**Министерство науки и высшего образования Российской Федерации** **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования** **«Московский государственный технический университет** **имени Н.Э. Баумана**

**(национальный исследовательский университет)»**

**(МГТУ им. Н.Э. Баумана)**

**Факультет «Информатика и системы управления»**

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

Лабораторная работа №3

по дисциплине «Базовые компоненты интернет-технологий»

Выполнил:

студент группы ИУ5-33Б

Зорькин А.В.

Проверил:

Канев А.И.

2021 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:

* 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.

**Текст программы:**

def field(items, \*args):  
 assert len(args) > 0, 'Передайте хотя бы один аргумент, не считая списка'  
 if len(args) == 1:  
 result = [items[i][args[0]] for i in range(len(items)) if (items[i][args[0]] != None)]  
 if result:  
 yield result;  
 else:  
 result = [{x: items[i][x] for x in args if (x in items[i].keys() and items[i][x] != None)} for i in  
 range(len(items))]  
 if result:  
 yield result;  
  
  
goods = [{'title': 'Ковер', 'price': 2000, 'color': 'green'},  
 {'title': 'Диван для отдыха', 'price': 5300, 'color': 'black'}]  
print(\*list(field(goods, 'title')))  
print(\*list(field(goods, 'title', 'price', 'color')))

**Результаты выполнения:**

****

**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.

**Текст программы:**

import random  
  
  
def gen\_random(num\_count, begin, end):  
 for i in range(num\_count):  
 yield (random.randint(begin, end))  
  
  
print(list(gen\_random(5, 1, 10)))

**Результаты выполнения:**



**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.

**Текст программы:**

import random  
from gen\_random import gen\_random  
  
  
class Unique(object):  
 def \_\_init\_\_(self, items, \*\*kwargs):  
 self.data = iter(items)  
 self.word = set()  
 if not kwargs:  
 self.app = False  
 else:  
 self.app = kwargs['ignore\_case']  
  
 def \_\_iter\_\_(self):  
 return self  
  
 def \_\_next\_\_(self):  
 while True:  
 x = next(self.data)  
 if self.app == True and type(x) != int:  
 x = x.lower()  
 if x not in self.word:  
 self.word.add(x)  
 return x  
  
  
def main():  
 data = gen\_random(2, 1, 15)  
 data = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']  
  
 for i in Unique(data, ignore\_case=False):  
 print(i)  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

**Результаты выполнения:**



Для генератора:



**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.

**Текст программы:**

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

**Результаты выполнения:**



**Задача 5 (файл print\_result.py)**

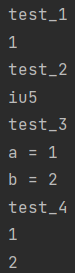
Необходимо реализовать декоратор print\_result, который выводит на экран результат выполнения функции.

* 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.

**Текст программы:**

def print\_result(func):  
 def dek(\*a):  
 print(func.\_\_name\_\_)  
 if type(func(\*a)) == list:  
 for x in func(\*a):  
 print (x)  
 elif type(func(\*a)) == dict:  
 for x in func(\*a).keys():  
 print(x, '=', func(\*a)[x])  
 else:  
 print(func(\*a))  
  
 return dek  
  
  
@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()

**Результаты выполнения:**



**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)

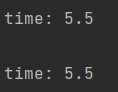
После завершения блока кода в консоль должно вывестись time: 5.5 (реальное время может несколько отличаться).

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).

**Текст программы:**

import time  
from contextlib import contextmanager  
  
  
class cm\_timer\_1:  
 def \_\_init\_\_(self, before='time: ', after=''):  
 self.before = before  
 self.after = after  
 self.time = time.time()  
  
 def \_\_enter\_\_(self):  
 print(self.before, end='')  
 return (self.time)  
  
 def \_\_exit\_\_(self, exc\_type, exc\_val, exc\_tb):  
 if exc\_type is not None:  
 print(exc\_type, exc\_val, exc\_tb)  
 else:  
 print(self.after)  
  
  
  
with cm\_timer\_1() as cm\_1:  
 time.sleep(5.5)  
 print('{0:2.1f}'.format(time.time() - cm\_1))  
  
  
@contextmanager  
def cm\_timer\_2(before='time: ', after=''):  
 print(before,end='')  
 yield time.time()  
 print(after)  
  
  
with cm\_timer\_2() as cm\_2:  
 time.sleep(5.5)  
 print('{0:2.1f}'.format(time.time() - cm\_2))

**Результаты выполнения:**

****

**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/ugapanyuk/BKIT_2021/blob/main/notebooks/fp/files/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:**

import json  
import sys  
import time  
from contextlib import contextmanager  
import random  
  
  
class Unique(object):  
 def \_\_init\_\_(self, items, \*\*kwargs):  
 self.data = iter(items)  
 self.word = set()  
 if not kwargs:  
 self.app = False  
 else:  
 self.app = kwargs['ignore\_case']  
  
 def \_\_iter\_\_(self):  
 return self  
  
 def \_\_next\_\_(self):  
 while True:  
 x = next(self.data)  
 if self.app == True:  
 x['job-name'] = x['job-name'].lower()  
 if x['job-name'] not in self.word:  
 self.word.add(x['job-name'])  
 return x['job-name']  
  
  
def print\_result(func):  
 def dek(\*a):  
 print(func.\_\_name\_\_)  
 if type(func(\*a)) == list:  
 for x in func(\*a):  
 print(x)  
 return func(\*a)  
 elif type(func(\*a)) == dict:  
 for x in func(\*a).keys():  
 print(x + func(\*a)[x])  
 else:  
 print(func(\*a))  
  
 return dek  
  
  
def gen\_random(num\_count, begin, end):  
 for i in range(num\_count):  
 s = random.randint(begin, end)  
 raise s  
  
  
@contextmanager  
def cm\_timer\_2(before='', after=''):  
 print(before, end='')  
 yield time.time()  
 print(after)  
  
  
# Сделаем другие необходимые импорты  
  
path = 'data\_light.json'  
  
# Необходимо в переменную path сохранить путь к файлу, который был передан при запуске сценария  
  
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 list(Unique(arg, ignore\_case=True))  
  
  
@print\_result  
def f2(arg):  
 return list(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):  
 a = (', зарплата ' + str(random.randint(100000, 200000)) + ' руб.' for x in arg)  
 return dict(zip(arg, a))  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 with cm\_timer\_2() as t:  
 f4(f3(f2(f1(data))))  
  
 print('time:', time.time() - t)

**Результаты выполнения:**

Приведена часть вывода после работы функции f1, так как встречается множество неповторяющихся профессий.

