Московский государственный технический университет имени Н. Э. Баумана

Факультет «Информатика и системы управления»

Кафедра ИУ5

Отчёт по

лабораторной работе № 3

«Функциональные возможности языка Python.»

Подготовил:

Алпеев Владислав Сергеевич

Группа ИУ5-54Б

Москва

2020

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.

Задачи.

Задача 1.

- Описание:

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 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 generatorFunctionField(dictionaryList, *fields):
    for currentDictionary in dictionaryList:
        keys = currentDictionary.keys()
        newDictionary = dict(currentDictionary)
        for key in keys:
            if key not in fields:
                newDictionary.pop(key)
            elif newDictionary[key] == None:
                newDictionary.pop(key)
        if newDictionary != {}:
            if len(fields) == 1:
                yield list(newDictionary.values())[0]
            else:
                yield newDictionary

def field(dictionaryList, *fields):
    if len(fields) > 0:
        generatorField = generatorFunctionField(dictionaryList, *fields)
        return list(generatorField)


if __name__ == "__main__":
    dictionaryList = [
        {'title': 'Ковер', 'price': 2000, 'color': 'green'},
        {'title': 'Диван для отдыха', 'color': 'black', 'size': None},
        {'size': None, 'weight': 15}
    ]

    print(field(dictionaryList, 'title'))
    print(field(dictionaryList, 'title', 'price'))
    print(field(dictionaryList, 'title', 'size'))
```

- Экранные формы с примерами выполнения программы.

```
['Ковер', 'Диван для отдыха']
[{'title': 'Ковер', 'price': 2000}, {'title': 'Диван для отдыха'}]
[{'title': 'Ковер'}, {'title': 'Диван для отдыха'}]
```

Задача 2.

- Описание:

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 randint

def gen_randomFunction(num_count, begin, end):
    for number in range(num_count):
        yield randint(begin, end)

def gen_random(num_count, begin, end):
    generatorRandom = gen_randomFunction(num_count, begin, end)
    return list(generatorRandom)

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

- Экранные формы с примерами выполнения программы.

```
[2, 2, 2, 3, 3]
```

Задача 3.

- Описание:

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):
    def __init__(self, items, **kwargs):
        self.items = list(items)
        self.uniqItems = list()
        self.index = 0
        try:
            self.ignore_case = bool(kwargs['ignore_case'])
        except (KeyError, TypeError):
            self.ignore_case = False

    def __next__(self):
        while True:
            if self.index >= len(self.items):
                raise StopIteration
            else:
                current = self.items[self.index]
                if (self.ignore_case == True) & (type(current) == str):
                    current = current.upper()
                if current not in self.uniqItems:
                    self.uniqItems.append(current)
                    return self.items[self.index]
                self.index = self.index + 1
```

```
    def __iter__(self):
        return self

if __name__ == "__main__":
    for i in Unique(['a', 'Б', 'в', 'в', 'В'], ignore_case = True):
        print(i)
    print('---')
    for i in Unique(['a', 'Б', 'в', 'в', 'В'], ignore_case = False):
        print(i)
    print('---')
    for i in Unique([1, 1, 1, 1, 2, 2]):
        print(i)
    print('---')
    for i in Unique([2, 'a', 'Б', 3, 'в', 'в', 'В'], ignore_case = True):
        print(i)
```

- Экранные формы с примерами выполнения программы.

```
a
Б
в
---
a
Б
в
В
---
1
2
---
2
a
Б
3
в
```

Задача 4.

- Описание:

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: с использованием lambda-функции, 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 = sorted(data, key = lambda a: abs(a), reverse=True)
    print(result_with_lambda)
```

- Экранные формы с примерами выполнения программы.

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

Задача 5.

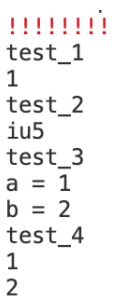
- Описание:

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

```
from colorama import Fore, Style

def print_result(funcForDecorator):
    def decoratedFunc(*args):
        result = funcForDecorator(*args)
        print(funcForDecorator.__name__)
        if type(result) == list:
            for i in result:
                print(i)
        elif type(result) == dict:
            for i,j in result.items():
                print(i, "=", j)
        else:
            print(result)
        return result
    return decoratedFunc


@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(Fore.RED + '!!!!!!!!' + Style.RESET_ALL)
    test_1()
    test_2()
    test_3()
    test_4()
```

- Экранные формы с примерами выполнения программы.



Задача 6.

- Описание:

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 time, sleep
from contextlib import contextmanager

class cm_timer_1():

    def init(self):
        self.timer = 0

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

    def __exit__(self, exp_type, exp_value, traceback):
        print(time() - self.timing)

@contextmanager
def cm_timer_2():
    timer = time()
    yield
    print(time() - timer)

if __name__ == "__main__":

    with cm_timer_1():
        sleep(5.5)

    with cm_timer_2():
        sleep(5.5)
```

- Экранные формы с примерами выполнения программы.

```
5.503041982650757
5.50521993637085
```

Задача 7.

- Описание:

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 lab_python_fp.cm_timer import cm_timer_1
from lab_python_fp.print_result import print_result
from lab_python_fp.unique import Unique
from lab_python_fp.field import field
from lab_python_fp.gen_random import gen_random

path = "/Users/vladislavalpeev/Documents/labs/labsPython/Lab3/data_light.json"

with open(path) as f:
    data = json.load(f)

@print_result
def f1(arg):
    return Unique(field(arg, "job-name"), ignore_case = True)

@print_result
def f2(arg):
    return filter(lambda str: str.capitalize().startswith('Программист'), arg)


@print_result
def f3(arg):
    return map(lambda str: "{} с опытом Python".format(str), arg)


@print_result
def f4(arg):
    arg = list(arg)
    listOfSalaries = list(map(lambda str: "зарплата {} руб.".format(str),
gen_random(len(list(arg)), 100000, 200000)))
    return list(zip(arg, listOfSalaries))


if __name__ == '__main__':
    with cm_timer_1():
        f4(f3(f2(f1(data))))
```

- Экранные формы с примерами выполнения программы.

```
f1
<lab_python_fp.unique.Unique object at 0x103cd4460>
f2
<filter object at 0x103cd44c0>
f3
<map object at 0x103cd44f0>
f4
('Программист с опытом Python', 'зарплата 131509 руб.')
('Программист C++/C#/Java с опытом Python', 'зарплата 142770 руб.')
('Программист 1С с опытом Python', 'зарплата 123883 руб.')
('Программистр-разработчик  информационных систем с опытом Python', 'зарплата 178676 руб.')
('Программист C++ с опытом Python', 'зарплата 119201 руб.')
('Программист/ Junior Developer с опытом Python', 'зарплата 120028 руб.')
('Программист / Senior Developer с опытом Python', 'зарплата 152686 руб.')
('Программист/ технический специалист с опытом Python', 'зарплата 119608 руб.')
('Программист C# с опытом Python', 'зарплата 195421 руб.')
0.10816407203674316
```