

**Московский государственный технический
университет им. Н.Э. Баумана**

Факультет «Информатика и системы управления»
Кафедра ИУ5 «Системы обработки информации и управления»

Курс «ПИКяП»

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

Выполнил:

студент группы ИУ5-36Б

Илюхин Илья

Подпись и дата: 27.11.2024

Проверил:

преподаватель каф. ИУ5

Нардид А. Н.

Подпись и дата:

Москва 2024

Цель лабораторной работы: изучение возможностей функционального программирования в языке Python.

Задача 1 (файл field.py)

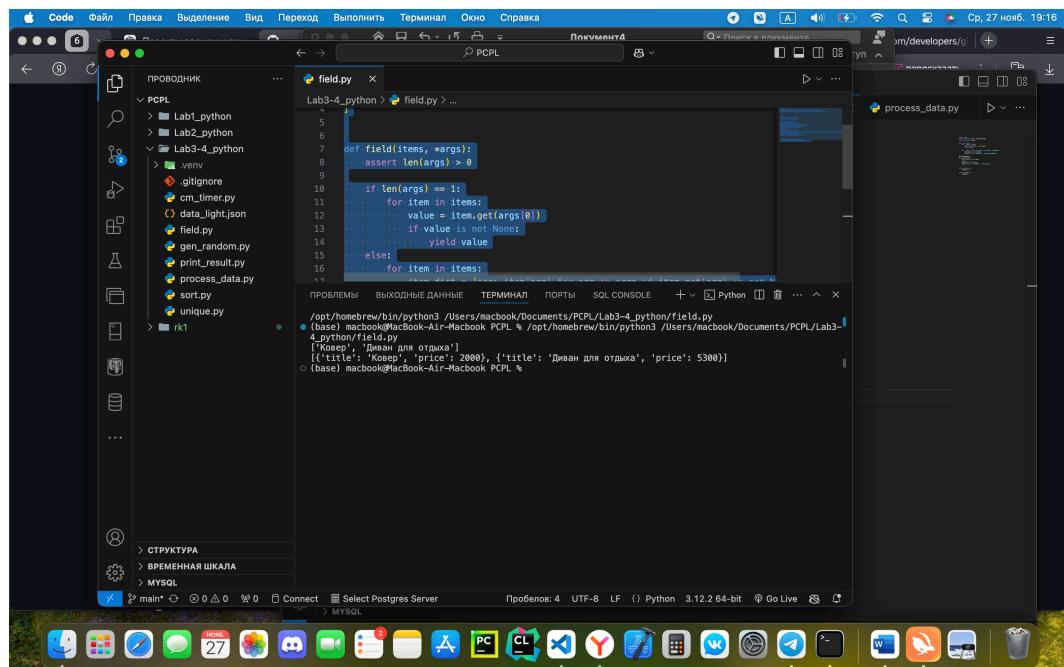
Необходимо реализовать генератор field. Генератор field последовательно выдает значения ключей словаря.

```
goods = [
    {'title': 'Ковер', 'price': 2000, 'color': 'green'},
    {'title': 'Диван для отдыха', 'price': 5300, 'color': 'black'}
]

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

    if len(args) == 1:
        for item in items:
            value = item.get(args[0])
            if value is not None:
                yield value
    else:
        for item in items:
            item_dict = {arg: item[arg] for arg in args if item.get(arg) is not None}
            if item_dict:
                yield item_dict

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

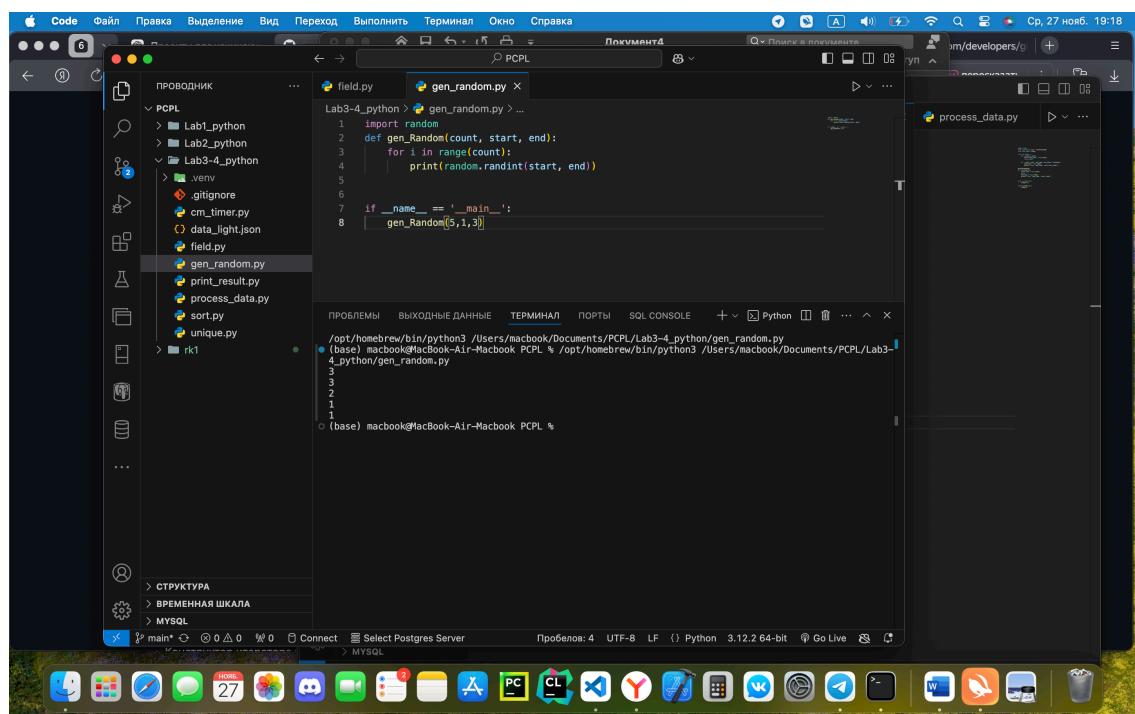


Задача 2 (файл gen_random.py)

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.

