**Московский государственный технический**

**университет им. Н.Э. Баумана.**

Факультет «Информатика и управление»

Кафедра ИУ5. Курс «Базовые компоненты интернет-технологий»

Отчет по лабораторной работе №3-4

«**Функциональные возможности языка Python.**»

|  |  |  |
| --- | --- | --- |
| Выполнил: |  | Проверил: |
| студент группы ИУ5-32Б  Казицин Алексей |  | преподаватель кафедры  Гапанюк Ю.Е. |
|  |  |  |
| Подпись и дата: |  | Подпись и дата: |

Москва, 2022 г.

**Цель лабораторной работы:** изучение возможностей функционального программирования в языке Python.

**Требования к отчету:**

Отчет по лабораторной работе должен содержать:

1. титульный лист;
2. общее описание задания;
3. по каждой задаче:
   * описание задачи;
   * текст программы;
   * экранные формы с примерами выполнения программы.

**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. 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.

Шаблон для реализации генератора:

# 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

# Необходимо реализовать генератор

**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

Шаблон для реализации генератора:

# 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):

pass

# Необходимо реализовать генератор

**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(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.

Шаблон для реализации класса-итератора:

# Итератор для удаления дубликатов

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

pass

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

# Нужно реализовать \_\_next\_\_

pass

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

return self

**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.

Шаблон реализации:

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

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

result = ...

print(result)

result\_with\_lambda = ...

print(result\_with\_lambda)

**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.

Шаблон реализации:

# Здесь должна быть реализация декоратора

@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()

Результат выполнения:

test\_1

1

test\_2

iu5

test\_3

a = 1

b = 2

test\_4

1

2

**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.

Шаблон реализации:

import json

import sys

# Сделаем другие необходимые импорты

path = None

# 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):

raise NotImplemented

@print\_result

def f2(arg):

raise NotImplemented

@print\_result

def f3(arg):

raise NotImplemented

@print\_result

def f4(arg):

raise NotImplemented

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

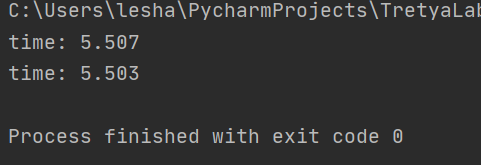
with cm\_timer\_1():

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

Текст программы и ее вывод:

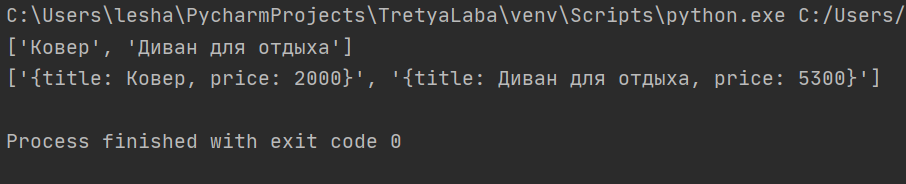
Cm\_timer.py

from contextlib import contextmanager  
import time  
from time import sleep  
  
  
class cm\_timer\_1:  
  
 def \_\_init\_\_(self):  
 self.t = None  
  
 def \_\_enter\_\_(self):  
 self.t = time.time()  
  
 def \_\_exit\_\_(self, exp\_type, exp\_value, traceback):  
 print('time: {:.3f}'.format(time.time() - self.t))  
  
  
@contextmanager  
def cm\_timer\_2():  
 t = time.time()  
 yield  
 print('time: {:.3f}'.format(time.time() - t))  
  
  
def main\_timer():  
 with cm\_timer\_1():  
 sleep(5.5)  
 with cm\_timer\_2():  
 sleep(5.5)  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main\_timer()



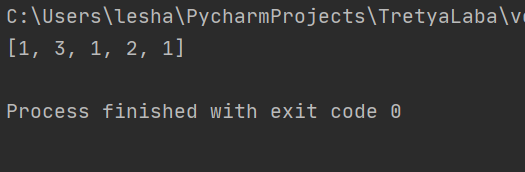
Field.py

goods = [  
 {'title': 'Ковер', 'price': 2000, 'color': 'green'},  
 {'title': 'Диван для отдыха', 'price': 5300, 'color': 'black'}  
]  
  
  
def field(items, \*args):  
 assert len(args) > 0  
 s = ''  
 if len(args) == 1:  
 return (items[j].get(args[0]) for j in range(0, len(items)))  
 else:  
 my\_list = list()  
 for i in range(0, len(items)):  
 s = '{'  
 j = 0  
 for arg in args:  
 if (j == 0):  
 s = s + str(arg) + ': ' + str(items[i].get(arg))  
 else:  
 s = s + ', ' + str(arg) + ': ' + str(items[i].get(arg))  
 j += 1  
 s = s + '}'  
 # print(s)  
 my\_list.append(s)  
 return my\_list  
  
  
def main\_f():  
 l1 = list(field(goods, 'title')) # должен выдавать 'Ковер', 'Диван для отдыха'  
 l2 = field(goods, 'title',  
 'price') # должен выдавать {'title': 'Ковер', 'price': 2000}, {'title': 'Диван для отдыха', 'price': 5300}  
 print(l1)  
 print(l2)  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main\_f()



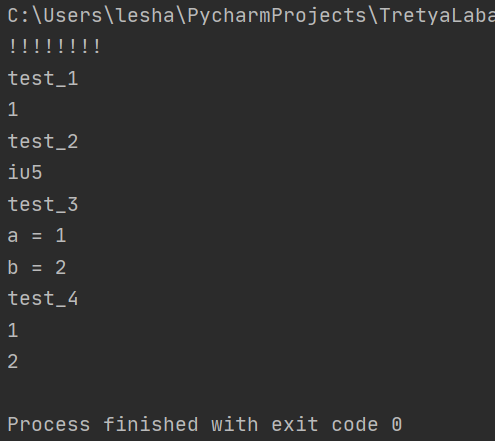
Gen\_random.py

import random  
  
def gen\_random(num\_count, begin, end):  
 for i in range(0, num\_count):  
 yield random.randint(begin, end)  
  
  
def main\_rand():  
 l = list(gen\_random(5, 1, 3))  
 print(l)  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main\_rand()



