

МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
им. Н.Э. Баумана

Факультет “Информатика и системы управления”
Кафедра ИУ5 “Системы обработки информации и управления”



Дисциплина “Парадигмы и конструкции языков программирования”

Отчет по лабораторной работе
“Функциональные возможности языка Python”

Выполнил:
Студент группы ИУ5-33Б
Просвирякова Д. С.

Преподаватель:
Гапанюк Ю. Е.

Москва 2025

Задание:

Задание лабораторной работы состоит из решения нескольких задач.

Файлы, содержащие решения отдельных задач, должны располагаться в пакете `lab_python_fp`. Решение каждой задачи должно располагаться 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 количество 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 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))))
```

Листинг кода

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

    if len(args) == 1:
        key = args[0]
        for item in items:
            if key in item and item[key] is not None:
                yield item[key]
    else:
        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__":
    goods = [
        {'title': 'Ковер', 'price': 2000, 'color': 'green'},
        {'title': 'Диван для отдыха', 'price': 5300, 'color': 'black'}
    ]

    print("Test field(goods, 'title'):")
    for title in field(goods, 'title'):
        print(title)

    print("\nTest field(goods, 'title', 'price'):")
    for item in field(goods, 'title', 'price'):
        print(item)
import random
```

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

if __name__ == "__main__":
    print("Test gen_random:")
    print("5 чисел от 1 до 3:")
    for num in gen_random(5, 1, 3):
        print(num, end=" ")
    print()

    print("\n10 чисел от 10 до 20:")
    for num in gen_random(10, 10, 20):
        print(num, end=" ")
    print()

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()

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 timer import timer_1
import os

path = None
if len(sys.argv) > 1:
    path = sys.argv[1]
else:
    path = 'data_light.json'

if not os.path.exists(path):
    test_data = [
        {"job-name": "Программист Python", "salary": 150000, "city": "Москва"}, 
        {"job-name": "Разработчик C++", "salary": 140000, "city": "Санкт-Петербург"}, 
        {"job-name": "Программист Java", "salary": 130000, "city": "Новосибирск"}, 
        {"job-name": "Программист JavaScript", "salary": 120000, "city": "Екатеринбург"}, 
        {"job-name": "Аналитик данных", "salary": 110000, "city": "Москва"}, 
        {"job-name": "Программист Python", "salary": 160000, "city": "Казань"}, 
        {"job-name": "программист Python", "salary": 170000, "city": "Москва"}]
    with open('data_light.json', 'w', encoding='utf-8') as f:
        json.dump(test_data, f, ensure_ascii=False, indent=2)
    path = 'data_light.json'

with open(path, 'r', encoding='utf-8') as f:
    data = json.load(f)

@print_result
def f1(arg):
    return sorted(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 [f'{job}, зарплата {salary} руб.' for job, salary in zip(arg, salaries)]

```

```
if __name__ == '__main__':
    with timer_1():
        f4(f3(f2(f1(data))))
data = [4, -30, 30, 100, -100, 123, 1, 0, -1, -4]

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

    result_with_lambda = sorted(data, key=lambda x: abs(x), reverse=True)
    print("С lambda:", result_with_lambda)
import time
from contextlib import contextmanager

class 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 timer_2():
    start_time = time.time()
    yield
    elapsed_time = time.time() - start_time
    print(f"time: {elapsed_time:.1f}")

if __name__ == "__main__":
    print("Test timer_1:")
    with timer_1():
        time.sleep(1.5)

    print("\nTest timer_2:")
    with timer_2():
        time.sleep(0.5)
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:
            try:
                item = next(self.items)
            except StopIteration:
                raise StopIteration

                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__":
    print("Test 1 - числа:")
    data = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]
    for item in Unique(data):
        print(item, end=" ")
    print()

    print("\nTest 2 - случайные числа:")
    from gen_random import gen_random
    data = gen_random(10, 1, 3)
    for item in Unique(data):
        print(item, end=" ")
    print()

    print("\nTest 3 - строки без ignore_case:")
    data = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']
    for item in Unique(data):
        print(item, end=" ")
    print()

    print("\nTest 4 - строки с ignore_case=True:")
    data = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']
    for item in Unique(data, ignore_case=True):
        print(item, end=" ")
    print()

```

Скриншот работы приложения

```

● daraprosvirakova@MacBook-Air-Dara-2 Prosviryakova-PCPL % python3 prosses_data.py
f1
Аналитик данных
Программист Java
Программист JavaScript
Программист Python
Разработчик C++
f2
Программист Java
Программист JavaScript
Программист Python
f3
Программист Java с опытом Python
Программист JavaScript с опытом Python
Программист Python с опытом Python
f4
Программист Java с опытом Python, зарплата 198734 руб.
Программист JavaScript с опытом Python, зарплата 107418 руб.
Программист Python с опытом Python, зарплата 154792 руб.
time: 0.0

```