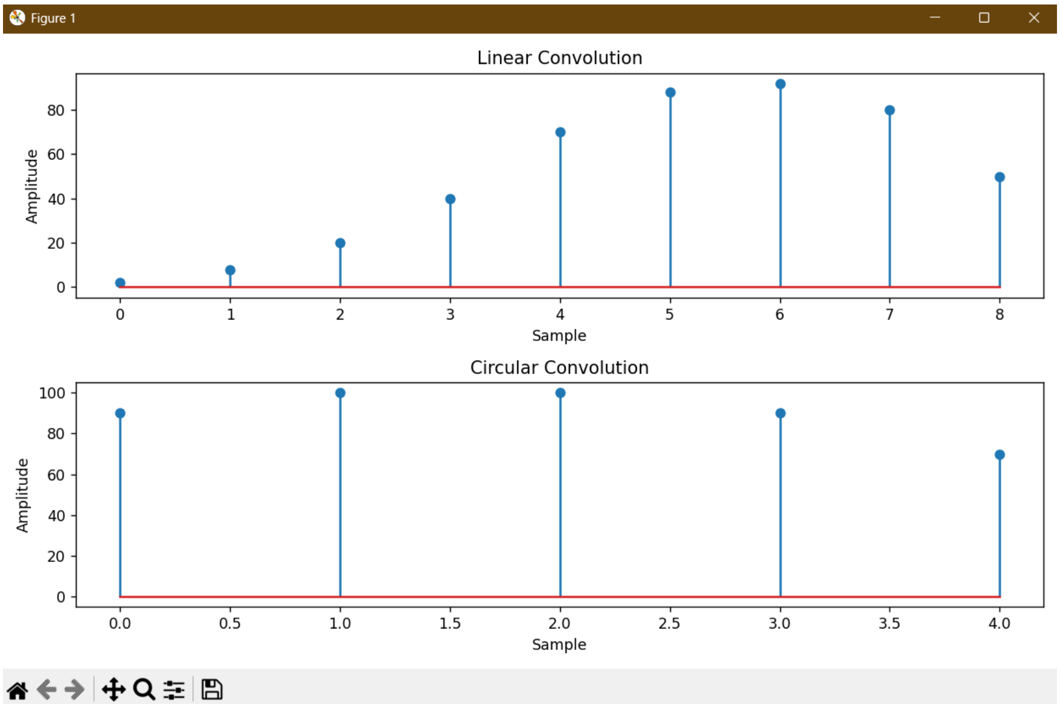
Code:

```
import numpy as np
import matplotlib.pyplot as plt
def linear_convolution(signal1, signal2):
    # Compute the linear convolution
    linear_conv = np.convolve(signal1, signal2, mode='full')
    return linear_conv
def circular_convolution(signal1, signal2):
    # Compute the circular convolution
    if len(signal1) > len(signal2):
        fft_length = len(signal1)
    else:
        fft_length = len(signal2)
    # Pad the shorter signal to match fft_length
    s1 = np.pad(signal1, (0, fft_length - len(signal1)), mode='constant')
    s2 = np.pad(signal2, (0, fft_length - len(signal2)), mode='constant')
    fft_signal1 = np.fft.fft(s1, fft_length)
    fft_signal2 = np.fft.fft(s2, fft_length)
    circular_conv = np.fft.ifft(fft_signal1 * fft_signal2)
    return np.real(circular_conv)
# Define the discrete-time signals
signal1 = np.array([1, 2, 3, 4, 5])
signal2 = np.array([2, 4, 6, 8, 10])
# Compute the linear convolution
linear_conv = linear_convolution(signal1, signal2)
# Compute the circular convolution
circular_conv = circular_convolution(signal1, signal2)
```

```
# Plot the linear and circular convolution results
plt.figure(figsize=(10, 6))
plt.subplot(2, 1, 1)
plt.stem(linear_conv)
plt.title('Linear Convolution')
plt.xlabel('Sample')
plt.ylabel('Amplitude')
plt.subplot(2, 1, 2)
plt.stem(circular_conv)
plt.title('Circular Convolution')
plt.xlabel('Sample')
plt.ylabel('Amplitude')
plt.tight_layout()
plt.show()
```

Output:



Conclusion:

In summary, we applied and contrasted linear and circular convolution for discrete-time signals in this experiment. For non-periodic signals, linear convolution, which is calculated using the direct convolution formula (np.convolve), accurately captures the system's response by producing an output length equal to the sum of the input lengths minus one. Using the Discrete Fourier Transform (FFT) method, circular convolution generates an output of the same length as the longest signal following zero-padding and assumes periodicity of the signals. The findings demonstrate that although the two approaches share mathematical similarities, they handle signal boundaries differently, which makes them appropriate for various digital signal processing applications.