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

\_Якшин Е.Д.

**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):  
 try:  
 assert len(args) > 0  
 except AssertionError:  
 print("Нет второго аргумента")  
 if len(args) == 1:  
 for i in range(len(items)):  
 if args[0] in items[i] and items[i].get(args[0])!=None:  
 yield items[i].get(args[0])  
 else:  
 for i in range(len(items)):  
 s={}  
 for j in range(len(args)):  
 if args[j] in items[i] and items[i].get(args[j]) != None:  
 s.update({args[j]: items[i].get(args[j])})  
 yield s  
  
  
goods = [  
 {'title': 'Ковер', 'price': 2000, 'color': 'green'},  
 {'title': 'Диван для отдыха', 'color': 'blue'},  
 {'title': 'Пуфик', 'price': 5000, 'color': 'red'}  
]  
f=field(goods,'title', 'price')  
for i in f:  
 print(i, end=', ')  
print('\n')  
f = field(goods, 'price', 'color')  
for i in f:  
 print(i, end=', ')  
print('\n')  
f = field(goods,'title')  
for i in f:  
 print(i, end=', ')

1. **Gen\_random.py**

from random import randint  
  
  
def gen\_random(num\_count, begin, end):  
 for i in range(num\_count):  
 yield randint(begin, end)  
  
  
def main():  
 gen = gen\_random(5, 1, 3)  
 for i in gen:  
 print(i, end=' ')  
 print('\n')  
 gen = gen\_random(5, 1, 3)  
 for i in gen:  
 print(i, end=' ')  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

1. **Unique.py**

import gen\_random  
  
  
class Unique(object):  
 def \_\_init\_\_(self, items, \*\*kwargs):  
 self.used\_elements = set()  
 self.items = items  
 self.index = 0  
 if len(kwargs) != 0:  
 self.ignore\_case = kwargs  
 else:  
 self.ignore\_case = False  
  
 def \_\_next\_\_(self):  
 while True:  
 for item in self.items:  
 temp\_item = item  
 self.index += 1  
 if (temp\_item not in self.used\_elements) \  
 and not(self.ignore\_case and temp\_item.swapcase() in self.used\_elements):  
 self.used\_elements.add(temp\_item)  
 return temp\_item  
 else:  
 raise StopIteration  
  
 def \_\_iter\_\_(self):  
 return self  
  
  
def main():  
 data1 = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 3, 3, 3, ]  
 print(data1)  
 iter1 = Unique(data1)  
 for i in iter1:  
 print(i, end=' ')  
 print('\n')  
 data2 = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']  
 print(data2)  
 iter2 = Unique(data2)  
 for i in iter2:  
 print(i, end=' ')  
 print('\n')  
 print(data2)  
 iter3 = Unique(data2, ignore\_case=True)  
 for i in iter3:  
 print(i, end=' ')  
 print('\n')  
 gen = gen\_random.gen\_random(3, 1, 3)  
 iter4 = Unique(gen)  
 for i in iter4:  
 print(i, end=' ')  
  
  
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(func\_to\_decorate):  
 def decorated\_func(\*args):  
 print(func\_to\_decorate.\_\_name\_\_)  
 result = func\_to\_decorate(\*args)  
 if type(result) is list:  
 for i in result:  
 print(i)  
 elif type(result) is dict:  
 for i in result:  
 print(i, result.get(i), sep=' = ')  
 else:  
 print(result)  
  
 return result  
  
 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()

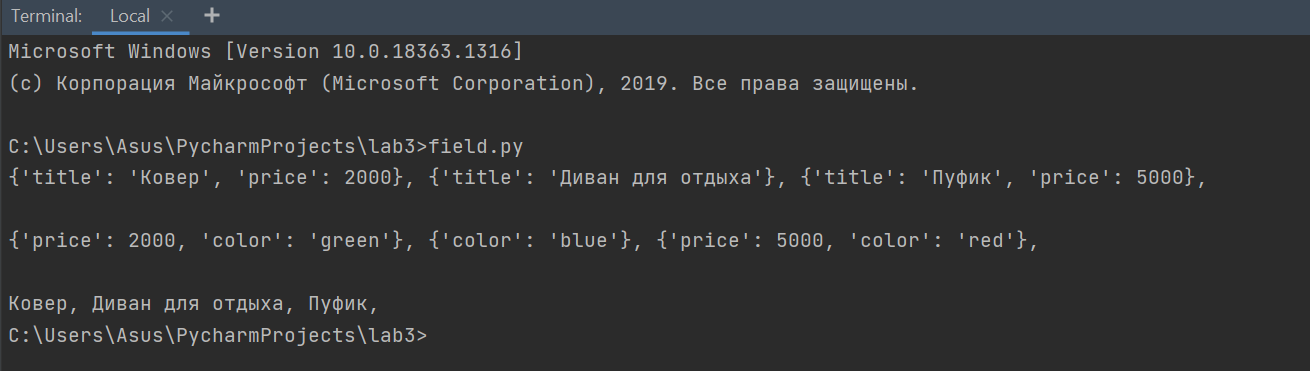
1. **Cm\_timer.py**

from contextlib import contextmanager  
from time import time, sleep  
  
  
class cm\_timer\_1:  
 def \_\_init\_\_(self):  
 self.begin\_time = time()  
  
 def \_\_enter\_\_(self):  
 pass  
  
 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.begin\_time)  
  
