**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 лабораторным работам №3-4

«Функциональные возможности языка Python»

|  |  |  |
| --- | --- | --- |
| Выполнил: |  | Проверил: |
| студент группы ИУ5-32Б |  | преподаватель каф. ИУ5 |
| Гайнуллин А. М. |  | 1. Гапанюк Ю.Е. |
|  |  |  |

**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 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.

Файл field.py

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):

if len(args) == 1:

for item in items:

if args[0] in item and item[args[0]] is not None:

yield item[args[0]]

else:

for item in items:

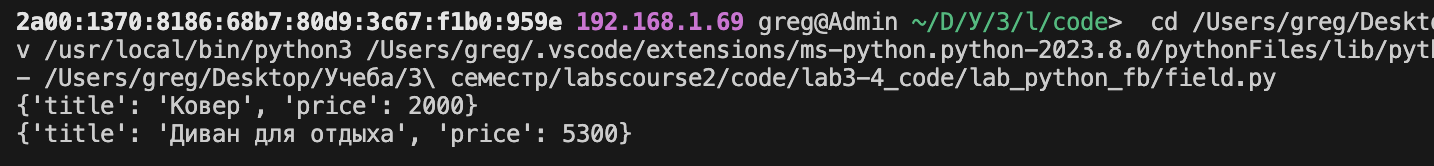
result = {arg: item[arg] for arg in args if arg in item and item[arg] is not None}

if result:

yield result

for item in field(goods, 'title', 'price'):

print(item)

****

**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

Файл gen\_random.py

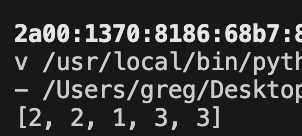
import random

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

rand\_numbers = [random.randint(begin,end) for i in range(num\_count)]

return rand\_numbers

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



**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.

Файл unique.py

from gen\_random import gen\_random

class Unique:

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

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

self.items = list(items)

self.index = 0

self.unique\_items = []

def \_get\_key(self, item):

if self.ignore\_case and isinstance(item, str):

return item.lower()

return item

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

while self.index < len(self.items):

current\_item = self.items[self.index]

key = self.\_get\_key(current\_item)

if key not in self.unique\_items:

self.unique\_items.append(key)

self.index += 1

return current\_item

else:

self.index += 1

raise StopIteration

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

return self

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

for i in Unique(data):

print(i)

print()

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

for i in Unique(data\_random):

print(i)

print()

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

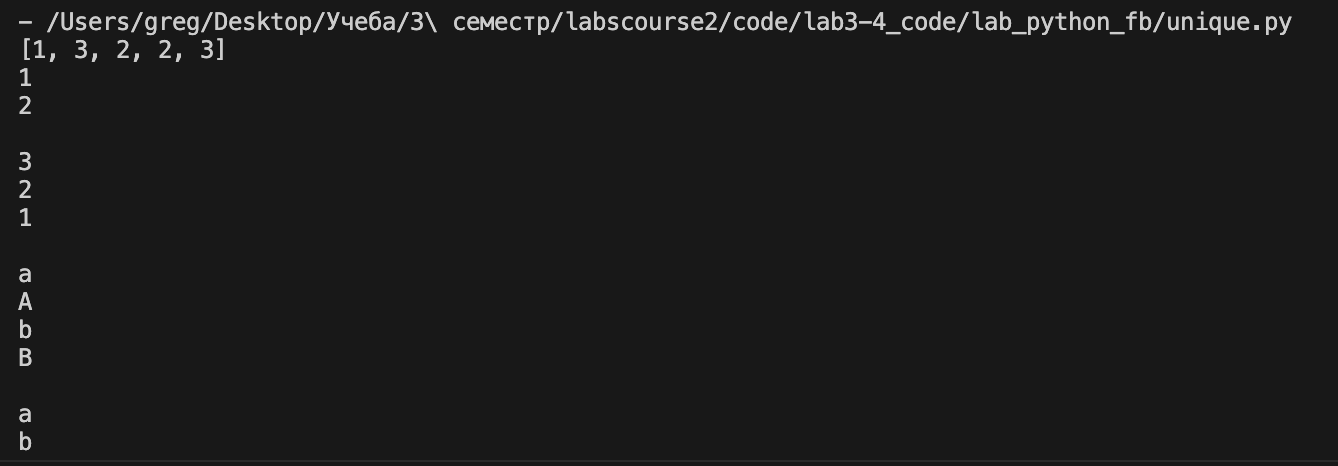
for i in Unique(data\_str):

print(i)

print()

