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 ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ

КАФЕДРА СИСТЕМЫ ОБРАБОТКИ ИНФОРМАЦИИ И УПРАВЛЕНИЯ

Отчет по лабораторной работе №3

Студент Вардумян Арсен Тигранович
фамилия, имя, отчество

Группа ИУ5-51Б

Студент 18.12.2021 **Вардумян А.Т.**
подпись, дата фамилия, и.о.

Преподаватель 18.12.2021 **Гапанюк Ю.Е.**
подпись, дата фамилия, и.о.

2021г.

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 = False, Абв и АБВ - разные строки
        #           ignore_case = True, 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:

“cmTimer1.py”:

```
import time


class cmTimer1():
    def __enter__(self):
        self._start_time = time.perf_counter()
        return self._start_time

    def __exit__(self, type, value, traceback):
        del type, value, traceback

        elapsedTime = time.perf_counter() - self._start_time
        self._start_time = None

        print()
        print(
            f'\u001b[37mElapsed time: \u001b[1;35m{elapsedTime:5f}\u001b[37m seconds\u001b[0m')
        print()
```

“cmTimer2.py”:

```
from contextlib import contextmanager
import time


@contextmanager
def cmTimer2():
```

```
    try:
        startTime = time.perf_counter()
        yield startTime

    finally:
        elapsedTime = time.perf_counter() - startTime

        print()
        print(
            f'\u001b[37mElapsed time: \u001b[1;35m{elapsedTime:5f}\u001b[37m seconds\u001b[0m')
        print()
```

“field.py”:

```
def fieldGenerator(items, *args):
    assert len(args) > 0

    result = []

    for item in items:
        temp = {}
        for arg in args:
            value = item.get(arg)
            if value != None:
                temp.update({arg: value})
        if temp:
            if len(args) == 1:
                result.append(temp[arg])
            else:
                result.append(temp)

    return result
```

“genRandom.py”:

```
import random


def genGenerator(num_count, begin, end):
    return (random.randint(begin, end) for _ in range(num_count))
```

“printResult.py”:

```
from sys import argv


def printResult(func):
    def wrapper(*args):
        print(f'\n\u001b[1;34;4m{func.__name__}\u001b[0m\n')

        result = func(*args)
```

```
        if isinstance(result, list):
            for elem in result:
                print(elem)
        elif isinstance(result, dict):
            for k, v in result.items():
                print(f"{k} = {v}")
        elif result:
            print(result)

        return result
    return wrapper
```

“sort.py”:

```
def sortLambda(data):
    return [sorted(data, reverse=True, key=abs), sorted(data, reverse=True, key=lambda x: x if (x >= 0) else -x)]
```

“unique.py”:

```
class Unique(object):
    def __init__(self, items, **kwargs):
        self.elements = {}
        ignoreCase = kwargs.get("ignoreCase")

        if ignoreCase == None:
            ignoreCase = False
        elif not isinstance(ignoreCase, bool):
            raise Exception(
                "\u001b[33mignoreCase\u001b[0m parameter is not of type \u001b[31mbool\u001b[0m")

        for item in items:
            if ignoreCase == True and isinstance(item, str):
                item = item.lower()

            self.elements[item] = item

    def __next__(self):
        return next(self.elements)

    def __iter__(self):
        return iter(self.elements)

    def printUnique(self):
        print("[", end="")
        print(*self.elements, sep=", ", end="")
        print("]")
```

“processData.py”:

```
import json
import sys
```

```
from lab_python_fp.cmTimer1 import cmTimer1
from lab_python_fp.printResult import printResult
from lab_python_fp.unique import Unique
from lab_python_fp.genRandom import genGenerator


if len(sys.argv) > 1:
    path = sys.argv[1]
else:
    path = '/Users/arsenvardumyan/bkitHM/hm3/data_row.json'


with open(path) as f:
    data = json.load(f)


@printResult
def f1(data):
    return sorted(Unique((data[x]["job-name"] for x, _ in enumerate(data)), ignoreCase=True), key=lambda x: x.lower())


@printResult
def f2(data):
    return list(filter(lambda x: x.startswith('программист'), data))


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


@printResult
def f4(data):
    salaris = genGenerator(len(data), 100000, 200000)
    return list(zip(data, salaris))


if __name__ == '__main__':
    with cmTimer1():
        f4(f3(f2(f1(data))))
```

“main.py”:

```
from time import sleep
from lab_python_fp.field import fieldGenerator
from lab_python_fp.genRandom import genGenerator
from lab_python_fp.unique import Unique
from lab_python_fp.sort import sortLambda
from lab_python_fp.printResult import printResult
from lab_python_fp.cmTimer1 import cmTimer1
from lab_python_fp.cmTimer2 import cmTimer2


@printResult
```

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

    return fieldGenerator(goods, 'title', 'color')


@printResult
def genRandomTest():
    test1 = genGenerator(5, 3, 10)
    test2 = genGenerator(10, 0, 100)
    test3 = genGenerator(2, -10, 10)

    return {"test1": [*test1], "test2": [*test2], "test3": [*test3]}


@printResult
def uniqueTest():
    data1 = [1, 1, 1, 1, 4, 1, 2, 2, 2, 2, 2, 3]
    data2 = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']

    try:
        unique = Unique(data1, ignoreCase=True)
        unique.printUnique()
    except Exception as e:
        print(e)

    try:
        unique = Unique(data2, ignoreCase=True)
        unique.printUnique()
    except Exception as e:
        print(e)

    try:
        unique = Unique(data2)
        unique.printUnique()
    except Exception as e:
        print(e)

    try:
        unique = Unique(data2, ignoreCase=43)
        unique.printUnique()
    except Exception as e:
        print(e)

    try:
        unique = Unique(genGenerator(5, 3, 10))
        unique.printUnique()
    except Exception as e:
        print(e)
```

```python
@printResult
def sortTest():
    data = [4, -30, 100, -100, 123, 1, 0, -1, -4]
    return sortLambda(data)


@printResult
def timerTest():
    with cmTimer1():
        print("Some actions that should take 1.3 sec...")
        sleep(1.3)

    with cmTimer2():
        print("Some actions that should take 1.83 sec...")
        sleep(1.83)


def main():
    fieldGeneratorTest()
    genRandomTest()
    uniqueTest()
    sortTest()
    timerTest()


if __name__ == "__main__":
    main()
```

Экранные формы с примерами выполнения программы:

```
arsenvardumyan@MacBook-Pro-Arsen hm3 % /usr/local/bin/python3 /Users/arsenvardumyan/bkitHM/hm3/main.py

fieldGeneratorTest

{'title': 'Ковер', 'color': 'green'}
{'title': 'Диван для отдыха', 'color': 'black'}

genRandomTest

test1 = [8, 4, 8, 9, 10]
test2 = [25, 36, 29, 13, 77, 77, 39, 69, 58, 85]
test3 = [5, 9]

uniqueTest

[1, 4, 2, 3]
[a, b]
[a, A, b, B]
ignoreCase parameter is not of type bool
[5, 10, 4, 6]

sortTest

[123, 100, -100, -30, 4, -4, 1, -1, 0]
[123, 100, -100, -30, 4, -4, 1, -1, 0]

timerTest

Some actions that should take 1.3 sec...

Elapsed time: 1.302810 seconds

Some actions that should take 1.83 sec...

Elapsed time: 1.831801 seconds
```