```
import random
def gen_Random(count, start, end):
    for i in range(count):
        print(random.randint(start, end))

if __name__ == '__main__':
    gen_Random(5,1,3)
```



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.

```

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

    def __next__(self):
        while True:
            try:
                item = next(self.items)
            except StopIteration:
                raise StopIteration

            if self.ignore_case:
                item = item.lower()

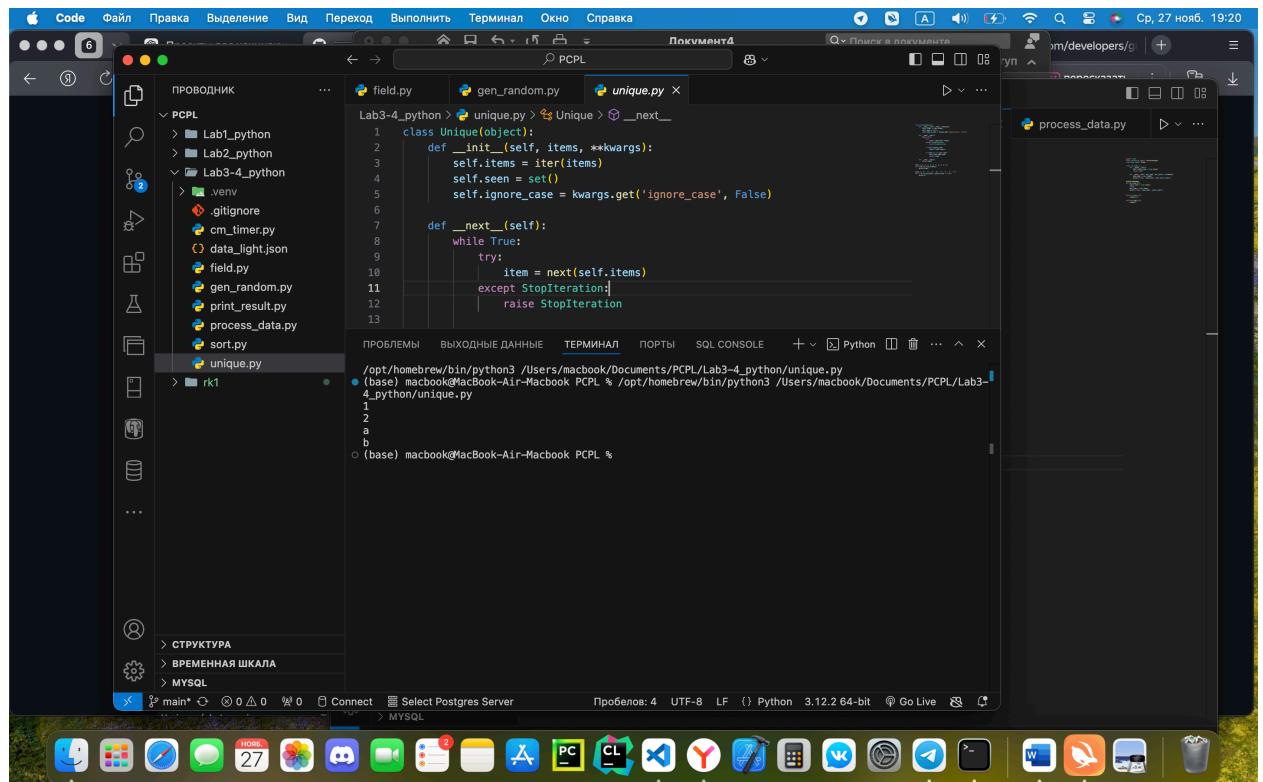
            if item not in self.seen:
                self.seen.add(item)
                return item

