**Московский государственный технический университет им. Н.Э. Баумана.**

Факультет «Информатика и управление»

Кафедра ИУ5 «Системы обработки информации и управления»

Курс «Базовые компоненты интернет-технологий»

Отчет по лабораторной работе № 3

«Функциональные возможности языка Python»

Выполнил: Проверил:

студент группы ИУ5-35Б преподаватель каф. ИУ5  
 Акулова Анастасия Алексеевна Гапанюк Юрий Евгеньевич

Подпись и дата: Подпись и дата:

Москва, 2021 г.

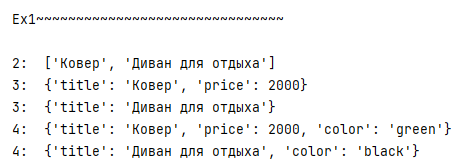
Задание. Задание лабораторной работы состоит из решения нескольких задач.

Файлы, содержащие решения отдельных задач, должны располагаться в пакете lab\_python\_fp. Решение каждой задачи должно располагаться в отдельном файле. При запуске каждого файла выдаются тестовые результаты выполнения соответствующего задания.

# Задача 1

Необходимо реализовать генератор field. Генератор field последовательно выдает значения ключей словаря.

def Solution() -> None:  
 goods = [  
 {'title': 'Ковер', 'price': 2000, 'color': 'green'},  
 {'title': 'Диван для отдыха', 'color': 'black'}  
 ]  
 for good in Field(goods):  
 print('1: ', good)  
 for good in Field(goods, 'title'):  
 print('2: ', good)  
 for good in Field(goods, 'title', 'price'):  
 print('3: ', good)  
 for good in Field(goods, 'title', 'price', 'color'):  
 print('4: ', good)  
  
  
def Field(goods, \*keys):  
 if (len(keys) > 1):  
 for good in goods:  
 yield GetLocalDicts(good, keys)  
 else:  
 for key in keys:  
 yield GetLocalParams(goods, key)  
  
  
def GetLocalDicts(good, keys) -> dict:  
 result = dict()  
 for key in keys:  
 result[key] = good.get(key)  
 result = {key: filter\_ for key, filter\_ in result.items() if filter\_ != None}  
 return result  
  
  
def GetLocalParams(goods, key) -> list:  
 result = list()  
 for good in goods:  
 result.append(good.get(key))  
 result = [filter\_ for filter\_ in result if filter\_ != None]  
 return result



# Задача 2

Необходимо реализовать генератор gen\_random(количество, минимум, максимум),

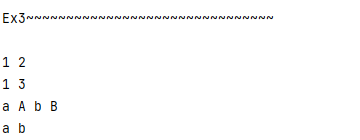
который последовательно выдает заданное количество случайных чисел в заданном диапазоне от минимума до максимума, включая границы диапазона.

import random  
  
def Solution() -> None:  
 print( Gen\_random(5, 1 ,3) )  
  
def Gen\_random(count, min, max) -> list:  
 result = list()  
 random.seed()  
 for i in range(0, count):  
 result.append( random.randint(min, max) )  
 return result



# Задача 3

from py\_lab\_3.Ex2 import Gen\_random  
  
  
def Solution():  
 data = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]  
 test(data)  
  
 data = Gen\_random(10, 1, 3)  
 test(data)  
  
 data = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']  
 test(data)  
 test(data, ignoreCase=True)  
  
  
def test(data, ignoreCase=False):  
 output = str()  
 for item in Unique(data, ignore\_case=ignoreCase):  
 output += str(item) + " "  
 print(output)  
  
  
class Unique:  
  
 def \_\_init\_\_(self, items, \*\*kwargs):  
 self.\_\_ignore\_case = bool(False)  
 for key, value in kwargs.items():  
 if (key == 'ignore\_case'):  
 self.\_\_ignore\_case = bool(value)  
 self.\_\_items\_set = set()  
 self.\_\_gen\_item = self.gen\_item(items)  
  
 def \_\_iter\_\_(self):  
 return self  
  
 def \_\_next\_\_(self):  
 if (self.\_\_ignore\_case):  
 return self.\_\_next\_ignore\_case()  
 else:  
 return self.\_\_next()  
  
