**Московский государственный технический**

**университет им. Н.Э. Баумана**

Факультет «Информатика и системы управления»

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 ИУ5-34Б |  |  |
| Свечникова Анна |  |  |
| Подпись и дата: |  | Подпись и дата: |

Москва, 2021г.

**Постановка задачи**

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.

Шаблон для реализации генератора:

### 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. Пример:

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

### 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.

Пример:

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

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

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

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.

### 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. 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.

### 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.

### 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).

### 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:**

**field.py**

**# Пример:**

# 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

for i in items:

if len(args) > 1:

res = {}

for arg in args:

if arg in i.keys():

if len(args) == 1:

yield i[arg]

else:

res[arg] = i[arg]

if len(args) > 1:

yield res

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

goods = [

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

{'title': 'Диван для отдыха', 'price': 5300, 'color': 'black'}

]

res = field(goods, 'title', 'price')

for i in res:

print(i)

**gen\_random.py**

**# Пример:**

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

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

import random

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

for i in range(num\_count):

yield random.randint(begin, end)

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

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

print(i)

**unique.py**

**# Итератор для удаления дубликатов**

import numbers

class Unique(object):

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

# Нужно реализовать конструктор

# В качестве ключевого аргумента, конструктор должен принимать bool-параметр ignore\_case,

# в зависимости от значения которого будут считаться одинаковыми строки в разном регистре

# Например: ignore\_case = True, Aбв и АБВ - разные строки

# ignore\_case = False, Aбв и АБВ - одинаковые строки, одна из которых удалится

# По-умолчанию ignore\_case = False

self.items = items

if isinstance(self.items, list):

self.items\_len = len(items)

else:

self.items\_len = 0;

self.ignore\_case = kwargs.get('ignore\_case', False)

self.set = set()

self.index = 0

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

elem = None

while True:

if isinstance(self.items, list):

if self.index < self.items\_len:

elem = self.items[self.index]

if isinstance(elem, numbers.Number):

self.ignore\_case = True

self.index += 1

if self.ignore\_case and elem not in self.set:

self.set.add(elem)

return elem

elif not self.ignore\_case and elem.lower() not in self.set:

self.set.add(elem.lower())

return elem

else:

raise StopIteration

else:

elem = next(self.items)

if self.ignore\_case and elem not in self.set:

self.set.add(elem)

return elem

elif not self.ignore\_case and elem.lower() not in self.set:

self.set.add(elem.lower())

return elem

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

return self

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

lst1 = ['a', 'b', 'A', 'B']

lst2 = [1, 2, 1, 2, 3]

for i in Unique(lst1):

print(i)

for i in Unique(lst2):

print(i)

**sort.py**

**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)

**print\_result.py**

# Здесь должна быть реализация декоратора

def print\_result(func):

def decorated\_func(args=None):

print(func.\_\_name\_\_)

if args:

res = func(args)

else:

res = func()

if isinstance(res, dict):

for k, v in res.items():

print('{} = {}'.format(k, v))

elif isinstance(res, list):

for i in res:

print(i)

else:

print(res)

return res

return decorated\_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()

**cm\_timer.py**

**from time import time, sleep**

from contextlib import contextmanager

class cm\_timer\_1:

def \_\_init\_\_(self):

self.begin = None

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

self.begin = time()

def \_\_exit\_\_(self, exp\_type, exp\_value, traceback):

print('time: {}'.format(time() - self.begin))

@contextmanager

def cm\_timer\_2():

begin = time()

yield

print('time: {}'.format(time() - begin))

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

with cm\_timer\_1():

sleep(5.5)

with cm\_timer\_2():

sleep(5.5)

**process\_data.py**

**import json**

import sys

from field import \*

from gen\_random import \*

from unique import \*

from print\_result import \*

from cm\_timer import \*

path = "/home/anna/PycharmProjects/lab\_python\_fp/data\_light.json"

with open(path, 'r') as f:

data = json.load(f)

@print\_result

def f1(x):

return sorted(list(Unique(field(x, "job-name"))), key=str.lower)

@print\_result

def f2(x):

return list(filter(lambda s: s.split()[0].lower() == "программист", x))

@print\_result

def f3(x):

