Московский государственный технический университет им. Н.Э. Баумана

Факультет “Радиотехнический”
Кафедра “Системы обработки информации и управления”

Курс «Парадигмы и конструкции языка»

Отчет по лабораторной работе №3-4 «Функциональные возможности языка Python.»

Выполнил:
студент группы РТ5-31Б:
Зеленева И.Е.

Проверил:
преподаватель каф. ИУ5
Гапанюк Ю.Е.

Москва, 2025 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'},
    {'title': 'Стол', 'price': None, 'color': 'white'},
    {'price': 5000}
]
```

field(goods, 'title') должен выдавать 'Ковер', 'Диван для отдыха'
field(goods, 'title', 'price') должен выдавать {'title': 'Ковер', 'price': 5000}, {'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.

Текст программы

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

    if len(args) == 1:
        # Если один аргумент - возвращаем только значения
        key = args[0]
        for item in items:
            if key in item and item[key] is not None:
                yield item[key]
    else:
        # Если несколько аргументов - возвращаем словари
        for item in items:
            result = {}
            has_valid_fields = False
            for key in args:
                if key in item and item[key] is not None:
```

```
                result[key] = item[key]
                has_valid_fields = True
        if has_valid_fields:
            yield result


if __name__ == '__main__':
    # Тестирование
    goods = [
        {'title': 'Ковер', 'price': 2000, 'color': 'green'},
        {'title': 'Диван для отдыха', 'color': 'black'},
        {'title': 'Стол', 'price': None, 'color': 'white'},
        {'price': 5000}
    ]

    print("Test 1 - один аргумент:")
    for item in field(goods, 'title'):
        print(item)

    print("\nTest 2 - несколько аргументов:")
    for item in field(goods, 'title', 'price'):
        print(item)
```

Результаты

```
C:\Users\User\Репозиторий\lab_34\lab_python_fp>python field.py
Test 1 - один аргумент:
Ковер
Диван для отдыха
Стол

Test 2 - несколько аргументов:
{'title': 'Ковер', 'price': 2000}
{'title': 'Диван для отдыха'}
{'title': 'Стол'}
{'price': 5000}
```

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:

Текст программы

```
import random
```

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

if __name__ == '__main__':
    # Тестирование
    print("Test gen_random:")
    for num in gen_random(5, 1, 3):
        print(num, end=' ')
    print()
```

Результаты

```
C:\Users\User\Репозиторий\lab_34\lab_python_fp>python gen_random.py
Test gen_random:
2 1 3 3 3
```

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:

Текст программы

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

```
    def __next__(self):
        while True:
            item = next(self.items)

            # Для проверки уникальности создаем ключ
            if isinstance(item, str) and self.ignore_case:
                check_item = item.lower()
            else:
                check_item = item

            if check_item not in self.seen:
                self.seen.add(check_item)
                return item

    def __iter__(self):
        return self


if __name__ == '__main__':
    # Тестирование
    print("Test 1 - числа:")
    data1 = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]
    for item in Unique(data1):
        print(item, end=' ')
    print()

    print("\nTest 2 - строки без ignore_case:")
    data2 = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']
    for item in Unique(data2):
        print(item, end=' ')
    print()

    print("\nTest 3 - строки с ignore_case=True:")
    for item in Unique(data2, ignore_case=True):
        print(item, end=' ')
    print()
```

Результаты

```
C:\Users\User\Репозиторий\lab_34\lab_python_fp>python unique.py
Test 1 - числа:
1 2

Test 2 - строки без ignore_case:
a A b B

Test 3 - строки с ignore_case=True:
a b
```

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.

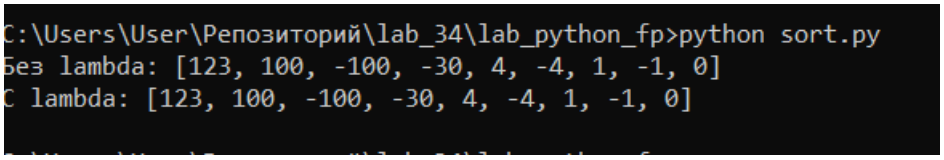
Шаблон реализации:

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

if __name__ == '__main__':
    result = ...
    print(result)

    result_with_lambda = ...
    print(result_with_lambda)
```

