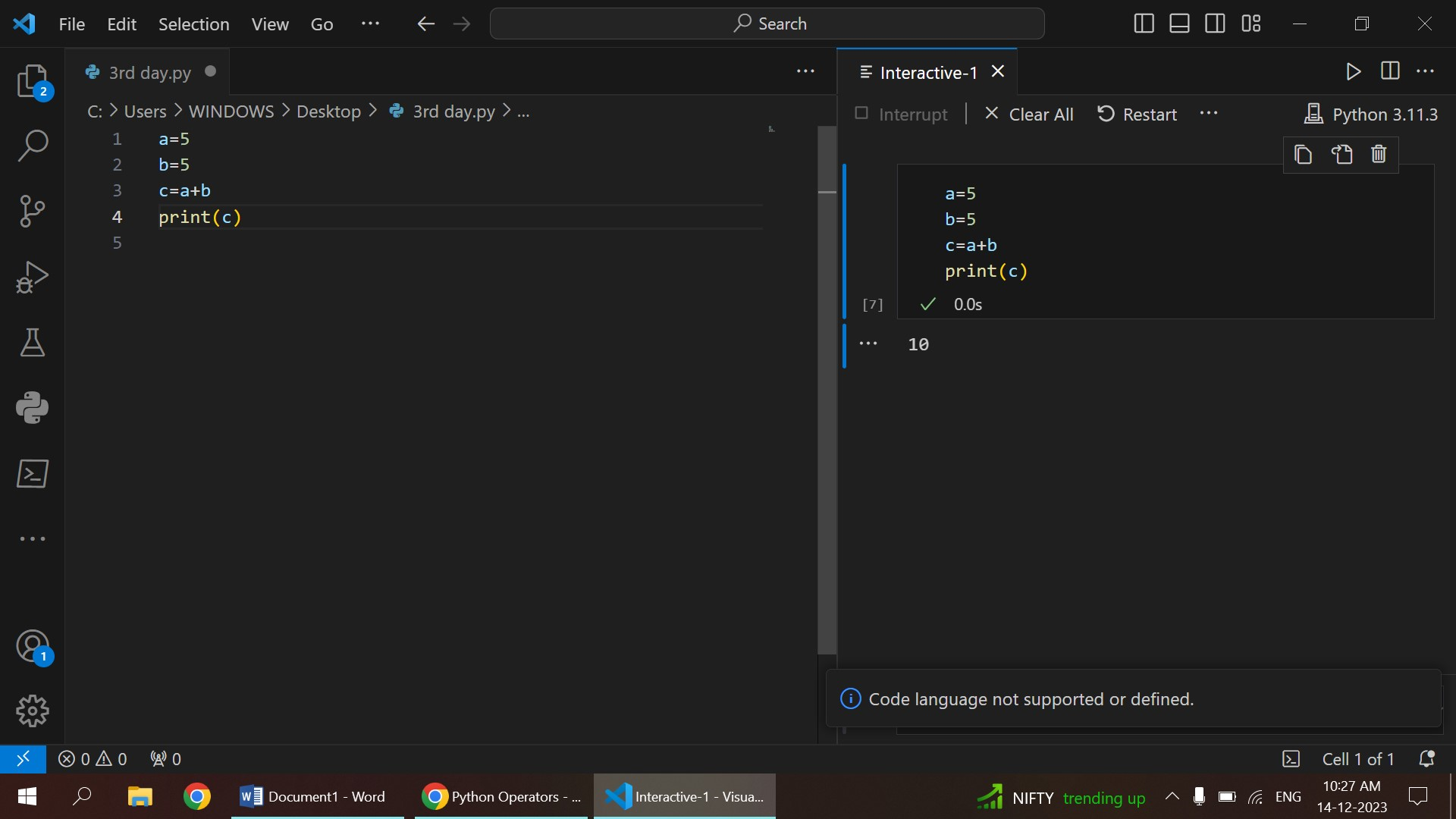
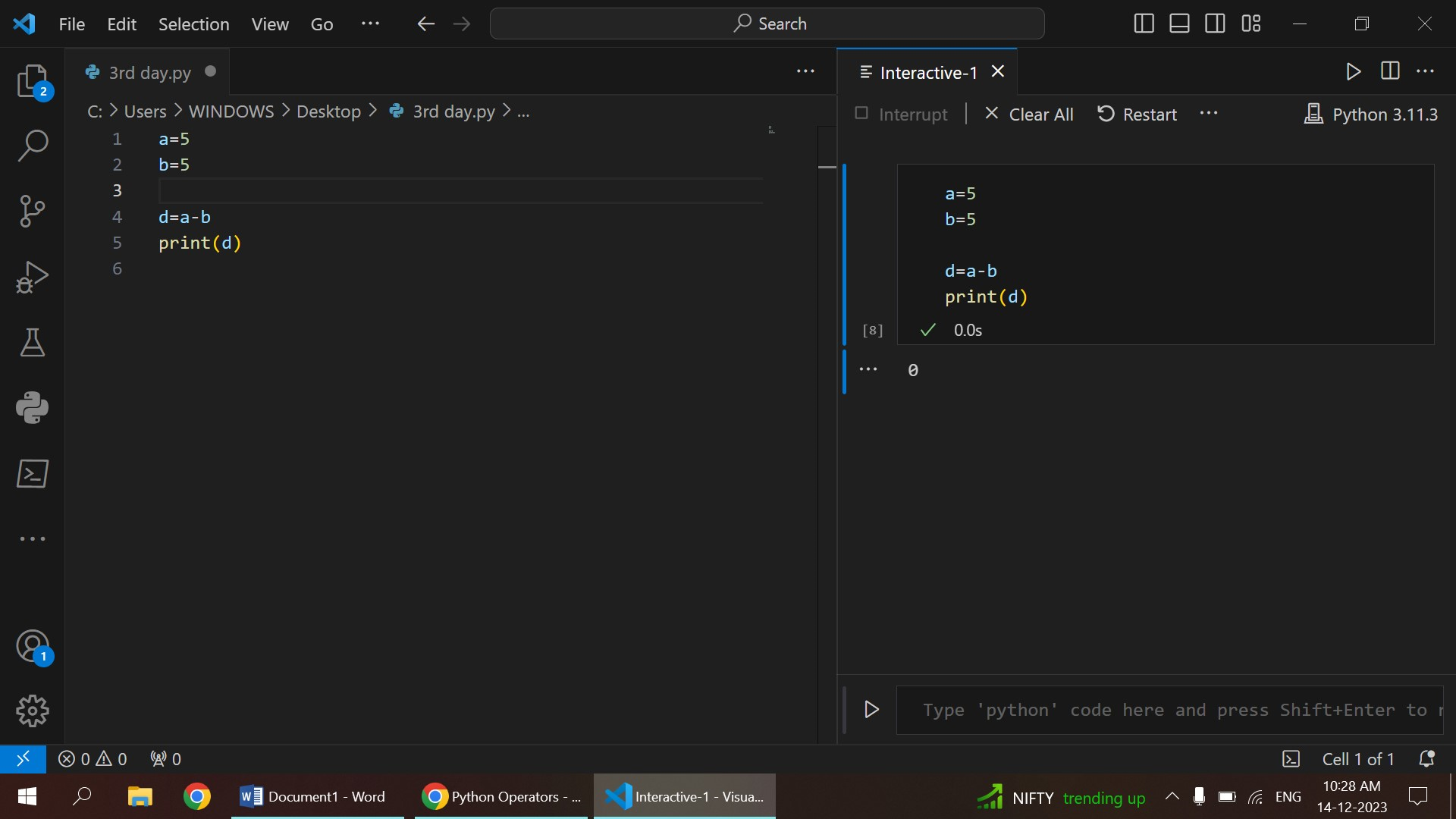
PYTHON

