Московский государственный технический

университет им. Н.Э. Баумана

Выполнил:

Студент группы ИУ5-33Б

Соколов Илья Владимирович

Подпись и дата:

Курс «Базовые компоненты интернет-технологий»

Отчёт по лабораторной работе №3-4

Факультет «Информатика и система управления»

Кафедра ИУ5 «Система обработки информации и управления»

Проверил:

Преподаватель каф. ИУ5

Гапанюк Юрий Евгеньевич

Подпись и дата:

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

Задачи:

**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.

Код:

def field(goods,\*args):

    if len(args)!=0:

        for d in goods:

            for key in args:

                if key in d:

                    print(d[key])

    else:

        for d in goods:

            for value in d.values():

                if value!=None:

                    print(value)

**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

Код:

from random import randint

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

    answer=[randint(begin,end) for \_ in range(num\_count) ]

    return answer

**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.

Код:

# Итератор для удаления дубликатов

class Unique(object):

    def \_\_init\_\_(self, data, \*\*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

        '''ignore\_case = True, Aбв и АБВ - разные строки

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

        self.data=data

        self.used\_elements=set()

        self.index=0

        self.ignore\_case=False

        if 'ignore\_case' in kwargs:

            self.ignore\_case=kwargs['ignore\_case']

    def \_\_next\_\_(self):

        # Нужно реализовать \_\_next\_\_

        while True:

            if self.index>=len(self.data):

                raise StopIteration

            else:

                current = self.data[self.index]

                if self.ignore\_case and type(current)==str:

                    current=current.lower()

                self.index+=1

                if current not in self.used\_elements:

                    self.used\_elements.add(current)

                    return current

    def \_\_iter\_\_(self):

        return self

# arr=[1,1,1,2]

# arr\_str= ['a',"A"]

# gen=(i\*i for i in range(10))

# print(type(gen))

# for i in Unique(gen):

#     print(i)

**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.

Код:

def sort(arr):

    return sorted(arr,reverse=True)

def lambda\_sort(arr):

    answer= lambda x: sorted(x,reverse=True)

    return answer(arr)

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

    print(sort(input()))

**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.

Код:

def print\_result(funk):

    def iterated(\*args,\*\*kwargs):

        print(funk.\_\_name\_\_)

        result=funk(\*args)

        if type(result)==tuple or type(result)==list:

            for i in result:

                print(i)

        elif type(result)==dict:

            for i in result:

                print('{} = {} '.format(i,result[i]))

        else:

            print(result)

        return result

    return iterated

**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).

Код:

import time

def cm\_timer(funk\_to\_measure):

    t=time.time()

    def iterable(\*args,\*\*kwargs):

        funk\_to\_measure(\*args,\*\*kwargs)

        print("Время выполнения: ",time.time()-t)

    return iterable

**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.

Код:

from gen\_random import gen\_random

from field import field

from print\_result import print\_result

from unique import Unique

from cm\_timer import cm\_timer

from sort import lambda\_sort

import json

import sys,os

# Сделаем другие необходимые импорты

path = 'data\_light.json'

# Необходимо в переменную path сохранить путь к файлу, который был передан при запуске сценария

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

    data = json.load(f)

#print(os.path.getsize(path))

@print\_result

def f1(data):

    return lambda\_sort(Unique([i['job-name'] for i in data],ignore\_case=False))

d=f1(data)

@print\_result

def f2(data):

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

f2(f1(data))

@print\_result

def f3(data):

    return list(map(lambda x: x+' с опытом Python',data))

@cm\_timer

def f4(data):

    return list(zip(data,gen\_random(len(data),100000,200000)))

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

    f4(f3(f2(f1(data))))

Результат:

