with cm\_timer\_2():

time.sleep(1.5)

with cm\_timer\_2():

time.sleep(5.5)

if \_\_name\_\_ == "\_\_main\_\_":

main()

**cm\_timer.py**

from contextlib import contextmanager

from time import time

class cm\_timer\_1:

def \_\_enter\_\_(self):

self.\_\_time\_begin = time()

def \_\_exit\_\_(self, type, value, traceback):

print(time() - self.\_\_time\_begin)

@contextmanager

def cm\_timer\_2():

time\_begin = time()

yield

print(time() - time\_begin)

**field.py**

def field(items, \*args):

assert len(args) > 0

if len(args) == 1:

temp = []

for i in items:

temp\_key = i.get(args[0], "None")

if temp\_key != "None":

temp.append(temp\_key)

return temp

else:

k = []

for i in items:

temp = {}

for j in args:

temp\_key = i.get(j, "None")

if temp\_key != "None":

temp.update({j: temp\_key})

k.append(temp)

return k

**gen\_random.py**

import random

def gen\_random(num\_count, begin, end):

assert num\_count > 0

temp = []

for i in range(num\_count):

temp.append(random.randrange(begin, end + 1))

print(\*temp, sep = ", ")

return temp

**print\_result.py**

#декоратор

def print\_result(fun):

def wrapper():

print(fun.\_\_name\_\_)

if isinstance(fun(), list):

print(\*fun(), sep = "\n")

elif isinstance(fun(), dict):

temp\_fun = fun()

for i in temp\_fun:

print(i, temp\_fun.get(i), sep = " = ")

else:

print(fun())

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 print\_result\_tests():

test\_1()

test\_2()

test\_3()

test\_4()

**procces\_data.py**

# -\*- coding: cp1251 -\*-

import json

from xml.etree.ElementTree import tostring

from field import field

from gen\_random import gen\_random

from unique import Unique

from sort import sort\_array

from print\_result import print\_result

from cm\_timer import cm\_timer\_1, cm\_timer\_2

from operator import concat

def f1(data):

return Unique(field(data, 'job-name'), ignore\_case = True).unique()

def f2(temp):

return filter(lambda a: a.startswith('программист'), temp)

def f3(temp):

return list(map(lambda x: concat(x, ' c опытом Python'), temp))

def f4(temp):

return zip(temp, gen\_random(len(temp), 100000, 200000))

if \_\_name\_\_ == '\_\_main\_\_':

with open('lab\_python\_fp\data\_light.json', encoding = "UTF-8-sig") as f:

data = json.load(f)

with cm\_timer\_1():

for i in f4(f3(f2(f1(data)))):

print(i)

**sort.py**

def sort\_array(data, temp):

if temp == 1:

result = sorted(data, key = abs)[::-1]

print(result)

elif temp == 2:

result\_with\_lambda = sorted(data, key = lambda x: abs(x))[::-1]

print(result\_with\_lambda)

else:

print("ERROR")

**unique.py**

class Unique(object):

def \_\_init\_\_(self, items, \*\*kwargs):

self.\_\_r = []

if kwargs == {}:

try:

self.\_\_r = sorted(set([i for i in items]))

finally:

return

for key, value in kwargs.items():

if key == "ignore\_case" and value == True:

try:

self.\_\_r = sorted(set([i.lower() for i in items]))

finally:

break

def unique(self):

return self.\_\_r

def \_\_next\_\_(self):

try:

temp = self.\_\_r[self.begin]

self.begin += 1

return temp

except:

raise StopIteration

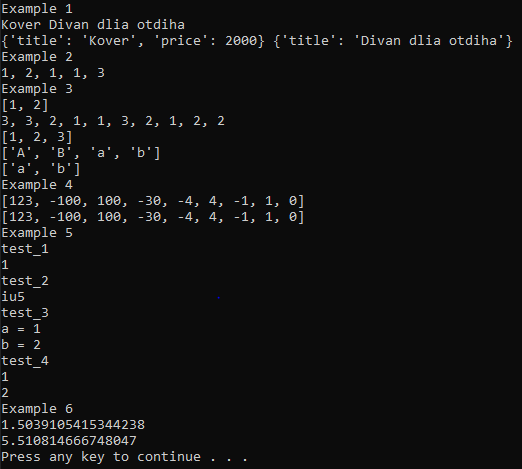
def \_\_str\_\_(self):

return str(self.\_\_r)

def \_\_iter\_\_(self):

return self

**Результаты тестирования:**



Для последнего задания:

