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 лабораторным работам №3-4
по дисциплине Базовые компоненты интернет технологии

Тема работы: "Функциональные возможности языка Python"

Выполнил:
Студент группы ИУ5Ц-54Б
Перевощиков Н.Д. 23.11.22 г.
(дата, подпись)

Проверил:
Преподаватель
Канев А.И. _____
(дата, подпись)

Москва, 2022

СОДЕРЖАНИЕ ОТЧЕТА

1. **Цель лабораторной работы**
   Изучение возможностей функционального программирования в языке Python.

2. **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 (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))))
```

3. Описание каждой задачи

3.1. Основная программа

LAB_03_04.py

```
import sys
from lab_python_fp.filed import *
from lab_python_fp.gen_random import *
from lab_python_fp.unique import *
from lab_python_fp.sort import *
from lab_python_fp.print_result import *
from lab_python_fp.cm_timer import *
# from lab_python_fp.process_data import *

def get_argv(index, prompt):
    try:
        # Получение значения из командной строки
        coef_str = sys.argv[index]
        print(index + 1, prompt, coef_str)
    except:
        # Вводим с клавиатуры
        print(index + 1, prompt, end='')
        coef_str = input()
    # По умолчанию строки
    return coef_str

def get_argv_value(index, prompt):
    try:
        # Получение значения из командной строки
        coef_str = sys.argv[index]
    except:
        # Вводим с клавиатуры
        print(prompt, end='')
        coef_str = input()
    # Переводим строку в целое число
    coef = int(coef_str)
    return coef

def into_tuple_from_str_in_value(str):
    tuple_buff = []
    str_buff = ''
    for i in range(len(str)):
        if (str[i] == ' '):
            tuple_buff.append(int(str_buff))
            str_buff = ''
        else:
            str_buff += str[i]

    tuple_buff.append(int(str_buff))

    return tuple_buff

def main():
    print('МЕНЮ')
    print('1. Задача 1 (файл field.py)')
    print('2. Задача 2 (файл gen_random.py)')
    print('3. Задача 3 (файл unique.py)')
    print('4. Задача 4 (файл sort.py)')
    print('5. Задача 5 (файл print_result.py)')
    print('6. Задача 6 (файл cm_timer.py)')
    # print('7. Задача 7 (файл process_data.py)')
    # Переключатель
    # switch = int(input('Введите номер пункта: '))
    switch = get_argv_value(1, 'Введите номер пункта: ')
```

```
    if(switch == 1):
        print('Задача 1 (файл field.py)')
        mas = ''
        if(len(sys.argv)==1):
            count_argv = int(input('Введите количество аргументов: '))
            if(count_argv > 1):
                for i in range(0, count_argv):
                    # print('{}-ый аргумент'.format(i + 1))
                    mas += (get_argv(i, '-ый аргумент: '))
                    # print('{}-ый аргумент: {}'.format(i + 1, mas[i]))
                    mas += ' '
                print(field(goods, mas))
            else:
                print('Ошибка введения количество аргументов!')

        elif (len(sys.argv) > 1):
            for i in range(0, len(sys.argv)):
                # print('{}-ый аргумент'.format(i + 1))
                mas += (get_argv(i, '-ый аргумент: '))
                # print('{}-ый аргумент: {}'.format(i + 1, mas[i]))
                mas += ' '
            print(field(goods, mas))
        else:
            print('Ошибка введения аргументов!')

    elif (switch == 2):
        print('Задача 2 (файл gen_random.py)')
        print('Генерация случайных чисел:')
        size = get_argv_value(2, 'Введите кол-во: ')
        if(size < 1):
            print('Ошибка! Разер больше 0 должен быть')
        else:
            numbers = gen_random(size, get_argv_value(3, 'Введите диапазон
от: '),
                                 get_argv_value(4, 'Введите диапазон до: '))
            print(numbers)

    elif (switch == 3):
        print('Задача 3 (файл unique.py)')
        data = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]
        print(data)
        mas1 = Unique(data)
        for i in Unique(mas1):
            print(i, end=' ')
        print()

        data_1 = gen_random(10, 1, 3)
        print(data_1)
        mas2 = Unique(data_1)
        for i in Unique(mas2):
            print(i, end=' ')
        print()

        data_2 = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']
        print(data_2)
        mas3 = Unique(data_2)
        for i in Unique(mas3):
            print(i, end=' ')
        print()

        mas4 = Unique(data_2, ignore_case=True)
        for i in Unique(mas4):
            print(i, end=' ')
```

