Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт перспективной инженерии
Департамент цифровых робототехнических систем и электроники

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ №6
дисциплины «Программирование на Python»

Выполнил:
Щегольков Савва Игоревич
2 курс, группа ИВТ-б-о-24-1,
09.03.01 «Информатика и
вычислительная техника»,
направленность (профиль)
«Автоматизированные системы
обработки информации и
управления», очная форма обучения

_____
(подпись)

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь, 2025 г.

Тема: Работа с функциями в языке Python.

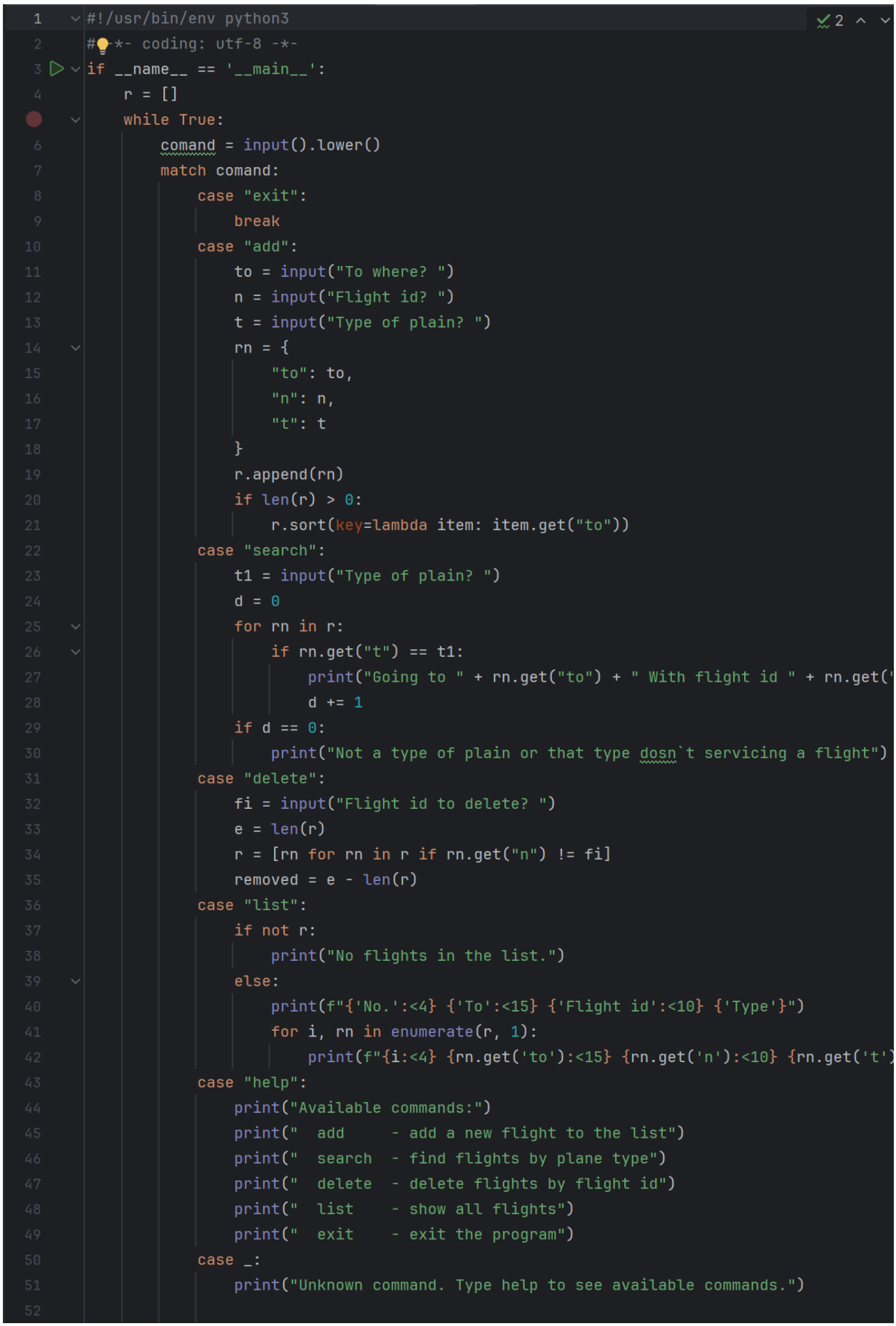
Цель: приобретение навыков по работе с функциями при написании программ с помощью языка программирования Python версии 3.x.

