Московский государственный технический университет имени Н.Э.Баумана

Факультет РТ Радиотехнический

Кафедра ИУ5 Системы обработки информации и управления

Отчет по лабораторной работе № 3 по курсу Базовые компоненты интернет-технологий

Исполнитель

Студент группы РТ5-31Б _____ Корсаков Н.А.

“___”_____2021 г.

Проверил

Доцент кафедры ИУ5 _____ Гапанюк Ю.Е.

“___”_____2021 г.

2021г.

Содержание

1. Описание задания

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. Пример:

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

field(goods, 'title') должен выдавать 'Ковер', 'Диван для отдыха'

field(goods, 'title', 'price') должен выдавать {'title': 'Ковер', 'price': 2000}, {'title': 'Диван для отдыха'}

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.

Пример:

data = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]

Unique(data) будет последовательно возвращать только 1 и 2.

data = gen_random(1, 3, 10)

Unique(data) будет последовательно возвращать только 1, 2 и 3.

data = [‘a’, ‘A’, ‘b’, ‘B’, ‘a’, ‘A’, ‘b’, ‘B’]

Unique(data) будет последовательно возвращать только a, A, b, B.

Unique(data, ignore_case=True) будет последовательно возвращать только a, b.

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.

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.

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. 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).

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.

2. Текст программы

field.py

```
def field(items, *args):
    assert len(args) > 0
    if len(args) == 1:
        for i in items:
            a = i.get(args[0])
            if a != None:
                yield a
    else:
        for i in items:
            dic = {}
            for ar in args:
                a = i.get(ar)
                if a != None:
                    dic[ar] = a
            yield dic
```

gen_random.py

```
from random import randint

def gen_random(num_count, begin, end):
```

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

unique.py

```
class Unique(object):
    def __init__(self, items, **kwargs):
        self.data = items
        self.index = 0
        self.res_data = set()
        if kwargs.get('ignore_case') == None:
            self.ignore_case = False
        else:
            self.ignore_case = kwargs['ignore_case']

    def __next__(self):
        while True:
            if self.index >= len(self.data):
                raise StopIteration
            else:
                current = self.data[self.index]
                self.index = self.index + 1
                if self.ignore_case == False:
                    if current not in self.res_data:
                        self.res_data.add(current)
                        return current
                else:
                    low_data = [i.lower() for i in self.res_data]
                    if current.lower() not in low_data:
                        self.res_data.add(current)
                        return current

    def __iter__(self):
        return self
```

sort.py

```
import operator

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

```
if__name__== '__main__':
    result = list(map(operator.itemgetter(0), sorted([(x, abs(x))  for x in data], key =
operator.itemgetter(1), reverse = True)))
    print(result)

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

print result.py

```
def  print_result(function):
    def decorated_func(*args):
        print(function._name_)
        res = function(*args)
        if type(res) == list:
            for i in res:
                print(i)
        elif type(res) == dict:
            for i in res.keys():
                print(i, ' = ', res[i])
        else:
            print(res)
        return res
    return decorated_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()
```

cm_timer.py

```
from contextlib import contextmanager
import time

class cm_timer_1:

    def__init_(self):
        pass

    def__enter_(self):
        self.time = time.time()

    def__exit_(self, exp_type, exp_value, traceback):
        if exp_type is not None:
            print(exp_type, exp_value, traceback)
        else:
            print(time.time() - self.time)

@contextmanager
def cm_timer_2():
    time_ = time.time()
    yield 7
    print(time.time() - time_)
```

process_data.py

```
from gen_random import gen_random

path = 'data_light.json'

with open(path, 'r', encoding='utf8') as f:
```

```
    data = json.load(f)

@print_result
def f1(arg):
    return sorted(Unique([data[i]["job-name"] for i in range(len(data))], ignore_case=True))

@print_result
def f2(arg):
    return list(filter(lambda x: x[:11].lower() == 'программист', arg))

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

@print_result
def f4(arg):
    salary = gen_random(len(arg), 100000, 200000)
    res = list(zip(arg, salary))
    return [res[i][0] + ', зарплата ' + str(res[i][1]) + ' руб.' for i in range(len(arg))]

if__name__== '_main_':
    with cm_timer_1():
        f4(f3(f2(f1(data))))
```

3. Экранные формы с примерами выполнения программы

```
1 4 8 f1
1С программист
2-ой механик
3-ий механик
4-ый механик
4-ый электромеханик
ASIC специалист
JavaScript разработчик
RTL специалист
Web-программист
[химик-эксперт
web-разработчик
Автожестянщик
Автоинструктор
Автомаляр
Автомойщик
Автор студенческих работ по различным дисциплинам
Автослесарь - моторист
Автоэлектрик
Агент
Агент банка
Агент нпф
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 123348 руб.
Программист / Senior Developer с опытом Python, зарплата 151047 руб.
Программист 1С с опытом Python, зарплата 129711 руб.
Программист C# с опытом Python, зарплата 146344 руб.
Программист C++ с опытом Python, зарплата 184314 руб.
Программист C++/C#/Java с опытом Python, зарплата 139883 руб.
Программист/ Junior Developer с опытом Python, зарплата 106413 руб.
Программист/ технический специалист с опытом Python, зарплата 155770 руб.
Программистр-разработчик  информационных систем с опытом Python, зарплата 109884 руб.
6.099703550338745
```