МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ  
им. Н.Э. Баумана

Факультет “Информатика и системы управления”  
Кафедра “Системы обработки информации и управления”

A logo with a gold star and blue ribbon

Description automatically generated

Дисциплина “Парадигмы и конструкции языков программирования”

Отчет по лабораторной работе №3  
“Объектно-ориентированные возможности языка Python”

**Выполнил:**Студент группы ИУ5-24Б  
Шерстеникин Н.С.  
**Преподаватель:**Нардид А.Н.

Москва 2025

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:

• 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

Шаблон для реализации генератора:

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:

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. Пример:

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.

КОД

Код(field.py)

from xml.etree.ElementTree import tostring  
  
goods = [{'title': 'Ковер', 'price': 2000, 'color': 'green'}, {'title': 'Диван для отдыха', 'price': 5300, 'color': 'black'}]  
def field(items, \*args):  
 # print(args[1])  
 answer = ""  
 if(len(args) == 1):  
 for i in range(0,len(goods)):  
 if(args[0] in goods[i]):  
 answer += goods[i][args[0]] + ","  
 print(answer[:len(answer)-1])  
 else:  
 answer = ""  
 for i in range(0, len(goods)):  
 for j in range(0,len(args)):  
 if(args[j] in goods[i]):  
 answer += ", " + "'" + args[j] + "'" + ":" + " '" + str(goods[i][args[j]]) + "'"  
 print("{" + answer[2:] + "}")  
 assert len(args) > 0  
  
field(goods, "title", "price")

Код(gen\_random.py)

from random import randint  
def Gen\_random(amount,min,max):  
 answer = []  
 for i in range(amount):  
 answer.append(randint(min,max))  
 return answer  
# print(Gen\_random(100,0,100))

Код(unique.py)

from gen\_random import Gen\_random  
  
class Unique():  
 def \_\_init\_\_(self,object,ignore\_case = False, \*\*kwargs):  
 self.ignore\_case = ignore\_case  
 self.data = object  
 self.answer = []  
 self.limit = 1  
 self.counter = 0  
  
 def \_\_next\_\_(self):  
 if(self.counter == self.limit):  
 raise StopIteration  
 if (self.ignore\_case):  
 for i in self.data:  
 if(isinstance(i,str)):  
 if(not(i.lower() in self.answer)):  
 self.answer.append(i.lower())  
 self.counter += 1  
 return self.answer  
 else:  
 for i in self.data:  
 if(not(i in self.answer)):  
 self.answer.append(i)  
 self.counter += 1  
 return self.answer  
  
 def \_\_iter\_\_(self):  
 return self

Код(sort.py)

def sort1(data):  
 sortedd = sorted(data, key=None, reverse=True)  
 return sortedd  
  
def sort2(data2):  
 answer = str(input("По какому параметру вы хотите сортировать?"))  
 sortedd = sorted(data2, key= lambda data2: data2[str(answer)], reverse=True)  
 return sortedd

Код(print\_result.py)

def print\_result(func):  
 def wrapper(\*args, \*\*kwargs):  
 result = func(\*args, \*\*kwargs)  
 print(f"Result: {result}")  
 return result  
 return wrapper

Код(cm\_timer.py)

from print\_result import print\_result  
import time  
from contextlib import contextmanager  
  
@contextmanager  
def cm\_timer\_1():  
 start = time.time()  
 yield  
 print(f"Execution time: {time.time() - start:.4f} seconds")

Код(process\_data.py)

import json  
from sys import exit  
  
from cm\_timer import cm\_timer\_1  
from gen\_random import Gen\_random  
from print\_result import print\_result  
from unique import Unique  
# Сделаем другие необходимые импорты  
  
@print\_result  
def f1(vacancies) -> list[str]:  
 Jobs = [vacancy["job-name"] for vacancy in vacancies]  
 return sorted(Unique(Jobs, case\_sensitive=False))  
  
  
@print\_result  
def f2(Jobs) -> list[str]:  
 return list(filter(lambda Job: Job.startswith("программист"), Jobs[0]))  
  
  
@print\_result  
def f3(Jobs) -> list[str]:  
 return list(map(lambda Job: Job + " с опытом Python", Jobs))  
  
  
@print\_result  
def f4(Jobs) -> list[str]:  
 Salaries = Gen\_random(len(Jobs), 100\_000, 200\_000)  
 return [f"{Job}, зарплата {Salarys} руб." for Job, Salarys in zip(Jobs, Salaries)]  
  
  
  
def main() -> int:  
 with open("data\_light.json", encoding="utf-8") as file:  
 data = json.load(file)  
 with cm\_timer\_1():  
 f4(f3(f2(f1(data))))  
  
 return 0  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 exit(main())

Работа приложения

