**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»

Выполнила:

студентка группы ИУ5-33Б

Шаповалова В.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:**

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  
 m = []  
 for arg in items:  
 for i in range(0, len(args)):  
 x = arg.get(args[i])  
 m.append(x)  
 return m  
  
  
print(\*field(goods, 'title'))  
print(\*field(goods, 'title', 'price'))

print(\*list(field(goods, 'title', 'price', 'color')))

**Результаты выполнения:**

**Изображение выглядит как текст

Автоматически созданное описание**

**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):  
 pass  
 # Необходимо реализовать генератор  
 mas = []  
 for i in range(0, num\_count):  
 mas.append(random.randint(begin, end))  
 return mas  
  
print(gen\_random(2,3,10))

**Результаты выполнения:**

Изображение выглядит как текст

Автоматически созданное описание

**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  
  
def gen\_random(num\_count, begin, end):  
 # Необходимо реализовать генератор  
 mas = []  
 for i in range(0, num\_count):  
 mas.append(random.randint(begin, end))  
 return mas  
  
data1 = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]  
data2 = gen\_random(10, 5, 10)  
data3 = ['a', 'A', 'B', 'a', 'A', 'b', 'B']  
# Итератор для удаления дубликатов  
ignore\_case = {False, True}  
  
class Unique(object):  
 def \_\_init\_\_(self, items, \*\*kwargs):  
 self.ignore\_case = False  
 if len(kwargs) > 0:  
 self.ignore\_case = kwargs['ignore\_case']  
 mas = sorted(list(set(items)))  
 if self.ignore\_case == True:  
 for x in range(0, len(mas)):  
 mas[x] = mas[x].lower()  
 mas2 = list(set(mas))  
 print(\*mas2)  
 else:  
 print(\*mas)  
  
 def \_\_next\_\_(self):  
 return self  
  
 def \_\_iter\_\_(self):  
 return self  
  
  
Unique(data1)  
Unique(data2)  
Unique(data3)  
Unique(data3, ignore\_case=False)  
Unique(data3, ignore\_case=True)

**Результаты выполнения:**

Изображение выглядит как текст

Автоматически созданное описание

**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\_\_':  
 result = sorted(data, reverse=True, key=abs)  
 print(result)  
  
 result\_with\_lambda = sorted(data, key=lambda mod: abs(mod), reverse=True)  
 print(result\_with\_lambda)

**Результаты выполнения:**

Изображение выглядит как текст, электроника

Автоматически созданное описание

**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(fn):  
 def wrapped(\*args, \*\*kwargs):  
 print(fn.\_\_name\_\_)  
 if isinstance(fn(\*args, \*\*kwargs), dict)==True:  
 # for dict.items in range()  
 m = fn(\*args, \*\*kwargs)  
 for key, value in m.items():  
 print('{0} = {1}'.format(key, value))  
 elif isinstance(fn(\*args, \*\*kwargs), list) == True:  
 m = fn(\*args, \*\*kwargs)  
 for value in m:  
 if isinstance(value, tuple) == True:  
 print('{0}, зарплата {1} руб'.format(value[0], value[1]))  
 else:  
 print(value)  
 elif isinstance(fn(\*args, \*\*kwargs), (int, str)):  
 print(fn(\*args, \*\*kwargs))  
 return fn(\*args, \*\*kwargs)  
 return wrapped  
  
  
@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()

**Результаты выполнения:**

Изображение выглядит как текст

Автоматически созданное описание

**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  
from time import sleep  
  
  
@contextmanager  
def cm\_timer\_2():  
 try:  
 begin = time.perf\_counter()  
 yield begin  
 finally:  
 print(time.perf\_counter() - begin)  
  
  
class cm\_timer\_1():  
  
 def start(self):  
 self.begin = time.perf\_counter()  
  
 def stop(self):  
 work = time.perf\_counter() - self.begin  
 print(work)  
  
 def \_\_enter\_\_(self):  
 self.start()  
  
 def \_\_init\_\_(self):  
 self.startTime = time.time()  
  
 def \_\_exit\_\_(self, \*args):  
 self.stop()  
  
  
with cm\_timer\_1():  
 sleep(5.5)  
  
  
with cm\_timer\_2():  
 sleep(5.5)

**Результаты выполнения:**

**Изображение выглядит как текст

Автоматически созданное описание**

**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  
from re import search  
from lab3.cm\_timer import cm\_timer\_1  
from lab3.field import field  
from lab3.gen\_random import gen\_random  
from lab3.print\_result import print\_result  
  
# Сделаем другие необходимые импорты  
  
path = r'C:\data\_light.json'  
#path = r'C:\Users\hoppl\PycharmProjects\LAB3\json\_file.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  
  
def Unique(items):  
 for x in range(0, len(items)):  
 items[x] = items[x].lower()  
 temp = []  
 temp2 = []  
 for x in items:  
 if x not in temp:  
 temp.append(x)  
 for x in temp:  
 temp2.append(x.capitalize())  
 return temp2  
  
  
@print\_result  
def f1(arg):  
 result = sorted(Unique(field(arg, 'job-name')), key=str.lower)  
 # result = sorted(Unique(field(arg, 'job-name')), key=lambda s: s.lower())  
 return result  
  
  
@print\_result  
def f2(arg):  
 result = list(filter(lambda x: search('^Программист', x)!=None , arg))  
 return result  
  
  
@print\_result  
def f3(arg):  
 result = list(map(lambda x: x + ' с опытом Python', arg))  
 return result  
  
  
@print\_result  
def f4(arg):  
 m = gen\_random(len(arg), 100000, 200000)  
 result = list(zip(arg, m))  
 return result  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 with cm\_timer\_1():  
 f4(f3(f2(f1(data))))

**Результаты выполнения:**

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

Изображение выглядит как текст

Автоматически созданное описание