```
        print()

        if (len(sys.argv) == 1):
            data_input = into_tuple_from_str(input('Введите любые значения в
списке (между значениями ставьте пробелом)\n'))

            print(data_input)
            mas3 = Unique(data_input)
            print('С чувствительным регистром')
            for i in Unique(mas3):
                print(i, end=' ')
            print()

            print(data_input)
            mas3 = Unique(data_input, ignore_case=True)
            print('Без чувствительного регистра')
            for i in Unique(mas3):
                print(i, end=' ')
            print()

        elif (len(sys.argv) > 1):
            buff = sys.argv[2]
            for i in range(2, len(sys.argv)):
                buff = buff + ' ' + sys.argv[i]

            data_3 = into_tuple_from_str(buff)

            print(data_3)
            c = Unique(data_3)
            print('С чувствительным регистром')
            for i in Unique(c):
                print(i, end=' ')
            print()

            print(data_3)
            c = Unique(data_3, ignore_case=True)
            print('Без чувствительного регистра')
            for i in Unique(c):
                print(i, end=' ')
            print()
        else:
            print('Ошибка введения аргументов!')

    elif (switch == 4):
        print('Задача 4 (файл sort.py)')
        sort()

        if (len(sys.argv) == 1):
            data_input = into_tuple_from_str_in_value(input('Введите любые
значения в списке (между значениями ставьте пробелом)\n'))

            print(f'Исходный список:\n {data_input}')

            result_with_lambda = sorted(data_input, key=lambda i: -abs(i))
            print(f'Отсортированный список с применением lambda-фнукции:\n
{result_with_lambda}')

            result = sorted(data_input, key=abs, reverse=True)
            print(f'Отсортированный список без применении lambda-функции:\n
{result}')

        elif (len(sys.argv) > 1):
            buff = sys.argv[2]
            for i in range(3, len(sys.argv)):
```

```
                buff = buff + ' ' + sys.argv[i]

            data_argv = into_tuple_from_str_in_value(buff)

            print(f'Исходный список:\n {data_argv}')

            result_with_lambda = sorted(data_argv, key=lambda i: -abs(i))
            print(f'Отсортированный список с применением lambda-фнукции:\n
{result_with_lambda}')

            result = sorted(data_argv, key=abs, reverse=True)
            print(f'Отсортированный список без применении lambda-функции:\n
{result}')
        else:
            print('Ошибка введения аргументов!')

    elif (switch == 5):
        print('Задача 5 (файл print_result.py)')
        fun_print_result()

    elif (switch == 6):
        print('Задача 6 (файл cm_timer.py)')
        cm_timer()

    # elif (switch == 7):
        # print('Задача 7 (файл process_data.py)')
        # print('Переходите в директорию lab_python_fp в файл
process_data.py, чтобы выполнить это задание')

    else:
        print('Ошибка! Нет такого пункта!')

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

Файлы, содержащие решения отдельных задач, должны располагаться в пакете lab_python_fp.

3.2.Задача №1 (файл filed.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.

