Московский государственный технический

университет им. Н.Э. Баумана

Отчёт по лабораторной работе №4
по курсу «Парадигмы и конструкции языков программирования»

Проверил:
преподаватель каф. ИУ5
Гапанюк Ю. Е.

Выполнил:
Студент группы РТ5-31Б
Иванов А. А.

Москва, 2024 г.

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 (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. Пример:

gen_random(5, 1, 3) должен выдать 5 случайных чисел в диапазоне от 1 до 3, например 2, 2, 3, 2, 1

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

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.

Код программы:

```
def field(items, *args):
    assert len(args) > 0, "Необходимо передать хотя бы одно поле для выборки."
    for item in items:
        if len(args) == 1:
            value = item.get(args[0])
            if value is not None:
                yield value
        else:
```

```
            filtered = {a: item[a] for a in args if item.get(a) is not None}
            yield filtered

if __name__ == "__main__":
    goods = [
        {'title': 'Ковер', 'price': 2000, 'color': 'green'},
        {'title': 'Диван для отдыха', 'color': 'black'},
        {'title': 'Стол', 'price': 1500, 'color': None},
    ]

    for i in field(goods, 'title'):
        print(i)

    for i in field(goods, 'title', 'price'):
        print(i)
import random

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

if __name__ == "__main__":
    random_numbers = list(gen_random(5, 1, 3))
    print(random_numbers)

from gen_random import gen_random

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

    def __iter__(self):
        return self

    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.seen:
                self.seen.add(key)
                return item

if __name__ == "__main__":

    data1 = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]
    unique1 = Unique(data1)
    print("Пример 1:")
    for i in unique1:
```

```
        print(i)

    data2 = gen_random(10, 1, 3)
    unique2 = Unique(data2)
    print("Пример 2:")
    for i in unique2:
        print(i)

    data3 = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']
    unique3 = Unique(data3)
    print("Пример 3:")
    for i in unique3:
        print(i)

    unique4 = Unique(data3, ignore_case=True)
    print("Пример 4:")
    for i in unique4:
        print(i)
data = [4, -30, 100, -100, 123, 1, 0, -1, -4]
def abs_key(x):
        return abs(x)

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

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



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
```

```python
@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()
import time
from contextlib import contextmanager
from time import sleep

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

    def __exit__(self, exc_type, exc_val, exc_tb):
        self.end = time.time()
        delta = self.end - self.start
        print(f"time: {delta:.2f} seconds")

@contextmanager
def cm_timer_2():
    start = time.time()
    try:
        yield
    finally:
        end = time.time()
        delta = end - start
        print(f"time: {delta:.2f} seconds")

if __name__ == "__main__":

    print("cm_timer_1:")
    with cm_timer_1():
        sleep(1.5)
```

```
    print("cm_timer_2:")
    with cm_timer_2():
        sleep(2)
import json
from field import field
from unique import Unique
from print_result import print_result
from cm_timer import cm_timer_1
from gen_random import gen_random


path = "C:/Users/Alexa/Desktop/lab4/data_light.json"
with open(path, encoding="utf8") 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 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 = gen_random(len(arg), 100000, 200000)
    return [f"{job}, зарплата {salary} руб." for job, salary in zip(arg,
salaries)]


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

Вывод:

```
Ковер
Диван для отдыха
Стол
{'title': 'Ковер', 'price': 2000}
{'title': 'Диван для отдыха'}
{'title': 'Стол', 'price': 1500}
```

```
[3, 1, 2, 3, 2]
```

```
!!!!!!!!
test_1
1
test_2
iu5
test_3
a = 1
b = 2
test_4
1
2
```

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

```
Пример 1:
1
2
Пример 2:
1
2
3
Пример 3:
a
A
b
B
Пример 4:
a
b
```

```
cm_timer_1:
time: 1.51 seconds
cm_timer_2:
time: 2.01 seconds
```