Bonus

Here is a rot13 function:

```python
def rot13(text):
    result = ""
    for char in text:
    if 'a' <= char <= 'z':
        result += chr((ord(char) - ord('a') + 13) % 26 + ord('a'))
    elif 'A' <= char <= 'Z':
        result += chr((ord(char) - ord('A') + 13) % 26 + ord('A'))
    else: result += char

    return result
```

Here's a generalized `rotN()` function that handles any non-negative integer `N`, and performs modulo early to avoid unnecessary rotations:

```python
def rotN(text, N):
    shift = N % 26 # Normalize the shift to within alphabet range
    result = ""
    for char intext:
        if 'a'<= char <= 'z':
            result += chr((ord(char) - ord('a') + shift) % 26+ ord('a'))
        elif 'A'<= char <= 'Z':
            result += chr((ord(char) - ord('A') + shift) % 26+ ord('A'))
        else:
            result += char
    return result

# Example usage
message = "Hello, Hacker!"
encoded = rotN(message, 13)
print("Encoded with ROT13:", encoded)

decoded = rotN(encoded, 13)
print("Decoded:", decoded)

# Iterate over all lower all 26 possible rotations:
for shift in range(0, 25):
    print(f'The value for offset {shift} is "{rotN(encoded, shift)}".')

# Bonus: absurdly large rotation
print("ROT1000000000000:", rotN(message, 1000000000000))
```

🧠 Teaching Hooks

- Why `% 26` is essential for efficiency and correctness.
- How this generalizes ROT13 to any Caesar-style shift.
- What happens if you use `rotN(text, 0)` or `rotN(text, 26)`?

Want to scaffold this into a student challenge where they write `rotN()` from scratch and then test it with a few known messages?