Порядок выполнения работы:

Ссылка на репозиторий: https://github.com/xouixao/lab6

5. Использовать словарь, содержащий следующие ключи: название пункта назначения рейса; номер рейса; тип самолета. Написать программу, выполняющую следующие действия: ввод с клавиатуры данных в список, состоящий из словарей заданной структуры; записи должны быть размещены в алфавитном порядке по названиям пунктов назначения; вывод на экран пунктов назначения и номеров рейсов, обслуживаемых самолетом, тип которого введен с клавиатуры; если таких рейсов нет, выдать на дисплей соответствующее сообщение.

Рисунок 1 – Условие задания 1



```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
if __name__ == '__main__':
    r = []
    while True:
        comand = input().lower()
        match comand:
            case "exit":
                break
            case "add":
                to = input("To where? ")
                n = input("Flight id? ")
                t = input("Type of plain? ")
                rn = {
                    "to": to,
                    "n": n,
                    "t": t
                }
                r.append(rn)
                if len(r) > 0:
                    r.sort(key=lambda item: item.get("to"))
            case "search":
                t1 = input("Type of plain? ")
                d = 0
                for rn in r:
                    if rn.get("t") == t1:
                        print("Going to " + rn.get("to") + " With flight id " + rn.get('
                        d += 1
                if d == 0:
                    print("Not a type of plain or that type dosn`t servicing a flight")
            case "delete":
                fi = input("Flight id to delete? ")
                e = len(r)
                r = [rn for rn in r if rn.get("n") != fi]
                removed = e - len(r)
            case "list":
                if not r:
                    print("No flights in the list.")
                else:
                    print(f"{'No.':<4} {'To':<15} {'Flight id':<10} {'Type'}")
                    for i, rn in enumerate(r, 1):
                        print(f"{i:<4} {rn.get('to'):<15} {rn.get('n'):<10} {rn.get('t')
            case "help":
                print("Available commands:")
                print("  add     - add a new flight to the list")
                print("  search  - find flights by plane type")
                print("  delete  - delete flights by flight id")
                print("  list    - show all flights")
                print("  exit    - exit the program")
            case _:
                print("Unknown command. Type help to see available commands.")

```

Рисунок 2 - Задание 1

24. Объединение и нормализация числовых данных

Задача:
Напишите функцию `normalize(*args, **kwargs)`,
которая принимает произвольное количество числовых аргументов или коллекций.
Если задан параметр `scale`, функция масштабирует все значения к диапазону `[0, scale]`.

Рисунок 3 – Условие задания 2

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
1 usage  xouixao
def normalize(*args, **kwargs):
    s = kwargs.get('scale', 1.0)
    v = []
    for a in args:
        if isinstance(a, (list, tuple, set, range)):
            v.extend(a)
        else:
            v.append(a)
    if not v:
        return []
    min_v = min(v)
    max_v = max(v)
    if min_v == max_v:
        return [0.0] * len(v)
    k = s / (max_v - min_v)
    return [(x - min_v) * k for x in v]
if __name__ == '__main__':
    print(normalize( *args: [1, 2, 3], (4, 5), scale=10))
```

```
Run  2
C:\Users\xumuk\AppData\Local\Programs\Python\Python311\python.exe C:\Users\xumuk\lab6\code\2.py
[0.0, 2.5, 5.0, 7.5, 10.0]

