LAB 1

```
# importing Python library
import numpy as np

# define Unit Step Function Activation function
def unitStep(v):
    if v >= 0:
        return 1
    else:
        return 0

# design Perceptron Model
def perceptronModel(x, w, b):
    v = np.dot(w, x) + b
    y = unitStep(v)
    return y

# AND Logic Function
# w1 = 1, w2 = 1, b = -1.5
def AND_logicFunction(x):
    w = np.array([1, 1])
    b = -1.5
    return perceptronModel(x, w, b)


# testing the Perceptron Model
test1 = np.array([0, 0])
test2 = np.array([0, 1])
test3 = np.array([1, 0])
test4 = np.array([1, 1])

print("AND(0, 0) =",AND_logicFunction(test1))
print("AND(0, 1) =",AND_logicFunction(test2))
print("AND(1, 0) =",AND_logicFunction(test3))
print("AND(1, 1) =",AND_logicFunction(test4))

#print("AND({}, {}) = {}".format(0, 0, AND_logicFunction(test1)))
#print("AND({}, {}) = {}".format(0, 1, AND_logicFunction(test2)))
#print("AND({}, {}) = {}".format(1, 0, AND_logicFunction(test3)))
#print("AND({}, {}) = {}".format(1, 1, AND_logicFunction(test4)))
```

```
AND(0, 0) = 0
AND(0, 1) = 0
AND(1, 0) = 0
AND(1, 1) = 1
```

```
def OR_logicFunction(x):
    w = np.array([1, 1])
    b = -0.5
    return perceptronModel(x, w, b)

# testing the Perceptron Model
test1 = np.array([0, 0])
test2 = np.array([0, 1])
test3 = np.array([1, 0])
test4 = np.array([1, 1])

print("OR(0, 0) =",OR_logicFunction(test1))
print("OR(0, 1) =",OR_logicFunction(test2))
print("OR(1, 0) =",OR_logicFunction(test3))
print("OR(1, 1) =",OR_logicFunction(test4))

#print("OR({}, {}) = {}".format(0, 1, OR_logicFunction(test1)))
#print("OR({}, {}) = {}".format(0, 1, OR_logicFunction(test2)))
#print("OR({}, {}) = {}".format(1, 0, OR_logicFunction(test3)))
#print("OR({}, {}) = {}".format(1, 1, OR_logicFunction(test4)))
```

```
OR(0, 0) = 0
OR(0, 1) = 1
OR(1, 0) = 1
OR(1, 1) = 1
```

```
# importing Python library
import numpy as np

# define Unit Step Function* Activation function
def unitStep(v):
    if v >= 0:
```

```
        return 1
    else:
        return 0

# design Perceptron Model
def perceptronModel(x, w, b):
    v = np.dot(w, x) + b
    y = unitStep(v)
    return y

# NAND Logic Function
# w1 =-1, w2 = -1, b = 1.5
def NAND_logicFunction(x):
    w = np.array([-1, -1])
    b = 1.5
    return perceptronModel(x, w, b)


# testing the Perceptron Model
test1 = np.array([0, 0])
test2 = np.array([0, 1])
test3 = np.array([1, 0])
test4 = np.array([1, 1])

print("NAND({}, {}) = {}".format(0, 0, NAND_logicFunction(test1)))
print("NAND({}, {}) = {}".format(0, 1, NAND_logicFunction(test2)))
print("NAND({}, {}) = {}".format(1, 0, NAND_logicFunction(test3)))
print("NAND({}, {}) = {}".format(1, 1, NAND_logicFunction(test4)))
```

```
NAND(0, 0) = 1
NAND(0, 1) = 1
NAND(1, 0) = 1
NAND(1, 1) = 0
```

```
def XOR_logicFunction(x):
    w = np.array([1, 1])
    b = -1.5
    h1=OR_logicFunction(x)
    h2=NAND_logicFunction(x)
    final_x=np.array([h1, h2])
    final_output=AND_logicFunction(final_x)
    return final_output


import numpy as np
a = [[1, 0], [0, 1]]
print(a)
b = [[4, 1], [2, 2]]
print(b)
np.dot(a, b)
```

```
[[1, 0], [0, 1]]
[[4, 1], [2, 2]]
array([[4, 1],
       [2, 2]])
```

```
test1 = np.array([0, 0])
test2 = np.array([0, 1])
test3 = np.array([1, 0])
test4 = np.array([1, 1])

print("XOR({}, {}) = {}".format(0, 0, XOR_logicFunction(test1)))
print("XOR({}, {}) = {}".format(0, 1, XOR_logicFunction(test2)))
print("XOR({}, {}) = {}".format(1, 0, XOR_logicFunction(test3)))
print("XOR({}, {}) = {}".format(1, 1, XOR_logicFunction(test4)))
```

```
XOR(0, 0) = 0
XOR(0, 1) = 1
XOR(1, 0) = 1
XOR(1, 1) = 0
```