3.2.1. Текст программы

```
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):
    try:
        # Преобразование в кортеж из строки
        argv = into_tuple_from_str(*args)
        # Необходимо реализовать генератор
        # len () возвращает длину (количество элементов) в объекте.
        assert len(argv) > 0, 'Ошибка! Нет аргументов! \nПримечание аргументы
не должны быть пустыми!'
        #range - диапазон, len - длина списка
        #items - это переменная, в которой на каждой итерации оказывается
элемент списка.
        r = [{} for i in range(len(items))]
        for i in range(len(items)):
            for j in items[i]:
                if j in argv:
                    # update - метод обновления словаря элементами из другого
объекта словаря.
                    r[i].update({j: items[i][j]})
        # возврат значения
        return r
    except:
        print('Ошибка! Нет списка в качестве переданного аргумента!')

# Преобразование в строку из кортежа
def into_tuple_from_str(str):
    cortes_buf = []
    str_buf = ''
    for i in range(len(str)):
        if (str[i] == ' '):
            cortes_buf.append(str_buf)
            str_buf = ''
        else:
            str_buf += str[i]
    # append - метод добавления элементов
    cortes_buf.append(str_buf)
    # возврат значения
    return cortes_buf
```

3.2.2. Результат Задача №1 (Pycharm)

```
МЕНЮ
1. Задача 1 (файл field.py)
2. Задача 2 (файл gen_random.py)
3. Задача 3 (файл unique.py)
4. Задача 4 (файл sort.py)
5. Задача 5 (файл print_result.py)
6. Задача 6 (файл cm_timer.py)
Введите номер пункта: 1
Задача 1 (файл field.py)
Введите количество аргументов: 3
1 -ый аргумент:  D:\Работа\МГТУ им. Н.Э.Баумана\Программирование\Программы\Программы за 5 семестр\LAB_03\LAB_03.py
2 -ый аргумент: title
3 -ый аргумент: color
[{'title': 'Ковер', 'color': 'green'}, {'title': 'Диван для отдыха', 'color': 'black'}]

Process finished with exit code 0
```

3.2.3. Результат Задача № 1 (Командная строка)

```
D:\Работа\МГТУ им. Н.Э.Баумана\Программирование\Программы\Программы за 5 семестр\LAB_03>python LAB_03.py
МЕНЮ
1. Задача 1 (файл field.py)
2. Задача 2 (файл gen_random.py)
3. Задача 3 (файл unique.py)
4. Задача 4 (файл sort.py)
5. Задача 5 (файл print_result.py)
6. Задача 6 (файл cm_timer.py)
Введите номер пункта: 1
Задача 1 (файл field.py)
Введите количество аргументов: 3
1 -ый аргумент:  LAB_03.py
2 -ый аргумент: title
3 -ый аргумент: color
[{'title': 'Ковер', 'color': 'green'}, {'title': 'Диван для отдыха', 'color': 'black'}]
```

3.3.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

3.3.1. Текст программы

```
4.  # 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

    import random
    # gen_random(количество, минимум, максимум)
    def gen_random(num_count, begin, end):
        mas = []
        # range - диапазон
        for i in range(0, num_count):
            # генерация случайных чисел
            mas.append(int(random.randint(begin, end)))
        # Возврат значения
        return mas
```

3.3.2. Результат Задача № 2 (Pycharm)

```
МЕНЮ
1. Задача 1 (файл field.py)
2. Задача 2 (файл gen_random.py)
3. Задача 3 (файл unique.py)
4. Задача 4 (файл sort.py)
5. Задача 5 (файл print_result.py)
6. Задача 6 (файл cm_timer.py)
Введите номер пункта: 2
Задача 2 (файл gen_random.py)
Генерация случайных чисел:
Введите кол-во: 5
Введите диапазон от: 1
Введите диапазон до: 3
[2, 1, 2, 2, 1]

Process finished with exit code 0
```

3.3.3. Результат Задача № 2 (Командная строка)

```
D:\Работа\МГТУ им. Н.Э.Баумана\Программирование\Программы\Программы за 5 семестр\LAB_03>python LAB_03.py

МЕНЮ
1. Задача 1 (файл field.py)
2. Задача 2 (файл gen_random.py)
3. Задача 3 (файл unique.py)
4. Задача 4 (файл sort.py)
5. Задача 5 (файл print_result.py)
6. Задача 6 (файл cm_timer.py)
Введите номер пункта: 2
Задача 2 (файл gen_random.py)
Генерация случайных чисел:
Введите кол-во: 5
Введите диапазон от: 1
Введите диапазон до: 3
[3, 3, 2, 1, 1]
```

3.4. Задача №3 (файл unigue.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.

