**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 |
| Егоров Виктор |  | Гапанюк Юрий |
|  |  |  |
|  |  |  |

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  
 if len(args) == 1:  
 key = args[0]  
 for item in items:  
 value = item.get(key)  
 if value is not None:  
 yield value  
 else:  
 for item in items:  
 filtered = {key: item.get(key) for key in args if item.get(key) is not None}  
 if filtered:  
 yield filtered  
  
def main1():  
  
 goods = [  
 {'title': 'Ковер', 'price': 2000, 'color': 'green'},  
 {'title': 'Диван для отдыха', 'price': 5300, 'color': 'black'}  
 ]  
  
 print("Тест с одним аргументом 'title':")  
 for value in field(goods, 'title'):  
 print(value)  
  
 print("\nТест с несколькими аргументами 'title', 'price':")  
 for item in field(goods, 'title', 'price'):  
 print(item)  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main1()

**gen\_random.py**

import random  
  
def gen\_random(num\_count, begin, end):  
 *"""  
 Рандомные числа  
 :param num\_count: число случайных чисел  
 :param begin: с какого числа  
 :param end: по какое  
 :return: картеж чисел  
 """* return (random.randint(begin, end) for \_ in range(num\_count))  
  
  
def main2():  
 print("Пример вывода gen\_random(5, 1, 3):")  
 for number in gen\_random(5, 1, 3):  
 print(number, end=' ')  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main2()

**unique.py**

class Unique(object):  
  
 def \_\_init\_\_(self, items, \*\*kwargs):  
 self.items = iter(items)  
 self.seen = set()  
 self.ignore\_case = kwargs.get('ignore\_case', False)  
  
 def \_\_next\_\_(self):  
 while True:  
 item = next(self.items)  
 if isinstance(item, str) and self.ignore\_case:  
 processed\_item = item.lower()  
 else:  
 processed\_item = item  
  
 if processed\_item not in self.seen:  
 self.seen.add(processed\_item)  
 return item  
  
 def \_\_iter\_\_(self):  
 return self  
  
  
  
def main3():  
 data1 = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]  
 print("Пример 1: Список с числами")  
 for item in Unique(data1):  
 print(item, end=' ')  
  
 import random  
 def gen\_random(num\_count, begin, end):  
 return (random.randint(begin, end) for \_ in range(num\_count))  
  
 data2 = gen\_random(10, 1, 3)  
 print("\nПример 2: Генератор случайных чисел")  
 for item in Unique(data2):  
 print(item, end=' ')  
  
 data3 = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']  
 print("\nПример 3: Список со строками без ignore\_case")  
 for item in Unique(data3):  
 print(item, end=' ')  
  
 print("\nПример 4: Список со строками с ignore\_case")  
 for item in Unique(data3, ignore\_case=True):  
 print(item, end=' ')  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main3()

**sort.py**

def main4():  
 data = [4, -30, 30, 100, -100, 123, 1, 0, -1, -4]  
  
 result\_with\_lambda = sorted(data, key=lambda x: abs(x), reverse=True)  
 print(result\_with\_lambda)  
  
 result = sorted(data, key=abs, reverse=True)  
 print(result)  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main4()

**print\_result.py**

def print\_result(func):  
 def wrapper(\*args, \*\*kwargs):  
 print(func.\_\_name\_\_)  
  
 result = func(\*args, \*\*kwargs)  
  
 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]  
  
def main5():  
 print('!!!!!!!!')  
 test\_1()  
 test\_2()  
 test\_3()  
 test\_4()  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main5()

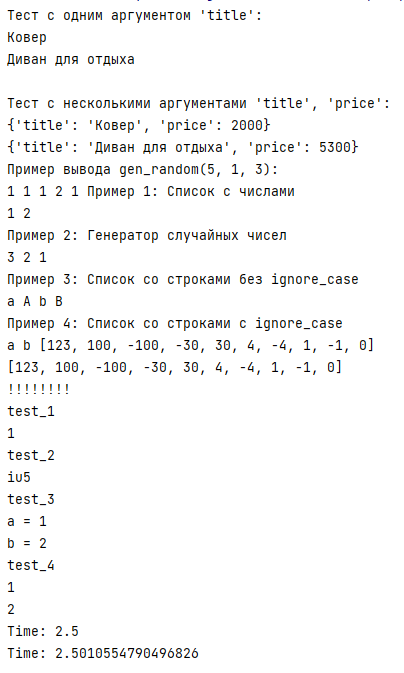
**cm\_timer.py**

from time import sleep, time  
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):  
 end\_time = time()  
 print(f"Time: {end\_time - self.start\_time:.1f}")  
  
@contextmanager  
def cm\_timer\_2():  
 st = time()  
 yield None  
 en = time()  
 print(f'Time: {en - st}')  
  
def main6():  
 with cm\_timer\_1():  
 sleep(2.5)  
  
 with cm\_timer\_2():  
 sleep(2.5)  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main6()

**process\_data.py**

from lab\_task.print\_result import print\_result  
from lab\_task.cm\_timer import cm\_timer\_1  
from lab\_task.gen\_random import gen\_random  
import json  
import sys  
  
try:  
 path = sys.argv[1]  
 print(path)  
except:  
 path = 'datalight.json'  
  
with open(path, encoding='utf-8') as f:  
 data = json.load(f)  
  
  
@print\_result  
def f1(arg) -> list:  
 *"""  
 Выводит отсортированный список профессий без повторений.  
 Сортировка должна игнорировать регистр.  
 :param arg: список  
 :return: отсортированный список  
 """* return sorted(list(set([el['job-name'] for el in arg])), key=lambda x: x.lower())  
  
  
@print\_result  
def f2(arg) -> list:  
 *"""  
 Фильтрует входной массив и возвращает только те элементы, которые начинаются со слова “программист”  
 :param arg:  
 :return:  
 """* return list(filter(lambda text: (text.split())[0].lower() == 'программист', arg))  
  
  
@print\_result  
def f3(arg) -> list:  
 *"""  
 Функция f3 должна модифицировать каждый элемент массива, добавив строку “с опытом Python”  
 :param arg:  
 :return:  
 """* return list(map(lambda lst: lst + ' с опытом Python', arg))  
  
  
@print\_result  
def f4(arg) -> list:  
 *"""  
 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.  
 :param arg:  
 :return:  
 """* return list(zip(arg, ['зарплата ' + str(el) + ' руб.' for el in gen\_random(len(arg), 100000, 200000)]))  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 with cm\_timer\_1():  
 f4(f3(f2(f1(data))))

Обработка результатов

****

