|  |  |  |
| --- | --- | --- |
| Защищено:  Нардид А.Н.    "\_\_"\_\_\_\_\_\_\_\_\_\_\_\_\_2024 г. |  | Демонстрация:  Нардид А.Н.  "\_\_"\_\_\_\_\_\_\_\_\_\_\_\_\_2024 г. |

**Отчет по лабораторной работе № 3-4 по курсу**

**Парадигмы и конструкции языков программирования**

**ГУИМЦ**

#### Тема работы: " Функциональные возможности языка Python. "

6

(количество листов)

|  |  |
| --- | --- |
| ИСПОЛНИТЕЛЬ: |  |
| студент группы ИУ5Ц-51Б | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
|  | (подпись) |
| Омарова З.Р. | "\_\_"\_\_\_\_\_\_\_\_\_\_\_\_\_2024 г. |

Москва, МГТУ - 2024 г.

# Тема и задание для выполнения лабораторной работы.

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)**

1. Необходимо реализовать генератор field. Генератор field последовательно выдает значения ключей словаря

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.

**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.

**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.

# Листинг программы

**field.py**

def field(items, \*args):

    assert len(args) > 0, "Нужно указать хотя бы один ключ"

    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:

            result = {key: item.get(key) for key in args if item.get(key) is not None}

            if result:

                yield result

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

    goods = [

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

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

    ]

    print(list(field(goods, 'title')))

    print(list(field(goods, 'title', 'price')))

**gen\_rundom.py**

import random

def gen\_random(num\_count, begin, end):

    for \_ in range(num\_count):

        yield random.randint(begin, end)

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

    print(list(gen\_random(5, 1, 3)))

**unique.py**

class Unique:

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

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

        self.seen = set()

        self.items = iter(items)

    def \_\_iter\_\_(self):

        return self

    def \_\_next\_\_(self):

        while True:

            item = next(self.items)

            comp\_item = item.lower() if self.ignore\_case and isinstance(item, str) else item

            if comp\_item not in self.seen:

                self.seen.add(comp\_item)

                return item

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

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

    print(list(Unique(data, ignore\_case=True)))

**sort.py**

data = [4, -30, 100, -100, 123, 1, 0, -1, -4]

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

    result\_with\_lambda = sorted(data, key=lambda x: abs(x), reverse=True)

    print(result\_with\_lambda)

    def abs\_sort(val):

        return abs(val)

    result = sorted(data, key=abs\_sort, reverse=True)

    print(result)

**print\_result.py**

def print\_result(func):

    def printer(\*args, \*\*kwargs):

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

        print(func.\_\_name\_\_)

        if isinstance(result, list):

            print("\n".join(map(str, result)))

        elif isinstance(result, dict):

            print("\n".join(f"{k} = {v}" for k, v in result.items()))

        else:

            print(result)

        return result

    return printer

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

    @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]

    test\_1()

    test\_2()

    test\_3()

    test\_4()

**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):

        print(f"time: {time.time() - self.start\_time:.2f}")

@contextmanager

def cm\_timer\_2():

    start\_time = time.time()

    yield

    print(f"time: {time.time() - start\_time:.2f}")

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

    with cm\_timer\_1():

        time.sleep(1.5)

    with cm\_timer\_2():

        time.sleep(1.5)

**processer\_data.py**

import json

import os

from .field import field

from .gen\_random import gen\_random

from .unique import Unique

from .print\_result import print\_result

from .cm\_timer import cm\_timer\_1

data\_file = 'data\_light.json'

@print\_result

def f1(data):

    return sorted(set(Unique(field(data, 'job-name'), ignore\_case=True)), key=str.lower)

@print\_result

def f2(data):

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

@print\_result

def f3(data):

    return list(map(lambda x: f"{x} с опытом Python", data))

@print\_result

def f4(data):

    salaries = list(gen\_random(len(data), 100\_000, 200\_000))

    return [f"{job}, зарплата {salary} руб." for job, salary in zip(data, salaries)]

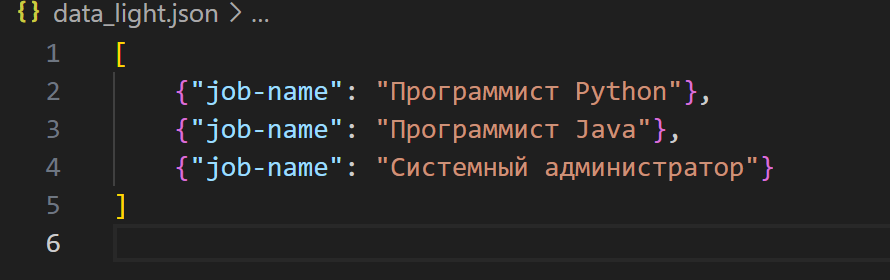
if \_\_name\_\_ == '\_\_main\_\_':

    with open(data\_file, encoding='utf-8') as f:

        data = json.load(f)

    with cm\_timer\_1():

        f4(f3(f2(f1(data))))



# 

# Результаты работы программы

**field.py**

['Ковер', 'Диван для отдыха']

[{'title': 'Ковер', 'price': 2000}, {'title': 'Диван для отдыха'}]

**gen\_random.py**

[2, 2, 1, 1, 1]

**unique.py**

['a', 'b']

**sort.py**

[123, 100, -100, -30, 4, -4, 1, -1, 0]

[123, 100, -100, -30, 4, -4, 1, -1, 0]

**print\_result.py**

test\_1

1

test\_2

iu5

test\_3

a = 1

b = 2

test\_4

1

2

**cm\_timer.py**

time: 1.50

time: 1.50

**process\_data.py**

f1

Программист Java

Программист Python

Системный администратор

f2

Программист Java

Программист Python

f3

Программист Java с опытом Python

Программист Python с опытом Python

f4

Программист Java с опытом Python, зарплата 101663 руб.

Программист Python с опытом Python, зарплата 121203 руб.

time: 0.00