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»

Лабораторная работа №3

Выполнил:
студент группы ИУ5-34Б
Ковыршин Павел
Подпись и дата:

Проверил:
Гапанюк Ю.Е.
Подпись и дата:

Москва, 2021 г.

Постановка задачи.
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.

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.
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.

field.py

```
def field(dictlist, *keys):
    assert len(keys) > 0
    if len(keys) == 1:
        for d in dictlist:
            yield d[keys[0]]
        return
    result = []
    for d in dictlist:
        rd = {}
        for key in keys:
            if key in d.keys():
                rd[key] = d[key]
        if len(rd) > 0:
            result.append(rd)
        yield rd

goods = [
    {'title': 'Ковер', 'price': 2000, 'color': 'green'},
    {'title': 'Диван для отдыха', 'color': 'black'}
]

if __name__ == "__main__":
    for item in field(goods, 'title'):
        print(item)
    print(list(field(goods, 'title', 'price')))
```

gen_random.py

```python
import random

def gen_random(begin, end, num_count):
    return [random.randint(begin, end) for i in range(num_count)]
```

unique.py

```python
class unique(object):
    def __init__(self, *items, ignore_case=False):
        all_items = []
        for item in items:
            if type(item) is not list:
                all_items.append(item)
                continue
            all_items += item

        self.items = []
        for item in all_items:
            if type(item) is str and ignore_case:
                if item.lower() not in list(map(str.lower, self.items)):
                    self.items.append(item)
                continue
            if item not in self.items:
                self.items.append(item)
        self.pos = -1

    def __next__(self):
        self.pos += 1
        if self.pos >= len(self.items):
            raise StopIteration
        return self.items[self.pos]

    def __iter__(self):
        return self

if __name__ == '__main__':
    for item in unique(1, 2, [1, 2, 3, 4], 7, 0, ignore_case=False):
        print(item)
```

sort.py

```python
def print_sorts(items):
    result = [t[1] for t in sorted([(abs(item), item) for item in items], reverse=True)]
    print("no lambda:", result)

    result_with_lambda = sorted(items, reverse=True, key=lambda x: abs(x))
    print("with lambda:", result_with_lambda)

if __name__ == '__main__':
    data = [-12, 1, -1, 2, 3, -14, 12, 14]
```

```
    print_sorts(data)
```

print_result.py

```
def print_result(func):
    def printer(*args, **kwargs):
        print(func.__name__)
        result = func(*args, **kwargs)
        if type(result) is list:
            for item in result:
                print(item)
            return result
        if type(result) is dict:
            for key in result.keys():
                print(key, '=', result[key])
            return result
        print(result)
        return result
    return printer

@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()
```

cm_timer.py

```
import time
from contextlib import contextmanager

class cm_timer_1:
    def __enter__(self):
        self.begin = time.time()

    def __exit__(self, type, value, traceback):
        print("time:", time.time() - self.begin)

@contextmanager
def cm_timer_2():
    begin = time.time()
    yield
    print("time:", time.time() - begin)

if __name__ == "__main__":
    with cm_timer_1():
        time.sleep(4.7)

    with cm_timer_2():
        time.sleep(4.7)
```

process_data.py

```
import urllib.request
import json
import field
import gen_random
import unique
import print_result
import cm_timer

data_url =
'https://raw.githubusercontent.com/ugapanyuk/BKIT_2021/main/notebooks/fp/files/data_light.
json'

@print_result.print_result
def f1(data):
    return sorted(unique.unique(list(field.field(data, 'job-name')), ignore_case= True))

@print_result.print_result
def f2(data):
    return list(filter(lambda name: name.lower().find('программист') == 0, data))

@print_result.print_result
def f3(data):
    return list(map(lambda word: word + ' с опытом Python', data))

@print_result.print_result
```

```python
def f4(data):
    salaries = gen_random.gen_random(100000, 200000, len(data))
    result = [job_name + ', зарплата ' + str(salary) + 'руб.' for job_name, salary in
zip(data, salaries)]
    return result

if __name__ == '__main__':
    data = json.load(urllib.request.urlopen(data_url))
    with cm_timer.cm_timer_2():
        f4(f3(f2(f1(data))))
```

Результат выполнения программы. (начало не поместилось)

```
шлифовщик механического цеха
эколог
электромонтер -линейщик по монтажу воздушных линий высокого напряжения и контактной сети
электромонтер по испытаниям и измерениям 4-6 разряд
электромонтер станционного телевизионного оборудования
электросварщик
энтомолог
юрисконсульт 2 категории
f2
Программист
Программист / Senior Developer
Программист 1С
Программист C#
Программист C++
Программист C++/C#/Java
Программист/ Junior Developer
Программист/ технический специалист
Программистр-разработчик  информационных систем
f3
Программист с опытом Python
Программист / Senior Developer с опытом Python
Программист 1С с опытом Python
Программист C# с опытом Python
Программист C++ с опытом Python
Программист C++/C#/Java с опытом Python
Программист/ Junior Developer с опытом Python
Программист/ технический специалист с опытом Python
Программистр-разработчик  информационных систем с опытом Python
f4
Программист с опытом Python, зарплата 129371руб.
Программист / Senior Developer с опытом Python, зарплата 199666руб.
Программист 1С с опытом Python, зарплата 177204руб.
Программист C# с опытом Python, зарплата 141554руб.
Программист C++ с опытом Python, зарплата 102850руб.
Программист C++/C#/Java с опытом Python, зарплата 171078руб.
Программист/ Junior Developer с опытом Python, зарплата 164124руб.
Программист/ технический специалист с опытом Python, зарплата 151926руб.
Программистр-разработчик  информационных систем с опытом Python, зарплата 187590руб.
time: 4.485225677490234
```