**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 ИУ5-33Б: |  | преподаватель каф. \_\_\_\_ |
| Номоконов В.А |  | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| 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**

**\lab3-4\cm\_timer.py**

import time

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()

        elapsed\_time = end\_time - self.start\_time

        print(f"time: {elapsed\_time:.1f}")

from contextlib import contextmanager

@contextmanager

def cm\_timer\_2():

    start\_time = time.time()

    try:

        yield

    finally:

        end\_time = time.time()

        elapsed\_time = end\_time - start\_time

        print(f"time: {elapsed\_time:.1f}")

with cm\_timer\_1():

    time.sleep(5.5)

with cm\_timer\_2():

    time.sleep(5.5)

**\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,

**…**

**\lab3-4\field.py**

def field(items, \*args):

    assert len(args) > 0

    for item in items:

        if len(args) == 1:

            value = item.get(args[0])

            if value is not None:

                yield value

        else:

            result = {}

            for arg in args:

                value = item.get(arg)

                if value is not None:

                    result[arg] = value

            if result:

                yield result

goods = [

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

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

]

for value in field(goods, 'title'):

    print(value)  # Output: 'Ковер', 'Диван для отдыха'

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

    print(result)  # Output: {'title': 'Ковер', 'price': 2000}, {'title': 'Диван для отдыха'}

**\lab3-4\gen\_random.py**

import random

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

    for \_ in range(num\_count):

        yield random.randint(begin, end)

for num in gen\_random(5, 1, 3):

    print(num)  # Вывод: 5 случайных чисел от 1 до 3, например 2, 2, 3, 2, 1

**\lab3-4\print\_result.py**

def print\_result(func):

    def wrapper(\*args, \*\*kwargs):

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

        print(f"Function '{func.\_\_name\_\_}' returned:")

        if isinstance(result, list):

            for item in result:

                print(item)

        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()

**\lab3-4\process\_data.py**

import json

import sys

import random

import os

import time

def print\_result(func):

    def wrapper(\*args, \*\*kwargs):

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

        print(f"Function '{func.\_\_name\_\_}' returned:")

        if isinstance(result, list):

            for item in result:

                print(item)

        elif isinstance(result, dict):

            for key, value in result.items():

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

        else:

            print(result)

        return result

    return wrapper

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()

        elapsed\_time = end\_time - self.start\_time

        print(f"time: {elapsed\_time:.1f}")

from contextlib import contextmanager

path = os.path.join(os.path.dirname(\_\_file\_\_), "data\_light.json")

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

    data = json.load(f)

@print\_result

def f1(arg):

    return sorted(set(item['job-name'].lower() for item in arg))

@print\_result

def f2(arg):

    return list(filter(lambda item: item.startswith('программист'), arg))

@print\_result

def f3(arg):

    return list(map(lambda item: item + ' с опытом Python', arg))

@print\_result

def f4(arg):

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

    return list(zip(arg, salaries))

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

    with cm\_timer\_1():

        f4(f3(f2(f1(data))))

**\lab3-4\sort.py**

data = [4, -30, 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)

**\lab3-4\unique.py**

class Unique(object):

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

        self.items = items

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

        self.seen = set()

    def \_\_next\_\_(self):

        for item in self.items:

            if self.ignore\_case and isinstance(item, str):

                item\_key = item.lower()

            else:

                item\_key = item

            if item\_key not in self.seen:

                self.seen.add(item\_key)

                return item

        raise StopIteration

    def \_\_iter\_\_(self):

        return self

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

unique\_data = Unique(data)

for item in unique\_data:

    print(item)  # prints 1, 2

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

unique\_data = Unique(data)

for item in unique\_data:

    print(item)  # prints 'a', 'A', 'b', 'B'

unique\_data = Unique(data, ignore\_case=True)

for item in unique\_data:

    print(item)  # prints 'a', 'b'

**Анализ результатов**













