KAZAKH-BRITISH TECHNICAL UNIVERSITY

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

Practice work №4

BASIC PROGRAMMING IN PYTHON

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

Almaty, 2025

CONTENT

INTRODUCTION

In this laboratory work, we explore the basic concepts of programming in Python. The tasks include creating simple programs that use conditional operators, loops, lists, and mathematical calculations. Through these exercises, we will learn how to perform arithmetic operations, work with repeated actions using loops, manipulate lists, and calculate values such as sums and factorials. This lab helps to develop fundamental programming logic and problem-solving skills using Python.

1. Calculator using conditional operators

1. The program asks the user to enter two numbers.
2. Then the program requests the operation to be performed with these numbers (+, -, *, /).
3. The program checks the entered operation and performs the corresponding arithmetic action.
4. If the user tries to perform division by zero, the program displays an error message.
5. The program displays the result of the operation.

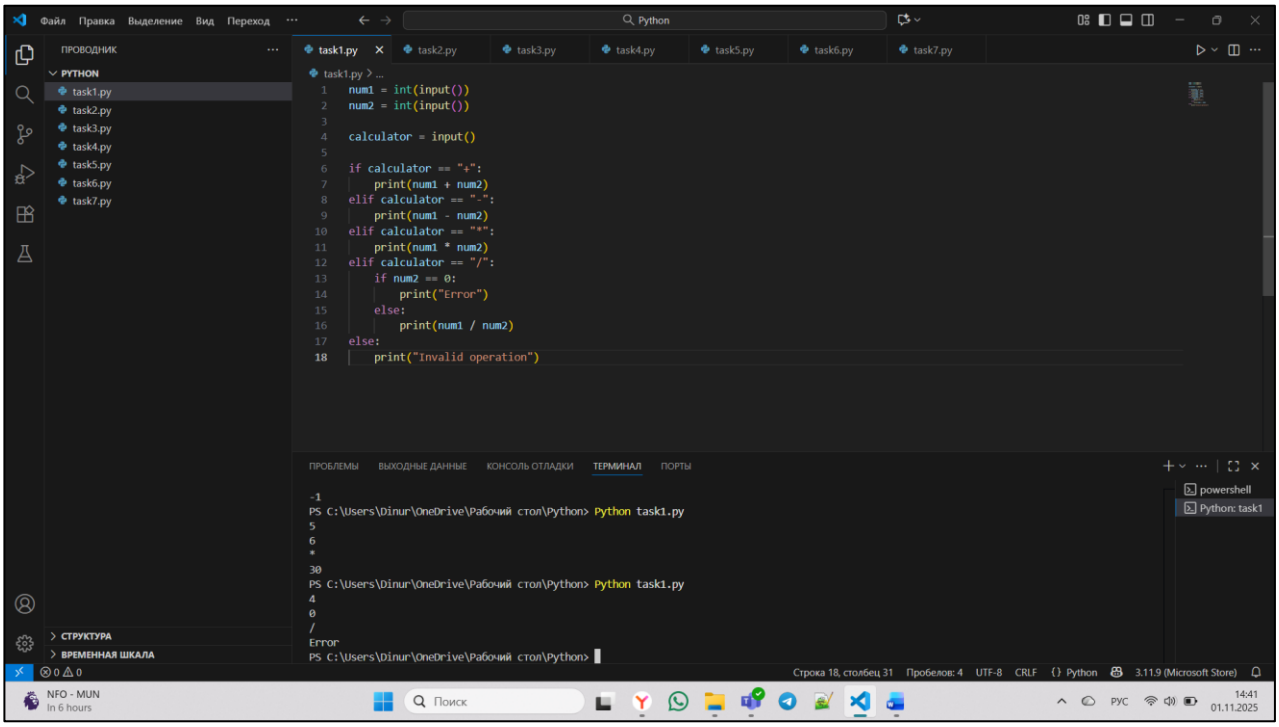

Figure 1 - Calculator program execution in Visual Studio Code

As shown in Figure 1, a simple calculator program written and executed in Visual Studio Code using Python. The program takes two numbers and an operator (+, -, *, /) as input, performs the corresponding arithmetic operation, and displays the result. If division by zero is attempted or an invalid operator is entered, an error message is shown.

