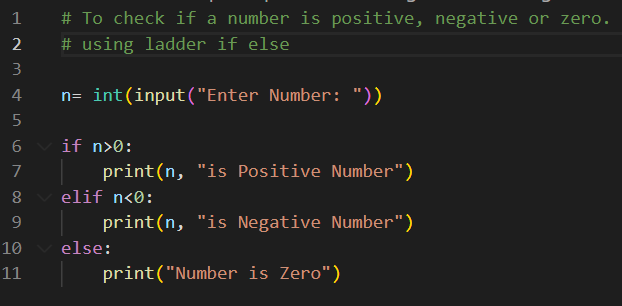
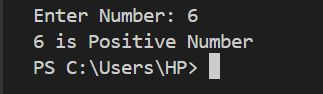
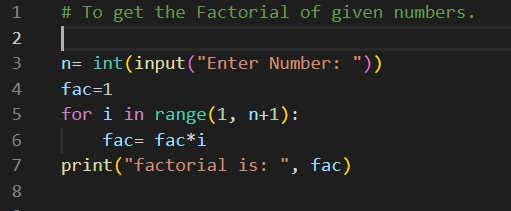
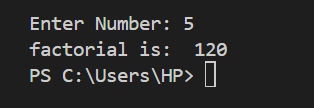
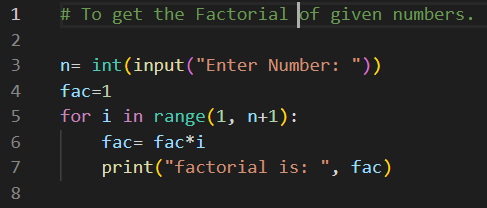
**PYTHON**

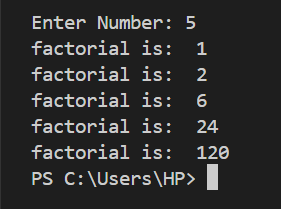
**Module – 2 (Fundamentals of python)**

1. **Write a Python program to check if a number is positive, negative or zero.**

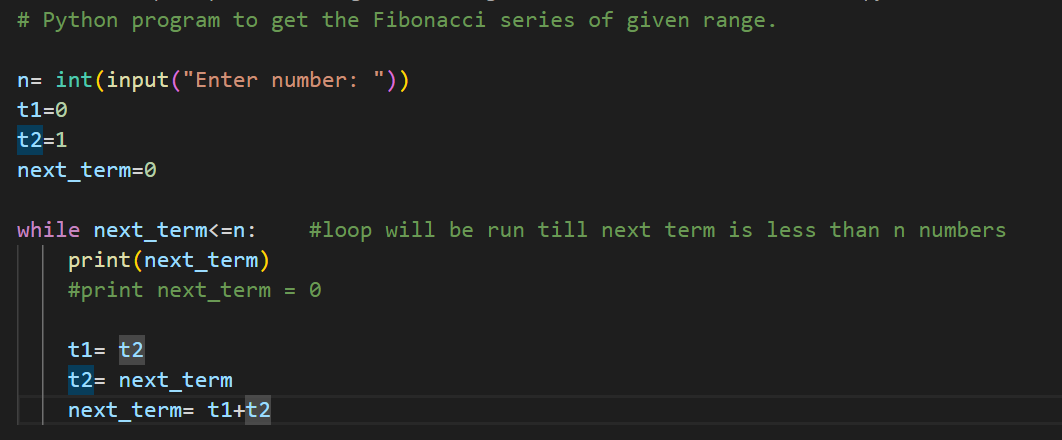
** Output:**

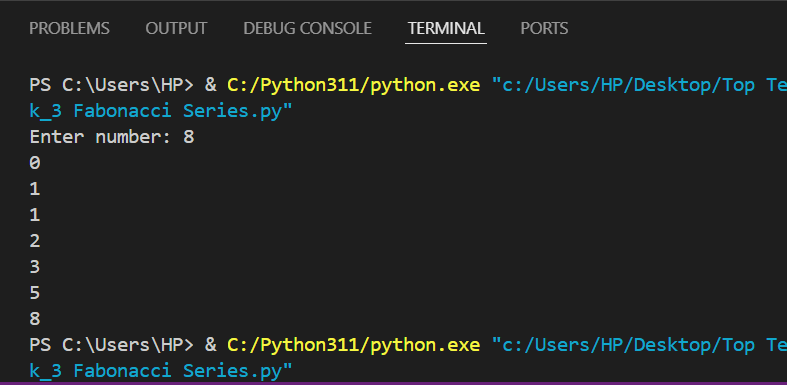
1. **Write a Python program to get the Factorial number of given numbers.**

**Output:**

**Output:**

1. **Write a Python program to get the Fibonacci series of given range.**

**Output:**

****

1. **How memory is managed in Python?**

Memory management is very important for software developers to work efficiently with any programming language. As we know, Python is a famous and widely used programming language. It is used almost in every technical domain. In contrast to a programming language, memory management is related to writing memory-efficient code. We cannot overlook the importance of memory management while implementing a large amount of data. Improper memory management leads to slowness on the application and the server-side components. It also becomes the reason of improper working. If the memory is not handled well, it will take much time while preprocessing the data.

In [Python](https://www.javatpoint.com/python-tutorial), memory is managed by the Python manager which determines where to put the application data in the memory. So, we must have the knowledge of Python memory manager to write efficient code and maintainable code.

Let's assume memory looks like an empty book and we want to write anything on the book's page. Then, we write data any data the manager find the free space in the book and provide it to the application. The procedure of providing memory to objects is called **allocation.**

On the other side, when data is no longer use, it can be deleted by the Python memory manager. But the question is, how? And where did this memory come from?

## Python Memory Allocation

Memory allocation is an essential part of the memory management for a developer. This process basically allots free space in the computer's virtual memory, and there are two types of virtual memory works while executing programs.

