Task 1:

Here, we'll practice working with *escape sequences* to represent special text. In particular, we'll see how to use the special string "\n" to represent a *new line*. Write a function, `format_address(name, street, city, state, zip_code)` which creates a formatted address block from the five parts of an address. Your function should *return* the formatted address. The expected format is:

Name
Street
City, State ZIP

Note the new lines after Name and Street, and the comma after City.

The signature of this function is `(string, string, string, string, string) -> string`.

Task 2:

Here, we'll see how to use the `len()` function, and work with a couple of string methods. Write a function, `mangle_text(some_string)`. Your function should:

- Take the first half of the string and convert it to lowercase
- Take the second half of the string and convert it to uppercase
- Concatenate in opposite order (second half first, then the first half)
- Return the concatenated string

For example, `mangle_text("SomeText")` should return `TEXTsome`

The signature of this function is `(string)` -> `string`.

Task 3:

This task is a buy-one-get-one-free deal, where we'll write *two* functions!

The first function, `roman(s)`, will take a single Roman numeral character as input and return its corresponding integer value. The Roman numeral system follows specific values: `'I'` is 1, `'V'` is 5, `'X'` is 10, `'L'` is 50, `'C'` is 100, `'D'` is 500, and `'M'` is 1000. If the input is not a valid Roman numeral character, the function should return 0. For example, `roman('I')` should return 1, and `roman('Z')`, an invalid input, should return 0.

The second function, `romanToInt(s)`, should take a string of Roman numerals (like `'XIII'` or `'IV'`) and convert it to its integer equivalent. Roman numerals are usually added together, **but** there are special cases where subtraction is used: for example, `'IV'` is 4 (5 - 1) and `'IX'` is 9 (10 - 1). The general rule for subtraction is that when a smaller numeral precedes a larger numeral, it is subtracted from the larger numeral (e.g., `'IV'` = 4 and `'IX'` = 9). The function needs to account for both the addition and subtraction rules. For example, `romanToInt('XIII')` should return 13, `romanToInt('IX')` should return 9, and `romanToInt('MCMXCIV')` should return 1994. You can assume that inputs are always valid arguments.

These two functions will help you work with Roman numerals, handling both single-character conversions and full numeral strings.

The signature of both functions is the same, `(string) ---> int`.