Московский государственный технический университет им. Н.Э. Баумана

Факультет Радиотехнический
Кафедра РТ5

Курс «Парадигмы и конструкции языков программирования»

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

Выполнил:

студент группы РТ5-31Б:
Пичурин В. Е.

Подпись и дата:

Проверил:

преподаватель каф. ИУ5
Гапанюк Ю. Е.

Подпись и дата:

Москва, 2023

Условие 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 раполагаться в отдельном файле.

При запуске каждого файла выдаются тестовые результаты выполнения соответствующего задания.

Задание 1

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.

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

    if len(args) == 1:
        for item in items:
            yield item.get(args[0], None)
    else:
        for item in items:
            result = {key: item.get(key, None) for key in args}
            if any(result.values()):
                yield result
```

Задание 2

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

```
import random


def gen_random(num_count, begin, end):
    for _ in range(num_count):
        yield random.randint(begin, end)
```

Задание 3

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.

```
class Unique(object):
    def __init__(self, items, **kwargs):
        self.items = iter(items)
        self.ignore_case = kwargs.get('ignore_case', False)
        self.unique_set = set()

    def __next__(self):
        while True:
            item = next(self.items)
            key = item.lower() if self.ignore_case and
isinstance(item, str) else item
            if key not in self.unique_set:
                self.unique_set.add(key)
                return item

    def __iter__(self):
        return self
```

Задание 4

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:

С использованием lambda-функции.

Без использования lambda-функции.

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

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

    result_without_lambda = sorted(data, key=abs, reverse=True)
    print(result_without_lambda)
```

Задание 5

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.

```
def print_result(func):
    def wrapper(*args, **kwargs):
        result = func(*args, **kwargs)
        print(func.__name__)
        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
```

Задача 6

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).

```
import time
from contextlib import contextmanager


class cm_timer_1:
    def __enter__(self):
        self.start_time = time.time()
        return self

    def __exit__(self, exc_type, exc_val, exc_tb):
        elapsed_time = time.time() - self.start_time
```

```
        print(f"time: {elapsed_time}")


@contextmanager
def cm_timer_2():
    start_time = time.time()
    yield
    elapsed_time = time.time() - start_time
    print(f"time: {elapsed_time}")
```

Задача 7

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.

```
import json

from lab_python_fp.cm_timer import cm_timer_1
from lab_python_fp.print_result import print_result

path = 'data_light.json'

with open(path, encoding='UTF-8') as f:
    data = json.load(f)
```

```python
@print_result
def f1(arg):
    return sorted(list(set(str(job).lower() for job in arg)),
                  key=lambda x: x.lower())


@print_result
def f2(arg):
    return list(filter(lambda x:
str(x).startswith('программист'), arg))


@print_result
def f3(arg):
    return list(map(lambda x: str(x) + ', с опытом Python',
arg))


@print_result
def f4(arg):
    salaries = [100000 + i for i in range(len(arg))]
    return list(
        map(lambda x, y: f'{str(x)}, зарплата {y} руб.', arg,
salaries))


if __name__ == '__main__':
    with cm_timer_1():
        f4(f3(f2(f1(data))))
```