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»

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

Выполнил:
Студент группы ИУ5-32Б:
Арзамасцев Артем
Подпись и дата:

Проверил:
преподаватель каф. ИУ5
Гапанюк Ю.Е.
Подпись и дата:

Москва, 2022 г.

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 располагаться 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.

Необходимо решить задачу двумя способами:

1. С использованием lambda-функции.
2. 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.

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(items, *args):
    assert len(args) > 0
    if len(args) == 1:
        for i in items:
            for j in i:
                    if j==args[0]:
                        yield i[j]
    else:
        for i in items:
            a={}
            for j in i:
                for h in args:
                    if(j==h):
                        a[h]=i[h]
            yield a

if __name__ == "__main__":
    goods = [
```

```
    {'title': 'Ковер', 'price': 2000, 'color': 'green'},
    {'title': 'Диван для отдыха', 'price': 5300, 'color': 'black'}
    ]
    for i in field(goods,"title"):
        print(i)
    for i in field(goods,"title","price"):
        print(i)
```

gen_random.py

```
import random

def gen_random(quantity,min,max):
    for i in range(quantity):
        yield random.randint(min,max)

if __name__ == "__main__":
    for i in gen_random(5,1,3):
        print(i)
```

unique.py

```
import field
from gen_random import gen_random


class Unique(object):


    def __init__(self,items, ignore_case = False, **kwargs):
        self.items = items
        self.a=kwargs
        self.ignore_case = ignore_case
        self.uniq=set()

        self.index=0

    def __iter__(self):
        return self
    def __next__(self):
        if self.ignore_case == False :
            for i in self.items:
                if(i not in self.uniq):
```

```
                self.uniq.add(i)
                return i
        raise StopIteration
    else:
        for i in self.items:
            try:

                if(i.upper() not in self.uniq):
                    self.uniq.add(i.upper())
                    return i
            except AttributeError:
                if (i not in self.uniq) :
                    self.uniq.add(i)
                    return i
        raise StopIteration



if __name__ == "__main__":
    data = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]
    print(list(Unique(data)))

    data = gen_random(10, 1, 3)
    print(list(Unique(data)))

    data = ["a", "A", "b", "B", "a", "A", "b", "B"]
    print(list(Unique(data)))

    print(list(Unique(data, ignore_case=True)))
```

sort.py

```
if __name__ == "__main__":
    data = [4, -30, 100, -100, 123, 1, 0, -1, -4]


    #без лямбда функции
    result=sorted(data,key=abs,reverse=True)
    print(result)


    #с лямбда функцией
    result=sorted(data,key = lambda x: abs(x),reverse=True)
    print(result)
```

print_result.py

```
def print_result(func):
    def wrapper(*args,**kwargs):
        print(func.__name__)
        a=func(*args,**kwargs)
        if isinstance(a,list):
            for i in a:
                print(i)

        elif isinstance(a,dict):
            for i in a:
                print("{} = {}".format(i,a[i]))
        else:
            print(a)

        return a
    return wrapper

@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__':
    test_1()
    test_2()
```

```
    test_3()
    test_4()
```

cm_timer.py

```
from contextlib import contextmanager
import time


@contextmanager
def cm_timer1():
    start=time.time()
    yield
    print(time.time()-start)



class cm_timer2():

    def __init__(self):
        self.start=0

    def __enter__(self):
        self.start = time.time()

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


if __name__ == "__main__":
    with cm_timer1():
        time.sleep(1)
    with cm_timer2():
        time.sleep(2)
```

process_data.py

```
import json
import field
import gen_random
import unique
import print_result
import cm_timer

@print_result.print_result

def f1(arg):
    return list(unique.Unique(field.field(arg,"job-name"),True))


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


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


@print_result.print_result
def f4(arg):
    return dict(zip(arg,["зарплата " + str(x)+" рублей" for x in
gen_random.gen_random(len(arg),100000,200000)]))

if __name__ == "__main__":

    with open("data_light.json",encoding="utf-8") as f:
        data = json.load(f)
        with cm_timer.cm_timer1():
            f4(f3(f2(f1(data))))
```

Примеры выполнения программ