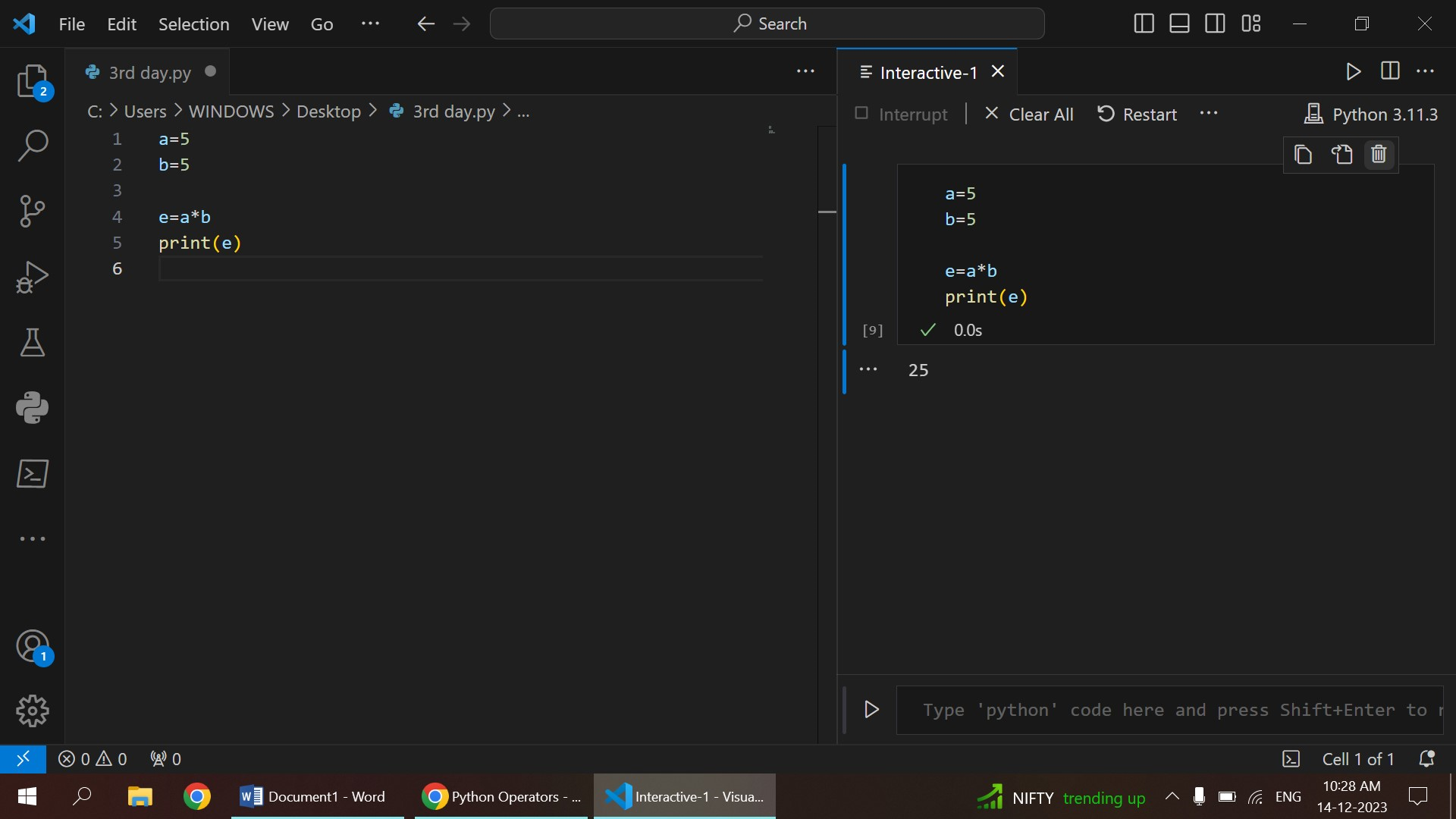
S.R.TAANUSRI

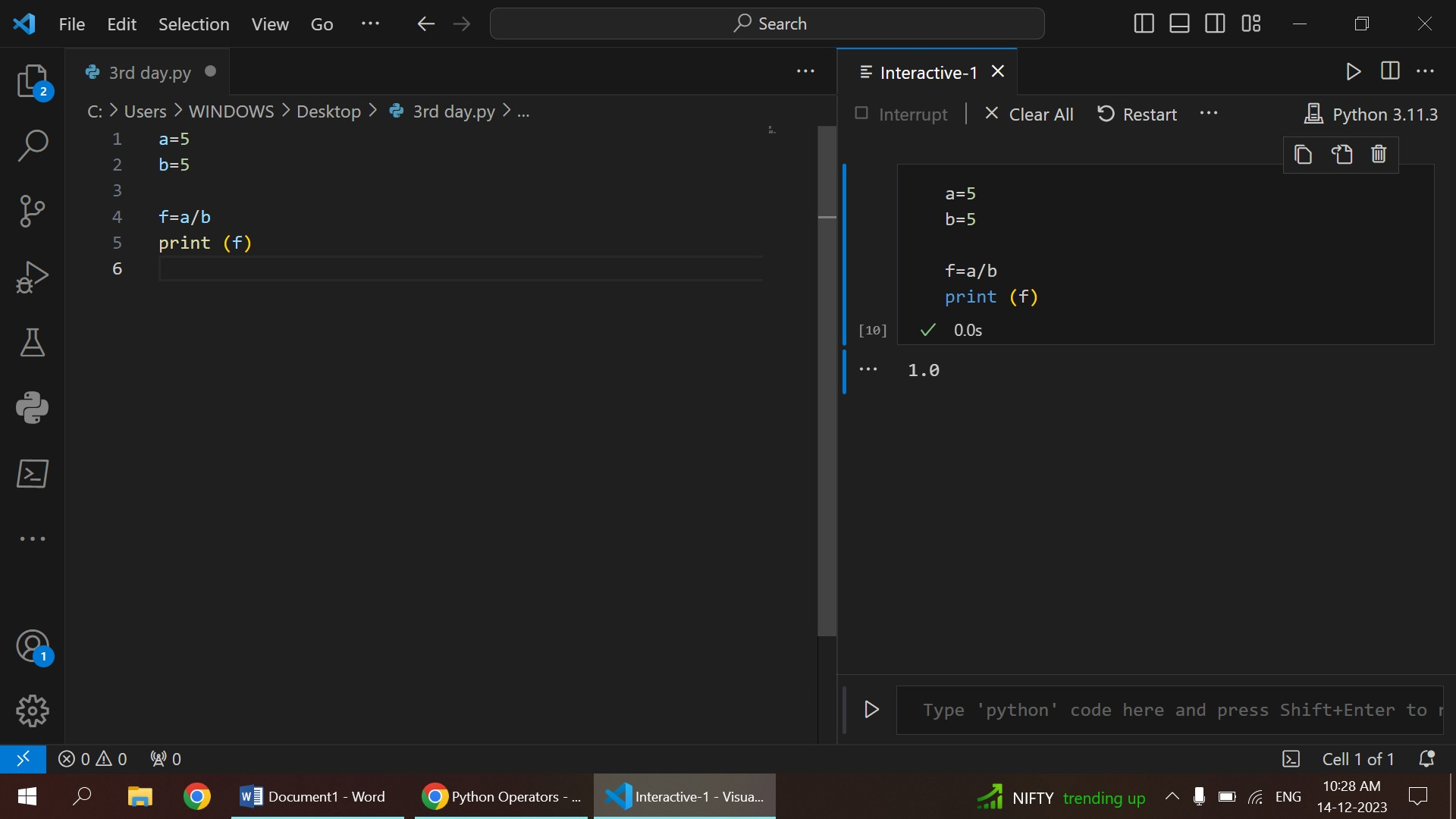
14.12.2023

**OPERATORS**

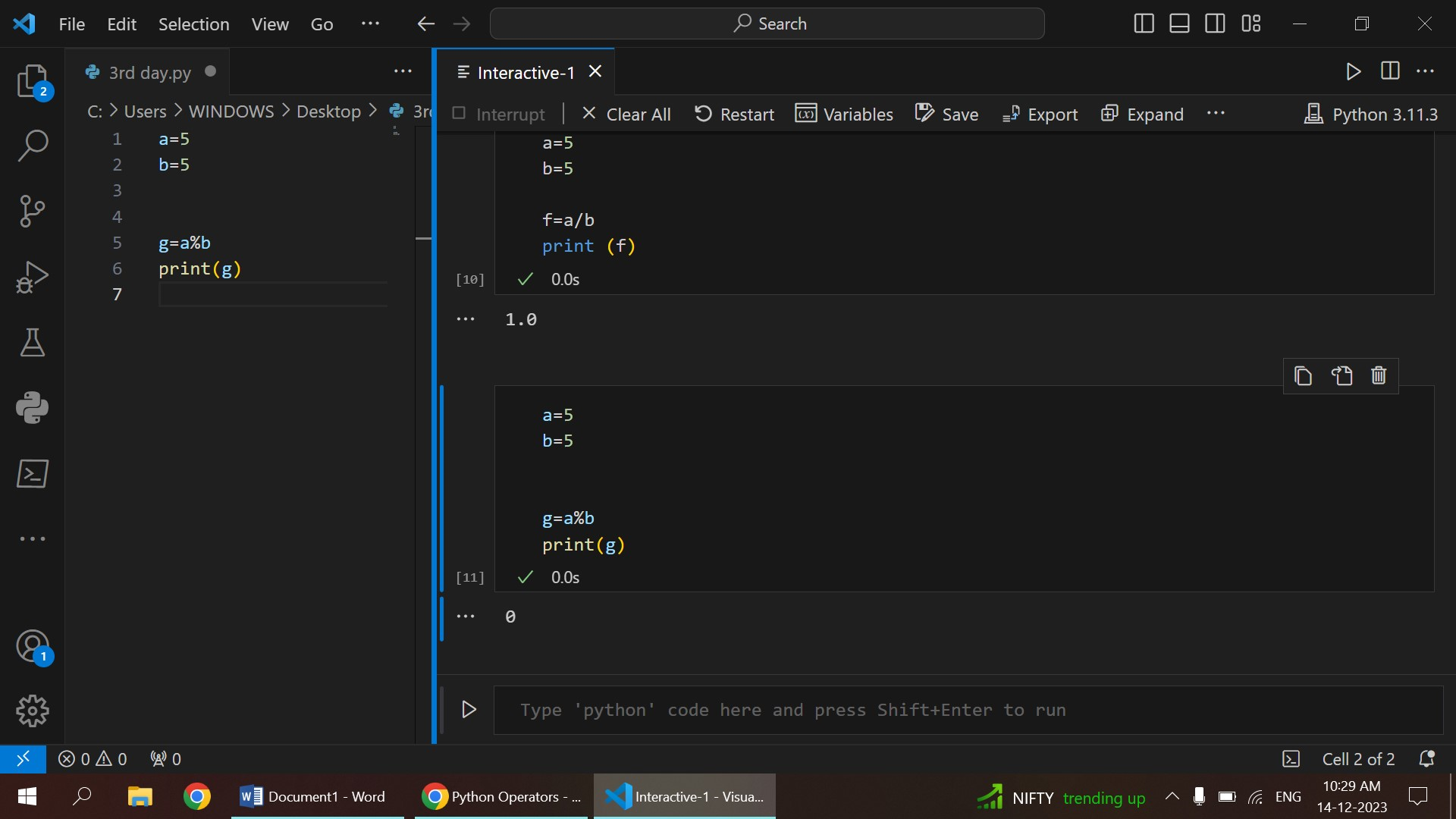
1.ARITHMETIC OPERATOR

2.ASSIGNMENT OPERATOR

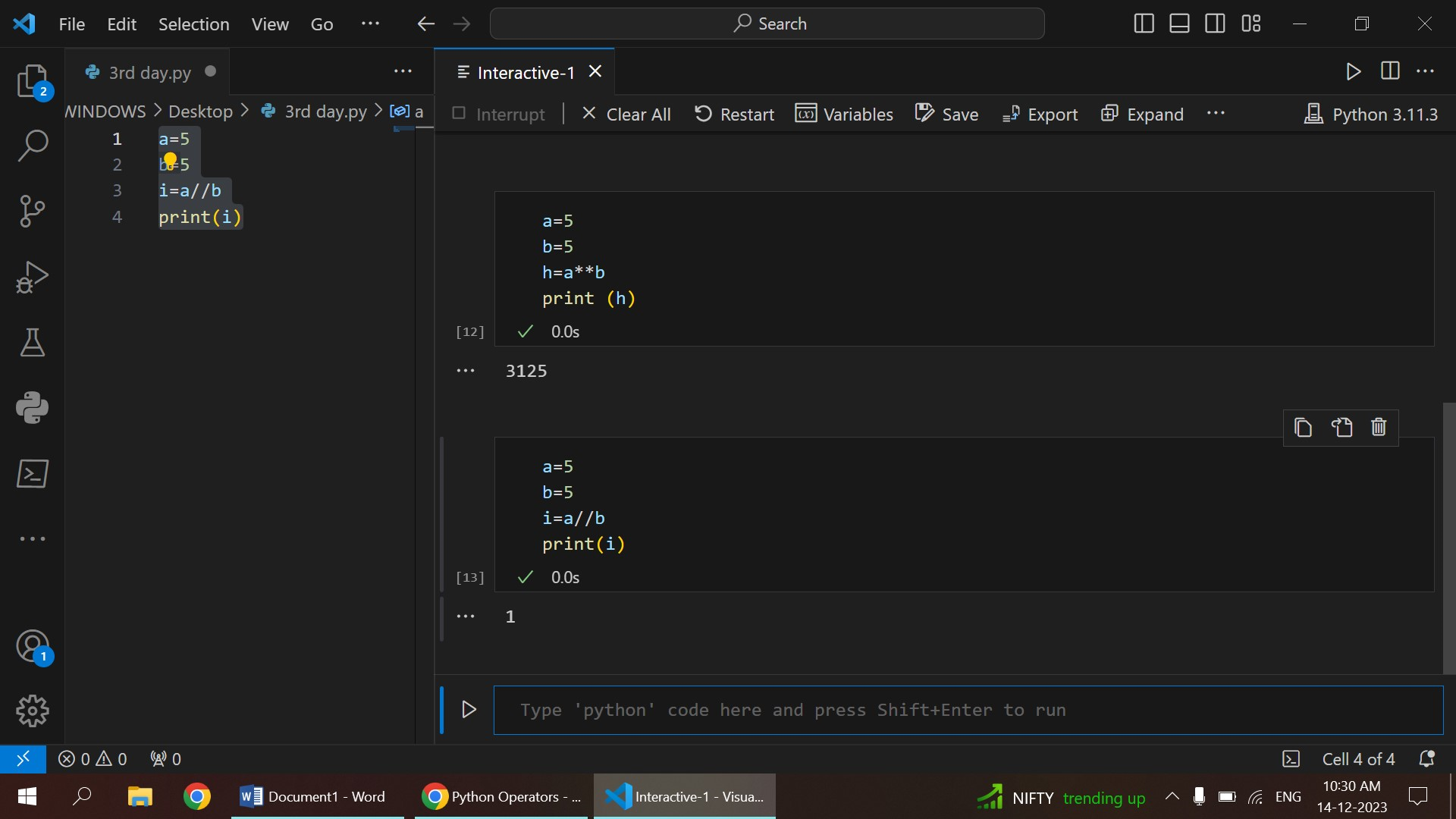