* Static Memory Allocation
* Dynamic Memory Allocation

### Static Memory Allocation -

Static memory allocation happens at the compile time. For example - In [C](https://www.javatpoint.com/c-programming-language-tutorial)/[C++](https://www.javatpoint.com/cpp-tutorial), we declare a static array with the fixed sizes. Memory is allocated at the time of compilation. However, we cannot use the memory again in the further program.

1. **static** **int** a=10;

**Stack Allocation**

The Stack data structure is used to store the static memory. It is only needed inside the particular function or method call. The function is added in program's call stack whenever we call it. Variable assignment inside the function is temporarily stored in the function call stack; the function returns the value, and the call stack moves to the text task. The compiler handles all these processes, so we don't need to worry about it.

Call stack (stack data structure) holds the program's operational data such as subroutines or function call in the order they are to be called. These functions are popped up from the stack when we called.

### Dynamic Memory Allocation

Unlike static memory allocation, Dynamic memory allocates the memory at the runtime to the program. For example - In C/C++, there is a predefined size of the integer of float data type but there is no predefine size of the data types. Memory is allocated to the objects at the run time. We use the [**Heap**](https://www.javatpoint.com/heap-sort-in-python) for implement dynamic memory management. We can use the memory throughout the program.

1. **int** \*a;
2. p = **new** **int**;

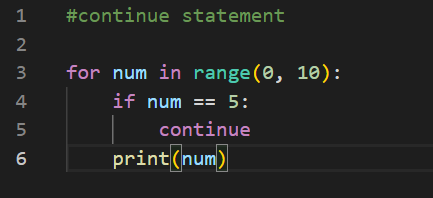
As we know, everything in Python is an object means dynamic memory allocation inspires the Python memory management. Python memory manager automatically vanishes when the object is no longer in use.

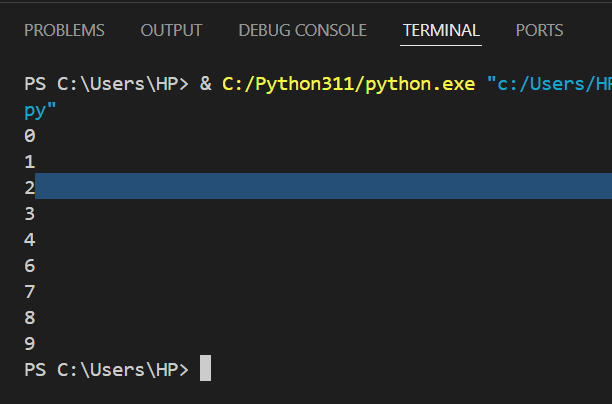
**Heap Memory Allocation**

Heap data structure is used for dynamic memory which is not related to naming counterparts. It is type of memory that uses outside the program at the global space. One of the best advantages of heap memory is to it freed up the memory space if the object is no longer in use or the node is deleted.

1. **What is the purpose continue statement in python?**

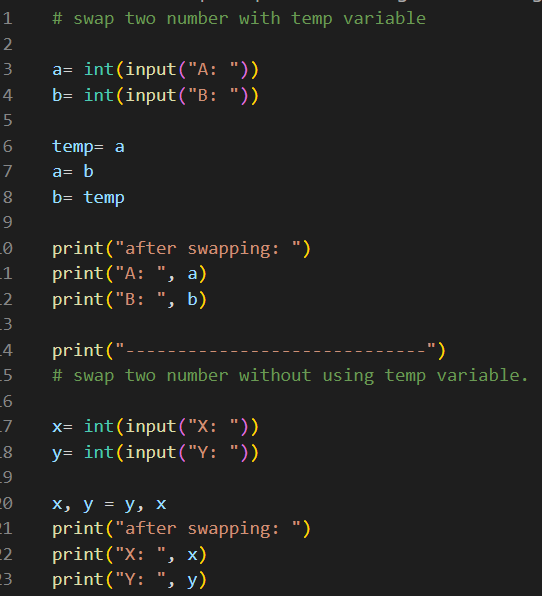
The continue statement is used to skip the remaining code inside a loop for the current iteration only.

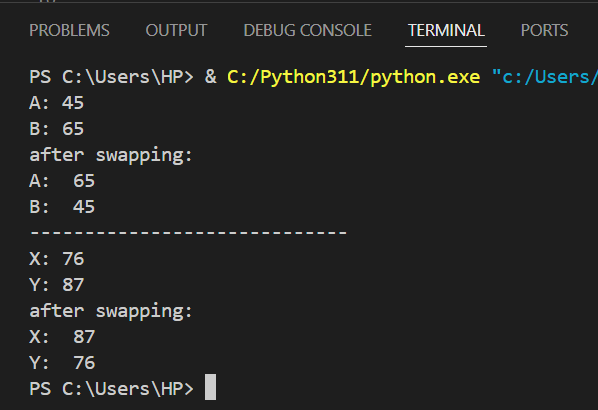
For instance, let’s use continue instead of a break statement

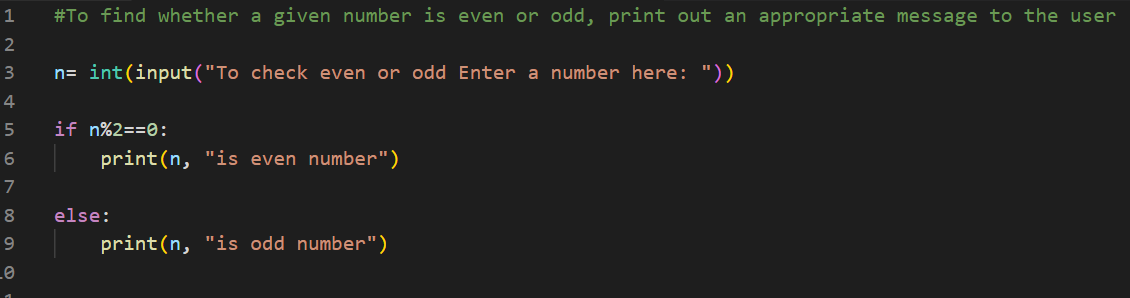
 **Output:**

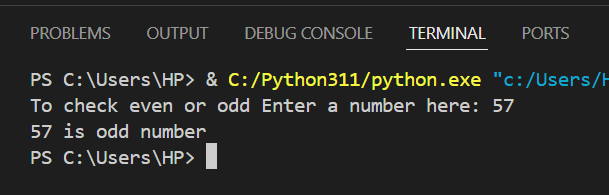
When the condition num == 5 becomes True, the continue statement gets executed. The remaining code in the loop is skipped only for that iteration. That’s why Iteration: 5 is missing from the above output.

Therefore, the continue statement works opposite to the break statement. Instead of terminating the loop, it forces it to execute the next iteration of the loop.

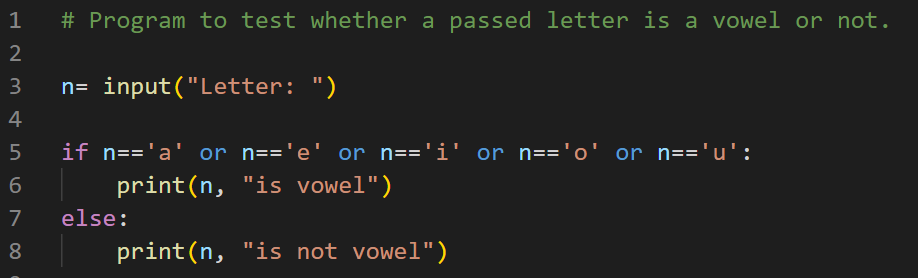
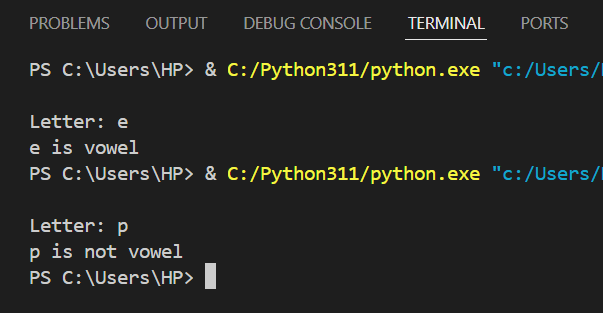
1. **Write python program that swap two number with temp variable and without temp variable.**

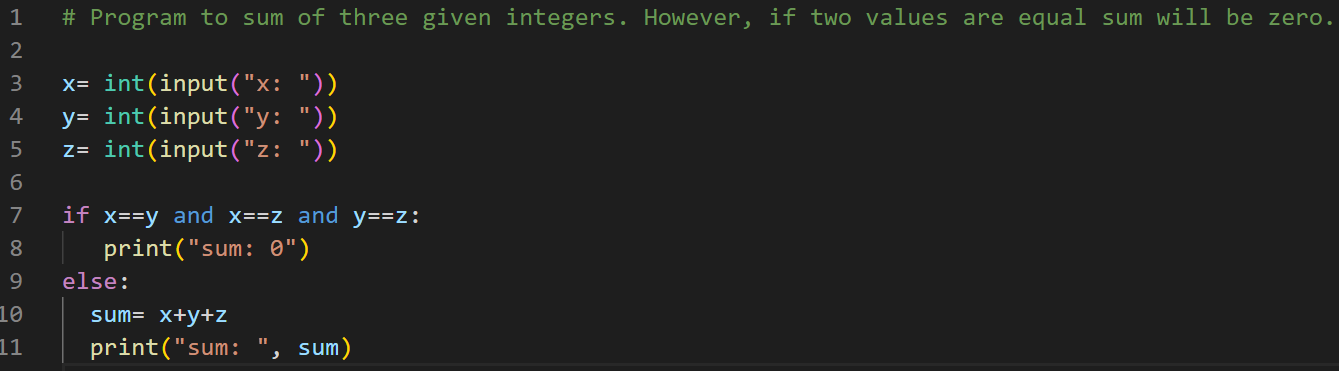
**Output:**

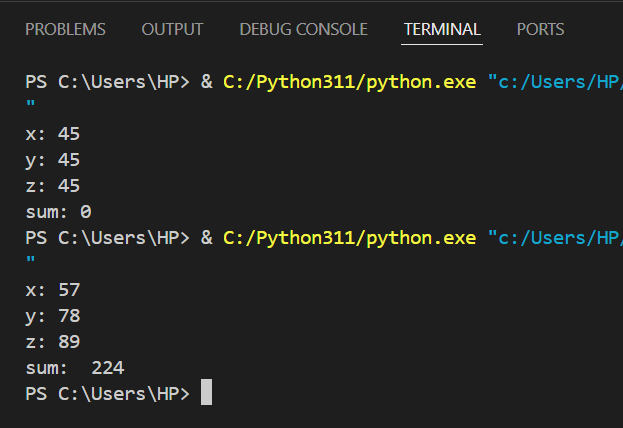
1. **Write a Python program to find whether a given number is even or odd, print out an appropriate message to the user.**

**Output:**

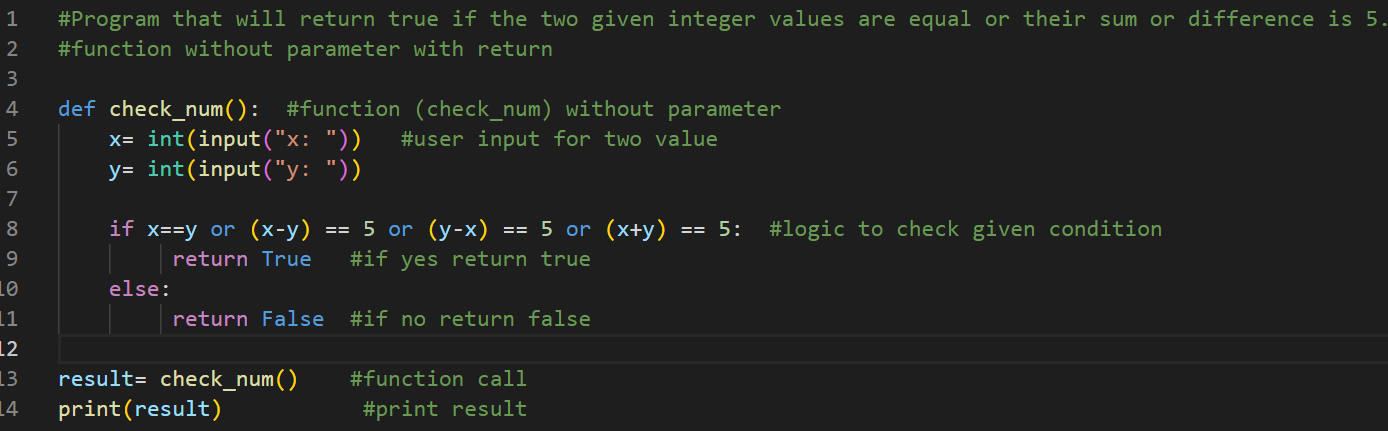
1. **Write a Python program to test whether a passed letter is a vowel or not.**

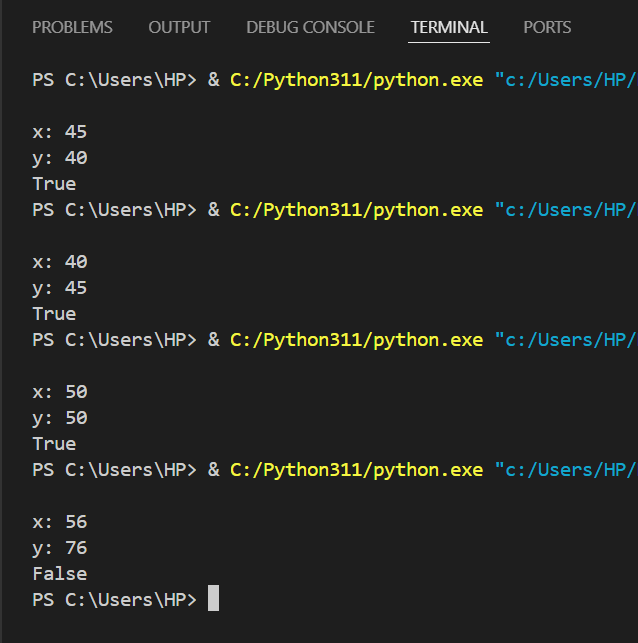
**Output:**

1. **Write a Python program to sum of three given integers. However, if two values are equal sum will be zero.**

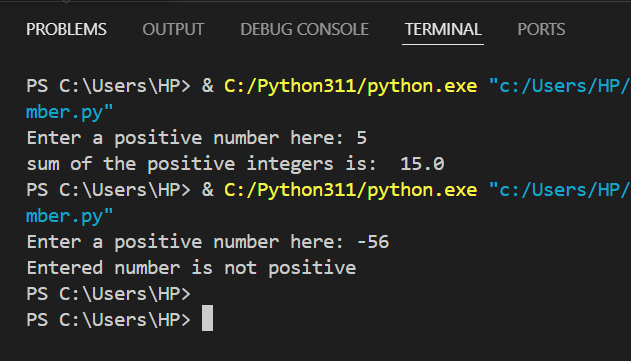
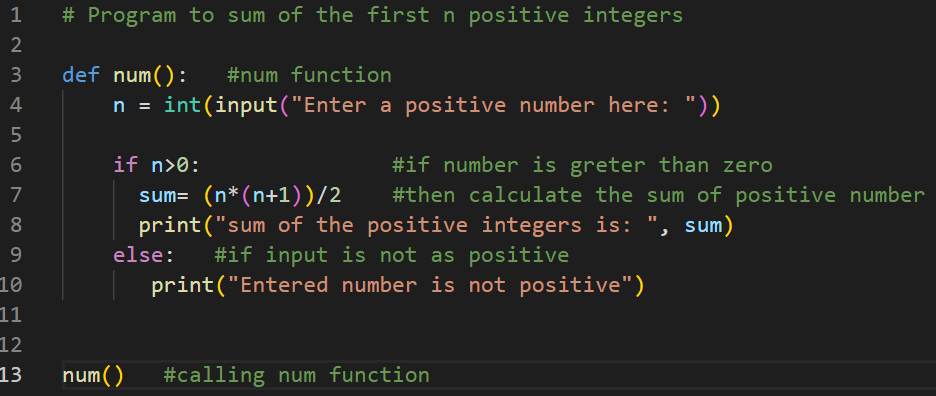
**Output:**

1. **Write a Python program that will return true if the two given integer values are equal or their sum or difference is 5.**

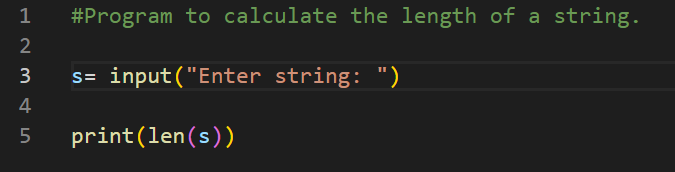
**Output:**

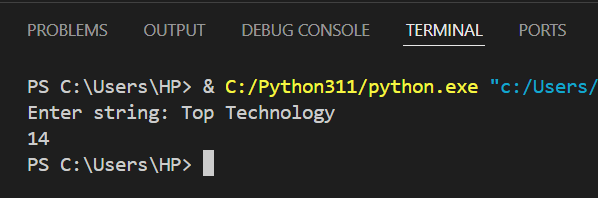


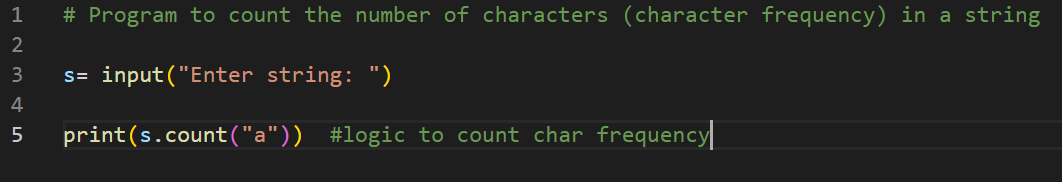
1. **Write a python program to sum of the first n positive integers.**

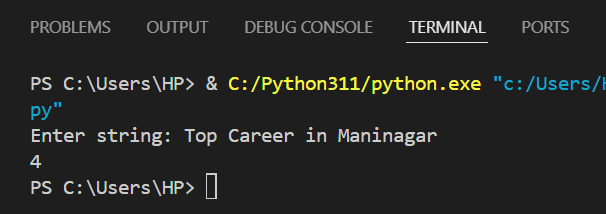
**Output:**

1. **Write a Python program to calculate the length of a string.**

**Output:**

****

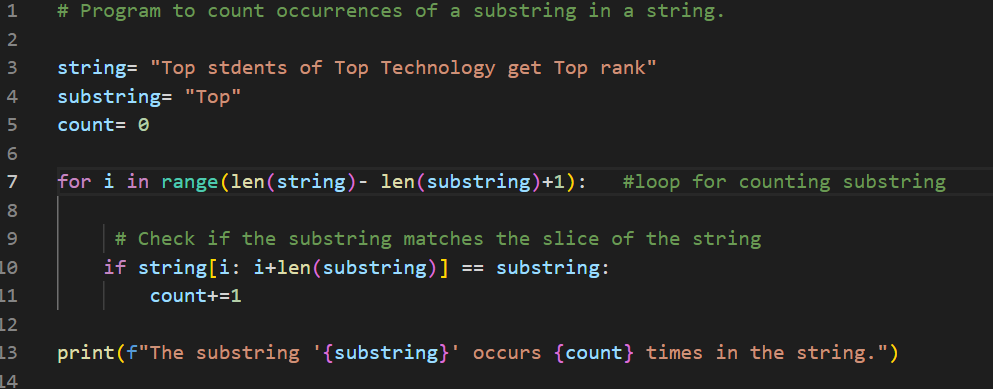
1. ** Write a Python program to count the number of characters (character frequency) in a string.**

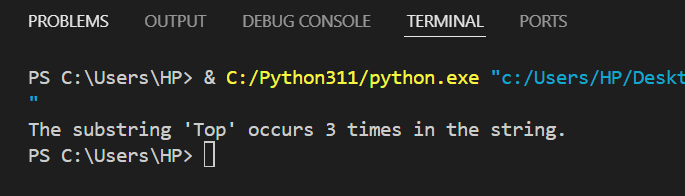
**Output:**

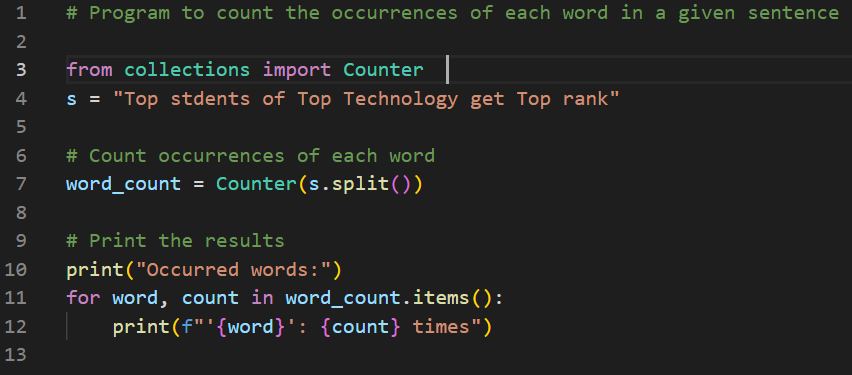
1. **What are negative indexes and why are they used?**

Negative Indexing is used into python to begin slicing from the end of the string. Slicing in python gets a sub-string from a string. The slicing range is set as parameters like start, stop and step. also, we can say that it is a powerful feature that allows us to access elements in a list from the end instead of beginning. This can be particularly useful when dealing with large lists or when we need to access the last few elements without knowing their exact position.

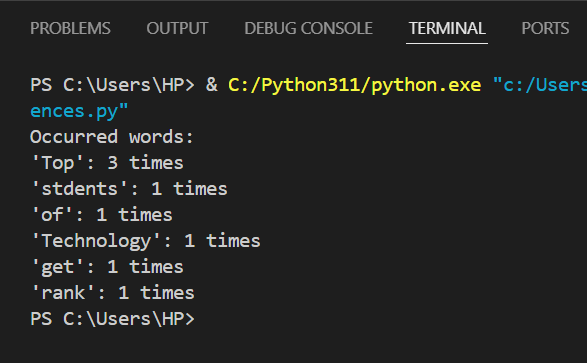
**Negative Indexing is Useful because of:**

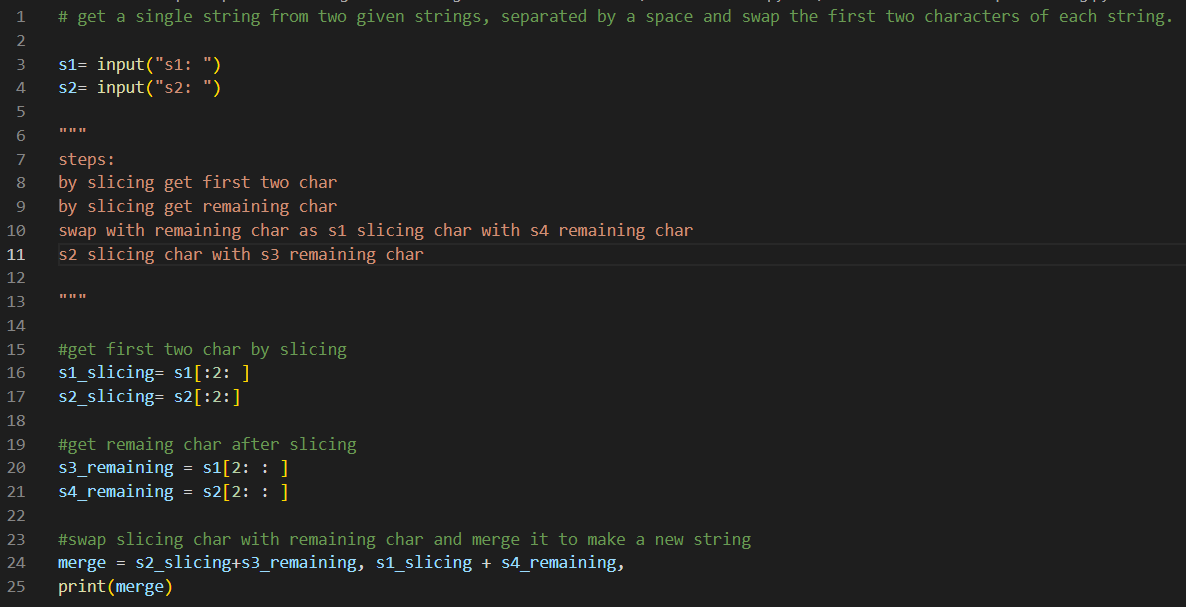
1. **Accessing Last Elements Easily**: Negative indexing provides a quick way to access the last elements without needing the exact length of the list or sequence.
2. **Simplifying Code**: Instead of calculating len(sequence) - 1 to get the last element, sequence[-1] achieves this directly.
3. **Flexible Slicing**: Negative indexing is particularly helpful with slicing, as it allows specifying ranges from the end of a list without knowing its length.
4. ** Write a Python program to count occurrences of a substring in a string.**

** Output:**

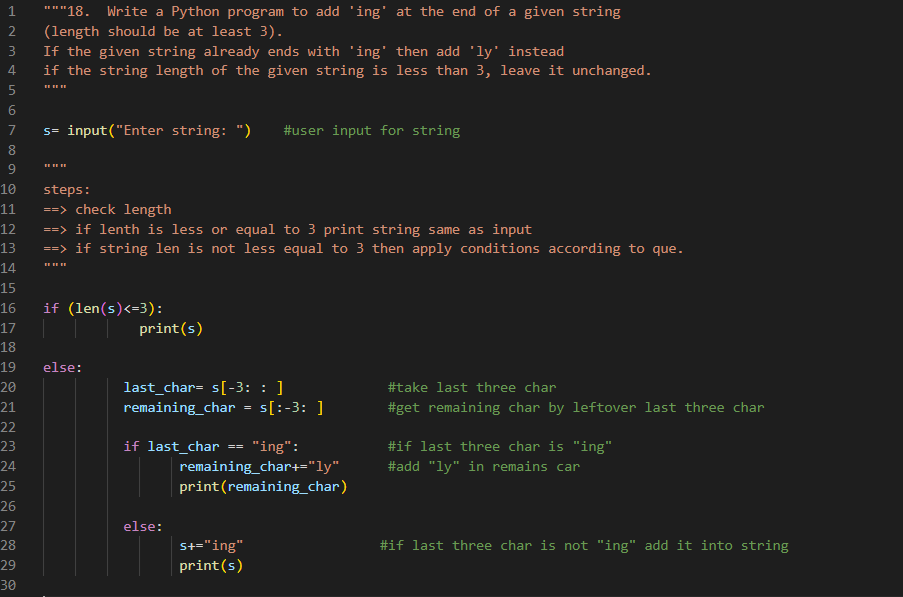
1. ** Write a Python program to count the occurrences of each word in a given sentence**

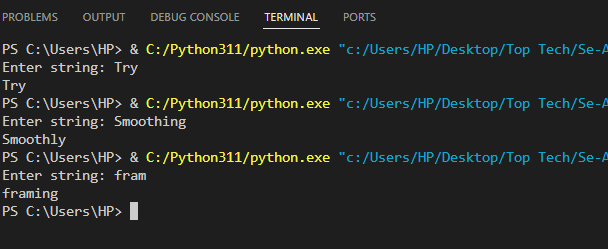
**Output:**

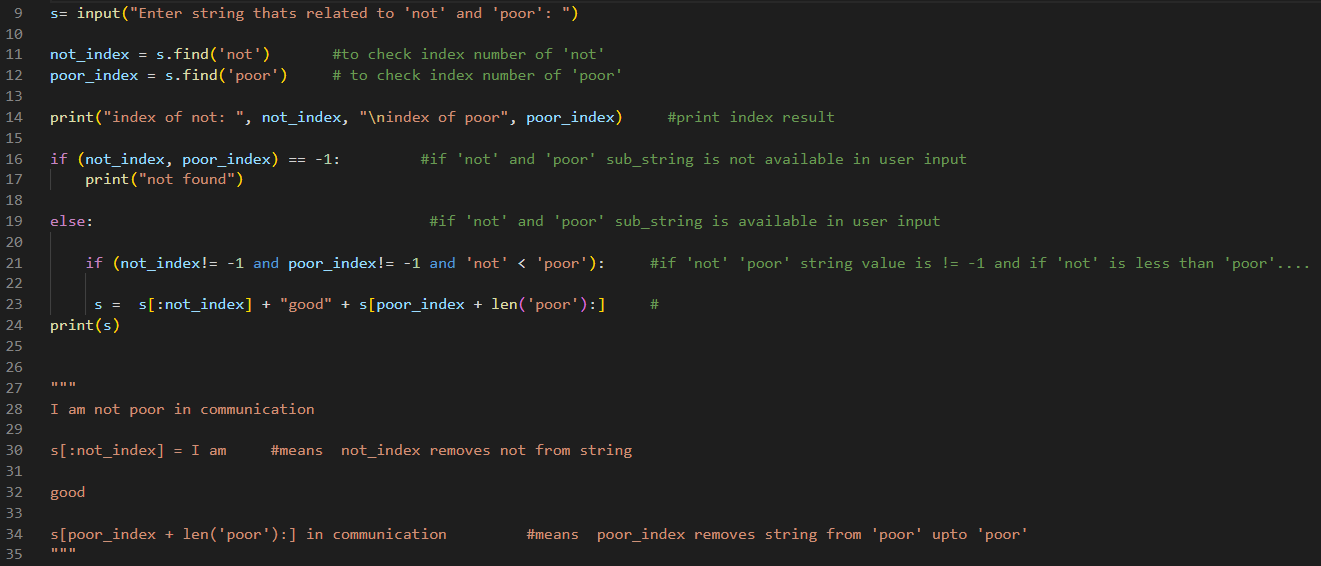
****

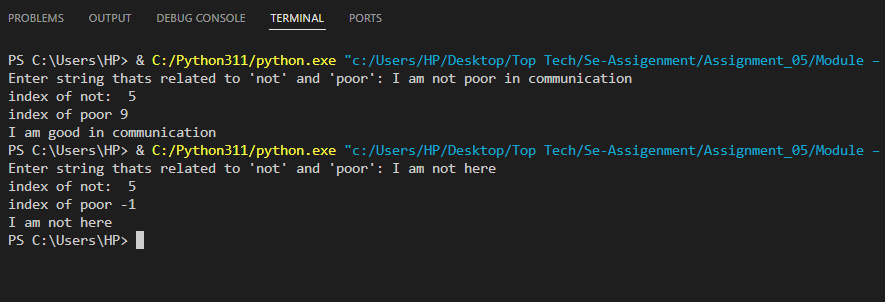
1. ** Write a Python program to get a single string from two given strings, separated by a space and swap the first two characters of each string.**

** Output:**

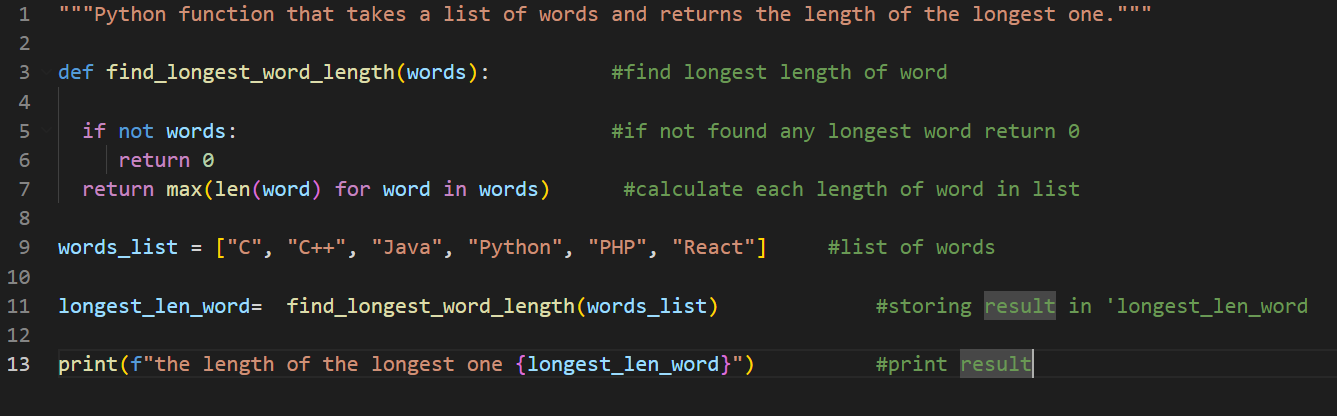
1. **Write a Python program to add 'ing' at the end of a given string (length should be at least 3). If the given string already ends with 'ing' then add 'ly' instead if the string length of the given string is less than 3, leave it unchanged. **

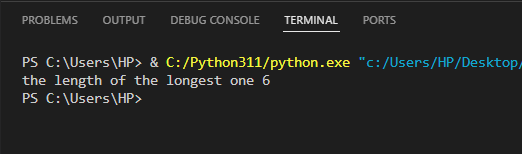
** Output:**

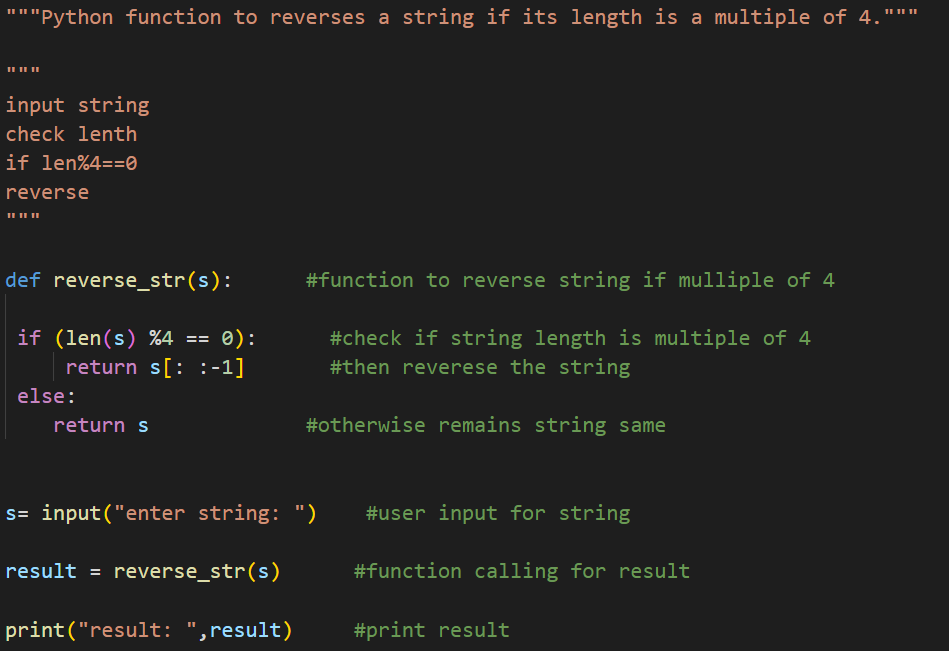
1. **Write a Python program to find the first appearance of the substring 'not' and 'poor' from a given string, if 'not' follows the 'poor', replace the whole 'not'...'poor' substring with 'good'. Return the resulting string.**