  
@contextmanager  
def cm\_timer\_2():  
 begin\_time=time()  
 yield 1  
 print('time:', time() - begin\_time)  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
  
 with cm\_timer\_1():  
 sleep(3.5)  
  
 with cm\_timer\_2():  
 sleep(2.5)

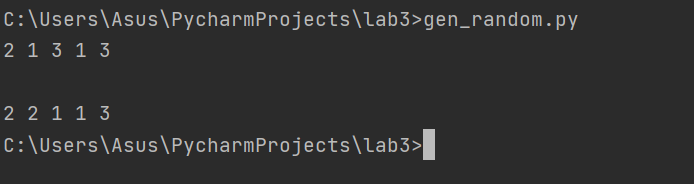
1. **Process\_sata.py**

import json  
from unique import Unique  
from print\_result import print\_result  
from cm\_timer import cm\_timer\_1  
from random import randint  
from gen\_random import gen\_random  
from field import field  
import re  
import sys  
  
path = 'data\_light.json'  
  
  
with open(path) as f:  
 data = json.load(f)  
  
  
@print\_result  
def f1(arg):  
 return Unique(field(data, "job-name"), ignore\_case=True)  
  
  
@print\_result  
def f2(arg):  
 return filter(lambda x: re.search(r'\bПрограммист\b', x) or re.search(r'\bпрограммист\b', x), arg)  
  
  
@print\_result  
def f3(arg):  
 return list(map(lambda x: x+' с опытом Python', arg))  
  
  
@print\_result  
def f4(arg):  
 price = gen\_random(len(arg), 100000, 200000)  
 res = list(zip(arg, (list(map(lambda x: ', зарплата' + x + 'руб', ''.join(str(list(price)))[1:-1].split(','))))))  
 return [''.join(i) for i in res]  
  
  
def main():  
 with cm\_timer\_1():  
  
 f4(f3(f2(f1(data))))  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()

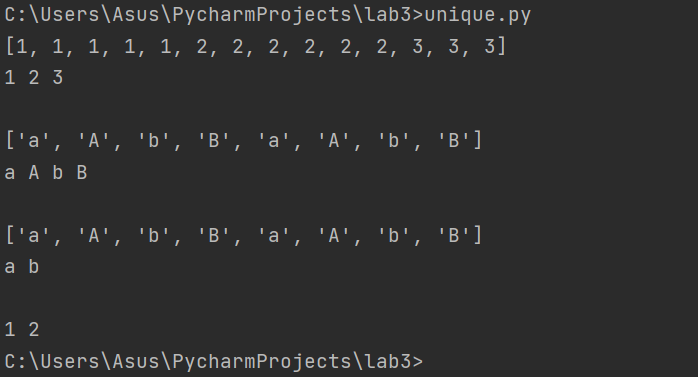
1. **Скриншоты выполнения**
   1. **Field.py**



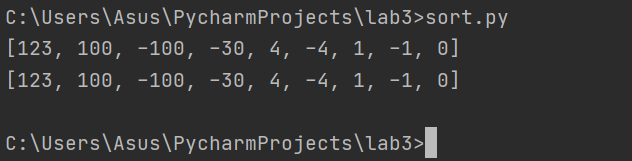
* 1. **Gen\_random.py**



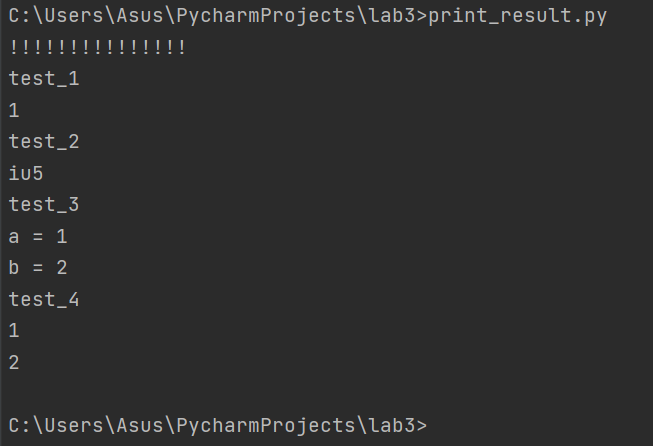
* 1. **Unique.py**



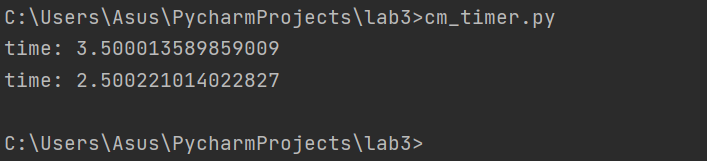
* 1. **Sort.py**



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



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



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