    def __iter__(self):
        return self

data = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]
for item in Unique(data):
    print(item)

data = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']
for i in Unique(data, ignore_case = True):
    print(i)

```



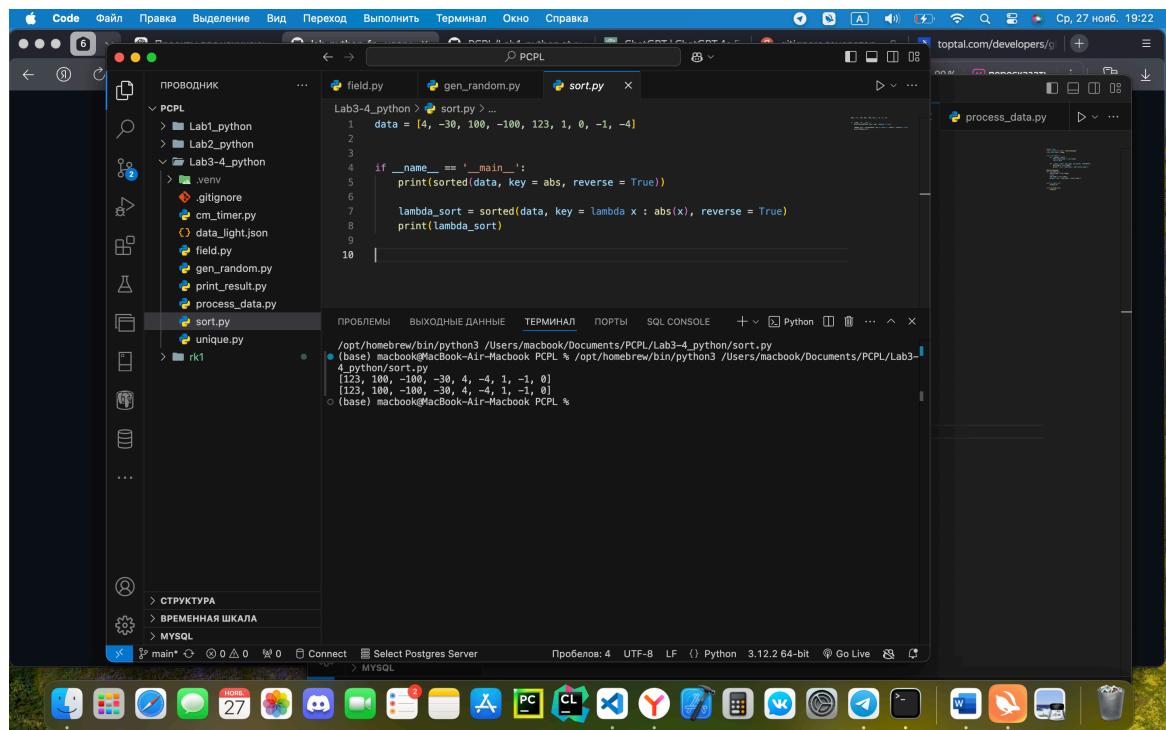
Задача 4 (файл sort.py)

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.

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

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

lambda_sort = sorted(data, key = lambda x : abs(x), reverse = True)
print(lambda_sort)
```



