*Государственное образовательное учреждение высшего профессионального образования*

|  |  |
| --- | --- |
|  | ***«Московский государственный технический университет имени Н.Э. Баумана»***  ***(МГТУ им. Н.Э. Баумана)*** |

**ЛАБОРАТОРНАЯ РАБОТА**

**№**  3

Функциональные возможности языка Python

Группа ИУ5-35Б

Студент /\_\_\_\_\_\_\_\_\_\_\_\_\_/\_\_\_\_\_\_\_\_\_Александров А.В\_\_\_\_\_**/**

(Подпись, дата) (И.О.Фамилия)

Преподаватель / **/\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/**

(Подпись, дата) (И.О.Фамилия)

2024

**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 располагаться 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.

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.

#1 ЗАДАНИЕ

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 # проверка на истинность, иначе AssertionError

if len(args) == 1:

for i in items:

result = i[args[0]]

if result != None:

yield result

else:

for item in items:

result = {i:item[i] for i in args if item[i] != None} #создаем словарь, если значение не None

yield result

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

C:\Users\79169\Pictures\Screenshots\Снимок экрана 2024-11-13 231040.png

**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

#2 ЗАДАНИЕ

# gen\_random(5, 1, 3) должен выдать выдать 5 случайных чисел

# в диапазоне от 1 до 3, например 2, 2, 3, 2, 1

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

import random

for i in range(num\_count):

yield random.randint(begin, end)

for num in gen\_random(5, 1, 3):

print(num)

C:\Users\79169\Pictures\Screenshots\Снимок экрана 2024-11-13 231139.png

**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.

#3 ЗАДАНИЕ

# 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

# Итератор для удаления дубликатов

class Unique(object):

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

self.items = items

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

self.Set = set() #для уникальных значений

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

for item in self.items:

to\_low = item.lower() if self.ignore and type(item) == str else item

if to\_low not in self.Set:

self.Set.add(to\_low)

return item

raise StopIteration #для генерации исключений

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

return self

data1 = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]

unique\_data1 = Unique(data1)

print(list(unique\_data1))

data2 = (x for x in gen\_random(10, 1, 3))

unique\_data2 = Unique(data2)

print(list(unique\_data2))

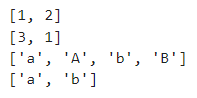
data3 = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']

unique\_data3 = Unique(data3)

print(list(unique\_data3))

unique\_data4 = Unique(data3, ignore\_case=True)

print(list(unique\_data4))



**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.

Необходимо решить задачу двумя способами:

С использованием lambda-функции.

Без использования lambda-функции.

#4 ЗАДАНИЕ

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)

C:\Users\79169\Pictures\Screenshots\Снимок экрана 2024-11-13 231312.png

**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.

#5 ЗАДАНИЕ

def print\_result(func):

def wrapper(\*args, \*\*kwargs):#при добавлении декоратора сначала вызывается wrapper

result = func(\*args, \*\*kwargs)#вызов функции после декоратора

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

if type(result) == list:

for item in result:

print(item)

elif type(result) == dict:

for k, v in result.items():

print(k, " = ", v)

else:

print(result)

return result #возвращаем результат

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\_\_':

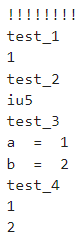
print('!!!!!!!!')

test\_1()

test\_2()

test\_3()

test\_4()



**Задача 6 (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).

#6 ЗАДАНИЕ

#1 cm\_timer\_1 на примере класса

import time

from contextlib import contextmanager

class cm\_timer\_1:

def \_\_enter\_\_(self): #дейтсвия до основного кода

self.start\_time = time.time() #сохраняем время начала

return self

def \_\_exit\_\_(self, exc\_type, exc\_value, traceback): #после выполнения with

#exc\_type - исключение внутри with

#exc\_value - сообщение об ошибке

#traceback показывает, где произошла ошибка

print(time.time() - self.start\_time, "- время выполнения cm\_timer\_1")

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

with cm\_timer\_1():

time.sleep(5.5)

@contextmanager

def cm\_timer\_2():

start\_time = time.time()

try:

yield #до кода выполняется, после передает управление with

finally: #после with возвращаемся в finally

print(time.time() - start\_time, "- время выполнения cm\_timer\_2")

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

with cm\_timer\_2():

time.sleep(5.5)

C:\Users\79169\Pictures\Screenshots\Снимок экрана 2024-11-13 231508.png

**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.

#7 ЗАДАНИЕ

import json

import sys

path = r"D:\Users\79169\Загрузки\data\_light.json"

with open(path, 'r', encoding='utf-8') as f:

data = json.load(f)

@print\_result

def f1(arg):

return sorted(Unique(field(arg, 'job-name')))

@print\_result

def f2(arg):

return list(filter(lambda x: True if x.lower().startswith('программист') else False, arg))

@print\_result

def f3(arg):

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

@print\_result

def f4(arg):

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

return [i + ', зарплата {} руб.'.format(j) for i, j in zip(arg, salary)]

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

with cm\_timer\_1():

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

