МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
им. Н.Э. Баумана

Факультет "Информатика и системы управления"
Кафедра "Системы обработки информации и управления"

Дисциплина "Парадигмы и конструкции языков программирования"

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

Выполнил:
Студент группы ИУ5-35Б
Шаньгин Н.А.
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 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', 'pri

def field(items, *args):
    assert len(args) > 0
    # Необходимо реализовать генератор
```

```python
# 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

    result = []
    for item in items:
        dict = {}
        for arg in args:
            if arg in item:
                dict[arg] = item[arg]

        if len(args) == 1:
            result.append(dict[args[0]])
        else:
            result.append(dict)

    return result

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

    print("Первое задание:\n")
    print(field(goods, 'title'))
    print(field(goods, 'title', 'price'))
    print("\nКонец первого задания\n")
```

Результат выполнения

```
PS D:\Учёба\3 семестр\ПиКЯП\Shangin-PCPL-Labs-2025\lab_3-4> py lab_python_fp/field.py
Первое задание:

['Ковер', 'Диван для отдыха']
[{'title': 'Ковер', 'price': 2000}, {'title': 'Диван для отдыха', 'price': 5300}]

Конец первого задания

PS D:\Учёба\3 семестр\ПиКЯП\Shangin-PCPL-Labs-2025\lab_3-4>
```

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
```

```python
import random

# 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):
    return [random.randint(begin, end) for i in range(num_count)]

if __name__ == "__main__":
    print("Второе задание:\n")
    print(gen_random(5, 1, 10))
    print("\nКонец второго задания\n")
```

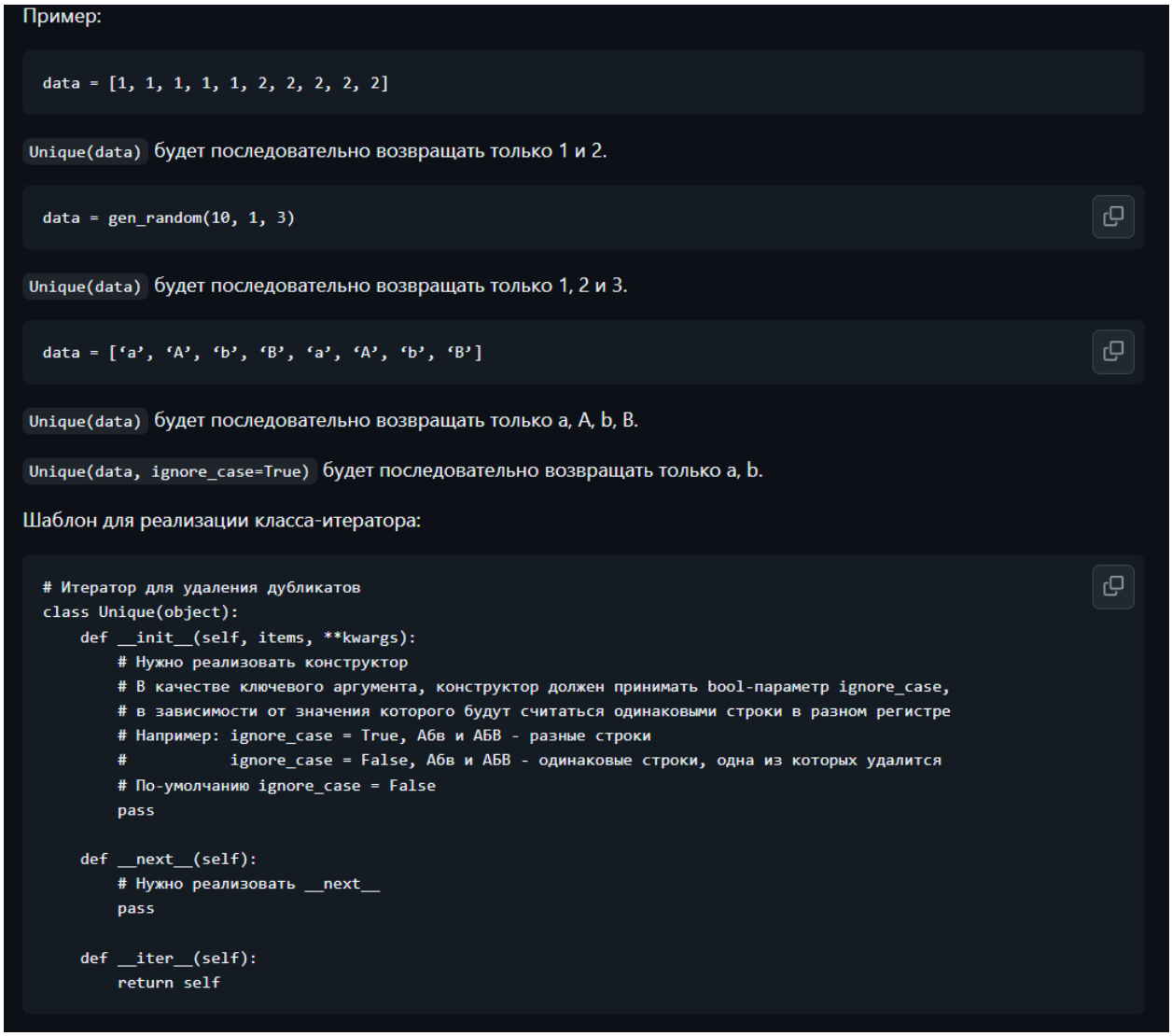
Результат выполнения

```
PS D:\Учёба\3 семестр\ПиКЯП\Shangin-PCPL-Labs-2025\lab_3-4> py lab_python_fp/gen_random.py
Второе задание:

[9, 1, 7, 6, 4]

Конец второго задания

PS D:\Учёба\3 семестр\ПиКЯП\Shangin-PCPL-Labs-2025\lab_3-4>
```

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
```

```
# 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.unique_items = []
        for item in items:
            compare_item = item
            if "ignore_case" in kwargs and isinstance(item, str):
                if kwargs["ignore_case"]:
                    compare_item = item.lower()
            if compare_item not in self.unique_items:
                self.unique_items.append(compare_item)
        self.current_index = 0

    def __next__(self):
        if self.current_index < len(self.unique_items):
            result = self.unique_items[self.current_index]
            self.current_index += 1
            return result
        else:
            raise StopIteration

    def __iter__(self):
        return self

if __name__ == "__main__":
    print("Третье задание:\n")
    not_uniq_mass = [4, 18, 4, 8, 1, 18, "AAA", "bobo", "aa", "Bobo", "aaA",
"aaa"]

    un_iterator = Unique(not_uniq_mass)
    low_un_iterator = Unique(not_uniq_mass, ignore_case=True)

    print("ignore_case = False/Not Stated\n")

    for i in range (len(un_iterator.unique_items)):
        print(un_iterator.__next__())

    print("\nignore_case = True\n")

    for i in range (len(low_un_iterator.unique_items)):
        print(low_un_iterator.__next__())

    print("\nКонец третьего задания\n")
```

Результат выполнения

```
PS D:\Учёба\3 семестр\ПиКЯП\Shangin-PCPL-Labs-2025\lab_3-4> py lab_python_fp/unique.py
Третье задание:

ignore_case = False/Not Stated

4
18
8
1
AAA
bobo
aa
Bobo
aaA
aaa

ignore_case = True

4
18
8
1
aaa
bobo
aa

Конец третьего задания

PS D:\Учёба\3 семестр\ПиКЯП\Shangin-PCPL-Labs-2025\lab_3-4>
```

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)
```

```
from math import fabs

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

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

    result_with_lambda = sorted(data, key=lambda date_el: fabs(date_el),
reverse=True)
    print(result_with_lambda)
```

Результат выполнения

```
PS D:\Учёба\3 семестр\ПиКЯП\Shangin-PCPL-Labs-2025\lab_3-4> py lab_python_fp/sort.py
[123, 100, -100, -30, 4, -4, 1, -1, 0]
[123, 100, -100, -30, 4, -4, 1, -1, 0]
PS D:\Учёба\3 семестр\ПиКЯП\Shangin-PCPL-Labs-2025\lab_3-4>
```

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
```

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

        if isinstance(result, list):
            print(*result, sep='\n')
        elif isinstance(result, dict):
            print(*[f"{key} = {result[key]}" for key in result.keys()], sep='\n')
        else:
            print(result)

        return result

    return decorator

@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

```
PS D:\Учёба\3 семестр\ПиКЯП\Shangin-PCPL-Labs-2025\lab_3-4> py lab_python_fp/print_result.py
!!!!!!!!
test_1
1
test_2
iu5
test_3
a = 1
b = 2
test_4
1
2
PS D:\Учёба\3 семестр\ПиКЯП\Shangin-PCPL-Labs-2025\lab_3-4>
```

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. 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).

```
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):
        print(f"time: {time.time() - self.start_time}")

@contextmanager
def cm_timer_2():
    start_time = time.time()

    try:
        yield start_time
    finally:
        print(f"time: {time.time() - start_time}")

if __name__ == "__main__":
    with cm_timer_1():
        time.sleep(1.5)
    with cm_timer_2():
        time.sleep(1.5)
```

Результат выполнения

```
PS D:\Учёба\3 семестр\ПиКЯП\Shangin-PCPL-Labs-2025\lab_3-4> py lab_python_fp/cm_timer.py
time: 1.5004286766052246
time: 1.5003864765167236
PS D:\Учёба\3 семестр\ПиКЯП\Shangin-PCPL-Labs-2025\lab_3-4>
```

Задача 7:

- 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))))
```

```python
import json

from pathlib import Path

from field import field
from gen_random import gen_random
from unique import Unique
from cm_timer import cm_timer_1
from print_result import print_result

path = Path("lab_python_fp/data/data_light.json")

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

with open(path, encoding="utf-8") 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(data):
    return list(Unique(field(data, "job-name"), ignore_case=True))


@print_result
def f2(data):
    return list(filter(lambda data_el: data_el.lower().startswith("программист"), data))


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


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

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

Результат выполнения:

```
инженер-технолог по покраске
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 1с
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 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 139659 руб
программист c++/c#/java с опытом Python, зарплата 190544 руб
программист 1с с опытом Python, зарплата 138791 руб
программистр-разработчик  информационных систем с опытом Python, зарплата 186971 руб
программист c++ с опытом Python, зарплата 156392 руб
программист/ junior developer с опытом Python, зарплата 150382 руб
программист / senior developer с опытом Python, зарплата 167518 руб
программист/ технический специалист с опытом Python, зарплата 151973 руб
программист c# с опытом Python, зарплата 153715 руб
time: 0.10700273513793945
PS D:\Учёба\3 семестр\ПиКЯП\Shangin-PCPL-Labs-2025\lab_3-4>
```