Code:

```
num1 = int(input())
num2 = int(input())

calculator = input()

if calculator == "+":
```

```
    print(num1 + num2)
elif calculator == "-":
    print(num1 - num2)
elif calculator == "*":
    print(num1 * num2)
elif calculator == "/":
    if num2 == 0:
        print("Error")
    else:
        print(num1 / num2)
else:
    print("Invalid operation")
```

Explanation:
Inputs (num1 and num2) — **num1 = int(input()), num2 = int(input())**
These two lines are the program's input data.
Operation input (calculator) — calculator = input()
`if / elif / else` — the main selection logic
if calculator == '+': perform addition → **`print(num1 + num2)`**
elif calculator == '-': perform subtraction → **`print(num1 - num2)`**
elif calculator == '*': perform multiplication → **`print(num1 * num2)`**
elif calculator == '/': go to the division block.
else: if the user typed any other symbol, print "Invalid" to indicate an unrecognized operation.

2. Calculator using a loop

1. The program enters an endless loop where the user is asked to enter two numbers and select an operation to process them.
2. After the operation is completed, the program outputs the result and asks for input again, allowing the user to continue performing operations without restarting the program.
3. The user can exit the program by entering a special character or command (for example, 'q').
4. The program checks for errors, such as division by zero or incorrect input, and reports these errors, prompting the user to repeat the input.

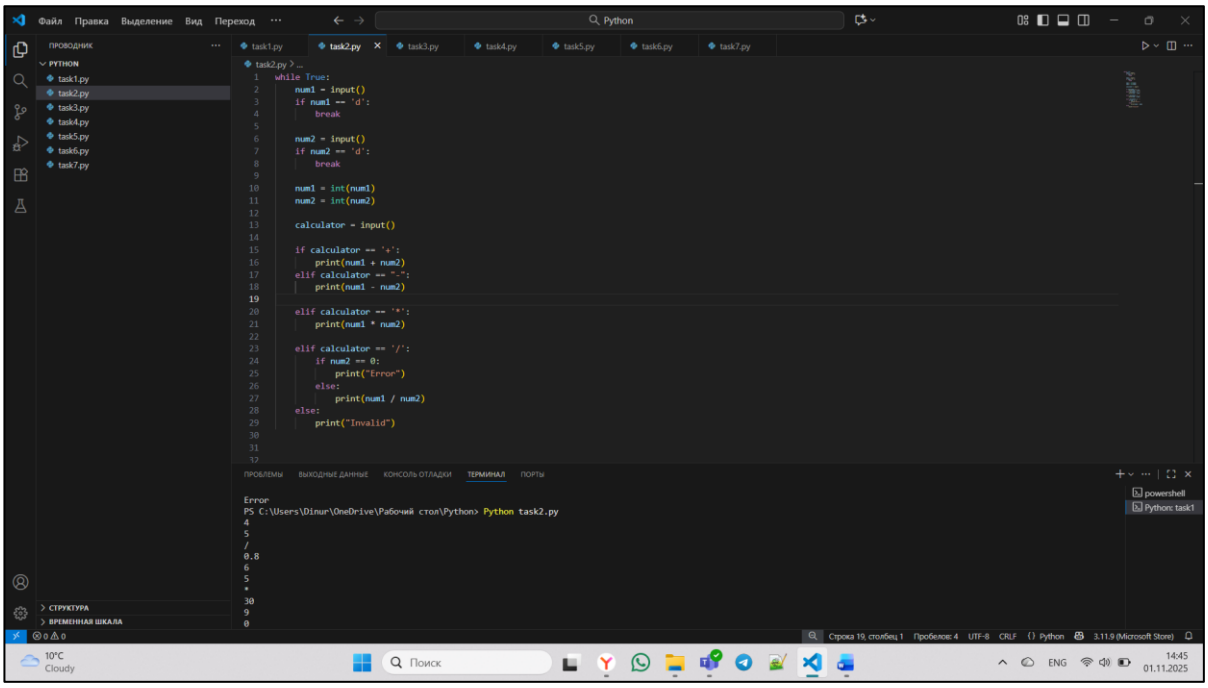

Figure 2 - Loop-based calculator program in Visual Studio Code

As presented in Figure 2, an improved version of the calculator program written in Python using a while True loop. This version allows continuous calculations until the user enters the letter ‘d’ to stop the program. It performs addition, subtraction, multiplication, and division, displaying the result for each operation. If the user tries to divide by zero or enters an invalid operator, the program shows an error message.