3.4.1. Текст программы

```
# 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

        #"self" - это объект экземпляра, автоматически передаваемый методу
экземпляра класса при вызове,
        # чтобы идентифицировать экземпляр, который его вызвал.
        self.mas = []

        # Используя кортежи, получаем ключ и значения
        for key, value in kwargs.items():
            # Если ключ пустой и значение TRUE, то
            if key == 'ignore_case' and value == True:
                # Методы lower () возвращают строку в нижнем регистре из
заданной строки.
                # Он преобразует все заглавные символы в строчные.
                items = [i.lower() for i in items]

        for index in items:
            # Если текущее значение из списка item не совпадает/не существует
в созданном списке mas
            if index not in self.mas:
                # То присвоем несуществующее значение в созданном списке mas
                self.mas.append(index)
        pass

    # Для перехода к следующему элементу используется метод __next__.
    def __next__(self):
        try:
            x = self.mas[self.begin]
            self.begin += 1
            return x
        except:
            # Оператор raise позволяет принудительно породить исключение.
(Завершение работы итератора)
            raise StopIteration
```

```
    # __iter__(self) метод, который возвращает объект итератора;
    def __iter__(self):
        self.begin = 0
        return self
```

3.4.2. Результат Задача № 3 (Pycharm)

```
МЕНЮ
1. Задача 1 (файл field.py)
2. Задача 2 (файл gen_random.py)
3. Задача 3 (файл unique.py)
4. Задача 4 (файл sort.py)
5. Задача 5 (файл print_result.py)
6. Задача 6 (файл cm_timer.py)
Введите номер пункта: 3
Задача 3 (файл unique.py)
[1, 1, 1, 1, 1, 2, 2, 2, 2, 2]
1 2
[3, 3, 3, 1, 1, 2, 1, 2, 1, 2]
3 1 2
['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']
a A b B
a b
Введите любые значения в списке (между значениями ставьте пробелом)
2 4 3 6 1 4 2 4 1 3 1 4 2 5 3
['2', '4', '3', '6', '1', '4', '2', '4', '1', '3', '1', '4', '2', '5', '3']
С чувствительным регистром
2 4 3 6 1 5
['2', '4', '3', '6', '1', '4', '2', '4', '1', '3', '1', '4', '2', '5', '3']
Без чувствительного регистра
2 4 3 6 1 5


Process finished with exit code 0
```

3.4.3. Результат Задача № 3 (Командная строка)

```
D:\Работа\МГТУ им. Н.Э.Баумана\Программирование\Программы\Программы за 5 семестр\LAB_03>python LAB_03.py

МЕНЮ
1. Задача 1 (файл field.py)
2. Задача 2 (файл gen_random.py)
3. Задача 3 (файл unique.py)
4. Задача 4 (файл sort.py)
5. Задача 5 (файл print_result.py)
6. Задача 6 (файл cm_timer.py)
Введите номер пункта: 3
Задача 3 (файл unique.py)
[1, 1, 1, 1, 1, 2, 2, 2, 2, 2]
1 2
[2, 1, 2, 3, 3, 3, 2, 1, 1, 2]
2 1 3
['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']
a A b B
a b
Введите любые значения в списке (между значениями ставьте пробелом)
2 4 3 6 1 4 2 4 1 3 1 4 2 5 3
['2', '4', '3', '6', '1', '4', '2', '4', '1', '3', '1', '4', '2', '5', '3']
С чувствительным регистром
2 4 3 6 1 5
['2', '4', '3', '6', '1', '4', '2', '4', '1', '3', '1', '4', '2', '5', '3']
Без чувствительного регистра
2 4 3 6 1 5
```

3.5. 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.

