**Московский государственный технический**

**университет им. Н.Э. Баумана**

**Факультет ИУ**

**Кафедра ИУ5**

**Курс «Основы информатики»**

**Отчет лабораторной работе №3-4**

Выполнил студент группы ИУ5-33Б:

Козлов А. А.

Подпись и дата:

Проверил преподаватель каф.:

Гапанюк Ю. Е.

Подпись и дата:

Москва, 2024 г

**Описание задания**

Задание лабораторной работы состоит из решения нескольких задач.

Файлы, содержащие решения отдельных задач, должны располагаться в пакете lab\_python\_fp. Решение каждой задачи должно располагаться 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.

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.

**Текст программы**

def field (items, \*args):

    assert len(args)>0, "Список пуст"

    for item in items:

        for arg in args:

            if arg not in item:

                pass

            else:

                print(f"{arg.title()}:{item[arg]}")

                print("-"\*20)

def main():

    goods = [

        {"name": "sofa", "color": "white"},

        {"name": "chair", "price": 1000, "color": "black"},

        {"name": "desk", "price": 2990, "color": "brown"},

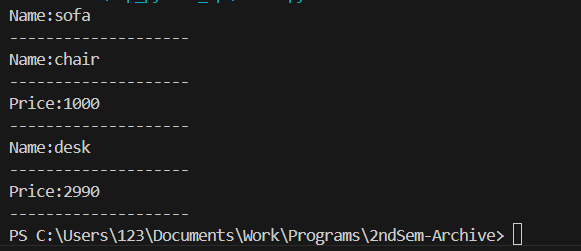
        {"color": "blue"},

    ]

    field(goods, "name", "price")

main()

**Выполнение программы**



**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.

**Текст программы**

def gen\_random(numCount, begin, end):

    import random

    result = []

    for \_ in range(numCount):

        result.append(random.randint(begin, end))

    return(result)

def main():

    result = gen\_random(10, 1, 3)

    for i in result:

        print(i, end=' ')

main()

**Выполнение программы**



**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.

**Текст программы**

def gen\_random(numCount, begin, end):

    import random

    result = []

    for \_ in range(numCount):

        result.append(random.randint(begin, end))

    return(result)

class Unique(object):

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

        self.ignore\_case = kwargs.get('ignore\_case', False)

        self.data = data

        self.index = 0

        self.seen = set()

    def \_\_next\_\_(self):

        while self.index < len(self.data):

            item = self.data[self.index]

            if isinstance(item, str):

                if self.ignore\_case == False:

                    key = item

                else:

                    key = item.lower()

            else:

                key = item

            if key not in self.seen:

                self.seen.add(key)

                return item

            self.index += 1

        raise StopIteration

    def \_\_iter\_\_(self):

        return self

"""

def main():

        data = ['a', 'b', 'c', 'a', 'b', 'd', 'A', 'B', 'C', 'D']

        dataRandom = gen\_random(10, 1,10)

        unique1 = Unique(data, ignore\_case=True)

        unique2 = Unique(dataRandom)

        for item in unique1:

            print(item, end=' ')

        print('\n')

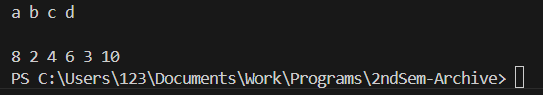
        for item in unique2:

            print(item, end=' ')

main()

"""

**Выполнение программы**



**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.

С использованием lambda-функции.

Без использования lambda-функции.

**Текст программы**

def sortArray(array):

    for i in range( len(array)-1):

        for j in range( len(array)-i-1):

            if abs(array[j]) < abs(array[j+1]):

                array[j], array[j+1] = array[j+1], array[j]

    return array

sort = lambda array: sorted(array, key=abs, reverse=True)

def main():

    from random import randint

    array = [randint(-100, 100) for \_ in range(10)]

    result = sortArray(array)

    resultWithLambda = sort(array)

    print("Without Lambda: ")

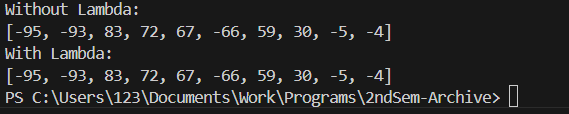
    print(result)

    print("With Lambda: ")

    print(resultWithLambda)

main()