3.MULTIPLICATION 4.DIVISION



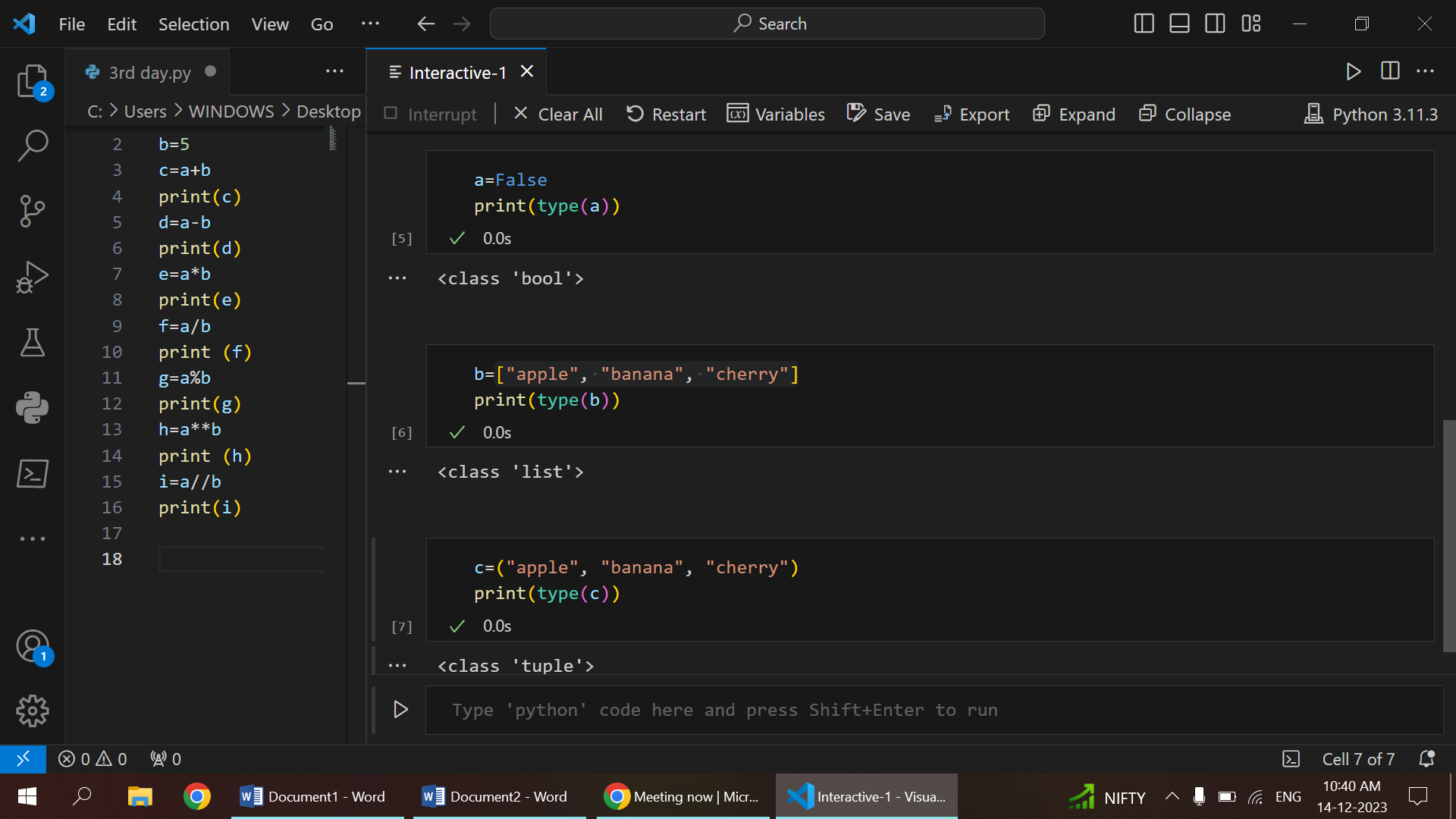
5.MODULO

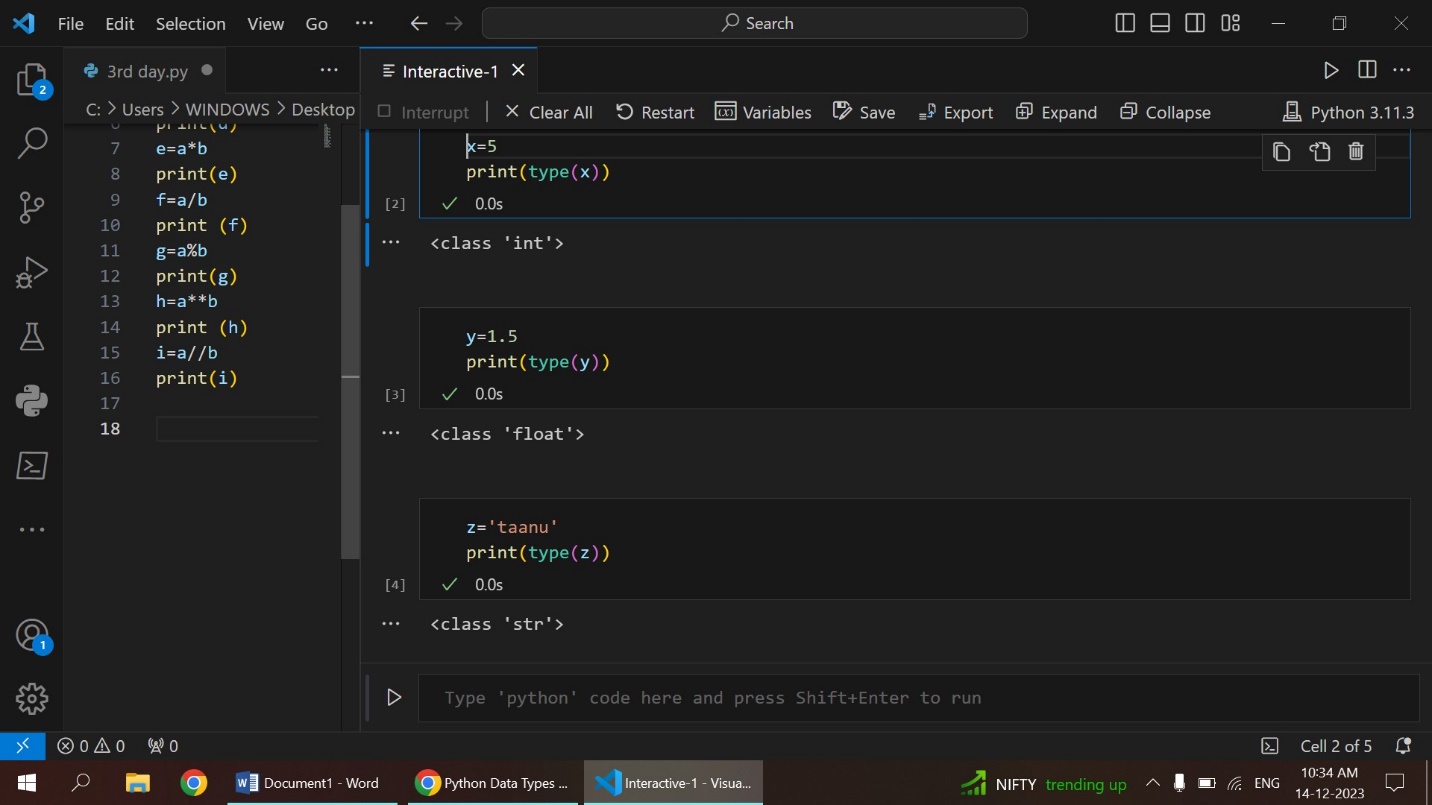


6.EXPONENTIAL AND FLOOR DIVISION



**DATA TYPES**





* Data types in Python can be categorized into mutable (modifiable) and immutable (unchangeable). Lists are mutable, enabling modification of elements, while tuples and strings are immutable, preserving their values once defined.
* Python facilitates seamless conversion between different data types through functions like int(), str(), float(), enabling flexibility in handling diverse types of data in a program.
* Python employs dynamic typing, allowing variables to take on different data types during runtime without the need for explicit type declarations.