 def \_\_next(self):  
 item = next(self.\_\_gen\_item)  
 while item in self.\_\_items\_set:  
 item = next(self.\_\_gen\_item)  
 self.\_\_items\_set.add(item)  
 return item  
  
 def \_\_next\_ignore\_case(self):  
 item = next(self.\_\_gen\_item).lower()  
 while item in self.\_\_items\_set:  
 item = next(self.\_\_gen\_item).lower()  
 self.\_\_items\_set.add(item)  
 return item  
  
 def gen\_item(self, items):  
 for item in items:  
 yield item



# Задача 4

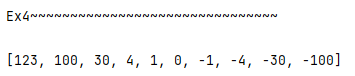
Дан массив 1, содержащий положительные и отрицательные числа. Необходимо одной строкой кода вывести на экран массив 2, которые содержит значения массива 1, отсортированные по модулю в порядке убывания. Сортировку необходимо осуществлять с помощью функции sorted.

Необходимо решить задачу двумя способами:

1. С использованием lambda-функции.

2. Без использования lambda-функции.

def Solution():  
 data = [4, -30, 30, 100, -100, 123, 1, 0, -1, -4]  
 data = sorted(data, reverse = True)  
 data = sorted(data, key = lambda x : -x)  
 print(data)

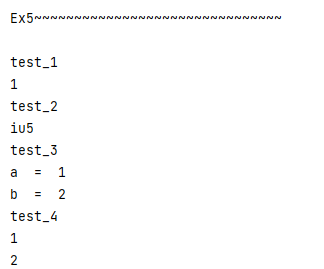


# Задача 5

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.

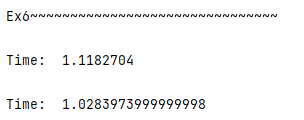
def Solution() -> None:  
 test\_1()  
 test\_2()  
 test\_3()  
 test\_4()  
  
def print\_result(func):  
 def wrapper(\*args):  
 data = func(\*args)  
 print(func.\_\_name\_\_)  
 if type(data) == type(list()):   
 for i in data : print(i)  
 elif type(data) == type(dict()):  
 for key, value in data.items() : print(key, ' = ', value)  
 else:  
 print(data)  
 return data  
 return wrapper  
  
@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]



# Задача 6

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

import time  
from contextlib import contextmanager  
  
*#Почему такая большая разница между вариантами реализации?*def Solution():  
 with cm\_timer1():  
 time.sleep(1)  
  
 print()  
  
 with cm\_timer2():  
 time.sleep(1)  
  
class cm\_timer1:  
 \_\_start = float()  
  
 def \_\_init\_\_(self):  
 self.\_\_start = 0  
  
 def \_\_enter\_\_(self):  
 \_\_start = time.perf\_counter()  
  
 def \_\_exit\_\_(self, \*tmp):  
 dtime = time.perf\_counter() - self.\_\_start  
 print("Time: ", dtime)  
  
@contextmanager  
def cm\_timer2():  
 start = time.perf\_counter()  
 yield  
 dtime = time.perf\_counter() - start  
 print("Time: ", dtime)



# Задача 7

* 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.

import json  
from py\_lab\_3.Ex3 import Unique  
from py\_lab\_3.Ex2 import Gen\_random  
from py\_lab\_3.Ex5 import print\_result  
  
def Solution():  
 data = None  
 with open(r"C:\Users\Анастасия\Downloads\PyLab\_3 (1)\PyLab\_3\PyLab\_3\py\_lab\_3\data\_light.json", encoding = "UTF8", mode = "r") as js:  
 data = json.load(js)  
 f4(f3(f2(f1(data))))  
  
@print\_result  
def f1(data):  
 return sorted(list(Unique([elem["job-name"] for elem in data], ignore\_case = True)), key = lambda x : x.lower())  
  
@print\_result  
def f2(jobs):  
 return list(filter(lambda job : job.lower().startswith("программист"), jobs))  
  
@print\_result  
def f3(jobs):  
 return list(map(lambda job : job + " с опытом Python ", jobs))  
  
@print\_result  
def f4(jobs):  
 return [job[0] + str(job[1]) for job in zip(jobs, Gen\_random(len(jobs), 100000, 200000))]