```
f1
Администратор на телефоне
Медицинская сестра
Охранник сутки-день-ночь-вахта
ВРАЧ АНЕСТЕЗИОЛОГ РЕАНИМАТОЛОГ
теплотехник
разнорабочий
Электро-газосварщик
Водитель Gett/Гетт и Yandex/Яндекс такси на личном автомобиле
Монолитные работы
Организатор - тренер
Помощник руководителя
Автоэлектрик
Врач ультразвуковой диагностики в детскую поликлинику
Менеджер по продажам ИТ услуг (B2B)
Менеджер по персоналу
Аналитик
Воспитатель группы продленного дня
Инженер по качеству
Инженер по качеству 2 категории (класса)
Водитель автомобиля
Пекарь
Переводчик
Терапевт
врач-анестезиолог-реаниматолог
Инженер-конструктор в наружной рекламе
Монтажник-сборщик рекламных конструкций
Оператор фрезерно-гравировального станка
Зоотехник
Сварщик
Рабочий-строитель
врач-трансфузиолог
Юрисконсульт
Специалист отдела автоматизации
Растворщик реагентов
Бармен
Официант
Технолог
Фельдшер-лаборант
Медицинская сестра по физиотерапии
врач функциональной диагностики
Рентгенолаборант
диспетчер по навигации
водитель погрузчика,штабелер
Машинист автогрейдера
наладчик ЧПУ
УПАКОВЩИК-ГРУЗЧИК
Слесарь по ремонту обогатительного оборудования
Слесарь тепловодоснабжения и вентиляции
главный специалист Отдела ЖКХ
Механик по ремонту спецтехники и тракторов
Мастер леса Сосновского участкового лесничества
```

Далее идет большое количество записей

```
Администратор ярмарок выходного дня
оператор тростильного,крутильного оборудования
мотальщица
Водитель категории BCDE
электросварщик
Коммерческий экспедитор
Судокорпусник
Специалист 1 разряда финансового отдела
заместитель директора по воспитательной работе
методист в отдел детского туризма
Разработчик мобильных приложений
директор загородного лагеря
Портной
специалист отдела аренды
Инженер-механик
Разработчик импульсных источников питания
Механик по эксплуатации транспортного отдела
Инженер-технолог по покраске
Бетонщик - арматурщик
главный инженер финансово-экономического отдела
Секретарь судебного заседания в аппарате мирового судьи Железнодорожного судебного района города Ростова-на-Дону
варщик зефира
варщик мармеладных изделий
Оператор склада
Специалист по электромеханическим испытаниям аппаратуры бортовых космических систем
Заведующий музеем в д.Копорье
Документовед
Специалист по испытаниям на электромагнитную совместимость аппаратуры бортовых космических систем
Менеджер (в промышленности)
f2
Программист
Программист C++/C#/Java
Программист 1C
Программистр-разработчик  информационных систем
Программист C++
Программист/ Junior Developer
Программист / Senior Developer
Программист/ технический специалист
Программист C#
f3
Программист с опытом Python
Программист C++/C#/Java с опытом Python
Программист 1C с опытом Python
Программистр-разработчик  информационных систем с опытом Python
Программист C++ с опытом Python
Программист/ Junior Developer с опытом Python
Программист / Senior Developer с опытом Python
Программист/ технический специалист с опытом Python
Программист C# с опытом Python
f4
Программист с опытом Python = зарплата 179107 рублей
Программист C++/C#/Java с опытом Python = зарплата 113569 рублей
Программист 1C с опытом Python = зарплата 117115 рублей
Программистр-разработчик  информационных систем с опытом Python = зарплата 131476 рублей
Программист C++ с опытом Python = зарплата 176057 рублей
Программист/ Junior Developer с опытом Python = зарплата 171119 рублей
Программист / Senior Developer с опытом Python = зарплата 192546 рублей
Программист/ технический специалист с опытом Python = зарплата 172976 рублей
Программист C# с опытом Python = зарплата 111926 рублей
13.601470708847046
>>>
```