3.5.1. Текст программы

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

def sort():
    print(f'Исходный список:\n {data}')
    result_with_lambda = sorted(data, key=lambda i: -abs(i))
    print(f'Отсортированный список с использованием lambda-функции:\n
{result_with_lambda}')
    # sorted - функция, который упорядочивает значения
    result = sorted(data, key=abs, reverse=True)
    print(f'Отсортированный список без использования lambda-функции:\n
{result}')
```

3.5.2. Результат Задача № 4 (Pycharm)

```
МЕНЮ
1. Задача 1 (файл field.py)
2. Задача 2 (файл gen_random.py)
3. Задача 3 (файл unique.py)
4. Задача 4 (файл sort.py)
5. Задача 5 (файл print_result.py)
6. Задача 6 (файл cm_timer.py)
Введите номер пункта: 4
Задача 4 (файл sort.py)
Исходный список:
 [4, -30, 30, 100, -100, 123, 1, 0, -1, -4]
Отсортированный список с использованием lambda-функции:
 [123, 100, -100, -30, 30, 4, -4, 1, -1, 0]
Отсортированный список без использования lambda-функции:
 [123, 100, -100, -30, 30, 4, -4, 1, -1, 0]
Введите любые значения в списке (между значениями ставьте пробелом)
234 -324 123 4659 1 -3 -2 765 324
Исходный список:
 [234, -324, 123, 4659, 1, -3, -2, 765, 324]
Отсортированный список с применением lambda-фнукции:
 [4659, 765, -324, 324, 234, 123, -3, -2, 1]
Отсортированный список без применении lambda-функции:
 [4659, 765, -324, 324, 234, 123, -3, -2, 1]

Process finished with exit code 0
```

3.5.3. Результат Задача №4 (Командная строка)

```
D:\Работа\МГТУ им. Н.Э.Баумана\Программирование\Программы\Программы за 5 семестр\LAB_03>python LAB_03.py

МЕНЮ
1. Задача 1 (файл field.py)
2. Задача 2 (файл gen_random.py)
3. Задача 3 (файл unique.py)
4. Задача 4 (файл sort.py)
5. Задача 5 (файл print_result.py)
6. Задача 6 (файл cm_timer.py)
Введите номер пункта: 4
Задача 4 (файл sort.py)
Исходный список:
 [4, -30, 30, 100, -100, 123, 1, 0, -1, -4]
Отсортированный список с использованием lambda-функции:
 [123, 100, -100, -30, 30, 4, -4, 1, -1, 0]
Отсортированный список без использования lambda-функции:
 [123, 100, -100, -30, 30, 4, -4, 1, -1, 0]
Введите любые значения в списке (между значениями ставьте пробелом)
234 -324 123 4659 1 -3 -2 765 324
Исходный список:
 [234, -324, 123, 4659, 1, -3, -2, 765, 324]
Отсортированный список с применением lambda-фнукции:
 [4659, 765, -324, 324, 234, 123, -3, -2, 1]
Отсортированный список без применении lambda-функции:
 [4659, 765, -324, 324, 234, 123, -3, -2, 1]
```

3.6. 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.

3.6.1. Текст программы

```
# Здесь должна быть реализация декоратора
'''
*args - неименованные аргументы (arguments), для которых важен порядок
передачи.
Также используется название "позиционные аргументы".
**kwargs - именованные аргументы (keywords), для которых не важен порядок
передачи.
Также используется название "аргументы, передаваемые по ключевым словам".
'''
# # Декоратор - это функция, ожидающая ДРУГУЮ функцию в качестве параметра.
# указываем имя декоратора и то, что он принимает function в качестве своей
переменной.
def print_result(function):
    #объявление функции-обёртки realization(). Тело обёртки описывает, что
именно мы будем делать с функцией, ранее принятой декоратором.
    def realization(mas=[], *args, **kwargs):
        # Печать название вызываемой функции
        print(function.__name__)
        if len(mas) == 0:
            result = function(*args, **kwargs)
        else:
            result = function(mas, *args, **kwargs)

        if type(result) == int or type(result) == str:
            print(result)
        # Если функция вернула список (list), то значения элементов списка
должны выводиться в столбик.
        elif type(result) is list:
            print('\n'.join(map(str, result)))
        # Если функция вернула словарь (dict), то ключи и значения должны
выводить в столбик через знак равенства.
        elif type(result) is dict:
            for key, meaning in result.items():
                print(f'{key} = {meaning}')
        # Функция zip создает итератор, который объединяет элементы из
нескольких источников данных.
        elif type(result) == zip:
            for name, chislo in result:
                print(name, chislo)
        else:
            print(result)
        #функция result возвращает переменную
        return result
    #декоратор возвращает нам уже саму функцию realization, точнее, результат
```