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.

```
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

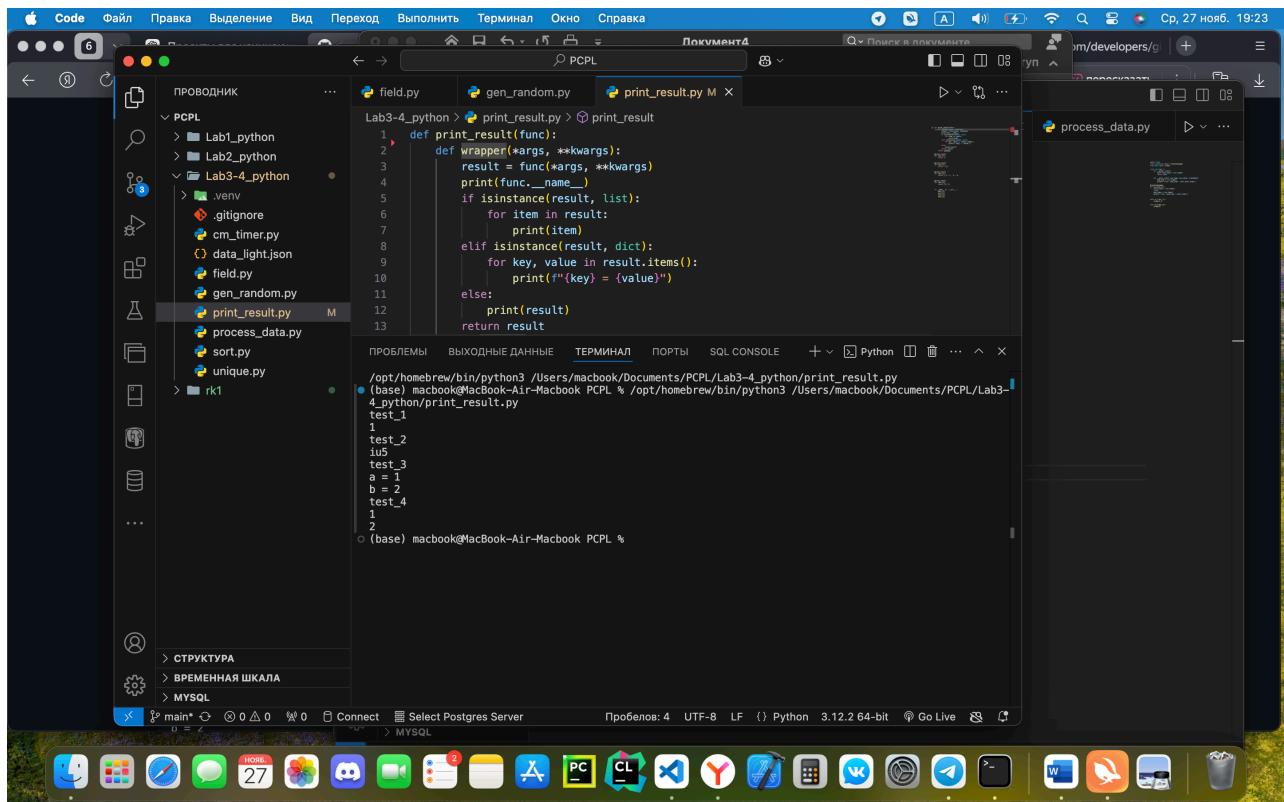
@print_result
def test_1():
    return 1

@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__':
    test_1()
    test_2()
    test_3()
    test_4()
```



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.

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

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

    def __exit__(self, exc_type, exc_value, traceback):
        end_time = time.time()
        print(f"time: {end_time - self.start_time}")

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

with cm_timer_1():
    sleep(5.5)

with cm_timer_2():
    sleep(3)
```

The screenshot shows the PyCharm IDE interface. The code editor displays a Python script named `cm_timer.py` with the following content:

```
1 import time
2 from contextlib import contextmanager
3 from time import sleep
4
5 class cm_timer_1:
6     def __enter__(self):
7         self.start_time = time.time()
8         return self
9
10    def __exit__(self, exc_type, exc_value, traceback):
11        end_time = time.time()
12        print(f"Time: {end_time - self.start_time}")
13
```

The terminal window shows the execution of the script:

```
(base) macbook@MacBook-Air-MacBook: /opt/homebrew/bin/python3 /Users/macbook/Documents/PCPL/Lab3-4_python/print_result.py
test_1
1
test_2
1
test_3
1
a = 1
b = 2
test_4
1
2
(base) macbook@MacBook-Air-MacBook: /opt/homebrew/bin/python3 /Users/macbook/Documents/PCPL/Lab3-4_python/cm_timer.py
Time: 5.5051047801971436
Time: 3.0038629986463623
(base) macbook@MacBook-Air-MacBook: /opt/homebrew/bin/python3 /Users/macbook/Documents/PCPL/Lab3-4_python/print_result.py
test_1
1
test_2
1
test_3
1
a = 1
b = 2
test_4
1
2
```

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.

```

import json
import sys
import random
from contextlib import contextmanager
import time

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

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

    def __exit__(self, exc_type, exc_val, exc_tb):
        end_time = time.time()
        print(f'time: {end_time - self.start_time:.2f} seconds')

path = sys.argv[1] if len(sys.argv) > 1 else "data_light.json"
with open(path, encoding="utf-8") as f:
    data = json.load(f)

@print_result
def f1(arg):
    return sorted(set(job['job-name'].strip().lower().capitalize() for job in arg))

@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 = [random.randint(100000, 200000) for _ in arg]
    return [f'{job}, зарплата {salary} руб.' for job, salary in zip(arg, salaries)]

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

```

