Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования "Московский государственный технический университет им. Н. Э. Баумана (национальный исследовательский университет)"

Факультет "Информатика и системы управления"
Кафедра ИУ5 "Системы обработки информации и управления"

Отчёт по лабораторной работе №3

По дисциплине "Базовые компоненты интернет-технологий"

Выполнил:
студент группы ИУ5-35Б
Чернецов С.А.

Москва, 2021 г.

Постановка задачи:
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.

Шаблон для реализации генератора:

```
# Пример:
# goods = [
#    {'title': 'Ковер', 'price': 2000, 'color': 'green'},
#    {'title': 'Диван для отдыха', 'price': 5300, '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', 'price': 5300}

def field(items, *args):
    assert len(args) > 0
    # Необходимо реализовать генератор
```

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`

Шаблон для реализации генератора:

```
# 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
def gen_random(num_count, begin, end):
    pass
    # Необходимо реализовать генератор
```

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(1, 3, 10)`
`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.
Шаблон для реализации класса-итератора:

```
# Итератор для удаления дубликатов
class Unique(object):
    def __init__(self, items, **kwargs):
        # Нужно реализовать конструктор
        # В качестве ключевого аргумента, конструктор должен принимать bool-параметр
ignore_case,
        # в зависимости от значения которого будут считаться одинаковыми строки в разном
регистре
        # Например: ignore_case = True, Абв и АБВ - разные строки
        #           ignore_case = False, Абв и АБВ - одинаковые строки, одна из которых
удалится
        # По-умолчанию ignore_case = False
        pass

    def __next__(self):
        # Нужно реализовать __next__
        pass

    def __iter__(self):
        return self
```

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.

Шаблон реализации:

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

if __name__ == '__main__':
    result = ...
    print(result)

    result_with_lambda = ...
    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.

Шаблон реализации:

```
# Здесь должна быть реализация декоратора

@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()
```
Результат выполнения:

```
test_1
1
test_2
iu5
test_3
a = 1
b = 2
test_4
1
2
```

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.

Шаблон реализации:

```
import json
import sys
# Сделаем другие необходимые импорты

path = None

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

with open(path) 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):
    raise NotImplemented


@print_result
def f2(arg):
    raise NotImplemented


@print_result
def f3(arg):
    raise NotImplemented


@print_result
def f4(arg):
    raise NotImplemented


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

Код:

field.py

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


def field(items, *args):
    assert len(args) > 0
    if len(args) == 1:
        for i in items:
            yield i.get(args[0])
    else:
        for i in items:
            a = dict()
            for arg in args:
                if i.get(arg) != None:
                    a[arg] = i.get(arg)
            yield a

if __name__ == '__main__':
    print(*field(goods, 'title', 'price'))
    print(list(field(goods, 'title')))
```

gen_random.py

```
import random
def gen_random(num_count, begin, end):
    for i in range(num_count):
        yield random.randint(begin, end)
if __name__ == '__main__':
    print(*gen_random(5, 1, 3))
```

unique.py

```
class Unique(object):
    def __init__(self, items, ignore_case = False, **kwargs):
        self.set = set()
        self.items = items
        self.ignore_case = ignore_case
        self.kwargs = kwargs

    def __next__(self):
        i = iter(self.items)
        while True:
            try:
                current = next(i)
            except StopIteration:
                raise
            else:
                if self.ignore_case == True and isinstance(current, str):
                    a = current[:]
                    if a.lower() not in self.set:
                        self.set.add(a.lower())
                        return current
                elif current not in self.set:
                    self.set.add(current)
                    return current

    def __iter__(self):
```

```
        return self
if __name__ == '__main__':
    data = ["a", "A", "b", "B", "a", "A", "b", "B"]
    data1 = ["A", "a", "b"]
    print(*Unique(data))
    print(*Unique(data, True))
    print(*Unique(data1, True))
```

sort.py

```
data = [4, -30, 100, -100, 123, 1, 0, -1, -4]
result = sorted(data, key=abs, reverse=True)
result_with_lambda = sorted(data, key=lambda x: x if x >= 0 else -x, reverse=True)

if __name__ == '__main__':
    print(result)
    print(result_with_lambda)
```

cm_timer.py

```
from contextlib import contextmanager
import time


class cm_timer_1:
    def __init__(self):
        self.start_time = None

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

    def __exit__(self, a, b, c):
        print(f'time: {time.time() - self.start_time}')


@contextmanager
def cm_timer_2():
    start_time = time.time()
    yield
    print(f'time: {time.time() - start_time}')

if __name__ == '__main__':
    with cm_timer_1():
        time.sleep(5.5)

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

print_result.py

