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 лабораторным работам №3-4

«Функциональные возможности языка Python»

Выполнил:

студент группы ИУ5-32Б: Кузьмин Я.А.

Руководитель:

преподаватель каф. ИУ5 Гапанюк Ю.Е.

Москва, 2022 г.

**Цель лабораторной работы:**

Изучение возможностей функционального программирования в языке Python.

**Задание лабораторной работы:** 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.

**Задание №1:**

Необходимо реализовать генератор field. Генератор field последовательно выдает значения ключей словаря. 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 количество 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: lab\_python\_fp/field.py

# Пример:

# 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 if len(args) == 1: arg = args[0]

return [ dict[arg] for dict in items if arg in dict and dict[arg] != None] else:

res = [{ arg:dict[arg] for arg in args if arg in dict and dict[arg] != None} for dict in items ]

return list(filter(lambda dict: len(dict) != 0, res))

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

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

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

]

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

Примеры выполнения:



**Задание №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.

Текст программы: lab\_python\_fp/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\_1(num\_count, begin, end):

return (randint(begin, end) for i in range(num\_count))

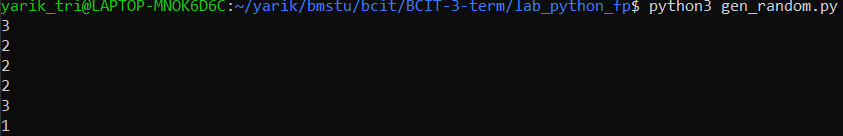
def gen\_random\_2(num\_count, begin, end): for i in range(num\_count): yield randint(begin, end)

gen\_random = gen\_random\_1

if \_\_name\_\_ == "\_\_main\_\_": for i in gen\_random(6, 1, 3):

print(i)

Примеры выполнения:



**Задание №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.

Текст программы: lab\_python\_fp/unique.py

from gen\_random import gen\_random

class Unique(object): def \_\_init\_\_(self, items, \*\*kwargs): self.used\_items = set() self.iterator = iter(items)

if 'ignore\_case' in kwargs and kwargs['ignore\_case'] == True:

self.ignore\_case = True

else:

self.ignore\_case = False

def \_\_next\_\_(self): while True: item = None

try:

item = next(self.iterator) except StopIteration: raise StopIteration

item\_original = item if self.ignore\_case:

item = str(item).lower()

if item not in self.used\_items: self.used\_items.add(item) return item\_original

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

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

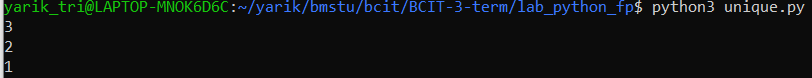
# data = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 'A', 3, 'a', '1567', 1567, 'AAb',

'AAB', 'aAb', 'aab'] data = gen\_random(100, 1, 3)

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

print(i)

Примеры выполнения:



**Задание №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.

Необходимо решить задачу двумя способами:

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

Текст программы: lab\_python\_fp/sort.py

data = [4, -30, 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 i: abs(i), reverse

= True) print(result\_with\_lambda)

Примеры выполнения:



**Задание №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.

Текст программы: lab\_python\_fp/print\_result.py

# Здесь должна быть реализация декоратора def print\_result(func): def wrapper(\*args, \*\*kwargs): print(func.\_\_name\_\_) res = func(\*args, \*\*kwargs)

if isinstance(res, list) or isinstance(res, tuple): 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 wrapper

@print\_result

def test\_1(a, b): return a + b

@print\_result

def test\_2(): return 'iu5'

@print\_result

def test\_3():

return {'a': 1, 'b': 2}

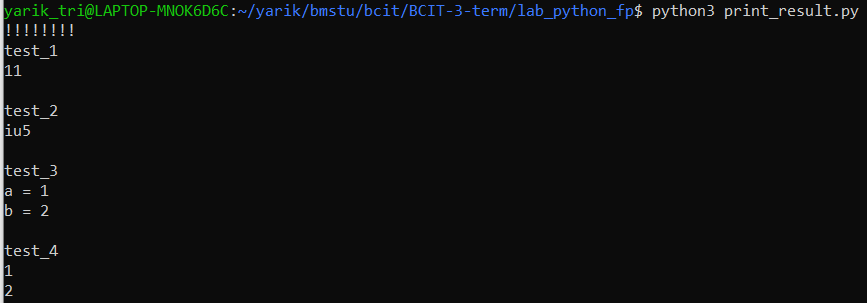
@print\_result

def test\_4(): return [1, 2]

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

print('!!!!!!!!') test\_1(5, 6) print() test\_2() print() test\_3() print() test\_4()

Примеры выполнения:



**Задание №6:**

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

После завершения блока кода в консоль должно вывестись time:

2.5 (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: lab\_python\_fp/cm\_timer.py

import time from contextlib import contextmanager

class cm\_timer\_1: def \_\_init\_\_(self): self.start = 0

def \_\_enter\_\_(self): self.start = time.time()

def \_\_exit\_\_(self, exp\_type, exp\_value, traceback): if exp\_type is not None:

print(exp\_type, exp\_value, traceback)

else:

end = time.time() t = end - self.start print("time = {}".format(t))

@contextmanager

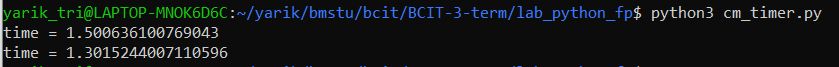
def cm\_timer\_2(): start = time.time() yield None end = time.time()

t = end - start print("time = {}".format(t))

if \_\_name\_\_ == "\_\_main\_\_": with cm\_timer\_1(): time.sleep(1.5)

with cm\_timer\_2(): time.sleep(1.3)

Примеры выполнения:



**Задание №7:**

* В предыдущих задачах были написаны все требуемые инструменты для работы с данными. Применим их на реальном примере.
* В файле **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:

lab\_python\_fp/process\_data.py

import json

import sys

from field import field

from gen\_random import gen\_random from unique import Unique from print\_result import print\_result

from cm\_timer import cm\_timer\_1

try:

path = sys.argv[1] except:

path = "data\_light.json" finally: print(path)

with open(path, 'r', encoding='utf8') as f:

data = json.load(f)

@print\_result def f1(arg):

return sorted([i for i in Unique(field(data, 'job-name'), ignore\_case=True)],

key = lambda x: x.lower())

@print\_result def f2(arg):

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

@print\_result def f3(arg): return list(map(lambda str: str + ' с опытом Python', arg))

@print\_result def f4(arg):

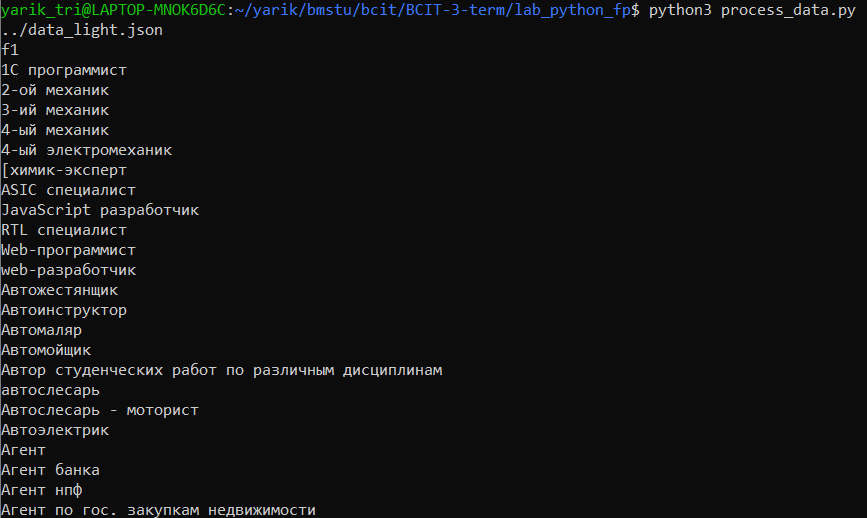
job\_salary = zip(arg, gen\_random(len(arg), 100000, 200000))

return ["{}, зарплата {} руб.".format(job, salary) for job, salary

in job\_salary]

if \_\_name\_\_ == '\_\_main\_\_': with cm\_timer\_1(): f4(f3(f2(f1(data))))

Примеры выполнения:



Далее cледует ещё огромное множество вакансий  
После чего следующий результат функций f2, f3, f4:

