**Python Control Flow**

**Indentation in python**

In Python, indentation is crucial as it defines the structure and flow of the code. Python uses indentation to indicate blocks of code, especially in constructs such as loops, functions, conditionals, and classes.

**Code Blocks**: Code inside constructs like if, for, while, def, and class is indented to show that it belongs to that construct.

Flow control

Flow control in Python refers to the mechanisms that allow us to dictate the order in which code is executed. Basically there are three type of control flow in python. They are:

1. Conditional/ Selective statements (if, if-else, if-elif-else)
2. Iterative statements (for, while)
3. Transfer statements (break, continue)

**Conditional statements**

Conditional statements in Python are used to execute different blocks of code based on certain conditions. The most common conditional statements are if, elif (short for "else if"), and else.

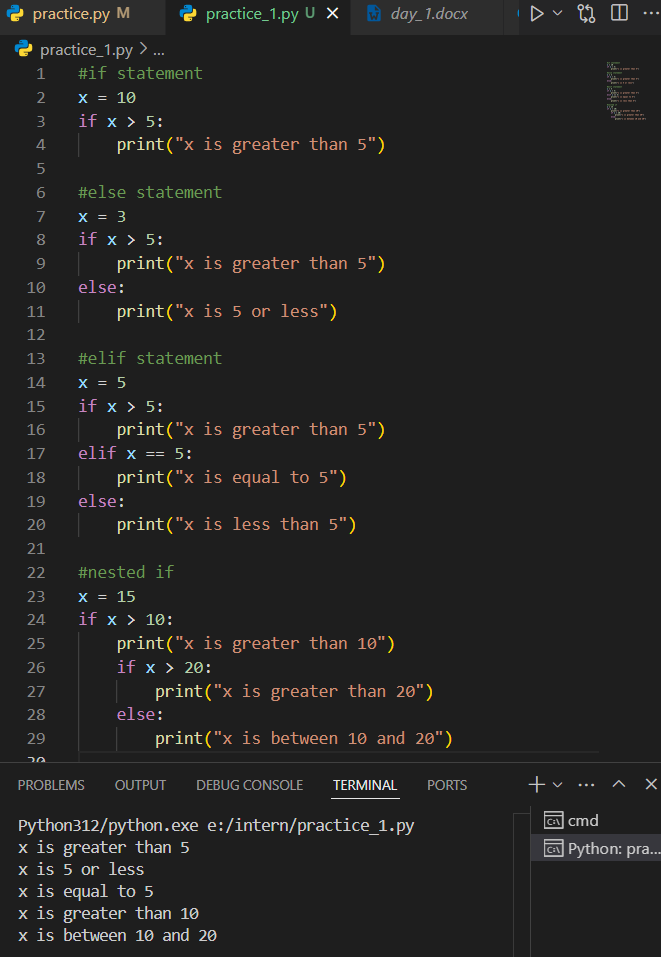
**If statement**: The if statement tests a condition and executes the block of code inside it if the condition is True.