1. Int

Represents integer (whole number) values.

1. Str

Represents a string of characters, i.e., text.

1. bool

Represents Boolean values, which are either True or False.

1. Float

Represents floating-point numbers (decimal numbers).

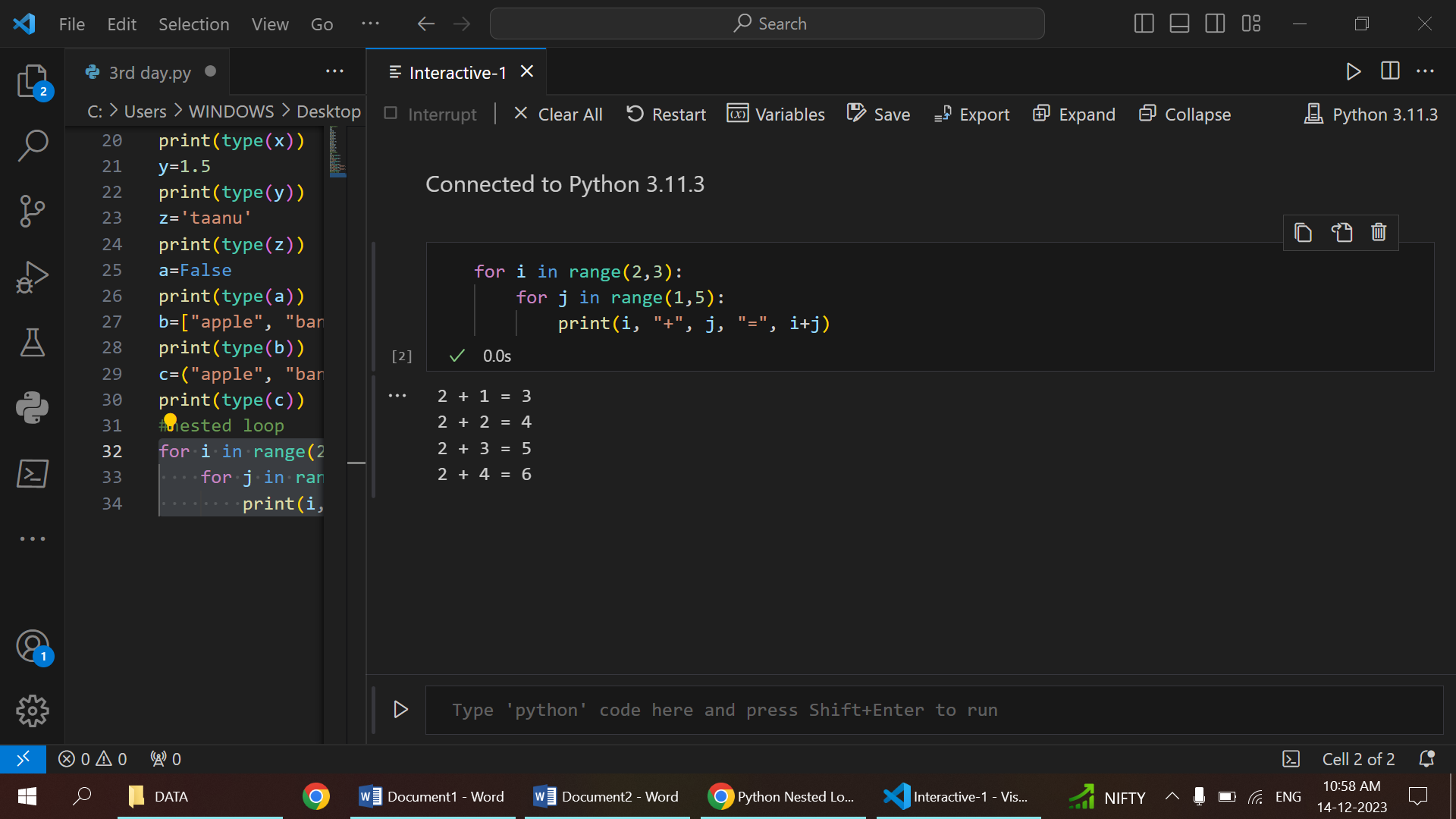
1. Tuple

Represents an immutable ordered collection of elements. Once created, the elements and their order cannot be changed.

1. List

Represents a mutable ordered collection of elements. You can add, remove, or modify elements after the list is created.

