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:
Сахарова О.П

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'}
]
```

field(goods, 'title') должен выдавать 'Ковер', 'Диван для отдыха'
field(goods, 'title', 'price') должен выдавать {'title': 'Ковер', 'price': 2000}, {'title': 'Диван для отдыха'}

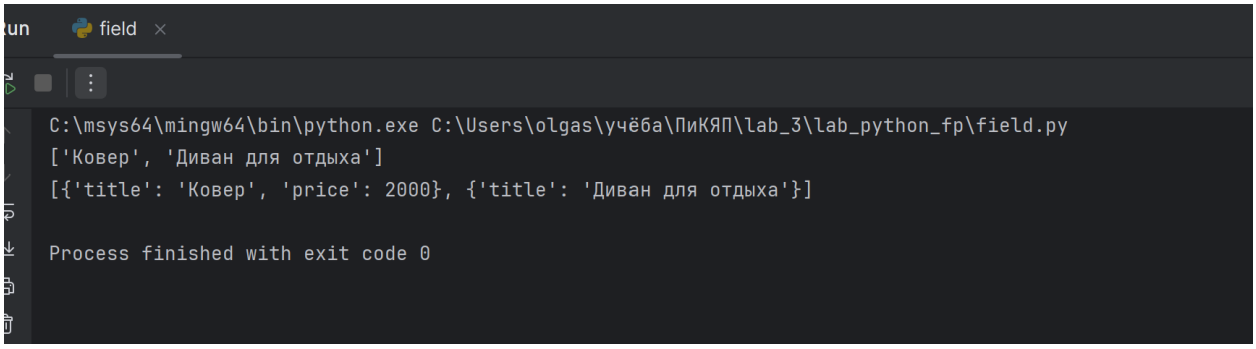
- В качестве первого аргумента генератор принимает список словарей, дальше через *args генератор принимает неограниченное количествово аргументов.
- Если передан один аргумент, генератор последовательно выдает только значения полей, если значение поля равно None, то элемент пропускается.
- Если передано 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.

Текст программы

```
def field(items, *args):
    assert len(args) > 0
    for item in items:
        if item is None:
            continue
        if len(args) == 1:
            if args[0] in item and item[args[0]] is not None:
                yield item[args[0]]
        else:
            d = {}
            flag = False
            for key in args:
                if key in item and item[key] is not None:
                    d[key] = item[key]
                    flag = True
            if flag:
                yield d

goods = [
```

```
    {'title': 'Ковер', 'price': 2000, 'color': 'green'},
    {'title': 'Диван для отдыха', 'color': 'black'}
]
print(list(field(goods, 'title')))

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

Результаты



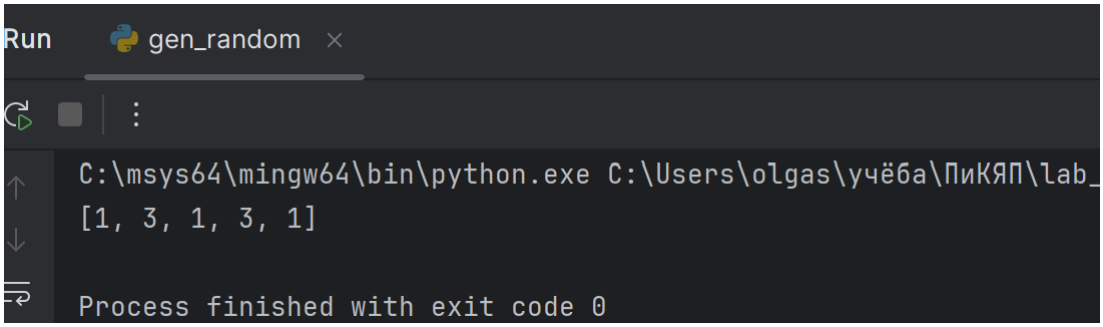
Задача 2 (файл gen_random.py)

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

```
from random import randint

def gen_random(num_count, begin, end):
    for i in range(num_count):
        yield randint(begin, end)

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

Результаты



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.

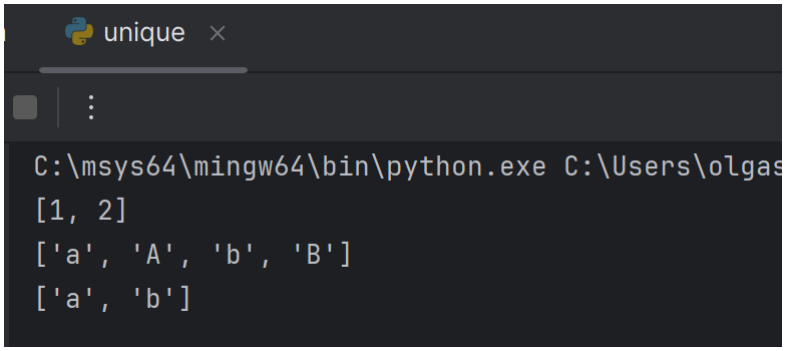
Пример:

```
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:
    def __init__(self, items, **kwargs):
        self.items = list(items)
        self.ignore_case = kwargs.get('ignore_case', False)
        self.unq = set()
        self.index = 0

    def __iter__(self):
        return self

    def __next__(self):
        while self.index < len(self.items):
            item = self.items[self.index]
            self.index += 1
            key = item.lower() if self.ignore_case and isinstance(item, str) else item
            if key not in self.unq:
                self.unq.add(key)
                return item
        raise StopIteration

data = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]
print(list(Unique(data)))

data = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']
print(list(Unique(data)))
print(list(Unique(data, ignore_case=True)))
```

Результаты



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.

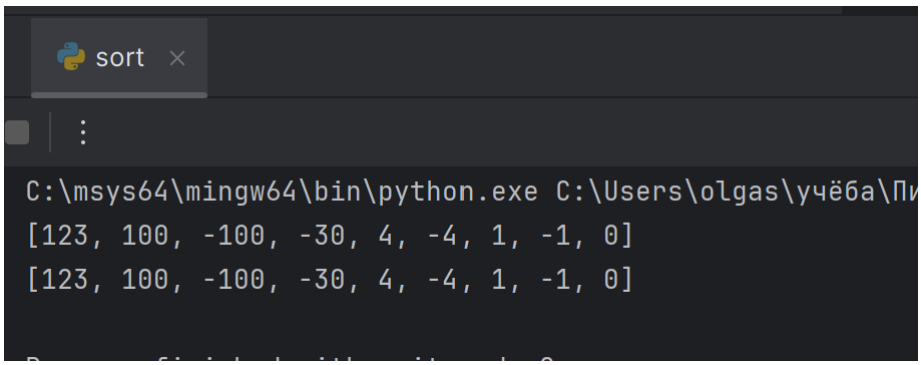
Шаблон реализации:

```
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__':
    result_with_lambda = sorted(data, key=lambda x: abs(x), reverse=True)
    print(result_with_lambda)

    result = sorted(data, key=abs, reverse=True)
    print(result)
```

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()
```

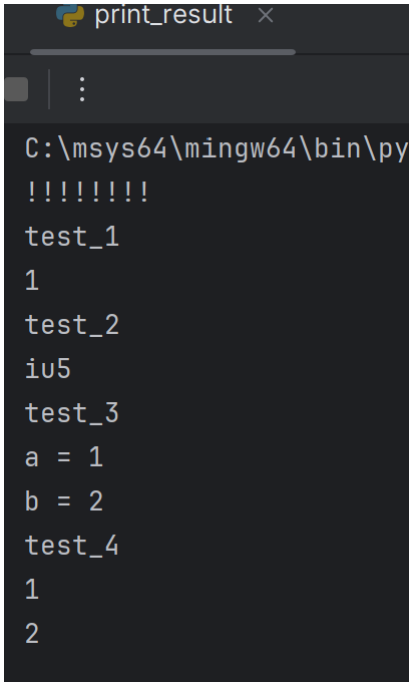
```
    test_3()
    test_4()
```
Результат выполнения:

```
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 i in result:
                print(i)
        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)
```

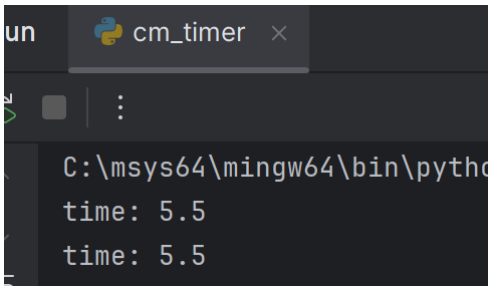
После завершения блока кода в консоль должно вывестись time: 5.5 (реальное время может несколько отличаться).
cm_timer_1 и cm_timer_2 реализуют одинаковую функциональность, но должны быть реализованы двумя различными способами (на основе класса и с использованием библиотеки contextlib).

Текст программы

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

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

    def __exit__(self, exc_type, exc_val, exc_tb):
        end = time()
        print(f"time: {end - self.start:.1f}")


@contextmanager
def cm_timer_2():
```

```
    start = time()
    yield
    end = time()
    print(f"time: {end - start:.1f}")


with cm_timer_1():
    sleep(5.5)

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

Результаты



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

from random import randint

from print_result import print_result

from cm_timer import cm_timer_1



path = sys.argv[1] if len(sys.argv) > 1 else None



with open(path, encoding='utf-8') as f:

    data = json.load(f)



@print_result

def f1(arg):

    return sorted({result_1['job-name'].lower() for result_1 in arg if 'job-name' in result_1},
key=str.lower)



@print_result

def f2(arg):

    return list(filter(lambda x: x.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 = [randint(100000, 200000) for i in arg]
    return [f"{profession}, зарплата {salary} руб." for profession, salary in zip(arg, salaries)]


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

Результаты

```
PS C:\Users\olgas\учёба\ПиКЯП\lab_3\lab_python_fp> python
f1
1с программист
2-ой механик
3-ий механик
4-ый механик
4-ый электромеханик
[химик-эксперт
asic специалист
javascript разработчик
rtl специалист
web-программист
web-разработчик
автожестянщик
автоинструктор
автомаляр
автомойщик
автор студенческих работ по различным дисциплинам
автослесарь
автослесарь - моторист
автоэлектрик
агент
агент банка
агент нпф
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 163481 руб.
программист / senior developer с опытом Python, зарплата 115570 руб.
программист 1с с опытом Python, зарплата 128649 руб.
программист c# с опытом Python, зарплата 100201 руб.
программист c++ с опытом Python, зарплата 126066 руб.
программист c++/c#/java с опытом Python, зарплата 144349 руб.
программист/ junior developer с опытом Python, зарплата 182410 руб.
программист/ технический специалист с опытом Python, зарплата 182152 руб.
программистр-разработчик  информационных систем с опытом Python, зарплата 181771 руб.
time: 0.1
```