Code:

```
while True:
    num1 = input()
    if num1 == 'd':
        break

    num2 = input()
    if num2 == 'd':
        break

    num1 = int(num1)
    num2 = int(num2)

    calculator = input()

    if calculator == '+':
        print(num1 + num2)
    elif calculator == "-":
```

```
        print(num1 - num2)

    elif calculator == '*':
        print(num1 * num2)

    elif calculator == '/':
        if num2 == 0:
            print("Error")
        else:
            print(num1 / num2)
    else:
        print("Invalid")
```

Explanation:

`while True:` This creates an infinite loop, meaning the code inside will keep repeating again and again until a break statement stops it. It allows the calculator to work continuously, so the user can perform many calculations without restarting the program.

`break statements:` These are used to exit the loop when the user enters "d".

`num1 = int(num1), num2 = int(num2)` - once the user enters valid numbers, these lines convert the text inputs into integers, so the program can perform arithmetic operations on them.

`calculator = input() -` This asks the user to type the operation symbol (+, -, *, or /).The value entered is stored in the variable calculator.

`if calculator` == '+': → adds the two numbers → `print(num1 + num2)`

`elif calculator` == '-': → subtracts → `print(num1 - num2)`

`elif calculator` == '*': → multiplies → `print(num1 * num2)`

`elif calculator` == '/': → goes to the division block

```
if num2 == 0:
            print("Error")
        else:
            print(num1 / num2)
```

– Before dividing, the program checks whether num2 is 0. If yes, it prints "Error" because division by zero is not allowed. If not, it performs the division and prints the result.

```
else:
        print("Invalid")
```

- If the user enters a symbol other than +, -, *, or /, the program prints "Invalid" to show the input was incorrect.

3. Iteration and Cycles

Write a Python program that uses the for loop and the range function to output numbers from 1 to 20, inclusive. Make sure that each number is printed on a new line.

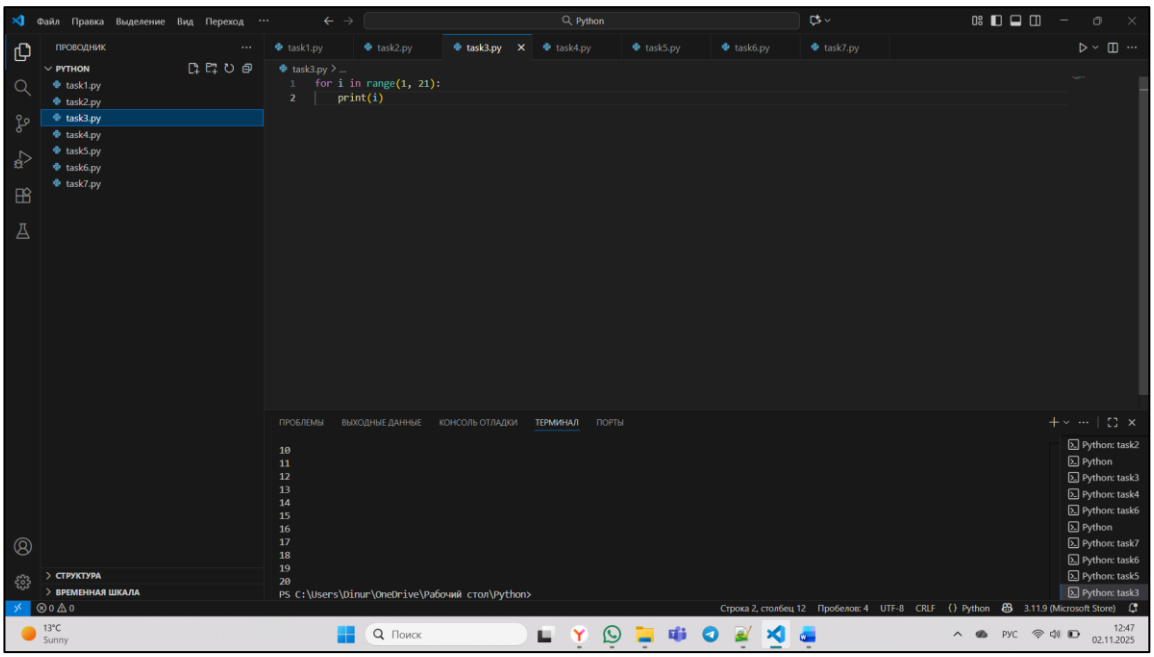

Figure 3 - Number Sequence Generator (1 to 20)

In this Figure 3, Python code demonstrates a simple loop structure that prints all integer numbers from 1 to 20 sequentially, where the range(1, 21) function generates numbers starting from 1 up to but not including 21, and each iteration of the for loop assigns the current number to variable i and prints it, resulting in a vertical list of numbers from 1 through 20 displayed in the console output.