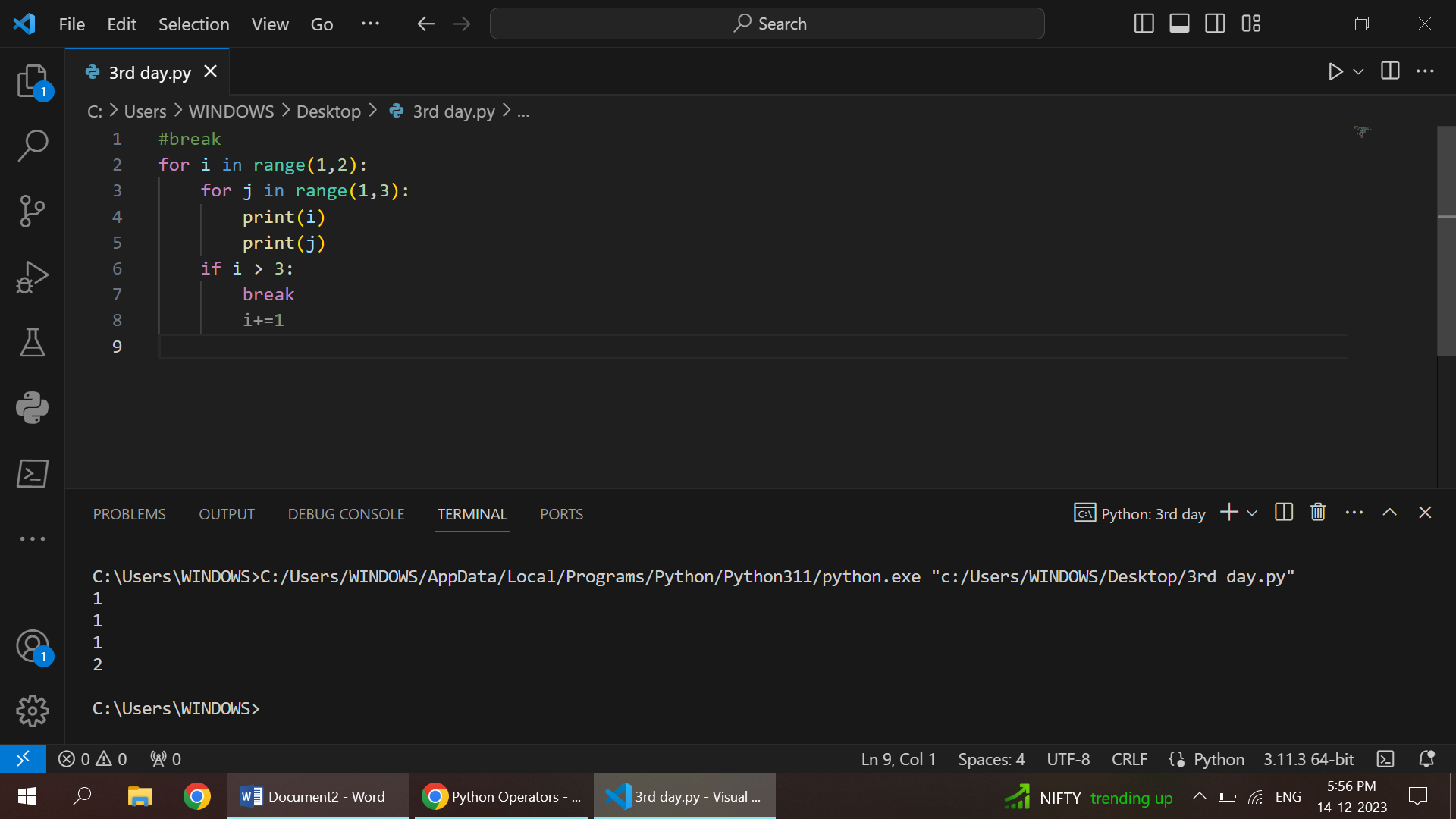
**NESTED LOOP**



* A nested loop in Python is a loop structure within another loop.
* This arrangement allows for the repetition of a block of code multiple times, with the inner loop executing its entire cycle for each iteration of the outer loop.
* This is particularly useful when working with two-dimensional data structures, like matrices.
* Nested loops are versatile and commonly used for tasks that involve working with elements in a

1. matrix
2. grid
3. any two-dimensional data structure

**BREAK**



Control Flow Alteration:

* used to alter the flow of control in a program.
* employed within loops to prematurely exit the loop's execution.

Loop Termination:

immediately terminates the innermost loop in which it is placed

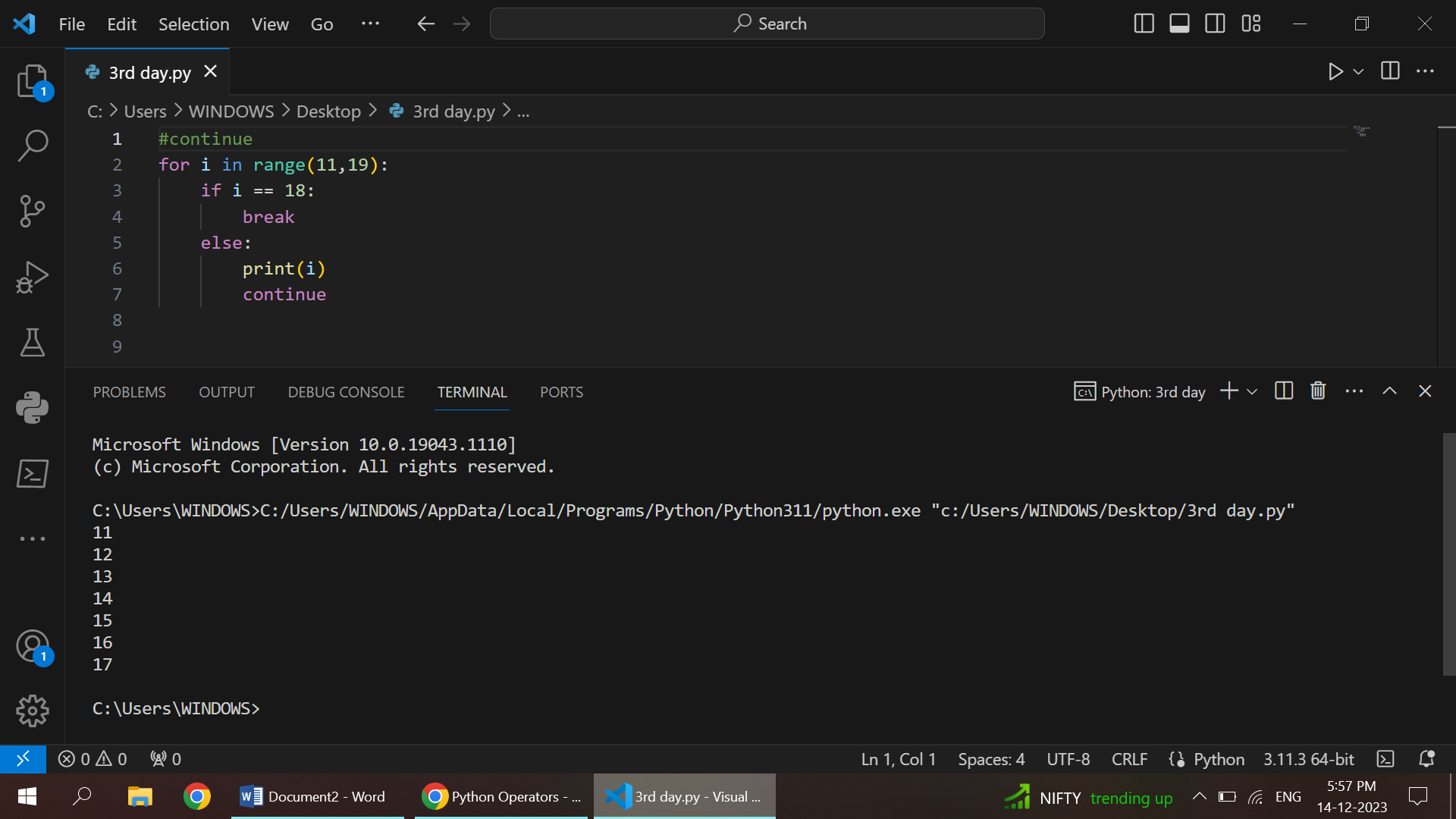
Conditional Exiting:

allows a loop to exit early based on a certain condition, providing a way to escape the loop when a specific criterion is met.

