**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: Никулин Данила Дмитриевич

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 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 количество 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:

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. 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. 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). Функция f4 должна сгенерировать для каждой специальности зарплату от 100 000 до 200 000 рублей и присоединить её к названию специальности.

**Приложение 1. Текст программы:**

**field.py**

def field(items, \*args):

if len(args) == 1:

arr = []

for item in items:

for key in item:

if key == args[0] and item[key] is not None:

arr.append(item[key])

return arr

else:

for item in items:

dictionary = dict()

for key in item:

for argument in args:

if key == argument and item[argument] is not None:

dictionary[key] = item[key]

return(dictionary)

def main():

goods = [

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

{'title': 'Диван для отдыха', 'price': 5300, 'color': 'black'}

]

# должен выдавать 'Ковер', 'Диван для отдыха'.

res = (field(goods, 'title'))

for el in res:

print(el , end = '; ')

print()

# должен выдавать {'title': 'Ковер', 'price': 2000}, {'title': 'Диван для отдыха', 'price': 5300}.

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

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

main()

**gen\_random.py**

from random import randint

def gen\_random(count\_num, begin, end):

rand\_arr = []

for i in range(count\_num):

rand\_arr.append(randint(begin,end))

return rand\_arr

def main():

numbers = gen\_random(5, 1, 3)

for i in numbers:

print(numbers[i], end = ' ')

print()

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

main()

**unique.py**

from gen\_random import gen\_random

class Unique(object):

def \_\_init\_\_(self, items, ignore\_case=False, \*\*kwargs):

self.\_data = items

self.\_ignore\_case = ignore\_case

self.\_\_used\_data = set()

self.\_\_index = 0

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

# Если игнорируем капс, то пробегаемся по списку и приводим всё к общему капсу.

if self.\_ignore\_case:

for counter, el in enumerate(self.\_data):

if type(el) is str:

self.\_data[counter] = el.lower()

while True:

if self.\_\_index >= len(self.\_data):

raise StopIteration

else:

current = self.\_data[self.\_\_index]

self.\_\_index += 1

# если текущего числа ещё не было, добавляем и возвращаем.

if current not in self.\_\_used\_data:

self.\_\_used\_data.add(current)

return current

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

return self

def main():

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

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

try:

while True:

print(it.\_\_next\_\_())

except StopIteration:

print('The error "StopInteration" was caught')

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

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

try:

while True:

print(it.\_\_next\_\_())

except StopIteration:

print('The error "StopInteration" was caught')

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

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

try:

while True:

print(it.\_\_next\_\_())

except StopIteration:

print('The error "StopInteration" was caught')

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

main()

**sort.py**

def main():

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

result = sorted(data, key=abs, reverse=True)

print(result)

result\_with\_lambda = sorted(data, key=lambda a: abs(a), reverse=True)

print(result\_with\_lambda)

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

main()

**print\_rezult.py**

def print\_result(func):

def wrapper(lst=[], \*args, \*\*kwargs):

print(func.\_\_name\_\_)

if len(lst) == 0:

result = func(\*args, \*\*kwargs)

else:

result = func(lst, \*args, \*\*kwargs)

if type(result) is dict:

for key, el in result.items():

print(f'{key} = {el}')

elif type(result) is list:

print('\n'.join(map(str, result)))

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():

print('!!!!!!!!')

test\_1()

test\_2()

test\_3()

test\_4()

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

main()

**cm\_timer.py**

from time import sleep, time

from contextlib import contextmanager

class cm\_timer\_1:

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

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

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

print(f'Time of work: {time()- self.\_\_time\_begin}')

@contextmanager

def cm\_timer\_2():

time\_begin = time()

yield

print(f'Time of work: {time() - time\_begin}')

def main():

with cm\_timer\_1():

sleep(2.0)

with cm\_timer\_2():

sleep(2.0)

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

main()

**process\_data.py**

from print\_result import print\_result

from cm\_timer import cm\_timer\_1

from gen\_random import gen\_random

import json

import sys

try:

path = sys.argv[1]

print(path)

except:

path = 'data\_light.json'

with open(path) as f:

data = json.load(f)

@print\_result

def f1(arg) -> list:

return sorted(list(set([el['job-name'] for el in arg])), key=lambda x: x.lower())

@print\_result

def f2(arg) -> list:

return list(filter(lambda text: (text.split())[0].lower() == 'программист', arg))

@print\_result

def f3(arg) -> list:

return list(map(lambda lst: lst + ' с опытом Python', arg))

@print\_result

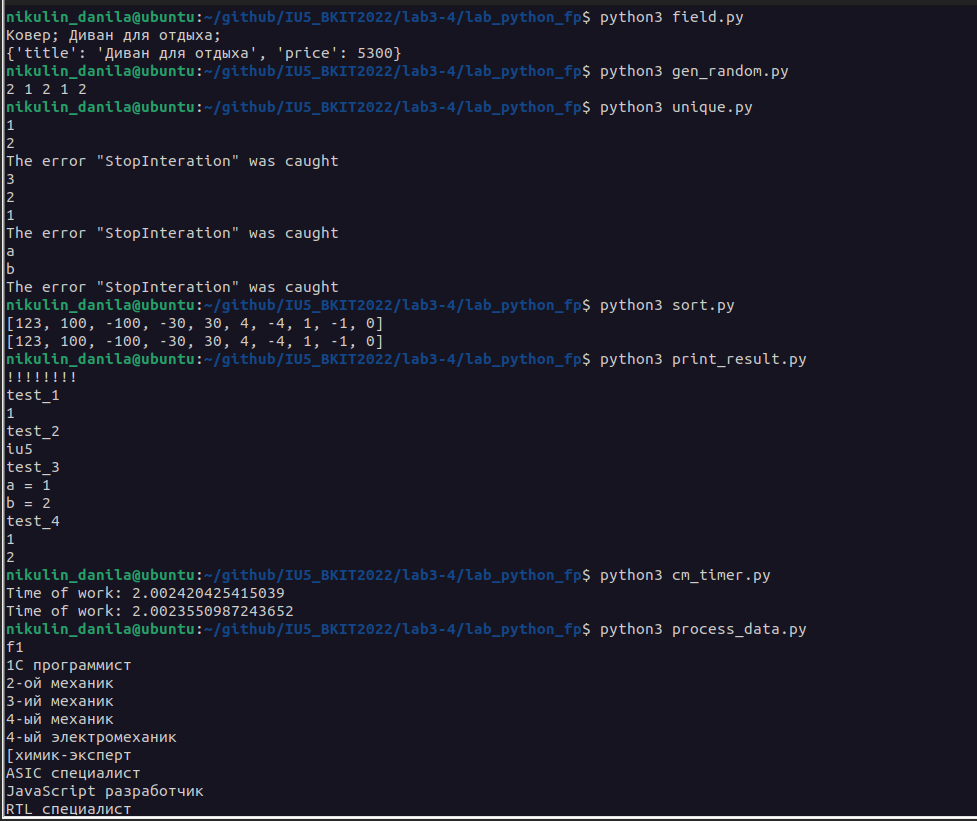
def f4(arg) -> list:

return list(zip(arg, ['зарплата ' + str(el) + ' руб.' for el in gen\_random(len(arg), 100000, 200000)]))

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

with cm\_timer\_1():

f4(f3(f2(f1(data))))

**Приложение 2. Результаты тестирования:**