**Московский государственный технический**

**университет им. Н.Э. Баумана**

Факультет «Информатика и системы управления»

Кафедра ИУ5 «Системы обработки информации и управления»

Курс

«**Парадигмы и конструкции языков программирования**»

**Отчет по лабораторной работе №3-4**

**«**[**Функциональные возможности языка Python**](https://github.com/ugapanyuk/BKIT_2022/wiki/lab_python_fp)**»**

|  |  |  |
| --- | --- | --- |
| **Выполнил:** |  | **Проверил:** |
| студент группы ИУ5-31Б |  | преподаватель каф. ИУ5 |
| Санников Н.А. |  |  |
|  |  |  |

Москва, 2024 г.

**Цель лабораторной работы**

Изучение возможностей функционального программирования в языке Python.

**Описание задания**

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 (файл field.py)**

***Задание:***

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

***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

else:

for item in items:

result = {arg: item.get(arg) for arg in args if item.get(arg) is not None}

if result:

yield result

# Тестирование

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

goods = [

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

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

]

print(list(field(goods, 'title')))

print(list(field(goods, 'title', 'price')))

#

***экранные формы с примерами выполнения программы:***

******

**Задача 2 (файл gen\_random.py)**

***Задание:***

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

***Текст программы:***

import random

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

for \_ in range(num\_count):

yield random.randint(begin, end)

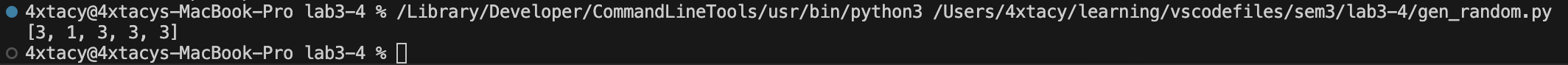
# Тестирование

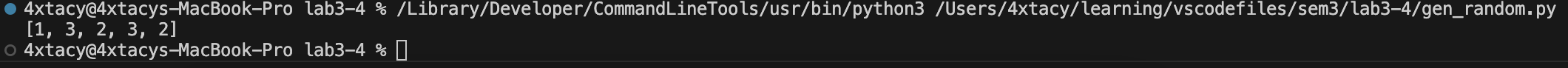
if \_\_name\_\_ == "\_\_main\_\_":

print(list(gen\_random(5, 1, 3)))

#

***экранные формы с примерами выполнения программы:***





**Задача 3 (файл unique.py)**

***Задание:***

* 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

***Текст программы:***

class Unique(object):

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

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

self.seen = set()

self.items = iter(items)

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

return self

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

while True:

item = next(self.items)

key = item.lower() if self.ignore\_case and isinstance(item, str) else item

if key not in self.seen:

self.seen.add(key)

return item

# Тестирование

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

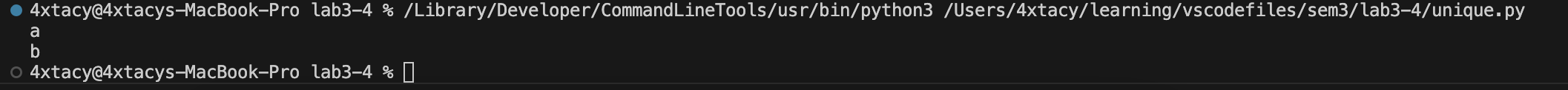
data = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']

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

print(item)

#

***экранные формы с примерами выполнения программы:***



**Задача 4 (файл sort.py)**

***Задание:***

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)

***Текст программы:***

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

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

result = sorted(data, key=abs, reverse=True)

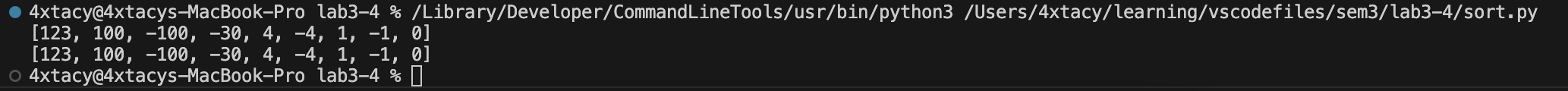
print(result)

result\_with\_lambda = sorted(data, key=lambda x: abs(x), reverse=True)

print(result\_with\_lambda)