Code:

```
for i in range(1, 21):
    print(i)
```

Explanation: In simple words:

`for` means “repeat for each number in the range.”

`i` represents the current number from that range — first it’s 1, then 2, then 3, and so on up to 20.

4. Working with Lists and Loops

Create a list of numbers from 1 to 10. Using the for loop, print the square of each number from the list.

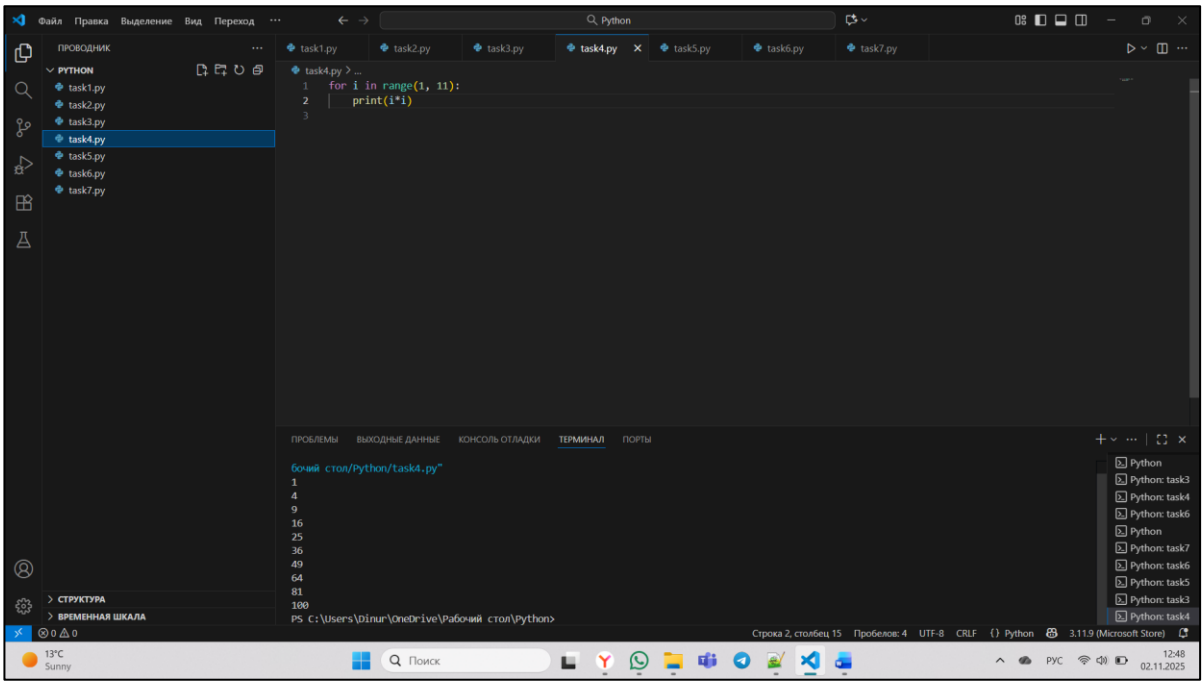

Figure 4 - Squares of Numbers Calculator (1 to 10)

As presented in Figure 4, this Python code calculates and prints the squares of all integer numbers from 1 to 10 using a for loop that iterates through the sequence generated by **range(1, 11),** where during each iteration the current number i is multiplied by itself **(**i*i**),** and the resulting squared value is printed to the console, producing a vertical list of perfect squares from 1 through 100 in sequential order.

Code:

```
for i in range(1, 11):
    print(i*i)
```

Explanation: The `range(1, 11)` function creates a sequence of numbers starting at 1 and ending at 10. In each loop, the variable `i` takes one of these numbers, and `print(i*i)` displays the result of multiplying that number by itself (its square).