Process finished with exit code 0
```

Рисунок 4 - Задание 2.

24. Проверка сбалансированности скобок

Задача:
Реализуйте функцию `check_brackets(s, balance=0)`,
которая рекурсивно проверяет, сбалансированы ли круглые скобки в строке.

Рисунок 5 – Условие задания 3.

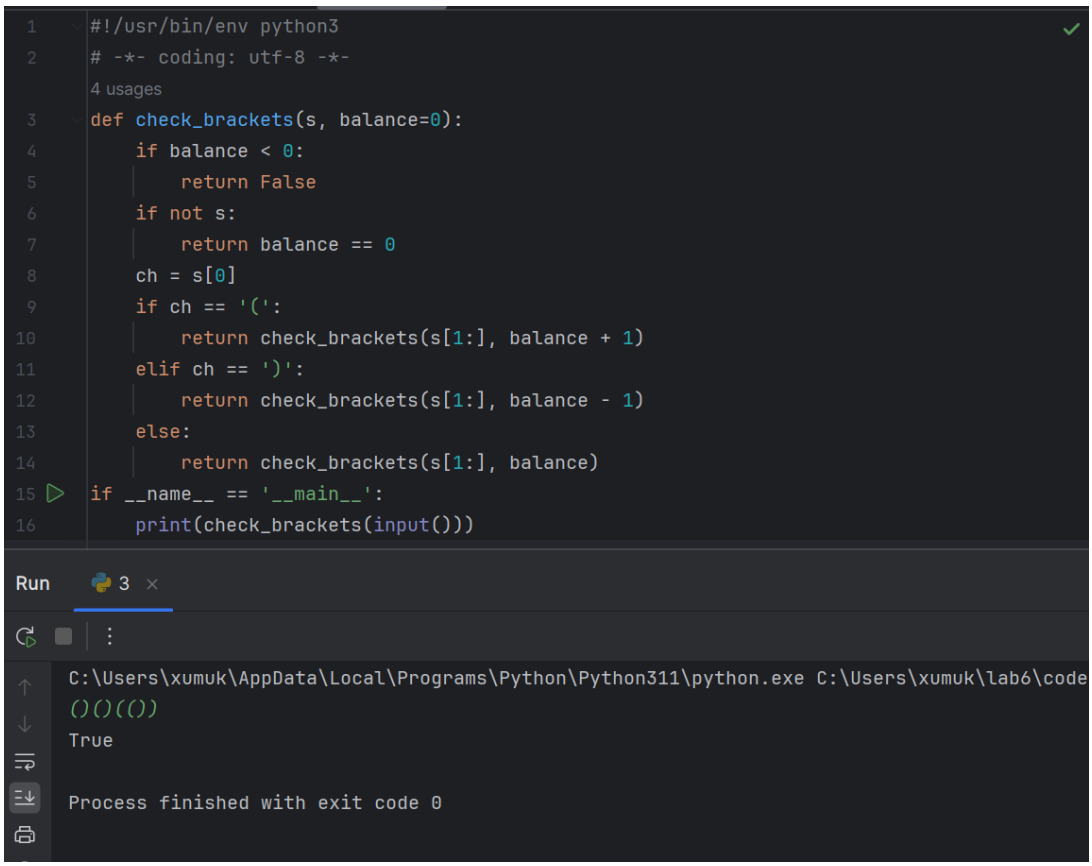


Рисунок 6 – Задание 3

Контрольные вопросы:

1. Функции в Python позволяют объединить несколько операторов в именованный блок кода, который можно многократно вызывать с разными аргументами, тем самым избавляя от дублирования кода, улучшая читаемость и позволяя логически разбивать программу на подзадачи.

2. Оператор def используется для объявления (определения) функции: задаёт её имя, список параметров и тело, а оператор return завершает выполнение функции и возвращает вызывающему коду значение (или несколько значений) вычисленного результата.

3. Локальные переменные существуют только внутри тела функции и используются для хранения её внутренних данных, не влияя на остальную программу, а глобальные объявляются на уровне модуля и доступны из функций (при необходимости с global), что позволяет хранить разделяемое состояние, но требует аккуратности.

4. В Python из функции можно вернуть несколько значений, перечислив их через запятую в операторе return (фактически возвращается кортеж), например: return a, b, c, а при вызове их можно распаковать: x, y, z = func().

5. Значения в функции передаются позиционно, по имени (именованные аргументы), с использованием значений по умолчанию, а также через распаковку коллекций с * (позиционные) и ** (словари именованных аргументов).

6. Значения по умолчанию задаются при определении функции, указывая после параметра знак = и выражение, например: def f(x, y=0, z=1): ... — если при вызове не передать y или z, будут использованы эти значения.

7. Lambda-выражения (lambda) используются для краткого объявления анонимных однострочных функций прямо в месте использования, например в качестве аргументов функций высшего порядка (map, sorted, filter и т.п.), когда создавать полноценную именованную функцию неудобно.

8. Согласно PEP 257 документирование кода осуществляется с помощью строк документации (docstrings) — строковых литералов сразу после заголовка модуля, функции, класса или метода, описывающих их назначение, параметры и возвращаемые значения, что затем может использоваться help(), IDE и генераторами документации.

9. Однострочные docstring-и используются для простых объектов и должны помещаться в одну строку с кратким описанием, а многострочные применяются для сложных случаев и включают краткое резюме первой строкой и более подробное объяснение с пустой строкой между резюме и остальным текстом.

10. Позиционными называются аргументы, значения которых передаются функции по порядку следования в списке параметров, без указания имён, например: f(10, 20) — здесь 10 и 20 попадают в первые два параметра по позиции.

11. Именованными (keyword arguments) называются аргументы, при передаче которых явно указывается имя параметра через =, например f(x=10, y=20), что позволяет менять порядок следования и делает вызов более читаемым.

