**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 ИУ5-32Б

Балабанов А.О.

Проверил:

Канев А.И.

2021 г.

# **Общее описание задания**

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**

## Описание задачи

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.

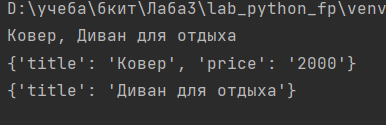
Если передан один аргумент, генератор последовательно выдает только значения полей, если значение поля равно None, то элемент пропускается.

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.

## Текст программы

def field(items, \*args):  
 assert len(args) > 0  
 if len(args) == 1:  
 for string in items:  
 text = string.get(args[0])  
 if text is not None:  
 yield text  
 else:  
 for j in items:  
 string = dict()  
 for key in args:  
 text = j.get(key)  
 if text is not None:  
 string[key] = text  
 if len(string) != 0:  
 yield string  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 goods = [  
 {'title': 'Ковер', 'price': 2000, 'color': 'green'},  
 {'title': 'Диван для отдыха', 'price': 5300, 'color': None},  
 {'title': 'Стол', 'price': None, 'color': 'white'},  
 {'title': None, 'price': 2021, 'color': 'black'}  
 ]  
  
 data1 = list()  
 for i in field(goods, 'title'):  
 data1.append(i)  
 print(data1)  
 data2 = list()  
  
 for i in field(goods, 'title', 'price'):  
 data2.append(i)  
 print(data2)  
 data3 = list()  
  
 for i in field(goods, 'title', 'price', 'color'):  
 data3.append(i)  
 print(data3)

## Экранные формы с примерами выполнения программы



# **Задача №2**

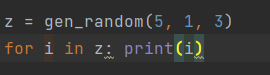
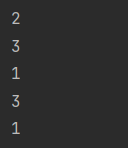
## Описание задачи

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\_random1(num\_count, begin, end):  
 for i in range(num\_count):  
 print(random.randint(begin, end))  
  
def gen\_random2(num\_count, begin, end):  
 for i in range(num\_count):  
 yield random.randint(begin, end)  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 gen\_random1(5,1,3)  
 print(list(gen\_random2(5, 1, 3)))

## Экранные формы с примерами выполнения программы

# **Задача №3**

## Описание задачи

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.

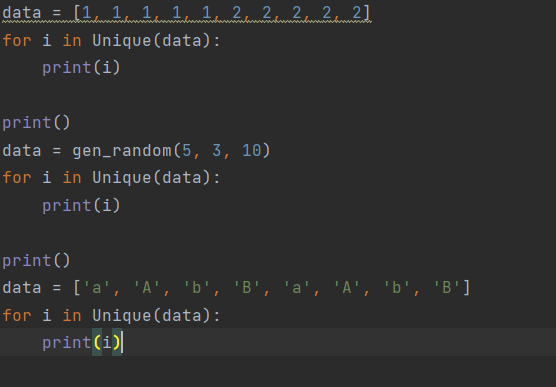
Итератор должен поддерживать работу как со списками, так и с генераторами.

Итератор не должен модифицировать возвращаемые значения.

## Текст программы

from gen\_random import gen\_random2  
  
class Unique(object):  
 def \_\_init\_\_(self, items, \*\*kwargs):  
 self.data = iter(items) # метод iter() используется для получения итератора  
 self.word = set() # создание пустого множества  
 if kwargs:  
 self.app = kwargs['ignore\_case']  
 else:  
 self.app = False  
  
 def \_\_iter\_\_(self): # возвращаем объект итератора  
 return self  
  
 def \_\_next\_\_(self): # вернуть следующий элемент в последовательности  
 while True:  
 x = next(self.data) # перебираются элементы  
 if self.app == True and type(x) != int: # если это буквенные символы  
 x = x.lower() # делает все элементы нижнего регистра  
 if x not in self.word:  
 self.word.add(x) # добавляет элемент если его ещё не было  
 return x  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 data1 = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]  
 data2 = gen\_random2(10, 1, 3)  
 data3 = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']  
  
 print('Первый пример')  
 print(list(Unique(data1)))  
 print('Второй пример')  
 print(list(Unique(data2)))  
 print('Третий пример')  
 print(list(Unique(data3, ignore\_case=False))) # игнорируем регистр  
 print('Четвёртый пример')  
 print(list(Unique(data3, ignore\_case=True))) # не игнорируем регистр

## Экранные формы с примерами выполнения программы

# **Задача №4**

## Описание задачи

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.

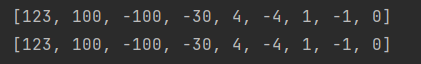
Необходимо решить задачу двумя способами:

1. С использованием lambda-функции.
2. Без использования lambda-функции.

## Текст программы

from functools import reduce  
  
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\_with\_lambda)

## Экранные формы с примерами выполнения программы



# **Задача №5**

## Описание задачи

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.

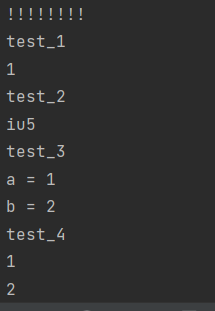
Если функция вернула список (list), то значения элементов списка должны выводиться в столбик.

Если функция вернула словарь (dict), то ключи и значения должны выводить в столбик через знак равенства.

## Текст программы

def print\_result(function): # реализация декоратора print\_result  
 def decorated(\*a): # \*a так как неизвестно количество аргументов  
 print(function.\_\_name\_\_)  
 if type(function(\*a)) == list:  
 for x in function(\*a):  
 print(x)  
 elif type(function(\*a)) == dict:   
 for x in function(\*a):  
 print(x, '=', function(\*a)[x])  
 else:  
 print(function(\*a))  
  
 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()

## Экранные формы с примерами выполнения программы



# **Задача №6**

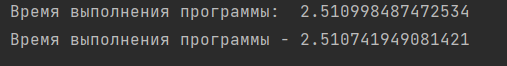
## Описание задачи

Необходимо написать контекстные менеджеры cm\_timer\_1 и cm\_timer\_2, которые считают время работы блока кода и выводят его на экран.

## Текст программы

import time  
from contextlib import contextmanager  
  
class cm\_timer\_1: #Первая реализация через класс  
 def \_\_enter\_\_(self):  
 self.time = time.time()  
 def \_\_exit\_\_(self, exc\_type, exc\_val, exc\_tb):  
 print("Время выполнения программы: ", time.time() - self.time)  
  
@contextmanager  
def cm\_timer\_2():  
 start\_time = time.time()  
 yield  
 end\_time = time.time() - start\_time  
 print("Время выполнения программы - {}".format(end\_time))

## Экранные формы с примерами выполнения программы



# **Задача №7**

## Описание задачи

В предыдущих задачах были написаны все требуемые инструменты для работы с данными. Применим их на реальном примере.

В файле 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  
from print\_result import print\_result  
from cm\_timer import cm\_timer\_1  
from unique import Unique  
from field import field  
from gen\_random import gen\_random  
  
path = "D:\учеба\бкит\Лаба3\lab\_python\_fp\data\_light.json"  
  
# Необходимо в переменную path сохранить путь к файлу, который был передан при запуске сценария  
  
with open(path, 'r', encoding='utf8') 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(field(arg, 'job-name')))  
  
@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):  
 salary = gen\_random(len(arg), 100000, 200000)  
 return list(zip(arg, salary))  
  
  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 with cm\_timer\_1():  
 f4(f3(f2(f1(data))))

## Экранные формы с примерами выполнения программы

