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

Лабораторная работа №3

по дисциплине «Базовые компоненты интернет-технологий»

Выполнил:
студент группы ИУ5-32Б
Панов Г.Д.

Проверил:
Канев А.И.

2021 г.

**Общее 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**

Описание задачи

Необходимо реализовать генератор field. Генератор field последовательно выдает значения ключей словаря.

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.

Текст программы

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

    for item in items:
        dict = {arg: item.get(arg) for arg in args if item.get(arg)}

        if len(dict) == 0: continue

        if len(args) == 1: yield dict[args[0]]
        else: yield dict
```

```
goods = [
    {'title': 'Ковер', 'price': 2000, 'color': 'green'},
    {'title': 'Диван для отдыха', 'color': 'black'},
    {'title': 'Стул', 'color': 'black', 'price': None}
]

print(list(field(goods, 'title')))
print(list(field(goods, 'title', 'price')))
```

Экранные формы с примерами выполнения программы

```
['Ковер', 'Диван для отдыха', 'Стул']
[{'title': 'Ковер', 'price': 2000}, {'title': 'Диван для отдыха'}, {'title': 'Стул'}]
```

**Задача №2**

Описание задачи

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.

Текст программы

```
from random import randint

def gen_random(num_count, begin, end):
    return (randint(begin, end) for _ in range(num_count))


print(list(gen_random(5, 1, 3)))
```

Экранные формы с примерами выполнения программы

```
[3, 2, 3, 1, 3]
```

**Задача №3**

Описание задачи

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.

Текст программы

```
from gen_random import gen_random

class Unique(object):

    def __init__(self, items, **kwargs):

        self.items = items

        if kwargs:
            seen, result = set(), []
            for item in self.items:
                if type(item) == str:
                    if str(item.lower()) not in seen:
                        seen.add(item.lower())
                        result.append(item)
                else:
                    if item not in seen:
                        seen.add(item)
                        result.append(item)
        else:
            result = list(set(self.items))

        self.items = result
```

```
    def __next__(self):
        self.items += 1
        return (self.items)

    def __iter__(self):
        return (el for el in self.items)


data = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]
[print(*((el) for el in Unique(data)), sep = ", ")]

data = gen_random(5, 1, 3)
[print(*((el) for el in Unique(data)), sep = ", ")]

data = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']
[print(*((el) for el in Unique(data)), sep = ", ")]
[print(*((el) for el in Unique(data, ignore_case = True)), sep = ", ")]
```

Экранные формы с примерами выполнения программы

```
1, 2
1, 2, 3
B, a, A, b
a, b
```

**Задача №4**

Описание задачи
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.

Необходимо решить задачу двумя способами:

1) С использованием lambda-функции.
2) Без использования lambda-функции.

Текст программы

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

if __name__ == '__main__':
    result = sorted(data, reverse=True, key=abs)
    print(result)

    result_with_lambda = sorted(data, reverse=True, key=lambda el: abs(el))
    print(result_with_lambda)
```

Экранные формы с примерами выполнения программы

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

**Задача №5**

Описание задачи

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.

Текст программы

```
def print_result(func_to_decorate):
    def wrapper(*args, **kwargs):
        #print(func_to_decorate.__name__)
        res = func_to_decorate(*args, **kwargs)
        if type(res) == list:
            print(*res, sep = "\n")
        elif type(res) == dict:
            for k, v in res.items():
                print(k, '=', v)
        else:
            print(res)

    return wrapper


@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()
```

Экранные формы с примерами выполнения программы

```
!!!!!!!!
1
iu5
a = 1
b = 2
1
2
```

**Задача №6**

Описание задачи

Необходимо написать контекстные менеджеры cm_timer_1 и cm_timer_2, которые считают время работы блока кода и выводят его на экран.

