Московский государственный технический университет им. Н.Э. Баумана

Факультет «Информатика и системы управления»

Кафедра «Системы обработки информации и управления»



**Отчет**

**Лабораторная работа № 3**

**По курсу «Разработка интернет-приложений»**

**«Функциональные возможности языка Python»**

**ИСПОЛНИТЕЛЬ:**

Группа ИУ5-55

\_Кожуро Б.Е.

"20"\_октября\_2020 г.

**ПРЕПОДАВАТЕЛЬ:**

\_\_\_Гапанюк Ю.Е.\_\_

"\_\_"\_\_\_\_\_\_\_\_2020 г.

Москва 2020

1. **Задание**
   1. Необходимо реализовать генератор field. Генератор field последовательно выдает значения ключей словаря.
      1. В качестве первого аргумента генератор принимает список словарей, дальше через \*args генератор принимает неограниченное количествово аргументов.
      2. Если передан один аргумент, генератор последовательно выдает только значения полей, если значение поля равно None, то элемент пропускается.
      3. 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
   2. 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. gen\_random(5, 1, 3) должен выдать 5 случайных чисел в диапазоне от 1 до 3, например 2, 2, 3, 2, 1
   3. 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.
   4. 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.
   5. 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.
   6. 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.
      1. Функция f1 должна вывести отсортированный список профессий без повторений (строки в разном регистре считать равными). Сортировка должна игнорировать регистр. Используйте наработки из предыдущих задач.
      2. Функция f2 должна фильтровать входной массив и возвращать только те элементы, которые начинаются со слова “программист”. Для фильтрации используйте функцию filter.
      3. 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.
      4. 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.
2. **Текст программы**
   1. **Field.py**

def field(items, \*args):  
 values = []  
 try:  
 assert len(args) > 0  
 except:  
 print('нет аргументов')  
 return []  
 if len(args) == 1:  
 for item in items:  
 val = item.get(args[0])  
 if val is not None:  
 values.append(val)  
 else:  
 for item in items:  
 dic = {}  
 for arg in args:  
 val = item.get(arg)  
 if val is not None:  
 dic[arg] = val  
 if len(dic) != 0:  
 values.append(dic)  
 return values  
  
# тест  
goods = [  
 {'title': 'Ковер', 'price': 2000, 'color': 'green'},  
 {'title': 'Диван для отдыха', 'price': 5300, 'color': 'black'},  
 {'title': 'Стол', 'price': None, 'color': 'white'},  
]  
  
def main():  
 print('Пример 1:')  
 print(field(goods, 'title'))  
 print('\n', 'Пример 2:')  
 print(field(goods, 'title', 'color'))  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()

* 1. **Gen\_random.py**

import random  
  
def gen\_random(count, s, f):  
 for i in range(count):  
 yield random.randint(s, f)  
  
# тест  
def main():  
 gen = gen\_random(30, 1, 10)  
 for i in gen:  
 print(i, end=' ')  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()

* 1. **Unique.py**

# Итератор для удаления дубликатов  
class Unique(object):  
 def \_\_init\_\_(self, items, \*\*kwargs):  
 self.position = 0  
 self.unique\_elements = set()  
 self.elements = items  
 if len(kwargs) != 0:  
 self.ignore\_case = kwargs  
 else:  
 self.ignore\_case = False  
  
 def \_\_next\_\_(self):  
 while True:  
 for element in self.elements:  
 current\_element = element  
 self.position = self.position + 1  
 if isinstance(current\_element, str):  
 if not (current\_element in self.unique\_elements) and \  
 not (self.ignore\_case and (current\_element.lower() in self.unique\_elements)):  
 if self.ignore\_case:  
 self.unique\_elements.add(current\_element.lower())  
 else:  
 self.unique\_elements.add(current\_element)  
 return current\_element  
 elif (current\_element not in self.unique\_elements):  
 self.unique\_elements.add(current\_element)  
 return current\_element  
 else:  
 raise StopIteration  
  
 def \_\_iter\_\_(self):  
 return self  
  
# Тест  
def main():  
 array = ['a', 'a', 'a' , 'A' ,'A', 'b', 'B', 10, 1, 1, 'Boris', 'boris', 'bOrIs', 'Boria']  
 iterator1 = Unique(array)  
 res1 = []  
 iterator2 = Unique(array, ignore\_case=True)  
 res2 = []  
 for i in iterator1:  
 res1.append(i)  
 print(res1)  
 for i in iterator2:  
 res2.append(i)  
 print(res2)  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()

* 1. **Sort.py**

data = [4, -30, 100, -100, 123, 1, 0, -1, -4]  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 result = sorted(data, key=abs, reverse=True)  
 print(result)  
  
 result\_with\_lambda = sorted(data, key=lambda x: abs(x), reverse=True)  
 print(result)

* 1. **Print\_result.py**

def print\_result(f):  
 def decorated(\*args):  
 print(f.\_\_name\_\_)  
 res = f(\*args)  
 #сверка типа  
 if isinstance(res, list):  
 for i in res:  
 print(i)  
 elif isinstance(res, dict):  
 for i in res:  
 print(i, ' = ', res.get(i))  
 else:  
 print(res)  
 return res  
 return decorated  
  
@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()

