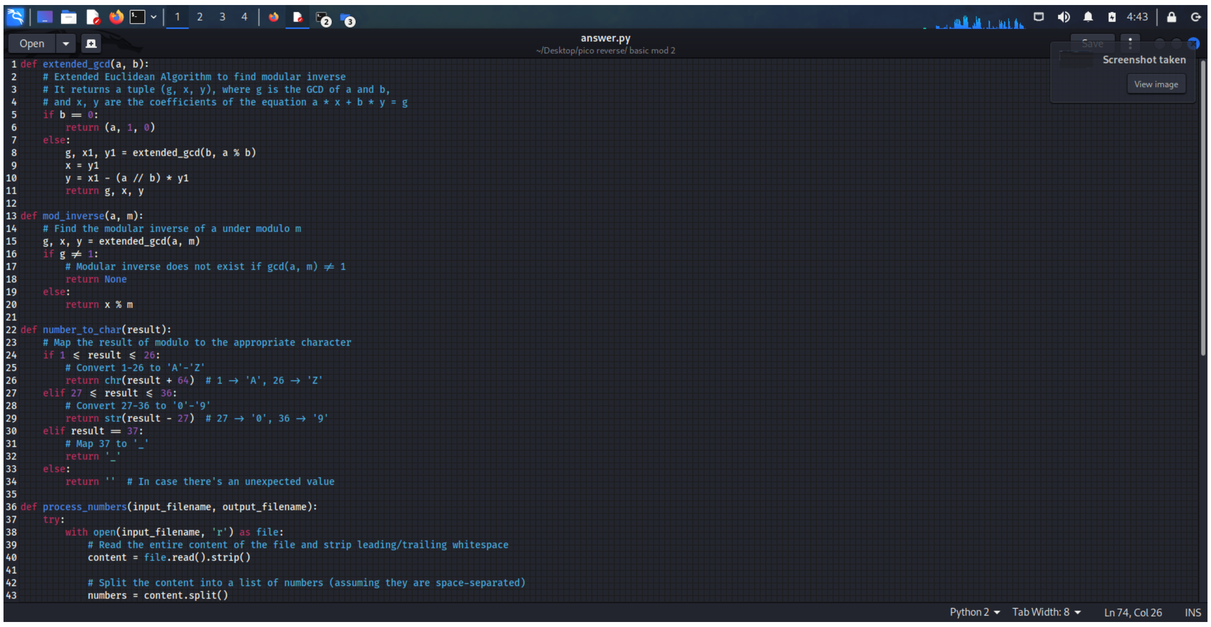
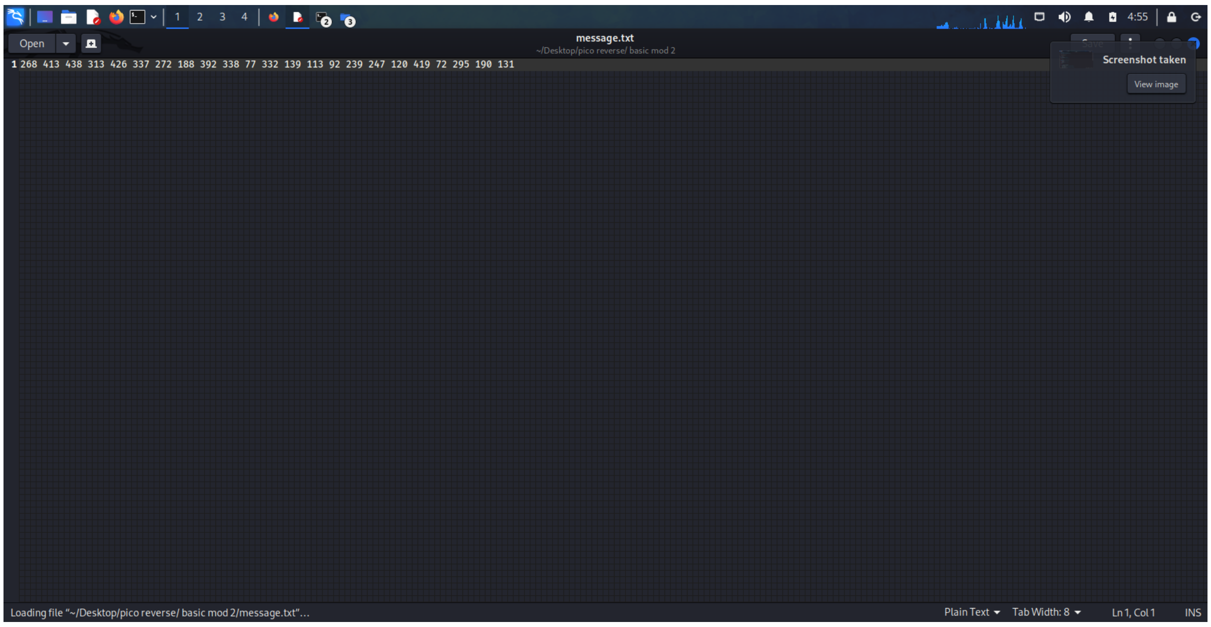
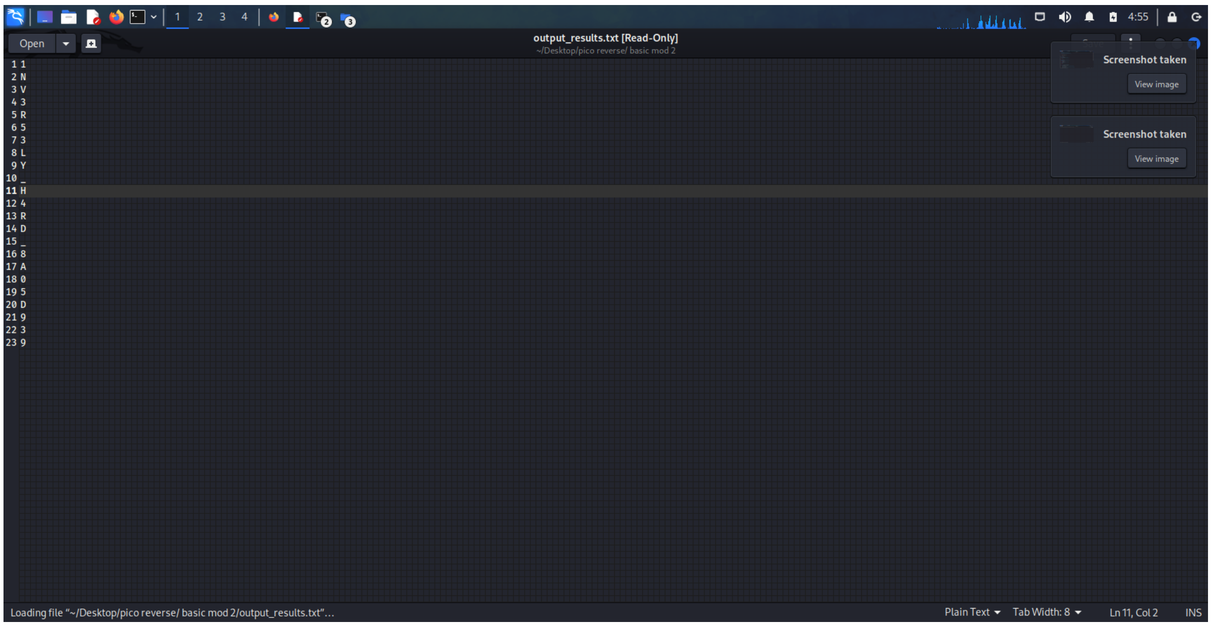


```python
def extended_gcd(a, b):
    # Extended Euclidean Algorithm to find modular inverse
    # It returns a tuple (g, x, y), where g is the GCD of a and b,
    # and x, y are the coefficients of the equation a * x + b * y = g
    if b == 0:
        return (a, 1, 0)
    else:
        g, x1, y1 = extended_gcd(b, a % b)
        x = y1
        y = x1 - (a // b) * y1
        return g, x, y

def mod_inverse(a, m):
    # Find the modular inverse of a under modulo m
    g, x, y = extended_gcd(a, m)
    if g != 1:
        # Modular inverse does not exist if gcd(a, m) != 1
        return None
    else:
        return x % m

def number_to_char(result):
    # Map the result of modulo to the appropriate character
    if 1 <= result <= 26:
        # Convert 1-26 to 'A'-'Z'
        return chr(result + 64)  # 1 -> 'A', 26 -> 'Z'
    elif 27 <= result <= 36:
        # Convert 27-36 to '0'-'9'
        return str(result - 27)  # 27 -> '0', 36 -> '9'
    elif result == 37:
        # Map 37 to '_'
        return '_'
    else:
        return ''  # In case there's an unexpected value

def process_numbers(input_filename, output_filename):
    try:
        with open(input_filename, 'r') as file:
            # Read the entire content of the file and strip leading/trailing whitespace
            content = file.read().strip()

            # Split the content into a list of numbers (assuming they are space-separated)
            numbers = content.split()
```



```