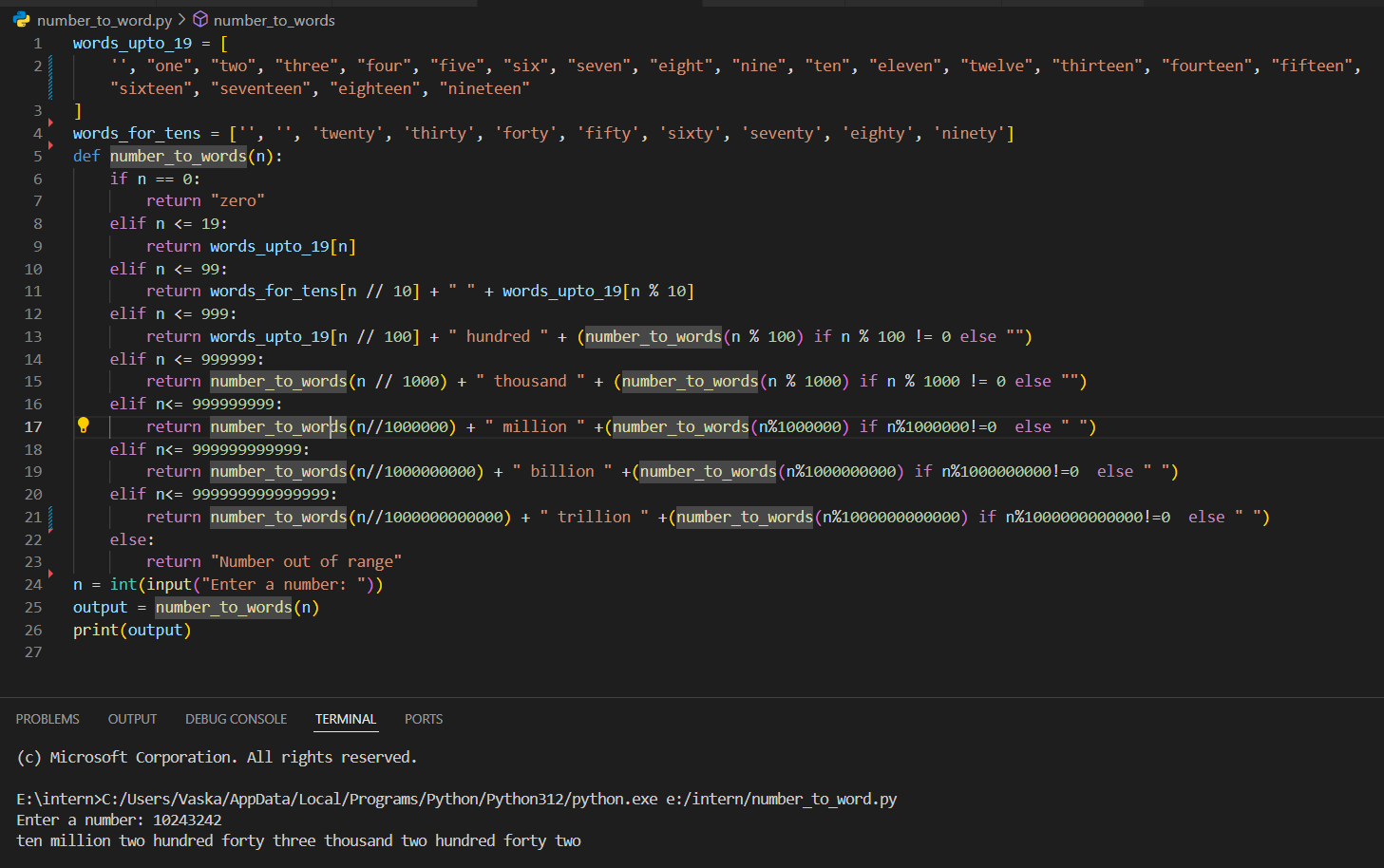
**Else statement**: The else statement is used to define a block of code that will run if the condition in the if statement is False.

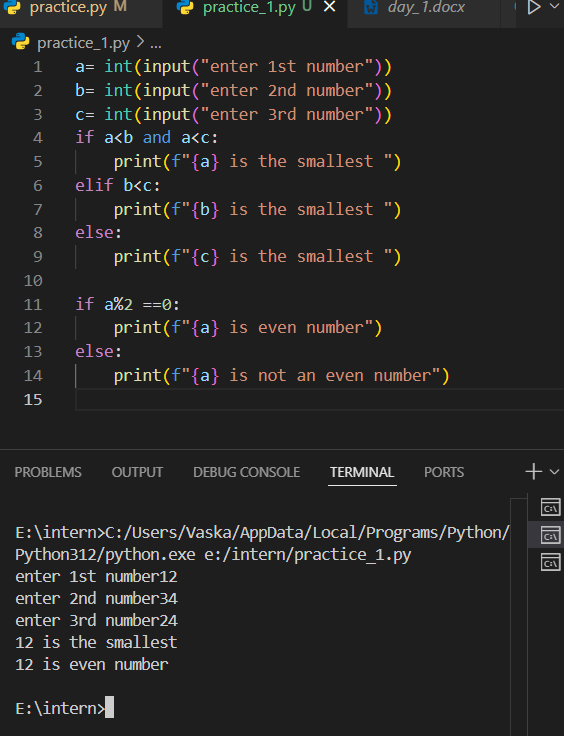
**Elif Statement**: The elif (short for "else if") statement checks multiple conditions. It runs the block of code if its condition is True and the previous if or elif conditions were False. We can have multiple elif conditions to check for different scenarios.