Текст программы

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

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

    # С lambda
    result_with_lambda = sorted(data, key=lambda x: abs(x), reverse=True)

    print("Без lambda:", result)
    print("С lambda:", result_with_lambda)
```

Результаты



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:

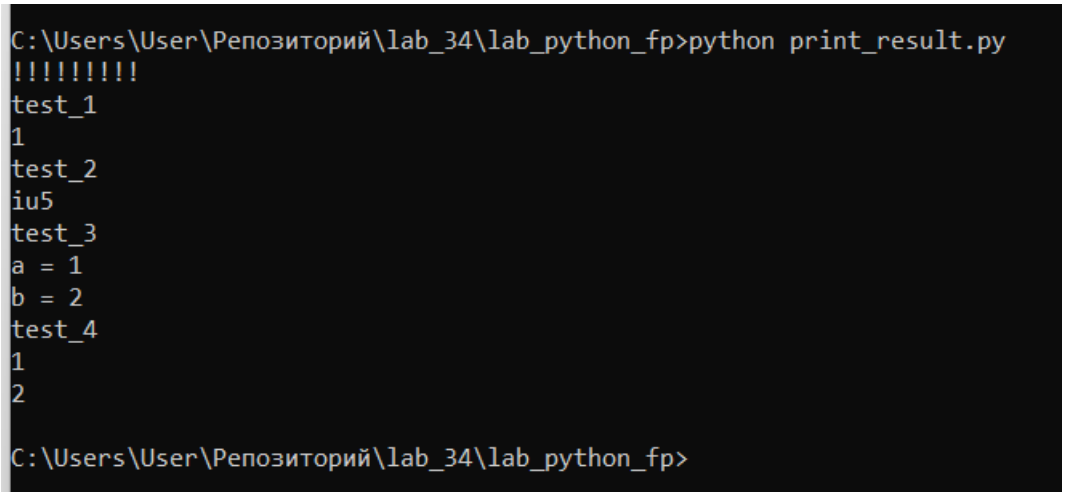
```
test_1
1
test_2
iu5
test_3
a = 1
b = 2
test_4
1
2
```

Текст программы

```
def print_result(func):
    def wrapper(*args, **kwargs):
        result = func(*args, **kwargs)
        print(func.__name__)

        if isinstance(result, list):
            for item in result:
                print(item)
        elif isinstance(result, dict):
            for key, value in result.items():
                print(f"{key} = {value}")
        else:
            print(result)

        return result

    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()
```

Результаты



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)
```

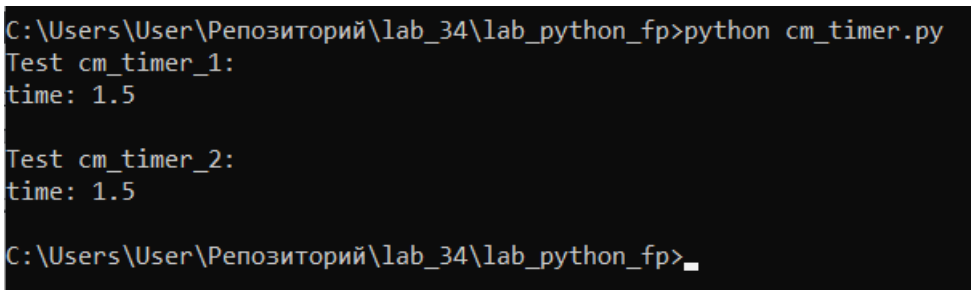
После завершения блока кода в консоль должно вывестись time: 5.5 (реальное время может несколько отличаться).
cm_timer_1 и cm_timer_2 реализуют одинаковую функциональность, но должны быть реализованы двумя различными способами (на основе класса и с использованием библиотеки contextlib).

Текст программы

Результаты

```
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:.1f}")


@contextmanager
def cm_timer_2():
    start_time = time.time()
    try:
        yield
    finally:
        elapsed_time = time.time() - start_time
        print(f"time: {elapsed_time:.1f}")


if __name__ == '__main__':
    # Тестирование
    print("Test cm_timer_1:")
    with cm_timer_1():
        time.sleep(1.5)

    print("\nTest cm_timer_2:")
    with cm_timer_2():
        time.sleep(1.5)
```



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.