Текст программы

```
from time import sleep, time
from contextlib import contextmanager


class cm_timer_1:
    def __init__(self):
        self.start = 0
```

```
        self.stop = 0
        self.res = 0

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

    def __exit__(self, exc_type, exc_val, exc_tb):
        self.stop = time()
        self.res = self.stop - self.start

        print(f"time: {self.res:0.1f} ")


@contextmanager
def cm_timer_2(*args, **kwds):
    start = time()

    yield

    print(f"time: {time() - start:0.1f} ")

with cm_timer_1():
    sleep(1.5)

with cm_timer_2():
    sleep(2.0)
```

Экранные формы с примерами выполнения программы

```
time: 1.5
time: 2.0
```

**Задача №7**

Описание задачи
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.

Текст программы

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


path = 'data_light.json'

with open(path, encoding='utf8') as f:
    data = json.load(f)


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


@print_result
def f2(arg):
```

```
    return list(filter(lambda job: job.startswith('Программист'), arg))


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


@print_result
def f4(arg):
    arg = list(arg)
    salary = gen_random(len(arg), 100000, 200000)

    return [job + f', зарплата {salary} руб' for salary, job in zip(salary,
arg)]


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

Экранные формы с примерами выполнения программы

```
1С программист
2-ой механик
3-ий механик
4-ый механик
4-ый электромеханик
ASIC специалист
JavaScript разработчик
RTL специалист
Web-программист
[химик-эксперт
web-разработчик
Автожестянщик
Автоинструктор
Автомаляр
Автомойщик
Автор студенческих работ по различным дисциплинам
Автослесарь - моторист
Автоэлектрик
Агент
Агент банка
Агент нпф
Агент по гос. закупкам недвижимости
Агент по недвижимости
```

```
Агент по недвижимости
Агент по недвижимости (стажер)
Агент по недвижимости / Риэлтор
Агент по привлечению юридических лиц
Агент по продажам (интернет, ТВ, телефония) в ПАО Ростелеком в населенных пунктах Амурской области: г. Благовещенск, г. Белогорск, г.
 Свободный, г. Шимановск, г. Зея, г. Тында
Агент торговый
Агроном-полевод
Администратор
Администратор (удаленно)
Администратор Active Directory
Администратор в парикмахерский салон
Администратор зала (предприятий общественного питания)
Администратор кофейни
Администратор на ресепшен
Администратор на телефоне
Администратор по информационной безопасности
Администратор ресторана
Администратор сайта
Администратор ярмарок выходного дня
Администратор-кассир
```

```
Акушерка женской консультации
Акушерка, АО
Акушерка, ВП
Альпинист промышленный
Аналитик
Анестезиолог - реаниматолог
Аппаратчик обработки зерна 5 разряда
Аппаратчик пастеризации
Аппаратчик установки опытного производства
Аппаратчик химводоочистки
Арматурщик
Артист (кукловод) театра кукол
Артист оркестра
Артист отдела социально - культурной деятельности Районного ЦНК
Артист хора
Артист-вокалист (солист)
Архивариус (Орехово-Зуевский филиал)
Архитектор, картограф, инженер-проектировщик
Ассистент главы отделения
Ассистент отдела продаж
Ассистент режиссера
БУХГАЛТЕР-Делопроизводитель
Бармен
Бармен-кассир в кафе
```

И т.д.

```
Программист
Программист / Senior Developer
Программист 1С
Программист C#
Программист C++
Программист C++/C#/Java
Программист/ Junior Developer
Программист/ технический специалист
Программистр-разработчик  информационных систем
time: 0.01591
```

```
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
time: 0.01563
```

```
Программист с опытом Python, зарплата 104794 руб
Программист / Senior Developer с опытом Python, зарплата 189086 руб
Программист 1С с опытом Python, зарплата 152029 руб
Программист C# с опытом Python, зарплата 137124 руб
Программист C++ с опытом Python, зарплата 119242 руб
Программист C++/C#/Java с опытом Python, зарплата 128151 руб
Программист/ Junior Developer с опытом Python, зарплата 131197 руб
Программист/ технический специалист с опытом Python, зарплата 100742 руб
Программистр-разработчик  информационных систем с опытом Python, зарплата 199034 руб
time: 0.01563
```