```
#Step 1: generate a vector of inputs and a vector of weights
import numpy as np
np.random.seed(seed=0)
I = np.random.choice([0,1], 3)# generate random vector I, sampling from {0,1}
W = np.random.choice([-1,1], 3) # generate random vector W, sampling from {-1,1}
print(f'Input vector:{I}, Weight vector:{W}')


#Step 2: compute the dot product between the vector of inputs and weights

dot = I @ W
print(f'Dot product: {dot}')

#Step 3: define the threshold activation function

def linear_threshold_gate(dot: int, T: float) -> int:
    '''Returns the binary threshold output'''
    if dot >= T:
        return 1
    else:
        return 0

#Step 4: compute the output based on the threshold value

T = 1
activation = linear_threshold_gate(dot, T)
print(f'Activation: {activation}')

T = 3
activation = linear_threshold_gate(dot, T)
print(f'Activation: {activation}')


#-----------The AND Function-----------

#Step 1: generate a vector of inputs and a vector of weights
# matrix of inputs
input_table = np.array([
    [0,0], # both no
    [0,1], # one no, one yes
    [1,0], # one yes, one no
    [1,1]  # bot yes
])

print(f'input table:\n{input_table}')

# array of weights
weights = np.array([1,1])
print(f'weights: {weights}')
```

```
#Step 2: compute the dot product between the matrix of inputs and weights

# dot product matrix of inputs and weights
dot_products = input_table @ weights
print(f'Dot products: {dot_products}')


#Step 3: define the threshold activation function
#We defined this already, so we will reuse our linear_threshold_gate function

#Step 4: compute the output based on the threshold value

T = 2
for i in range(0,4):
    activation = linear_threshold_gate(dot_products[i], T)
    print(f'Activation: {activation}')


#---------------------The OR Function---------------------

#Step 1: generate a vector of inputs and a vector of weights
#Neither the matrix of inputs nor the array of weights changes, so we can reuse our
input_table and weights vector.

#Step 2: compute the dot product between the matrix of inputs and weights
#Since neither the matrix of inputs nor the vector of weights changes, the dot
product of those stays the same.

#Step 3: define the threshold activation function
#We can use the linear_threshold_gate function again.

# Step 4: compute the output based on the threshold value

T = 1
for i in range(0,4):
    activation = linear_threshold_gate(dot_products[i], T)
    print(f'Activation: {activation}')



#-----------------The NOR function-------------------

#Step 1: generate a vector of inputs and a vector of weights
# The matrix of inputs remain the same, but we need a new vector of weights

# array of weights
weights = np.array([-1,-1])
print(f'weights: {weights}')
```

```
#Step 2: compute the dot product between the matrix of inputs and weights
# dot product matrix of inputs and weights
dot_products = input_table @ weights
print(f'Dot products: {dot_products}')

#Step 3: define the threshold activation function
#The function remains the same.

#Step 4: compute the output based on the threshold value

T = 0
for i in range(0,4):
    activation = linear_threshold_gate(dot_products[i], T)
    print(f'Activation: {activation}')
```