**Nested if Statements**: We can also nest if statements inside each other to handle more complex conditions.









**Iterative Statements**

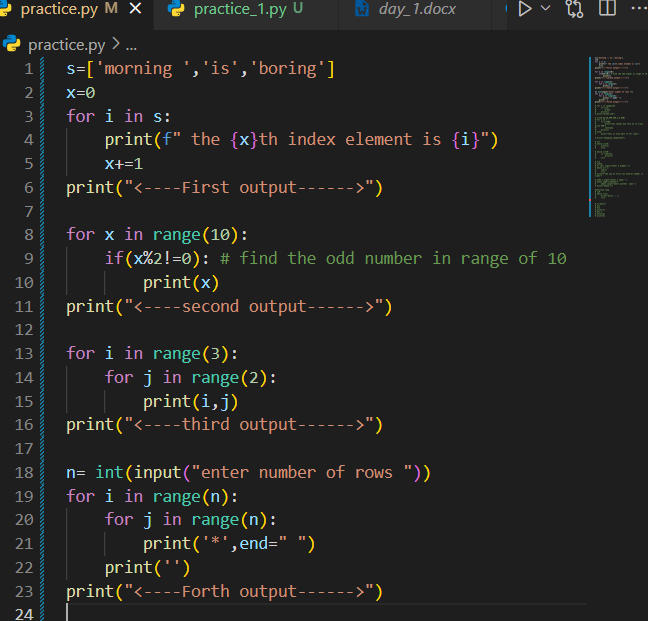
**Iterative statements** (loops) are used to execute a block of code repeatedly as long as a condition is met. Python provides two primary iterative statements: the for loop and the while loop

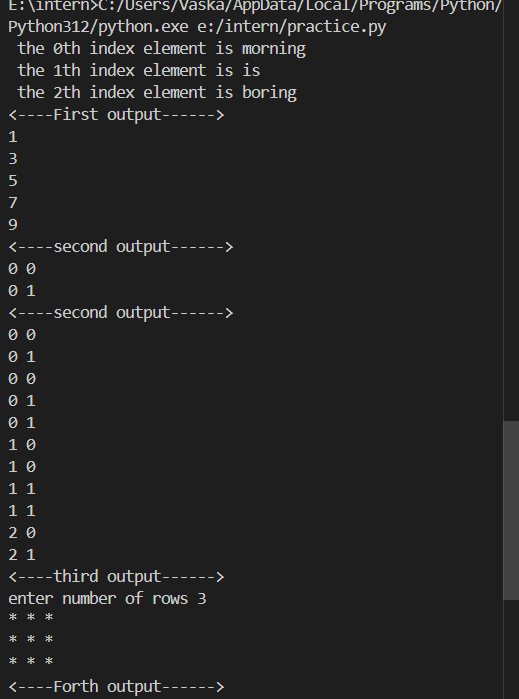
**For loop :**

The for loop in Python is used to iterate over a sequence (like a list, tuple, string, or range) and executes the block of code for each element in the sequence.

Looping with range: We can use the range() function to generate a sequence of numbers.

Nested loop: A for loop can also be nested inside another for loop then it is said to be nested for loop.





**While loop:**

The while loop repeatedly executes a block of code as long as the condition provided is True.

**Control Flow in Loops**: We can control the execution of loops using the following statements:

Break: The break statement is used to exit the loop prematurely, regardless of the condition.

Continue: The continue statement skips the current iteration and moves to the next one.

Else with loops: In Python, loops can also have an else block, which is executed when the loop completes normally (i.e., no break was encountered).

Nested while loop: We can also nest while loops inside other loops.

**Transfer Statements (break, continue, pass):**

**Transfer statements** in Python are used to alter the normal flow of execution by transferring control from one part of the code to another. The most common transfer statements are:

**break Statement:**

The break statement is used to immediately exit a loop, regardless of the iteration condition. It can be used in both for and while loops.

**continue Statement:**

The continue statement skips the rest of the code inside the current iteration of the loop and moves to the next iteration.