**Выполнение программы**

****

**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.

**Текст программы**

def print\_result(func):

    def wrapper(\*args, \*\*kwargs):

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

        if isinstance(result, list):

            for item in result:

                print(item, end=' | ')

        elif isinstance(result, dict):

            for key, value in result.items():

                print(f"{key}: {value}")

        else:

            print(result)

        print('\n' + '-'\*100)

        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('!!!!!!!!')

    print('test\_1')

    test\_1()

    print('test\_2')

    test\_2()

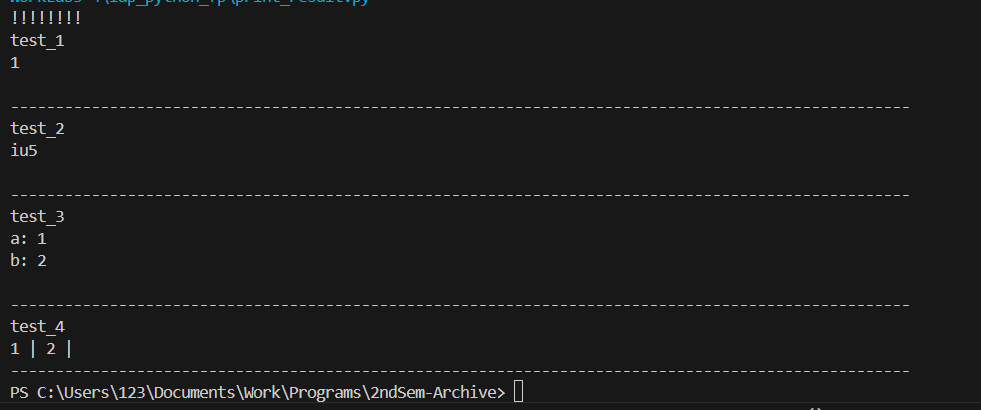
    print('test\_3')

    test\_3()

    print('test\_4')

    test\_4()

**Выполнение программы**

****

**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).

**Текст программы**

import time

from contextlib import contextmanager

class cm\_timer\_1:

    def \_\_enter\_\_(self):

        self.startTime = time.time()

        return self

    def \_\_exit\_\_(self, type, value, trecaback):

        self.endTime = time.time()

        self.elapsedTime = self.endTime - self.startTime

        print(f'Elapsed time: {self.elapsedTime} секунд')

@contextmanager

def cm\_timer\_2():

    startTime = time.time()

    try:

        yield

    finally:

        endTime = time.time()

        elapsedTime = endTime - startTime

        print(f'Elapsed time: {elapsedTime} секунд')

"""

def main():

    with cm\_timer\_1():

        print('Hello, world!')

        print('Блок кода выполнен')

    with cm\_timer\_2():

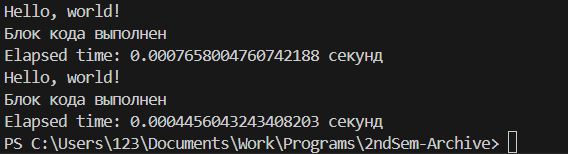
        print('Hello, world!')

        print('Блок кода выполнен')

main()

"""

**Выполнение программы**



**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**

import json

import random

from print\_result import print\_result

from cm\_timer import cm\_timer\_1

from unique import Unique

path = "C:/Users/123/Documents/Work/Programs/2ndSem-WorkLab3-4/lap\_python\_fp/data\_light.json"

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

    data = json.load(f)

@print\_result

def f1(data):

    jobNames = [items['job-name'] for items in data]

    unique = Unique(jobNames)

    return sorted(unique)

@print\_result

def f2(data):

    return list(filter(lambda items: items.startswith('программист'), data))

@print\_result

def f3(data):

    return list(map(lambda items: items + ' с опытом Python', data))

@print\_result

def f4(data):

    salaries = [random.randint(100000, 200000) for \_ in data]

    result = [f"{spec}, зарплата {salary} руб." for spec, salary in zip(data, salaries)]

    return result

def main():

    with cm\_timer\_1():

        f4(f3(f2(f1(data))))

main()

**Выполнение программы**

