Отчёт

По лр-3

Дисциплина «Парадигмы и конструкции языков программирования»

Студент: Якубович Анна

Группа: ИБМЗ-23Б

Преподаватель: Гапанюк Ю.Е.

Задание № 1:

Задание:

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

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.

Код программы:

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

    if len(args) == 1:
        field_name = args[0]
        for item in items:
            if field_name in item and item[field_name] is not None:
                yield item[field_name]
    else:
        for item in items:
            result = {}
            has_valid_fields = False
            for field_name in args:
```

```
                if field_name in item and item[field_name] is not None:
                    result[field_name] = item[field_name]
                    has_valid_fields = True

            if has_valid_fields:
                yield result

goods = [
    {'title': 'Ковер', 'price': 2000, 'color': 'green'},
    {'title': 'Диван для отдыха', 'color': 'black'},
    {'title': None, 'price': 1500},
    {'price': None, 'color': 'red'}
]
if __name__ == "__main__":
    print("Тест 1 - один аргумент:")
    for item in field(goods, 'title'):
        print(item)

    print("Тест 2 - несколько аргументов:")
    for item in field(goods, 'title', 'price'):
        print(item)
```

Пример выполнения программы:

```
Тест 1 - один аргумент:
Ковер
Диван для отдыха
Тест 2 - несколько аргументов:
{'title': 'Ковер', 'price': 2000}
{'title': 'Диван для отдыха'}
{'price': 1500}
```

Задача №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.

Код программы:

```
import random



def gen_random(num_count, begin, end):

    for _ in range(num_count):

        yield random.randint(begin, end)

if __name__ == "__main__":

    print("Тест gen_random:")

    for num in gen_random(5, 1, 3):

        print(num, end=" ")

    print()
```

Пример работы программы:

```
Тест gen_random:
1 2 3 2 2
```

Задача №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.

Код программы:

```
        class Unique(object):

    def __init__(self, items, **kwargs):

        self.ignore_case = kwargs.get('ignore_case', False)
```

```
        self.items = iter(items)
        self.seen = set()

    def __next__(self):
        while True:
            item = next(self.items)

            if self.ignore_case and isinstance(item, str):
                check_key = item.lower()
            else:
                check_key = item

            if check_key not in self.seen:
                self.seen.add(check_key)
                return item

    def __iter__(self):
        return self

if __name__ == "__main__":
    print("Тест 1 - числа:")
    data1 = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]
    for item in Unique(data1):
        print(item, end=" ")
    print()

    print("Тест 2 - строки без ignore_case:")
    data2 = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']
    for item in Unique(data2):
        print(item, end=" ")
    print()

    print("Тест 3 - строки с ignore_case=True:")
    for item in Unique(data2, ignore_case=True):
        print(item, end=" ")
```

```
    print()
```

Пример работы программы:

```
Тест 1 - числа:
1 2
Тест 2 - строки без ignore_case:
a A b B
Тест 3 - строки с ignore_case=True:
a b
```

Задача №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.

Необходимо решить задачу двумя способами:

1. С использованием lambda-функции.
2. Без использования lambda-функции.

Код программы:

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

    print(result_with_lambda)
```

Пример работы программы:

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

Задача №5

Необходимо реализовать декоратор print_result, который выводит на экран результат выполнения функции.

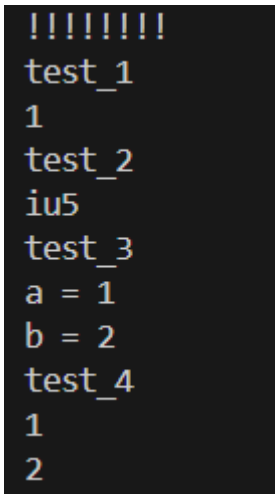
- Декоратор должен принимать на вход функцию, вызывать её, печатать в консоль имя функции и результат выполнения, после чего возвращать результат выполнения.
- Если функция вернула список (list), то значения элементов списка должны выводиться в столбик.
- Если функция вернула словарь (dict), то ключи и значения должны выводить в столбик через знак равенства.

Код программы:

```
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


@print_result
def test_1():
    return 1
@print_result
def test_2():
    return 'iu5'
@print_result
```

```python
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()
```

Пример работы программы:



Задача №6

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).

Код программы:

```python
import time
```

```python
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:.1f}")

@contextmanager

def cm_timer_2():

    start_time = time.time()

    try:

        yield

    finally:

        elapsed_time = time.time() - start_time

        print(f"time: {elapsed_time:.1f}")

if __name__ == "__main__":

    print("Тест cm_timer_1:")

    with cm_timer_1():

        time.sleep(1.5)


    print("Тест cm_timer_2:")

    with cm_timer_2():

        time.sleep(1.5)
```

Пример работы программы:

```
Тест cm_timer_1:
time: 1.5
Тест cm_timer_2:
time: 1.5
```

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.

Код программы:

```
import sys

import os

import json



sys.path.append(os.path.dirname(os.path.abspath(__file__)))

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

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)
```

```python
@print_result

def f1(arg):

    return sorted(Unique(field(arg, 'job-name'), ignore_case=True), key=str.lower)

@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):

    salaries = list(gen_random(len(arg), 100000, 200000))

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

if __name__ == '__main__':

    with cm_timer_1():

        f4(f3(f2(f1(data))))
```

Пример работы программы:

Монтажник санитарно-технических систем и оборудования
монтажник связи
Монтажник систем кондиционирования
Монтажник технологических трубопроводов
Монтажник технологического оборудования и связанных с ним конструкций
Монтажник технологического трубопровода
Монтажник технологичиского трубопровода
Монтажник трубопровода
Монтажник-отделочник
Монтажник-сантехник (ОВ, ВК)
Монтажник-сборщик рекламных конструкций
Монтажники ЖБК (Керченский мост)
Монтажники металлоконструкций
Монтажники технологического оборудования
монтажники труб пнд
Монтер пути
монтеры пути и бригадиры пути
Монтёр пути
мотальщица
Моторист
Моторист (машинист)
Моторист, дизелист
Музыкальный руководитель
НАБОРЩИК