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
“Функциональные возможности языка Python”

Выполнил:
Студент группы ИУ5-35Б
Рябов М.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

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.

Листинг кода для Задачи №1

field.py

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

    result = []

    if len(args) == 1:
        result = [item[args[0]] for item in items if args[0] in item.keys()]
    else:
```

```
        filtered_items = [item for item in items if any(arg in item.keys() for
arg in args)]
        result = [{arg:item[arg] for arg in item.keys() if arg in args} for item
in filtered_items]

    return result

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

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

Выполнение программы для Задачи №1

```
(.venv) mmmm@LAPTOPIC:~/projects/pcpl_2025/lab_3$ python3 lab_python_fp/field.py
['Ковер', 'Диван для отдыха']
[{'title': 'Ковер', 'price': 2000}, {'title': 'Диван для отдыха', 'price': 5300}]
```

Задача №2

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.

Листинг кода для Задачи №2

gen_random.py

```
import random

def gen_random(num_count, begin, end):
    nums = [random.randint(begin, end) for i in range(num_count)]

    return nums

if __name__ == "__main__":
    print(gen_random(5, 1, 3))
```

Выполнение программы для Задачи №2

```
(.venv) mmmm@LAPTOPIC:~/projects/pcpl_2025/lab_3$ python3 lab_python_fp/gen_random.py
[1, 1, 3, 2, 3]
```

Задача №3

- 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.

Листинг кода для Задачи №3

unique.py

```
class Unique:
    def __init__(self, data, **kwargs):
        self.used_elements = set()
        self.data = data
        self.index = 0

        if len(kwargs) == 1 and "ignore_case" in kwargs.keys():
            self.ignore_case = kwargs["ignore_case"]
        else:
            self.ignore_case = False

    def __iter__(self):
```

```
        return self

    def __next__(self):
        while True:
            if self.index >= len(self.data):
                raise StopIteration
            else:
                current = self.data[self.index]
                self.index = self.index + 1

                if self.ignore_case and str(current).lower() in [str(i).lower()
for i in self.used_elements]:
                    continue

                if not self.ignore_case and current in self.used_elements:
                    continue

                self.used_elements.add(current)
                return current

if __name__ == "__main__":
    data = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']

    print(list(Unique(data)))
    print(list(Unique(data, ignore_case=True)))
```

Выполнение программы для Задачи №3

```
(.venv) mmmmm@LAPTOPIC:~/projects/pcpl_2025/lab_3$ python3 lab_python_fp/unique.py
['a', 'A', 'b', 'B']
['a', 'b']
```

Задача №4

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.

Листинг кода для Задачи №4

sort.py

```python
if __name__ == '__main__':
    data = [4, -30, 100, -100, 123, 1, 0, -1, -4]

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

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

Выполнение программы для Задачи №4

```
(.venv) mmmmm@LAPTOPIC:~/projects/pcpl_2025/lab_3$ python3 lab_python_fp/sort.py
[123, 100, -100, -30, 4, -4, 1, -1, 0]
[123, 100, -100, -30, 4, -4, 1, -1, 0]
```

Задача №5

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.

Листинг кода для Задачи №5

print_result.py

```python
def print_result(func):
    def wrapper(*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 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()
```

Выполнение программы для Задачи №5

```
(.venv) mmmmm@LAPTOPIC:~/projects/pcpl_2025/lab_3$ python3 lab_python_fp/print_result.py
!!!!!!!!
test_1
1
test_2
iu5
test_3
a = 1
b = 2
test_4
1
2
```

Задача №6

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).

Листинг кода для Задачи №6

cm_timer.py

```
import time
from contextlib import contextmanager

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

    def __exit__(self, exception_type, exception_value, traceback):
        elapsed_time = time.time() - self.start_time
        print(f"time: {elapsed_time}")

@contextmanager
def cm_timer_2(*args, **kwds):
    start_time = time.time()

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

if __name__ == "__main__":
    with cm_timer_1():
        time.sleep(5.5)

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

Выполнение программы для Задачи №6

```
(.venv) mmmmm@LAPTOPIC:~/projects/pcpl_2025/lab_3$ python3 lab_python_fp/cm_timer.py
time: 5.500105381011963
time: 5.500105619430542
```

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.

Листинг кода для Задачи №7

process_data.py

```
import json

from field import field
from unique import Unique
from gen_random import gen_random
```

```python
from print_result import print_result
from cm_timer import cm_timer_1

path = "./data_light.json"

with open(path) as f:
    data = json.load(f)

@print_result
def f1(arg):
    return list(Unique(field(arg, "job-name"), ignore_case=True))

@print_result
def f2(arg):
    return list(filter(lambda x: x.startswith("программист") or
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 = gen_random(len(arg), 100_000, 200_000)

    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))))
```

Выполнение программы для Задачи №7

```
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 168331 руб
Программист C++/C#/Java с опытом Python, зарплата 177359 руб
Программист 1С с опытом Python, зарплата 138565 руб
Программистр-разработчик  информационных систем с опытом Python, зарплата 142644 руб
Программист C++ с опытом Python, зарплата 184233 руб
Программист/ Junior Developer с опытом Python, зарплата 198337 руб
Программист / Senior Developer с опытом Python, зарплата 174968 руб
Программист/ технический специалист с опытом Python, зарплата 174542 руб
Программист C# с опытом Python, зарплата 121869 руб
time: 0.919961452484130
(.venv) mmmm@LAPTOPIC:~/projects/pcpl_2025/lab_3/lab_python_fp$
```