Jumping Out of Switch:

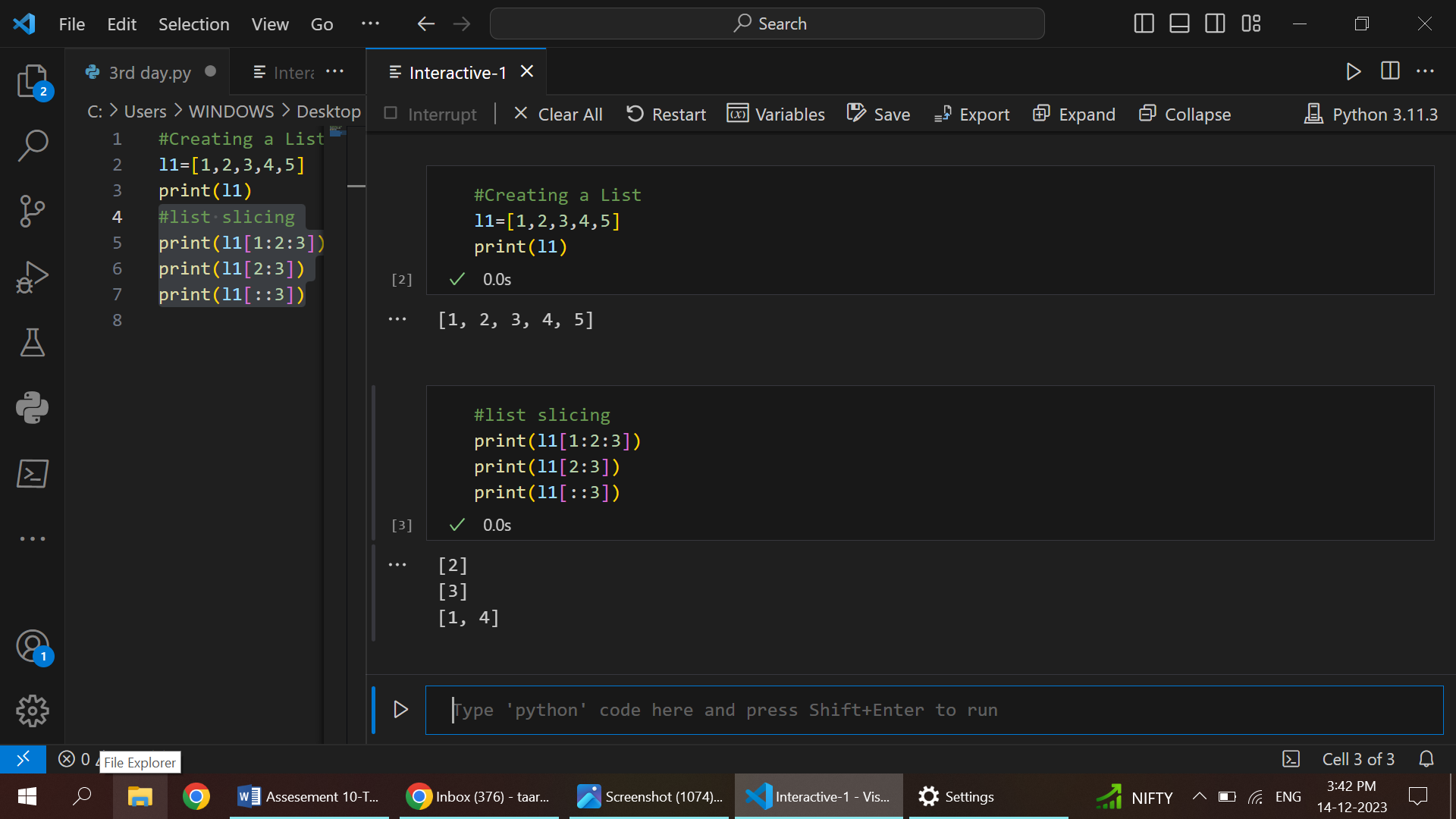
used to exit out of a block of code when certain conditions are met

**CONTINUE**



* used to alter the flow of control within loops. When encountered,
* causes the remaining code inside the loop for the current iteration to be skipped, and the loop proceeds to the next iteration.
* Unlike the break statement that completely exits a loop, continue allows a loop to skip the remaining code for the current iteration and move on to the next iteration of the loop.
* allows for the selective skipping of code based on specific conditions, enabling more fine-grained control over loop execution.
* The continue statement is sometimes used in loop optimization strategies to skip unnecessary computations or operations for certain loop iterations, improving the overall efficiency of the loop.

**LIST SLICING**



**LIST**

1. Lists in Python maintain the order of elements, allowing for indexed access.
2. They are mutable, enabling the addition, removal, and modification of elements.
3. Lists can store a mix of data types within the same structure.
4. Dynamic sizing allows lists to grow or shrink as elements are added or removed.
5. Common operations include appending, inserting, removing, slicing, and concatenating elements.

**LIST SLICING**

print(l1[1:2:3])

Start Index (1):

* The slicing starts at the index specified by the start value, which is 1 in this case.
* The element at index 1 is 2.(i(1)=2)

End Index (2):

* The slicing continues up to, but not including, the element at the end index.
* Here, the end index is 2, so the result includes elements up to index 1 (not including the element at index 2).

Step Value (3):

* The step value determines the interval between the elements to be included.
* In this case, the step is 3, so only every third element is included.
* The result is [2].

print(l1[2:3])

Start Index (2):

* The slicing starts at the index specified by the start value, which is 2.
* The element at index 2 is 3.

End Index (3):

* The slicing continues up to, but not including, the element at the end index.
* Here, the end index is 3, so the result includes elements up to index 2 (not including the element at index 3).
* The result is [3].

print(l1[::3])

Default Start and End:

* When the start and end indices are not specified (using ::)
* the slicing includes all elements in the list.

Step Value (3):

* The step value determines the interval between the elements to be included.
* Here, the step is 3, so only every third element is included.
* The result is [1, 4].