```
def print_result(func):
    def decorated_func(*args, **kwargs):
        fun = func(*args, **kwargs)
        print(func.__name__)
        if isinstance(fun, list):
            if isinstance(fun[0], tuple):
                for i in fun:
                    print(*i, end=' ')
            else:
                print(*fun)
        elif isinstance(fun, dict):
            for a, b in fun.items():
                print(f'{a} = {b}')
```

```
        else:
            print(fun)
        return fun
    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__':
    test_1()
    test_2()
    test_3()
    test_4()
```

process_data.py

```
# -*- coding utf-8 -*-
import json

import print_result, field, gen_random, unique, cm_timer

path = 'C:\data_light.json'
try:
    with open(path, "r", encoding="utf-8") as f:
        data = json.load(f)
except FileNotFoundError:
    print('Ошибка')
    raise SystemExit

@print_result.print_result
def f1(arg):
    return sorted(unique.Unique(list(field.field(arg, 'job-name'))))


@print_result.print_result
def f2(arg):
    return list(filter(lambda x: True if 'программист' in x else False, 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):
    a = list(gen_random.gen_random(len(arg), 100000, 200000))
    a = ['зарплата ' + str(i) + ' руб.' for i in a]
    return list(zip(arg, a))


if __name__ == '__main__':
```

```
    with cm_timer.cm_timer_1():
        f4(f3(f2(f1(data))))
```

Анализ результатов:

```
C:\Users\Семён\PycharmProjects\Lab3\venv\Scripts\python.exe C:/Users/Семён/PycharmProjects/Lab3/field.py
{'title': 'Ковер', 'price': 2000} {'title': 'Диван для отдыха'}
['Ковер', 'Диван для отдыха']

Process finished with exit code 0
```

```
C:\Users\Семён\PycharmProjects\Lab3\venv\Scripts\python.exe C:/Users/Семён/PycharmProjects/Lab3/gen_random.py
1 1 2 3 3

Process finished with exit code 0
```

```
C:\Users\Семён\PycharmProjects\Lab3\venv\Scripts\python.exe C:/Users/Семён/PycharmProjects/Lab3/unique.py
a A b B
a b
A b

Process finished with exit code 0
```

```
C:\Users\Семён\PycharmProjects\Lab3\venv\Scripts\python.exe C:/Users/Семён/PycharmProjects/Lab3/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\Семён\PycharmProjects\Lab3\venv\Scripts\python.exe C:/Users/Семён/PycharmProjects/Lab3/cm_timer.py
time: 5.50337028503418
```

```
1
test_2
iu5
test_3
a = 1
b = 2
test_4
1 2

Process finished with exit code 0
```

```
1
1С программист 2-ой механик 3-ий механик 4-ый механик 4-ый электромеханик ASIC специалист JavaScript разработчик RTL специалист Web-программист Web-разработчик [химик-эксперт web-р
f2
1С программист Web-программист Веб - программист (PHP, JS) / Web разработчик Веб-программист Ведущий инженер-программист Ведущий программист Инженер - программист АСУ ТП Инженер-пр
f3
1С программист с опытом Python Web-программист с опытом Python Веб - программист (PHP, JS) / Web разработчик с опытом Python Веб-программист с опытом Python Ведущий инженер-програм
f4
1С программист с опытом Python зарплата 180845 руб. Web-программист с опытом Python зарплата 164042 руб. Веб - программист (PHP, JS) / Web разработчик с опытом Python зарплата 1784

Process finished with exit code 0
```