5. Combining Food

Using the provided lists of products (bread, meat, vegetables, sauces), write a program that generates and outputs all possible sandwich combinations. Extract the lists for each product category from the text.

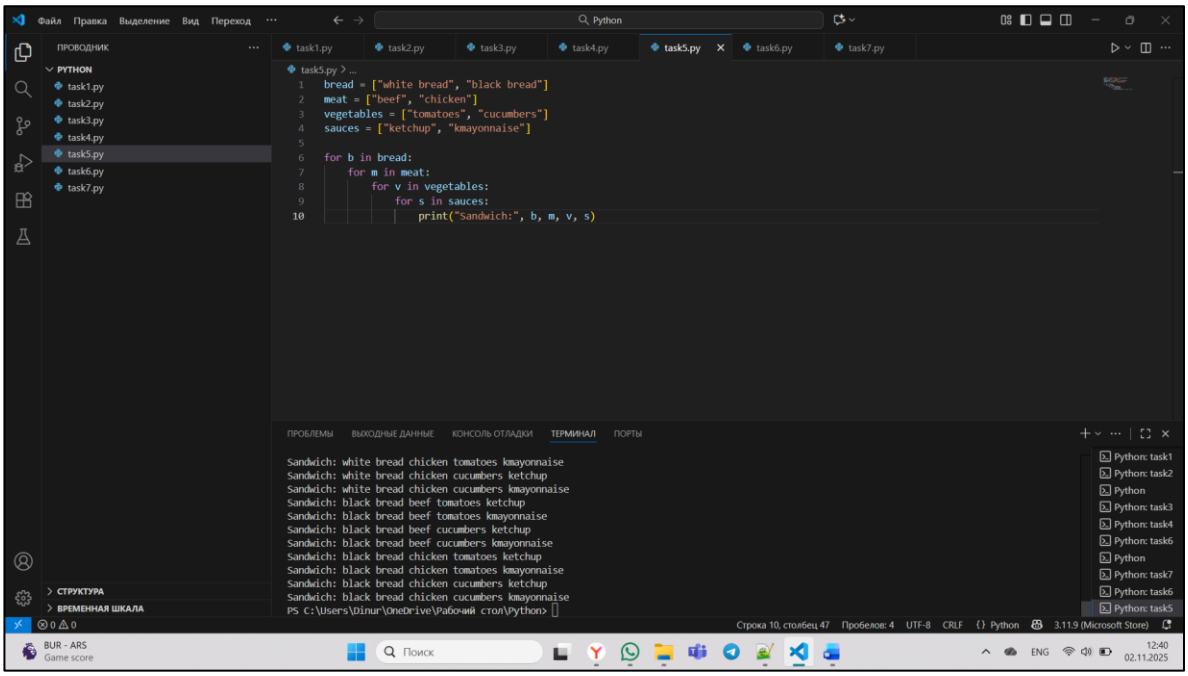

Figure 5 - Sandwich Ingredient Combination Algorithm

As shown in Figure 5, this Python code systematically generates all possible sandwich combinations by using nested loops to iterate through four predefined ingredient lists (bread, meat, vegetables, and sauces), where for each type of bread, it loops through every meat option, then for each bread-meat pair, it loops through all vegetable choices, and finally for each bread-meat-vegetable combination, it loops through all available sauces, printing out a complete sandwich description for every unique combination, resulting in a total of $2 \times 2 \times 2 \times 2 = 16$ different sandwich variations that represent the Cartesian product of all the ingredient sets.

Code:

```
bread = ["white bread", "black bread"]
meat = ["beef", "chicken"]
vegetables = ["tomatoes", "cucumbers"]
sauces = ["ketchup", "kmayonnaise"]

for b in bread:
    for m in meat:
        for v in vegetables:
            for s in sauces:
                print("Sandwich:", b, m, v, s)
```

Explanation: There are four lists: **bread, meat, vegetables, sauces**. They serve as data sources for the sandwich combinations.

The nested `for` loops go through every possible combination: The outer loop picks a type of bread, the next loop picks a type of meat, the next one picks a vegetable, and the innermost loop picks a sauce.

```
for b in bread:
    for m in meat:
        for v in vegetables:
            for s in sauces:
```

- this is a nested loops(loops inside loops) .Each loop goes through one list: The first loop picks a type of bread (b). The second loop picks a type of meat (m). The third loop picks a type of vegetable (v). The fourth loop picks a type of sauce (s). Together, they go through **every possible combination** of ingredients.

`print("Sandwich:", b, m, v, s)` - This line prints one complete sandwich combination made from the current choices of bread, meat, vegetable, and sauce. It shows which ingredients are used in that particular sandwich.

6. Summation Of Numbers

Write a program that counts the sum of all even numbers and the sum of all odd numbers from 1 to 100. The program must output both amounts separately.

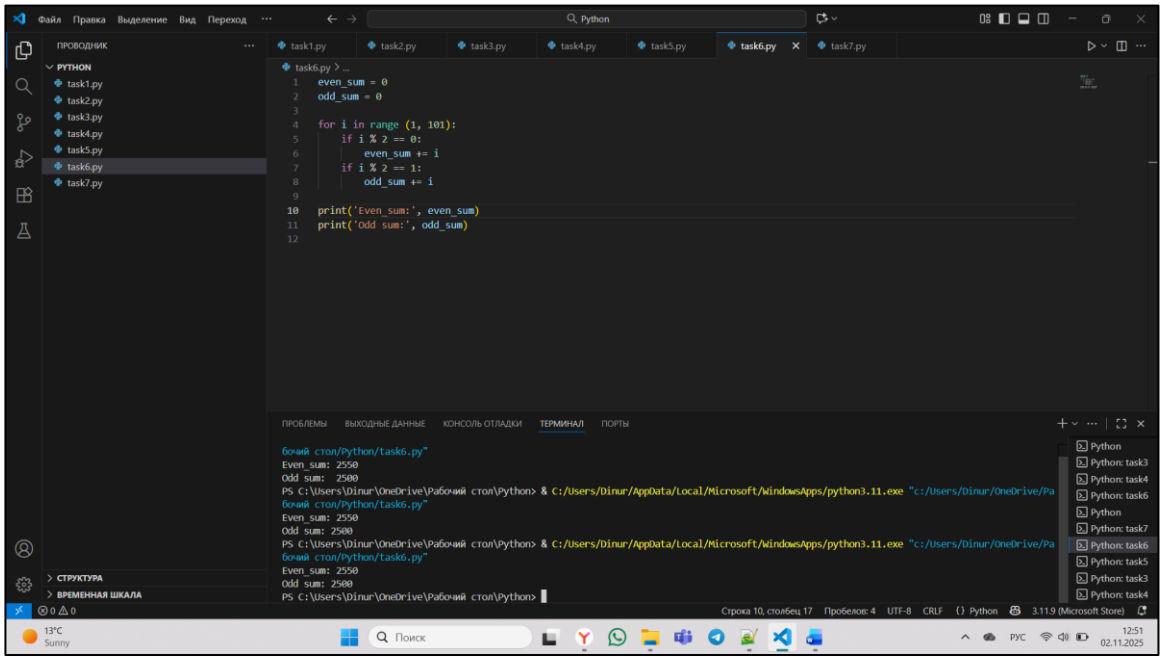

Figure 6 - Even and Odd Numbers Sum Calculator (1 to 100)

In this Figure 6, Python code calculates the sum of all even numbers and the sum of all odd numbers between 1 and 100 using a for loop that iterates through the range from 1 to 100, where during each iteration the conditional statement **if** i **%** 2 **==**

0 checks if the current number is even and adds it to even_sum, while the condition if i **%** 2 == 1 identifies odd numbers and accumulates them in odd_sum, finally printing both calculated sums after completing the loop to display the total of all even numbers (2550) and all odd numbers (2500) within the specified **range.**

Code:

```
even_num = 0
odd_num = 0

for i in range (1, 101):
    if i % 2 == 0:
        even_num = even_num + i
    elif i % 2 == 1:
        odd_num = odd_num + i

print("Even sum:", even_num)
print("Odd sum:", odd_num)
```

Explanation: Two variables are created at the start: `even_sum = 0` to store the total of even numbers, `odd_sum = 0` to store the total of odd numbers.
The for `i in range(1, 101):` loop goes through all numbers from 1 to 100.
Inside the loop:
The condition if `i % 2 == 0:` checks if the number is even (divisible by 2). If it's true, that number is added to even_sum.
The condition if `i % 2 == 1:` checks if the number is odd. If it's true, that number is added to odd_sum.

