Московский государственный технический университет им. Н.Э. Баумана

Факультет «Информатика и системы управления»

Кафедра «Автоматизированные системы обработки информации и управления»



**Отчет**

**По курсу «Разработка интернет приложений»**

**«Лабораторная работа №3»**

**ИСПОЛНИТЕЛЬ:**

Группа ИУ5-53Б

Кокозов С.И.

“18" января 2021 г.

**ПРЕПОДАВАТЕЛЬ:**

Гапанюк Ю.Е.

Москва 2020

**Задание:**

### 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.

### 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.

### 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:

1. С использованием lambda-функции.
2. 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.

### 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](https://github.com/iu5team/iu5web-fall-2020/tree/master/labs/lab3/data/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.

**Файл field.py**

# goods = [

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

# {'title': 'Диван для отдыха', 'price': 5300, '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', 'price': 5300}

def field(items, \*args):

assert len(args) > 0

if len(args) == 1:

for item in items:

for arg in args:

if arg in item:

yield item[arg]

else:

for item in items:

new\_item = {}

for arg in args:

if arg in item:

new\_item[arg] = item[arg]

if len(new\_item.keys()) > 0:

yield new\_item

goods = [

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

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

{'color': 'black'}

]

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

for i in field(goods, 'title'):

print (i)

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

**Файл gen\_random.py**

import random

# Пример:

# gen\_random(5, 1, 3) должен выдать выдать 5 случайных чисел

# в диапазоне от 1 до 3, например 2, 2, 3, 2, 1

# Hint: типовая реализация занимает 2 строки

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

for i in range(num\_count):

yield random.randint(begin, end)

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

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

for i in a:

print(i)

**Файл unique.py**

from gen\_random import \*

class Unique(object):

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

self.used\_elements = set()

self.case = False

for key in kwargs:

if key == "ignore\_case":

if kwargs[key] is True:

self.case = True

self.data = iter(items)

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

while True:

current = self.data.\_\_next\_\_()

if self.case is True and type(current) is str:

new\_item = current.lower()

else:

new\_item = current

if new\_item not in self.used\_elements:

self.used\_elements.add(new\_item)

return new\_item

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

return self

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

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

#data = gen\_random(6, 3, 10)

for i in Unique(data, ignore\_case=True):

print(i)

**Файл sort.py**

from math import fabs

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

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

result = sorted(data, reverse=True)

print(result)

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

print(result\_with\_lambda)

**Файл print\_result.py**

def print\_result(func\_to\_decorate):

def decorated\_func(\*arg):

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

a = func\_to\_decorate(\*arg)

if type(a) is list:

for i in a:

print(i)

elif type(a) is dict:

for key in a:

print("{} = {}".format(key, a[key]))

else:

print(a)

return a

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\_\_':

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

test\_1()

test\_2()

test\_3()

test\_4()

**Файл cm\_timer.py**

import time

from contextlib import contextmanager

class cm\_time\_1:

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

self.time = time.time()

def \_\_exit\_\_(self, exp\_type, exp\_value, traceback):

if exp\_type is not None:

print(exp\_type, exp\_value, traceback)

else:

print(time.time() - self.time)

@contextmanager

def cm\_time\_2():

timer = time.time()

yield

print(time.time() - timer)

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

with cm\_time\_1():

time.sleep(1)

with cm\_time\_2():

time.sleep(2)

**Файл process\_data.py**

import json

from unique import Unique

from print\_result import print\_result

from cm\_timer import cm\_time\_1

from field import field

from gen\_random import gen\_random

import sys

# Сделаем другие необходимые импорт

path = "data\_light.json"

# Необходимо в переменную path сохранить путь к файлу, который был передан при запуске сценария

with open(path) 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=lambda x: str.casefold(x))

@print\_result

def f2(arg):

return list(filter(lambda x: "программист" in x.lower(), arg))

@print\_result

def f3(arg):

return list(map(lambda x: x + " с опытом Python", arg))

@print\_result

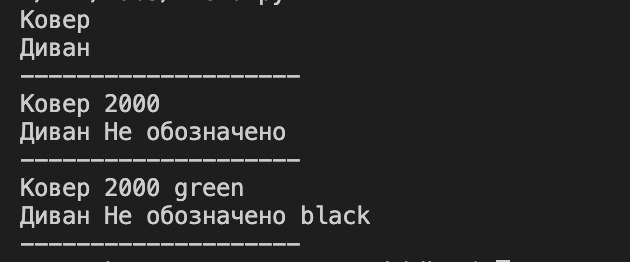
def f4(arg):

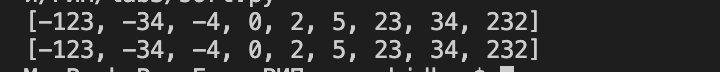
return dict((zip(arg, gen\_random(len(arg), 100000, 200000))))

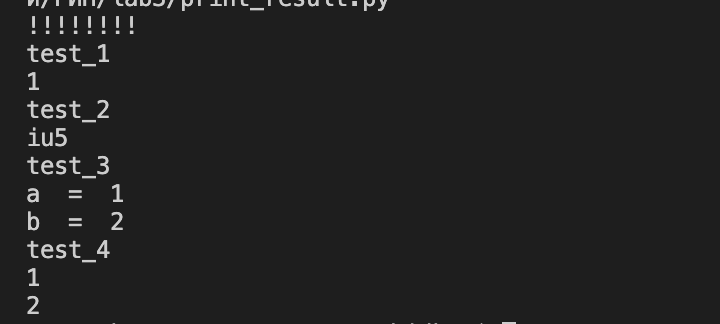
if \_\_name\_\_ == '\_\_main\_\_':

with cm\_time\_1():

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

**Вывод файлов в той же последовательности: **

****

****