1)

```
def isomorphic_strings(s, t):
    if len(s) != len(t):
    return False

    char_map = {}
    used_chars = set()

    for i in range(len(s)):
    if s[i] in char_map:
    if char_map[s[i]] != t[i]:
        return False
    else:
    if t[i] in used_chars:
        return False
    char_map[s[i]] = t[i]
    used_chars.add(t[i])

    return True

s = "egg"
t = "add"

print(isomorphic_strings(s, t))  # Output: True
```

2)

```
def is_strobogrammatic(num):
    valid_digits = {'0', '1', '6', '8', '9'}
    strobogrammatic_pairs = {'0': '0', '1': '1', '6': '9', '8': '8', '9': '6'}

    left = 0
    right = len(num) - 1

    while left <= right:
    if num[left] not in valid_digits or num[right] not in valid_digits:
    return False

    if num[left] != strobogrammatic_pairs[num[right]]:
    return False

    left += 1
    right -= 1

    return True

num = "69"
print(is_strobogrammatic(num))  # Output: True
```

3)

```
def sum_strings(num1, num2):
  """
  Returns the sum of two non-negative integers represented as strings.

  Args:
        num1: The first integer, represented as a string.
        num2: The second integer, represented as a string.

  Returns:
        The sum of num1 and num2, represented as a string.
  """

  # Reverse the strings so that we can add the digits from right to left.
  num1 = num1[::-1]
  num2 = num2[::-1]

  # Initialize the result string.
  result = ""

  # Add the digits from the two strings, one at a time.
  carry = 0
  for i in range(max(len(num1), len(num2))):
        digit1 = int(num1[i]) if i < len(num1) else 0
        digit2 = int(num2[i]) if i < len(num2) else 0
        sum_digit = digit1 + digit2 + carry
        result += str(sum_digit % 10)
        carry = sum_digit // 10

  # If there is any carry left over, append it to the result string.
  if carry > 0:
        result += str(carry)

  # Reverse the result string so that it is in the correct order.
  return result[::-1]
```

4)

```
def reverse_words(s):
        words = s.split()  # Split the sentence into words
        reversed_words = [word[::-1] for word in words]  # Reverse each word

        return ' '.join(reversed_words)  # Join the reversed words back into a sentence with
whitespace
```

5)

```
def reverse_alternate_k_characters(s, k):
```

```
  """Reverses the first k characters for every 2k characters counting from the start of the
string.

  Args:
        s: The string to reverse.
        k: The number of characters to reverse.

  Returns:
        The string with the first k characters reversed for every 2k characters.
  """

  if len(s) < k:
        return s[::-1]

  result = ""
  for i in range(0, len(s), 2 * k):
        result += s[i:i + k][::-1] + s[i + k:i + 2 * k]

  return result
```

6)

```
def is_rotation(s, goal):
  """Returns true if and only if s can be rotated to goal.

  Args:
        s: The string to check.
        goal: The string to compare to.

  Returns:
        True if s can be rotated to goal, False otherwise.
  """

  if len(s) != len(goal):
        return False

  for i in range(len(s)):
        if s[i:] + s[:i] == goal:
        return True

  return False

>>> is_rotation("abcde", "cdeab")
True
```

7)

```
def backspace_compare(s, t):
  """Returns true if and only if s and t are equal after backspaces.
```

```
    Args:
        s: The first string.
        t: The second string.

    Returns:
        True if s and t are equal after backspaces, False otherwise.
    """

    i = 0
    j = 0

    while i < len(s) and j < len(t):
        while i < len(s) and s[i] == '#':
        i += 1

        while j < len(t) and t[j] == '#':
        j += 1

        if i == len(s) or j == len(t):
        return False

        if s[i] != t[j]:
        return False

        i += 1
        j += 1

    return i == len(s) and j == len(t)

>>> backspace_compare("ab#c", "ad#c")
True
```