**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»**

# **Курс «Базовые компоненты интернет-технологий»**

## Отчёт по лабораторной работе №3-4

## «Функциональные возможности языка Python.»

Выполнил:

студент группы ИУ5-33Б Кузнецов В. А.

подпись: , дата:

Проверил:

лектор Гапанюк Ю. Е.

подпись: , дата:

2022 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.

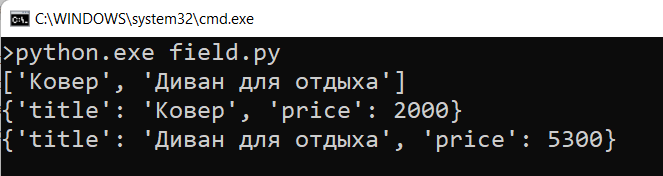
В качестве первого аргумента генератор принимает список словарей, дальше через \*args генератор принимает неограниченное количествово аргументов.

* 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 item in items:  
 value = item.get(args[0])  
 if value is not None:  
 yield value  
 return  
 for item in items:  
 selected = {}  
 for key in args:  
 value = item.get(key)  
 if value is not None:  
 selected[key] = value  
 yield selected  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 goods = [  
 {'title': 'Ковер', 'price': 2000, 'color': 'green'},  
 {'title': 'Диван для отдыха', 'price': 5300, 'color': 'black'}  
 ]  
 print(list(field(goods, 'title')))  
 for line in field(goods, 'title', 'price'):  
 print(line)

Пример выполнения



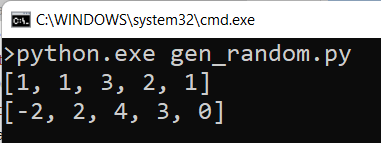
### Задача 2 (файл gen\_random.py)

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

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

Пример выполнения



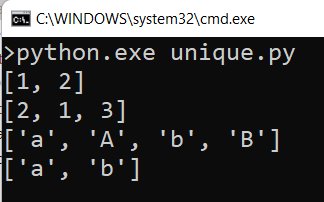
### Задача 3 (файл unique.py)

* 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

from lab\_python\_fp.gen\_random import gen\_random  
  
  
class Unique(object):  
 def \_\_init\_\_(self, items, \*\*kwargs):  
 self.ignore\_case = kwargs.get('ignore\_case', False)  
 self.iter = iter(items)  
 self.used = set()  
  
 def \_\_next\_\_(self):  
 for current in self.iter:  
 if (current.lower() if self.ignore\_case else current) not in self.used:  
 self.used.add(current)  
 return current  
 raise StopIteration  
  
 def \_\_iter\_\_(self):  
 return self  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 print(list(Unique([1, 1, 1, 1, 1, 2, 2, 2, 2, 2])))  
 print(list(Unique(gen\_random(10, 1, 3))))  
 print(list(Unique(['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B'])))  
 print(list(Unique(['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B'], ignore\_case=True)))

Пример выполнения



### 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.

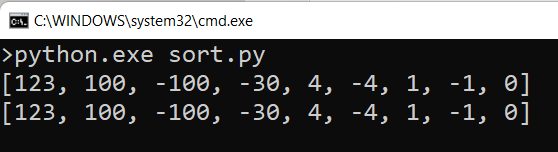
Необходимо решить задачу двумя способами:

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

Текст программы

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 a: a if a > 0 else -a, reverse=True)  
 print(result\_with\_lambda)

Пример выполнения



### Задача 5 (файл print\_result.py)

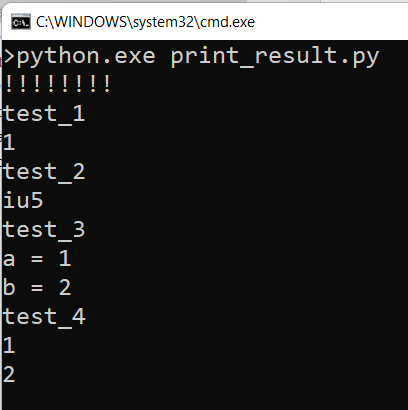
Необходимо реализовать декоратор print\_result, который выводит на экран результат выполнения функции.

* 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):  
 original\_func = func  
  
 def new\_func(\*args, \*\*kwargs):  
 print(original\_func.\_\_name\_\_)  
 result = original\_func(\*args, \*\*kwargs)  
 if isinstance(result, dict):  
 for key, item in result.items():  
 print(f'{key} = {item}')  
 elif isinstance(result, list):  
 for item in result:  
 print(item)  
 else:  
 print(result)  
 return result  
 return new\_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()

Пример выполнения



### Задача 6 (файл cm\_timer.py)

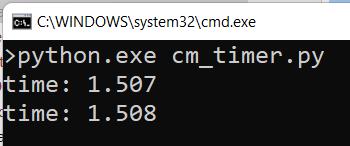
Необходимо написать контекстные менеджеры cm\_timer\_1 и cm\_timer\_2, которые считают время работы блока кода и выводят его на экран.

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

from time import perf\_counter, sleep  
from contextlib import contextmanager  
  
  
class cm\_timer\_1:  
 def \_\_init\_\_(self):  
 self.start = None  
  
 def \_\_enter\_\_(self):  
 self.start = perf\_counter()  
  
 def \_\_exit\_\_(self, exp\_type, exp\_value, traceback):  
 print('time: {:.3f}'.format(perf\_counter() - self.start))  
  
  
@contextmanager  
def cm\_timer\_2():  
 start = perf\_counter()  
 yield  
 print('time: {:.3f}'.format(perf\_counter() - start))  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 with cm\_timer\_1():  
 sleep(1.5)  
 with cm\_timer\_2():  
 sleep(1.5)

Пример 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  
from lab\_python\_fp.field import field  
from lab\_python\_fp.gen\_random import gen\_random  
from lab\_python\_fp.unique import Unique  
from lab\_python\_fp.print\_result import print\_result  
from lab\_python\_fp.cm\_timer import cm\_timer\_1  
  
  
path = r'...\data\_light.json'

with open(path, encoding='UTF-8') as f:  
 data = json.load(f)  
  
  
# @print\_result  
def f1(arg):  
 return sorted(Unique(field(arg, 'job-name'), ignore\_case=True), key=lambda s: s.lower())  
  
  
# @print\_result  
def f2(arg):  
 return list(filter(lambda s: s[:11].lower() == 'программист', arg))  
  
  
# @print\_result  
def f3(arg):  
 return list(map(lambda s: s + 'с опытом Python', arg))  
  
  
@print\_result  
def f4(arg):  
 return [f'{prof}, зарплата {salary} руб' for prof, salary in zip(arg, gen\_random(len(arg), 100000, 200000))]  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 with cm\_timer\_1():  
 f4(f3(f2(f1(data))))

Пример выполнения

