**Московский государственный технический**

**университет им. Н.Э. Баумана**

Факультет ИУ

Кафедра ИУ5

Курс «Основы информатики»

Отчет по лабораторной работе №\_3-4\_

«Функциональные возможности языка Python»

|  |  |  |
| --- | --- | --- |
| Выполнил: |  | Проверил: |
| студент группы ИУ5-23Б: |  | преподаватель каф. \_\_\_\_ |
| Селиханович Д.С. |  | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| Подпись и дата: |  | Подпись и дата: |

Москва, 2024 г.

**Постановка задачи**

**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.

**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.

**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.

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**

**\lab\_03-04\cm\_timer.py**

import time

from contextlib import contextmanager

class cm\_timer\_1:

    def \_\_enter\_\_(self):

        self.start\_time = time.time()

    def \_\_exit\_\_(self, exc\_type, exc\_val, exc\_tb):

        self.end\_time = time.time()

        print(f"time: {self.end\_time - self.start\_time}")

@contextmanager

def cm\_timer\_2():

    start\_time = time.time()

    try:

        yield

    finally:

        end\_time = time.time()

        print(f"time: {end\_time - start\_time}")

**\lab\_03-04\data\_light.json**

    {

        "mobile-url": "https://trudvsem.ru/vacancy/card/1027739174033/6bf457e6-51d8-11e6-853e-037acc02728d",

        "description": "<p>Умение общаться по телефону и лично, доброжелательность, ответственность, стрессоустойчивость.</p>",

        "update-date": "2016-10-02 01:33:38 MSK",

        "employment": "Частичная занятость",

        "job-name": "Администратор на телефоне",

        "company": {

            "email": "on.klinik@mail.ru",

            "contact-name": "Светлана",

            "hr-agency": true,

            "phone": "+7(495)6084488",

            "name": "ООО РОЯЛ КЛИНИК"

        },

        "term": "<p>Присутствуют по результатам работы</p>",

        "addresses": {

            "address": {

                "location": "г. Москва, Кузнецкий Мост улица, 1",

                "lat": 55.760808,

**…**

**\lab\_03-04\field.py**

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):

    try:

        assert len(args) > 0

    except:

        print("Нет подходящих аргументов.")

        return

    if len(args) == 1:

            for i in items:

                yield i[args[0]]

    else:

        for i in range(len(items)):

            answer = {}

            flag=0

            for j in range(len(args)):

                if(args[j] in items[i]):

                    answer[args[j]] = items[i][args[j]]

            yield answer

def main():

    for i in field(goods, 'title'):

        print(i)

    for i in field(goods, 'title', 'price'):

        print(i)

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

    main()

**\lab\_03-04\gen\_random.py**

from random import randint

def gen\_random(num, min, max):

    yield from (randint(min, max) for \_ in range(num))

def main():

    for i in gen\_random(5, 1, 3):

        print(i)

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

    main()

**\lab\_03-04\print\_result.py**

def print\_result(func):

    def wrapper(\*args, \*\*kwargs):

        print(func.\_\_name\_\_)

        result = func(\*args, \*\*kwargs)

        if isinstance(result, list):

            for i in result:

                print(i)

        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]

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

    print('!!!!!!!!')

    test\_1()

    test\_2()

    test\_3()

    test\_4()

def main():

    with cm\_timer\_1():

        time.sleep(5.5)

    with cm\_timer\_2():

        time.sleep(5.5)

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

    main()

**\lab\_03-04\process\_data.py**

import json

import sys

from print\_result import print\_result

from gen\_random import gen\_random

from cm\_timer import cm\_timer\_1

# Сделаем другие необходимые импорты

path = "c:\\Users\\user\\Desktop\\prog\\Python\\lab\_03-04\\data\_light.json"

# Необходимо в переменную path сохранить путь к файлу, который был передан при запуске сценария

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

    data = json.load(f)

# Далее необходимо реализовать все функции по заданию, заменив `raise NotImplemented`

# Предполагается, что функции f1, f2, f3 будут реализованы в одну строку

# В реализации функции f4 может быть до 3 строк

@print\_result

def f1(arg):

    return list(sorted(list(map(lambda x : x["job-name"], arg))))

@print\_result

def f2(arg):

    return list(filter(lambda x: x.lower().startswith("программист"), arg))

@print\_result

def f3(arg):

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

@print\_result

def f4(arg):

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

    return [f"{spec}, зарплата {sal} руб." for spec, sal in salary]

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

    with cm\_timer\_1():

        f4(f3(f2(f1(data))))

**\lab\_03-04\sort.py**

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)

**\lab\_03-04\unique.py**

# Итератор для удаления дубликатов

from gen\_random import gen\_random

class Unique(object):

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

        self.answer=[]

        self.index=0

        try:

            if kwargs["ignore\_case"] == True:

                try:

                    for i in range(len(items)):

                        items[i]=items[i].lower()

                except:

                    pass

                for i in items:

                    if not i in self.answer:

                        self.answer.append(i)

            else:

                for i in items:

                    if not i in self.answer:

                        self.answer.append(i)

        except:

            for i in items:

                if not i in self.answer:

                    self.answer.append(i)

    def \_\_next\_\_(self):

        if self.index < len(self.answer):

            result = self.answer[self.index]

            self.index += 1

            return result

        else:

            raise StopIteration

    def \_\_iter\_\_(self):

        return self

def main():

    data = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]

    for i in Unique(data):

        print(i,end = ' ')

    print()

    data = gen\_random(10, 1, 3)

    for i in Unique(data):

        print(i,end = ' ')

    print()

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

    for i in Unique(data):

        print(i,end = ' ')

    print()

    for i in Unique(data, ignore\_case=True):

        print(i,end = ' ')

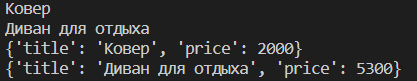
    print()

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

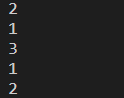
    main()

**Анализ результатов**

**Задача 1**

****

**Задача 2**

****

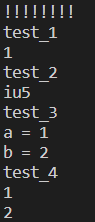
**Задача 3**

****

**Задача 4**

****

**Задача 5**

****

**Задача 6**

****

**Задача 7**

…