12. Оператор * в контексте функций и вызовов используется для распаковки и группировки позиционных аргументов: в определении функции перед параметром (*args) он собирает произвольное число позиционных аргументов в кортеж, а в вызове (f(*seq)) — распаковывает элементы последовательности как отдельные позиционные аргументы.

13. Конструкция *args в определении функции собирает все дополнительные позиционные аргументы в кортеж args, а **kwargs собирает дополнительные именованные аргументы в словарь kwargs, что позволяет писать функции, принимающие переменное число параметров и гибко прокидывать их дальше.

14. Рекурсия нужна для решения задач, которые естественно определяются через себя (самоподобные структуры), например обход деревьев, вычисление факториала, чисел Фибоначчи, разбиение задач «разделяй и властвуй», когда проще описать решение через вызов функции самой себя на меньших подзадачах.

15. Базой рекурсии (базовым случаем) называют условие и соответствующую часть кода, при котором рекурсивные вызовы прекращаются и функция возвращает конкретный (обычно простой) результат без дальнейшего самовызова, предотвращая бесконечную рекурсию.

16. Стек программы (стек вызовов) — это структура данных типа LIFO, в которой при каждом вызове функции создаётся новый стековый кадр с локальными переменными и адресом возврата; по завершении функции этот кадр снимается со стека, и управление возвращается в вызывающую функцию.

17. Текущее значение максимальной глубины рекурсии в Python можно получить с помощью функции sys.getrecursionlimit() из модуля sys, предварительно выполнив import sys.

18. Если число рекурсивных вызовов превысит установленную максимальную глубину рекурсии, интерпретатор Python сгенерирует исключение RecursionError, тем самым останавливая дальнейшее углубление стека вызовов и защищая программу от переполнения стека.

19. Максимальную глубину рекурсии можно изменить с помощью функции sys.setrecursionlimit(n) из модуля sys, где n — новое целое значение лимита, однако увеличивать его нужно осторожно, чтобы не вызвать переполнение стека и падение интерпретатора.

20. Декоратор functools.lru_cache используется для автоматического кеширования результатов вызовов функции: при повторном вызове с теми же аргументами результат берётся из кеша, что значительно ускоряет вычисления для дорогостоящих или рекурсивных функций с повторяющимися подзадачами.

21. Хвостовая рекурсия — это форма рекурсии, при которой рекурсивный вызов является последней операцией в функции, и её результат сразу возвращается без дальнейших вычислений; в некоторых языках такие вызовы оптимизируются (tail call optimization) путём переиспользования одного стекового кадра, но в стандартном Python (CPython) оптимизация хвостовых вызовов не выполняется, и глубина хвостовой рекурсии по-прежнему ограничена лимитом рекурсии.

Вывод: в ходе выполнения лабораторной работы были приобретены навыки по работе с функциями при написании программ.