**pass Statement:**

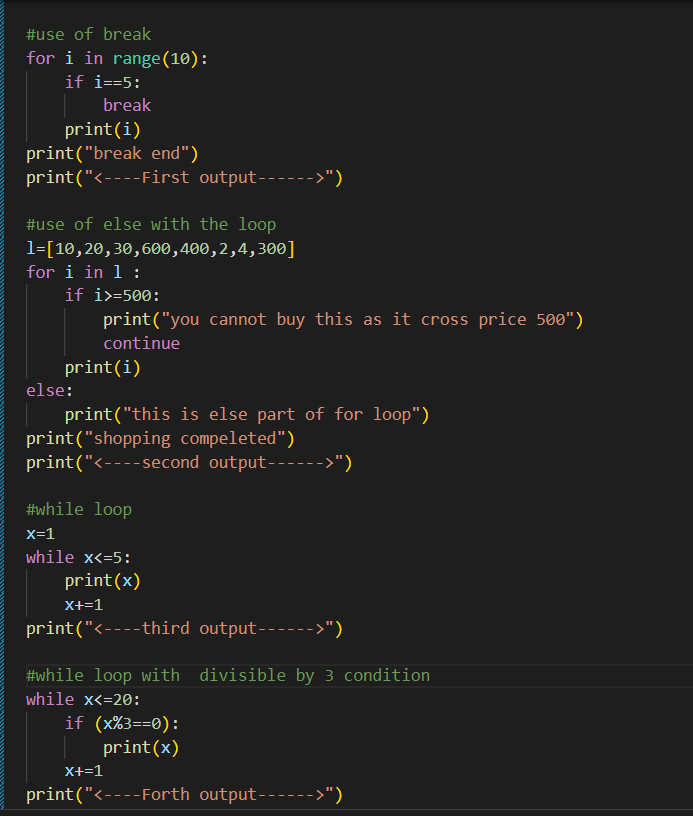
The pass statement is a null operation; it does nothing. It is often used as a placeholder when code is syntactically required but not yet implemented.

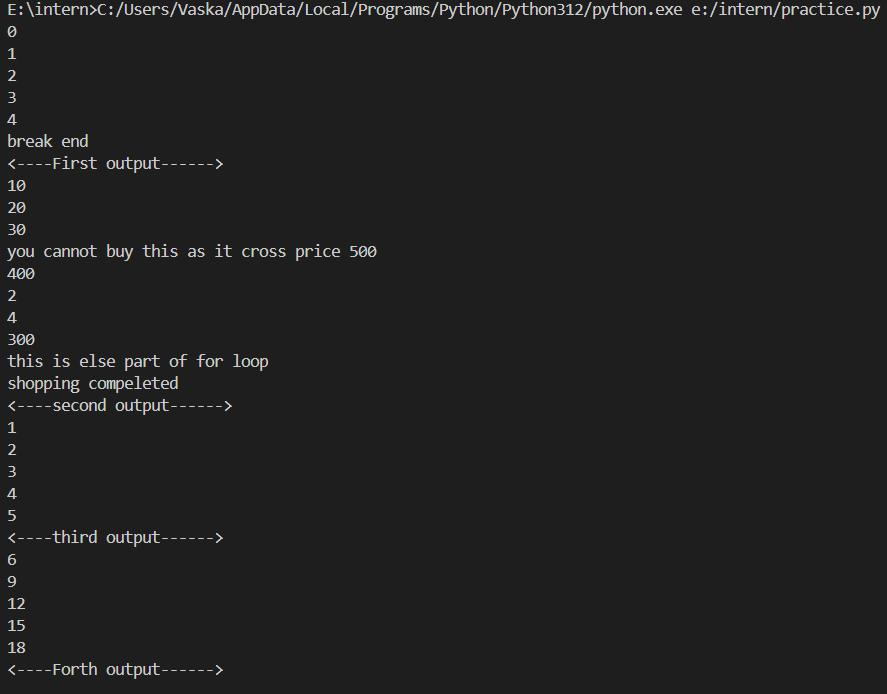
**Del statement:** The **del statement** in Python is used to delete objects, such as variables, list items, or dictionary entries. It helps in managing memory by removing references to objects that are no longer needed. Once an object is deleted, trying to access it will result in an error. The garbage collector collect the value with no reference and allocate the memory space.