#

***экранные формы с примерами выполнения программы:***



**Задача 5 (файл print\_result.py)**

***Задание:***

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

***Текст программы:***

def print\_result(func):

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

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

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

if isinstance(result, list):

for item in result:

print(item)

elif isinstance(result, dict):

for key, value in result.items():

print(f"{key} = {value}")

else:

print(result)

return result

return wrapper

# Тестирование

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

@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]

test\_1()

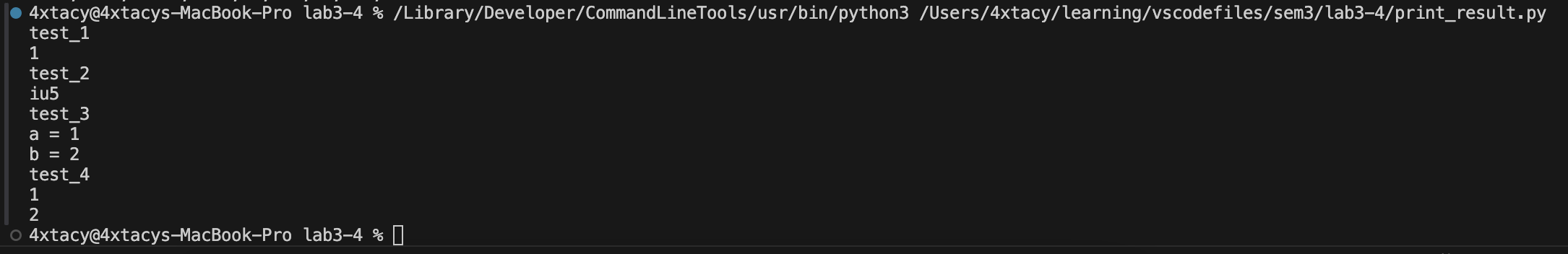
test\_2()

test\_3()

test\_4()

#

***экранные формы с примерами выполнения программы:***



**Задача 6 (файл cm\_timer.py)**

***Задание:***

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).

***Текст программы:***

import time

from contextlib import contextmanager

class cm\_timer\_1:

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

self.start\_time = time.time()

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

print(f"time: {time.time() - self.start\_time}")

@contextmanager

def cm\_timer\_2():

start\_time = time.time()

yield

print(f"time: {time.time() - start\_time}")

# Тестирование

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

with cm\_timer\_1():

time.sleep(1)

with cm\_timer\_2():

time.sleep(1)

#

***экранные формы с примерами выполнения программы:***

Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание

**Задача 7 (файл process\_data.py)**

***Задание:***

* 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:***

import json

from print\_result import print\_result

from cm\_timer import cm\_timer\_1

from gen\_random import gen\_random

from unique import Unique

path = 'data\_light.json'

with open(path) as f:

data = json.load(f)

@print\_result

def f1(arg):

return sorted(Unique([job['job-name'].lower() for job in arg if 'job-name' in job]), key=lambda x: x.lower())

#извлекает значения по ключу 'job-name', приводит их к нижнему регистру, удаляет дубликаты и сортирует в алфавитном порядке.

@print\_result

def f2(arg):

# Фильтрация профессий, которые начинаются со слова "программист"

return list(filter(lambda x: x.startswith('программист'), arg))

@print\_result

def f3(arg):

# Добавление строки "с опытом Python" к каждой профессии

return list(map(lambda x: f"{x} с опытом Python", arg))

@print\_result

def f4(arg):

# Генерация зарплаты и объединение с названием профессии

salaries = gen\_random(len(arg), 100000, 200000)

return list(map(lambda x, y: f"{x}, зарплата {y} руб.", arg, salaries))

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

with cm\_timer\_1():

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

#

***экранные формы с примерами выполнения программы:***

Изображение выглядит как текст, снимок экрана, меню

Автоматически созданное описание