7. The Factorial Of The Number

Implement a Python program that asks the user for a number and calculates the factorial of that number. Recall that the factorial of the number n (denoted as n!) is the product of all positive integers from 1 to n.

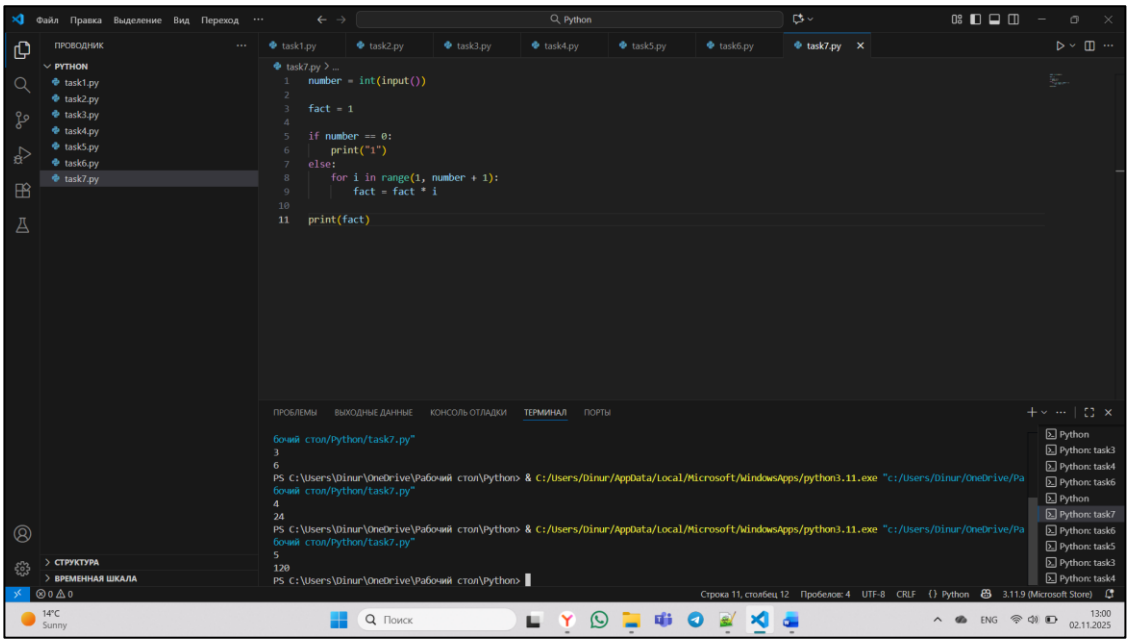

Figure 7 - Factorial Calculator

As shown in Figure 7, this Python code calculates the factorial of a user-**input** number by first reading an integer value, then initializing a variable fact to 1, handling the special case where **input** is 0 by directly printing 1 (since 0! = 1), and for any positive number using a for loop that iterates from 1 to the input number, progressively multiplying each integer with the accumulated fact value, ultimately printing the final computed factorial result which represents the product of all integers from 1 to the given number.

Code:

```
num = int(input())
factorail = 1
if num == 0:
        print("1")
else:
    for i in range(1, num + 1):
        factorail = factorail * i
    print(factorail)
```

Explanation: `number = int(input())` — asks the user to enter a number and converts it to an integer.

`fact = 1` – creates a variable to store the result of the factorial (it starts from 1).

`if number == 0:` - checks if the user entered 0. By definition, the factorial of 0 is 1, so it prints "1".

```
else:
    for i in range(1, number + 1):
            fact = fact * I
```
- if the number is not 0, the program uses a for loop to multiply all numbers from 1 up to the entered number.
The for loop goes through all numbers from 1 to the given number (`number + 1` makes the range inclusive). For each `i`, the program multiplies the current fact by i to build up the factorial result step by step.
For example, if the number is 5, it calculates $1 \times 2 \times 3 \times 4 \times 5 = 120$.
Finally, `print(fact)` prints the factorial result on the screen.

CONCLUSION

In conclusion, this laboratory work allowed us to understand and apply key Python programming concepts such as conditions, loops, and list operations. We created programs that perform calculations, repeat tasks efficiently, and handle user input correctly. Completing these exercises strengthened our understanding of basic programming structures and prepared us for more complex Python applications in future work.