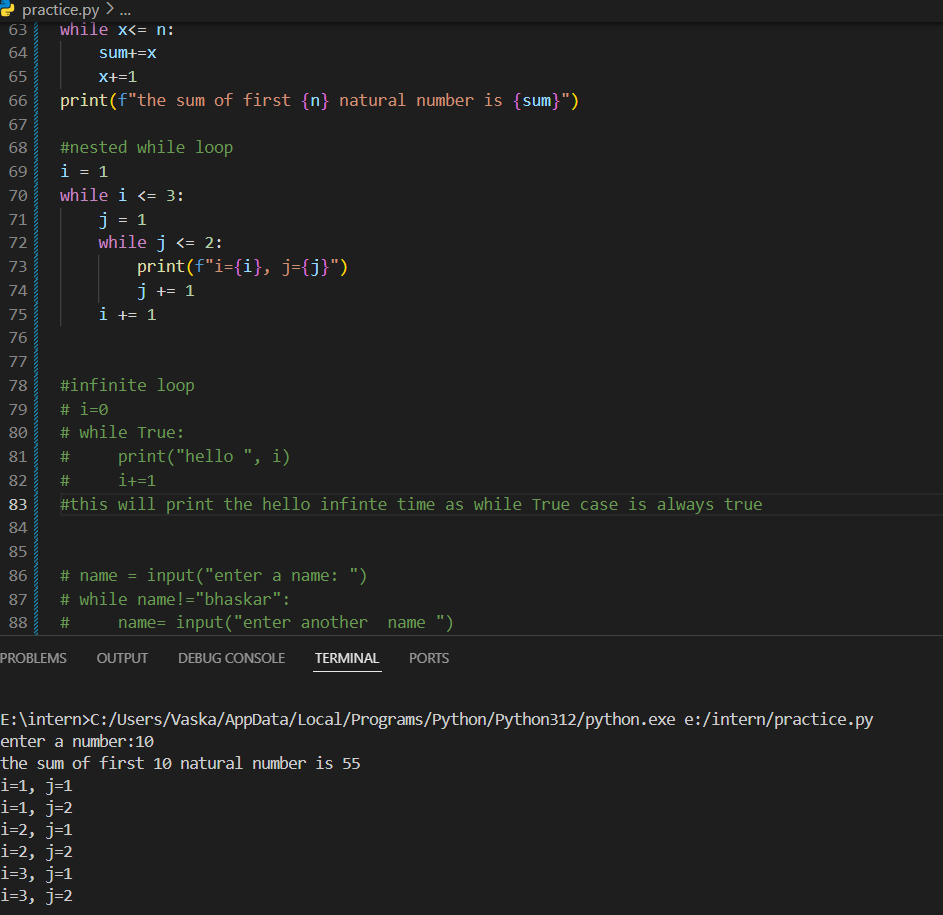
Difference between del and none:

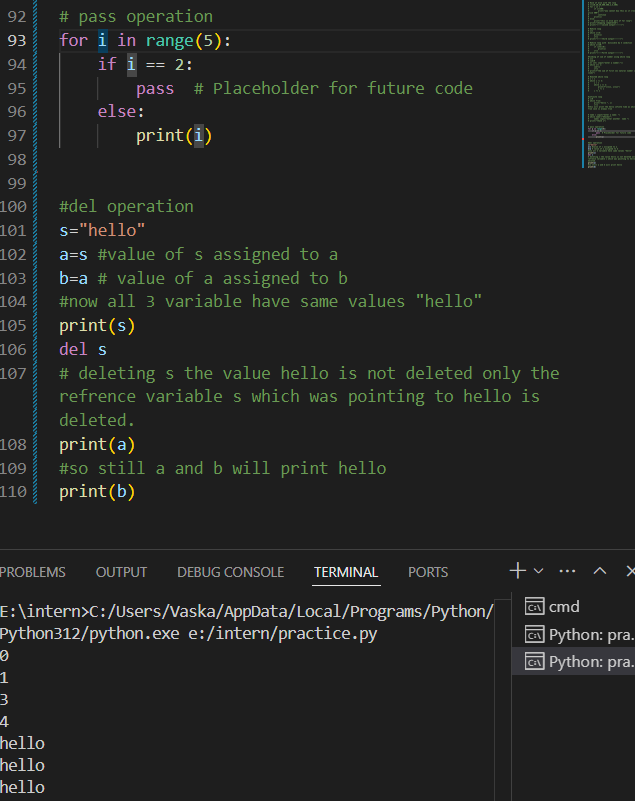
**del**: Used when we want to completely remove a variable, object, or element from memory or data structure.

**None**: Used when we want to keep the variable but indicate that it has no meaningful value for now, or as a placeholder until it's assigned something else.









**Assignment programs:**

