Московский государственный технический университет им. Н.Э. Баумана

Факультет «Радиотехнический»
Кафедра ИУ5 «Системы обработки информации и управления»

Отчёт по лабораторной работе №3.
“ Функциональные возможности языка Python.”

Выполнил:
студент группы РТ5-31Б
Агеев Алексей
Подпись и дата:

Проверил:
преподаватель каф. ИУ5
Гапанюк Ю.Е.
Подпись и дата:

Москва, 2021 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(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))))
```

Текст программы и результат работы

Field.py

```
def field(items, *args):
    num  = len(args)
    for temp in items:
        gen = {} #возвращаемое значение
        for attr in temp:
            if(num): # если есть доп аргументы
                for requared in args:
                    if (attr == requared or temp[attr] == requared):
                        gen[attr] = temp[attr]
                        yield gen
            else:
                gen.append(temp[attr])
                yield gen




goods = [
   {'title': 'Ковер', 'price': 2000, 'color': 'green'},
   {'title': 'Диван для отдыха', 'price': 5300, 'color': 'black'}
   ]
for i in field(goods, "Ковер"):
    print(i)
```

```
{'title': 'Ковер'}
```

gen_random.py

```
from random import randrange
def gen_random(num_count, begin, end):
    for i in range(0, num_count):
        yield randrange(begin, end+1)

for i in gen_random(10, 100000, 200000):
    print(i)
```

```
130800
109559
182399
124798
104991
166501
166127
128454
176479
111460
```

unique.py

```python
from typing import Iterator
from gen_random import gen_random
class Unique(object):
    def __init__(self, items, ignore_case = False):
        self.used_elem = []
        self.data = items
        self.index = 0
        self.ignore_case = ignore_case
        self.iterator = iter(self.data)

    def __iter__(self):
        return self

    def __next__(self):
        while True:
            current = next(self.iterator)
            try:
                if(self.ignore_case):
                    try:
                        temp = current.lower()
                    except:
                        temp = current
                    if temp not in self.used_elem:
                        self.used_elem.append(temp)
                        return current
            except StopIteration:
                break
        raise StopIteration

data = ["Hello", "Hi", "Hello"]

for i in Unique(data, ignore_case=True):
    print(i)
```

```
Hello
Hi
```

sorted1.py

```python
def sorter(list, compare = None):
    return sorted(list, key = compare, reverse = True)
```

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

goods = {'title': 200, 'price': 2000, 'color': 900}

print(sorter(goods.items(), lambda value: value[1]))
```

```
[('price', 2000), ('color', 900), ('title', 200)]
```

print_result.py

```
def print_result(function):
    def printer_of_func(*args):
        print("Name of func: {}".format(function.__name__))
        res = function(*args)
        if (type(res) == list):
            for i in res:
                print(i)
        elif (type(res) == dict):
            for key in res:
                print("{} = {}".format(key, res[key]))
        return res
    return printer_of_func

def Hello():
    return ["Hello", "Hi"]

["Hello", "Hi"]
{'title': 'Ковер', 'price': 2000, 'color': 'green'}
print_result(Hello)
```

```
Name of func: Hello
Hello
Hi
```

cm_timer.py

```
import time
from contextlib import contextmanager
class cm_timer_1:

    def __init__(self):
        self.before_time = float(0)

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

    def __exit__(self, exp_type, exp_value, traceback):
        if exp_type is not None:
            print(exp_type, exp_value, traceback)
        else:
            print("time: {:0.4f}".format(time.time() - self.before_time))

@contextmanager
```

```python
def cm_timer_2():
    before = time.time()
    yield
    print("time: {:0.4f}".format(time.time() - before))

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

```
time: 5.5090
```

process_data.py

```python
from os import closerange
from cm_timer import cm_timer_1
from print_result import print_result
from sorted1 import sorter
from gen_random import gen_random
from Unique import Unique
from field import field
import json

@print_result
def f1(arg):
    return sorted(list(Unique(list(field(arg, "job-name")), ignore_case=True)),
key=lambda x: x["job-name"].lower())

@print_result
def f2(item):
    return list(filter(field, item))

@print_result
def f3(arg):
    return list(map(lambda x: x["job-name"] +' с опытом Python', arg))

@print_result
def f4(item):
    payroll = list(gen_random(len(item), 100000, 200000))
    return ["{}, зарплата {} руб.".format(job, salary) for job, salary in
zip(item, payroll)]

path = r"C:\Users\prite\lab_python_fp\data.json"