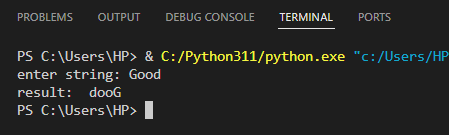
**Output:**

1. **Write a Python function that takes a list of words and returns the length of the longest one.**

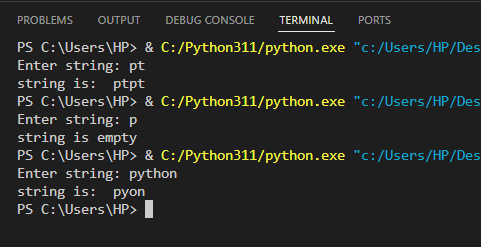
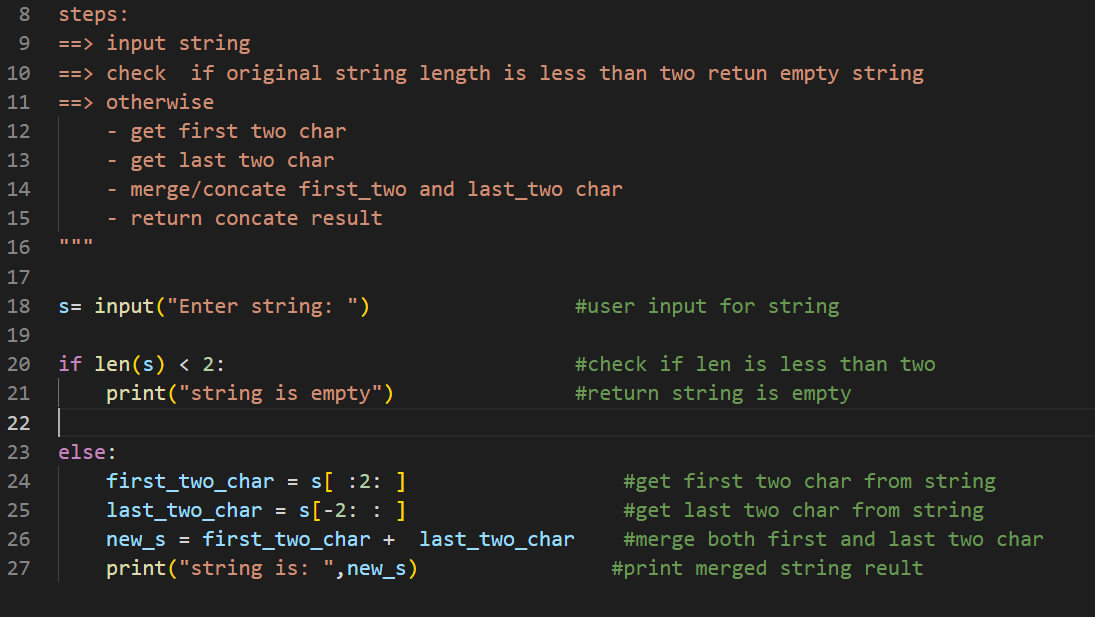
**Output:**

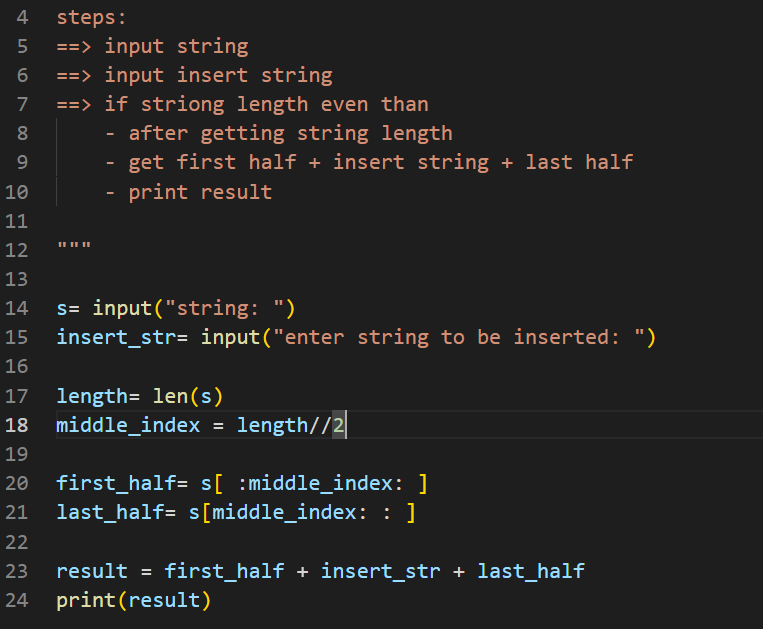
****

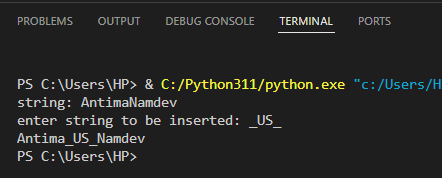
1. **Write a Python function to reverses a string if its length is a multiple of 4.**

**Output:**

1. **Write a Python program to get a string made of the first 2 and the last 2 chars from a given a string. If the string length is less than 2, return instead of the empty string.**

**Output:**

1. ** Write a Python function to insert a string in the middle of a string.**

**Output:**