Московский государственный технический университет им. Н.Э. Баумана

Факультет «Информатика и системы управления»
Кафедра ИУ5 «Системы обработки информации и управления»

Курс «Базовые компоненты интернет-технологий»

Отчет по лабораторной работе №3-4
«Функциональные возможности языка Python»

Выполнил:
студент группы ИУ5-32Б:
Бекетов Роман Александрович
Подпись и дата:

Проверил:
преподаватель каф. ИУ5
Гапанюк Ю.Е.
Подпись и дата:

Москва, 2022 г.

Задание

Задача 1 (файл field.py)

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 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

cm_timer.py

```
from contextlib import contextmanager
import time

from unicodedata import name

@contextmanager
def cm_timer_1():
    start_time = time.time()
    yield
    print("time: ", time.time() - start_time)

class cm_timer_2:
    def __init__(self):
        self.start_time = 0
    def __enter__(self):
        self.start_time = time.time()
    def __exit__(self, type, value, traceback):
        print("time: ", time.time() - self.start_time)

if __name__ == "__main__":

    with cm_timer_1():
        time.sleep(1.5)

    with cm_timer_2():
        time.sleep(1.5)
```

field.py

```
def field(items, *args):
    assert len(args) > 0
    assert type(items) == list

    if len(args) == 1:
        for it in items:
            yield it[args[0]]
    else:
        for _dict in items:
            yield {_key:_dict[_key] for _key in args}
```

gen_random.py

```python
from random import randint

def gen_random(num_count, begin, end):
    for i in range(num_count):
        yield randint(begin, end)
```

print_result.py

```python
def print_result(func_to_decorate):

    def decoreate_func(*args, **kwargs):
        print(func_to_decorate.__name__)
        data = func_to_decorate(*args, **kwargs)
        if type(data) == list:
            for _ in data:
                print(_)
        elif type(data) == dict:
            for _ in data:
                key = data[_]
                print(f"{_} = {key}")
        else:
            print(data)
        return data
    return decoreate_func

@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()
```

sort.py

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

if __name__ == '__main__':
    sort_lambda = lambda x : sorted(x, reverse=True)
    result = sorted(data, reverse=True)
    print(result)
    print(sort_lambda(data))
```

unique.py

```
from gen_random import gen_random

class Unique:
    def __init__(self, data, **kwargs):
        self.used_elements = set()
        self.data = data
        self.index = 0
        if len(kwargs) == 0:
            self.bool_ignore_case = False
        else:
            self.bool_ignore_case = kwargs["bool_ignore_case"]
        self.tmp_list = [i for i in self.data]

    def __iter__(self):
        return self

    def __next__(self):
        while True:
            if self.index >= len(self.tmp_list):
                raise StopIteration
            else:
                current = self.tmp_list[self.index]
                if (type(current) == str and self.bool_ignore_case):
                    current = current.lower()
                self.index = self.index + 1
                if current not in self.used_elements:
                    self.used_elements.add(current)
                    return current
```

process_data.py

```
import json
import sys

from cm_timer import cm_timer_1
from gen_random import gen_random
from unique import Unique
from print_result import print_result
# Сделаем другие необходимые импорты

path = "data_light.json"

with open(path) 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 sort_unique(data_arg):
    return sorted([it for it in Unique([i["job-name"] for i in data_arg], bool_ig-
nore_case = True)])

@print_result
def check_start(data_arg):
    return [it.title() for it in list(filter(lambda x: x.startswith("программист"),
data_arg))]


@print_result
def add_lit(data_arg):
    return list(map(lambda x: x + " с опытом Python", data_arg))


@print_result
def add_salary(data_arg):
    return list(zip(data_arg, gen_random(len(data_arg), 100000, 200000)))

if __name__ == '__main__':
    with cm_timer_1():
        add_salary(add_lit(check_start(sort_unique(data))))
```

Тесты

Корректная работа модуля process_data.py подтверждает корректность работы других модулей.