PythonGuides.com

51 PYTHON PROGRAMS

BY: BIJAY KUMAR

Table of Contents

ABOUT AUTHOR – BIJAY KUMAR

Bijay is a Microsoft MVP and the founder of PythonGuides.com. He has more than 17 years of experience in various technologies. He started his journey with Microsoft technologies like .Net, SharePoint, etc. And then got a chance to work with Python and machine learning.

He got working experience in various companies like Hewlett Packard (HP), TCS, KPIT, etc. before starting TSinfo Technologies.

Over the last few years, he has had expertise in various Python libraries such as Pandas, NumPy, Matplotlib, Tkinter, Turtle, Django, Tensorflow, Pytorch, scipy, etc.

Connect With Him:

- MVP Profile: https://mvp.microsoft.com/en-us/PublicProfile/5000972
- Twitter: https://twitter.com/fewlines4biju
- LinkedIn: http://in.linkedin.com/in/fewlines4biju/

Subscribe to PythonGuides.com YouTube Channel:
https://www.youtube.com/@pythonguides?sub_confirmation=1

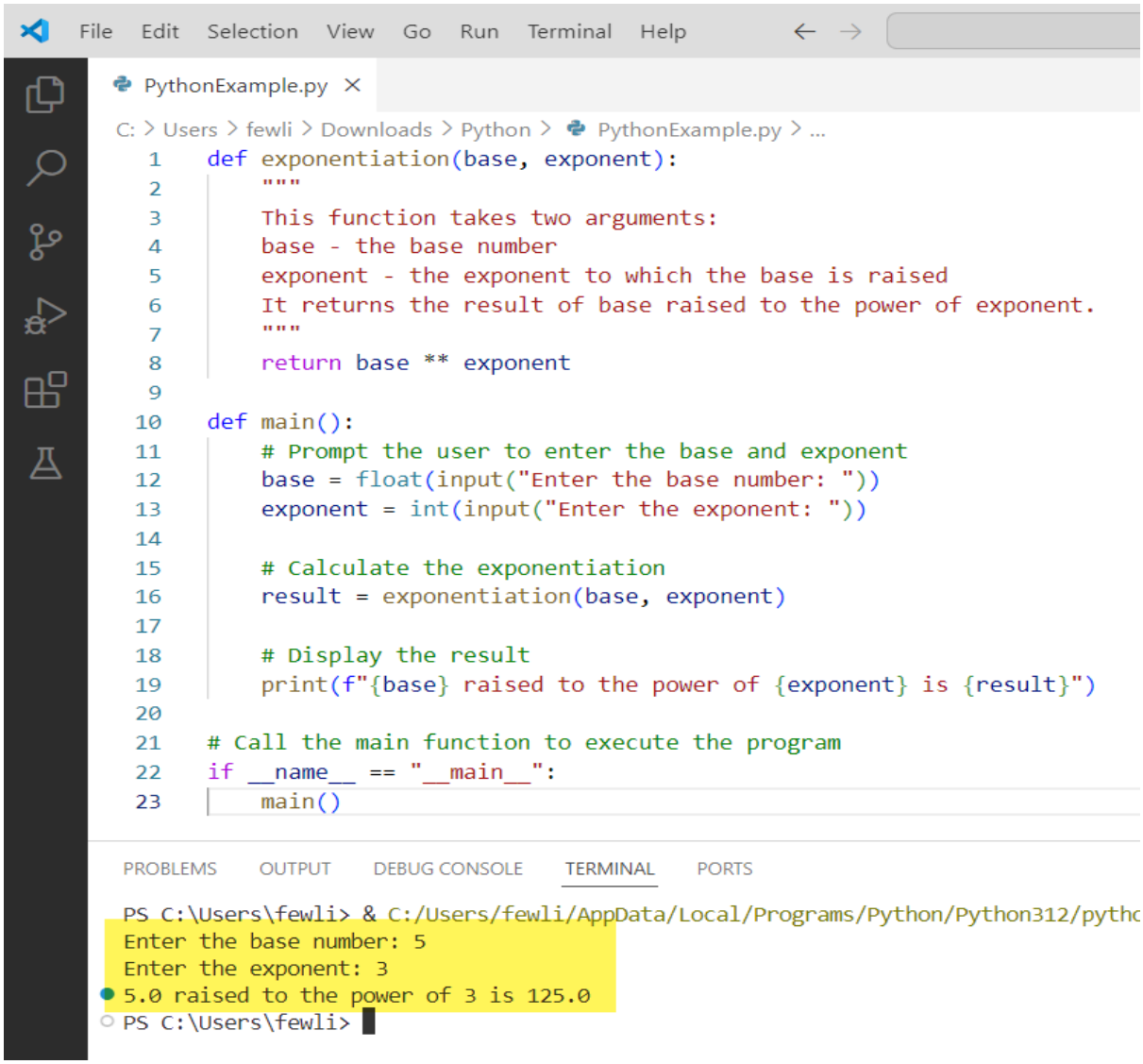
1# PYTHON PROGRAM TO FIND THE EXPONENTIATION OF A NUMBER

Exponentiation is a mathematical operation that involves raising a number (the base) to the power of an exponent. In Python, exponentiation can be performed using the ** operator or the pow() function.

Here is the complete Python code:

```
def exponentiation(base, exponent):
    """
    This function takes two arguments:
    base - the base number
    exponent - the exponent to which the base is raised
    It returns the result of base raised to the power of exponent.
    """
    return base ** exponent

def main():
    # Prompt the user to enter the base and exponent
    base = float(input("Enter the base number: "))
    exponent = int(input("Enter the exponent: "))

    # Calculate the exponentiation
    result = exponentiation(base, exponent)

    # Display the result
    print(f"{base} raised to the power of {exponent} is {result}")

# Call the main function to execute the program
if __name__ == "__main__":
    main()
```

Here is the output you can see in the screenshot below:



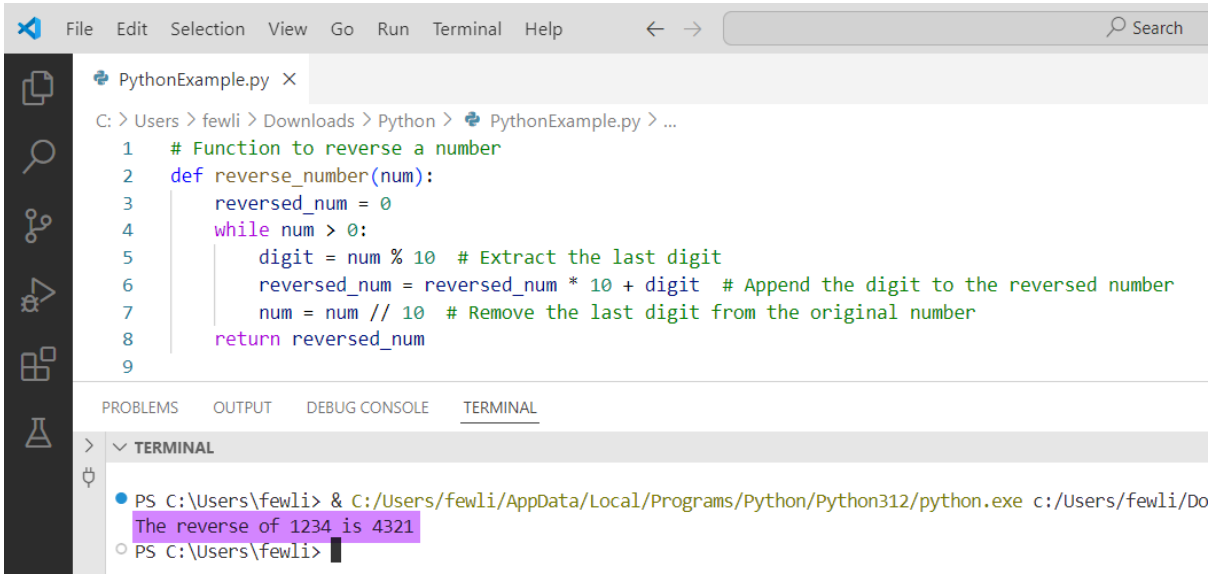
2# PYTHON PROGRAM TO FIND REVERSE OF A NUMBER

Here is a Python program to reverse a number using a while loop, along with an explanation:

```
# Function to reverse a number
def reverse_number(num):
    reversed_num = 0
    while num > 0:
        digit = num % 10  # Extract the last digit
        reversed_num = reversed_num * 10 + digit  # Append the digit to the
reversed number
        num = num // 10  # Remove the last digit from the original number
    return reversed_num

# Example usage
number = 1234
```

```
reversed_number = reverse_number(number)
print(f"The reverse of {number} is {reversed_number}")
```

Here is the output you can see in the screenshot below:



Explanation:

1. Extract the Last Digit: We use num % 10 to get the last digit of the number.
2. Append the Digit: We multiply the current reversed_num by 10 and add the extracted digit to it.
3. Remove the Last Digit: We use integer division num // 10 to remove the last digit from the original number.
4. Loop Until Number is Zero: The loop continues until all digits are processed.

This method ensures that the digits of the original number are reversed and concatenated correctly to form the reversed number.

3# PYTHON PROGRAM TO CHECK IF A NUMBER IS A PRIME NUMBER

Here is a Python program to check if a number is prime, along with an example and a detailed explanation:

```
# Function to check if a number is prime
def is_prime(num):
    if num <= 1:
```

```
        return False  # Numbers less than or equal to 1 are not prime
    for i in range(2, int(num**0.5) + 1):  # Check divisors up to the square
root of num
        if num % i == 0:
            return False  # If divisible, num is not prime
    return True  # If no divisors found, num is prime

# Example usage
number = 29
if is_prime(number):
    print(f"{number} is a prime number.")
else:
    print(f"{number} is not a prime number.")
```

Explanation:

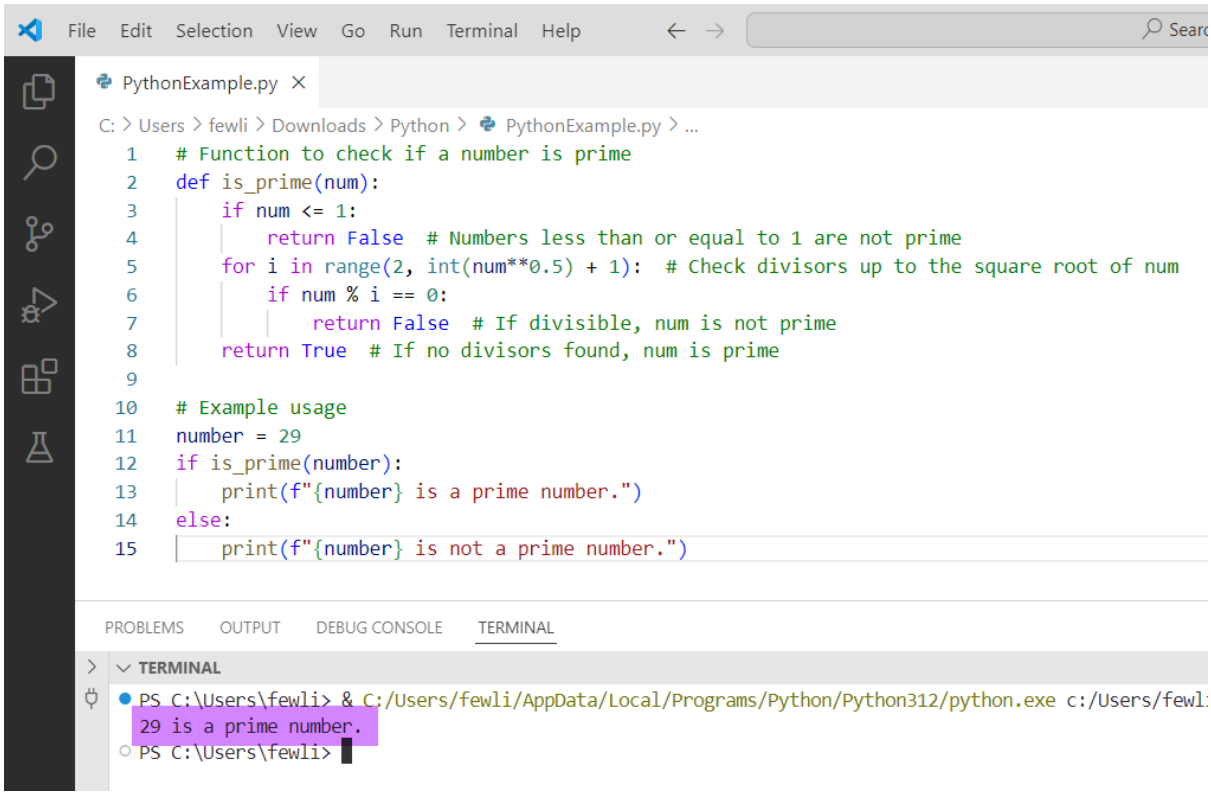
1. Check if Number is Less Than or Equal to 1: Prime numbers are greater than 1. So, if num <= 1, the function returns False.
2. Check Divisors Up to the Square Root: We only need to check for factors up to the square root of the number (int(num**0.5) + 1). This is because a larger factor of the number would necessarily be paired with a smaller factor that we would have already checked.
3. Check for Divisibility: We use a for loop to iterate from 2 up to the square root of the number. If num % i == 0 for any i, the number is not prime, and we return False.
4. Return True if No Divisors Found: If no divisors are found in the loop, the number is prime, and we return True.

Example:

For the number 29:

- It is greater than 1.
- The square root of 29 is approximately 5.39, so we check divisors from 2 to 5.
- 29 is not divisible by 2, 3, 4, or 5.
- Since no divisors are found, 29 is a prime number.

Here is the exact output:



4# PYTHON PROGRAM TO FIND LARGEST OF THREE NUMBERS

Here is a Python program to find the largest of three numbers.

```
# Function to find the largest of three numbers
def find_largest(num1, num2, num3):
    if (num1 >= num2) and (num1 >= num3):
        largest = num1
    elif (num2 >= num1) and (num2 >= num3):
        largest = num2
    else:
        largest = num3
    return largest

# Example usage
number1 = 10
number2 = 25
number3 = 20
largest_number = find_largest(number1, number2, number3)
print(f"The largest number among {number1}, {number2}, and {number3} is
{largest_number}")
```

Explanation:

1. Define the Function: The function find_largest takes three arguments: num1, num2, and num3.

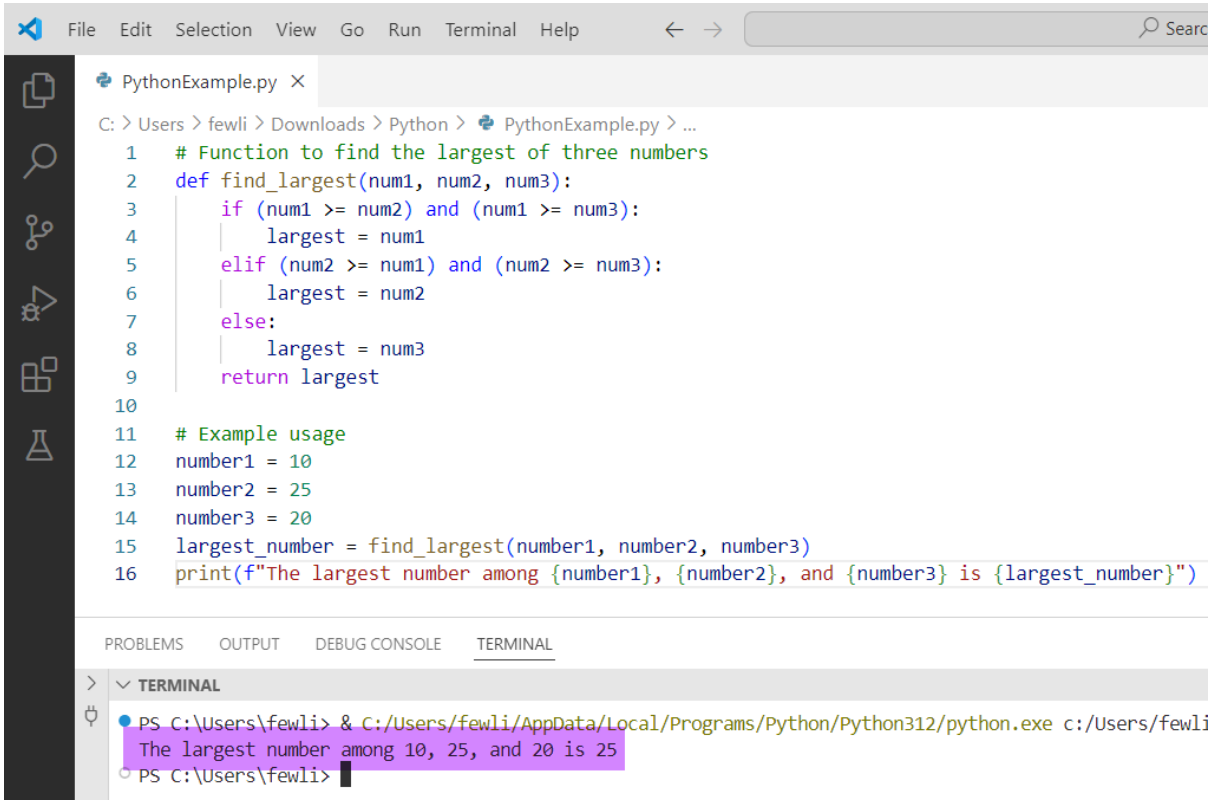
2. Compare the Numbers:
    - First, it checks if num1 is greater than or equal to both num2 and num3. If true, num1 is the largest.
    - If the above condition is false, it checks if num2 is greater than or equal to both num1 and num3. If true, num2 is the largest.
    - If neither of the above conditions is true, num3 is the largest.
3. Return the Largest Number: The function returns the largest number.

Example:

For the numbers 10, 25, and 20:

- The function first checks if 10 is greater than or equal to 25 and 20, which is false.
- Then it checks if 25 is greater than or equal to 10 and 20, which is true.
- Therefore, 25 is determined to be the largest number.

Here is the output you can see:



5# PYTHON PROGRAM TO FIND SUM AND AVERAGE OF THREE NUMBERS

Here is a Python program to find the sum and average of three numbers.

```
# Function to calculate the sum and average of three numbers
def sum_and_average(num1, num2, num3):
    # Calculate the sum
    total_sum = num1 + num2 + num3
    # Calculate the average
    average = total_sum / 3
    return total_sum, average

# Example usage
number1 = 10
number2 = 20
number3 = 30
total_sum, average = sum_and_average(number1, number2, number3)
print(f"The sum of {number1}, {number2}, and {number3} is {total_sum}")
print(f"The average of {number1}, {number2}, and {number3} is {average}")
```

Explanation:

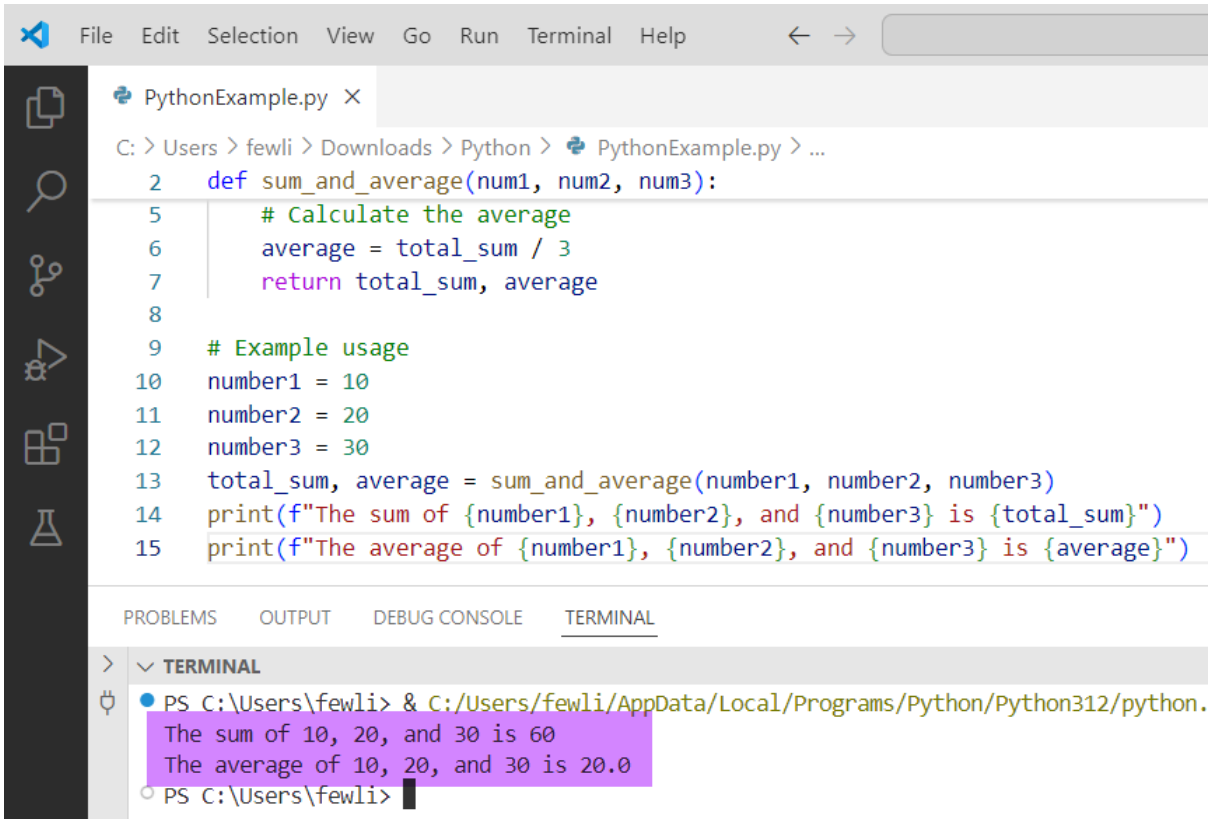
1. Define the Function: The function sum_and_average takes three arguments: num1, num2, and num3.
2. Calculate the Sum: The sum of the three numbers is calculated using the expression num1 + num2 + num3 and stored in the variable total_sum.
3. Calculate the Average: The average is calculated by dividing the total_sum by 3 and stored in the variable average.
4. Return the Results: The function returns both the total_sum and the average.

Example:

For the numbers 10, 20, and 30:

- The sum is calculated as 10 + 20 + 30 = 60.
- The average is calculated as 60 / 3 = 20.
- The function returns 60 as the sum and 20 as the average.

Here is the exact output:



6# PYTHON PROGRAM TO FIND GCD AND LCM OF TWO NUMBERS

Here is a Python program to find the Greatest Common Divisor (GCD) and Least Common Multiple (LCM) of two numbers.

Here is the Python program.

```
import math

# Function to calculate GCD
def calculate_gcd(num1, num2):
    return math.gcd(num1, num2)

# Function to calculate LCM
def calculate_lcm(num1, num2):
    gcd = calculate_gcd(num1, num2)
    return abs(num1 * num2) // gcd

# Example usage
number1 = 12
number2 = 15
gcd = calculate_gcd(number1, number2)
lcm = calculate_lcm(number1, number2)
print(f"The GCD of {number1} and {number2} is {gcd}")
print(f"The LCM of {number1} and {number2} is {lcm}")
```

Explanation:

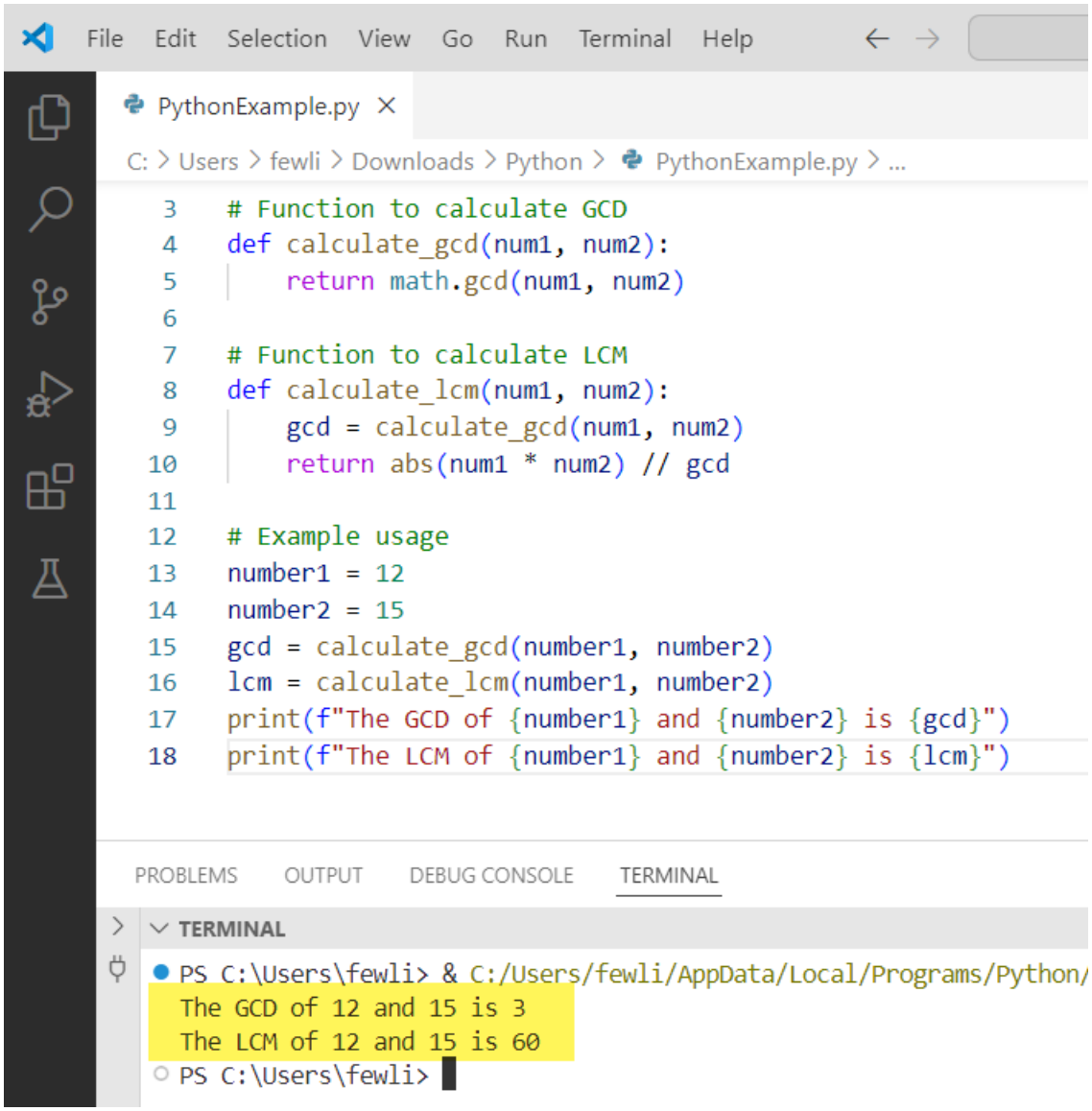
1. Import the math Module: We use Python's built-in math module to calculate the GCD.
2. Calculate GCD:
    - The function calculate_gcd uses math.gcd(num1, num2) to compute the GCD of num1 and num2.
3. Calculate LCM:
    - The function calculate_lcm first calculates the GCD using the calculate_gcd function.
    - The LCM is then calculated using the formula abs(num1 * num2) // gcd. This formula works because the product of the GCD and LCM of two numbers equals the product of the numbers themselves.
4. Example Usage:
    - For the numbers 12 and 15:
        - The GCD is calculated as math.gcd(12, 15), which is 3.
        - The LCM is calculated as abs(12 * 15) // 3, which is 60.

Example:

For the numbers 12 and 15:

- The GCD is 3.
- The LCM is 60.

Here is the exact output you can see in the screenshot below:



7# PYTHON PROGRAM TO FIND HCF AND LCM OF TWO NUMBERS

Here is a Python program to find the Greatest Common Divisor (GCD) and Least Common Multiple (LCM) of two numbers.

```
import math

# Function to calculate HCF (GCD)
def calculate_hcf(num1, num2):
    return math.gcd(num1, num2)

# Function to calculate LCM
def calculate_lcm(num1, num2):
    hcf = calculate_hcf(num1, num2)
    return abs(num1 * num2) // hcf

# Example usage
number1 = 12
```

```
number2 = 15
hcf = calculate_hcf(number1, number2)
lcm = calculate_lcm(number1, number2)
print(f"The HCF of {number1} and {number2} is {hcf}")
print(f"The LCM of {number1} and {number2} is {lcm}")
```

Explanation:

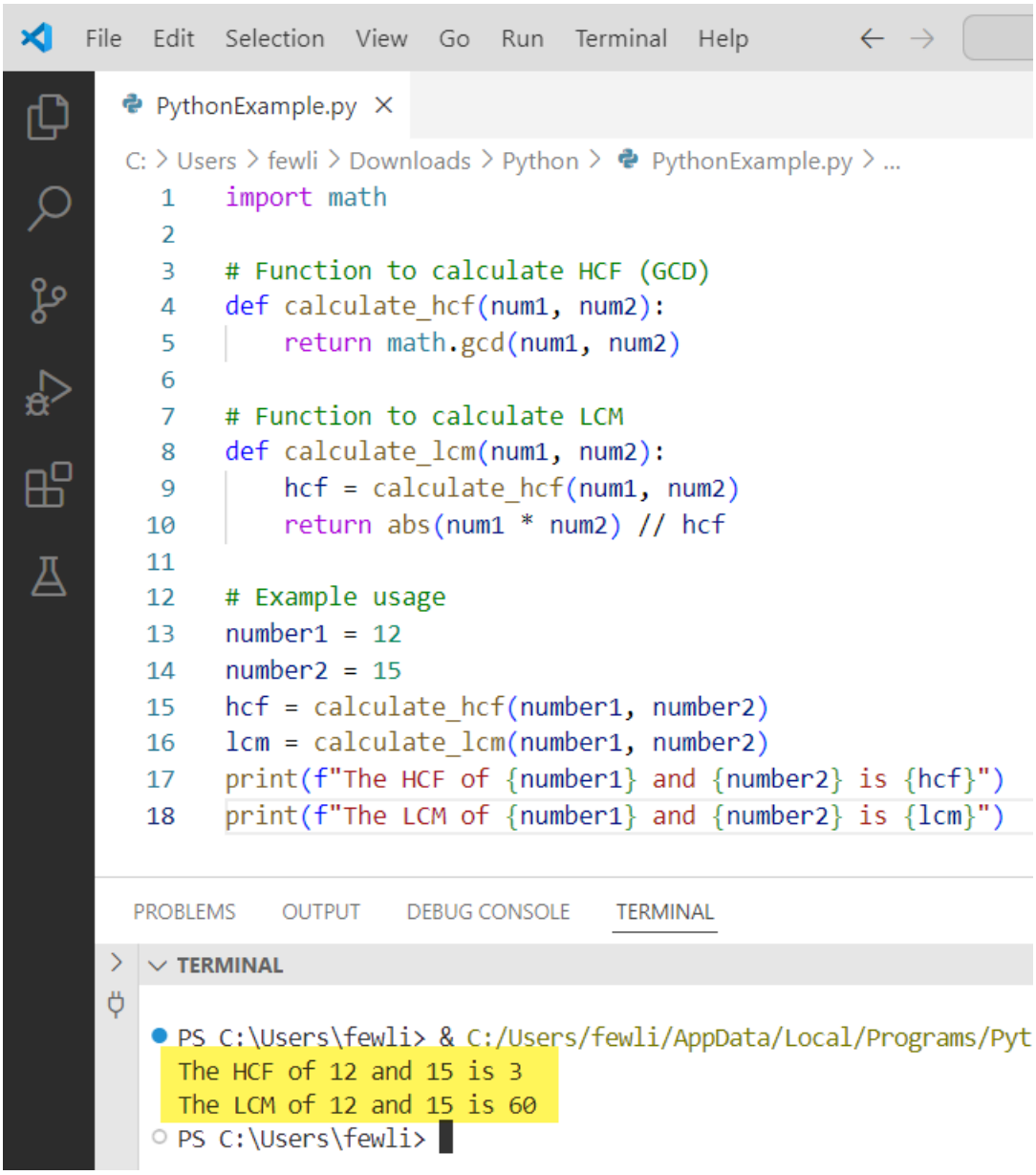
1. Import the math Module: The math module in Python provides a built-in function to calculate the GCD.
2. Calculate HCF (GCD):
    - The function calculate_hcf uses math.gcd(num1, num2) to compute the HCF of num1 and num2.
3. Calculate LCM:
    - The function calculate_lcm first calculates the HCF using the calculate_hcf function.
    - The LCM is then calculated using the formula abs(num1 * num2) // hcf. This formula works because the product of the HCF and LCM of two numbers equals the product of the numbers themselves.
4. Example Usage:
    - For the numbers 12 and 15:
        - The HCF is calculated as math.gcd(12, 15), which is 3.
        - The LCM is calculated as abs(12 * 15) // 3, which is 60.

Example:

For the numbers 12 and 15:

- The HCF is 3.
- The LCM is 60.

Here is the exact output in the screenshot below:



8# PYTHON PROGRAM FOR PALINDROME NUMBER

Here is a Python program to check if a number is a palindrome.

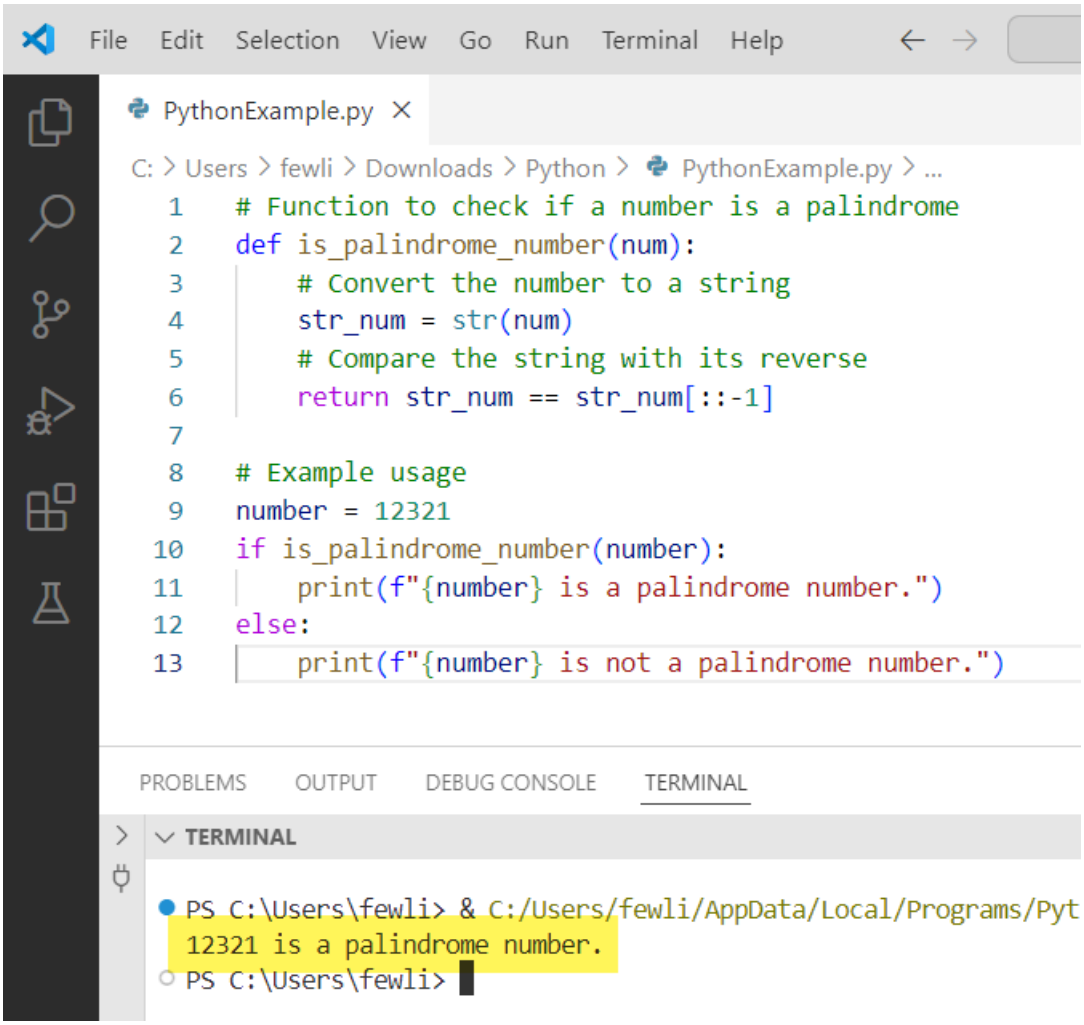
```
# Function to check if a number is a palindrome
def is_palindrome_number(num):
    # Convert the number to a string
    str_num = str(num)
    # Compare the string with its reverse
    return str_num == str_num[::-1]

# Example usage
number = 12321
if is_palindrome_number(number):
    print(f"{number} is a palindrome number.")
else:
```

```
    print(f"{number} is not a palindrome number.")
```

Explanation:

1. Define the Function: The function is_palindrome_number takes a single argument num, which is the number to be checked.
2. Convert Number to String:
    - str_num = str(num): This converts the number to a string. This step is necessary because checking for a palindrome involves comparing the sequence of digits.
3. Check Palindrome:
    - str_num == str_num[::-1]: This compares the string representation of the number with its reverse. The slicing operation str_num[::-1] creates a reversed copy of the string.
    - If the string is equal to its reversed copy, the number is a palindrome, and the function returns True. Otherwise, it returns False.
4. Example Usage:
    - The example number 12321 is converted to the string "12321".
    - The reversed string is also "12321".
    - Since they are equal, the number is identified as a palindrome.

Here is the exact output you can see in the screenshot below:



9# PYTHON PROGRAM FOR FIBONACCI SERIES

Here is a Python program to generate the Fibonacci series up to n terms.

```
# Function to generate Fibonacci series up to n terms
def fibonacci_series(n):
    # Initialize the first two terms of the series
    fib_sequence = [0, 1]
    # Generate the Fibonacci series
    for i in range(2, n):
        next_term = fib_sequence[-1] + fib_sequence[-2]
        fib_sequence.append(next_term)
    return fib_sequence[:n]

# Example usage
n_terms = 10
fib_series = fibonacci_series(n_terms)
print(f"The first {n_terms} terms of the Fibonacci series are: {fib_series}")
```

Explanation:

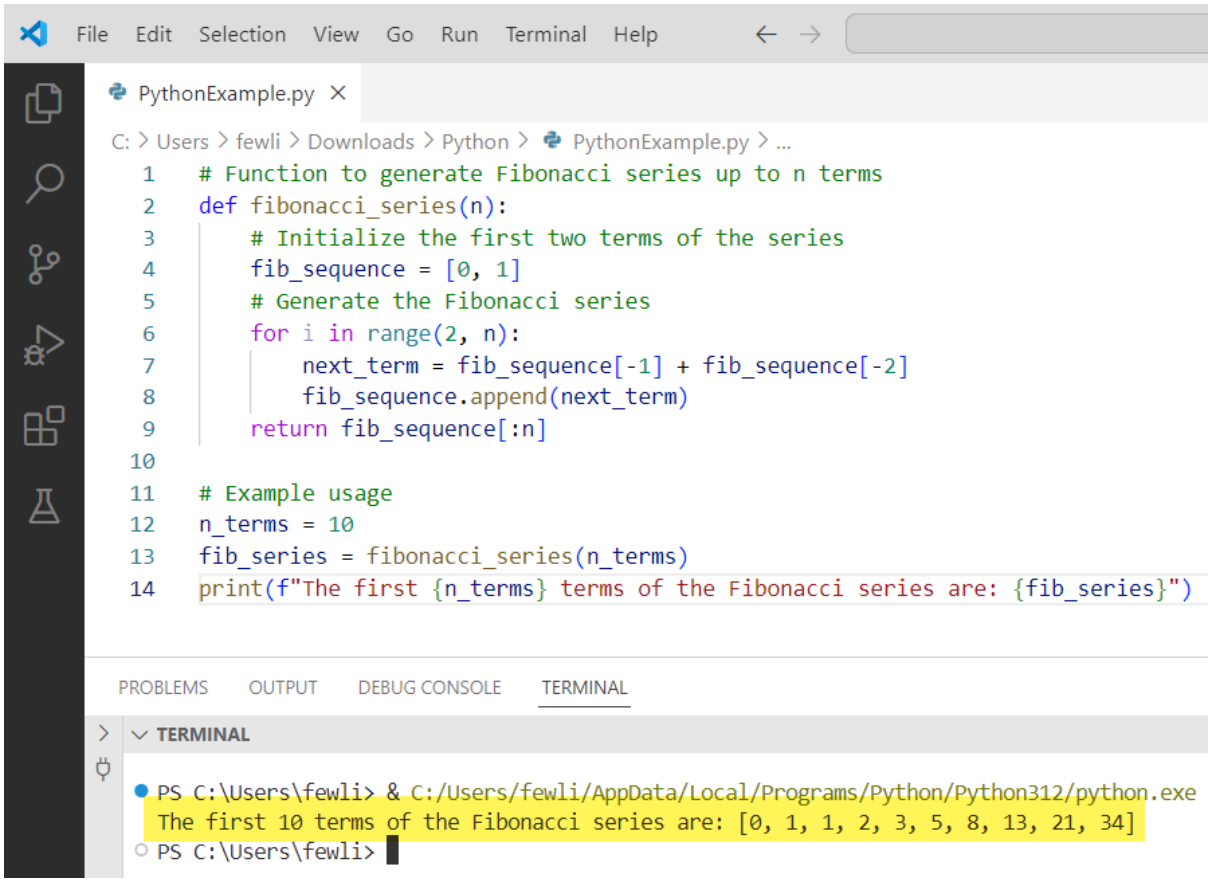
1. Define the Function: The function fibonacci_series takes a single argument n, which specifies the number of terms in the Fibonacci series to be generated.
2. Initialize the First Two Terms:
    - fib_sequence = [0, 1]: This initializes the list with the first two terms of the Fibonacci series, 0 and 1.
3. Generate the Fibonacci Series:
    - The for loop starts from 2 and runs up to n-1, generating the next terms of the series.
    - next_term = fib_sequence[-1] + fib_sequence[-2]: This calculates the next term by adding the last two terms in the current sequence.
    - fib_sequence.append(next_term): This appends the newly calculated term to the sequence.
4. Return the Series:
    - return fib_sequence[:n]: This returns the first n terms of the Fibonacci series. The slicing is necessary in case n is less than 2, ensuring the function returns the correct number of terms.
5. Example Usage:
    - For n_terms = 10, the function generates the first 10 terms of the Fibonacci series.
    - The output will be: [0, 1, 1, 2, 3, 5, 8, 13, 21, 34].

Example:

For n_terms = 10:

- The first 10 terms of the Fibonacci series are [0, 1, 1, 2, 3, 5, 8, 13, 21, 34].

Here is the exact output:



10# PYTHON PROGRAM FOR ARMSTRONG NUMBER

Here is a Python program to check if a number is an Armstrong number.

```
# Function to check if a number is an Armstrong number
def is_armstrong_number(num):
    # Convert the number to a string to easily iterate over each digit
    num_str = str(num)
    # Calculate the number of digits
    num_digits = len(num_str)
    # Initialize the sum to 0
    sum_of_powers = 0
    # Calculate the sum of each digit raised to the power of the number of
digits
    for digit in num_str:
        sum_of_powers += int(digit) ** num_digits
    # Check if the sum of powers is equal to the original number
    return sum_of_powers == num

# Example usage
number = 153
if is_armstrong_number(number):
    print(f"{number} is an Armstrong number.")
else:
    print(f"{number} is not an Armstrong number.")
```

Explanation:

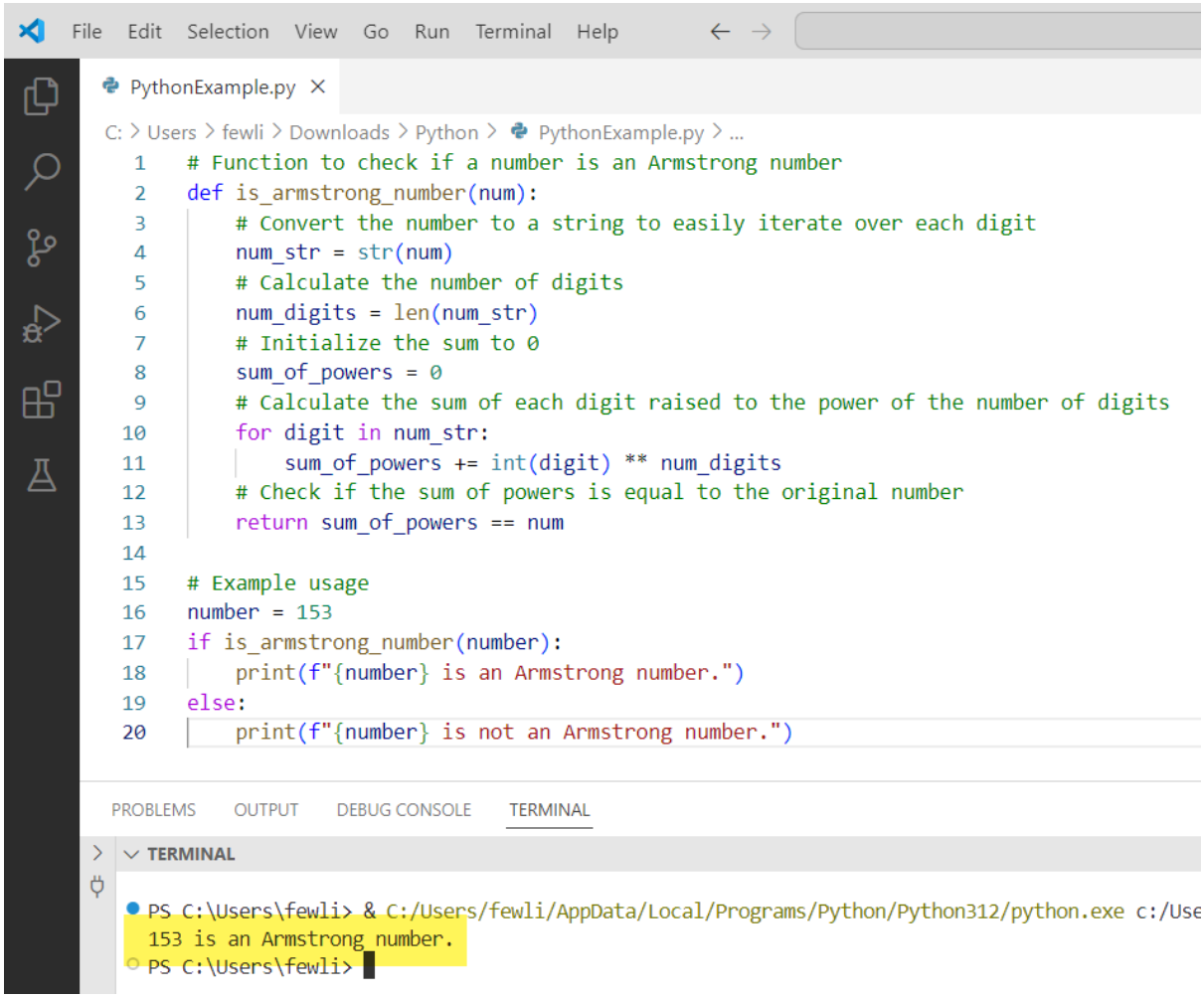
1. Define the Function: The function is_armstrong_number takes a single argument num, which is the number to be checked.
2. Convert the Number to a String:
    - num_str = str(num): This converts the number to a string to easily iterate over each digit.
3. Calculate the Number of Digits:
    - num_digits = len(num_str): This calculates the number of digits in the number.
4. Initialize the Sum:
    - sum_of_powers = 0: This initializes the sum of the digits raised to the power of the number of digits.
5. Calculate the Sum of Powers:
    - The for loop iterates over each digit in the string representation of the number.
    - sum_of_powers += int(digit) ** num_digits: This converts the digit back to an integer, raises it to the power of num_digits, and adds it to sum_of_powers.
6. Check if the Sum Equals the Original Number:
    - return sum_of_powers == num: This checks if the sum of the digits raised to the power of the number of digits is equal to the original number. If true, the number is an Armstrong number.
7. Example Usage:
    - For the number 153:
        - The number of digits is 3.
        - The sum of the digits raised to the power of 3 is (1^3 + 5^3 + 3^3 = 1 + 125 + 27 = 153).
        - Since the sum equals the original number, 153 is an Armstrong number.

Example:

For the number 153:

- The sum of the digits raised to the power of the number of digits is (1^3 + 5^3 + 3^3 = 153).
- Since the sum equals the original number, 153 is an Armstrong number.

Here is the exact output in the screenshot below:



11# PYTHON PROGRAM FOR BINARY SEARCH

Here is a Python program to perform a binary search.

```
# Function to perform binary search
def binary_search(arr, target):
    left, right = 0, len(arr) - 1

    while left <= right:
        mid = (left + right) // 2
```

```
        # Check if target is present at mid
        if arr[mid] == target:
            return mid
        # If target is greater, ignore the left half
        elif arr[mid] < target:
            left = mid + 1
        # If target is smaller, ignore the right half
        else:
            right = mid - 1

    # Target is not present in the array
    return -1

# Example usage
arr = [2, 3, 4, 10, 40]
target = 10

result = binary_search(arr, target)

if result != -1:
    print(f"Element {target} is present at index {result}.")
else:
    print(f"Element {target} is not present in the array.")
```

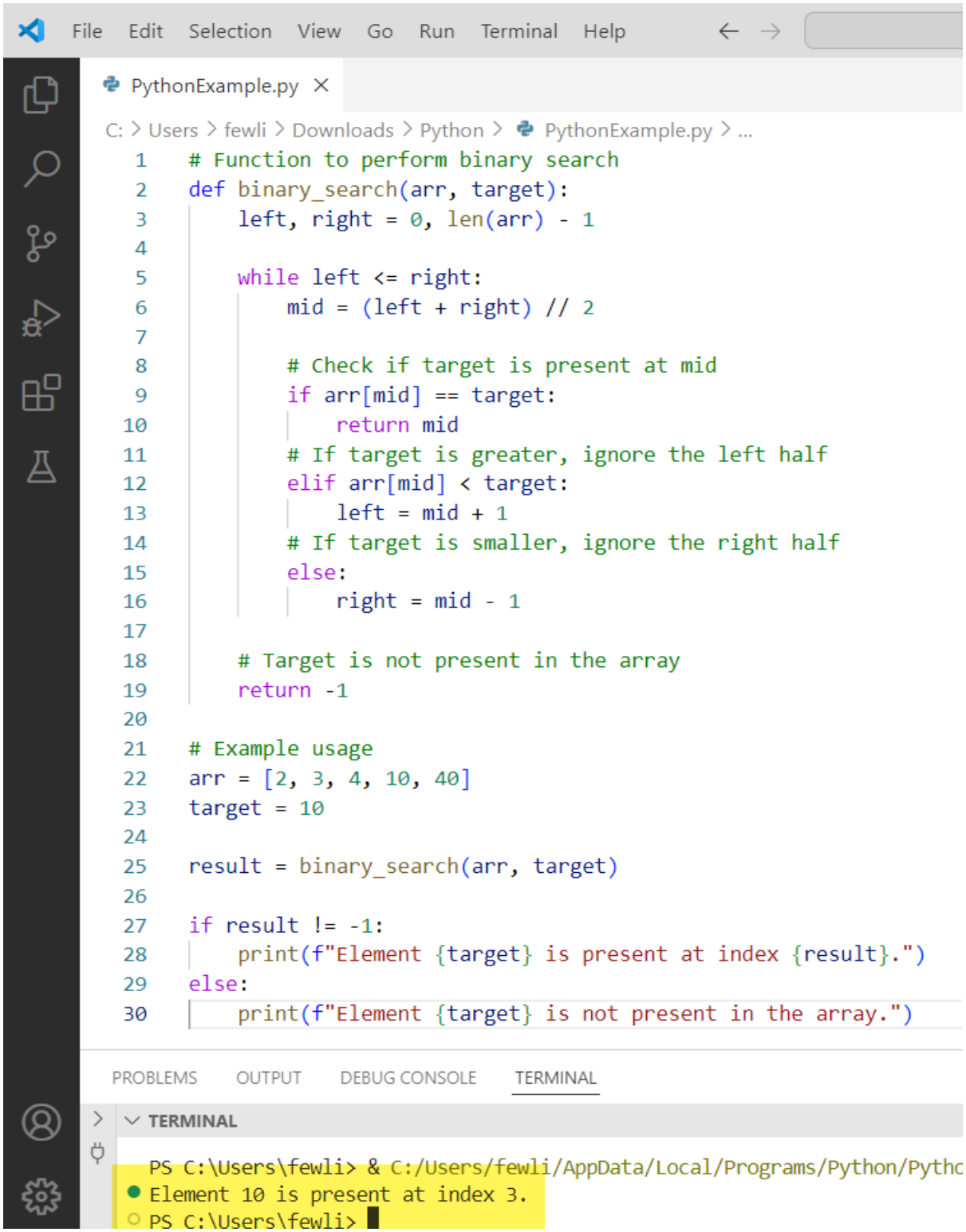
Explanation:

1. Define the Function: The function binary_search takes two arguments: arr (the sorted list of elements) and target (the element to be searched).
2. Initialize Pointers:
    - left is initialized to 0 (the start of the array).
    - right is initialized to len(arr) - 1 (the end of the array).
3. Iterate Until the Search Space is Exhausted:
    - The while loop continues as long as left is less than or equal to right.
4. Calculate the Middle Index:
    - mid = (left + right) // 2: This calculates the middle index of the current search space.
5. Check if the Target is at the Middle:

- If arr[mid] == target, the target is found, and the function returns the index mid.

6. Adjust the Search Space:
    - If arr[mid] < target, the target must be in the right half of the current search space, so left is updated to mid + 1.
    - If arr[mid] > target, the target must be in the left half of the current search space, so right is updated to mid - 1.
7. Return -1 if the Target is Not Found:
    - If the loop exits without finding the target, the function returns -1, indicating that the target is not present in the array.
8. Example Usage:
    - For the sorted array [2, 3, 4, 10, 40] and target 10:
        - The middle index is calculated as (0 + 4) // 2 = 2, and arr[2] is 4.
        - Since 4 < 10, the search continues in the right half (left is updated to 3).
        - The new middle index is (3 + 4) // 2 = 3, and arr[3] is 10, which matches the target.
        - The function returns the index 3.

Here is the exact output:



12# PYTHON PROGRAM FOR BUBBLE SORT

Bubble Sort is a simple sorting algorithm that works by repeatedly stepping through the list to be sorted, comparing each pair of adjacent items, and swapping them if they are in the wrong order. This process is repeated until the list is sorted.

Here is a Python implementation of the Bubble Sort algorithm:

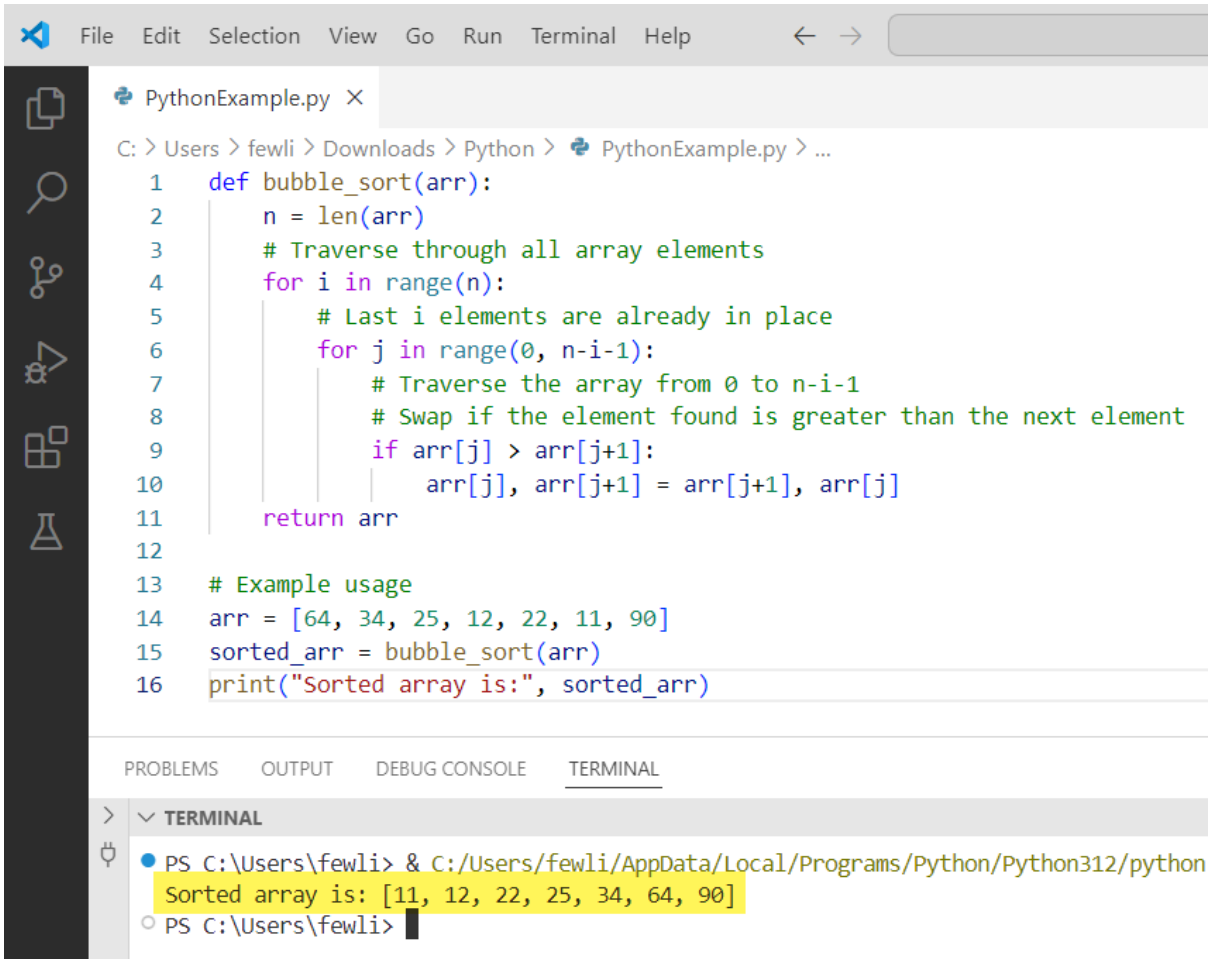
```
def bubble_sort(arr):
    n = len(arr)
    # Traverse through all array elements
    for i in range(n):
        # Last i elements are already in place
        for j in range(0, n-i-1):
            # Traverse the array from 0 to n-i-1
            # Swap if the element found is greater than the next element
            if arr[j] > arr[j+1]:
                arr[j], arr[j+1] = arr[j+1], arr[j]
    return arr

# Example usage
arr = [64, 34, 25, 12, 22, 11, 90]
sorted_arr = bubble_sort(arr)
print("Sorted array is:", sorted_arr)
```

Explanation

1. Outer Loop: The outer loop runs n times, where n is the length of the array. This ensures that all elements are sorted.
2. Inner Loop: The inner loop runs from the start of the array to n-i-1. This is because after each pass, the largest element moves to its correct position, so we can ignore the last i elements in the subsequent passes.
3. Comparison and Swap: During each pass, adjacent elements are compared. If the current element is greater than the next element, they are swapped.

This algorithm has a time complexity of (O(n^2)) in the worst and average cases, making it inefficient for large lists. However, it is easy to understand and implement, which makes it useful for educational purposes.

Here is the exact output:



13# PYTHON PROGRAM TO FIND AREA OF CIRCLE

To find the area of a circle in Python, you can use the formula:

[\text{Area} = \pi \times r^2]

where (r) is the radius of the circle and (\pi) (pi) is a constant approximately equal to 3.14159. Python's math module provides a more precise value for (\pi).

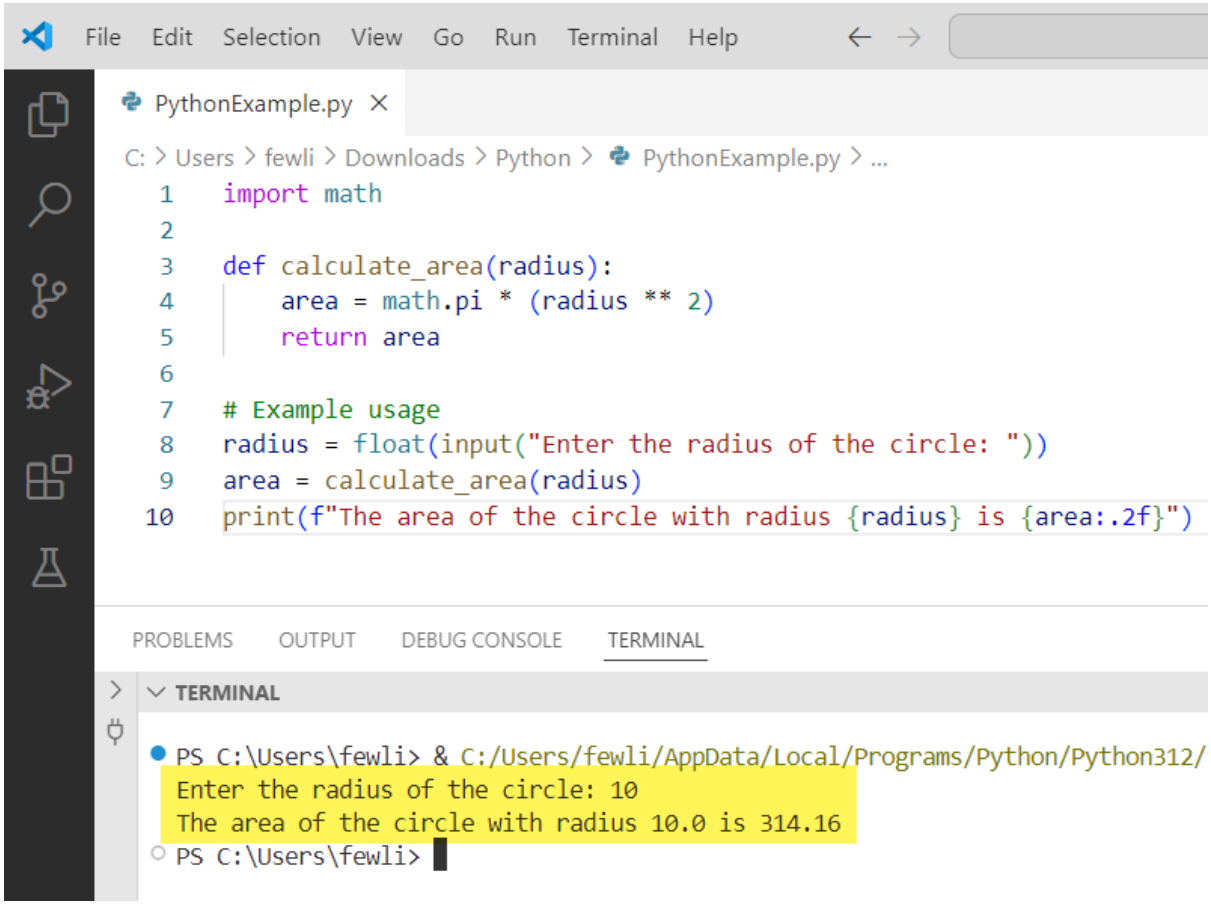
Here is a simple Python program to calculate the area of a circle:

```
import math

def calculate_area(radius):
    area = math.pi * (radius ** 2)
    return area

# Example usage
radius = float(input("Enter the radius of the circle: "))
area = calculate_area(radius)
print(f"The area of the circle with radius {radius} is {area:.2f}")
```

Explanation

1. Import math module: This module provides access to the mathematical constant (\pi) and other mathematical functions.
2. Define calculate_area function: This function takes the radius as an argument, calculates the area using the formula (\pi \times r^2), and returns the result.
3. User Input: The program prompts the user to enter the radius of the circle.
4. Calculate and Print Area: The program calculates the area using the calculate_area function and prints the result formatted to two decimal places.

Here is the exact output:



14# PYTHON PROGRAM TO FIND AREA OF TRIANGLE

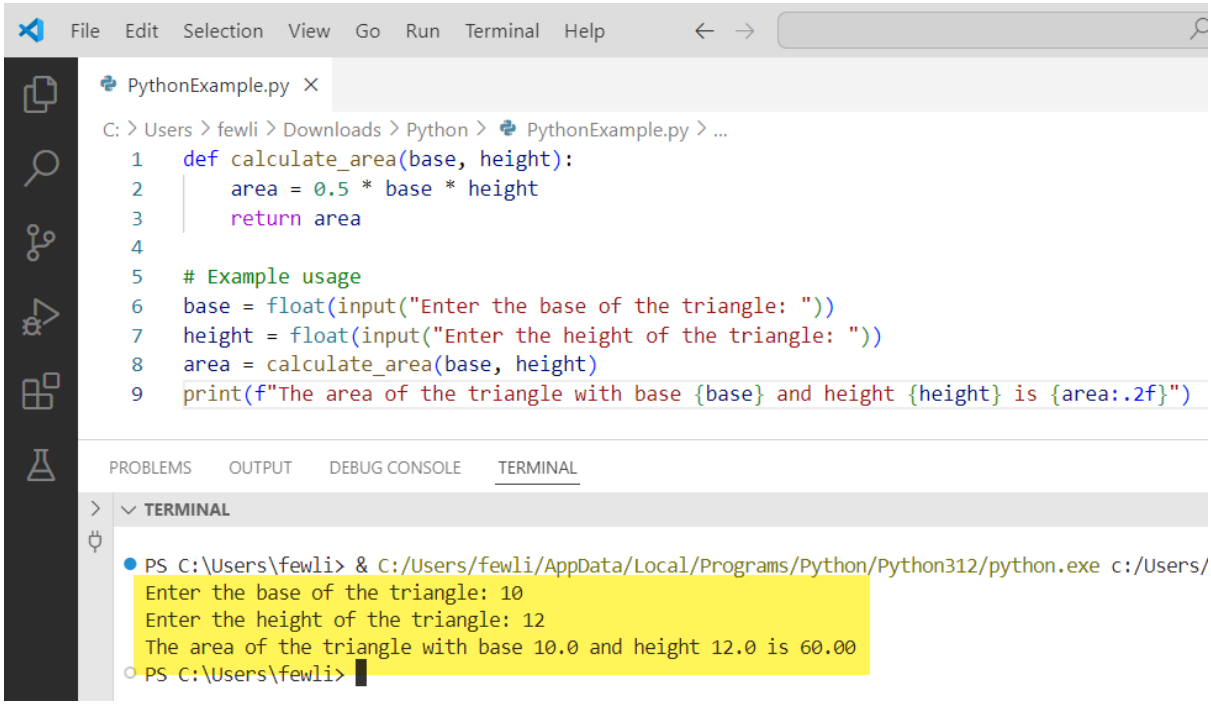
The formula for the area of a triangle when the base and height are known is:

[\text{Area} = \frac{1}{2} \times \text{base} \times \text{height}]

Here's a Python program using this formula:

```
def calculate_area(base, height):
    area = 0.5 * base * height
    return area

# Example usage
base = float(input("Enter the base of the triangle: "))
height = float(input("Enter the height of the triangle: "))
area = calculate_area(base, height)
print(f"The area of the triangle with base {base} and height {height} is
{area:.2f}")
```

Here is the exact output you can see in the screenshot below:



15# PYTHON PROGRAM TO FIND AREA OF SQUARE

To find the area of a square in Python, you can use the simple formula:

[\text{Area} = \text{side}^2]

where "side" is the length of one side of the square.

Here is a Python program to calculate the area of a square:

```
def calculate_area(side):
    area = side ** 2
    return area
```

```python
# Example usage
side = float(input("Enter the length of the side of the square: "))
area = calculate_area(side)
print(f"The area of the square with side length {side} is {area:.2f}")
```

Explanation

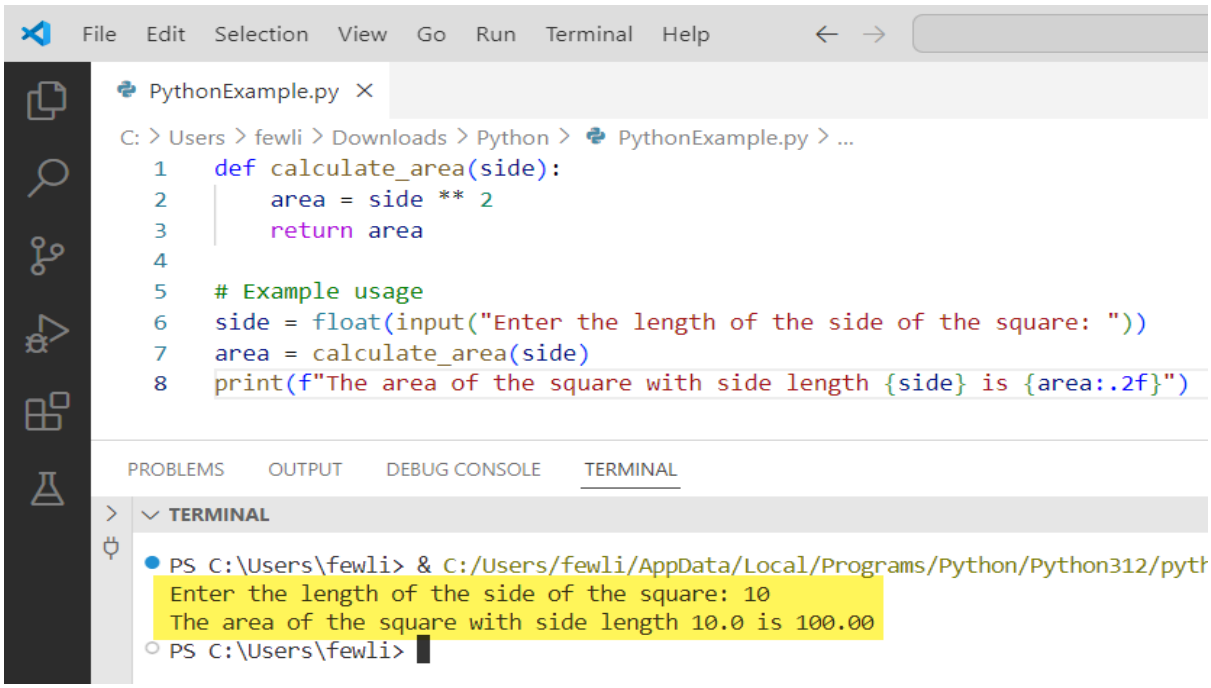
1. Define calculate_area function: This function takes the length of one side of the square as an argument, calculates the area using the formula (\text{side}^2), and returns the result.
2. User Input: The program prompts the user to enter the length of the side of the square.
3. Calculate and Print Area: The program calculates the area using the calculate_area function and prints the result formatted to two decimal places.

Example

Let's say the user inputs a side length of 5 units.

- The program will calculate the area as (5^2 = 25) square units.
- The output will be: "The area of the square with side length 5.0 is 25.00"

Here is the exact output in the screenshot below:



16# PYTHON PROGRAM TO FIND EVEN OR ODD USING FUNCTION

To determine whether a number is even or odd in Python, you can use the modulus operator %. If a number divided by 2 has a remainder of 0, it is even; otherwise, it is odd.

Here is a Python program that uses a function to check if a number is even or odd:

```
def check_even_or_odd(number):
    if number % 2 == 0:
        return "Even"
    else:
        return "Odd"

# Example usage
number = int(input("Enter a number: "))
result = check_even_or_odd(number)
print(f"The number {number} is {result}.")
```

Explanation

1. Define check_even_or_odd function: This function takes a single argument number.
    - It uses the modulus operator % to check if the number is divisible by 2.
    - If number % 2 == 0, it returns "Even".
    - Otherwise, it returns "Odd".
2. User Input: The program prompts the user to enter a number.
    - The input function is used to get the user's input, which is then converted to an integer using int().
3. Check and Print Result: The program calls the check_even_or_odd function with the user's input and prints whether the number is even or odd.

Example

Let's say the user inputs the number 7.

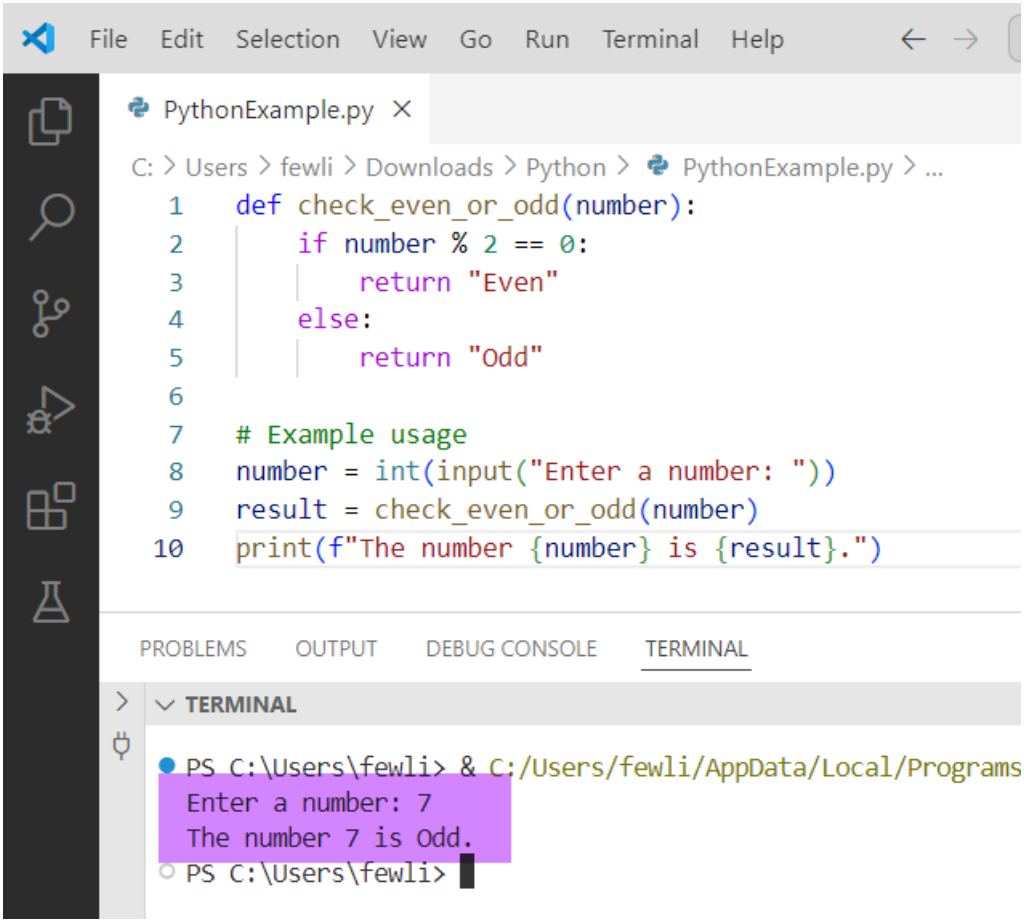
- The program will call check_even_or_odd(7).

- Inside the function, it checks 7 % 2, which equals 1 (not 0), so it returns "Odd".
- The output will be: "The number 7 is Odd."

Similarly, if the user inputs the number 8:

- The program will call check_even_or_odd(8).
- Inside the function, it checks 8 % 2, which equals 0, so it returns "Even".
- The output will be: "The number 8 is Even."

Here is the exact output:



17# PYTHON PROGRAM TO FIND EVEN OR ODD USING FUNCTION

To determine whether a number is even or odd in Python, you can use the modulus operator %. If a number divided by 2 has a remainder of 0, it is even; otherwise, it is odd.

Here is a Python program that uses a function to check if a number is even or odd:

```
def check_even_or_odd(number):
    if number % 2 == 0:
        return "Even"
    else:
        return "Odd"

# Example usage
number = int(input("Enter a number: "))
result = check_even_or_odd(number)
print(f"The number {number} is {result}.")
```

Explanation

1. Define check_even_or_odd function: This function takes a single argument number.
    - It uses the modulus operator % to check if the number is divisible by 2.
    - If number % 2 == 0, it returns "Even".
    - Otherwise, it returns "Odd".
2. User Input: The program prompts the user to enter a number.
    - The input function is used to get the user's input, which is then converted to an integer using int().
3. Check and Print Result: The program calls the check_even_or_odd function with the user's input and prints whether the number is even or odd.

Example

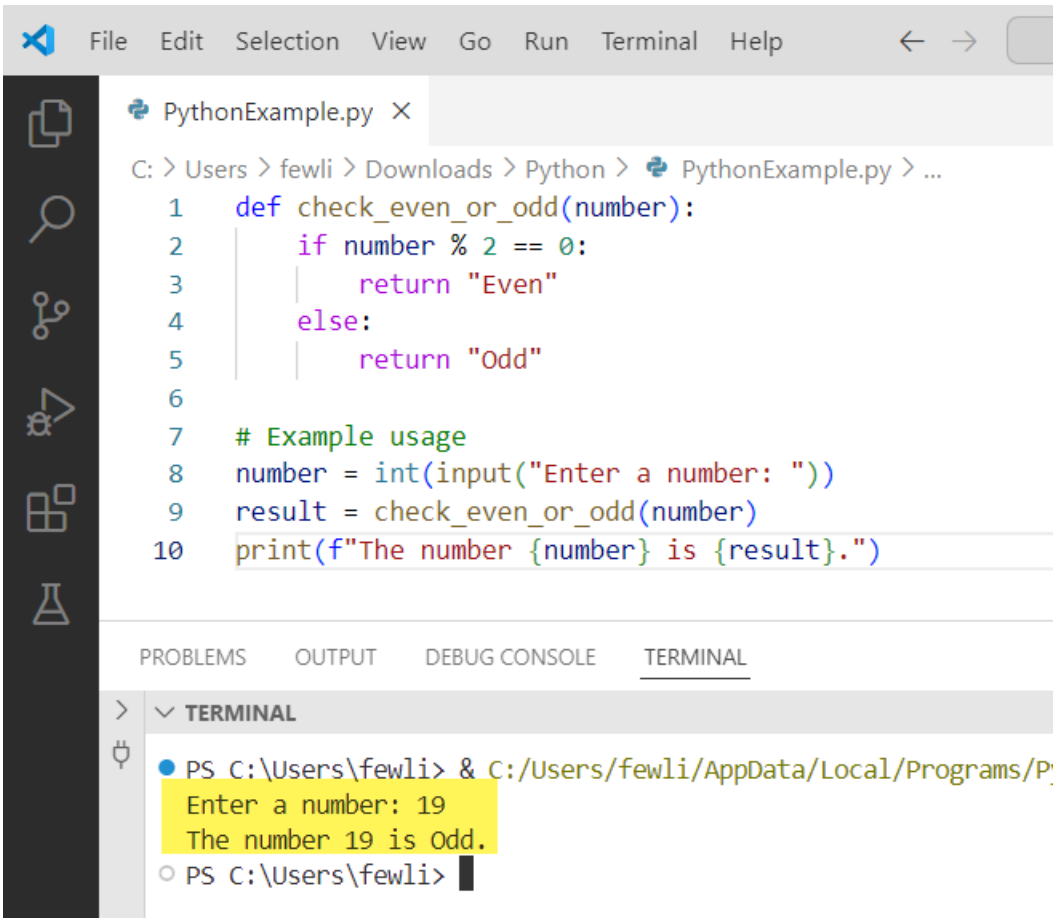
Let's say the user inputs the number 7.

- The program will call check_even_or_odd(7).
- Inside the function, it checks 7 % 2, which equals 1 (not 0), so it returns "Odd".
- The output will be: "The number 7 is Odd."

Similarly, if the user inputs the number 8:

- The program will call check_even_or_odd(8).
- Inside the function, it checks 8 % 2, which equals 0, so it returns "Even".
- The output will be: "The number 8 is Even."

Here is the exact output in the screenshot below:



18# PYTHON PROGRAM TO FIND REPEATED WORDS IN A STRING

To find repeated words in a string in Python, you can use a dictionary to count the occurrences of each word. This method involves splitting the string into words, counting each word's occurrences, and then identifying which words appear more than once.

Here is a Python program to find repeated words in a string:

```
def find_repeated_words(input_string):
    # Split the string into words
    words = input_string.split()

    # Create a dictionary to count occurrences of each word
    word_count = {}
```

```
    for word in words:
        # Convert word to lowercase to make the count case-insensitive
        word = word.lower()
        if word in word_count:
            word_count[word] += 1
        else:
            word_count[word] = 1

    # Find words that are repeated
    repeated_words = {word: count for word, count in word_count.items() if
count > 1}

    return repeated_words

# Example usage
input_string = "This is a test. This test is only a test."
repeated_words = find_repeated_words(input_string)
print("Repeated words and their counts:", repeated_words)
```

Explanation

1. Define find_repeated_words function: This function takes an input_string as an argument.
    - It splits the string into words using the split() method, which by default splits by whitespace.
2. Create a Dictionary to Count Words:
    - It initializes an empty dictionary word_count to store the count of each word.
    - It iterates over each word in the list of words, converts each word to lowercase (to make the counting case-insensitive), and updates the word count in the dictionary.
3. Identify Repeated Words:
    - It uses a dictionary comprehension to create a new dictionary repeated_words that contains only the words that have a count greater than 1.
4. Return Repeated Words: The function returns the repeated_words dictionary.

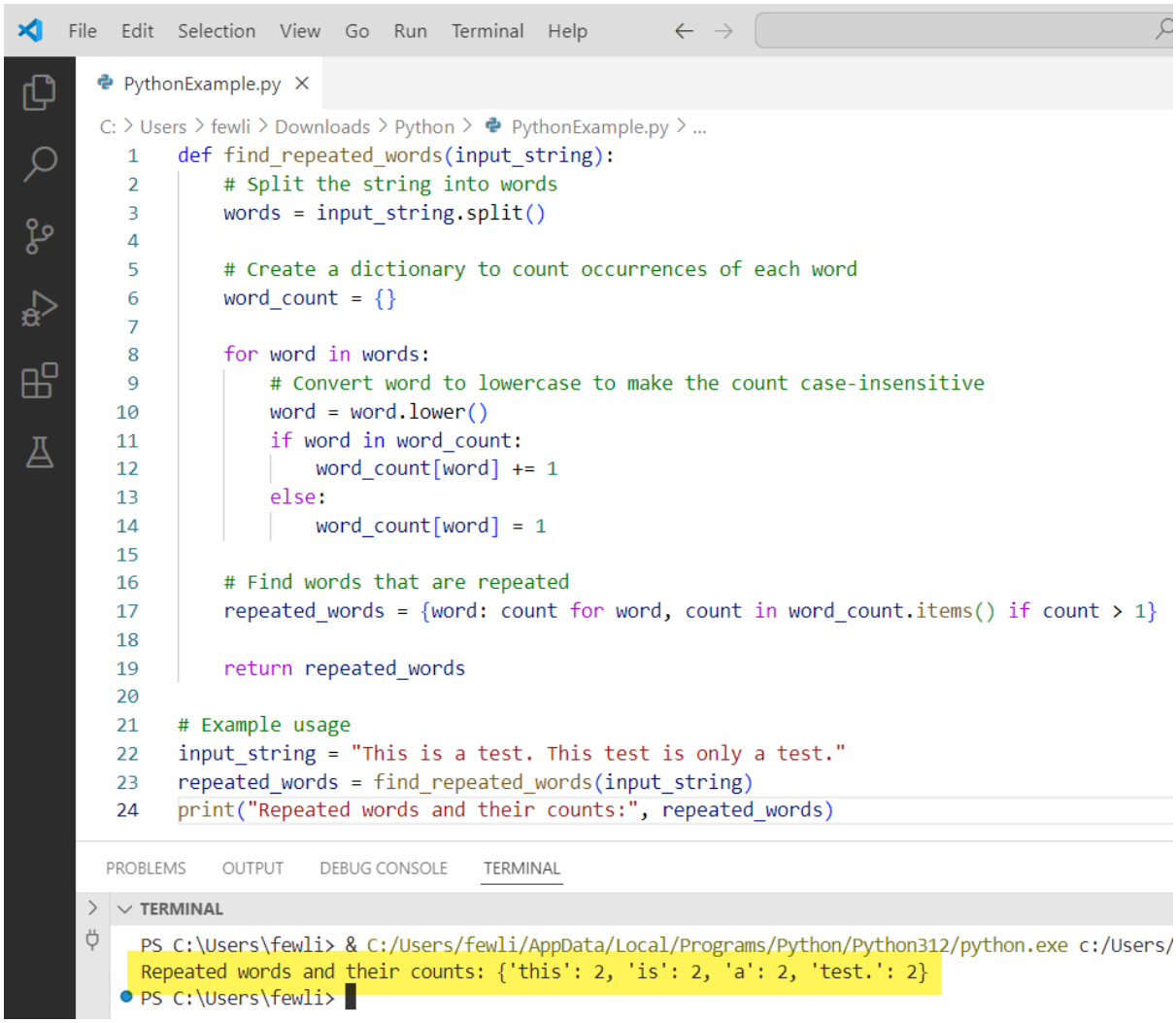
Example

Let's say the input string is:

"This is a test. This test is only a test."

- The program splits the string into words: ["This", "is", "a", "test.", "This", "test", "is", "only", "a", "test."].
- It converts each word to lowercase and counts the occurrences:
  - {'this': 2, 'is': 2, 'a': 2, 'test.': 2, 'test': 1, 'only': 1}
- The program identifies the repeated words: {'this': 2, 'is': 2, 'a': 2, 'test.': 2}.
- The output will be: Repeated words and their counts: {'this': 2, 'is': 2, 'a': 2, 'test.': 2}.

Here is the exact output in the screenshot below:



19# PYTHON PROGRAM FOR NUMBER GUESSING GAME

A number guessing game in Python involves the computer generating a random number within a specified range, and the user attempting to guess that number. The program provides feedback on whether the guess is too high, too low, or correct, and continues until the user guesses the number correctly.

Here is a simple implementation of a number guessing game in Python:

```
import random

def number_guessing_game():
    # Generate a random number between 1 and 100
    number_to_guess = random.randint(1, 100)

    # Initialize the number of attempts
    attempts = 0

    while True:
        # Prompt the user to enter a guess
        guess = int(input("Guess a number between 1 and 100: "))

        # Increment the number of attempts
        attempts += 1

        # Check if the guess is correct
        if guess < number_to_guess:
            print("Too low! Try again.")
        elif guess > number_to_guess:
            print("Too high! Try again.")
        else:
            print(f"Congratulations! You guessed the number in {attempts}
attempts.")
            break

# Start the game
number_guessing_game()
```

Explanation

1. Import random module: This module is used to generate a random number.
2. Define number_guessing_game function: This function contains the logic for the game.

- Generate a Random Number: random.randint(1, 100) generates a random integer between 1 and 100.
- Initialize Attempts: A variable attempts is initialized to count the number of guesses.

3. Game Loop: A while True loop is used to repeatedly prompt the user for a guess until the correct number is guessed.
    - User Input: The user is prompted to enter a guess, which is converted to an integer.
    - Increment Attempts: The number of attempts is incremented each time the user makes a guess.
    - Provide Feedback: The program checks if the guess is too low, too high, or correct:
        - If the guess is less than the number to guess, it prints "Too low! Try again."
        - If the guess is greater than the number to guess, it prints "Too high! Try again."
        - If the guess is correct, it prints a congratulatory message with the number of attempts and breaks out of the loop.

Example

When you run the program, it might look something like this:

Guess a number between 1 and 100: 50

Too high! Try again.

Guess a number between 1 and 100: 25

Too low! Try again.

Guess a number between 1 and 100: 37

Too high! Try again.

Guess a number between 1 and 100: 31

Congratulations! You guessed the number in 4 attempts.

20# PYTHON PROGRAM TO PRINT A STAR PATTERN

Printing star patterns is a common exercise in programming, often used to practice nested loops. There are various star patterns you can print, such as right-angled triangles, pyramids, diamonds, and more. Here, I'll explain how to print a simple right-angled triangle star pattern in Python.

Right-Angled Triangle Star Pattern

A right-angled triangle star pattern looks like this for 5 rows:

```
*

**

***

****

*****
```

Here is a Python program to print this pattern:

```python
def print_right_angled_triangle(rows):
    for i in range(1, rows + 1):
        print('*' * i)

# Example usage
rows = int(input("Enter the number of rows: "))
print_right_angled_triangle(rows)
```

Explanation

1. Define print_right_angled_triangle Function:
    - This function takes one argument, rows, which specifies the number of rows in the triangle.
    - It uses a for loop to iterate from 1 to rows (inclusive).
    - In each iteration, it prints a string consisting of i asterisks ('*' * i).
2. User Input:
    - The program prompts the user to enter the number of rows for the triangle.

- The input is read as a string and then converted to an integer using int().

3. Print the Pattern:
    - The program calls the print_right_angled_triangle function with the user-provided number of rows.
    - The function prints the star pattern.

Example Output

If the user inputs 5, the output will be:

*

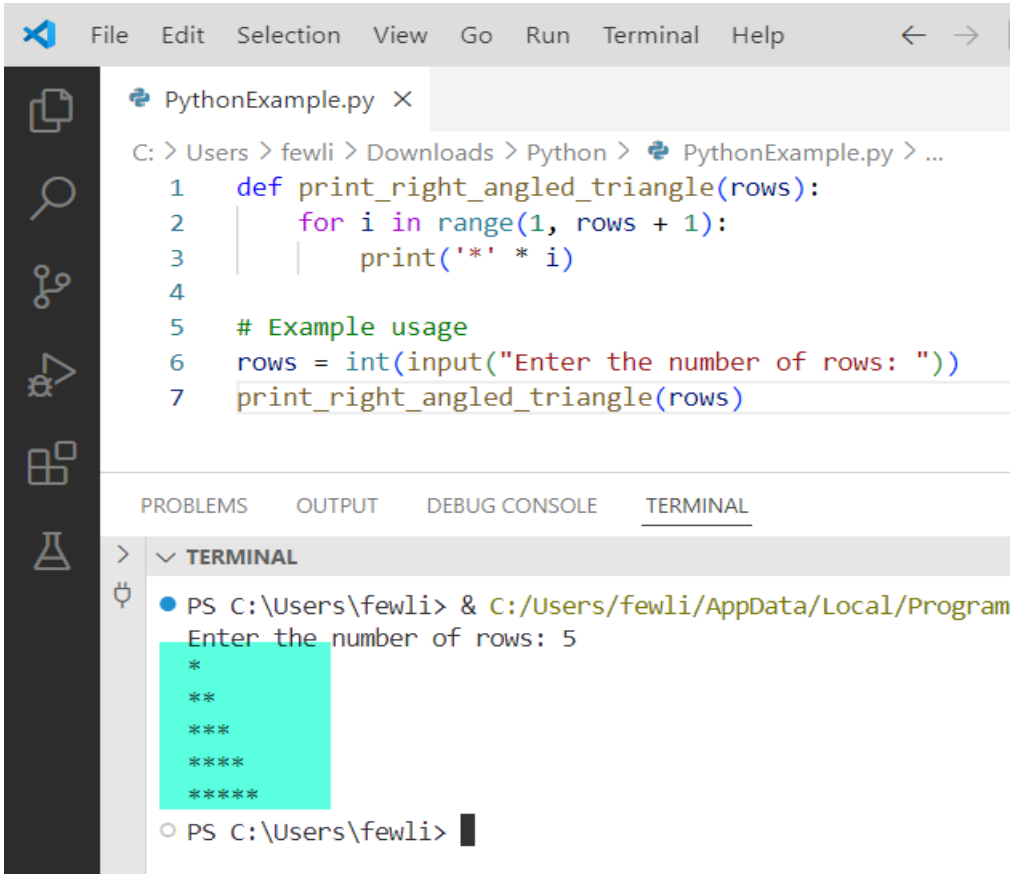
**

***

****

*****

Here is the exact output:



Pyramid Star Pattern

A pyramid star pattern looks like this for 5 rows:

```
  *
 ***
*****
*******
*********
```

Here is a Python program to print this pattern:

```
def print_pyramid(rows):
    for i in range(1, rows + 1):
        # Print leading spaces
        print(' ' * (rows - i), end='')
        # Print stars
        print('*' * (2 * i - 1))

# Example usage
rows = int(input("Enter the number of rows: "))
print_pyramid(rows)
```

Explanation

1. Define print_pyramid Function:
    - This function takes one argument, rows, which specifies the number of rows in the pyramid.
    - It uses a for loop to iterate from 1 to rows (inclusive).
    - In each iteration, it prints leading spaces (' ' * (rows - i)) to center the stars.
    - It then prints the stars ('*' * (2 * i - 1)), where the number of stars in each row is 2 * i - 1.
2. User Input:
    - The program prompts the user to enter the number of rows for the pyramid.
    - The input is read as a string and then converted to an integer using int().

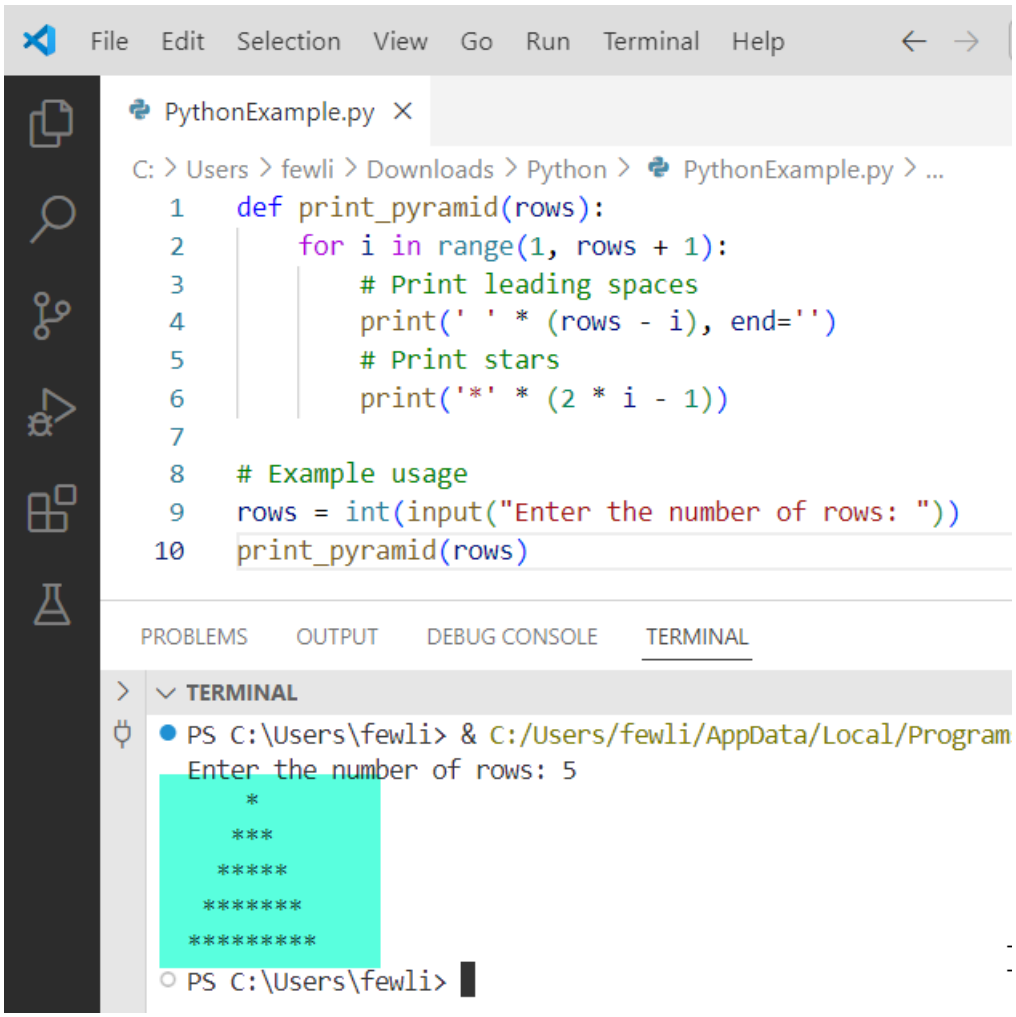
3. Print the Pattern:
    - The program calls the print_pyramid function with the user-provided number of rows.
    - The function prints the star pattern.

Example Output

If the user inputs 5, the output will be:

```
    *
   ***
  *****
 *******
*********
```

Here is the exact output in the screenshot below:



21# PYTHON PROGRAM TO PRINT DIAMOND PATTERN

A diamond star pattern looks like this for 5 rows (9 total rows):

```
  *
 ***
*****
*******
*********
*******
 *****
  ***
   *
```

Here is a Python program to print this pattern:

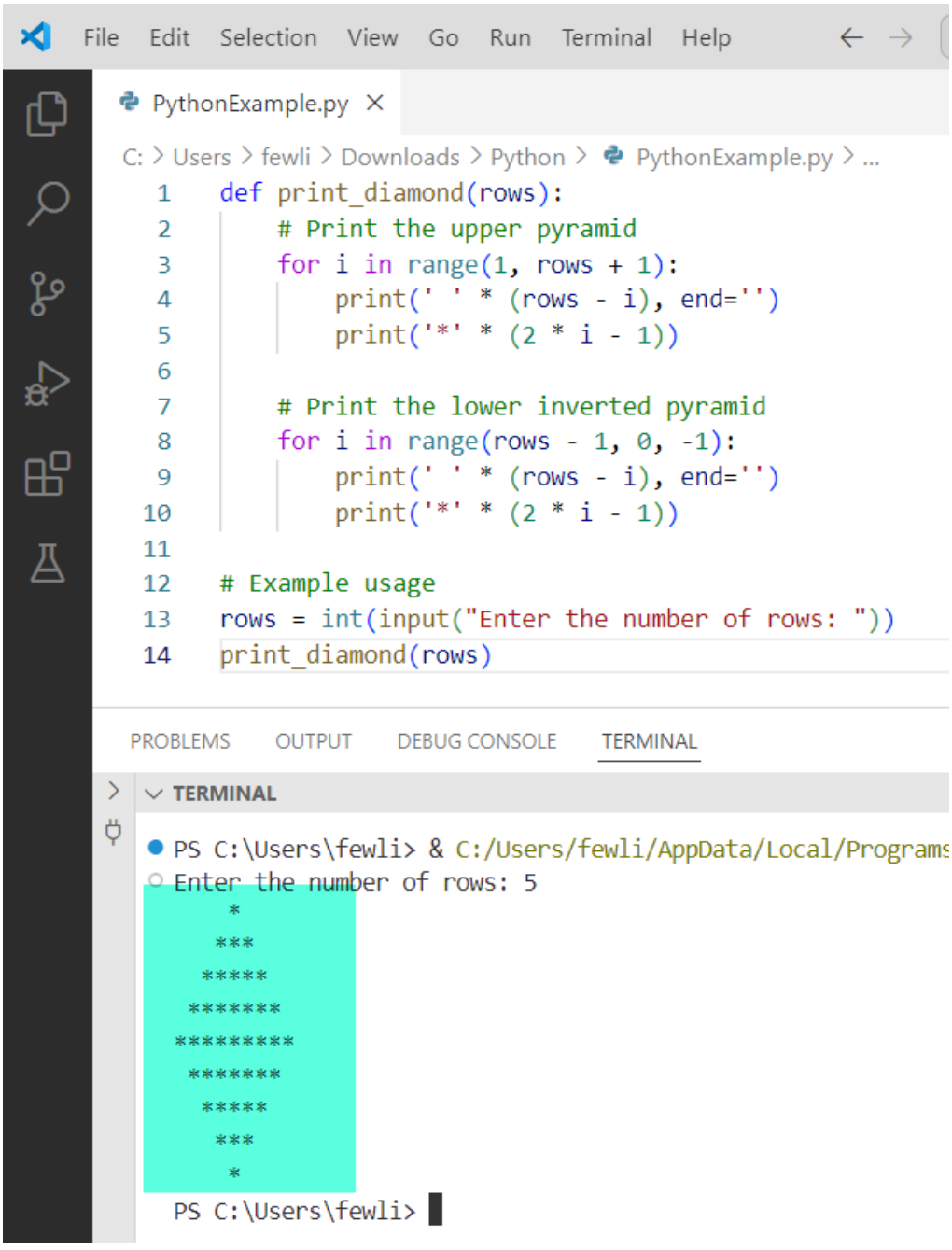
Explanation

1. Define print_diamond Function:
    - This function takes one argument, rows, which specifies the number of rows in the upper half of the diamond.
    - It first prints the upper pyramid using a for loop from 1 to rows (inclusive).
    - It then prints the lower inverted pyramid using a for loop from rows - 1 to 1 (inclusive, in reverse order).
2. User Input:
    - The program prompts the user to enter the number of rows for the diamond.
    - The input is read as a string and then converted to an integer using int().
3. Print the Pattern:
    - The program calls the print_diamond function with the user-provided number of rows.

- The function prints the star pattern.

Example Output

If the user inputs 5, the output will be:



22# PYTHON PROGRAM TO PRINT MULTIPLICATION TABLE

Printing a multiplication table in Python can be done using nested loops. Here is a Python program to print the multiplication table for a given number:

```
def print_multiplication_table(number, up_to=10):
```

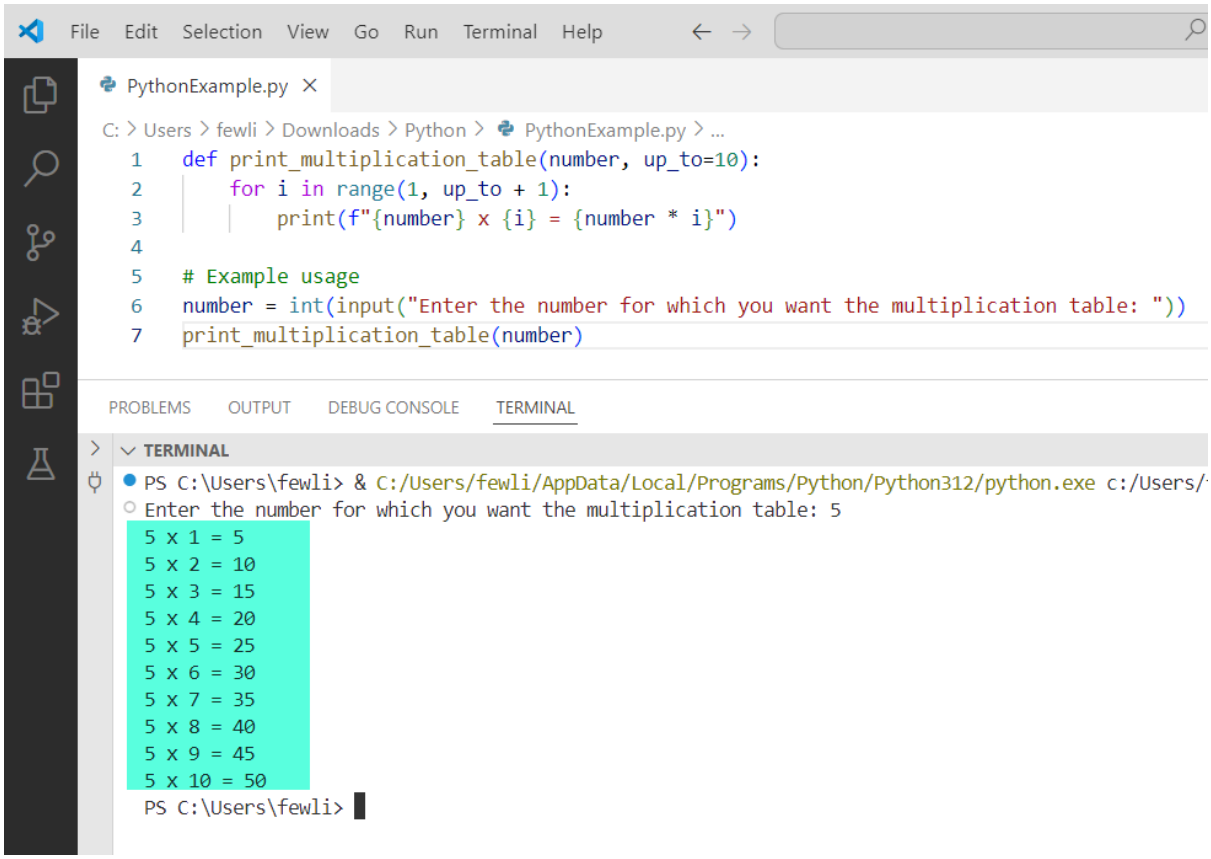
```
    for i in range(1, up_to + 1):
        print(f"{number} x {i} = {number * i}")

# Example usage
number = int(input("Enter the number for which you want the multiplication
table: "))
print_multiplication_table(number)
```

Explanation

1. Define print_multiplication_table Function:
    - This function takes two arguments: number (the number for which the multiplication table is to be printed) and up_to (the range up to which the table should be printed, default is 10).
    - It uses a for loop to iterate from 1 to up_to (inclusive).
    - In each iteration, it calculates the product of number and i, and prints it in the format number x i = product.
2. User Input:
    - The program prompts the user to enter the number for which they want the multiplication table.
    - The input is read as a string and then converted to an integer using int().
3. Print the Multiplication Table:
    - The program calls the print_multiplication_table function with the user-provided number.
    - The function prints the multiplication table for the specified number.

Example Output

If the user inputs 5, the output will be:



23# PYTHON PROGRAM FOR MULTIPLICATION TABLES FOR NUMBERS 1 THROUGH 10

Here is a Python program to print the multiplication tables for numbers 1 through 10:

```
def print_all_multiplication_tables(up_to=10):
    for number in range(1, up_to + 1):
        print(f"Multiplication Table for {number}")
        for i in range(1, up_to + 1):
            print(f"{number} x {i} = {number * i}")
        print()  # Print a blank line between tables

# Example usage
print_all_multiplication_tables()
```

Explanation

1. Define print_all_multiplication_tables Function:
    - This function takes one optional argument up_to (the range up to which the tables should be printed, default is 10).
    - It uses a nested for loop:

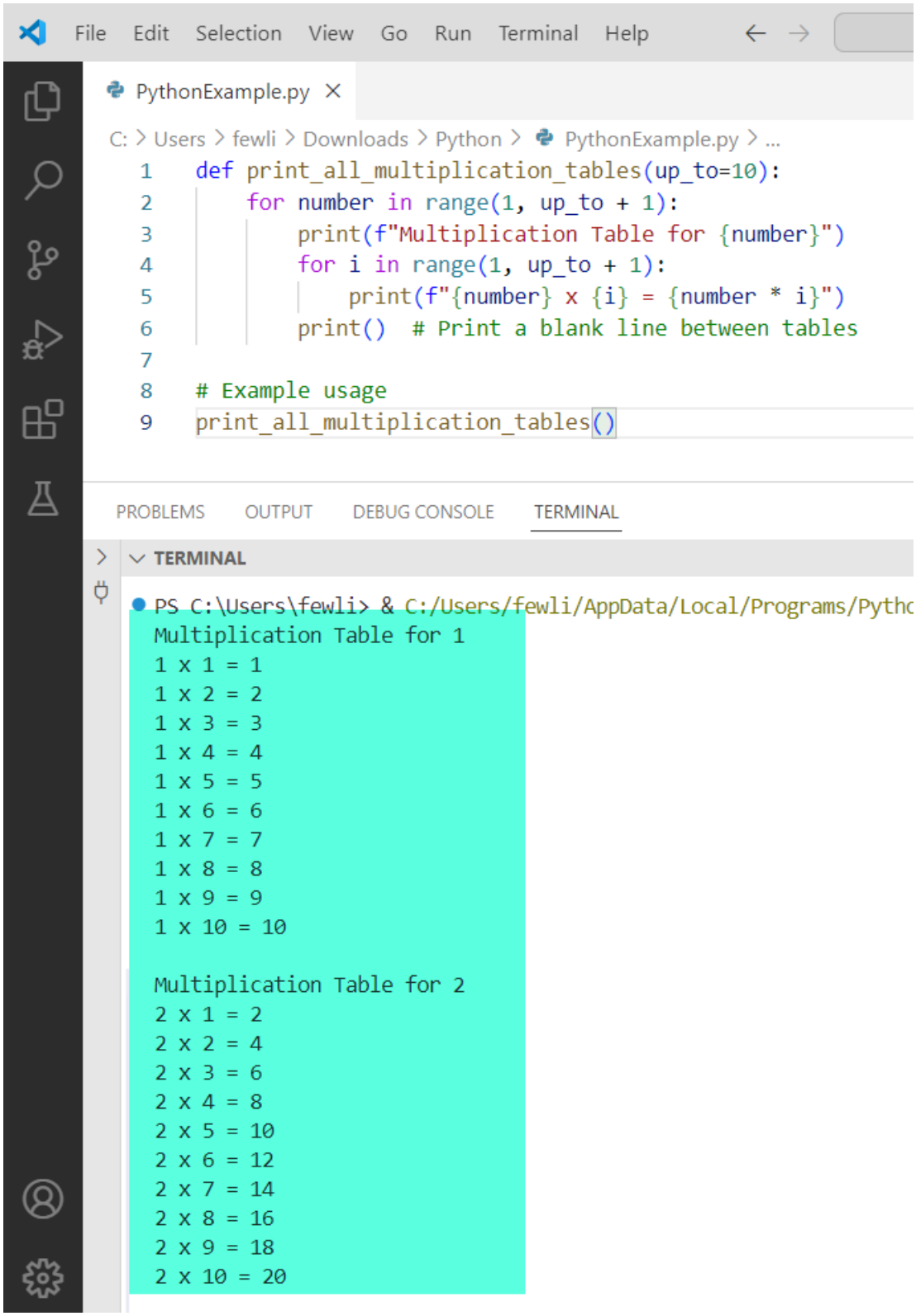
- The outer loop iterates over each number from 1 to up_to (inclusive).
- The inner loop iterates from 1 to up_to (inclusive) for each number.
- In each iteration of the inner loop, it calculates the product of number and i, and prints it in the format number x i = product.

- After each table, it prints a blank line for better readability.

2. Print the Multiplication Tables:
    - The program calls the print_all_multiplication_tables function.
    - The function prints the multiplication tables for numbers 1 through 10.

Example Output

The output will be:



24# PYTHON PROGRAM FOR COMPOUND INTEREST

To calculate compound interest in Python, you need to use the compound interest formula. The formula for compound interest is:

[A = P \left(1 + \frac{r}{n}\right)^{nt}]

Where:

- (A) is the amount of money accumulated after n years, including interest.
- (P) is the principal amount (the initial amount of money).
- (r) is the annual interest rate (in decimal form).
- (n) is the number of times that interest is compounded per year.
- (t) is the time the money is invested for, in years.

The compound interest (CI) can be calculated as:

[CI = A - P]

Here is a Python program to calculate compound interest:

```
def calculate_compound_interest(principal, rate, times_compounded, years):
    # Calculate the amount after the given number of years
    amount = principal * (1 + rate / times_compounded) ** (times_compounded *
years)

    # Calculate the compound interest
    compound_interest = amount - principal

    return amount, compound_interest

# Example usage
principal = float(input("Enter the principal amount: "))
rate = float(input("Enter the annual interest rate (as a percentage): ")) /
100
times_compounded = int(input("Enter the number of times interest is compounded
per year: "))
years = float(input("Enter the number of years the money is invested for: "))

amount, compound_interest = calculate_compound_interest(principal, rate,
times_compounded, years)
print(f"The amount after {years} years is: {amount:.2f}")
print(f"The compound interest is: {compound_interest:.2f}")
```

Explanation

1. Define calculate_compound_interest Function:
    - This function takes four arguments:
        - principal: The initial amount of money (P).

    - rate: The annual interest rate as a decimal (r).
    - times_compounded: The number of times interest is compounded per year (n).
    - years: The time the money is invested for, in years (t).
  - It calculates the amount after the given number of years using the compound interest formula: [A = P \left(1 + \frac{r}{n}\right)^{nt}]
  - It calculates the compound interest by subtracting the principal from the amount: [CI = A - P]
  - It returns both the amount and the compound interest.
2. User Input:
  - The program prompts the user to enter the principal amount.
  - The input is read as a string and then converted to a floating-point number using float().
  - The program prompts the user to enter the annual interest rate as a percentage.
  - The input is read as a string, converted to a floating-point number, and then divided by 100 to convert it to a decimal.
  - The program prompts the user to enter the number of times interest is compounded per year.
  - The input is read as a string and then converted to an integer using int().
  - The program prompts the user to enter the number of years the money is invested for.
  - The input is read as a string and then converted to a floating-point number using float().
3. Calculate and Print:
  - The program calls the calculate_compound_interest function with the user-provided values.

- It prints the amount after the given number of years, formatted to two decimal places.
- It prints the compound interest, formatted to two decimal places.

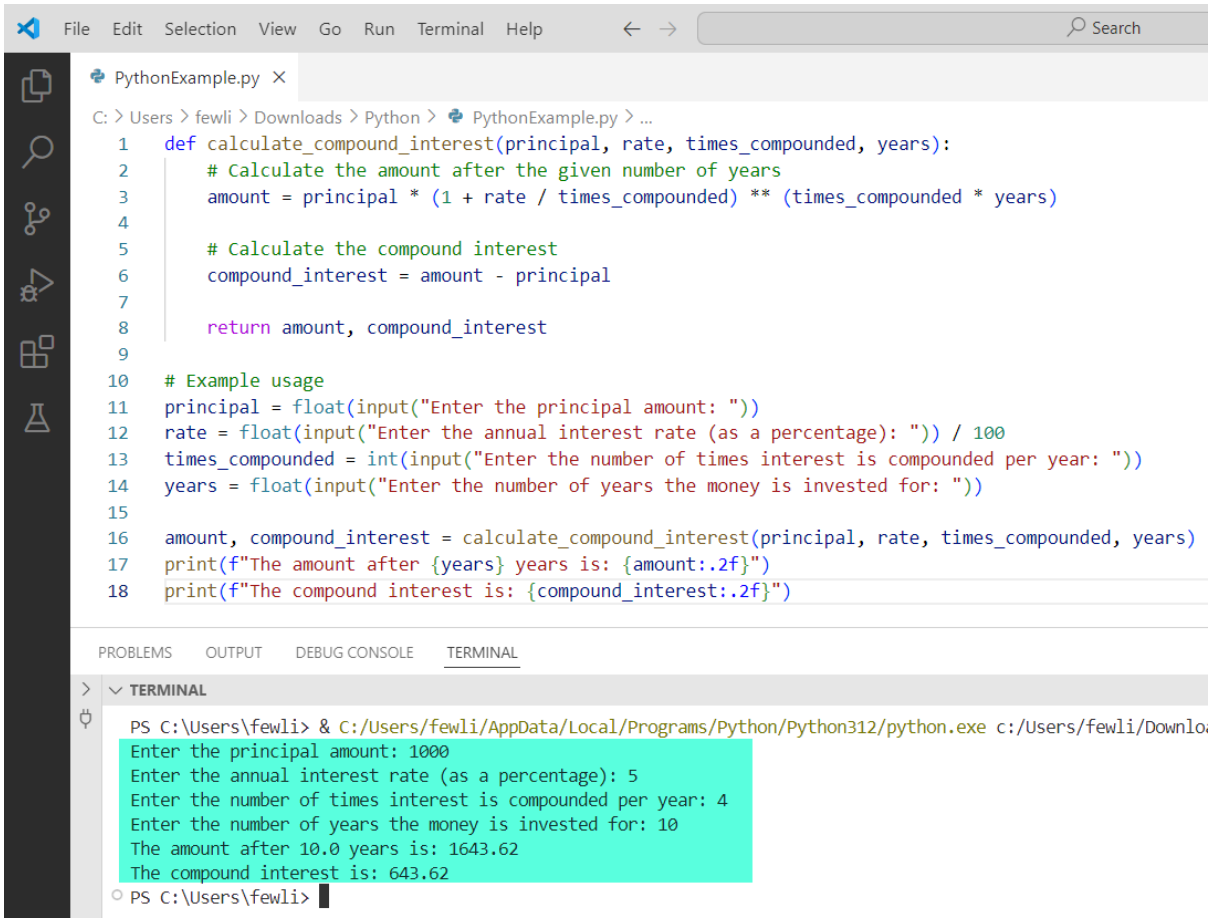
Example

Let's say the user inputs the following values:

- Principal amount: $1000
- Annual interest rate: 5%
- Times compounded per year: 4
- Number of years: 10
- The program will call calculate_compound_interest(1000, 0.05, 4, 10).
- Inside the function:
  - It calculates the amount: [A = 1000 \left(1 + \frac{0.05}{4}\right)^{4 \times 10} = 1000 \left(1 + 0.0125\right)^{40} \approx 1647.01]
  - It calculates the compound interest: [CI = 1647.01 - 1000 = 647.01]
- The output will be:
- The amount after 10.0 years is: 1647.01

The compound interest is: 647.01

Here is the exact output:



25# PYTHON PROGRAM TO CONVERT CELSIUS TO FAHRENHEIT

To convert a temperature from Celsius to Fahrenheit, you can use the following formula:

[\text{Fahrenheit} = (\text{Celsius} \times \frac{9}{5}) + 32]

Here is a Python program to perform this conversion:

```
def celsius_to_fahrenheit(celsius):
    fahrenheit = (celsius * 9/5) + 32
    return fahrenheit

# Example usage
celsius = float(input("Enter temperature in Celsius: "))
fahrenheit = celsius_to_fahrenheit(celsius)
print(f"{celsius} degrees Celsius is equal to {fahrenheit:.2f} degrees
Fahrenheit.")
```

Explanation

1. Define celsius_to_fahrenheit Function:
    - This function takes a temperature in Celsius as an argument.

- It applies the conversion formula ((\text{Celsius} \times \frac{9}{5}) + 32) to calculate the equivalent temperature in Fahrenheit.
- It returns the calculated Fahrenheit temperature.

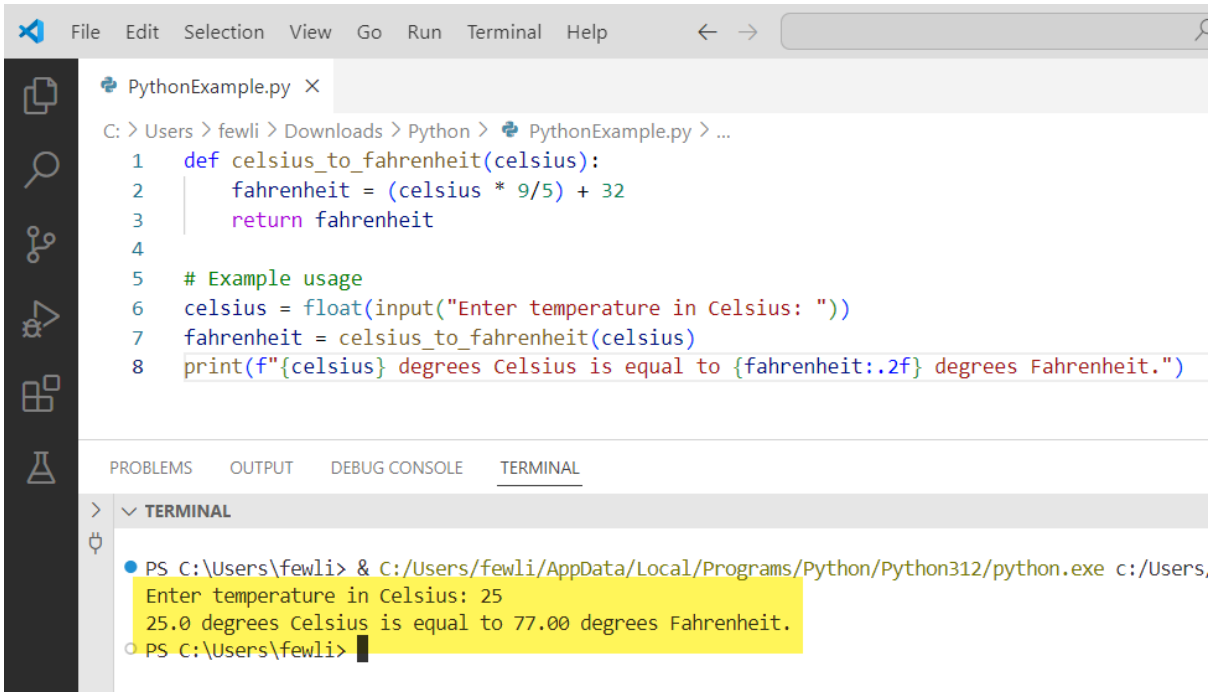
2. User Input:
    - The program prompts the user to enter a temperature in Celsius.
    - The input is read as a string and then converted to a floating-point number using float().
3. Convert and Print:
    - The program calls the celsius_to_fahrenheit function with the user-provided Celsius temperature.
    - It prints the result, formatted to two decimal places.

Example

Let's say the user inputs a temperature of 25 degrees Celsius.

- The program will call celsius_to_fahrenheit(25).
- Inside the function, it calculates ((25 \times \frac{9}{5}) + 32 = 77) degrees Fahrenheit.
- The output will be: "25.0 degrees Celsius is equal to 77.00 degrees Fahrenheit."

Here is the exact output you can see in the screenshot below:



26# PYTHON PROGRAM TO CONVERT FAHRENHEIT INTO CELSIUS

To convert a temperature from Fahrenheit to Celsius, you can use the following formula:

[\text{Celsius} = (\text{Fahrenheit} - 32) \times \frac{5}{9}]

Here is a Python program to perform this conversion:

```
def fahrenheit_to_celsius(fahrenheit):
    celsius = (fahrenheit - 32) * 5/9
    return celsius

# Example usage
fahrenheit = float(input("Enter temperature in Fahrenheit: "))
celsius = fahrenheit_to_celsius(fahrenheit)
print(f"{fahrenheit} degrees Fahrenheit is equal to {celsius:.2f} degrees
Celsius.")
```

Explanation

1. Define fahrenheit_to_celsius Function:
    - This function takes a temperature in Fahrenheit as an argument.
    - It applies the conversion formula ((\text{Fahrenheit} - 32) \times \frac{5}{9}) to calculate the equivalent temperature in Celsius.

- It returns the calculated Celsius temperature.

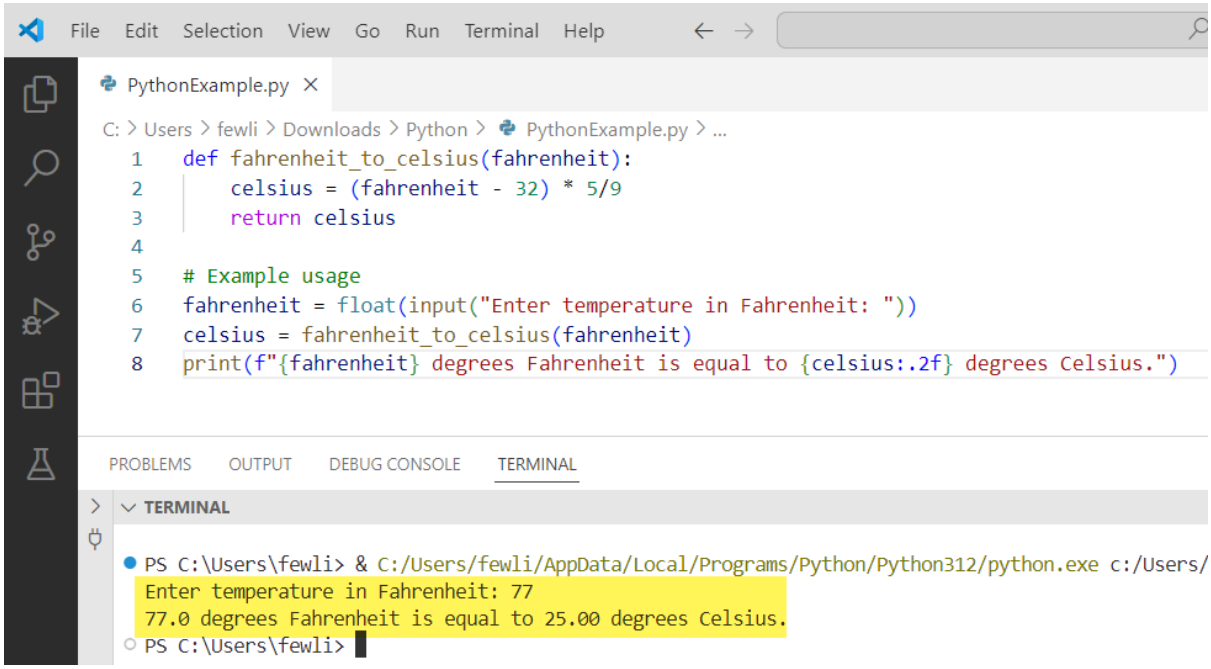
2. User Input:
    - The program prompts the user to enter a temperature in Fahrenheit.
    - The input is read as a string and then converted to a floating-point number using float().
3. Convert and Print:
    - The program calls the fahrenheit_to_celsius function with the user-provided Fahrenheit temperature.
    - It prints the result, formatted to two decimal places.

Example

Let's say the user inputs a temperature of 77 degrees Fahrenheit.

- The program will call fahrenheit_to_celsius(77).
- Inside the function, it calculates ((77 - 32) \times \frac{5}{9} = 25) degrees Celsius.
- The output will be: "77.0 degrees Fahrenheit is equal to 25.00 degrees Celsius."

Here is the exact output:



27# PYTHON PROGRAM TO CONVERT KILOMETERS TO MILES

To convert a distance from kilometers to miles, you can use the following conversion factor:

[\text{Miles} = \text{Kilometers} \times 0.621371]

Here is a Python program to perform this conversion:

```
def kilometers_to_miles(kilometers):
    miles = kilometers * 0.621371
    return miles

# Example usage
kilometers = float(input("Enter distance in kilometers: "))
miles = kilometers_to_miles(kilometers)
print(f"{kilometers} kilometers is equal to {miles:.2f} miles.")
```

Explanation

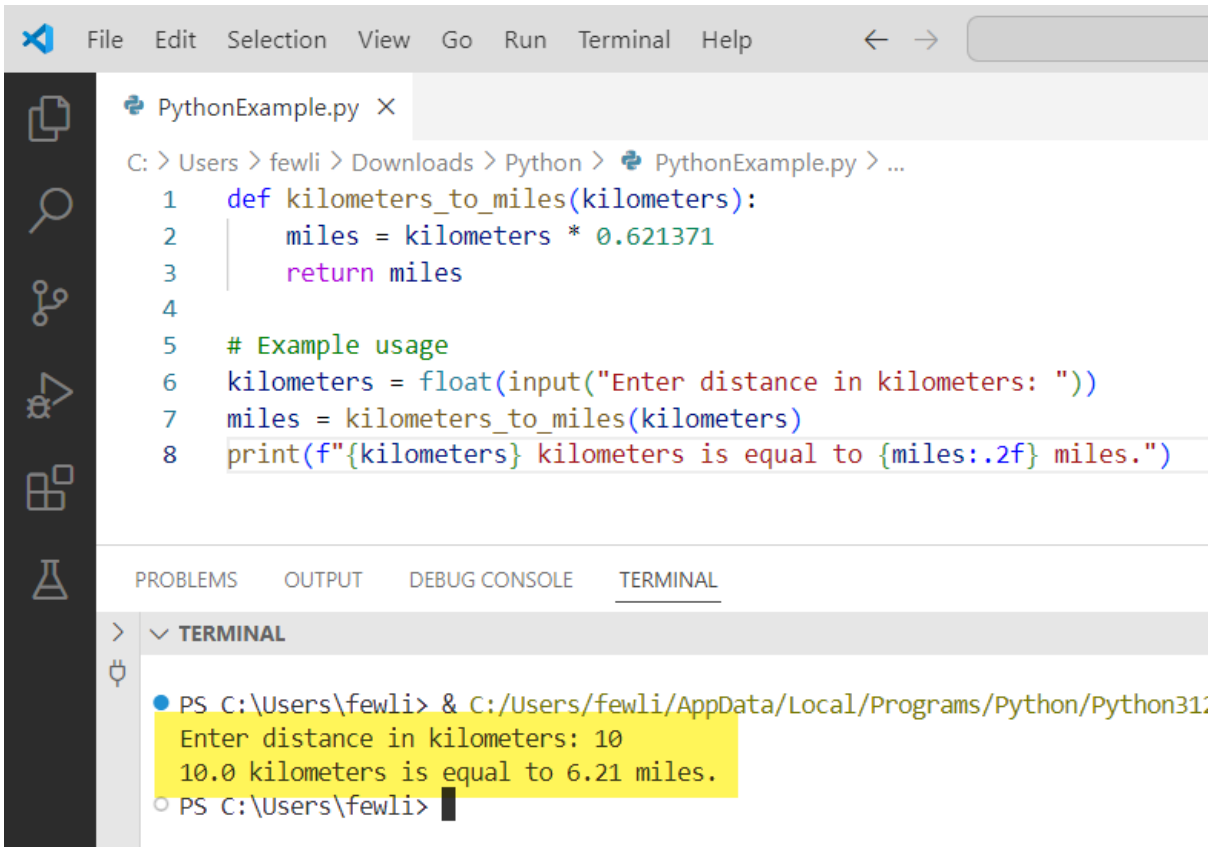
1. Define kilometers_to_miles Function:
    - This function takes a distance in kilometers as an argument.
    - It applies the conversion factor (\text{Kilometers} \times 0.621371) to calculate the equivalent distance in miles.
    - It returns the calculated distance in miles.
2. User Input:
    - The program prompts the user to enter a distance in kilometers.
    - The input is read as a string and then converted to a floating-point number using float().
3. Convert and Print:
    - The program calls the kilometers_to_miles function with the user-provided distance in kilometers.
    - It prints the result, formatted to two decimal places.

Example

Let's say the user inputs a distance of 10 kilometers.

- The program will call kilometers_to_miles(10).
- Inside the function, it calculates (10 \times 0.621371 = 6.21371) miles.
- The output will be: "10.0 kilometers is equal to 6.21 miles."

Here is the exact output in the screenshot below:



28# PYTHON PROGRAM TO CONVERT MILES TO KILOMETERS

To convert a distance from miles to kilometers, you can use the following conversion factor:

[\text{Kilometers} = \text{Miles} \times 1.60934]

Here is a Python program to perform this conversion:

```
def miles_to_kilometers(miles):
    kilometers = miles * 1.60934
    return kilometers

# Example usage
miles = float(input("Enter distance in miles: "))
kilometers = miles_to_kilometers(miles)
print(f"{miles} miles is equal to {kilometers:.2f} kilometers.")
```

Explanation

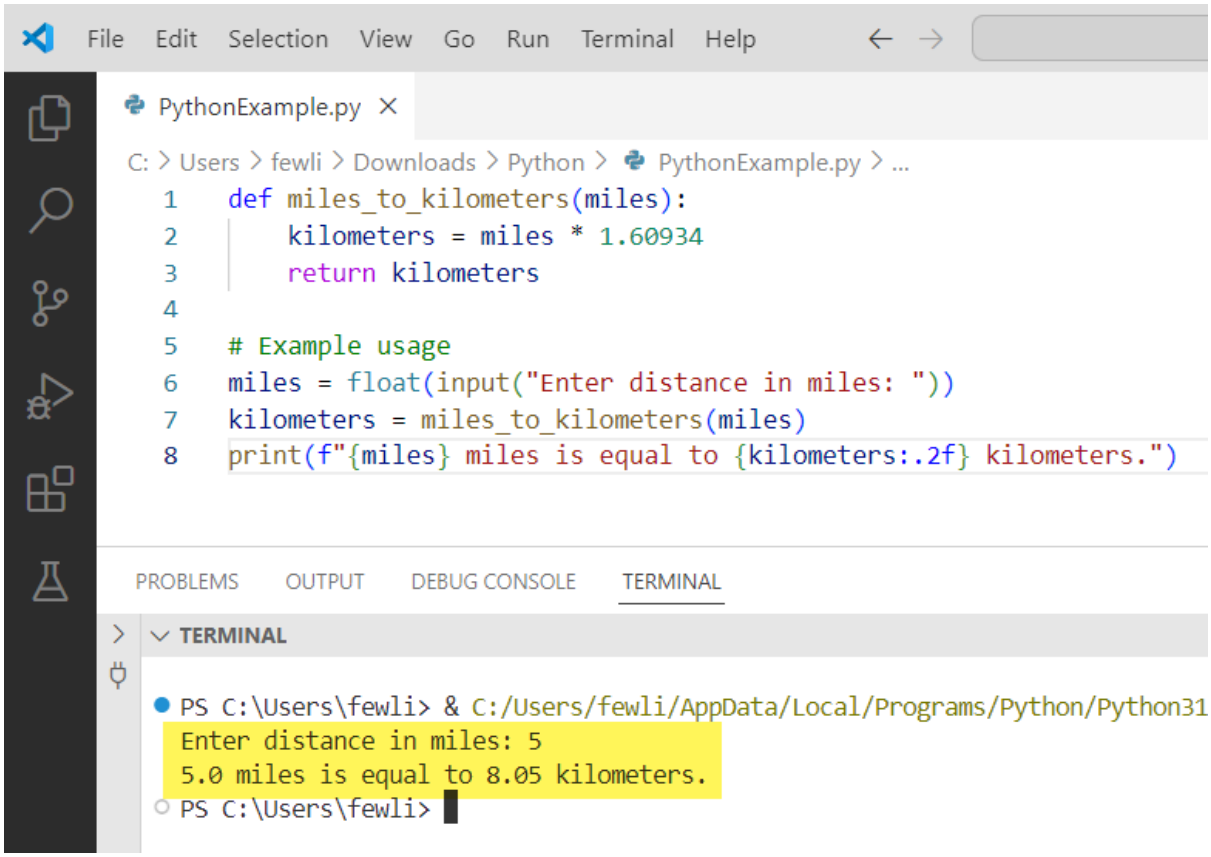
1. Define miles_to_kilometers Function:
    - This function takes a distance in miles as an argument.
    - It applies the conversion factor (\text{Miles} \times 1.60934) to calculate the equivalent distance in kilometers.
    - It returns the calculated distance in kilometers.
2. User Input:
    - The program prompts the user to enter a distance in miles.
    - The input is read as a string and then converted to a floating-point number using float().
3. Convert and Print:
    - The program calls the miles_to_kilometers function with the user-provided distance in miles.
    - It prints the result, formatted to two decimal places.

Example

Let's say the user inputs a distance of 5 miles.

- The program will call miles_to_kilometers(5).
- Inside the function, it calculates (5 \times 1.60934 = 8.0467) kilometers.
- The output will be: "5.0 miles is equal to 8.05 kilometers."

Here is the exact output:



29# PYTHON PROGRAM TO CONVERT BITS TO MEGABYTES GIGABYTES AND TERABYTES

To convert bits to megabytes, gigabytes, and terabytes, you need to understand the following relationships:

1. Bits to Bytes: [\text{Bytes} = \frac{\text{Bits}}{8}]
2. Bytes to Kilobytes: [\text{Kilobytes} = \frac{\text{Bytes}}{1024}]
3. Kilobytes to Megabytes: [\text{Megabytes} = \frac{\text{Kilobytes}}{1024}]
4. Megabytes to Gigabytes: [\text{Gigabytes} = \frac{\text{Megabytes}}{1024}]
5. Gigabytes to Terabytes: [\text{Terabytes} = \frac{\text{Gigabytes}}{1024}]

Here is a Python program that performs these conversions:

```
def bits_to_other_units(bits):
    # Convert bits to bytes
    bytes = bits / 8
```

```
    # Convert bytes to kilobytes
    kilobytes = bytes / 1024

    # Convert kilobytes to megabytes
    megabytes = kilobytes / 1024

    # Convert megabytes to gigabytes
    gigabytes = megabytes / 1024

    # Convert gigabytes to terabytes
    terabytes = gigabytes / 1024

    return megabytes, gigabytes, terabytes

# Example usage
bits = float(input("Enter the number of bits: "))
megabytes, gigabytes, terabytes = bits_to_other_units(bits)
print(f"{bits} bits is equal to {megabytes:.6f} megabytes, {gigabytes:.6f}
gigabytes, and {terabytes:.6f} terabytes.")
```

Explanation

1. Define bits_to_other_units Function:
    - This function takes the number of bits as an argument.
    - It converts bits to bytes by dividing by 8.
    - It converts bytes to kilobytes by dividing by 1024.
    - It converts kilobytes to megabytes by dividing by 1024.
    - It converts megabytes to gigabytes by dividing by 1024.
    - It converts gigabytes to terabytes by dividing by 1024.
    - It returns the values in megabytes, gigabytes, and terabytes.
2. User Input:
    - The program prompts the user to enter a number of bits.
    - The input is read as a string and then converted to a floating-point number using float().
3. Convert and Print:
    - The program calls the bits_to_other_units function with the user-provided number of bits.

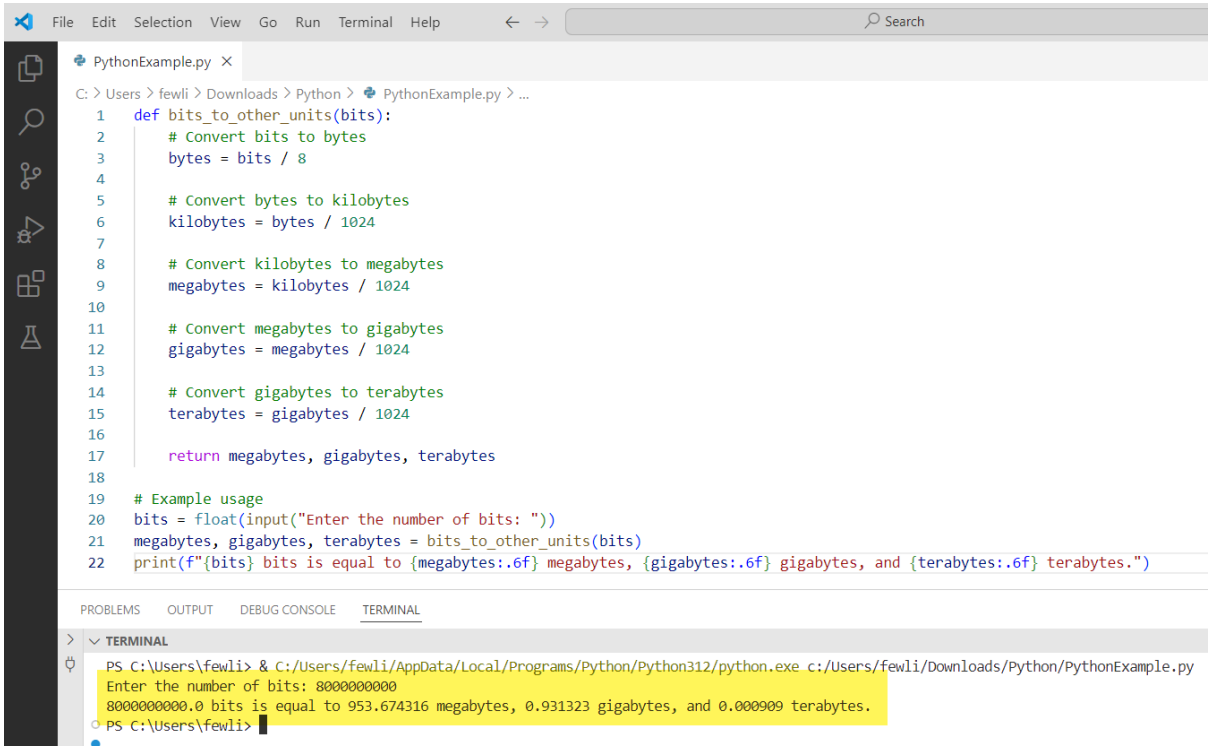
- It prints the results, formatted to six decimal places.

Example

Let's say the user inputs 8000000000 bits.

- The program will call bits_to_other_units(8000000000).
- Inside the function, it performs the following conversions:
  - (\text{Bytes} = \frac{8000000000}{8} = 1000000000)
  - (\text{Kilobytes} = \frac{1000000000}{1024} \approx 976562.5)
  - (\text{Megabytes} = \frac{976562.5}{1024} \approx 953.674316)
  - (\text{Gigabytes} = \frac{953.674316}{1024} \approx 0.931323)
  - (\text{Terabytes} = \frac{0.931323}{1024} \approx 0.000909)
- The output will be: "8000000000 bits is equal to 953.674316 megabytes, 0.931323 gigabytes, and 0.000909 terabytes."

Here is the exact output:



30# PYTHON PROGRAM TO CONVERT CENTIMETERS TO INCHES

To convert a length from centimeters to inches, you can use the following conversion factor:

[\text{Inches} = \text{Centimeters} \times 0.393701]

Here is a Python program to perform this conversion:

```
def centimeters_to_inches(centimeters):
    inches = centimeters * 0.393701
    return inches

# Example usage
centimeters = float(input("Enter length in centimeters: "))
inches = centimeters_to_inches(centimeters)
print(f"{centimeters} centimeters is equal to {inches:.2f} inches.")
```

Explanation

1. Define centimeters_to_inches Function:
    - This function takes a length in centimeters as an argument.
    - It applies the conversion factor (\text{Centimeters} \times 0.393701) to calculate the equivalent length in inches.
    - It returns the calculated length in inches.
2. User Input:
    - The program prompts the user to enter a length in centimeters.
    - The input is read as a string and then converted to a floating-point number using float().
3. Convert and Print:
    - The program calls the centimeters_to_inches function with the user-provided length in centimeters.
    - It prints the result, formatted to two decimal places.

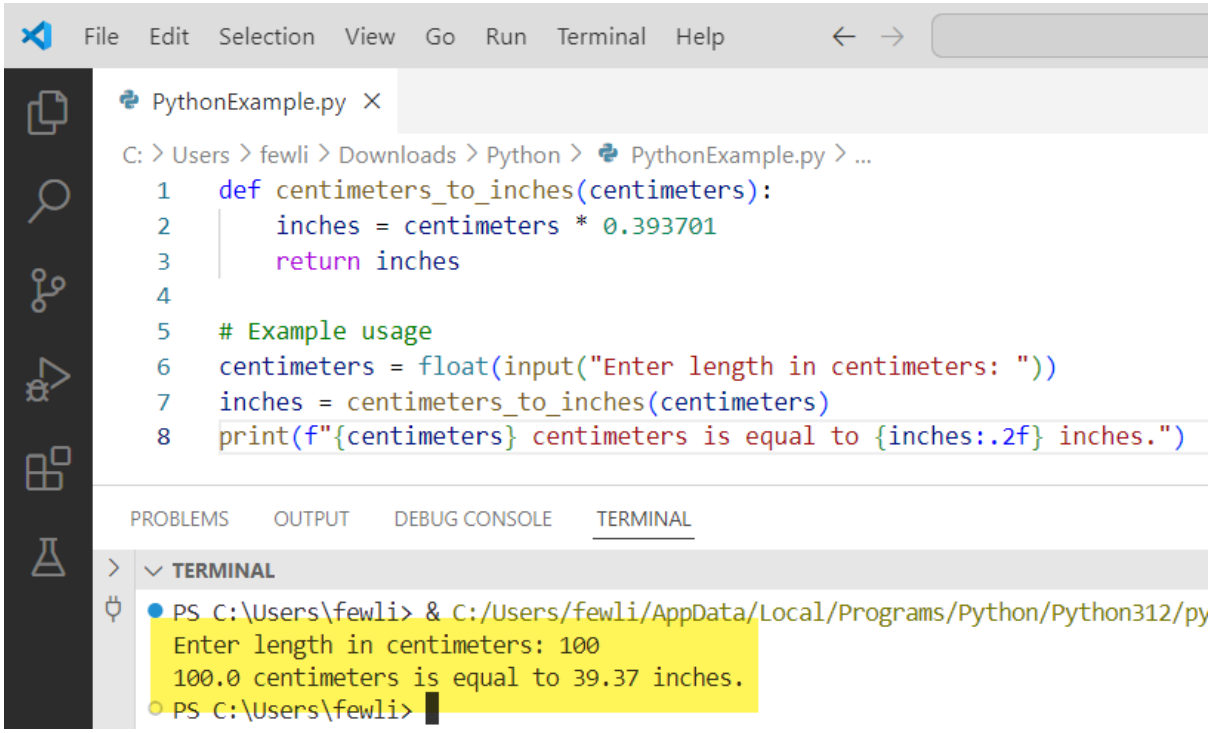
Example

Let's say the user inputs a length of 100 centimeters.

- The program will call centimeters_to_inches(100).

- Inside the function, it calculates (100 \times 0.393701 = 39.3701) inches.
- The output will be: "100.0 centimeters is equal to 39.37 inches."

Here is the exact output:



31# PYTHON PROGRAM TO CONVERT DAYS INTO YEARS WEEKS AND DAYS

To convert a given number of days into years, weeks, and remaining days, you need to understand the following relationships:

1. Days to Years:
    - 1 year = 365 days (assuming a non-leap year for simplicity)
2. Days to Weeks:
    - 1 week = 7 days

Here is a Python program to perform this conversion:

```
def convert_days(days):
    # Calculate the number of years
    years = days // 365
    remaining_days = days % 365
```

```
    # Calculate the number of weeks from the remaining days
    weeks = remaining_days // 7
    days_left = remaining_days % 7

    return years, weeks, days_left

# Example usage
total_days = int(input("Enter the number of days: "))
years, weeks, days_left = convert_days(total_days)
print(f"{total_days} days is equal to {years} years, {weeks} weeks, and
{days_left} days.")
```

Explanation

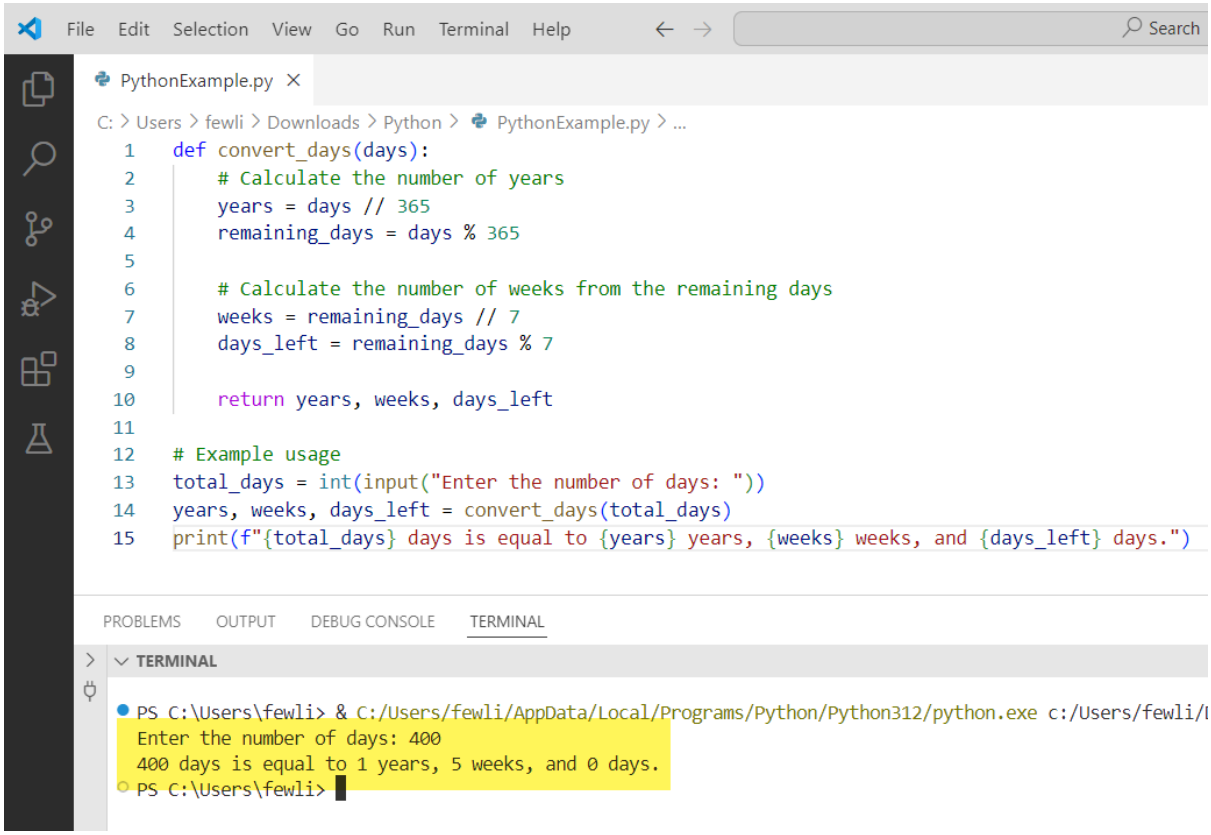
1. Define convert_days Function:
    - This function takes a total number of days as an argument.
    - It calculates the number of years by performing integer division of the total days by 365.
    - It calculates the remaining days after extracting the years using the modulus operator %.
    - It calculates the number of weeks from the remaining days by performing integer division by 7.
    - It calculates the remaining days after extracting the weeks using the modulus operator %.
    - It returns the number of years, weeks, and remaining days.
2. User Input:
    - The program prompts the user to enter the total number of days.
    - The input is read as a string and then converted to an integer using int().
3. Convert and Print:
    - The program calls the convert_days function with the user-provided number of days.
    - It prints the result, showing the equivalent number of years, weeks, and days.

Example

Let's say the user inputs 400 days.

- The program will call convert_days(400).
- Inside the function, it performs the following calculations:
  - (\text{Years} = 400 // 365 = 1)
  - (\text{Remaining Days} = 400 % 365 = 35)
  - (\text{Weeks} = 35 // 7 = 5)
  - (\text{Days Left} = 35 % 7 = 0)
- The output will be: "400 days is equal to 1 years, 5 weeks, and 0 days."

Here is the exact output:



32# PYTHON PROGRAM TO CONVERT DEGREE TO RADIAN

To convert an angle from degrees to radians, you can use the following conversion formula:

[\text{Radians} = \text{Degrees} \times \frac{\pi}{180}]

In Python, you can use the math module which provides the constant math.pi for the value of (\pi).

Here is a Python program to perform this conversion:

```
import math

def degrees_to_radians(degrees):
    radians = degrees * (math.pi / 180)
    return radians

# Example usage
degrees = float(input("Enter angle in degrees: "))
radians = degrees_to_radians(degrees)
print(f"{degrees} degrees is equal to {radians:.6f} radians.")
```

Explanation

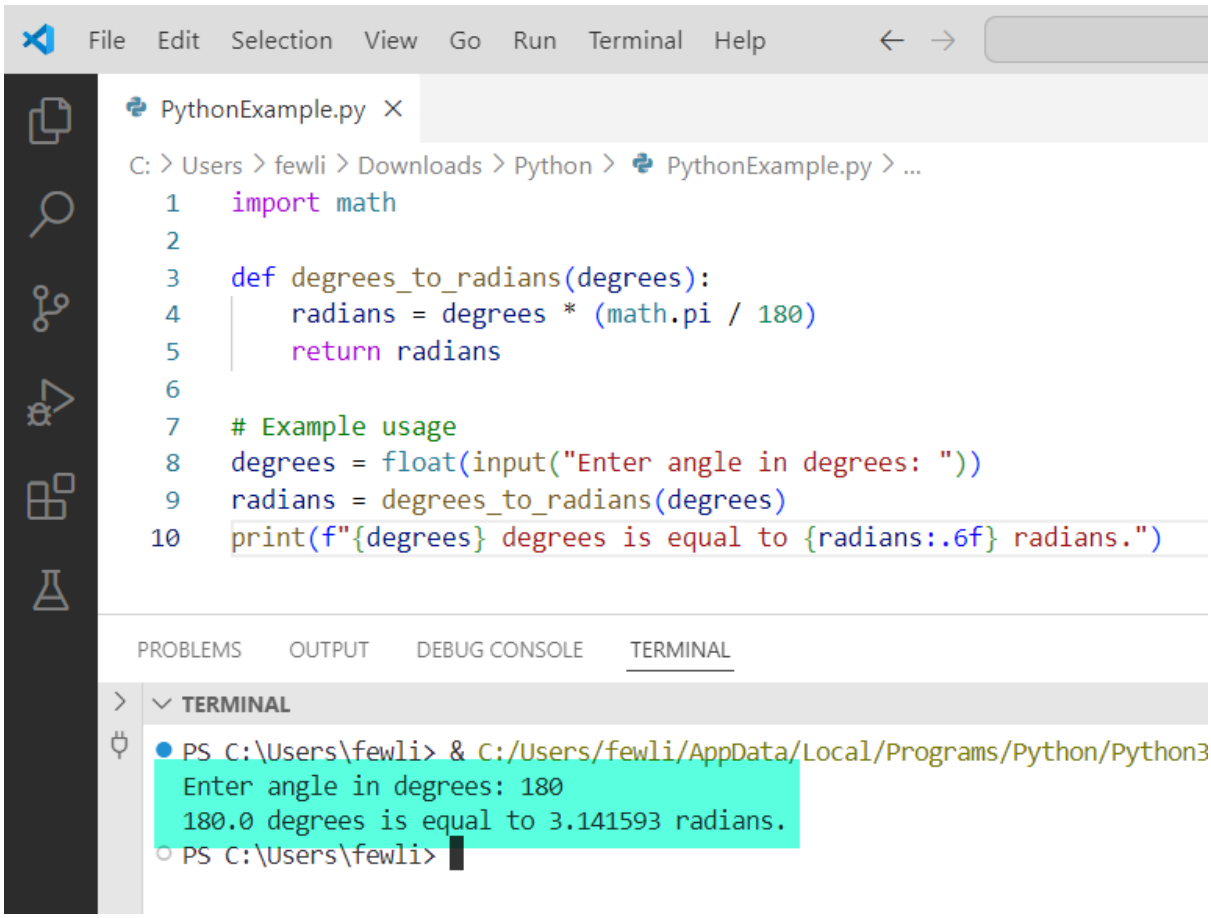
1. Import math Module:
    - The math module provides mathematical functions and constants, including math.pi.
2. Define degrees_to_radians Function:
    - This function takes an angle in degrees as an argument.
    - It applies the conversion formula (\text{Degrees} \times \frac{\pi}{180}) to calculate the equivalent angle in radians.
    - It returns the calculated angle in radians.
3. User Input:
    - The program prompts the user to enter an angle in degrees.
    - The input is read as a string and then converted to a floating-point number using float().
4. Convert and Print:
    - The program calls the degrees_to_radians function with the user-provided angle in degrees.
    - It prints the result, formatted to six decimal places.

Example

Let's say the user inputs an angle of 180 degrees.

- The program will call degrees_to_radians(180).
- Inside the function, it calculates (180 \times \frac{\pi}{180} = \pi \approx 3.141593) radians.
- The output will be: "180.0 degrees is equal to 3.141593 radians."

Here is the output in the screenshot:



33# PYTHON PROGRAM TO CONVERT POUNDS TO KILOGRAMS

To convert a weight from pounds to kilograms, you can use the following conversion factor:

[\text{Kilograms} = \text{Pounds} \times 0.453592]

Here is a Python program to perform this conversion:

```
def pounds_to_kilograms(pounds):
    kilograms = pounds * 0.453592
    return kilograms

# Example usage
pounds = float(input("Enter weight in pounds: "))
kilograms = pounds_to_kilograms(pounds)
```

```
print(f"{pounds} pounds is equal to {kilograms:.2f} kilograms.")
```

Explanation

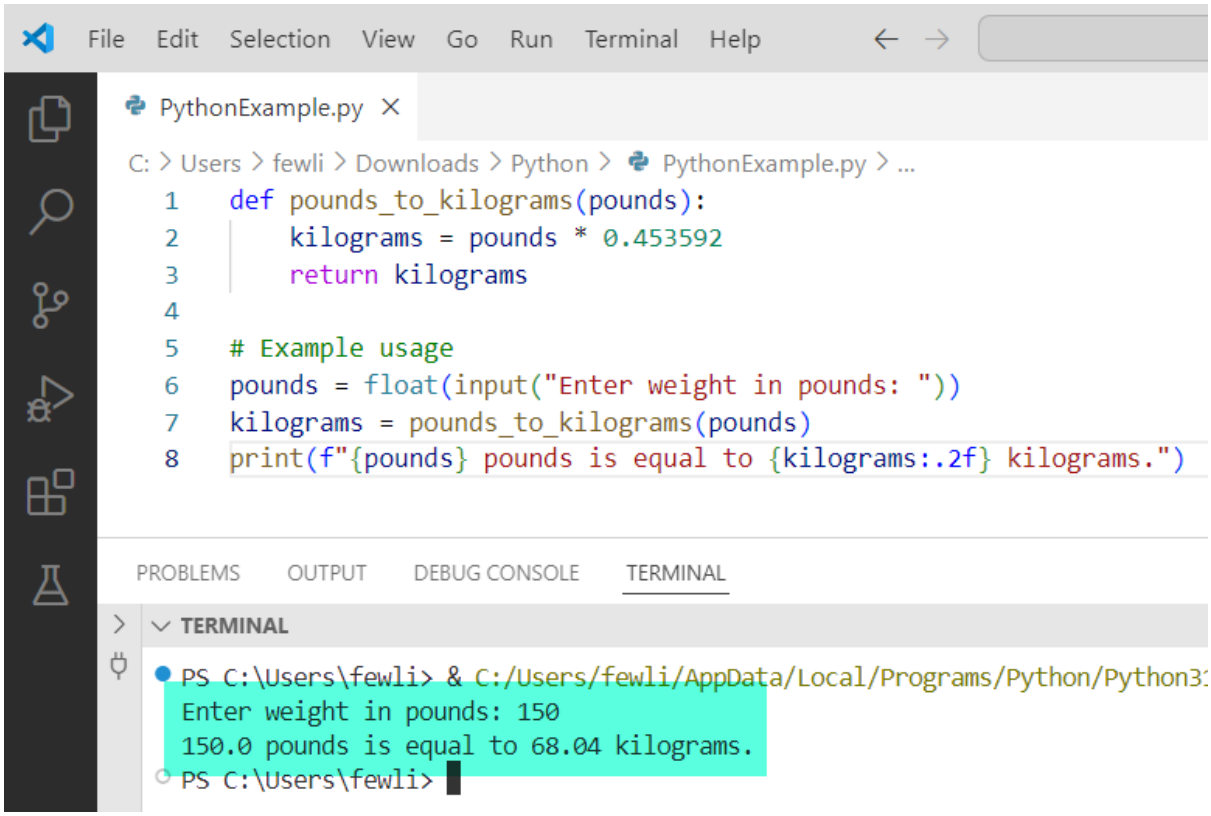
1. Define pounds_to_kilograms Function:
    - This function takes a weight in pounds as an argument.
    - It applies the conversion factor (\text{Pounds} \times 0.453592) to calculate the equivalent weight in kilograms.
    - It returns the calculated weight in kilograms.
2. User Input:
    - The program prompts the user to enter a weight in pounds.
    - The input is read as a string and then converted to a floating-point number using float().
3. Convert and Print:
    - The program calls the pounds_to_kilograms function with the user-provided weight in pounds.
    - It prints the result, formatted to two decimal places.

Example

Let's say the user inputs a weight of 150 pounds.

- The program will call pounds_to_kilograms(150).
- Inside the function, it calculates (150 \times 0.453592 = 68.0388) kilograms.
- The output will be: "150.0 pounds is equal to 68.04 kilograms."

Here is the exact output in the screenshot below:



34# PYTHON PROGRAM TO CONVERT LOWER CASE TO UPPER CASE

To convert a string from lowercase to uppercase in Python, you can use the built-in upper() method of string objects. This method returns a new string with all the characters converted to uppercase.

Here is a Python program to perform this conversion:

```
def convert_to_uppercase(input_string):
    # Convert the string to uppercase using the upper() method
    uppercase_string = input_string.upper()
    return uppercase_string

# Example usage
input_string = input("Enter a string in lowercase: ")
uppercase_string = convert_to_uppercase(input_string)
print(f"The string in uppercase is: {uppercase_string}")
```

Explanation

1. Define convert_to_uppercase Function:
    - This function takes a string input_string as an argument.

- It uses the upper() method to convert all characters in the string to uppercase.
- It returns the converted string.

2. User Input:
    - The program prompts the user to enter a string in lowercase.
    - The input is read as a string using the input() function.
3. Convert and Print:
    - The program calls the convert_to_uppercase function with the user-provided string.
    - It prints the result, showing the string converted to uppercase.

Example

Let's say the user inputs the string "hello world".

- The program will call convert_to_uppercase("hello world").
- Inside the function, it converts the string to uppercase using the upper() method: "hello world".upper() results in "HELLO WORLD".
- The output will be: "The string in uppercase is: HELLO WORLD".

35# PYTHON PROGRAM TO CONVERT UPPER CASE TO LOWER CASE

To convert a string from uppercase to lowercase in Python, you can use the built-in lower() method of string objects. This method returns a new string with all the characters converted to lowercase.

Here is a Python program to perform this conversion:

```
def convert_to_lowercase(input_string):
    # Convert the string to lowercase using the lower() method
    lowercase_string = input_string.lower()
    return lowercase_string

# Example usage
input_string = input("Enter a string in uppercase: ")
lowercase_string = convert_to_lowercase(input_string)
print(f"The string in lowercase is: {lowercase_string}")
```

Explanation

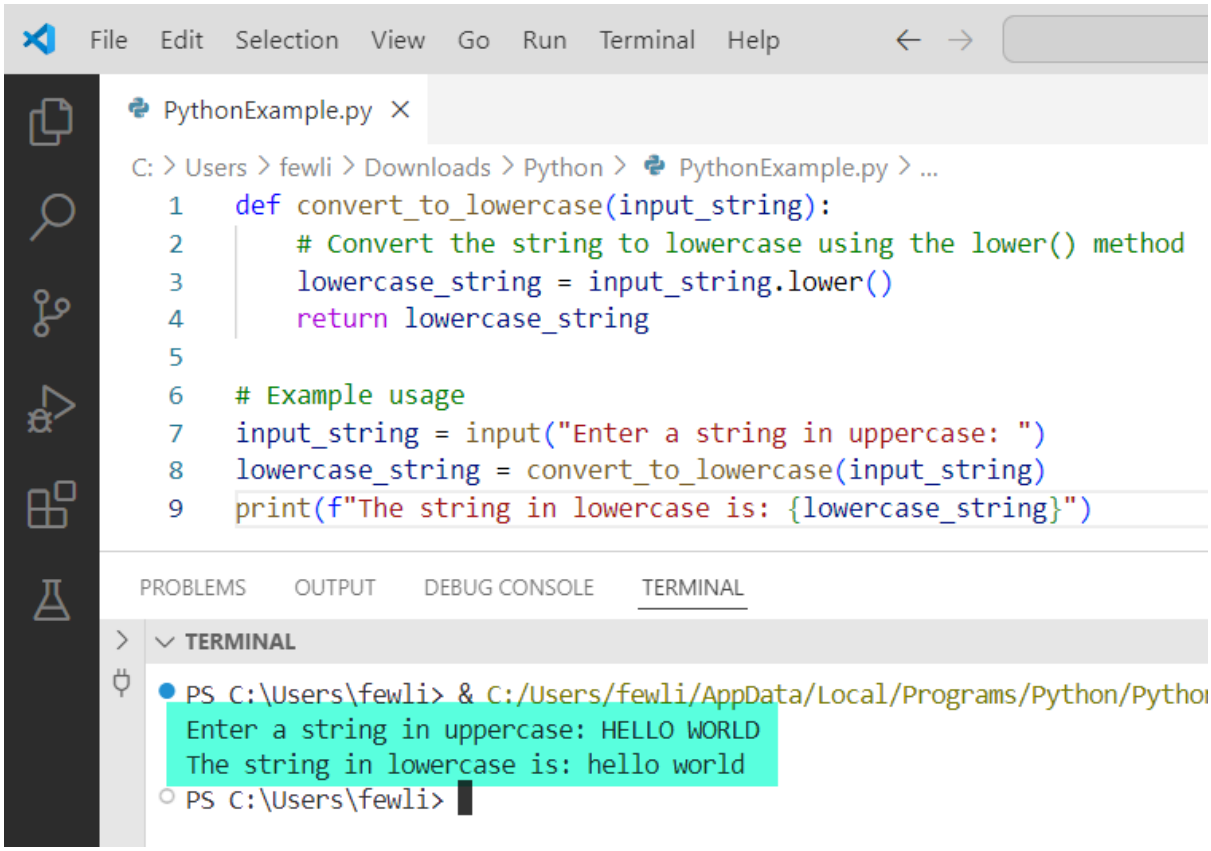
1. Define convert_to_lowercase Function:
    - This function takes a string input_string as an argument.
    - It uses the lower() method to convert all characters in the string to lowercase.
    - It returns the converted string.
2. User Input:
    - The program prompts the user to enter a string in uppercase.
    - The input is read as a string using the input() function.
3. Convert and Print:
    - The program calls the convert_to_lowercase function with the user-provided string.
    - It prints the result, showing the string converted to lowercase.

Example

Let's say the user inputs the string "HELLO WORLD".

- The program will call convert_to_lowercase("HELLO WORLD").
- Inside the function, it converts the string to lowercase using the lower() method: "HELLO WORLD".lower() results in "hello world".
- The output will be: "The string in lowercase is: hello world".

Here is the exact screenshot:



36# PYTHON PROGRAM TO FIND NUMBER OF VOWELS IN A STRING

To find the number of vowels in a string in Python, you can iterate through each character in the string and check if it is a vowel. You can use a set to store the vowels for quick lookup.

Here is a Python program to count the number of vowels in a string:

```
def count_vowels(input_string):
    # Define a set of vowels
    vowels = set("aeiouAEIOU")

    # Initialize a counter for vowels
    vowel_count = 0

    # Iterate through each character in the string
    for char in input_string:
        if char in vowels:
            vowel_count += 1

    return vowel_count

# Example usage
input_string = input("Enter a string: ")
vowel_count = count_vowels(input_string)
print(f"The number of vowels in the string is: {vowel_count}")
```

Explanation

1. Define count_vowels Function:
    - This function takes a string input_string as an argument.
    - It defines a set of vowels vowels = set("aeiouAEIOU") to include both lowercase and uppercase vowels.
    - It initializes a counter vowel_count to zero.
2. Iterate Through the String:
    - The function iterates through each character in the string using a for loop.
    - For each character, it checks if the character is in the set of vowels.
    - If the character is a vowel, it increments the vowel_count by 1.
3. Return the Count:
    - After iterating through the entire string, the function returns the total count of vowels.
4. User Input:
    - The program prompts the user to enter a string.
    - The input is read as a string using the input() function.
5. Count and Print:
    - The program calls the count_vowels function with the user-provided string.
    - It prints the result, showing the number of vowels in the string.

Example

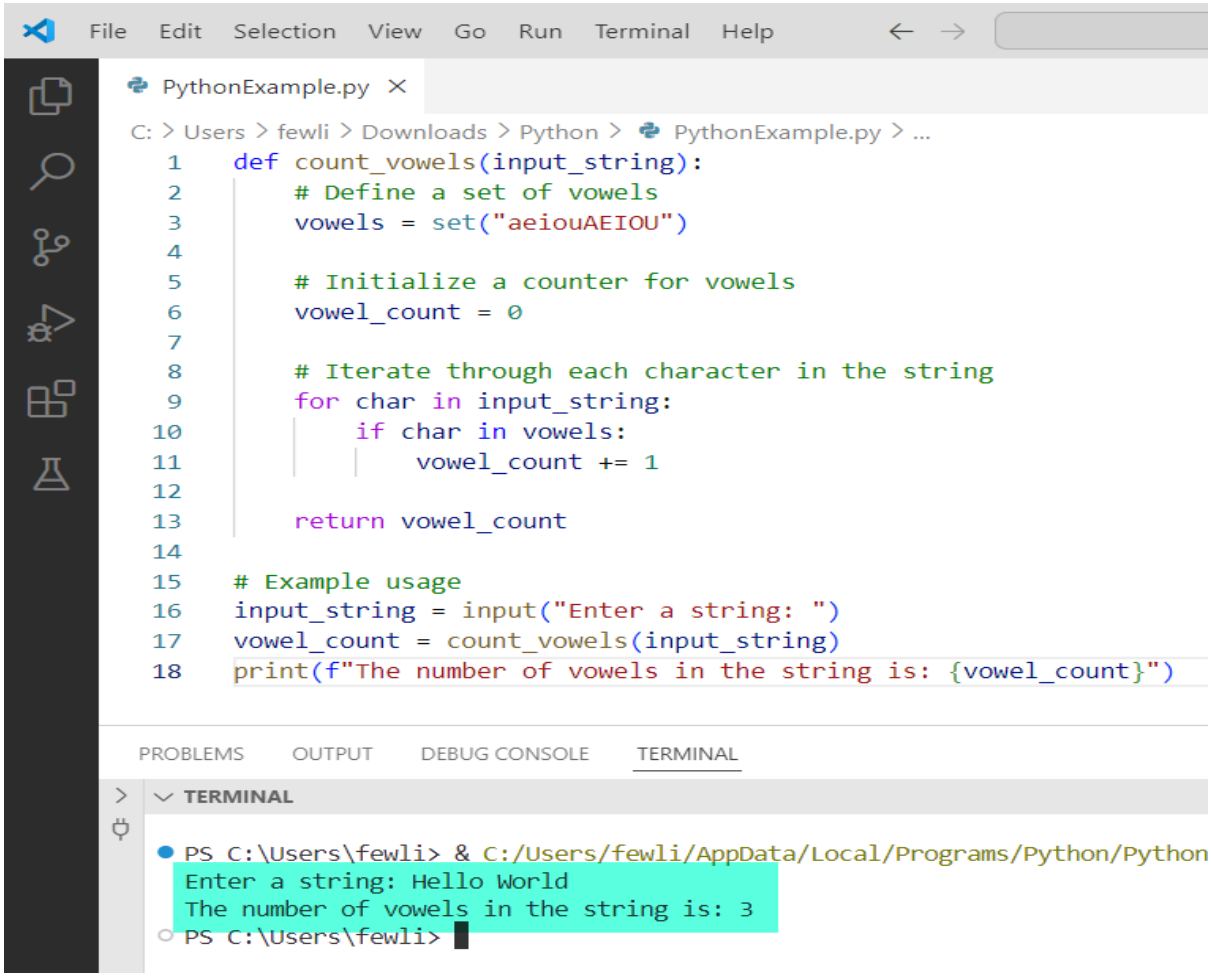
Let's say the user inputs the string "Hello World".

- The program will call count_vowels("Hello World").
- Inside the function, it iterates through each character:
    - 'H' (not a vowel)

- 'e' (vowel, count becomes 1)
- 'l' (not a vowel)
- 'l' (not a vowel)
- 'o' (vowel, count becomes 2)
- ' ' (not a vowel)
- 'W' (not a vowel)
- 'o' (vowel, count becomes 3)
- 'r' (not a vowel)
- 'l' (not a vowel)
- 'd' (not a vowel)

- The output will be: "The number of vowels in the string is: 3".

Here is the exact output in the screenshot below:



37# PYTHON PROGRAM FOR BINARY TO DECIMAL CONVERSION

To convert a binary number to a decimal number in Python, you can use the built-in int function with the base parameter set to 2. This method simplifies the conversion process.

```
def binary_to_decimal(binary_str):
    # Convert binary string to decimal using int function
    decimal_number = int(binary_str, 2)
    return decimal_number

# Example usage
binary_str = input("Enter a binary number: ")
decimal_number = binary_to_decimal(binary_str)
print(f"The decimal equivalent of binary {binary_str} is {decimal_number}.")
```

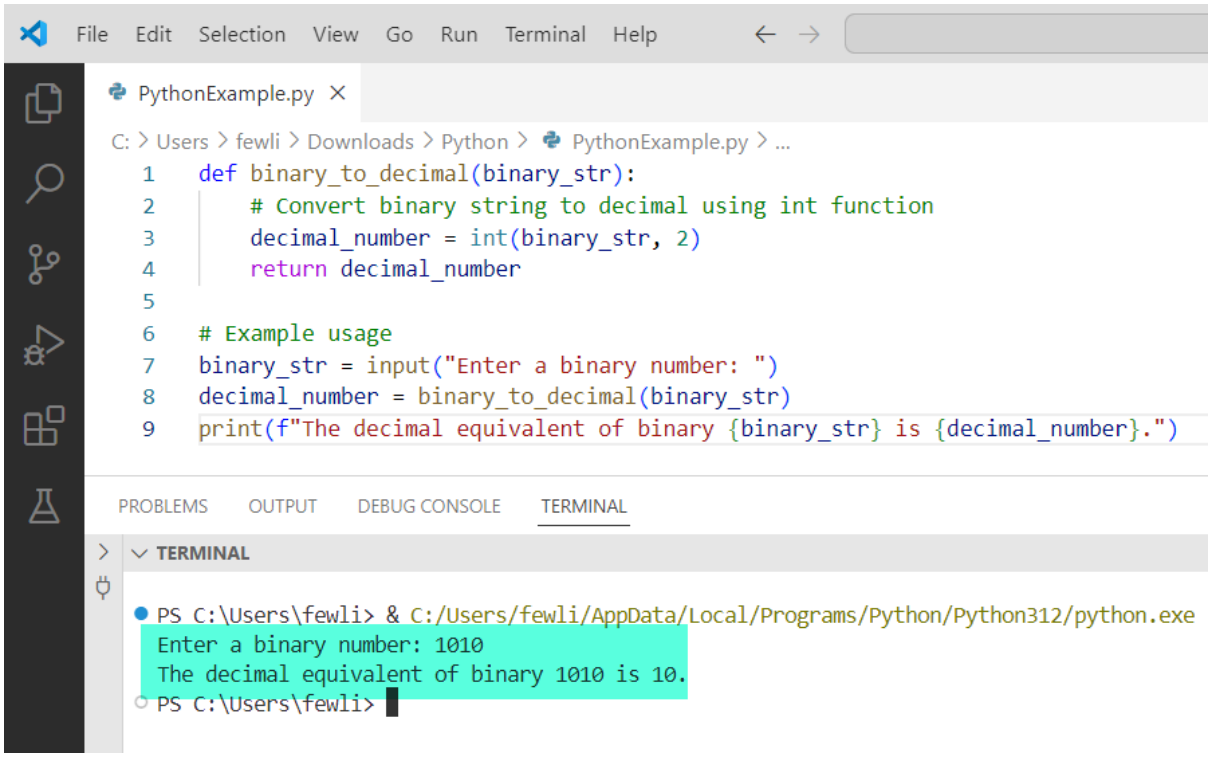
Explanation

- The int function can convert a binary string to a decimal number by specifying the base as 2.
- int(binary_str, 2) converts the binary string binary_str to its decimal equivalent.
- The function binary_to_decimal takes a binary string as an argument, uses the int function to perform the conversion, and returns the decimal number.

Let's say the user inputs the binary number "1010".

- Using the int Function:
    - The program calls binary_to_decimal("1010").
    - Inside the function, int("1010", 2) converts "1010" to 10.
    - The output will be: "The decimal equivalent of binary 1010 is 10."

Here is the exact output in the screenshot below:



38# PYTHON PROGRAM TO CALCULATE DISCOUNTED AMOUNT WITH DISCOUNT

To calculate the discounted amount with a given discount percentage in Python, you can follow these steps:

1. Get the original price of the item.
2. Get the discount percentage.
3. Calculate the discount amount.
4. Subtract the discount amount from the original price to get the discounted price.

Here is a Python program to perform this calculation:

```
def calculate_discounted_amount(original_price, discount_percentage):
    # Calculate the discount amount
    discount_amount = (original_price * discount_percentage) / 100

    # Calculate the discounted price
    discounted_price = original_price - discount_amount

    return discounted_price, discount_amount

# Example usage
original_price = float(input("Enter the original price: "))
```

```
discount_percentage = float(input("Enter the discount percentage: "))
discounted_price, discount_amount =
calculate_discounted_amount(original_price, discount_percentage)
print(f"The discount amount is: {discount_amount:.2f}")
print(f"The discounted price is: {discounted_price:.2f}")
```

Explanation

1. Define calculate_discounted_amount Function:
    - This function takes two arguments: original_price and discount_percentage.
    - It calculates the discount amount using the formula: [\text{Discount Amount} = \left(\frac{\text{Original Price} \times \text{Discount Percentage}}{100} \right)]
    - It calculates the discounted price by subtracting the discount amount from the original price: [\text{Discounted Price} = \text{Original Price} - \text{Discount Amount}]
    - It returns both the discounted price and the discount amount.
2. User Input:
    - The program prompts the user to enter the original price of the item.
    - The input is read as a string and then converted to a floating-point number using float().
    - The program prompts the user to enter the discount percentage.
    - The input is read as a string and then converted to a floating-point number using float().
3. Calculate and Print:
    - The program calls the calculate_discounted_amount function with the user-provided original price and discount percentage.
    - It prints the discount amount and the discounted price, formatted to two decimal places.

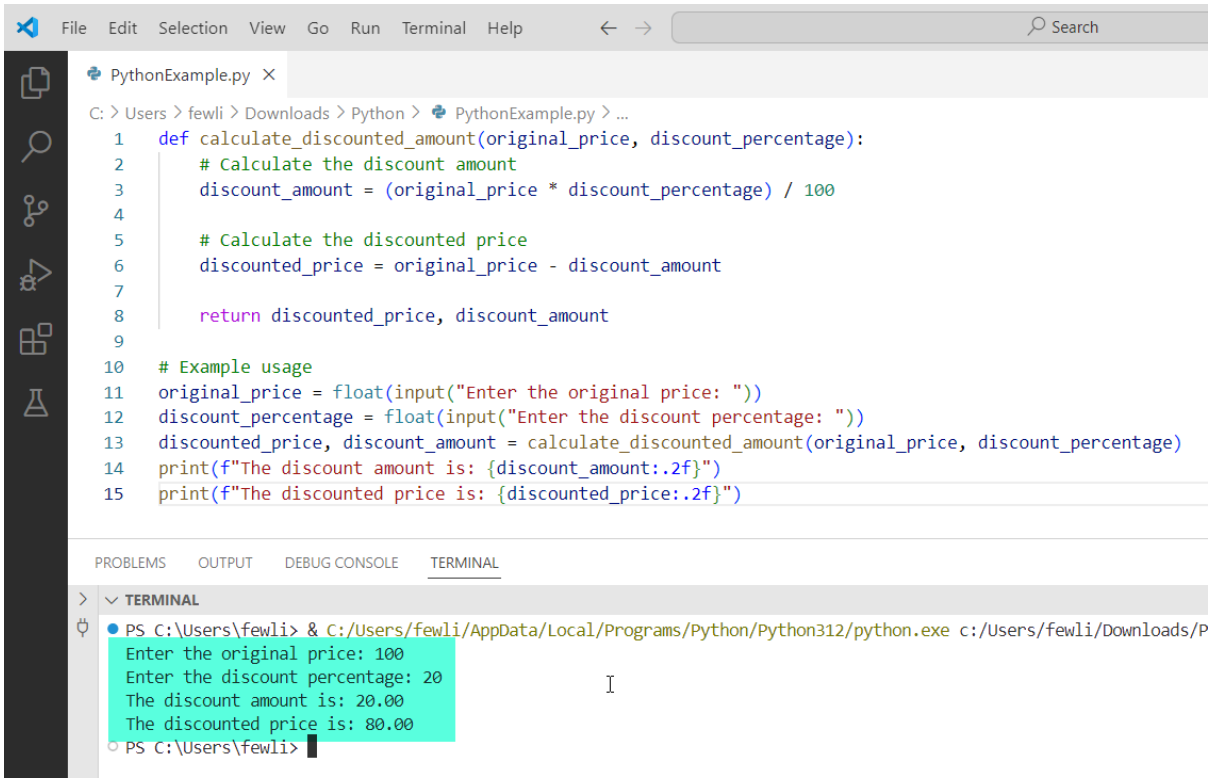
Example

Let's say the user inputs an original price of $100 and a discount percentage of 20%.

- The program will call calculate_discounted_amount(100, 20).
- Inside the function:
    - It calculates the discount amount: [\text{Discount Amount} = \left(\frac{100 \times 20}{100} \right) = 20]
    - It calculates the discounted price: [\text{Discounted Price} = 100 - 20 = 80]
- The output will be:
- The discount amount is: 20.00

The discounted price is: 80.00

Here is exact output in the screenshot below:



39# PYTHON PROGRAM TO CALCULATE DEPRECIATION

Depreciation is the reduction in the value of an asset over time. There are several methods to calculate depreciation, but one of the most commonly used methods is the Straight-Line Depreciation method. This method spreads the cost of the asset evenly over its useful life.

Straight-Line Depreciation Formula

The formula for calculating straight-line depreciation is:

[\text{Depreciation Expense} = \frac{\text{Cost of Asset} - \text{Salvage Value}}{\text{Useful Life}}]

- Cost of Asset: The initial cost of the asset.
- Salvage Value: The value of the asset at the end of its useful life.
- Useful Life: The expected time period over which the asset will be used.

Here is a Python program to calculate depreciation using the straight-line method:

```
def calculate_depreciation(cost, salvage_value, useful_life):
    # Calculate the annual depreciation expense
    depreciation_expense = (cost - salvage_value) / useful_life
    return depreciation_expense

# Example usage
cost = float(input("Enter the cost of the asset: "))
salvage_value = float(input("Enter the salvage value of the asset: "))
useful_life = float(input("Enter the useful life of the asset (in years): "))

annual_depreciation = calculate_depreciation(cost, salvage_value, useful_life)
print(f"The annual depreciation expense is: {annual_depreciation:.2f}")
```

Explanation

1. Define calculate_depreciation Function:
    - This function takes three arguments: cost, salvage_value, and useful_life.
    - It calculates the annual depreciation expense using the formula: [\text{Depreciation Expense} = \frac{\text{Cost of Asset} - \text{Salvage Value}}{\text{Useful Life}}]
    - It returns the calculated annual depreciation expense.
2. User Input:
    - The program prompts the user to enter the cost of the asset.

- The input is read as a string and then converted to a floating-point number using float().
- The program prompts the user to enter the salvage value of the asset.
- The input is read as a string and then converted to a floating-point number using float().
- The program prompts the user to enter the useful life of the asset in years.
- The input is read as a string and then converted to a floating-point number using float().

3. Calculate and Print:
    - The program calls the calculate_depreciation function with the user-provided cost, salvage value, and useful life.
    - It prints the annual depreciation expense, formatted to two decimal places.

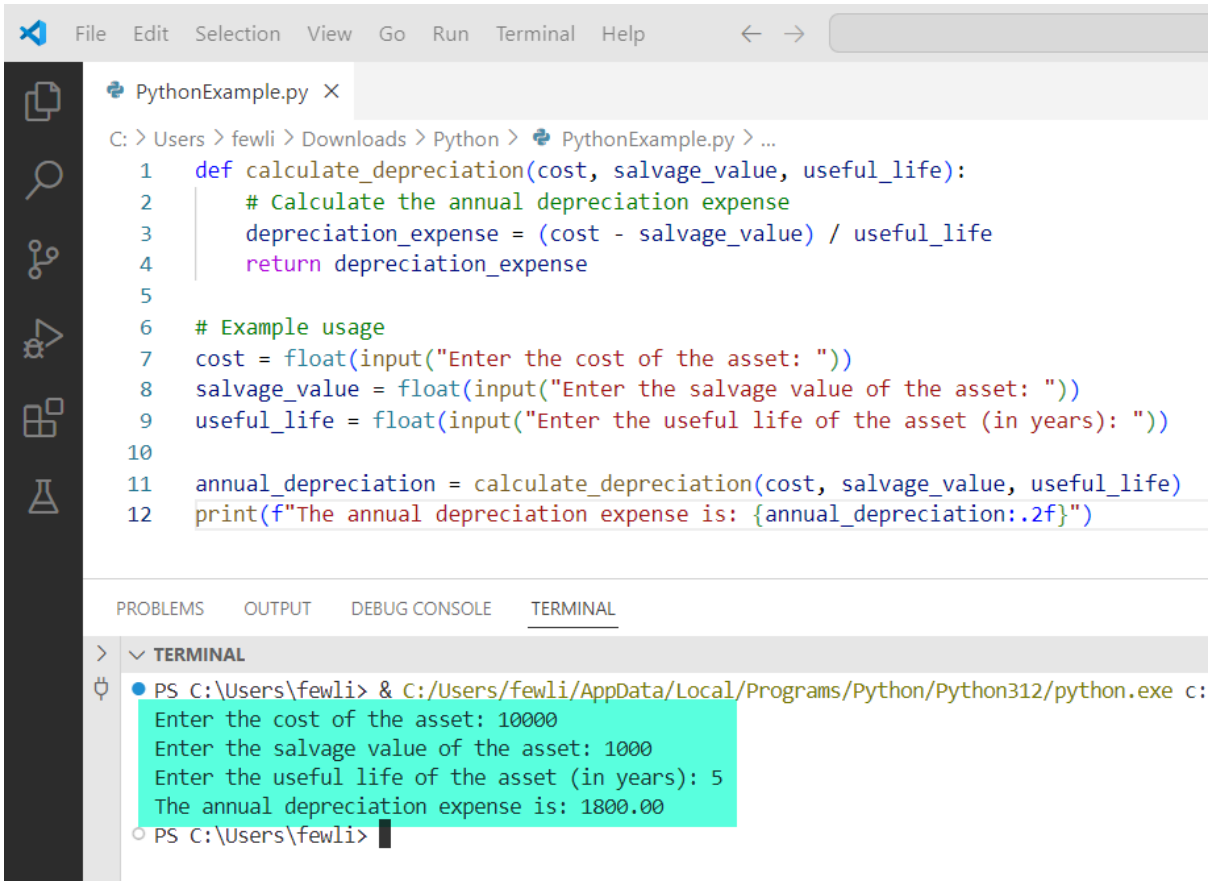
Example

Let's say the user inputs the following values:

- Cost of the asset: $10,000
- Salvage value: $1,000
- Useful life: 5 years
- The program will call calculate_depreciation(10000, 1000, 5).
- Inside the function:
    - It calculates the annual depreciation expense: [\text{Depreciation Expense} = \frac{10000 - 1000}{5} = 1800]
- The output will be:

The annual depreciation expense is: 1800.00

Here is the exact output:



40# PYTHON PROGRAM TO CALCULATE BODY MASS INDEX

Body Mass Index (BMI) is a measure that uses a person's weight and height to estimate whether they have a healthy body weight. The formula to calculate BMI is:

[\text{BMI} = \frac{\text{weight (kg)}}{\text{height (m)}^2}]

Here is a Python program to calculate BMI:

```
def calculate_bmi(weight, height):
    # Calculate BMI using the formula
    bmi = weight / (height ** 2)
    return bmi

# Example usage
weight = float(input("Enter your weight in kilograms: "))
height = float(input("Enter your height in meters: "))

bmi = calculate_bmi(weight, height)
print(f"Your BMI is: {bmi:.2f}")

# Optional: Categorize the BMI result
def categorize_bmi(bmi):
    if bmi < 18.5:
```

```
        return "Underweight"
    elif 18.5 <= bmi < 24.9:
        return "Normal weight"
    elif 25 <= bmi < 29.9:
        return "Overweight"
    else:
        return "Obesity"

category = categorize_bmi(bmi)
print(f"Your BMI category is: {category}")
```

Explanation

1. Define calculate_bmi Function:
    - This function takes two arguments: weight (in kilograms) and height (in meters).
    - It calculates BMI using the formula: [\text{BMI} = \frac{\text{weight}}{\text{height}^2}]
    - It returns the calculated BMI.
2. User Input:
    - The program prompts the user to enter their weight in kilograms.
    - The input is read as a string and then converted to a floating-point number using float().
    - The program prompts the user to enter their height in meters.
    - The input is read as a string and then converted to a floating-point number using float().
3. Calculate and Print:
    - The program calls the calculate_bmi function with the user-provided weight and height.
    - It prints the BMI, formatted to two decimal places.
4. Optional: Categorize the BMI Result:
    - The categorize_bmi function takes the BMI value as an argument.

- It categorizes the BMI based on standard BMI ranges:
    - BMI < 18.5: Underweight
    - 18.5 <= BMI < 24.9: Normal weight
    - 25 <= BMI < 29.9: Overweight
    - BMI >= 30: Obesity
- It returns the category as a string.
- The program prints the BMI category.

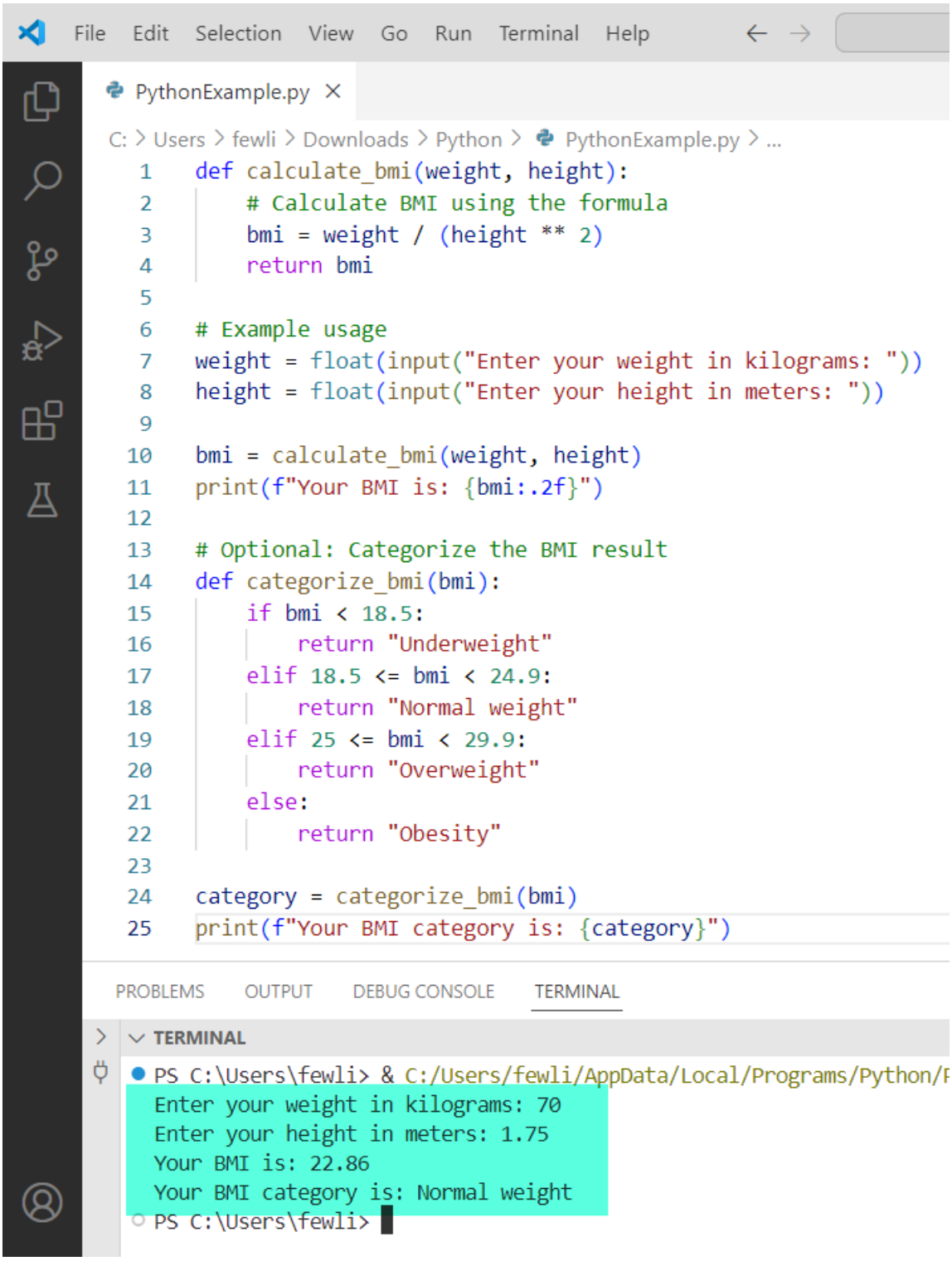
Example

Let's say the user inputs the following values:

- Weight: 70 kg
- Height: 1.75 m
- The program will call calculate_bmi(70, 1.75).
- Inside the function:
    - It calculates the BMI: [\text{BMI} = \frac{70}{1.75^2} \approx 22.86]
- The output will be:
- Your BMI is: 22.86
- The program will call categorize_bmi(22.86).
- Inside the function:
    - It determines that 22.86 falls within the "Normal weight" range.
- The output will be:

Your BMI category is: Normal weight

Here is the exact output:



41# PYTHON PROGRAM TO CALCULATE DISTANCE BETWEEN TWO POINTS

To calculate the distance between two points in a 2D plane, you can use the Euclidean distance formula. The formula for the distance (d) between two points ((x_1, y_1)) and ((x_2, y_2)) is:

[d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}]

Here is a Python program to calculate the distance between two points using this formula:

```
import math

def calculate_distance(x1, y1, x2, y2):
    # Calculate the distance using the Euclidean distance formula
    distance = math.sqrt((x2 - x1) ** 2 + (y2 - y1) ** 2)
    return distance

# Example usage
x1 = float(input("Enter the x-coordinate of the first point: "))
y1 = float(input("Enter the y-coordinate of the first point: "))
x2 = float(input("Enter the x-coordinate of the second point: "))
y2 = float(input("Enter the y-coordinate of the second point: "))

distance = calculate_distance(x1, y1, x2, y2)
print(f"The distance between the points ({x1}, {y1}) and ({x2}, {y2}) is
{distance:.2f}")
```

Explanation

1. Import math Module:
    - The math module provides access to mathematical functions, including math.sqrt for square root calculation.
2. Define calculate_distance Function:
    - This function takes four arguments: the (x) and (y) coordinates of the first point ((x1, y1)) and the (x) and (y) coordinates of the second point ((x2, y2)).
    - It calculates the distance using the Euclidean distance formula: [d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}]
    - It returns the calculated distance.
3. User Input:
    - The program prompts the user to enter the (x) and (y) coordinates of the first point.
    - The inputs are read as strings and then converted to floating-point numbers using float().

- The program prompts the user to enter the (x) and (y) coordinates of the second point.
- The inputs are read as strings and then converted to floating-point numbers using float().

4. Calculate and Print:
    - The program calls the calculate_distance function with the user-provided coordinates.
    - It prints the distance, formatted to two decimal places.

Example

Let's say the user inputs the following coordinates:

- First point: ((3, 4))
- Second point: ((6, 8))
- The program will call calculate_distance(3, 4, 6, 8).
- Inside the function:
    - It calculates the distance: [d = \sqrt{(6 - 3)^2 + (8 - 4)^2} = \sqrt{3^2 + 4^2} = \sqrt{9 + 16} = \sqrt{25} = 5]
- The output will be:

The distance between the points (3.0, 4.0) and (6.0, 8.0) is 5.00

42# PYTHON PROGRAM TO FIND DAY OF THE WEEK FOR A GIVEN DATE

To find the day of the week for a given date in Python, you can use the datetime module, which provides classes for manipulating dates and times. Specifically, the datetime.date class and its weekday() or strftime() methods can be used to determine the day of the week.

Here is a Python program to find the day of the week for a given date:

```
import datetime

def find_day_of_week(year, month, day):
    # Create a date object for the given date
    date = datetime.date(year, month, day)
```

```
    # Get the day of the week (0=Monday, 6=Sunday)
    day_of_week_index = date.weekday()

    # List of days of the week
    days_of_week = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday",
"Saturday", "Sunday"]

    # Get the name of the day
    day_of_week = days_of_week[day_of_week_index]

    return day_of_week

# Example usage
year = int(input("Enter year (e.g., 2024): "))
month = int(input("Enter month (1-12): "))
day = int(input("Enter day (1-31): "))

day_of_week = find_day_of_week(year, month, day)
print(f"The day of the week for {year}-{month:02d}-{day:02d} is
{day_of_week}.")
```

Explanation

1. Import datetime Module:
    - The datetime module provides classes for manipulating dates and times.
2. Define find_day_of_week Function:
    - This function takes three arguments: year, month, and day.
    - It creates a date object for the given date using datetime.date(year, month, day).
    - It uses the weekday() method of the date object to get the day of the week as an index (0 for Monday, 6 for Sunday).
    - It defines a list days_of_week containing the names of the days of the week.
    - It retrieves the name of the day corresponding to the index using days_of_week[day_of_week_index].
    - It returns the name of the day.
3. User Input:

- The program prompts the user to enter the year, month, and day.
- The inputs are read as strings and then converted to integers using int().

4. Find and Print:
    - The program calls the find_day_of_week function with the user-provided year, month, and day.
    - It prints the day of the week for the given date.

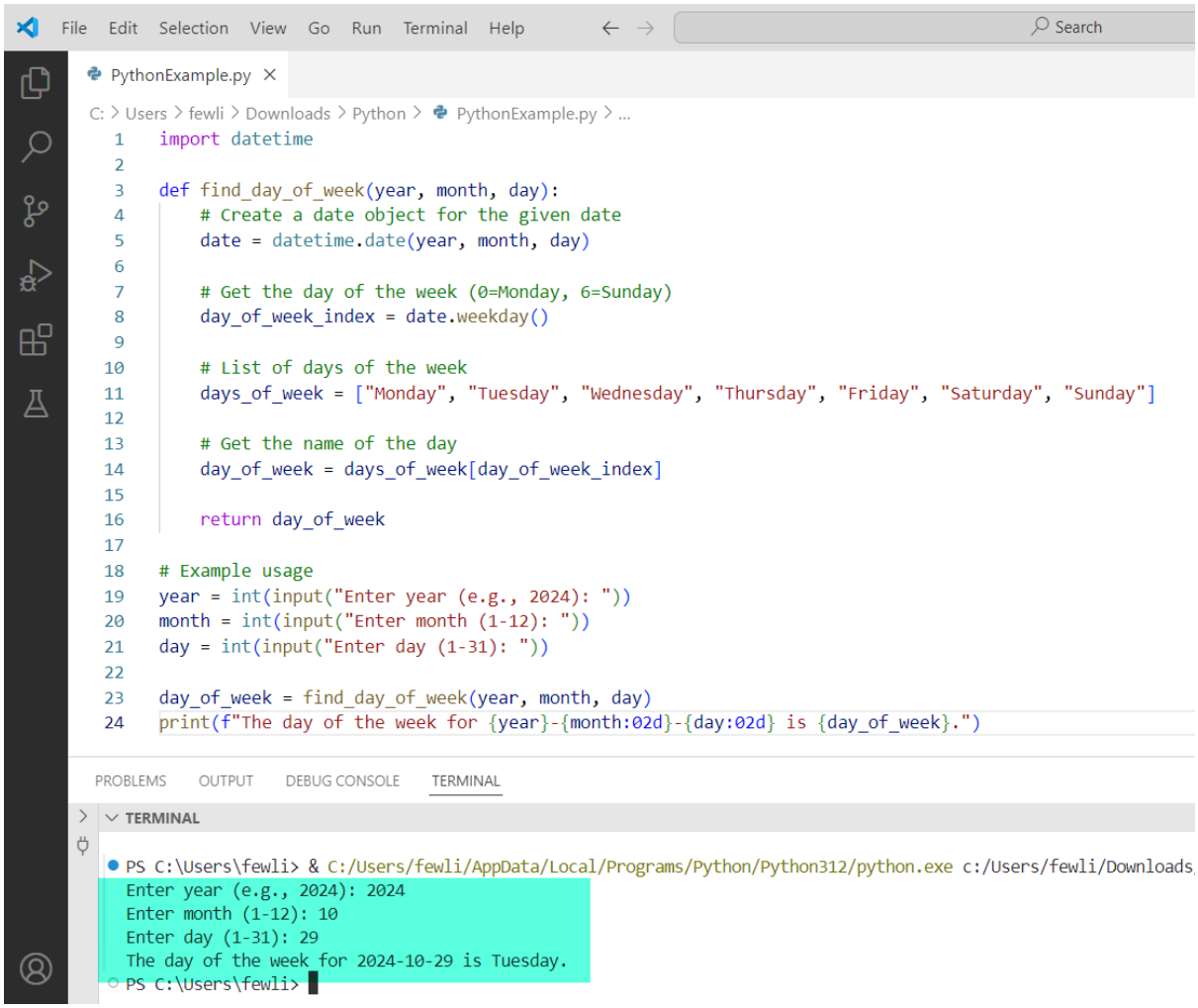
Example

Let's say the user inputs the following date:

- Year: 2024
- Month: 10
- Day: 29
- The program will call find_day_of_week(2024, 10, 29).
- Inside the function:
    - It creates a date object for 2024-10-29.
    - It uses the weekday() method to get the day of the week index, which is 1 (Tuesday).
    - It retrieves the name "Tuesday" from the days_of_week list.
- The output will be:

The day of the week for 2024-10-29 is Tuesday.

Here is the exact output:



43# PYTHON PROGRAM TO FIND PERCENTAGE OF 5 SUBJECTS

To calculate the percentage of marks obtained in five subjects, you need to follow these steps:

1. Get the marks for each of the five subjects.
2. Calculate the total marks obtained.
3. Calculate the maximum possible marks.
4. Calculate the percentage using the formula:

[\text{Percentage} = \left(\frac{\text{Total Marks Obtained}}{\text{Maximum Possible Marks}} \right) \times 100]

Here is a Python program to perform this calculation:

```
def calculate_percentage(marks):
    # Calculate the total marks obtained
    total_marks = sum(marks)
```

```
    # Calculate the maximum possible marks
    max_marks = len(marks) * 100  # Assuming each subject is out of 100 marks

    # Calculate the percentage
    percentage = (total_marks / max_marks) * 100

    return percentage

# Example usage
marks = []
for i in range(1, 6):
    mark = float(input(f"Enter marks for subject {i} (out of 100): "))
    marks.append(mark)

percentage = calculate_percentage(marks)
print(f"The percentage of marks obtained is: {percentage:.2f}%")
```

Explanation

1. Define calculate_percentage Function:
    - This function takes a list of marks for the five subjects.
    - It calculates the total marks obtained using the sum() function.
    - It calculates the maximum possible marks by multiplying the number of subjects by 100 (assuming each subject is out of 100 marks).
    - It calculates the percentage using the formula: [\text{Percentage} = \left(\frac{\text{Total Marks Obtained}}{\text{Maximum Possible Marks}} \right) \times 100]
    - It returns the calculated percentage.
2. User Input:
    - The program prompts the user to enter the marks for each of the five subjects.
    - The inputs are read as strings and then converted to floating-point numbers using float().
    - The marks are stored in a list called marks.
3. Calculate and Print:

- The program calls the calculate_percentage function with the list of marks.
- It prints the percentage of marks obtained, formatted to two decimal places.

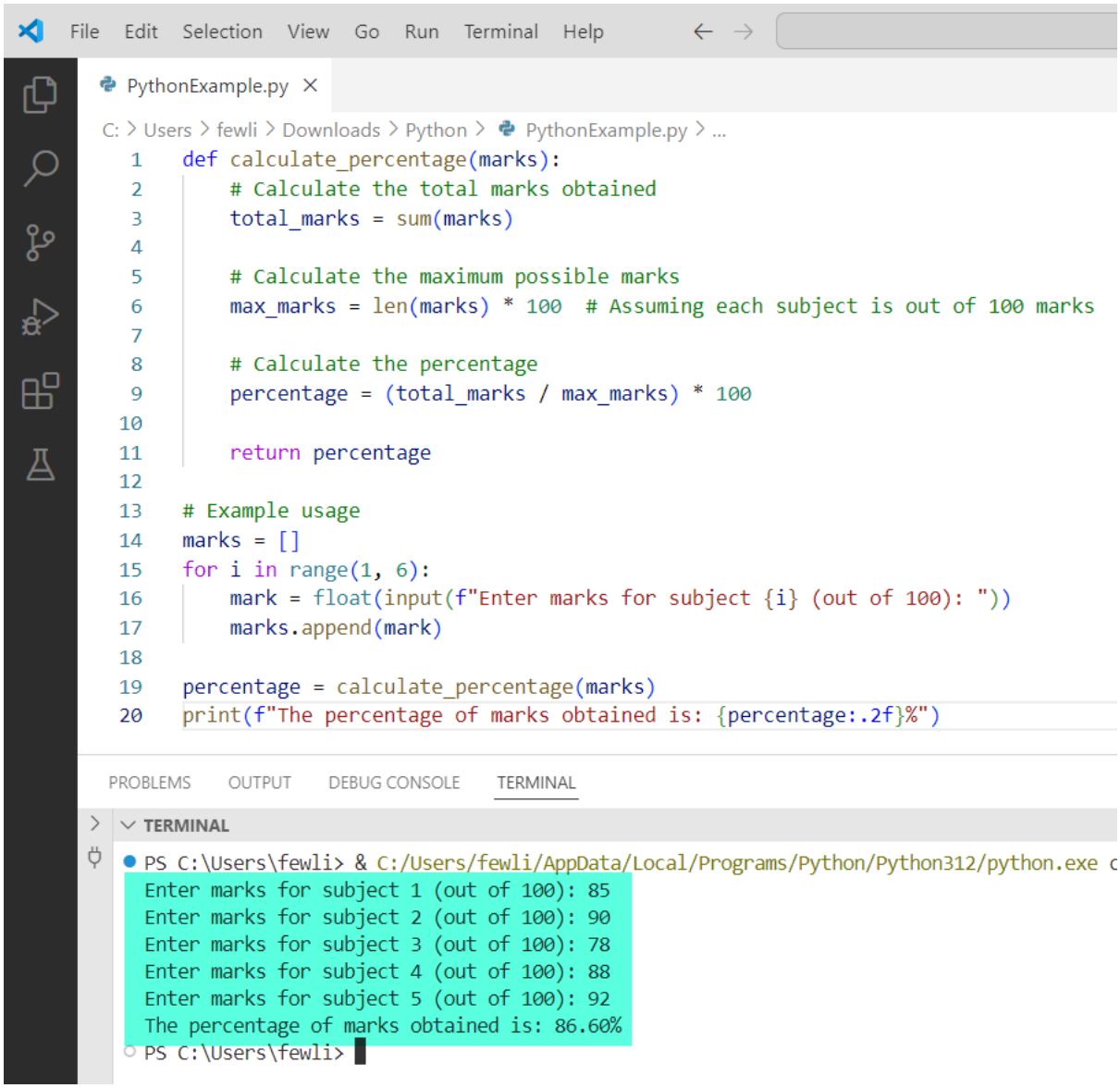
Example

Let's say the user inputs the following marks:

- Subject 1: 85
- Subject 2: 90
- Subject 3: 78
- Subject 4: 88
- Subject 5: 92
- The program will call calculate_percentage([85, 90, 78, 88, 92]).
- Inside the function:
  - It calculates the total marks obtained: [\text{Total Marks} = 85 + 90 + 78 + 88 + 92 = 433]
  - It calculates the maximum possible marks: [\text{Max Marks} = 5 \times 100 = 500]
  - It calculates the percentage: [\text{Percentage} = \left(\frac{433}{500} \right) \times 100 = 86.60]
- The output will be:

The percentage of marks obtained is: 86.60%

Here is the exact output in the screenshot below:



44# PYTHON PROGRAM TO FIND POWER OF A NUMBER

To find the power of a number in Python, you can use several methods, including the built-in ** operator, the pow() function, or the math.pow() function from the math module. Here, I'll explain a simple Python program using the ** operator to calculate the power of a number.

The formula to calculate the power of a number (a) raised to the power of (b) is:

[\text{result} = a^b]

Here is a Python program to perform this calculation:

```
def power(base, exponent):
    # Calculate the power using the ** operator
    result = base ** exponent
```

```
    return result

# Example usage
base = float(input("Enter the base number: "))
exponent = float(input("Enter the exponent: "))

result = power(base, exponent)
print(f"{base} raised to the power of {exponent} is {result:.2f}")
```

Explanation

1. Define power Function:
    - This function takes two arguments: base (the base number) and exponent (the exponent to which the base number is raised).
    - It calculates the power using the ** operator: [\text{result} = \text{base} ** \text{exponent}]
    - It returns the calculated result.
2. User Input:
    - The program prompts the user to enter the base number.
    - The input is read as a string and then converted to a floating-point number using float().
    - The program prompts the user to enter the exponent.
    - The input is read as a string and then converted to a floating-point number using float().
3. Calculate and Print:
    - The program calls the power function with the user-provided base and exponent.
    - It prints the result, formatted to two decimal places.

Example

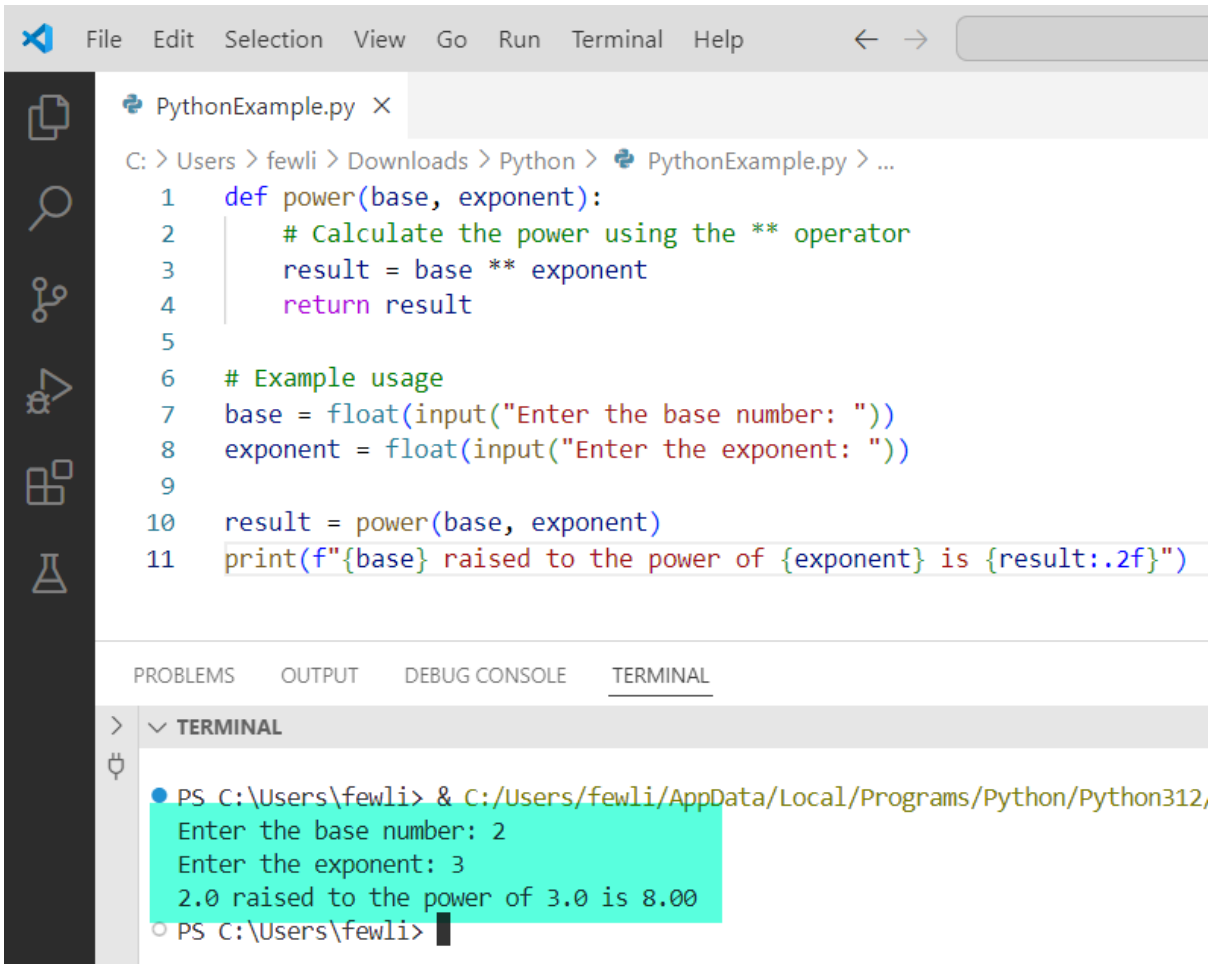
Let's say the user inputs the following values:

- Base number: 2
- Exponent: 3

- The program will call power(2, 3).
- Inside the function:
    - It calculates the power: [\text{result} = 2 ** 3 = 8]
- The output will be:

2.0 raised to the power of 3.0 is 8.00

Here is the exact output:



45# PYTHON PROGRAM TO FIND QUOTIENT AND REMAINDER

To find the quotient and remainder of two numbers in Python, you can use the // operator for integer division (quotient) and the % operator for the modulus (remainder). These operators are straightforward and provide an efficient way to perform the required calculations.

Here is a Python program to find the quotient and remainder of two numbers:

```
def find_quotient_and_remainder(dividend, divisor):
```

```
    # Calculate the quotient using integer division
    quotient = dividend // divisor

    # Calculate the remainder using the modulus operator
    remainder = dividend % divisor

    return quotient, remainder

# Example usage
dividend = int(input("Enter the dividend: "))
divisor = int(input("Enter the divisor: "))

quotient, remainder = find_quotient_and_remainder(dividend, divisor)
print(f"The quotient of {dividend} divided by {divisor} is {quotient}")
print(f"The remainder of {dividend} divided by {divisor} is {remainder}")
```

Explanation

1. Define find_quotient_and_remainder Function:
    - This function takes two arguments: dividend (the number to be divided) and divisor (the number by which to divide).
    - It calculates the quotient using the // operator (integer division): [\text{quotient} = \text{dividend} // \text{divisor}]
    - It calculates the remainder using the % operator (modulus): [\text{remainder} = \text{dividend} % \text{divisor}]
    - It returns both the quotient and the remainder.
2. User Input:
    - The program prompts the user to enter the dividend.
    - The input is read as a string and then converted to an integer using int().
    - The program prompts the user to enter the divisor.
    - The input is read as a string and then converted to an integer using int().
3. Calculate and Print:
    - The program calls the find_quotient_and_remainder function with the user-provided dividend and divisor.

- It prints the quotient and the remainder.

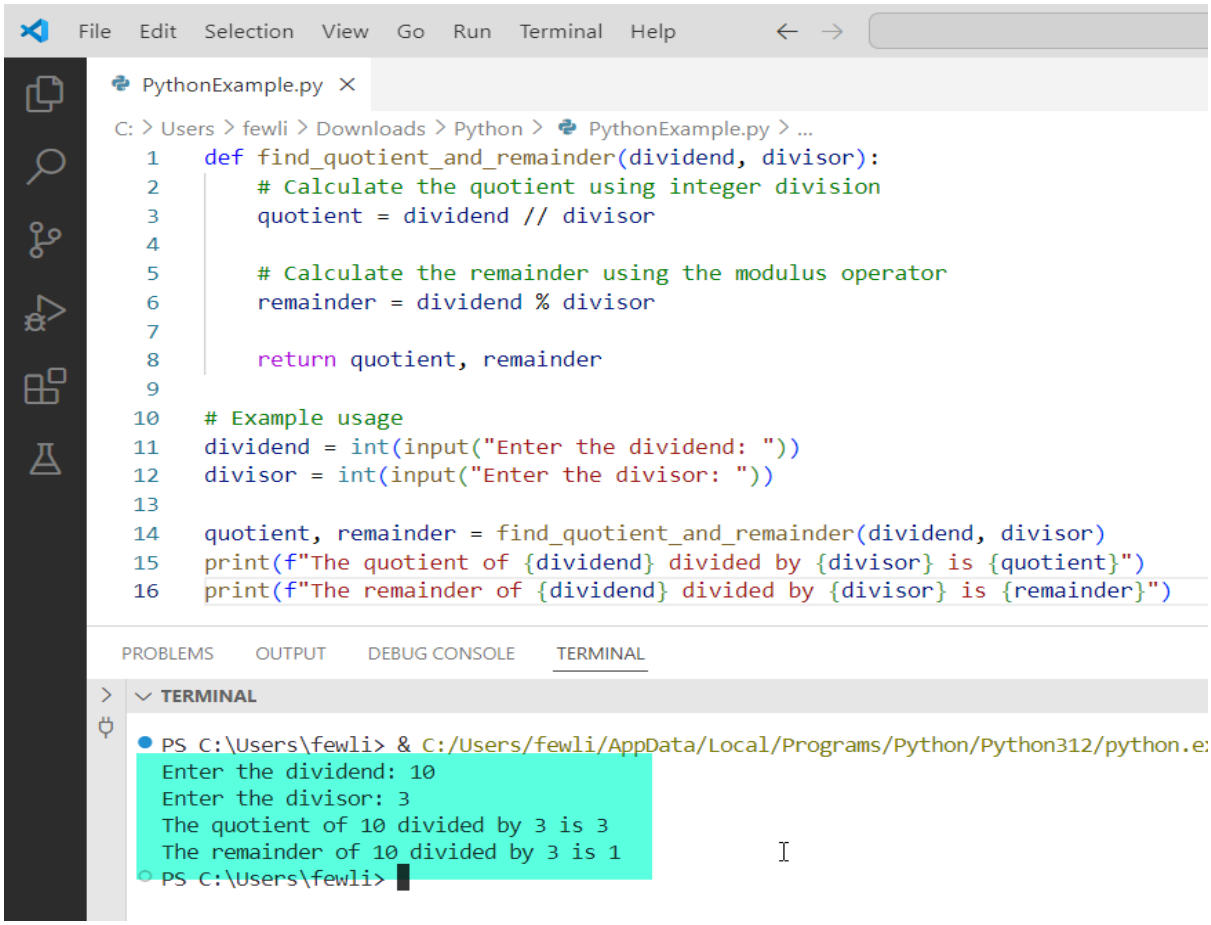
Example

Let's say the user inputs the following values:

- Dividend: 10
- Divisor: 3
- The program will call find_quotient_and_remainder(10, 3).
- Inside the function:
  - It calculates the quotient: [\text{quotient} = 10 // 3 = 3]
  - It calculates the remainder: [\text{remainder} = 10 % 3 = 1]
- The output will be:
- The quotient of 10 divided by 3 is 3

The remainder of 10 divided by 3 is 1

Here is the exact output:



46# PYTHON PROGRAM TO FIND OUT LEAP YEAR

To determine whether a given year is a leap year in Python, you need to follow these rules:

1. A year is a leap year if it is divisible by 4.
2. However, if the year is also divisible by 100, it is not a leap year unless it is also divisible by 400.

These rules can be summarized as:

- A year is a leap year if it is divisible by 4 and not divisible by 100, or it is divisible by 400.

Here is a Python program to check if a given year is a leap year:

```
def is_leap_year(year):
    # Check if the year is divisible by 4 but not by 100, or it is divisible
by 400
    if (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0):
        return True
    else:
        return False

# Example usage
year = int(input("Enter a year: "))

if is_leap_year(year):
    print(f"{year} is a leap year.")
else:
    print(f"{year} is not a leap year.")
```

Explanation

1. Define is_leap_year Function:
    - This function takes one argument: year.
    - It checks if the year is a leap year using the conditions:
        - The year is divisible by 4 and not divisible by 100, or
        - The year is divisible by 400.
    - If either condition is met, the function returns True, indicating it is a leap year.
    - Otherwise, it returns False.

2. User Input:
    - The program prompts the user to enter a year.
    - The input is read as a string and then converted to an integer using int().
3. Check and Print:
    - The program calls the is_leap_year function with the user-provided year.
    - It prints whether the year is a leap year or not based on the function's return value.

Example

Let's say the user inputs the year 2024.

- The program will call is_leap_year(2024).
- Inside the function:
    - It checks the conditions:
        - 2024 is divisible by 4 (2024 % 4 == 0 is True).
        - 2024 is not divisible by 100 (2024 % 100 != 0 is True).
    - Since both conditions are met, the function returns True.
- The output will be:
- 2024 is a leap year.

Let's consider another example where the user inputs the year 1900.

