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

Факультет «Информатика и управление»

Кафедра «Системы обработки информации и управления»

Курс «БКИТ»
Отчет по лабораторной работе №3-4
«Функциональные возможности языка Python»

Выполнил:
студент группы ИУ5-35Б
Большаков Георгий
Подпись и дата:

Проверил:
преподаватель каф. ИУ5
Нардид А.Н.
Подпись и дата:

Москва, 2022 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. 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(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.
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))))
```

Текст программы

Файлы пакета lab_python_fp:

field.py

```
def field(items, *args):

    assert len(args) > 0
    if len(args) == 1:
        for data in items:
            current = data.get(args[0])
            if current is not None:
                yield current
    else:
        for elem in items:
            data = dict()
            for arg in args:
                current = elem.get(arg)
                if current is not None:
                    data[arg] = current
            if len(data) != 0:
                yield data
```

gen_random.py

```
import random

def gen_random(num_count, begin, end):
    for x in range(num_count):
        yield random.randrange(begin, end+1)
```

unique.py

```
from gen_random import gen_random

class Unique(object):
    def __init__(self, items, **kwargs):
        self.used_elements = set()
        self.data = list(items)
        #print(self.data)
        self.index = 0

        self.ignore_case = False
        if len(kwargs) > 0:
            self.ignore_case = kwargs['ignore_case']

    def __next__(self):
        while True:
            if self.index >= len(self.data):
                raise StopIteration
            else:
                current = self.data[self.index]
                self.index += 1
                if self.ignore_case == True and type(current) == str:
```

```
                    current = current.lower()
                if current not in self.used_elements:
                    self.used_elements.add(current)
                    return current

    def __iter__(self):
        return self
```

sort.py

```
def lambdaSort(lst):
    return sorted(lst, key=lambda x: abs(x), reverse=True)


def no_lambdaSort(lst):
    return sorted(lst, key=abs, reverse=True)
```

print_result.py

```
def print_result(func):
    def wrapper(*args, **kwargs):
        funcInn = func(*args, **kwargs)
        print()
        print(func.__name__)
        if type(funcInn) == list:
            for i in funcInn:
                print(i)

        if type(funcInn) == dict:
            for k, v in funcInn.items():
                print(k, ' = ', v)

        else:
            print(funcInn)

        return funcInn

    return wrapper
```

cm_timer.py

```
from contextlib import contextmanager
import time


class cm_timer_1():

    def __init__(self):
        self.start_tm = None

    def __enter__(self):
        self.start_tm = time.time()
        return self

    def __exit__(self, exp_type, exp_value, traceback):  # для обработки
```

```
исключений
        print(time.time() - self.start_tm)


@contextmanager
def cm_timer_2():
    start = time.time()
    yield
    print(time.time() - start)
```

process_data.py

```
from print_result import print_result
from field import field
from unique import Unique
from gen_random import gen_random
from cm_timer import cm_timer_1
import json

with open('data_light.json', encoding='utf-8') as f:
    data = json.load(f)

@print_result
def f1(arg):
    return sorted(list(Unique(field(arg, 'job-name'), ignore_case=True)),
key=str.lower)

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

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


@print_result
def f4(arg):
    zip_str = zip(arg, gen_random(len(arg), 100000, 200000))
    str_ans = ['{}, зарплата {} руб.'.format(a, b) for a, b in zip_str]
    return str_ans

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

Экранные формы

```
f1
1с программист
2-ой механик
3-ий механик
4-ый механик
4-ый электромеханик
[химик-эксперт
asic специалист
javascript разработчик
rtl специалист
web-программист
web-разработчик
автожестянщик
автоинструктор
автомаляр
автомойщик
автор студенческих работ по различным дисциплинам
автослесарь
автослесарь - моторист
автоэлектрик
агент
агент банка
агент нпф
агент по гос. закупкам недвижимости
агент по недвижимости

f2
программист
программист / senior developer
программист 1с
программист c#
программист c++
программист c++/c#/java
программист/ junior developer
программист/ технический специалист
программистр-разработчик  информационных систем
['программист', 'программист / senior developer', 'программист 1с',
```

```
f3
программист с опытом Python
программист / senior developer с опытом Python
программист 1с с опытом Python
программист c# с опытом Python
программист c++ с опытом Python
программист c++/c#/java с опытом Python
программист/ junior developer с опытом Python
программист/ технический специалист с опытом Python
программистр-разработчик  информационных систем с опытом Python
['программист с опытом Python', 'программист / senior developer с опытом Python',
f4
программист с опытом Python, зарплата 176520 руб.
программист / senior developer с опытом Python, зарплата 129923 руб.
программист 1с с опытом Python, зарплата 173719 руб.
программист c# с опытом Python, зарплата 166000 руб.
программист c++ с опытом Python, зарплата 190878 руб.
программист c++/c#/java с опытом Python, зарплата 123075 руб.
программист/ junior developer с опытом Python, зарплата 102899 руб.
программист/ технический специалист с опытом Python, зарплата 167323 руб.
программистр-разработчик  информационных систем с опытом Python, зарплата 136530 руб.
['программист с опытом Python, зарплата 176520 руб.', 'программист / senior developer с опытом Python,
0.05381011962890625
```