Print\_result.py

# Здесь должна быть реализация декоратора  
def print\_result(func):  
 def decorated\_func(\*args, \*\*kwargs):  
 res = func(\*args, \*\*kwargs)  
 print(func.\_\_name\_\_)  
 if isinstance(res, list):  
 for i in res:  
 print(i)  
 elif isinstance(res, dict):  
 for key, value in res.items():  
 print('{} = {}'.format(key, value))  
 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()

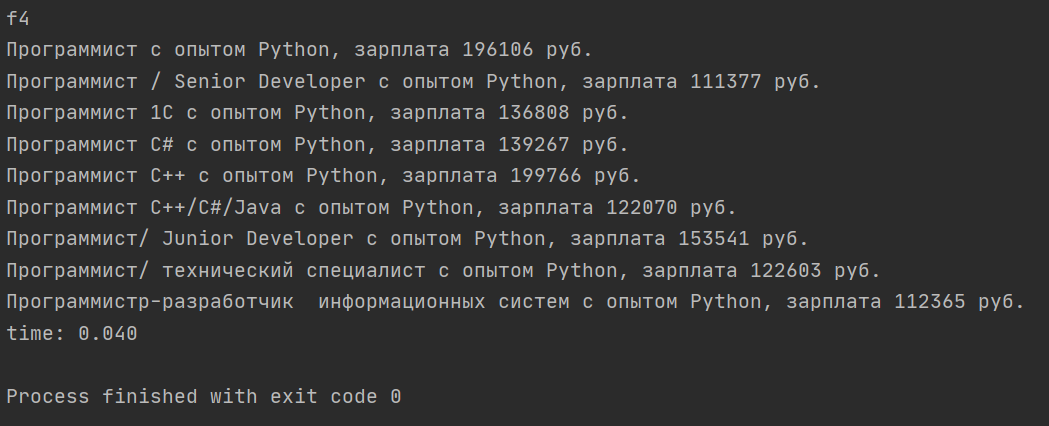


Process\_data.py

import json  
  
# Сделаем другие необходимые импорты  
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'  
  
# Необходимо в переменную path сохранить путь к файлу, который был передан при запуске сценария  
  
with open(path, encoding='UTF-8') 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 sorted(Unique(field(arg, 'job-name'), ignore\_case = True), key = lambda let: let.lower())  
  
@print\_result  
def f2(arg):  
 return list(filter(lambda prog: (prog.lower()).startswith('программист'), arg))  
  
@print\_result  
def f3(arg):  
 return list(map(lambda prog\_py: prog\_py + ' с опытом Python', arg))  
  
  
@print\_result  
def f4(arg):  
 return ['{}, зарплата {} руб.'.format(job, money) for job, money in zip(arg, gen\_random(len(arg), 100000, 200000))]  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 with cm\_timer\_1():  
 f4(f3(f2(f1(data))))

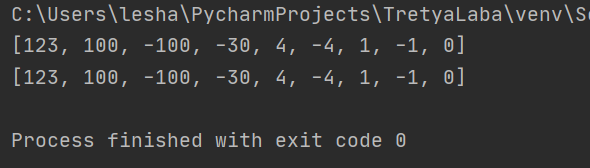
СЛИШКОМ ДЛИННЫЙ ВЫВОД

ПОСЛЕДНЯЯ ЧАСТЬ:



Sort.py

data = [4, -30, 100, -100, 123, 1, 0, -1, -4]  
  
def main\_s():  
 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)  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main\_s()



Unique.py

from lab\_python\_fp.gen\_random import gen\_random  
  
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 = False, Aбв и АБВ - разные строки  
 # ignore\_case = True, Aбв и АБВ - одинаковые строки, одна из которых удалится  
 # По-умолчанию ignore\_case = False  
 self.used\_elements = set()  
 self.iterator = iter(items)  
 self.ignore\_case = False  
 if 'ignore\_case' in kwargs:  
 self.ignore\_case = kwargs['ignore\_case']  
  
 def \_\_next\_\_(self):  
 # Нужно реализовать \_\_next\_\_  
 while True:  
 try:  
 current = next(self.iterator)  
 except StopIteration:  
 raise StopIteration  
 else:  
 cur\_line = str(current)  
 if self.ignore\_case == True:  
 if cur\_line.lower() not in self.used\_elements:  
 self.used\_elements.add(cur\_line.lower())  
 return current  
 else:  
 if cur\_line not in self.used\_elements:  
 self.used\_elements.add(cur\_line)  
 return current  
  
 def \_\_iter\_\_(self):  
 return self  
  
  
def main\_u():  
 # будет последовательно возвращать только 1 и 2  
 data = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]  
 for i in Unique(data):  
 print(i, end = " ")  
 print()  
 # будет последовательно возвращать только 1, 2 и 3  
 data = gen\_random(10, 1, 3)  
 for i in Unique(data):  
 print(i, end = " ")  
 print()  
 # будет последовательно возвращать только a, A, b, B  
 data = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']  
 for i in Unique(data):  
 print(i, end = " ")  
 print()  
 # будет последовательно возвращать только a, b  
 for i in Unique(data, ignore\_case = True):  
 print(i, end = " ")  
 print()  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main\_u()