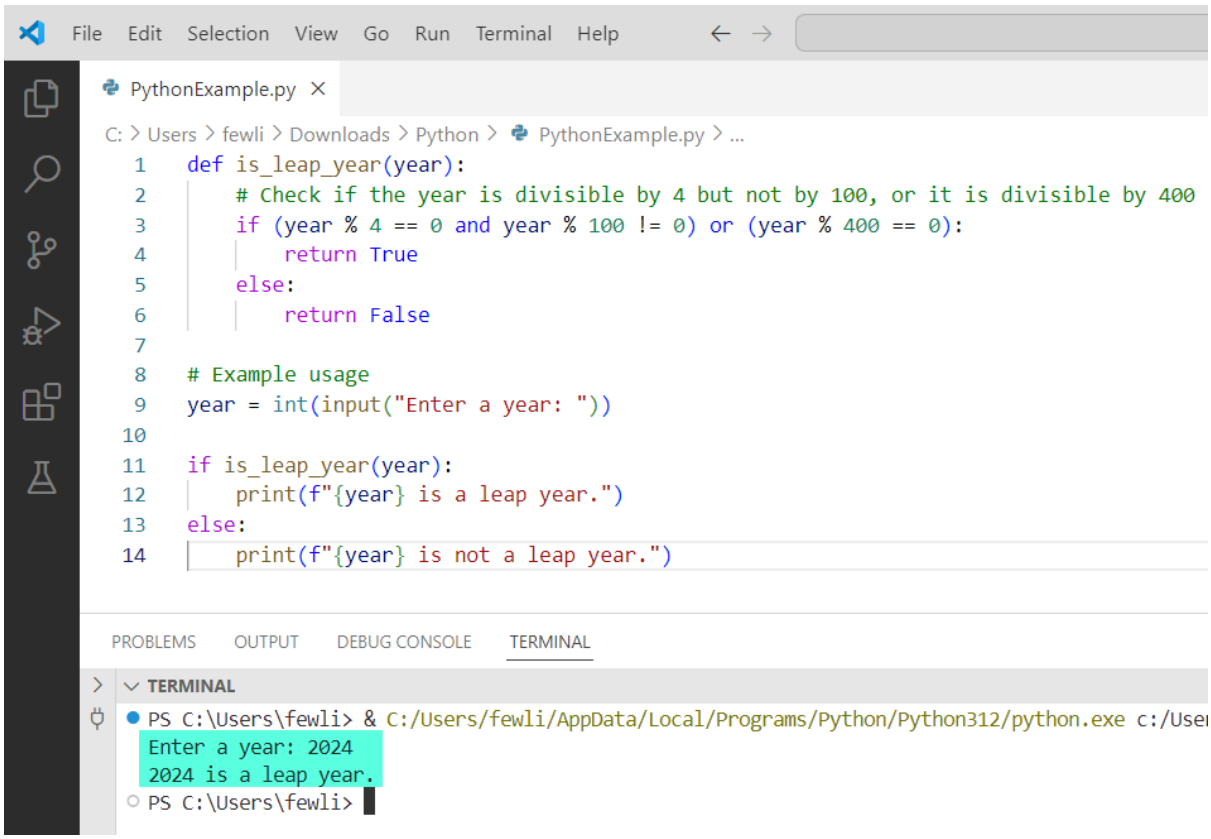
- The program will call is_leap_year(1900).
- Inside the function:
    - It checks the conditions:
        - 1900 is divisible by 4 (1900 % 4 == 0 is True).
        - 1900 is divisible by 100 (1900 % 100 == 0 is True).
        - 1900 is not divisible by 400 (1900 % 400 != 0 is True).

- Since 1900 is divisible by 100 but not by 400, the function returns False.

- The output will be:

1900 is not a leap year.

Here is the output:



47# PYTHON PROGRAM TO FIND CURRENT DATE AND TIME

To find the current date and time in Python, you can use the datetime module, which provides various classes to manipulate dates and times. The datetime.datetime class includes methods to get the current date and time.

Here is a Python program to find and display the current date and time:

```
from datetime import datetime

def get_current_date_time():
    # Get the current date and time
    now = datetime.now()

    # Format the date and time as a string
    current_date_time = now.strftime("%Y-%m-%d %H:%M:%S")
```

```
    return current_date_time

# Example usage
current_date_time = get_current_date_time()
print(f"The current date and time is: {current_date_time}")
```

Explanation

1. Import datetime Module:
    - The datetime module provides classes for manipulating dates and times.
    - Specifically, datetime.now() returns the current local date and time.
2. Define get_current_date_time Function:
    - This function gets the current date and time using datetime.now().
    - It formats the date and time as a string using the strftime method, which formats a datetime object to a string according to the specified format codes.
    - The format "%Y-%m-%d %H:%M:%S" represents:
        - %Y: Year with century (e.g., 2024)
        - %m: Month as a zero-padded decimal number (e.g., 01 for January)
        - %d: Day of the month as a zero-padded decimal number (e.g., 01)
        - %H: Hour (24-hour clock) as a zero-padded decimal number (e.g., 14 for 2 PM)
        - %M: Minute as a zero-padded decimal number (e.g., 30)
        - %S: Second as a zero-padded decimal number (e.g., 59)
    - It returns the formatted current date and time as a string.
3. Get and Print Current Date and Time:

- The program calls the get_current_date_time function to get the current date and time.
- It prints the result.

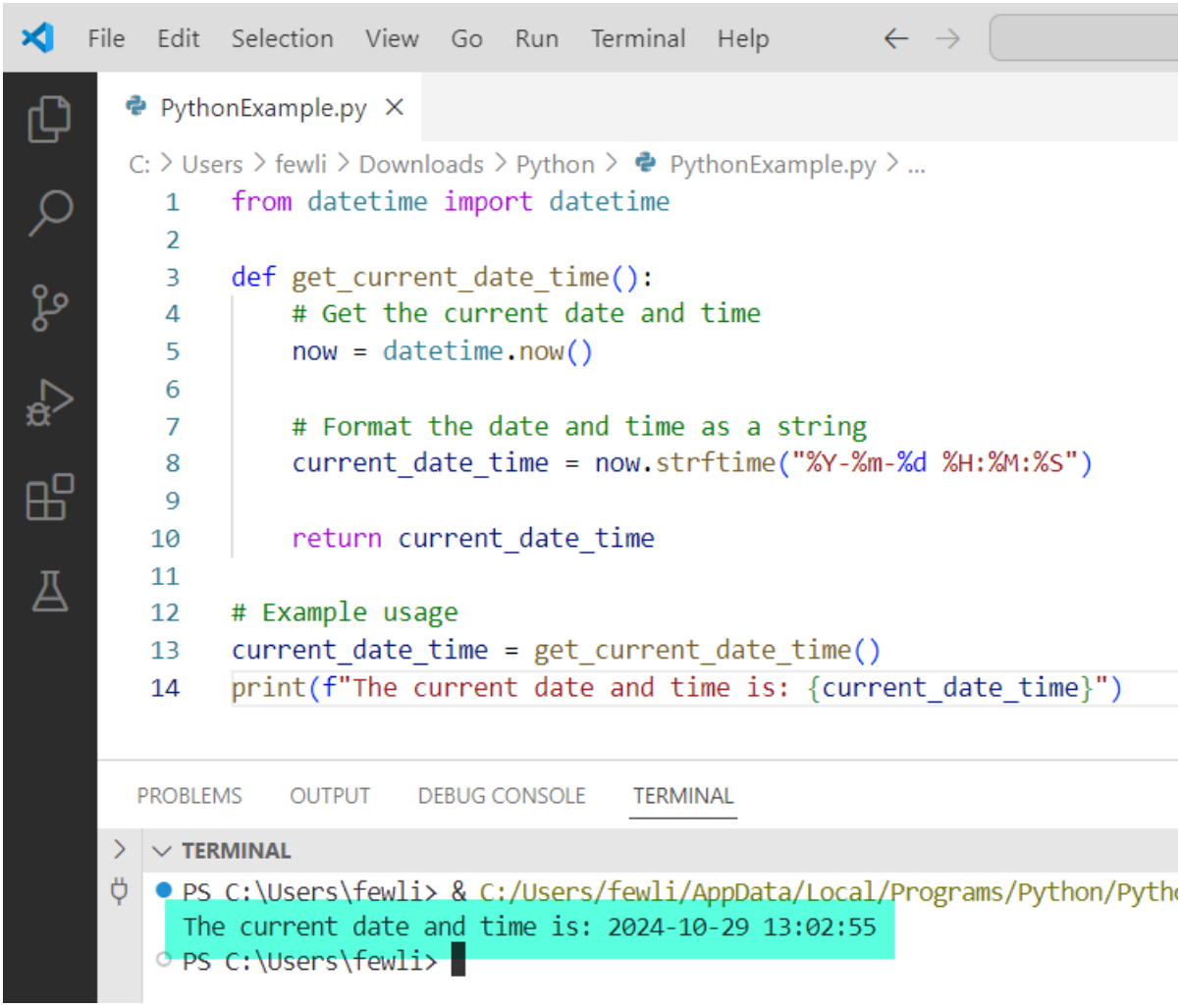
Example

When you run the program, it will output the current date and time formatted as specified. For example:

The current date and time is: 2024-10-29 14:30:45

This output will vary depending on the exact moment you run the program.

Here is the exact output:



48# PYTHON PROGRAM TO GET DAYS BETWEEN TWO DATES

To calculate the number of days between two dates in Python, you can use the datetime module, which provides classes for manipulating dates and times. Specifically, you can create datetime.date objects for the two dates

and then subtract them to get a datetime.timedelta object, which represents the difference between the two dates.

Here is a Python program to calculate the number of days between two dates:

```
from datetime import date

def days_between_dates(date1, date2):
    # Convert the input strings to date objects
    date1 = date(*map(int, date1.split('-')))
    date2 = date(*map(int, date2.split('-')))

    # Calculate the difference between the two dates
    delta = date2 - date1

    # Return the difference in days
    return abs(delta.days)

# Example usage
date1 = input("Enter the first date (YYYY-MM-DD): ")
date2 = input("Enter the second date (YYYY-MM-DD): ")

days_difference = days_between_dates(date1, date2)
print(f"The number of days between {date1} and {date2} is {days_difference}
days.")
```

Explanation

1. Import date Class from datetime Module:
    - The date class is used to create date objects.
2. Define days_between_dates Function:
    - This function takes two arguments: date1 and date2, which are strings representing the dates in the format YYYY-MM-DD.
    - It converts the input strings to date objects using the date class:
        - date(*map(int, date1.split('-'))) splits the string by the hyphen, maps each part to an integer, and unpacks the integers as arguments to the date constructor.

- It calculates the difference between the two dates by subtracting one date object from the other, resulting in a datetime.timedelta object.
- It returns the absolute value of the difference in days using abs(delta.days) to ensure the result is always positive.

3. User Input:
    - The program prompts the user to enter the first date in the format YYYY-MM-DD.
    - The input is read as a string.
    - The program prompts the user to enter the second date in the same format.
    - The input is read as a string.
4. Calculate and Print:
    - The program calls the days_between_dates function with the user-provided dates.
    - It prints the number of days between the two dates.

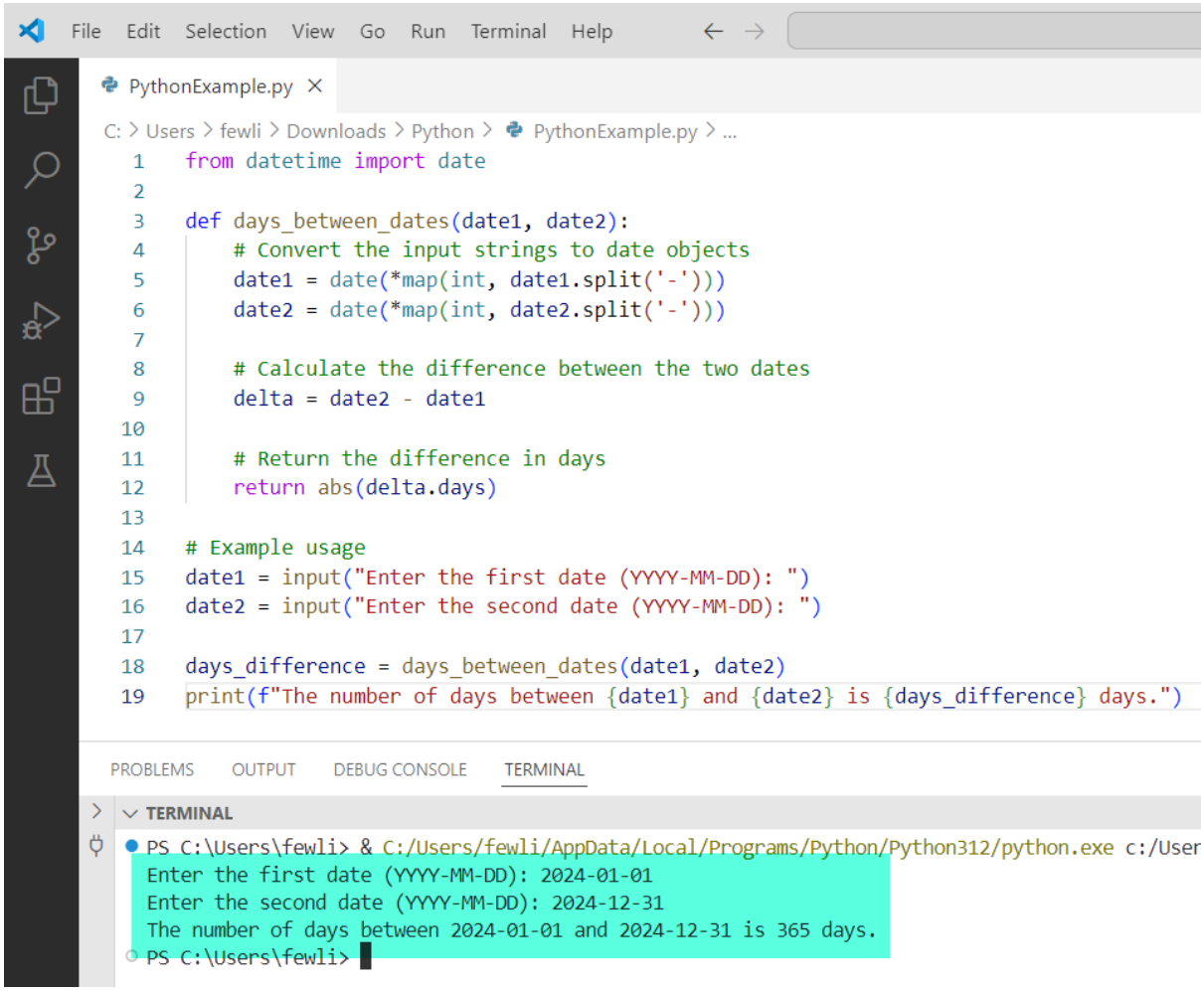
Example

Let's say the user inputs the following dates:

- First date: 2024-01-01
- Second date: 2024-12-31
- The program will call days_between_dates("2024-01-01", "2024-12-31").
- Inside the function:
    - It converts the strings to date objects: [\text{date1} = \text{date}(2024, 1, 1)] [\text{date2} = \text{date}(2024, 12, 31)]
    - It calculates the difference between the two dates: [\text{delta} = \text{date2} - \text{date1} = \text{date}(2024, 12, 31) - \text{date}(2024, 1, 1)] [\text{delta.days} = 364]
- The output will be:

The number of days between 2024-01-01 and 2024-12-31 is 364 days.

Here is the exact output in the screenshot below:



49# PYTHON PROGRAM TO CALCULATE AGE FROM DATE OF BIRTH

To calculate a person's age from their date of birth in Python, you can use the datetime module. This module provides classes for manipulating dates and times, making it easy to calculate the difference between the current date and the date of birth.

Here is a Python program to calculate age from a date of birth:

```
from datetime import date

def calculate_age(birthdate):
    # Get today's date
    today = date.today()

    # Calculate the difference in years
    age = today.year - birthdate.year
```

```
    # Adjust the age if the birthdate has not occurred yet this year
    if (today.month, today.day) < (birthdate.month, birthdate.day):
        age -= 1

    return age

# Example usage
birth_year = int(input("Enter your birth year (e.g., 1990): "))
birth_month = int(input("Enter your birth month (1-12): "))
birth_day = int(input("Enter your birth day (1-31): "))

birthdate = date(birth_year, birth_month, birth_day)
age = calculate_age(birthdate)
print(f"Your age is: {age} years")
```

Explanation

1. Import date Class from datetime Module:
    - The date class is used to create date objects.
2. Define calculate_age Function:
    - This function takes one argument: birthdate, which is a date object representing the person's date of birth.
    - It gets today's date using date.today().
    - It calculates the initial age by subtracting the birth year from the current year: [\text{age} = \text{today.year} - \text{birthdate.year}]
    - It adjusts the age if the birthdate has not occurred yet this year by checking if today's month and day are less than the birthdate's month and day: [\text{if (today.month, today.day) < (birthdate.month, birthdate.day):}] [\text{age} -= 1]
    - It returns the calculated age.
3. User Input:
    - The program prompts the user to enter their birth year, month, and day.
    - The inputs are read as strings and then converted to integers using int().

4. Create birthdate Object:
    - The program creates a date object for the birthdate using the provided year, month, and day: [\text{birthdate} = \text{date}(birth_year, birth_month, birth_day)]
5. Calculate and Print Age:
    - The program calls the calculate_age function with the birthdate object.
    - It prints the calculated age.

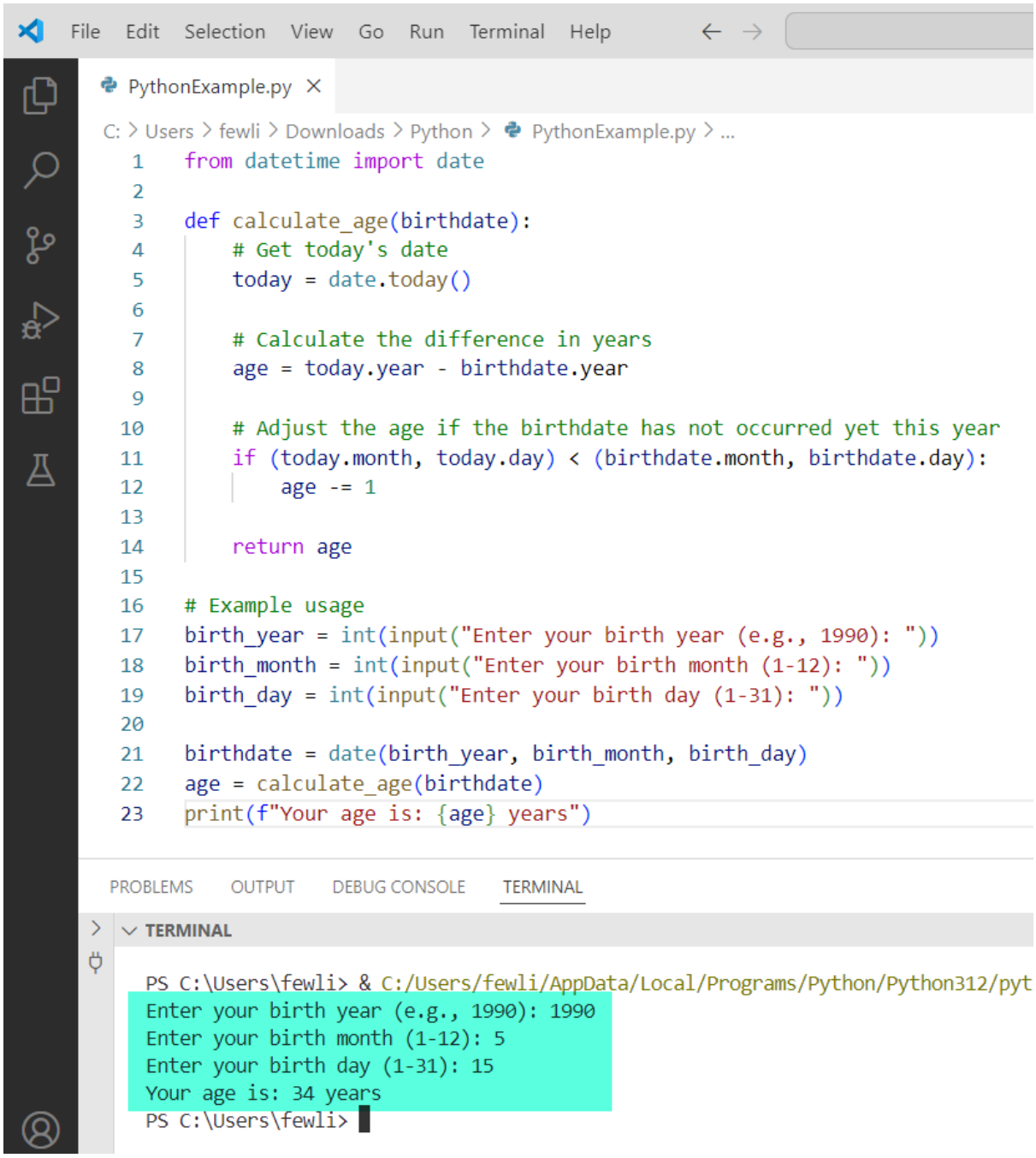
Example

Let's say the user inputs the following birthdate:

- Birth year: 1990
- Birth month: 5
- Birth day: 15
- The program will call calculate_age(date(1990, 5, 15)).
- Inside the function:
    - It gets today's date, for example, date(2024, 10, 29).
    - It calculates the initial age: [\text{age} = 2024 - 1990 = 34]
    - It checks if the birthdate has occurred yet this year: [\text{if (10, 29) < (5, 15):} \quad \text{(False)}]
    - Since the condition is false, the age remains 34.
- The output will be:

Your age is: 34 years

Here is the exact output:



50# PYTHON PROGRAM TO PRINT YESTERDAY TODAY TOMORROW

To print the dates for yesterday, today, and tomorrow in Python, you can use the datetime module, which provides classes for manipulating dates and times. Specifically, you can use the datetime.date class and the timedelta class to perform date arithmetic.

Here is a Python program to print the dates for yesterday, today, and tomorrow:

```python
from datetime import date, timedelta

def print_yesterday_today_tomorrow():
    # Get today's date
    today = date.today()

    # Calculate yesterday's date
    yesterday = today - timedelta(days=1)

    # Calculate tomorrow's date
    tomorrow = today + timedelta(days=1)

    # Print the dates
    print(f"Yesterday's date was: {yesterday}")
    print(f"Today's date is: {today}")
    print(f"Tomorrow's date will be: {tomorrow}")

# Example usage
print_yesterday_today_tomorrow()
```

Explanation

1. Import date and timedelta Classes from datetime Module:
    - The date class is used to create date objects representing specific dates.
    - The timedelta class is used to represent the difference between two dates or to perform date arithmetic.
2. Define print_yesterday_today_tomorrow Function:
    - This function calculates and prints the dates for yesterday, today, and tomorrow.
    - It gets today's date using date.today().
    - It calculates yesterday's date by subtracting one day from today's date using timedelta(days=1): [\text{yesterday} = \text{today} - \text{timedelta(days=1)}]
    - It calculates tomorrow's date by adding one day to today's date using timedelta(days=1): [\text{tomorrow} = \text{today} + \text{timedelta(days=1)}]
    - It prints the calculated dates.

3. Example Usage:

    - The program calls the print_yesterday_today_tomorrow function to print the dates for yesterday, today, and tomorrow.

Example Output

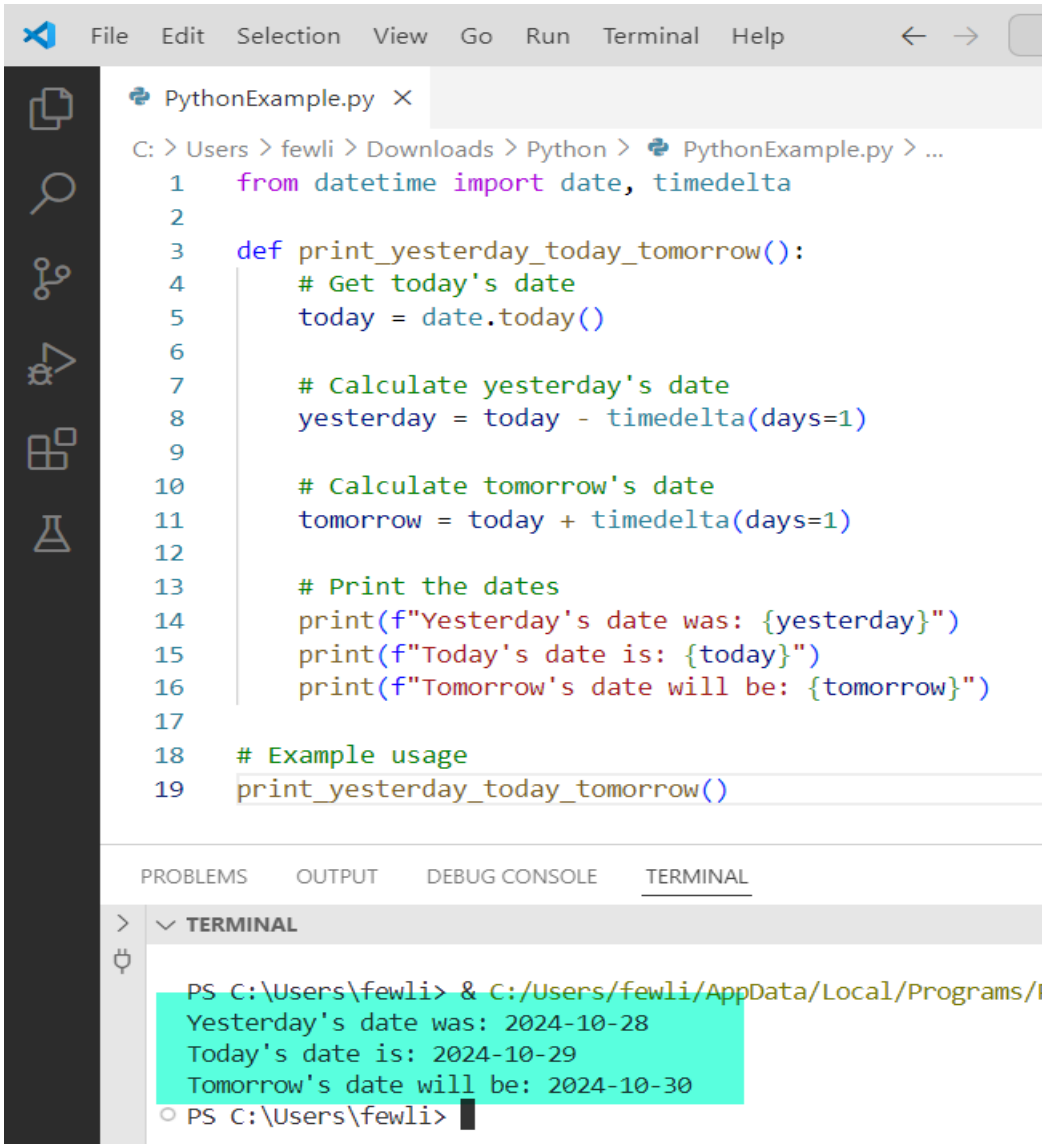
When you run the program, it will output the dates for yesterday, today, and tomorrow. For example, if today is October 29, 2024, the output will be:

Yesterday's date was: 2024-10-28

Today's date is: 2024-10-29

Tomorrow's date will be: 2024-10-30

Here is the exact output:



51# PYTHON PROGRAM TO PRINT DAYS OF THE WEEK

To print the days of the week in Python, use the calendar module, which provides useful functions for working with dates and times. The calendar module has a day_name attribute that contains the names of the days of the week.

Here is a Python program using the calendar module:

```
import calendar

def print_days_of_the_week():
    # Get the day names from the calendar module
    days_of_week = list(calendar.day_name)

    # Iterate over the list and print each day
    for day in days_of_week:
        print(day)

# Example usage
print_days_of_the_week()
```

Explanation

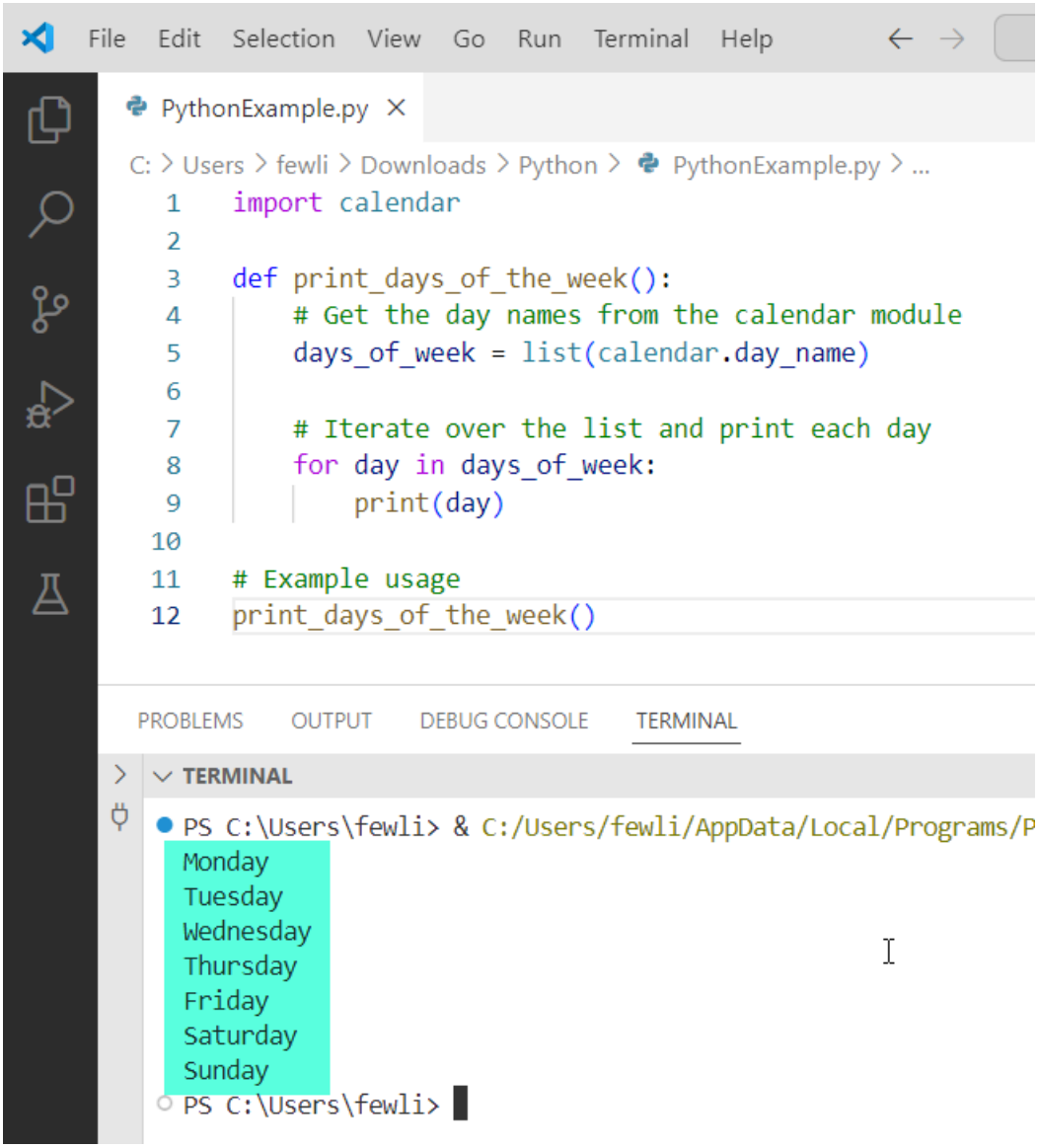
1. Import calendar Module:
    - The calendar module provides functions and classes for working with dates and calendars.
2. Define print_days_of_the_week Function:
    - This function retrieves the names of the days of the week from calendar.day_name, which is an iterable containing the names of the days of the week.
    - It converts calendar.day_name to a list called days_of_week.
    - It uses a for loop to iterate over each element in the days_of_week list.
    - Inside the loop, it prints each day.
3. Example Usage:
    - The program calls the print_days_of_the_week function to print the days of the week.

Example Output

When you run the program, it will output the same result as the previous method:

- Monday
- Tuesday
- Wednesday
- Thursday
- Friday
- Saturday
- Sunday

You can see the exact output:



CONCLUSION

I hope you liked all these Python programs. Do check out more Python tutorials on our website: https://pythonguides.com