**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 лабораторной работе №3

«Функциональные возможности языка Python»

|  |  |  |
| --- | --- | --- |
| Выполнил: |  | Проверил: |
| студент группы ИУ5-32Б |  | преподаватель каф. ИУ5 |
| Турдиев Ихтияр |  | Гапанюк Юрий Евгеньевич |
|  |  |  |
|  |  |  |

Москва, 2024 г.

Постановка задачи

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.

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.

• 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. 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.

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.

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).

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

field.py

def field(items, \*args):  
 assert len(args) > 0  
 result = []  
 dict\_count = 0  
 for item in items:  
 result.append(dict())  
 for key in item.keys():  
 if key in args and item[key] is not None:  
 result[dict\_count].update({key: item[key]})  
 dict\_count += 1  
  
 if len(args) == 1:  
 is\_first = True  
 for dictionary in result:  
 if len(dictionary) == 0: continue  
 # print(f"'{dictionary[args[0]]}'", end='') if is\_first else print(f", '{dictionary[args[0]]}'", end='')  
 is\_first = False  
  
 elif len(args) > 1:  
 is\_first = True  
 for dictionary in result:  
 if len(dictionary) == 0: continue  
 # print(dictionary, end='') if is\_first else print(f", {dictionary}", end='')  
 is\_first = False  
 return [x[args[0]] for x in result]  
  
  
goods = [  
 {'title': 'Ковер', 'price': 2000, 'color': None},  
 {'title': 'Диван для отдыха', 'color': 'black'}  
]  
  
# field(goods, 'title', 'color')

gen\_random.py

from random import randint, seed  
from time import time  
  
  
def gen\_random(num\_count, begin, end):  
 seed(time())  
 result = []  
 for i in range(num\_count):  
 result.append(randint(begin, end))  
 return result  
  
  
# print(gen\_random(5, 1, 3))

unique.py

class Unique(object):  
 def \_\_init\_\_(self, items, \*\*kwargs):  
 if "ignore\_case" in kwargs.keys():  
 self.\_\_ignore\_case = kwargs.get("ignore\_case")  
 else: self.\_\_ignore\_case = False  
 self.\_\_data = items  
 self.\_\_elems\_set = set()  
 self.\_\_number = -1  
  
 def \_\_next\_\_(self):  
 self.\_\_number += 1  
 try:  
 if isinstance(self.\_\_data[self.\_\_number], str):  
 obj = self.\_\_data[self.\_\_number].lower() if self.\_\_ignore\_case else self.\_\_data[self.\_\_number]  
 if obj not in self.\_\_elems\_set:  
 self.\_\_elems\_set.add(obj)  
 return obj  
 else:  
 return self.\_\_next\_\_()  
 elif self.\_\_data[self.\_\_number] not in self.\_\_elems\_set:  
 self.\_\_elems\_set.add(self.\_\_data[self.\_\_number])  
 return self.\_\_data[self.\_\_number]  
 else:  
 return self.\_\_next\_\_()  
 except IndexError:  
 self.\_\_number -= 1  
 raise StopIteration  
  
 def \_\_iter\_\_(self):  
 return self  
  
  
data1 = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]  
data2 = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']  
# for i in Unique(data1):  
# print(i)  
  
# for i in Unique(data2, ignore\_case=True):  
# print(i)

sort.py

data = [4, -30, 100, -100, 123, 1, 0, -1, -4]  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 result = sorted(data, reverse=True)  
 print(result)  
  
 result\_with\_lambda = sorted(data, key=lambda n: n, reverse=True)  
 print(result\_with\_lambda)

print\_result.py

# def print\_result(func):  
# def wrapped():  
# func\_result = func()  
# print(func.\_\_name\_\_)  
# if isinstance(func\_result, list):  
# for i in func\_result:  
# print(i)  
# elif isinstance(func\_result, dict):  
# for key in func\_result:  
# print(f"{key} = {func\_result[key]}")  
# else:  
# print(func\_result)  
# return func\_result  
# return wrapped  
  
  
def print\_result(func):  
 def wrapped(arg):  
 func\_result = func(arg)  
 print(func.\_\_name\_\_)  
 print(func\_result)  
 return func\_result  
 return wrapped  
  
  
@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()

cm\_timer.py

from time import time, sleep  
from contextlib import contextmanager  
  
  
class cm\_timer\_1:  
 def \_\_enter\_\_(self):  
 self.\_\_start\_time = time()  
 return self  
  
 def \_\_exit\_\_(self, exc\_type, exc\_val, exc\_tb):  
 work\_time = time() - self.\_\_start\_time  
 print(f"time: {round(work\_time, 2)}")  
  
  
@contextmanager  
def cm\_timer\_2():  
 start\_time = time()  
 try:  
 yield start\_time  
 finally:  
 work\_time = time() - start\_time  
 print(f"time: {round(work\_time, 2)}")  
  
  
# with cm\_timer\_2():  
# sleep(5.5)

process\_data.py

import json  
from print\_result import print\_result  
from cm\_timer import cm\_timer\_1  
from unique import Unique  
from field import field  
from gen\_random import gen\_random  
  
path = "../data\_light.json"  
  
with open(path, encoding='utf8') as f:  
 data = json.load(f)  
  
  
@print\_result  
def f1(arg) -> list:  
 return sorted([str(x) for x in Unique(field(arg, "job-name"), ignore\_case=True)])  
  
  
@print\_result  
def f2(arg: list):  
 return list(filter(lambda x: x.startswith("программист"), arg))  
  
  
@print\_result  
def f3(arg):  
 return list(map(lambda x: x + " с опытом Python", arg))  
  
  
@print\_result  
def f4(arg):  
 salaries = gen\_random(len(arg), 100000, 200000)  
 return [f"{job\_name}, зарплата {salary} руб." for job\_name, salary in zip(arg, salaries)]  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 with cm\_timer\_1():  
 f4(f3(f2(f1(data))))

Обработка результатов    



 