**Московский государственный технический**

**университет им. Н.Э. Баумана**

**Факультет ИУ**

**Кафедра ИУ5**

**Курс «Основы информатики»**

**Отчет лабораторной работе №3-4**

**lab\_python\_fp**

Выполнил студент группы ИУ5-33Б:

Османов З. Ш.

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 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

**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.

**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:

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](https://github.com/ugapanyuk/BKIT_2021/blob/main/notebooks/fp/files/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 программы:**

**field.py**

goods = [

    {'title': 'Ковер', 'price': 2000, 'color': 'green'},

    {'title': 'Диван для отдыха', 'color': 'black'},

    {'title': 'Пылесос', 'color': 'yellow'},

    {'title': 'Телевизор', 'price': 34000}

]

args = ['title','color']

def field(items, \*args):

    assert len(args) > 0

    #print(type(\*args))

    if len(args) == 1:

        return [ str(element.get(args[0])) for element in items if (element.get(args[0]) != None) ]

    else:

        #all\_keys = ["title", "price", "color"]

        args = [key for key in ["title", "price", "color"] if key in args]

        res = []

        for element in items:

            arr = {}

            for key in args:

                if element.get(key) != None: arr[key] = element[key]

            res.append(arr)

        return res

print(field(goods, "color", "title"))

**get\_random.py**

from random import \*

def get\_random(amount, start, end):

    for i in range(amount):

        yield randint(start, end)

for item in get\_random(5, 10, 100):

    print(item)

**unique.py**

class Unique:

    def \_\_init\_\_(self, items, ignore\_case = False):

        self.items = []

        if ignore\_case:                 #items = [item.lower() if type(item) == str else item for item in items]

            arr = []

            for item in items:

                if (type(item) == str and item.lower() not in arr):

                    self.items.append(item)

                    arr.append(item.lower())

                elif (type(item) != str and item not in self.items):

                    self.items.append(item)

        else:

            for item in items:

                if (item not in self.items): self.items.append(item)

    def \_\_iter\_\_(self):

        return self

    def \_\_next\_\_(self):

        for item in self.items:

            self.items.remove(item)

            return item

        raise StopIteration

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

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

    data = ["A", "A", 2, 455,"a", "a"]

    for item in Unique(data):

        print(item)

    data = [1, 1, 2, 2, 3, 3]

    for item in Unique(data):

        print(item)

**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 = lambda data: sorted(data, key = abs, reverse = True)

    print(result\_with\_lambda(data))

**print\_result.py**

def print\_result(input\_func):

    def output\_func(\*arg):

        print(input\_func.\_\_name\_\_)                                # print(str(input\_func)[10:str(input\_func).index("a")-1])

        result = input\_func(\*arg)

        if type(result) == list: print(\*result, sep = "\n")       # [ print(i) for i in result ]

        elif type(result) == dict: [print(f"{i} = {result[i]}") for i in result ]

        else: print(result)

        return result

    return output\_func

@print\_result

def test\_1():

    return 1

@print\_result

def test\_2():

    return 'iu5'

@print\_result

def test\_3():

    return {'a': 1, 'b': 13}

@print\_result

def test\_4():

    return [1, 2, 3, 4, 5]

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

    print('!!!!!!!!')

    test\_1()

    test\_2()

    test\_3()

    test\_4()

**cm\_timer.py**

import time

from contextlib import contextmanager

@contextmanager

def cm\_timer\_2():

    start\_time = time.time()

    try:

        yield

    finally:

        end\_time = time.time()

        print(f"time: {(end\_time - start\_time)//0.001/1000}")

class cm\_timer\_1:

    def \_\_enter\_\_(self):

        self.start\_time = time.time()

        return self

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

        end\_time = time.time()

        print(f"time: {(end\_time - self.start\_time)//0.001/1000}")

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

    with cm\_timer\_1():

        time.sleep(5.5)

    with cm\_timer\_2():

        time.sleep(5.5)

**process\_data.py**

import json

import sys

from unique import Unique

from cm\_timer import cm\_timer\_1

from print\_result import print\_result

from get\_random\_ import get\_random

from field import field

path = sys.argv[1] if len(sys.argv) > 1 else "data\_light.json"

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

    data = json.load(f)

@print\_result

def f1(arg):

    return sorted([item for item in Unique(field(arg, "job-name"))])

@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: f"{x} с опытом Python", arg))

@print\_result

def f4(arg):

    #return [f"{job}, зарплата {random.randint(100000, 200000)} руб." for job in arg]

    return [f"{job}, зарплата {next(get\_random(len(arg), 100000, 200000))} руб." for job in arg]

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

    with cm\_timer\_1():

        f4(f3(f2(f1(data))))

**Выполнение программы**