```
её работы над функцией function.
    return realization

# Синтаксис для обертывания функции в декоратор
@print_result
def test_1():
    return 1

# Синтаксис для обертывания функции в декоратор
@print_result
def test_2():
    return 'iu5'

# Синтаксис для обертывания функции в декоратор
@print_result
def test_3():
    return {'a': 1, 'b': 2}

# Синтаксис для обертывания функции в декоратор
@print_result
def test_4():
    return [1, 2]

# Синтаксис для обертывания функции в декоратор
def fun_print_result():
    test_1()
    test_2()
    test_3()
    test_4()
```

3.6.2. Результат Задача № 5 (Pycharm)

```
МЕНЮ
1. Задача 1 (файл field.py)
2. Задача 2 (файл gen_random.py)
3. Задача 3 (файл unique.py)
4. Задача 4 (файл sort.py)
5. Задача 5 (файл print_result.py)
6. Задача 6 (файл cm_timer.py)
Введите номер пункта: 5
Задача 5 (файл print_result.py)
test_1
1
test_2
iu5
test_3
a = 1
b = 2
test_4
1
2

Process finished with exit code 0
```

3.6.3. Результат Задача № 5 (Командная строка)

```
D:\Работа\МГТУ им. Н.Э.Баумана\Программирование\Программы\Программы за 5 семестр\LAB_03>python LAB_03.py

МЕНЮ
1. Задача 1 (файл field.py)
2. Задача 2 (файл gen_random.py)
3. Задача 3 (файл unique.py)
4. Задача 4 (файл sort.py)
5. Задача 5 (файл print_result.py)
6. Задача 6 (файл cm_timer.py)
Введите номер пункта: 5
Задача 5 (файл print_result.py)
test_1
1
test_2
iu5
test_3
a = 1
b = 2
test_4
1
2
```

3.7. 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).

3.7.1. Текст программы

```python
#time - время, sleep - задержка
from time import time, sleep
from contextlib import contextmanager

# На основе класса
class cm_timer_1:
    def __int__(self):
        self._start = 0
        self._end = 0

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

    def __exit__(self, the_type, the_value, the_backing):
        self._end = time()
        print(f'Время работы блок кода: {self._end - self._start}')

# С использованием библиотеки contextlib
@contextmanager
def cm_timer_2():
    start_time = time()
    # Yield - ключевое слово, которое используется вместо return. С его
помощью функция возвращает значение
```

```
    # без уничтожения переменных, кроме того, при каждом последующем вызове
функция начинает своё
    # выполнение с оператора yield.
    yield None
    end_time = time()
    print(f'Время работы блок кода: {end_time - start_time}')

def cm_timer():
    with cm_timer_1():
        sleep(5.5)

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

3.7.2. Результат Задача № 6 (Pycharm)

```
МЕНЮ
1. Задача 1 (файл field.py)
2. Задача 2 (файл gen_random.py)
3. Задача 3 (файл unique.py)
4. Задача 4 (файл sort.py)
5. Задача 5 (файл print_result.py)
6. Задача 6 (файл cm_timer.py)
Введите номер пункта: 6
Задача 6 (файл cm_timer.py)
Время работы блок кода: 5.511905670166016
Время работы блок кода: 5.514033794403076

Process finished with exit code 0
```

3.7.3. Результат Задача № 6 (Командная строка)

```
D:\Работа\МГТУ им. Н.Э.Баумана\Программирование\Программы\Программы за 5 семестр\LAB_03>python LAB_03.py

МЕНЮ
1. Задача 1 (файл field.py)
2. Задача 2 (файл gen_random.py)
3. Задача 3 (файл unique.py)
4. Задача 4 (файл sort.py)
5. Задача 5 (файл print_result.py)
6. Задача 6 (файл cm_timer.py)
Введите номер пункта: 6
Задача 6 (файл cm_timer.py)
Время работы блок кода: 5.51444935798645
Время работы блок кода: 5.5075085163116455
```

3.8. 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.

