Московский государственный технический

университет им. Н.Э. Баумана

Факультет «Радиотехнический»

Кафедра ИУ5 «Информатика и вычислительная техника»

**Курс «Разработка интернет-приложений»**

**Отчет по лабораторной работе № 3**

**«Функциональные возможности языка Python»**

|  |  |  |
| --- | --- | --- |
| Выполнил: |  | Проверил: |
| студент группы РТ5-51 |  | преподаватель |
| Алиев Т.М. |  | Гапанюк Ю. Е. |
| Подпись и дата: |  | Подпись и дата: |

Москва, 2021 г.

Цель работы

Изучение возможностей функционального программирования в языке Python.

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.

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.

*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

*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.

*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:

1) С использованием lambda-функции.

2) Без использования lambda-функции.

*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.

*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.

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).

*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 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:

1. Файл field.py

from \_\_future\_\_ import print\_function

goods = [

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

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

]

def field(items, \*args):

#assert len(args) > 0

if len(args) == 1:

for x in items:

print('\'' + x.get(args[0]) + '\',', end=' ')

if len(args) > 1:

for x in items:

print (end='{')

for y in args:

print('\'' + y + '\': \'' + str(x.get(y)) + '\',', end='')

print (end='}, ')

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

1. Файл gen\_random.py

import random

random.seed(version=2)

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

rand\_list = list()

for x in range(num\_count):

rand\_list.append(random.randint(begin, end))

return rand\_list

1. Файл unique.py

class SimpleIterator:

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

self.items = items

self.counter = 0

self.myLits = list()

if 'ignore\_case' in kwargs:

if (kwargs['ignore\_case']):

self.items = [item.lower() for item in self.items]

seen = {}

self.new\_list = [seen.setdefault(x, x) for x in self.items if x not in seen]

self.limit = len(self.new\_list)

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

return self

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

if self.counter < self.limit:

self.counter += 1

return self.new\_list[self.counter - 1]

else:

raise StopIteration

1. Файл sort.py

def MySort(d):

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

return(result)

def MySortLambda(d):

result\_with\_lambda = sorted(d, key=lambda v: -v if v < 0 else v, reverse=True)

return(result\_with\_lambda)

1. Файл print\_result.py

def print\_result(func):

def wrapper():

print("\n")

print ('название функции: ', func.\_\_name\_\_)

result = func()

print (result)

if isinstance(result, (dict)):

for x in result:

print(x, '=', result.get(x))

if isinstance(result, (list)):

for x in result:

print (x)

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]

1. Файл cm\_timer.py

import time

import contextlib

#Timer 1

class cm\_timer\_1:

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

self.start = time.time()

def \_\_exit\_\_(self, \*args):

self.end = time.time()

self.interval = self.end - self.start

print("time: %0.3f" % (self.interval))

return False

#Timer 2

@contextlib.contextmanager

def cm\_timer\_2():

start = time.time()

yield

end = time.time()

print('time: {:.3f}'.format(end - start))

1. Файл main.py

import json

import gen\_random

import cm\_timer

def print\_result(func):

def wrapper(arg):

print("\n")

print('название функции: ', func.\_\_name\_\_)

result = func(arg)

print(result)

if isinstance(result, (dict)):

for x in result:

print(x, '=', result.get(x))

if isinstance(result, (list)):

for x in result:

print(x)

return result

return wrapper

path = 'data.json'

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

data = json.load(f)

# Функция f1 должна вывести отсортированный список профессий без повторений

# (строки в разном регистре считать равными). Сортировка должна игнорировать регистр.

@print\_result

def f1(arg):

my\_list = list()

for i in arg:

my\_list.append(i['job-name'])

my\_list = [item.lower() for item in my\_list]

seen = {}

my\_list = [seen.setdefault(x, x) for x in my\_list if x not in seen]

my\_list = sorted(my\_list)

return my\_list

# Функция f2 должна фильтровать входной массив и возвращать только те элементы,

# которые начинаются со слова “программист”. Для фильтрации используйте функцию filter

@print\_result

def f2(arg):

return [name for name in arg if name.startswith("программист")]

# 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.

@print\_result

def f3(arg):

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

# 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.

@print\_result

def f4(arg):

rand\_list = gen\_random.gen\_random(len(arg), 100000, 200000)

zip\_obj = zip(arg, rand\_list)

for i in zip\_obj: print(i[0], ', зарплата ', i[1])

return -1

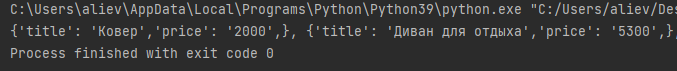
if \_\_name\_\_ == '\_\_main\_\_':

with cm\_timer.cm\_timer\_1():

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

Экранные формы с примерами выполнения программы:

1. Файл field.py



1. Файл gen\_random.py

Изображение выглядит как текст

Автоматически созданное описание

1. Файл unique.py

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

1. Файл sort.py

Изображение выглядит как текст

Автоматически созданное описание

1. Файл print\_result.py

Изображение выглядит как текст

Автоматически созданное описание

1. Файл cm\_timer.py

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

1. Файл main.py

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

Вывод

Изучение возможности функционального программирования в языке Python.