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

print(i)

****

**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.

Файл sort.py

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

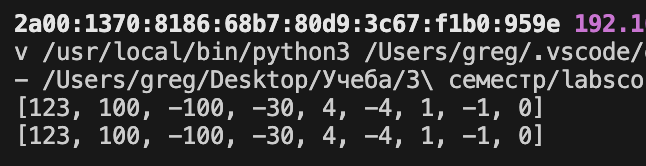
if \_\_name\_\_ == '\_\_main\_\_':

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)

****

**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.

Файл print\_result.py

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

@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\_\_':

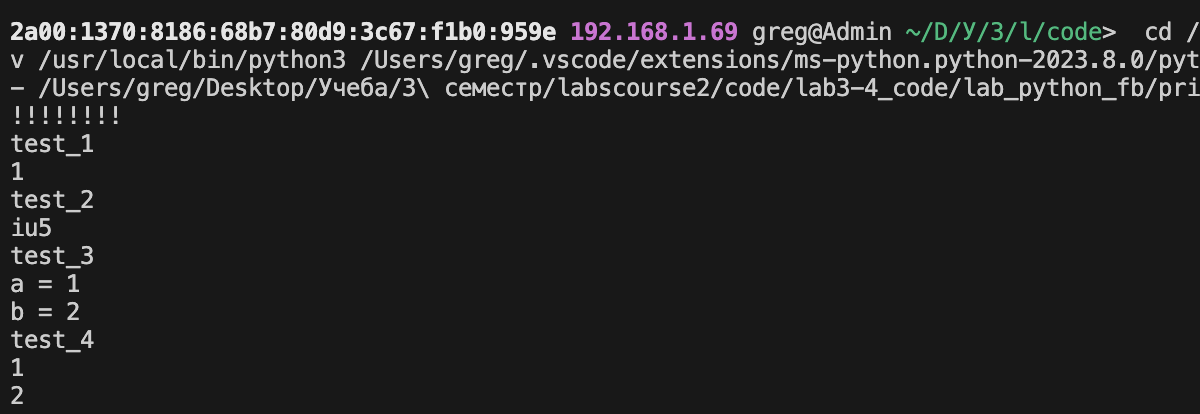
print('!!!!!!!!')

test\_1()

test\_2()

test\_3()

test\_4()



**Задача 6**

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

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).

Файл 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):

self.end\_time = time.time()

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

print(f"time: {cool\_time}")

@contextmanager

def cm\_timer\_2():

start\_time = time.time()

yield

end\_time = time.time()

cool\_time = end\_time - start\_time

print(f"time: {cool\_time}")

# with cm\_timer\_1():

# time.sleep(5.5)

# with cm\_timer\_2():

# time.sleep(5.5)

**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.

Файл process\_data.py

import json

from unique import Unique

from cm\_timer import cm\_timer\_1

from field import field

from print\_result import print\_result

from gen\_random import gen\_random

path = "code/lab3-4\_code/data\_light.json"

with open(path) as f:

data = json.load(f)

def programmist\_filter(data):

return list(filter(lambda x: x.startswith("Программист"), data))

def unique\_things(data):

return list(Unique(field(data,"job-name")))

def add\_python\_exp(data):

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

def add\_salary(data):

salaries = gen\_random(2000,100000,200000)

return [f"{job}, зарплата {salary} руб." for job, salary in zip(data, salaries)]

@print\_result

def f1(arg):

return unique\_things(arg)

@print\_result

def f2(arg):

return programmist\_filter(arg)

@print\_result

def f3(arg):

return add\_python\_exp(arg)

@print\_result

def f4(arg):

return add\_salary(arg)

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

with cm\_timer\_1():

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