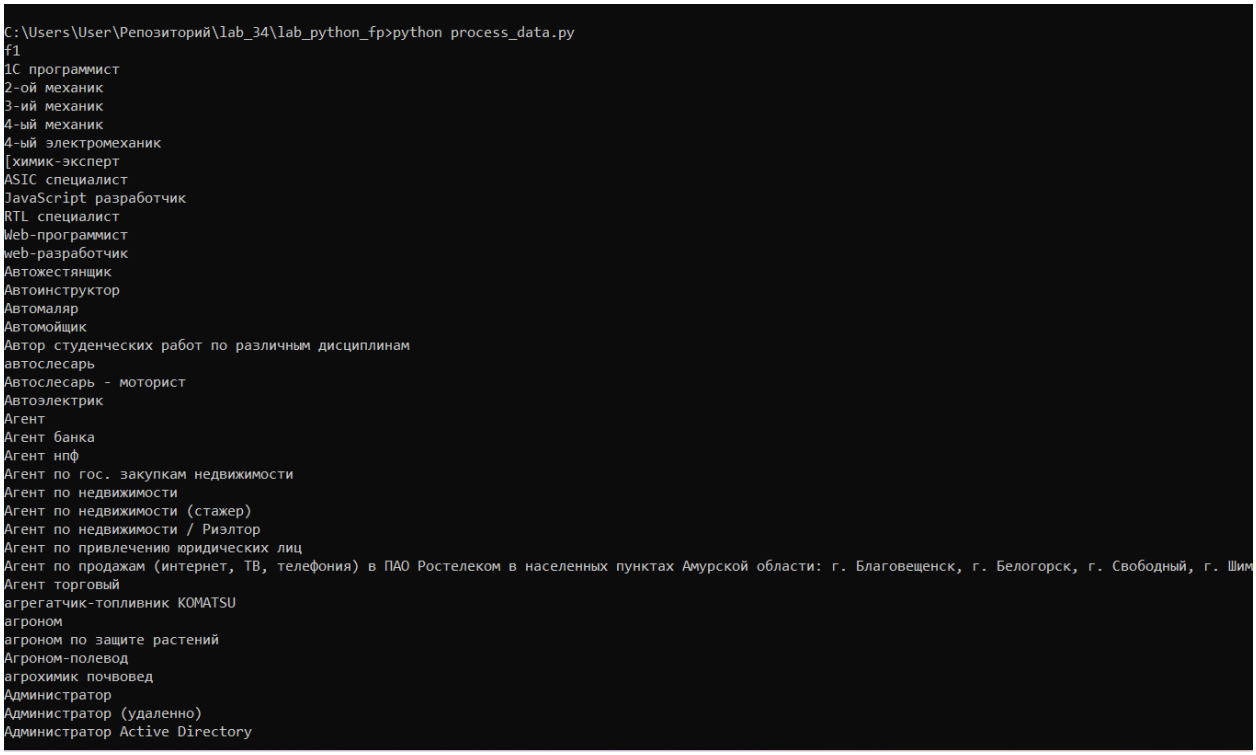
- Функция f4 должна сгенерировать для каждой специальности зарплату от 100 000 до 200 000 рублей и присоединить её к названию специальности. Пример: Программист C# с опытом Python, зарплата 137287 руб. Используйте zip для обработки пары специальность — зарплата.

Текст программы

```
import json
import sys

# Импорты из того же пакета
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

# Получаем путь к файлу из аргументов командной строки
path = sys.argv[1] if len(sys.argv) > 1 else 'data_light.json'

try:
    with open(path, encoding='utf-8') as f:
        data = json.load(f)
except FileNotFoundError:
    print(f"Файл {path} не найден!")
    data = []


@print_result
```

```
def f1(arg):
    return sorted(list(Unique(field(arg, 'job-name'), ignore_case=True)),
key=str.lower)


@print_result
def f2(arg):
    return list(filter(lambda x: x.lower().startswith('программист'), arg))


@print_result
def f3(arg):
    return list(map(lambda x: f"{x} с опытом Python", arg))


@print_result
def f4(arg):
    salaries = list(gen_random(len(arg), 100000, 200000))
    return list(map(lambda x: f"{x[0]}, зарплата {x[1]} руб.", zip(arg, salaries)))


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

Результаты



```
Электросварщик на автоматических и полуавтоматических машинах
Электросварщик ручной сварки
Электросварщики ручной сварки
Электрослесарь (слесарь) дежурный и по ремонту оборудования, старший
Электрослесарь по ремонту и обслуживанию автоматики и средств измерений электростанций
Электрослесарь по ремонту оборудования в карьере
Электроэрозионист
Эндокринолог
Энергетик
Энергетик литейного производства
энтомолог
Юрисконсульт
юрисконсульт 2 категории
Юрисконсульт. Контрактный управляющий
Юрист
Юрист (специалист по сопровождению международных договоров, английский - разговорный)
Юрист волонтер
Юристконсульт
f2
Программист
Программист / Senior Developer
Программист 1С
Программист C#
Программист C++
Программист C++/C#/Java
Программист/ Junior Developer
Программист/ технический специалист
Программистр-разработчик  информационных систем
f3
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
f4
Программист с опытом Python, зарплата 103375 руб.
Программист / Senior Developer с опытом Python, зарплата 160737 руб.
Программист 1С с опытом Python, зарплата 173627 руб.
Программист C# с опытом Python, зарплата 102027 руб.
Программист C++ с опытом Python, зарплата 102900 руб.
Программист C++/C#/Java с опытом Python, зарплата 115063 руб.
Программист/ Junior Developer с опытом Python, зарплата 170655 руб.
Программист/ технический специалист с опытом Python, зарплата 134199 руб.
Программистр-разработчик  информационных систем с опытом Python, зарплата 120332 руб.
time: 0.4

C:\Users\User\Репозиторий\lab_34\lab_python_fp>
```