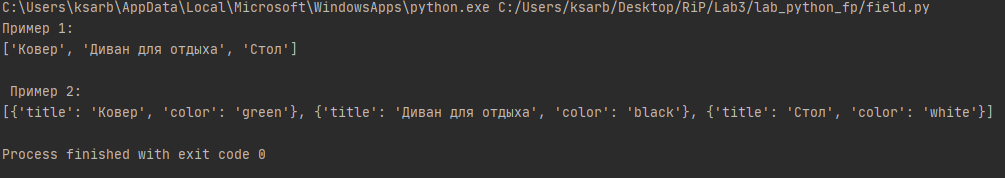
* 1. **Cm\_timer.py**

import time  
from contextlib import contextmanager  
  
  
class cm\_timer\_1:  
  
 def \_\_init\_\_(self):  
 self.start = 0  
  
 def \_\_enter\_\_(self):  
 self.start = time.time()  
  
 def \_\_exit\_\_(self, exc\_type, exc\_val, traceback):  
 end\_time = time.time()  
 if exc\_type is not None:  
 print(exc\_type, exc\_val, traceback)  
 else:  
 print('time: ', end\_time - self.start)  
  
  
@contextmanager  
def cm\_timer\_2():  
 start = time.time()  
 yield  
 end = time.time()  
 print('time: ', end - start)  
  
  
def main():  
 with cm\_timer\_1():  
 time.sleep(2.5)  
  
 with cm\_timer\_2():  
 time.sleep(2.5)  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()

* 1. **Process\_sata.py**

from Lab3.lab\_python\_fp.cm\_timer import cm\_timer\_1  
from Lab3.lab\_python\_fp.print\_result import print\_result  
from Lab3.lab\_python\_fp.unique import Unique  
from Lab3.lab\_python\_fp.field import field  
from Lab3.lab\_python\_fp.gen\_random import gen\_random  
import json  
import sys  
import os  
  
  
path = os.getcwd()+'\\data\_light.json'  
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 Unique(field(arg, 'job-name'), ignore\_case=True)  
  
  
@print\_result  
def f2(arg):  
 return filter(lambda x: x.startswith('Программист'), arg)  
  
  
@print\_result  
def f3(arg):  
 return list(map(lambda x: x + ' с опытом Python', arg))  
  
  
@print\_result  
def f4(arg):  
 salary = gen\_random(len(arg), 100000, 200000)  
 return list(zip(arg, salary))  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 with cm\_timer\_1():  
 x= f3(f2(f1(data)))  
 f4(x)

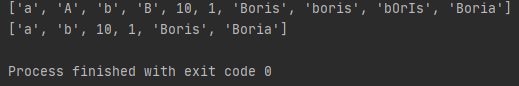
1. **Скриншоты выполнения**
   1. **Field.py**

****

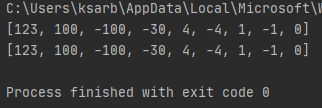
* 1. **Gen\_random.py**

****

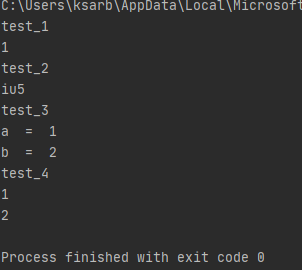
* 1. **Unique.py**

****

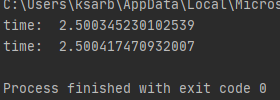
* 1. **Sort.py**

****

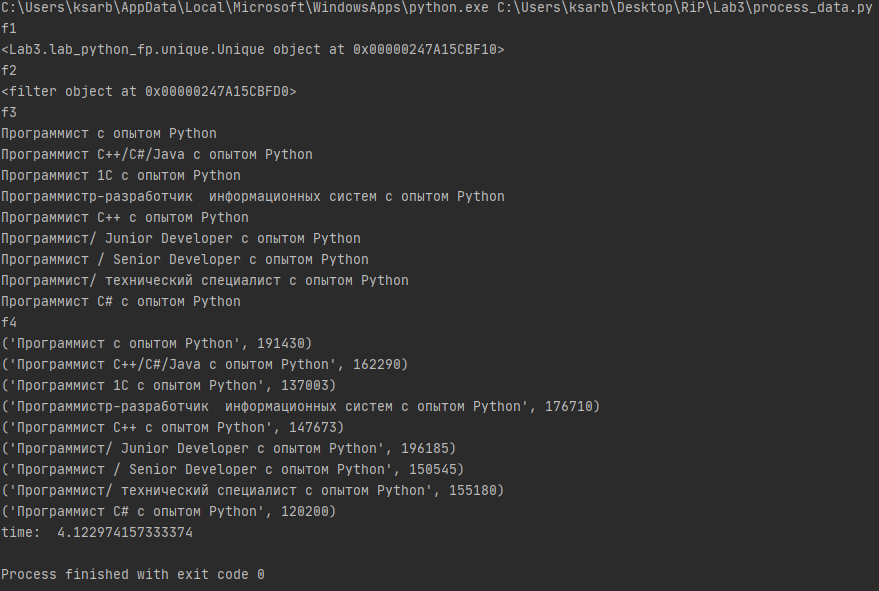
* 1. **Print\_result.py**

****

* 1. **Cm\_timer.py**

****

* 1. **Process\_sata.py**

****