268 413 438 313 426 337 272 188 392 338 77 332 139 113 92 239 247 120 419 72 295 190 131
```

Loading file "~/Desktop/pico reverse/ basic mod 2/message.txt"…



```
1
N
V
3
R
5
3
L
Y
_
H
4
R
D
_
8
A
0
5
D
9
3
9
```

Loading file "~/Desktop/pico reverse/ basic mod 2/output_results.txt"...

```
def extended_gcd(a, b):
    # Extended Euclidean Algorithm to find modular inverse
    # It returns a tuple (g, x, y), where g is the GCD of a and b,
    # and x, y are the coefficients of the equation a * x + b * y = g
    if b == 0:
        return (a, 1, 0)
    else:
        g, x1, y1 = extended_gcd(b, a % b)
        x = y1
        y = x1 - (a // b) * y1
        return g, x, y


def mod_inverse(a, m):
    # Find the modular inverse of a under modulo m
    g, x, y = extended_gcd(a, m)
    if g != 1:
        # Modular inverse does not exist if gcd(a, m) != 1
        return None
    else:
        return x % m
```

```
def number_to_char(result):
    # Map the result of modulo to the appropriate character
    if 1 <= result <= 26:
        # Convert 1-26 to 'A'-'Z'
        return chr(result + 64)  # 1 -> 'A', 26 -> 'Z'
    elif 27 <= result <= 36:
        # Convert 27-36 to '0'-'9'
        return str(result - 27)  # 27 -> '0', 36 -> '9'
    elif result == 37:
        # Map 37 to '_'
        return '_'
    else:
        return ''  # In case there's an unexpected value

def process_numbers(input_filename, output_filename):
    try:
        with open(input_filename, 'r') as file:
            # Read the entire content of the file and strip leading/trailing whitespace
            content = file.read().strip()

            # Split the content into a list of numbers (assuming they are space-separated)
            numbers = content.split()

            # Open the output file in write mode
            with open(output_filename, 'w') as output_file:
                # Process each number
                for num in numbers:
                    try:
                        # Convert the number to an integer and apply modulo 41
                        result = int(num) % 41
                        # Find the modular inverse of the result modulo 41
                        mod_inv = mod_inverse(result, 41)

                        if mod_inv is None:
                            output = '?'  # If no modular inverse exists, map to '?'
                        else:
                            # Map the modular inverse to the appropriate character
                            output = number_to_char(mod_inv)

                        # Write the result to the output file
                        output_file.write(output + "\n")
                    except ValueError:
                        print(f"Error: '{num}' is not a valid number. Skipping.")
                        continue  # Skip invalid numbers

        print(f"Processing complete. Results written to '{output_filename}'.")

    except FileNotFoundError:
        print(f"Error: The file '{input_filename}' was not found.")

# Example usage:
input_filename = "message.txt"  # Replace with your actual input file path
output_filename = "output_results.txt"  # Replace with your desired output file path

process_numbers(input_filename, output_filename)
```