Московский государственный технический университет
им. Н. Э. Баумана

Факультет «Информатика и системы управления»

Отчёт по лабораторной работе №3
по курсу «Разработка интернет-приложений»
Функциональные возможности языка Python

Выполнил:
студент группы ИУ5-51Б
Гришин С. В.

Проверил:
преподаватель каф. ИУ5
Гапанюк Ю. Е.

Подпись и дата:
25.10.2021

Подпись и дата:
25.10.2021

Москва, 2021 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 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.

Текст программы.

```
def field(items,*args):
    if (len(args)==0):
        print("Ошибка ввода")
        return 0
    elif (len(args)==1):
        return [dicts[args[0]] for dicts in items if args[0] in dicts]
    else:
        dictionary=list()
        for dicts in items:
            dictionary.append({key:dicts[key]
            for key in dicts if (key in args and (dicts[key]))})
            return dictionary
def main():
    goods = [
    {'title': 'Ковер', 'price': 2000, 'color': 'green'},
    {'title': 'Диван для отдыха', 'color': 'black'}
]
    print(field(goods))
    print(field(goods, 'title'))
```

```
    print(field(goods, 'title', 'price'))
if __name__=="__main__":
    main()
```

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.

Текст программы.

```
from random import randrange
def gen_random(num_count, begin, end):
    return [randrange(begin,end+1) for i in range(num_count)]
if __name__=="__main__":
    print(gen_random(10,1,5))
```

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.

Текст программы.

```
class Unique(object):
    """Итератор, оставляющий только уникальные значения."""
    def __init__(self, items, **kwargs):
        self.used_elements = list()
        self.data = items
        self.index = 0
        if 'ignore_case' in kwargs.keys() and kwargs['ignore_case']==True:
            self.ignore_case=True
        else:
            self.ignore_case=False

    def __iter__(self):
```

```
        return self

    def __next__(self):
        while True:
            if self.index < len(self.data):
                if self.ignore_case==False:
                    current = self.data[self.index]
                    self.index = self.index + 1
                    if current not in self.used_elements:
                    # Добавление в множество производится
                    # с помощью метода add
                        self.used_elements.add(current)
                        return current
                else:
                    current = self.data[self.index]
                    self.index = self.index + 1
                    if isinstance(current, str):
                        current= current.lower()
                    if current not in self.used_elements:
                    # Добавление в множество производится
                    # с помощью метода add
                        self.used_elements.append(current)
                        return current
            else:
                raise StopIteration
if __name__ == "__main__":
    data = ['a','A','b','B','c','C','C','A','D']
    print(data)
    for i in Unique (data,ignore_case=True):
        print (i, end='\t')
    print ('n')
```

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.

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

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.

Текст программы.

```
def print_result(func):
    def decorated_func(*args,**kwargs):
        print(func.__name__)
        if isinstance(func(*args,**kwargs), dict):
            for key in func(*args,**kwargs):
                print(key,'=',(func(*args,**kwargs)[key]))
        elif isinstance(func(*args,**kwargs), list):
            for item in (func(*args,**kwargs)):
                print(item)
        else:
            print(func(*args,**kwargs))
        return (func(*args,**kwargs))
    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()
```

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.
Текст программы.

```
from time import sleep,time
from contextlib import contextmanager

class cm_timer_1():
    def __init__(self):
        self.timer = time()
    def __enter__(self):
        pass
    def __exit__ (self,exp_type,exp_value,traceback):
        self.timer = time() - self.timer
        print ("time:{0:0.1f}".format(self.timer))

@contextmanager
def cm_timer_2():
    t = time()
    yield
    t=time()-t
    print ("time:{0:0.1f}".format(t))
if __name__=="__main__":
    with cm_timer_1():
        sleep(5.5)
    with cm_timer_1():
        sleep(5.5)
```

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.

```
import json
import sys
from gen_random import gen_random
from print_result import print_result
from unique import Unique
from field import field
from cm_timer import cm_timer_1, cm_timer_2

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


@print_result
def f2(data):
   return list(filter(lambda x:'программист' in x[:11], data))
```

```
@print_result
def f3(data):
    return list(map(lambda l: l+ " с опытом Python",data))


@print_result
def f4(data):
   return list(zip(data,list(map(lambda x: "Зарплата " + x +"
руб",map(str,(gen_random(len(data),100000,200000)))))))


if __name__ == '__main__':
    with
open("C:\\Users\\Станислав\\Desktop\\Python\\lab3\\data_light.json","r",encoding=
"utf8") as f:
        data = json.load(f)
    with cm_timer_1():
        f4(f3(f2(f1(data))))
```

Экранные формы с примерами выполнения программы.

```
электросварщик  на полуавтомат
электросварщик на автоматических и полуавтоматических машинах
электросварщик ручной сварки
электросварщики ручной сварки
электрослесарь (слесарь) дежурный и по ремонту оборудования, старший
электрослесарь по ремонту и обслуживанию автоматики и средств измерений электростанций
электрослесарь по ремонту оборудования в карьере
электроэрозионист
эндокринолог
энергетик
энергетик литейного производства
энтомолог
юрисконсульт
юрисконсульт 2 категории
юрисконсульт. контрактный управляющий
программист
программист / senior developer
программист 1с
программист с#
программист с++
программист с++/с#/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 с# с опытом Python
программист с++ с опытом Python
программист с++/с#/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 141875 руб')
('программист / senior developer с опытом Python', 'Зарплата 122817 руб')
('программист 1с с опытом Python', 'Зарплата 161126 руб')
('программист с# с опытом Python', 'Зарплата 192073 руб')
('программист с++ с опытом Python', 'Зарплата 188302 руб')
('программист с++/с#/java с опытом Python', 'Зарплата 133990 руб')
('программист/ junior developer с опытом Python', 'Зарплата 160355 руб')
('программист/ технический специалист с опытом Python', 'Зарплата 181132 руб')
('программистр-разработчик  информационных систем с опытом Python', 'Зарплата 190237 руб')
time:2.1
```