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

**Факультет «Радиотехнический»**

**Кафедра ИУ5 «Системы обработки информации и управления»**

Отчет по Лаб.3-4
«Парадигмы и конструкции языков программирования»

Выполнил:
студент группы РТ5-31Б
Иванченко Д.А.

Проверил:
Гапанюк Ю. Е

2023 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.

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:

С использованием lambda-функции.

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

**field.py**

```
def field(items, *args):
    assert len(args) > 0 # для отладки

    for i in items:
        if(len(args) == 1):
            yield i.get(args[0])
        else:
            dict = {}
            for j in args:
                if(i.get(j)!=None): dict[j]=i.get(j)
            yield dict

def act(data, str):
    a=[]
    for i in field(data, str):
        a.append(i)
    return a

def main():
```

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

**gen_random.py**

```
from random import randint

def gen_random(count, begin, end):
    return (f', зарплата {randint(begin, end)} руб.' for i in range(count))

def main():
    nums = gen_random(5, 1, 3)
    for i in nums:
        print(i, end=' ')
```

**unique.py**

```
import gen_random
class Unique:
    def __init__(self, items, **kwargs):
        self.items = iter(items)
        self.ignore_case = kwargs.get('ignore_case', False)
        self.set = set()

    def __iter__(self):
        return self

    def __next__(self):
        item = next(self.items)
        if isinstance(item, str) and self.ignore_case:
            key = item.lower()
        else:
            key = item
        if key not in self.set:
            self.set.add(key)
            return key
def run(data, bool):
    unique_iter = Unique(data, ignore_case=bool)
    a=[]
    for item in unique_iter:
        output = item
        if (output != None): a.append(output)
    data.clear()
    return a

def main():
    # data = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]
    data = gen_random.gen_random(10, 1, 5)
    # data = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']
    unique_iter = Unique(data, ignore_case=False)
    for item in unique_iter:
        output = item
        if (output != None): print(output, end=' ')
```

**sort.py**

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

def mysort(mass):
    return sorted(mass, key=abs, reverse=True)
```

```
def main():
    result = sorted(data, key=abs, reverse=True)
    print(result)
    lambda_r = sorted(data, key=lambda i:abs(i), reverse=True)
    print(lambda_r)
```

### print_result.py

```
def print_result(func):
    def wrapper(*args):
        print(func.__name__)
        a=func(*args)
        if(isinstance(a, list)):
            for i in a:
                if(isinstance(i, tuple)):
                    print(*i)
                else:
                    print(i)
        elif(isinstance(a, dict)):
            for i in a.keys():
                print(f'{i} = {a[i]}')


        else: print(a)
        return a
    return wrapper



def main():
    @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]
    test_1()
    test_2()
    test_3()
    test_4()
```

### cm_timer.py

```
import time
from contextlib import contextmanager

class cm_timer_1:
    def __enter__(self):
        self.start_time = time.time()
        return self

    def __exit__(self, exc_type, exc_val, exc_tb):
        elapsed_time = time.time() - self.start_time
        print(f"time: {elapsed_time}")
```

```
@contextmanager
def cm_timer_2():
    start_time = time.time()
    yield
    end_time = time.time()
    execution_time = end_time - start_time
    print(f"time: {execution_time}")
```

**process_data.py**

```
import json
import sys
import print_result
import field
import cm_timer
import sort
import unique
import gen_random
# Сделаем другие необходимые импорты

path = 'data light.json'

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

with open(path, encoding='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.print_result
def f1():
    return sort.mysort(unique.run(field.act(data, 'job-name'), True))

@print_result.print_result
def f2(arg):
    return list(filter(lambda it: 'программист' in it, arg))
@print_result.print_result
def f3(arg):
    return list(map(lambda i: i+' с опытом Python', arg))
@print_result.print_result
def f4(arg):
    return list(zip(arg, gen_random.gen_random(len(arg), 100000, 200000)))


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

**Вывод**

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

.....

1с программист
web-программист
веб - программист (php, js) / web разработчик
веб-программист
ведущий инженер-программист
ведущий программист
инженер - программист
инженер - программист асу тп
инженер-программист
инженер-программист (клинский филиал)
инженер-программист (орехово-зуевский филиал)
инженер-программист 1 категории
инженер-программист ккт
инженер-программист плис
инженер-программист сапоу (java)
инженер-электронщик (программист асу тп)
педагог программист
помощник веб-программиста
программист
программист / senior developer
программист 1с
программист c#
программист c++
программист c++/c#/java
программист/ junior developer
программист/ технический специалист
программистр-разработчик  информационных систем
системный программист (c, linux)

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-программист с опытом Python
ведущий программист с опытом Python
инженер - программист с опытом Python
инженер - программист асу тп с опытом Python
инженер-программист с опытом Python
инженер-программист (клинский филиал) с опытом Python
инженер-программист (орехово-зуевский филиал) с опытом Python
инженер-программист 1 категории с опытом Python
инженер-программист ккт с опытом Python
инженер-программист плис с опытом Python
инженер-программист сапоу (java) с опытом Python
инженер-электронщик (программист асу тп) с опытом Python
педагог программист с опытом Python
помощник веб-программиста с опытом Python
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

f4

1с программист с опытом Python , зарплата 171579 руб.
web-программист с опытом Python , зарплата 184174 руб.
веб - программист (php, js) / web разработчик с опытом Python , зарплата 128440 руб.
веб-программист с опытом Python , зарплата 151625 руб.
ведущий инженер-программист с опытом Python , зарплата 108755 руб.
ведущий программист с опытом Python , зарплата 195005 руб.
инженер - программист с опытом Python , зарплата 183546 руб.
инженер - программист асу тп с опытом Python , зарплата 112060 руб.
инженер-программист с опытом Python , зарплата 192596 руб.
инженер-программист (клинский филиал) с опытом Python , зарплата 164605 руб.
инженер-программист (орехово-зуевский филиал) с опытом Python , зарплата 149506 руб.
инженер-программист 1 категории с опытом Python , зарплата 161655 руб.
инженер-программист ккт с опытом Python , зарплата 153435 руб.
инженер-программист плис с опытом Python , зарплата 190729 руб.
инженер-программист сапоу (java) с опытом Python , зарплата 127529 руб.
инженер-электронщик (программист асу тп) с опытом Python , зарплата 153346 руб.
педагог программист с опытом Python , зарплата 161126 руб.
помощник веб-программиста с опытом Python , зарплата 101792 руб.
программист с опытом Python , зарплата 187965 руб.
программист / senior developer с опытом Python , зарплата 194288 руб.
программист 1с с опытом Python , зарплата 133266 руб.
программист c# с опытом Python , зарплата 138575 руб.
программист c++ с опытом Python , зарплата 103397 руб.
программист c++/c#/java с опытом Python , зарплата 157375 руб.
программист/ junior developer с опытом Python , зарплата 158603 руб.