CODE:

In [1]:
```
import numpy as np
def unitStep(v):
    if v >= 0:
        return 1
    else:
        return 0
```

In [2]:
```
def perceptronModel(x, w, b):
    v = np.dot(w, x) + b
    y = unitStep(v)
    return y
```

In [3]:
```
def NOT_logicFunction(x):
    w = -1
    b = 0.5
    return perceptronModel(x, w, b)
```

In [4]:
```
def OR_logicFunction(x):
    w = np.array([1, 1])
    bOR = -0.5
    return perceptronModel(x, w, bOR)
```

In [5]:
```
def NOR_logicFunction(x):
    output_OR = OR_logicFunction(x)
    output_NOT = NOT_logicFunction(output_OR)
    return output_NOT
```

In [6]:
```
def AND_logicFunction(x):
    w = np.array([1, 1])
    bAND = -1.5
    return perceptronModel(x, w, bAND)
```

In [7]:
```
def NAND_logicFunction(x):
    output_AND = AND_logicFunction(x)
    output_NOT = NOT_logicFunction(output_AND)
    return output_NOT
```

In [8]:
```
not1 = np.array(1)
not2 = np.array(0)
test1 = np.array([0, 1])
test2 = np.array([1, 1])
test3 = np.array([0, 0])
test4 = np.array([1, 0])
```

In [9]:
```
print("NOT({}) = {}".format(1, NOT_logicFunction(not1)))
print("NOT({}) = {}".format(0, NOT_logicFunction(not2)))
```

```
NOT(1) = 0
NOT(0) = 1
```

In [10]:
```
print("NAND({}, {}) = {}".format(0, 1, NAND_logicFunction(test1)))
print("NAND({}, {}) = {}".format(1, 1, NAND_logicFunction(test2)))
print("NAND({}, {}) = {}".format(0, 0, NAND_logicFunction(test3)))
print("NAND({}, {}) = {}".format(1, 0, NAND_logicFunction(test4)))
```

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

In [11]:
```
print("NOR({}, {}) = {}".format(0, 1, NOR_logicFunction(test1)))
print("NOR({}, {}) = {}".format(1, 1, NOR_logicFunction(test2)))
print("NOR({}, {}) = {}".format(0, 0, NOR_logicFunction(test3)))
print("NOR({}, {}) = {}".format(1, 0, NOR_logicFunction(test4)))
```

```
NOR(0, 1) = 0
NOR(1, 1) = 0
NOR(0, 0) = 1
NOR(1, 0) = 0
```

RESULT:

Hence, we implemented a Single Layer Perceptron to construct logical NOT, NAND and NOR successfully.