```
import numpy as np

# Define the activation function (step function)
def step_function(x):
    if x >= 0:
        return 1
    else:
        return 0

# Define the logical functions
def logical_NOT(x):
    w = np.array([-1]) # Set the weight to -1
    b = 0 # Set the bias to 0
    x = np.array(x) # Convert the input to a numpy array
    z = np.dot(w, x) + b # Calculate the weighted sum
    y = step_function(z) # Apply the activation function
    return y

def logical_NAND(x):
    w = np.array([-1, -1]) # Set the weights to -1
    b = 2 # Set the bias to 2
    x = np.array(x) # Convert the input to a numpy array
    z = np.dot(w, x) + b # Calculate the weighted sum
    y = step_function(z) # Apply the activation function
    return y

def logical_NOR(x):
    w = np.array([-1, -1]) # Set the weights to -1
    b = 0 # Set the bias to 0
    x = np.array(x) # Convert the input to a numpy array
    z = np.dot(w, x) + b # Calculate the weighted sum
    y = step_function(z) # Apply the activation function
    return y

# Test the functions
print(logical_NOT(0)) # Should output 1
print(logical_NOT(1)) # Should output 0

print(logical_NAND([0, 0])) # Should output 1
print(logical_NAND([0, 1])) # Should output 1
print(logical_NAND([1, 0])) # Should output 1
print(logical_NAND([1, 1])) # Should output 0

print(logical_NOR([0, 0])) # Should output 1
print(logical_NOR([0, 1])) # Should output 0
print(logical_NOR([1, 0])) # Should output 0
print(logical_NOR([1, 1])) # Should output 0
```

```
1
0
1
1
1
1
1
0
0
0
```