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
«Функциональные возможности языка Python»

Выполнил:
студент группы ИУ5-
32Б
Зайцев А.Д.

Москва, 2021 г.

Общее 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)

Описание задания:

Необходимо реализовать генератор `field`, который последовательно выдает значения ключей словаря.

- В качестве первого аргумента генератор принимает список словарей, дальше через *args генератор принимает неограниченное количество 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):
    assert len(args) > 0
    if len(args) == 1:
        for i in items:
            for j in i: # i - один словарь из списка j - ключ от словаря
                    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)
```

Примеры выполнения программы:

```
C:\Users\Scare\Documents\BKIT1\lab3>python field.py
Ковер
Диван для отдыха
{'title': 'Ковер', 'price': 2000}
{'title': 'Диван для отдыха', 'price': 5300}
```

Задача 2 (файл gen_random.py)

Описание задания:

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:

```
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)
```

Примеры выполнения программы:

```
PS C:\Users\Scare\Documents\BKIT1\lab3> & C:/Users/Scare/AppData/Local/Programs/Python/Python39/python.exe c:/Users/Scare/Documents/BKIT1/lab3/gen_random.py
1
3
3
3
1
PS C:\Users\Scare\Documents\BKIT1\lab3>
```

Задача 3 (файл unique.py)

Описание задания:

- 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:

```
import field



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

    c=True

    a=unique(field.field(goods,"title"),c)
    for i in a:
        print(i)
```

Примеры выполнения программы:

```
PS C:\Users\Scare\Documents\BKIT1\lab3> & C:/Users/Scare/AppData/Local/Programs/Python/Python39/python.exe c:/Users/Scare/Documents/BKIT1/lab3/unique.py
ковер
Диван для отдыха
PS C:\Users\Scare\Documents\BKIT1\lab3>
```

Задача 4 (файл sort.py)

Описание задания:

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:

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


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

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

Примеры выполнения программы:

```
PS C:\Users\Scare\Documents\BKIT1\lab3> & C:/Users/Scare/AppData/Local/Programs/Python/Python39/python.exe c:/Users/Scare/Documents/BKIT1/lab3/sort.py
[123, 100, -100, -30, 4, -4, 1, -1, 0]
[123, 100, -100, -30, 4, -4, 1, -1, 0]
PS C:\Users\Scare\Documents\BKIT1\lab3>
```

Задача 5 (файл print_result.py)

Описание задания:

Необходимо реализовать декоратор `print_result`, который выводит на экран результат выполнения функции.

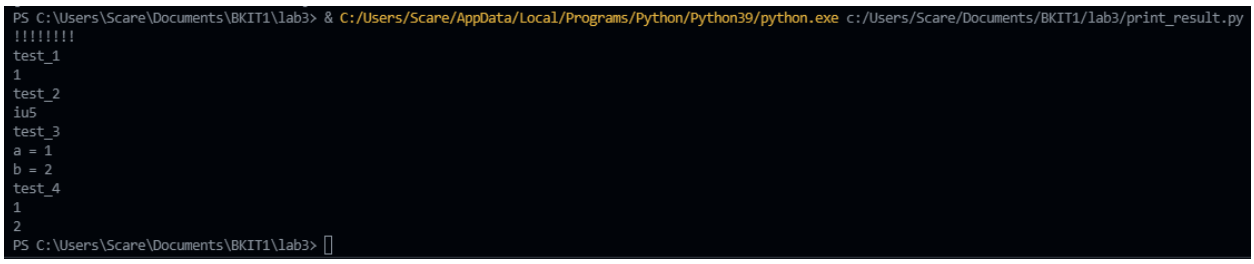
- Декоратор должен принимать на вход функцию, вызывать её, печатать в консоль имя функции и результат выполнения, после чего возвращать результат выполнения.
- Если функция вернула список (`list`), то значения элементов списка должны выводиться в столбик.
- Если функция вернула словарь (`dict`), то ключи и значения должны выводить в столбик через знак равенства.

Текст программы:

```
def print_result(func):
    def 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
```

```python
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()
```

Примеры выполнения программы:



Задача 6 (файл cm_timer.py)

Описание задания:

Необходимо написать контекстные менеджеры `cm_timer_1` и `cm_timer_2`, которые считают время работы блока кода и выводят его на экран.

После завершения блока кода в консоль должно вывестись `time: 5.5`.

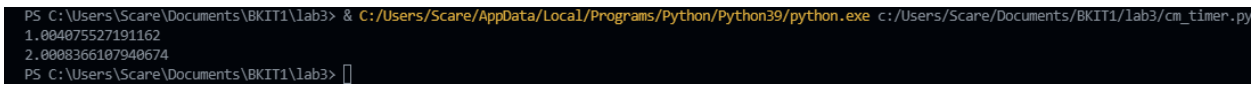
`cm_timer_1` и `cm_timer_2` реализуют одинаковую функциональность, но должны

быть реализованы двумя различными способами (на основе класса и с использованием библиотеки `contextlib`).

Текст программы:

```python
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)
```

Примеры выполнения программы:



Задача 7 (файл process_data.py)

Описание задания:

- 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.

- Функция `f1` должна вывести отсортированный список профессий без повторений (строки в разном регистре считать равными). Сортировка должна игнорировать регистр.

- Функция `f2` должна фильтровать входной массив и возвращать только те элементы, которые начинаются со слова “программист”.

- Функция `f3` должна модифицировать каждый элемент массива, добавив строку “с опытом Python” (все программисты должны быть знакомы с Python).

- Функция `f4` должна сгенерировать для каждой специальности зарплату от 100 000 до 200 000 рублей и присоединить её к названию специальности.

Текст программы:

```
            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))))
```

Пример выполнения программы:

```
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 153693 рублей
Программист C++/C#/Java с опытом Python = зарплата 156246 рублей
Программист 1C с опытом Python = зарплата 160201 рублей
Программистр-разработчик  информационных систем с опытом Python = зарплата 189999 рублей
Программист C++ с опытом Python = зарплата 127645 рублей
Программист/ Junior Developer с опытом Python = зарплата 194696 рублей
Программист / Senior Developer с опытом Python = зарплата 128626 рублей
Программист/ технический специалист с опытом Python = зарплата 152482 рублей
Программист C# с опытом Python = зарплата 168173 рублей
0.11504960060119629

C:\Users\Scare\Documents\BKIT1\lab3>
```