KAZAKH-BRITISH TECHNICAL UNIVERSITY

JSC Kazakh-British Technical University
School of Information Technology and Engineering

Practice work №5

PYTHON PROGRAMMING: FUNCTIONS, COLLECTIONS, AND CODE REFACTORING

Prepared by: Gainullin D.
Checked by: Abildayeva T.

Almaty, 2025

CONTENT

INTRODUCTION

In this assignment, we will develop three Python programs that demonstrate the use of functions, lists, and dictionaries. Each task focuses on a specific programming concept: calculating averages, managing collections of data, and refactoring code for efficiency. By completing these tasks, we will strengthen our understanding of structured programming and data manipulation in Python.

Task 1: Average Grade Calculator Using Functions and Collections

Task Description:
Write a program to calculate the average grade of students using a list and a function.
1. Create a function `calculate_average(grades)` that takes a list of grades and returns their average.
2. In the main program:
Create an empty list `grades`.
Input grades for five students, adding each grade to the list.
Call the `calculate_average` function and display the average grade for the entire group.

Solution:

Code:

```
def calculate_average(grades):
    # Calculate the average by summing all grades and
dividing by the number of grades
    average = sum(grades) / len(grades)
    return average

# Create an empty list to store student grades
grades = []

# Loop 5 times to get grades for 5 students
for i in range(5):
    # Get grade input from user and convert to float
    grade = float(input())
    # Add the grade to the grades list
    grades.append(grade)

# Calculate and display the average grade by calling the
function
print(calculate_average(grades))
```