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

@print\_result

def f4(x):

res = list(zip(x, list(gen\_random(len(x),100000,200000))))

return list(map(lambda s: s[0] + ", зарплата " + str(s[1]) + " руб.", res))

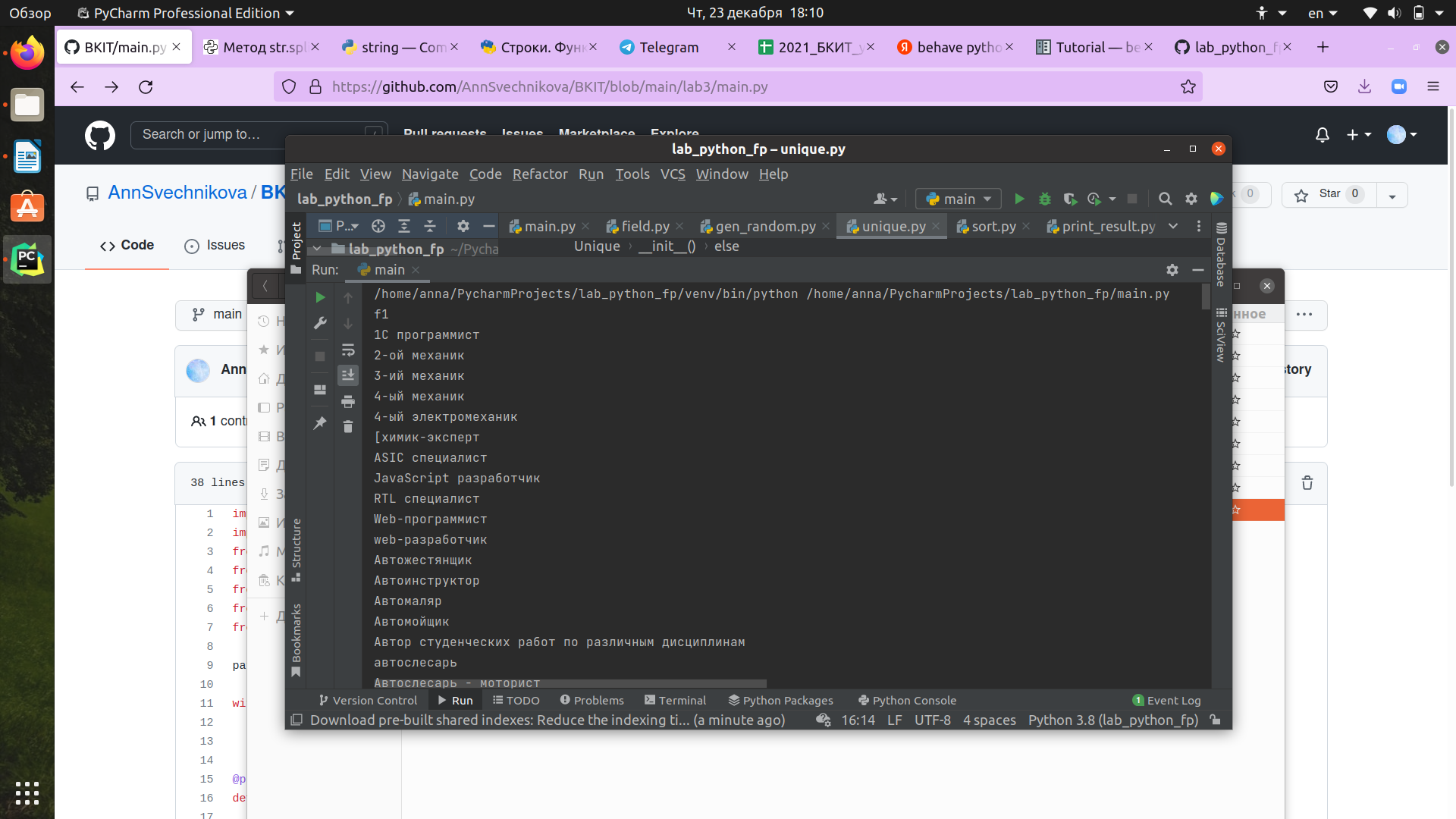
if \_\_name\_\_ == '\_\_main\_\_':

with cm\_timer\_1():

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

**Примеры выполнения**

Фрагмент отсортированного списка профессий:



Результаты выполнения остальных функций:

