*Государственное образовательное учреждение высшего профессионального образования*



***«***

***Московский***

***государственный***

***технический***

***университет***

***имени***

***Н***

***.***

***Э***

***.***

***Баумана***

***»***

***(***

***МГТУ***

***им***

***.***

***Н***

***.***

***Э***

***.***

***Баумана***

***)***

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Системы обработки информации и управления» (ИУ-5)

**ЛАБОРАТОРНАЯ РАБОТА**

**№ 3-****4**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_Функциональные возможности языка Python\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Группа ИУ5-35Б

Студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ **16.12.2024** /**Д.Е. Мушкарин /**

(Подпись, дата) (И.О.Фамилия)

Преподаватель \_\_\_\_\_\_\_\_\_\_\_ **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** /**Ю. Е. Гапанюк/**

(Подпись, дата) (И.О.Фамилия)

2024

### **Общее 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:**

**Необходимо реализовать генератор field. Генератор field последовательно выдает значения ключей словаря. Пример:**

goods = [

{'title': 'Ковер', 'price': 2000, 'color': 'green'},

{'title': 'Диван для отдыха', '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'}

* 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.

**Код:**

**field.py**

# Пример:

goods = [

{'title': 'Ковер', 'price': 2000},

{'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

for i in items:

if (len(args)>1):

field\_dict = {}

for j in args:

res = i.get(j)

if (res is not None):

field\_dict[j] = res

if field\_dict:

yield field\_dict

else:

res = i.get(args[0])

if res:

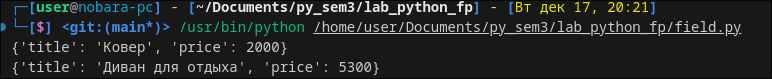
yield res

if \_\_name\_\_ == "\_\_main\_\_":

for i in field(goods,"title","price"):

print(i)

**Вывод:**



**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. Пример:**

gen\_random(5, 1, 3) должен выдать 5 случайных чисел в диапазоне от 1 до 3, например 2, 2, 3, 2, 1

**Код:**

**gen\_random.py**

from random import randint

# Пример:

# gen\_random(5, 1, 3) должен выдать выдать 5 случайных чисел

# в диапазоне от 1 до 3, например 2, 2, 3, 2, 1

# Hint: типовая реализация занимает 2 строки

def gen\_random(num\_count, begin, end):

for \_ in range(num\_count):

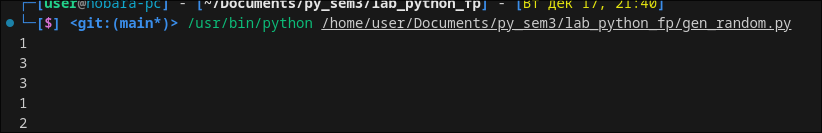
yield randint(begin,end)

if \_\_name\_\_ == "\_\_main\_\_":

for i in gen\_random(5,1,3):

print(i)

**Вывод:**

**Задача 3:**

**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.

Пример:

data = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]

Unique(data) будет последовательно возвращать только 1 и 2.

data = gen\_random(10, 1, 3)

Unique(data) будет последовательно возвращать только 1, 2 и 3.

data = [‘a’, ‘A’, ‘b’, ‘B’, ‘a’, ‘A’, ‘b’, ‘B’]

Unique(data) будет последовательно возвращать только a, A, b, B.

Unique(data, ignore\_case=True) будет последовательно возвращать только a, b.

**Код:**

**unique.py**

# Итератор для удаления дубликатов

from gen\_random import gen\_random

class Unique(object):

def \_\_init\_\_(self, items, \*\*kwargs):

ignore\_case = kwargs.get("ignore\_case", False)

self.current = 0

if ignore\_case:

temp = set()

self.\_items = []

for i in items:

if isinstance(i,str):

if i.lower() not in temp:

temp.add(i.lower())

self.\_items.append(i)

else:

if i not in temp:

temp.add(i)

self.\_items.append(i)

else:

self.\_items = list({i for i in items})

def \_\_iter\_\_(self):

return self

def \_\_next\_\_(self):

if self.current >= len(self.\_items):

raise StopIteration

item = self.\_items[self.current]

self.current+=1

return item

if \_\_name\_\_ == "\_\_main\_\_":

data = [i for i in "waterfallF"] \* 5

for i in Unique(gen\_random(100,1,5)):

print(i, end=" ")

print("\n\_\_\_\_\_\_\_\_\_\_\_\_")

for i in Unique(data):

print(i, end=" ")

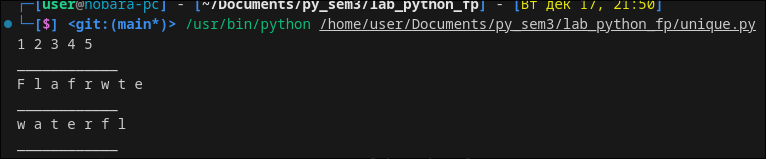
print("\n\_\_\_\_\_\_\_\_\_\_\_\_")

for i in Unique(data, ignore\_case = True):

print(i,end=" ")

print("\n\_\_\_\_\_\_\_\_\_\_\_\_")

**Вывод:**

 **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. Пример:**

data = [4, -30, 30, 100, -100, 123, 1, 0, -1, -4]

Вывод: [123, 100, -100, -30, 30, 4, -4, 1, -1, 0]

Необходимо решить задачу двумя способами:

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

**Код:**

**sort.py**

from cm\_tymer import cm\_timer\_2

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)

**Вывод:**

**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.

**Код:  
print\_result.py**

def print\_result(function):

def wrapper(\*args, \*\*kwargs):

result = function(\*args, \*\*kwargs)

print(function.\_\_name\_\_)

if isinstance(result, list):

for i in result:

print(i)

elif isinstance(result, dict):

for i in result:

print(f"{i}={result[i]}")

else:

print(result)

return result

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]

if \_\_name\_\_ == '\_\_main\_\_':

print('!!!!!!!!')

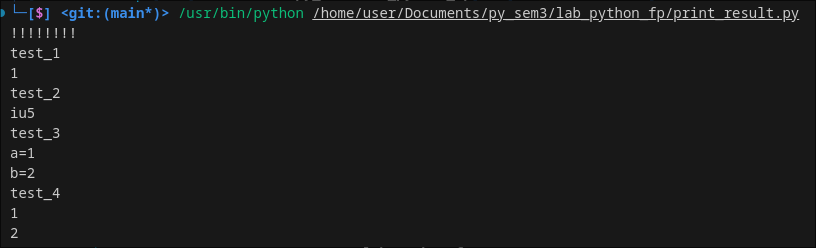
test\_1()

test\_2()

test\_3()

test\_4()

### **Вывод:**



**Задача 6:**

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).

### **Код: cm\_timer.py**

import time

from contextlib import contextmanager

class cm\_timer\_1:

def \_\_enter\_\_(self):

self.start\_time = time.time()

return self

def \_\_exit\_\_(self, exc\_type, exc\_val, exc\_tb):

end\_time = time.time()

elapsed\_time = end\_time - self.start\_time

print(f"time: {elapsed\_time:.10f}")

@contextmanager

def cm\_timer\_2():

start\_time = time.time()

try:

yield

finally:

end\_time = time.time()

elapsed\_time = end\_time - start\_time

print(f"time: {elapsed\_time:.10f}")

if \_\_name\_\_ == '\_\_main\_\_':

from time import sleep

with cm\_timer\_2():

sleep(3.5)

with cm\_timer\_1():

sleep(3.5)

**Вывод:**



### **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.

**Код:**

### **process\_data.py**

import json

import sys

from print\_result import print\_result

from cm\_tymer import cm\_timer\_1

from time import sleep

from unique import Unique

from gen\_random import gen\_random

# Сделаем другие необходимые импорты

path = "lab2/data/data\_light.json"

# Необходимо в переменную path сохранить путь к файлу, который был передан при запуске сценария

with open(path) 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 [i for i in Unique(list(j["job-name"] for j in arg),ignore\_case = True)]

@print\_result

def f2(arg):

return [i for i in filter(lambda x: x.lstrip()[:11].lower() == "программист", arg)]

@print\_result

def f3(arg):

return list(map(lambda x: x + " с опытом Python", arg))

@print\_result

def f4(arg):

pairs = zip(arg,gen\_random(len(arg),100000,200000))

return list(i+" " + str(j) for i,j in pairs)

if \_\_name\_\_ == '\_\_main\_\_':

with cm\_timer\_1():

f4(f3(f2(f1(data))))

**Вывод:**\* результат f1 включает ровно 1000 строк.