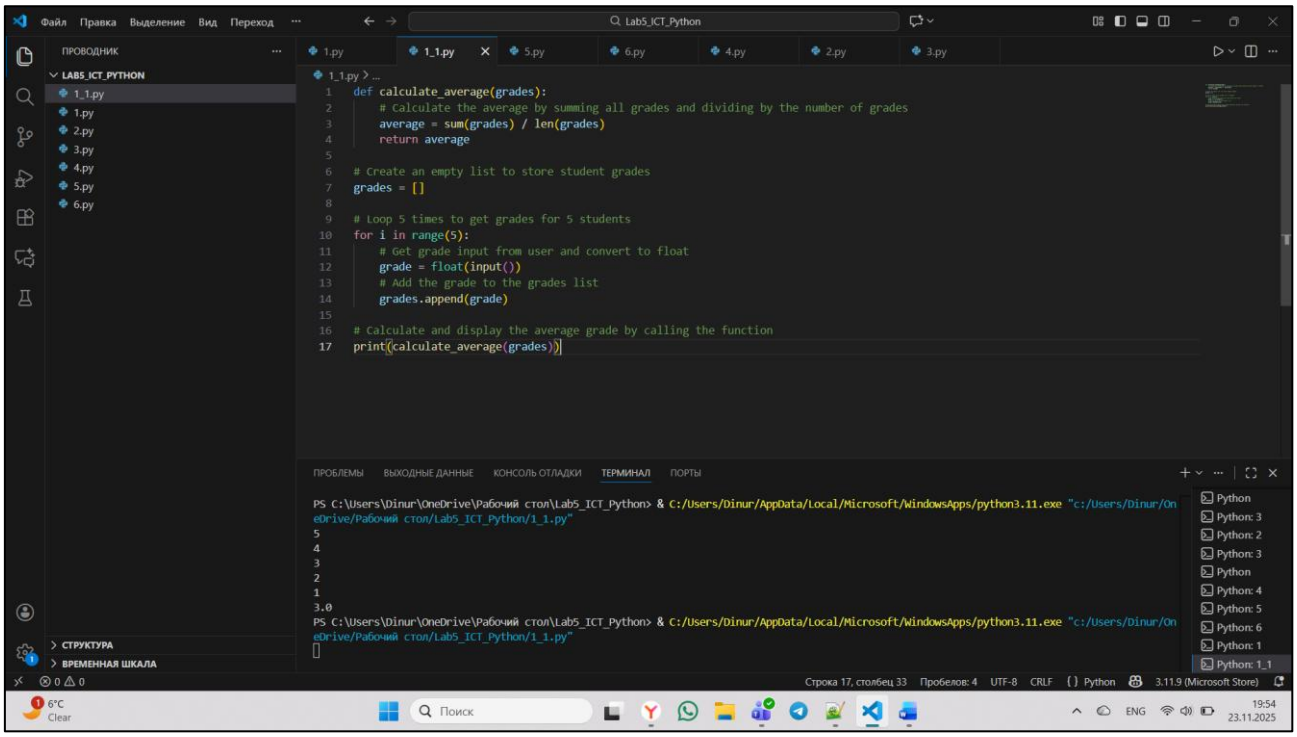

Figure 1 - Calculate the Average Grade

This program defines a function, calculate_average, that takes a list of grades and returns their arithmetic mean. The main part of the script then collects five grades from the user via input, stores them in a list, and finally calls the function to calculate and display the average grade.

Expalanation:

`calculate_average(grades)` - defines a function that takes a list of grades as parameter.
`um(grades) / len(grades)` - computes the mathematical average.
`return average` - returns the calculated average to the caller
`grades = []` - creates an empty list to store grades
`for i in range(5):` - iterates 5 times for 5 students
`grade = float(input())` - reads user input and converts to floating-point number
`grades.append(grade)` - adds each grade to the grades list
`print(calculate_average(grades))` - calls the average function and prints the result

Task 2: Product Manager Using Dictionary and Functions

Task Description:
Create a program to manage a list of products in a store using a dictionary and functions.
1. Create the following functions:
add_product(products, name, price): adds a new product with its price to the dictionary.
remove_product(products, name): removes a product by name.
view_products(products): displays all products and their prices.
2. In the main program:
Create an empty dictionary products.
Create a loop where the user can add products, delete products, and view the list of all products.

Solution:

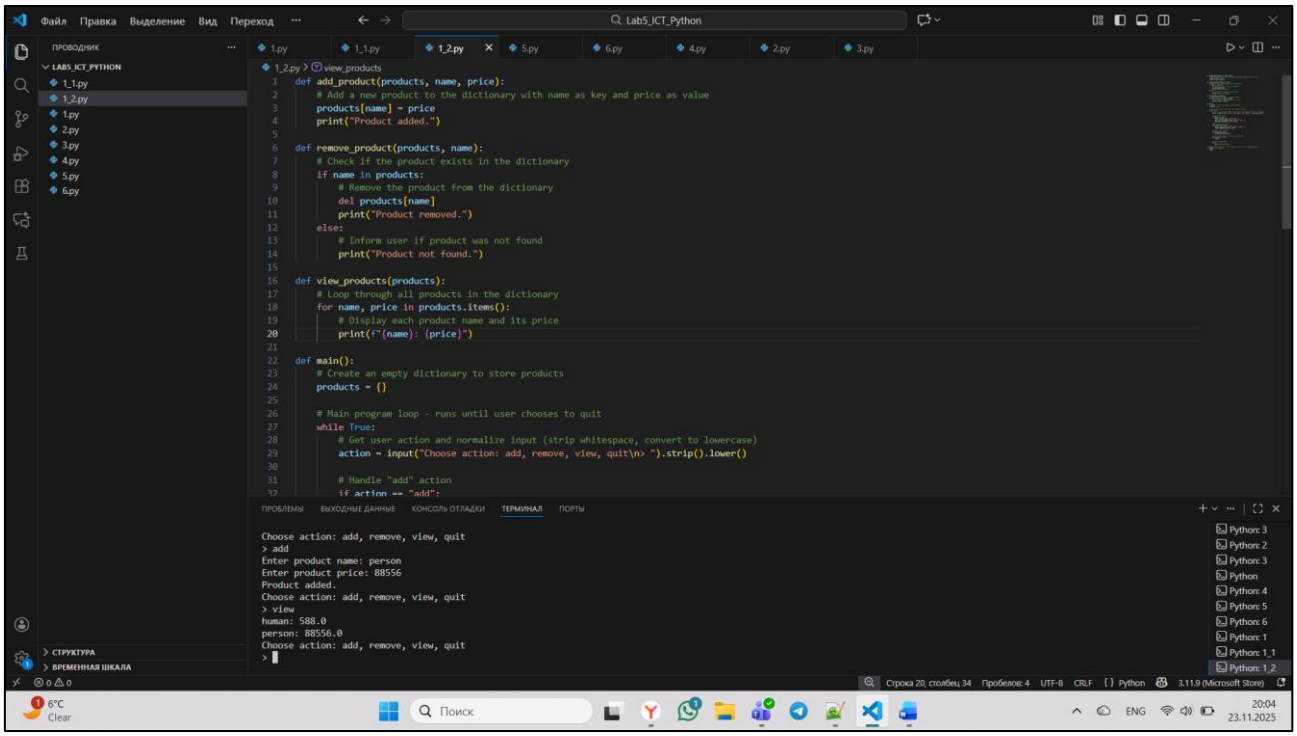

Figure 2 - Product Inventory System

As shown in Figure 2, this program uses a dictionary to store product names as keys and their prices as values. It implements three functions: add_product,

remove_product, and view_products. The main loop allows the user to interactively choose an action (add, remove, view, or quit) to manage the product list.

Code:

```
def add_product(products, name, price):
    # Add a new product to the dictionary with name as key and price as
value
    products[name] = price
    print("Product added.")
def remove_product(products, name):
    # Check if the product exists in the dictionary
    if name in products:
        # Remove the product from the dictionary
        del products[name]
        print("Product removed.")
    else:
        # Inform user if product was not found
        print("Product not found.")
def view_products(products):
    # Loop through all products in the dictionary
    for name, price in products.items():
        # Display each product name and its price
        print(f"{name}: {price}")
def main():
    # Create an empty dictionary to store products
    products = {}
    # Main program loop - runs until user chooses to quit
    while True:
        # Get user action and normalize input (strip whitespace, convert
to lowercase)
        action = input("Choose action: add, remove, view, quit\n>
").strip().lower()
        # Handle "add" action
        if action == "add":
            name = input("Enter product name: ")
            price = float(input("Enter product price: "))
            add_product(products, name, price)
        # Handle "remove" action
        elif action == "remove":
            name = input("Enter product name to remove: ")
            remove_product(products, name)
        # Handle "view" action
        elif action == "view":
            view_products(products)
        # Handle "quit" action - break out of the loop
        elif action == "quit":
            break
        # Handle invalid input
        else:
            print("Invalid action.")
# Standard Python practice - only run main() if this file is executed
directly
if __name__ == "__main__":
    main()
```

Explanation:

`def add_product(products, name, price):` - Adds new entries to the products dictionary
`def remove_product(products, name):` - Safely removes products with existence check
`def view_products(products):` - Displays all current products using dictionary iteration
`def main():` - Contains the program loop and user interface logic
`products = {}` - products dictionary stores products as key-value pairs (name: price)
`action = input("Choose action: add, remove, view, quit\n> ").strip().lower()` - Normalizes user input to handle case variations and whitespace

```
elif action == "remove":
            name = input("Enter product name to remove: ")
            remove_product(products, name)
        # Handle "view" action
        elif action == "view":
            view_products(products)
        # Handle "quit" action - break out of the loop
        elif action == "quit":
            break
```

- Continues running until user explicitly chooses "quit"

Task 3: Code Refactoring for Finding the Maximum Number

Task Description:
Refactor the following code, which finds the maximum value in a list of numbers, by creating a function for finding the maximum and making the code cleaner and more compact.

Solution:

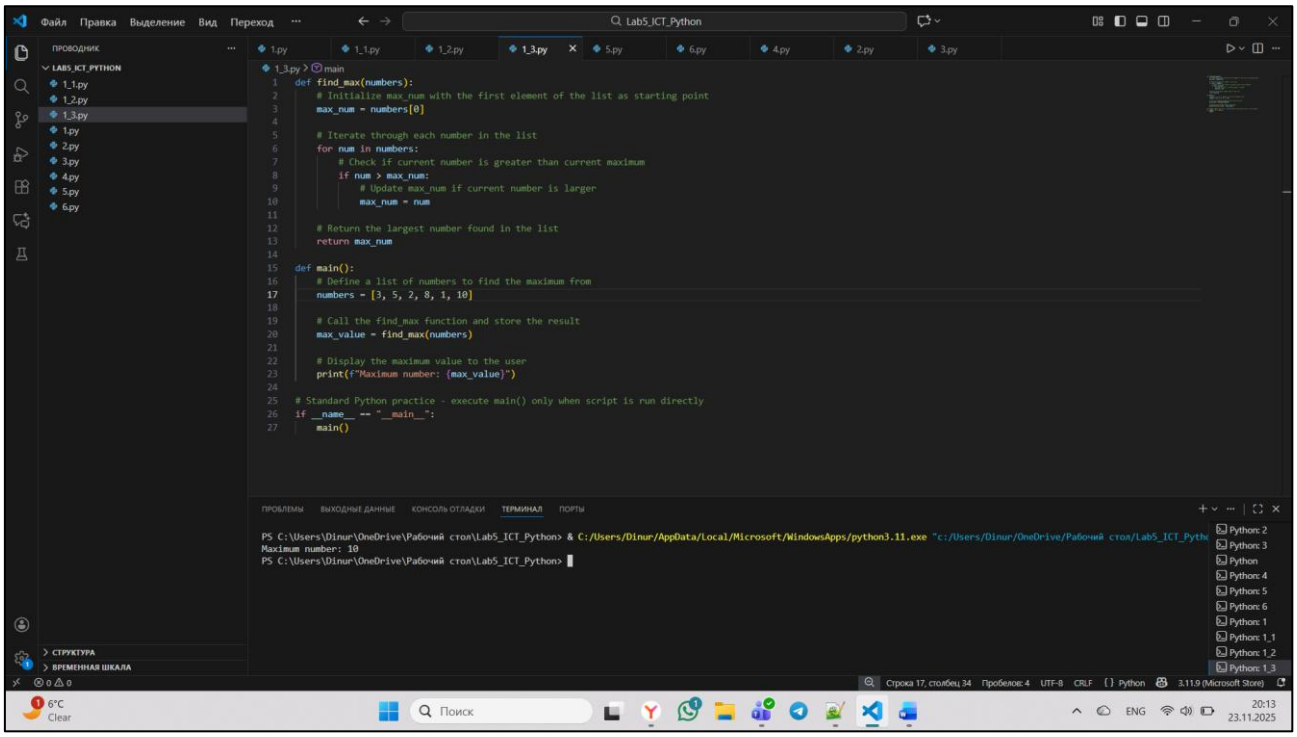

Figure 3 - Find the Maximum Value in a List

As presented in Figure 3, the program defines a function, find_max, which iterates through a given list of numbers to determine the largest value by tracking the current maximum. The main function initializes a list of integers and then calls find_max to calculate and subsequently display the maximum value to the user.

Code:

```
def find_max(numbers):
    # Initialize max_num with the first element of the
list as starting point
    max_num = numbers[0]

    # Iterate through each number in the list
    for num in numbers:
        if num > max_num:
            max_num = num
    # Return the largest number found in the list
    return max_num

def main():
    numbers = [3, 5, 2, 8, 1, 10]
    max_value = find_max(numbers)
    print(f"Maximum number: {max_value}")
```

```
# Standard Python practice - execute main() only when
script is run directly
if __name__ == "__main__":
    main()
```

Explanation:

`find_max(numbers)` – Takes a list of numbers as input parameter
`numbers[0]` – Sets initial maximum to the first element
`if num > max_num` – compares each subsequent number against current maximum
`max_num = num` - Updates maximum when a larger number is found
`return max_num` – Returns the final maximum value
`[3, 5, 2, 8, 1, 10]` - Creates a predefined list:
Calls `find_max()` with this list
`max_value` - Stores the returned value
`print(f"Maximum number: {max_value}")` – Prints the result in a formatted string

CONCLUSION

In conclusion, this assignment demonstrates how functions, lists, and dictionaries can be used to write clean, modular, and reusable code. We created programs to calculate averages, manage products, and find maximum values efficiently. These examples highlight the importance of structured logic and user interaction in Python programming.