**Explanation of Code to Reverse an Integer Without Zeros**

**Step-by-Step Explanation of the Original Code**

**1. Input Statement:**

x = int(input("Enter the integer number: "))

* **Purpose**: The code takes an integer input from the user and stores it in the variable x.
* **Explanation**: The input() function prompts the user for input. The int() function ensures the input is converted into an integer.

**2. Convert Integer to String:**

integer\_string = str(x)

* **Purpose**: The integer x is converted into a string.
* **Explanation**: This allows the program to process each digit individually.

**3. Convert String to List:**

list\_integrs = list(integer\_string)

* **Purpose**: The string representation of the number is converted into a list of its characters (digits).
* **Explanation**: Each character in the list corresponds to a single digit of the number.

**4. Find Length of the List:**

length = len(list\_integrs)

* **Purpose**: Calculate the length of the list.
* **Explanation**: This value helps in iterating through the list in reverse order.

**5. Create an Empty List for Reversed Digits:**

list\_integrs2 = []

* **Purpose**: Initialize an empty list to store the reversed digits without zeros.
* **Explanation**: This list will be populated during the reversal process.

**6. Reverse the List and Filter Zeros:**

for i in range(length - 1, -1, -1):

if list\_integrs[i] != '0':

list\_integrs2.append(list\_integrs[i])

else:

continue

* **Purpose**: Iterate through the list in reverse order and filter out zeros.
* **Explanation**:
  + range(length - 1, -1, -1) generates indices to iterate through the list from the last to the first element.
  + If the current digit is not '0', it is added to list\_integrs2 using .append().
  + If the current digit is '0', the continue statement skips to the next iteration.

**7. Convert the Reversed List Back to a String:**

String\_of\_intergs = ''.join(list\_integrs2)

* **Purpose**: Combine the elements of list\_integrs2 into a single string.
* **Explanation**: The join() function concatenates the list elements into a string.

**8. Print the Result:**

print(String\_of\_intergs)

* **Purpose**: Display the reversed integer without zeros.

**Refactored Code with Functions**

Refactoring improves modularity, readability, and reusability. Below is the refactored version:

def get\_input():

"""Prompt the user to enter an integer."""

return int(input("Enter the integer number: "))

def reverse\_integer\_no\_zeros(x):

"""Reverse the digits of an integer and remove zeros."""

integer\_string = str(x) # Convert integer to string

list\_integrs = list(integer\_string) # Convert string to list

list\_integrs2 = [

digit for digit in reversed(list\_integrs) if digit != '0'

] # Reverse and filter zeros

return ''.join(list\_integrs2) # Join the list into a string

def main():

"""Main function to handle the process."""

x = get\_input()

reversed\_integer = reverse\_integer\_no\_zeros(x)

print("Reversed integer without zeros:", reversed\_integer)

if \_\_name\_\_ == "\_\_main\_\_":

main()

**How to Document in a Word File**

1. **Use Headings**: Organize the document using headings like:
   * Input Handling
   * String Conversion
   * Reversing the Integer
   * Output
2. **Explain Each Section**: Add explanations beside the code snippets.

**Include Refactored Code**: Showcase the refactored code and explain the benefits of modularity.

1. **Add Visuals**: Use screenshots of the code and output for clarity.
2. **Format Properly**: Use bullet points, bold text, and tables for better readability.

**Benefits of Refactoring**

* **Readability**: Clear separation of logic into functions makes the code easier to understand.
* **Reusability**: Functions can be reused in other programs.
* **Debugging**: Errors are easier to locate in modular code.

Would you like further assistance with creating the Word file or adding visuals?