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

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

```
ЭЛЕКТРОМОНТЕР  ПО  РЕМОНТУ  И  ОБСЛУЖИВАНИЮ  ОБОРУДОВАНИЯ с опытом Python, зарплата 140826 руб.
электромонтер -линейщик по монтажу воздушных линий высокого напряжения и контактной сети с опытом Python, зарплата 174421 руб.
Электромонтер контактной сети с опытом Python, зарплата 115528 руб.
Электромонтер линейных сооружений телефонной связи и радиофикации с опытом Python, зарплата 171426 руб.
Электромонтер охранно-пожарной сигнализации с опытом Python, зарплата 141496 руб.
Электромонтер охранно-пожарной сигнализации и систем видеонаблюдения с опытом Python, зарплата 149991 руб.
электромонтер по испытаниям и измерениям 4-6 разряд с опытом Python, зарплата 124475 руб.
Электромонтер по обслуживанию электрооборудования электростанций с опытом Python, зарплата 143291 руб.
Электромонтер по ремонту и обслуживанию  и ремонту электрооборудования с опытом Python, зарплата 155196 руб.
Электромонтер по ремонту и обслуживанию оборудования с опытом Python, зарплата 154991 руб.
Электромонтер по ремонту и обслуживанию оборудования подстанций с опытом Python, зарплата 125421 руб.
Электромонтер по ремонту и обслуживанию электрооборудования с опытом Python, зарплата 113926 руб.
электромонтер по ремонту и обслуживанию электрооборудования с опытом Python, зарплата 176664 руб.
Электромонтер по ремонту и обслуживанию электрооборудования 4 разряда-5 разряда с опытом Python, зарплата 190337 руб.
Электромонтер по ремонту и обслуживанию электрооборудования 5 р. с опытом Python, зарплата 111605 руб.
Электромонтер по ремонту и обслуживанию электрооборудования 5 разряда с опытом Python, зарплата 100994 руб.
Электромонтер по ремонту и обслуживанию электрооборудования 6 разряда-6 разряда с опытом Python, зарплата 105985 руб.
Электромонтер по ремонту оборудования ЗИФ с опытом Python, зарплата 162432 руб.
Электромонтер по ремонту электрооборудования ГПМ с опытом Python, зарплата 107800 руб.
Электромонтер по эксплуатации и ремонту оборудования с опытом Python, зарплата 160119 руб.
электромонтер станционного телевизионного оборудования с опытом Python, зарплата 119472 руб.
Электронщик с опытом Python, зарплата 159958 руб.
электросварщик с опытом Python, зарплата 167966 руб.
Электросварщик  на полуавтомат с опытом Python, зарплата 189570 руб.
Электросварщик на автоматических и полуавтоматических машинах с опытом Python, зарплата 107416 руб.
электросварщик на автоматических и полуавтоматических машинах с опытом Python, зарплата 193033 руб.
Электросварщик ручной сварки с опытом Python, зарплата 182600 руб.
Электросварщики ручной сварки с опытом Python, зарплата 195586 руб.
Электрослесарь (слесарь) дежурный и по ремонту оборудования, старший с опытом Python, зарплата 197076 руб.
Электрослесарь по ремонту и обслуживанию автоматики и средств измерений электростанций с опытом Python, зарплата 117395 руб.
Электрослесарь по ремонту оборудования в карьере с опытом Python, зарплата 120547 руб.
Электроэрозионист с опытом Python, зарплата 158098 руб.
Эндокринолог с опытом Python, зарплата 153204 руб.
Энергетик с опытом Python, зарплата 101103 руб.
Энергетик литейного производства с опытом Python, зарплата 197134 руб.
энтомолог с опытом Python, зарплата 178281 руб.
Юрисконсульт с опытом Python, зарплата 185339 руб.
юрисконсульт с опытом Python, зарплата 170413 руб.
юрисконсульт 2 категории с опытом Python, зарплата 162356 руб.
Юрисконсульт. Контрактный управляющий с опытом Python, зарплата 141706 руб.
Юрист с опытом Python, зарплата 171070 руб.
юрист с опытом Python, зарплата 178110 руб.
Юрист (специалист по сопровождению международных договоров, английский - разговорный) с опытом Python, зарплата 164738 руб.
Юрист волонтер с опытом Python, зарплата 156892 руб.
Юристконсульт с опытом Python, зарплата 130625 руб.
time: 2.9139
```