In [3]:

```
# importing Python library
import numpy as np

# define Unit Step 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

# OR Logic Function
# w1 = 1, w2 = 1, b = -0.5
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, 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)))
```

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

In [4]:

```
# importing Python library
import numpy as np

# define Unit Step 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, 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)))
```

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

In [5]:

```
# importing Python library
import numpy as np

# define Unit Step 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

# NOT Logic Function
# wNOT = -1, bNOT = 0.5
def NOT_logicFunction(x):
 wNOT = -1
 bNOT = 0.5
 return perceptronModel(x, wNOT, bNOT)

# AND Logic Function
# here w1 = wAND1 = 1,
# w2 = wAND2 = 1, bAND = -1.5
def AND_logicFunction(x):
 w = np.array([1, 1])
 bAND = -1.5
 return perceptronModel(x, w, bAND)

# OR Logic Function
# w1 = 1, w2 = 1, bOR = -0.5
def OR_logicFunction(x):
 w = np.array([1, 1])
 bOR = -0.5
 return perceptronModel(x, w, bOR)

# XOR Logic Function
# with AND, OR and NOT
# function calls in sequence
def XOR_logicFunction(x):
 y1 = AND_logicFunction(x)
 y2 = OR_logicFunction(x)
 y3 = NOT_logicFunction(y1)
 final_x = np.array([y2, y3])
 finalOutput = AND_logicFunction(final_x)
 return finalOutput

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

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

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

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