In [1]:
```
# importing Python library
import numpy as np
```

In [2]:
```
# define Unit Step Function
def unitStep(v):
    if v >= 0:
        return 1
    else:
        return 0
```

In [3]:
```
# design Perceptron Model
def perceptronModel(x, w, b):
    v = np.dot(w, x) + b  # weighted sum
    y = unitStep(v)  # apply the activation function (unit step)
    return y
```

In [4]:
```
# OR Logic Function
# w1 = 1, w2 = 1, b = -0.5
def OR_logicFunction(x):
    w = np.array([1, 1])  # weights for the OR gate
    b = -0.5  # bias term
    return perceptronModel(x, w, b)
```

In [5]:
```
# AND Logic Function
# w1 = 1, w2 = 1, b = -1.5
def AND_logicFunction(x):
    w = np.array([1, 1])  # weights for the AND gate
    b = -1.5  # bias term
    return perceptronModel(x, w, b)
```

In [6]:
```
# testing the Perceptron Model for OR
print("Testing OR Logic Function:")
test1 = np.array([0, 1])
test2 = np.array([1, 1])
test3 = np.array([0, 0])
test4 = np.array([1, 0])

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

```
Testing OR Logic Function:
OR(0, 1) = 1
OR(1, 1) = 1
OR(0, 0) = 0
OR(1, 0) = 1
```

In [7]:
```
# testing the Perceptron Model for AND
print("\nTesting AND Logic Function:")
test1 = np.array([0, 1])
test2 = np.array([1, 1])
test3 = np.array([0, 0])
test4 = np.array([1, 0])

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

```
Testing AND Logic Function:
AND(0, 1) = 0
AND(1, 1) = 1
AND(0, 0) = 0
AND(1, 0) = 0
```