Московский государственный технический университет им. Н.Э. Баумана.

Факультет «Информатика и управление»

Кафедра «Системы обработки информации и управления»

Курс «Базовые компоненты интернет-технологий»

Отчет по лабораторной работе №3-4

«Функциональные возможности языка Python»

Выполнил:
студент группы ИУ5-35Б
Терехова Вероника
Подпись:
Дата:

Проверил:
преподаватель каф. ИУ5
Гапанюк Юрий Евгеньевич
Подпись:
Дата:

Москва, 2022 г.

Задание

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(10, 1, 3)

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:

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. 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.

Текст программы

field.py

```
goods = [
    {'title': 'Ковер', 'price': 2000, 'color': 'green'},
```

```
    {'title': 'Диван для отдыха', 'color': 'black'}
]


def field(dicts, *args):
    assert len(args) > 0
    if len(args) == 1:
        for i in dicts:
            if args[0] in i.keys() and i[args[0]]:
                yield i[args[0]]
    elif len(args) > 1:
        for i in dicts:
            new_elem = {}
            for key in args:
                if key in i.keys() and i[key]:
                    new_elem[key] = i[key]
            yield new_elem


# for i in field(goods, 'title', 'price'):
#     print(i)
```

gen_random.py

```
# Пример:
# gen_random(5, 1, 3) должен выдать выдать 5 случайных чисел
# в диапазоне от 1 до 3, например 2, 2, 3, 2, 1
# Hint: типовая реализация занимает 2 строки
from random import randint


def gen_random(num_count, begin, end):
    for _ in range(num_count):
        yield randint(begin, end)


#for i in gen_random(2, 1, 10):
#    print(i)
```

unique.py

```
# Итератор для удаления дубликатов
class Unique(object):
    def __init__(self, items, **kwargs):
        self.items = items
        self.used_elements = set()
        self.index = 0
        self.ignore_case = kwargs['ignore_case'] if 'ignore_case' in
kwargs.keys() else False

    def __iter__(self):
        return self

    def __next__(self):
        while True:
            if next(self.items) is None:
                raise StopIteration
            else:
                current = next(self.items)
                self.index = self.index + 1
                if current.lower() not in self.used_elements:
```

```
                    self.used_elements.add(current.lower())
                    return current
```

sort.py

```
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 x: abs(x), reverse=True)
    print(result_with_lambda)
```

print_result.py

```
# Здесь должна быть реализация декоратора
def print_result(some_func):
    def wrapper(*args, **kwargs):
        print(some_func.__name__)
        res = some_func(*args, **kwargs)
        if type(res) == list:
            for i in res:
                print(i)
        elif type(res) == dict:
            for i in res.keys():
                print(f"{i} = {res[i]}")
        else:
            print(res)
        return res
    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
from time import perf_counter, sleep
```

```
class cm_timer_1:
    def __enter__(self):
        self.start = perf_counter()

    def __exit__(self, exp_type, exp_value, traceback):
        print(perf_counter() - self.start)


@contextmanager
def cm_timer_2():
    start = perf_counter()
    yield
    print(perf_counter() - start)



#with cm_timer_1():
#    sleep(5.5)


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

process_data.py

```
import codecs
import json
from field import field
from gen_random import gen_random
from unique import Unique
from print_result import print_result
from lab_python_fp.cm_timer import cm_timer_1, cm_timer_2

path = "data_light.json"

# Необходимо в переменную path сохранить путь к файлу, который был передан
при запуске сценария

with codecs.open(path, "r", "utf-8") as f:
    data = json.load(f)

# Далее необходимо реализовать все функции по заданию, заменив `raise
NotImplemented`
# Предполагается, что функции f1, f2, f3 будут реализованы в одну строку
# В реализации функции f4 может быть до 3 строк


@print_result
def f1(arg):
    return Unique(field(sorted(arg, key=lambda x: x['job-name']), 'job-
name'))


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


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

```python
@print_result
def f4(arg):
    return list(map(lambda x: f"{x[0]}, зарплата {x[1]} руб.", zip(arg,
gen_random(len(arg), 100000, 200000))))


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

Экранные формы с примерами выполнения программы

```
C:\Users\user\PycharmProjects\lab3\venv\Scripts\python.exe C:\Users\user\PycharmProjects\lab3\lab_python_fp\field.py
{'title': 'Ковер', 'price': 2000}
{'title': 'Диван для отдыха'}

Process finished with exit code 0
```

```
C:\Users\user\PycharmProjects\lab3\venv\Scripts\python.exe C:\Users\user\PycharmProjects\lab3\lab_python_fp\gen_random.py
8
1

Process finished with exit code 0
```

```
C:\Users\user\PycharmProjects\lab3\venv\Scripts\python.exe C:\Users\user\PycharmProjects\lab3\lab_python_fp\sort.py
[123, 100, -100, -30, 4, -4, 1, -1, 0]
[123, 100, -100, -30, 4, -4, 1, -1, 0]

Process finished with exit code 0
```

```
C:\Users\user\PycharmProjects\lab3\venv\Scripts\python.exe C:\Users\user\PycharmProjects\lab3\lab_python_fp\print_result.py
test_1
1
test_2
iu5
test_3
a = 1
b = 2
test_4
1
2

Process finished with exit code 0
```

```
C:\Users\user\PycharmProjects\lab3\venv\Scripts\python.exe C:\Users\user\PycharmProjects\lab3\lab_python_fp\cm_timer.py
5.5007397000000005
5.508943900000001

Process finished with exit code 0
```

```
C:\Users\user\PycharmProjects\lab3\venv\Scripts\python.exe C:\Users\user\PycharmProjects\lab3\lab_python_fp\process_data.py
f1
<unique.Unique object at 0x000002241F77B088>
f2
Программист
Программист 1С
Программист C#
Программист C++/C#/Java
Программист/ технический специалист
f3
Программист с опытом Python
Программист 1С с опытом Python
Программист C# с опытом Python
Программист C++/C#/Java с опытом Python
Программист/ технический специалист с опытом Python
f4
Программист с опытом Python, зарплата 111007 руб.
Программист 1С с опытом Python, зарплата 189692 руб.
Программист C# с опытом Python, зарплата 139974 руб.
Программист C++/C#/Java с опытом Python, зарплата 127105 руб.
Программист/ технический специалист с опытом Python, зарплата 107268 руб.
0.015467000000000009

Process finished with exit code 0
```