3.8.1. Текст программы

```
import json
# Сделаем другие необходимые импорты
from operator import concat
from filed import field
from unique import Unique
from sort import sort
from gen_random import gen_random
from cm_timer import cm_timer_1
from gen_random import gen_random
from print_result import print_result
# путь файла
path = '../data_light.json'

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

# Преобразуем в UTF-8 кодировку, иначе программа неправильно прочтет файл
with open(path, 'r', encoding='UTF-8') as f:
    data = json.load(f)
    # print(data)

# 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):
    # Подборка названий работы, которые не совпадают друг друга в списке
    # info_name_work = Unique([i['name-work'] for i in field(data, 'name-
work')], ignore_case=True)
    # Отсортируем
    # info_name_work_sorted = sorted(info_name_work, key=str, reverse =
False)
    # return info_name_work.mas.sort()
    # return sorted(info_job_name, key=str, reverse = False)
    # return sorted(list(set([el['name-work'] for meaning in arg])),
key=lambda a: a.lower())
    return list(Unique([i['job-name'] for i in field(data, 'job-name')],
ignore_case=True))

@print_result
def f2(arg):
    # list() - создает или преобразует переданный объект в список.
    # filter() - применяет другую функцию к заданному итерируемому объекту
(список, строка, словарь и так далее),
    # проверяя, нужно ли сохранить конкретный элемент или нет.
    # startswith() - начинается ли строка с определенного шаблона или нет.
    return list(filter(lambda i: i.startswith('программист'), arg))

@print_result
def f3(arg):
    # list() - создает или преобразует переданный объект в список.
    # map() — это функция, позволяющая обрабатывать и преобразовывать все
элементы в итерируемом объекте без цикла for.
    # concat - сложение строк.
    return list(map(lambda x: concat(x, ' с опытом Python'), arg))

@print_result
def f4(arg):
    # list() - создает или преобразует переданный объект в список.
    # zip - создает итератор, который объединяет элементы из нескольких
источников данных.
    return list(zip(arg, ['зарплата ' + str(meaning) + ' руб.' for meaning in
gen_random(len(arg), 100000, 200000)]))

if __name__ == '__main__':
    with cm_timer_1():
        # ex_1 = f1(data)
        # ex_2 = f2(f1(data))
        # ex_3 = f3(f2(f1(data)))
        ex_4 = (f4(f3(f2(f1(data)))))
        print()
```

3.8.2. Результат Задача №7 (Pycharm)

Для 7 пункта включаем: “Curret File”:
Начало результата 7 пункта:

```
f1
администратор на телефоне
медицинская сестра
охранник сутки-день-ночь-вахта
врач анестезиолог реаниматолог
теплотехник
разнорабочий
электро-газосварщик
водитель gett/гетт и yandex/яндекс такси на личном автомобиле
монолитные работы
организатор – тренер
помощник руководителя
автоэлектрик
врач ультразвуковой диагностики в детскую поликлинику
менеджер по продажам ит услуг (b2b)
менеджер по персоналу
аналитик
воспитатель группы продленного дня
инженер по качеству
инженер по качеству 2 категории (класса)
водитель автомобиля
пекарь
переводчик
терапевт
врач-анестезиолог-реаниматолог
инженер-конструктор в наружной рекламе
монтажник-сборщик рекламных конструкций
оператор фрезерно-гравировального станка
зоотехник
```

…

Конец результата программы 7 пункта:

```
программист/ junior developer
программист / senior developer
программист/ технический специалист
программист c#
f3
программист с опытом Python
программист c++/c#/java с опытом Python
программист 1с с опытом Python
программистр-разработчик  информационных систем с опытом Python
программист c++ с опытом Python
программист/ junior developer с опытом Python
программист / senior developer с опытом Python
программист/ технический специалист с опытом Python
программист c# с опытом Python
f4
('программист с опытом Python', 'зарплата 138843 руб.')
('программист c++/c#/java с опытом Python', 'зарплата 160508 руб.')
('программист 1с с опытом Python', 'зарплата 159151 руб.')
('программистр-разработчик  информационных систем с опытом Python', 'зарплата 103743 руб.')
('программист c++ с опытом Python', 'зарплата 182059 руб.')
('программист/ junior developer с опытом Python', 'зарплата 106853 руб.')
('программист / senior developer с опытом Python', 'зарплата 149083 руб.')
('программист/ технический специалист с опытом Python', 'зарплата 154943 руб.')
('программист c# с опытом Python', 'зарплата 180562 руб.')

Время работы блок кода: 0.06202220916748047

Process finished with exit code 0
```

3.8.3. Результат Задача №7 (Командная строка)

```
D:\Работа\МГТУ им. Н.Э.Баумана\Программирование\Программы\Программы за 5 семестр\LAB_03\lab_python_fp>python process_data.py
f1
администратор на телефоне
медицинская сестра
охранник сутки-день-ночь-вахта
врач анестезиолог реаниматолог
теплотехник
разнорабочий
электро-газосварщик
водитель gett/гетт и yandex/яндекс такси на личном автомобиле
монолитные работы
организатор – тренер
помощник руководителя
автоэлектрик
врач ультразвуковой диагностики в детскую поликлинику
менеджер по продажам ит услуг (b2b)
менеджер по персоналу
аналитик
воспитатель группы продленного дня
инженер по качеству
инженер по качеству 2 категории (класса)
водитель автомобиля
пекарь
переводчик
терапевт
врач-анестезиолог-реаниматолог
инженер-конструктор в наружной рекламе
монтажник-сборщик рекламных конструкций
оператор фрезерно-гравировального станка
зоотехник
сварщик
рабочий-строитель
врач-трансфузиолог
юрисконсульт
специалист отдела автоматизации
растворщик реагентов
бармен
официант
технолог
фельдшер-лаборант
медицинская сестра по физиотерапии
врач функциональной диагностики
рентгенолаборант
диспетчер по навигации
водитель погрузчика,штабелер
машинист автогрейдера
наладчик чпу
упаковщик-грузчик
```

...

```
бетонщик - арматурщик
главный инженер финансово-экономического отдела
секретарь судебного заседания в аппарате мирового судьи железнодорожного судебного района города ростова
-на-дону
варщик зефира
варщик мармеладных изделий
оператор склада
специалист по электромеханическим испытаниям аппаратуры бортовых космических систем
заведующий музеем в д.копорье
документовед
специалист по испытаниям на электромагнитную совместимость аппаратуры бортовых космических систем
менеджер (в промышленности)
f2
программист
программист c++/c#/java
программист 1c
программистр-разработчик  информационных систем
программист c++
программист/ junior developer
программист / senior developer
программист/ технический специалист
программист c#
f3
программист с опытом Python
программист c++/c#/java с опытом Python
программист 1c с опытом Python
программистр-разработчик  информационных систем с опытом Python
программист c++ с опытом Python
программист/ junior developer с опытом Python
программист / senior developer с опытом Python
программист/ технический специалист с опытом Python
программист c# с опытом Python
f4
('программист с опытом Python', 'зарплата 119167 руб.')
('программист c++/c#/java с опытом Python', 'зарплата 170007 руб.')
('программист 1c с опытом Python', 'зарплата 172006 руб.')
('программистр-разработчик  информационных систем с опытом Python', 'зарплата 155542 руб.')
('программист c++ с опытом Python', 'зарплата 115501 руб.')
('программист/ junior developer с опытом Python', 'зарплата 192820 руб.')
('программист / senior developer с опытом Python', 'зарплата 198772 руб.')
('программист/ технический специалист с опытом Python', 'зарплата 150797 руб.')
('программист c# с опытом Python', 'зарплата 119042 руб.')

Время работы блок кода: 0.06841611862182617
```