```
print("Hello")
```

```
Hello
```

AND GATE USING WEIGHTS

```
inp = [[0,0],[0,1],[1,0],[1,1]]
bias = -1
weight = 1
```

```
def AND (inp):
  for i in inp:
    result = i[0]*i[1]*weight + bias
    if(result==0):
      print("1")
    else:
      print('0')
```

```
AND(inp)
```

```
0
0
0
1
```

OR GATE USING WEIGHTS

```
inp = [[0,0],[0,1],[1,0],[1,1]]
bias = -1
weight = 1
```

```
def OR (inp):
  for i in inp:
    result = i[0]+i[1]*weight + bias
    if(result<0):
      print("0")
    else:
      print('1')
```

```
OR(inp)
```

```
0
1
1
1
```

NOT GATE USING WEIGHTS

```
inp = [0,1]
bias = -1
weight = 1
```

```
def NOT(inp):
  for i in inp:
    result = i*weight + bias
    if(result==0):
      print("0")
    else:
      print('1')
```

```
NOT(inp)
```

```
1
0
```

NAND GATE USING WEIGHTS

```
# NOT(AND(input))
```

```
inp = [[0,0],[0,1],[1,0],[1,1]]
bias = -1
weight = 1
```

```
def AND_NAND(inp):
  ans = []
  for i in inp:
    result = i[0]*i[1]*weight + bias
    if(result==0):
      ans.append(1)
    else:
      ans.append(0)
  return ans
```

```
def NOT_NAND(inp):
  ans2 = []
  for i in inp:
    result = i*weight + bias
    if(result==0):
      ans2.append(0)
    else:
      ans2.append(1)
  return ans2
```

```
result = AND_NAND(inp)
```

```
NOT_NAND(result)
```

```
     [1, 1, 1, 0]
```

NOR USING WEIGHTS

```
inp = [[0,0],[0,1],[1,0],[1,1]]
bias = -1
weight = 1
```

```
def OR_NOR (inp):
  ans1 = []
  for i in inp:
    result = i[0]+i[1]*weight + bias
    if(result<0):
      ans1.append(0)
    else:
      ans1.append(1)
  return ans1
```

```
def NOT_NOR(inp):
  ans2 = []
  for i in inp:
    result = i*weight + bias
    if(result==0):
      ans2.append(0)
    else:
      ans2.append(1)
  return ans2
```

```
result = OR_NOR(inp)
print(result)
```

```
     [0, 1, 1, 1]
```

```
print(NOT_NOR(result))

     [1, 0, 0, 0]
```

XOR GATE USING WEIGHTS

```
inp = [[0,0],[0,1],[1,0],[1,1]]

def XOR(inp):
  ans = []
  for i in inp:
    if(i[0]==i[1]):
      ans.append(0)
    else:
      ans.append(1)
  return ans


result = XOR(inp)
print(result)

     [0, 1, 1, 0]
```

XNOR GATE

```
inp = [[0,0],[0,1],[1,0],[1,1]]

def XNOR(inp):
  result = XOR(inp)
  return result


result = XNOR(inp)
print(result)

     [0, 1, 1, 0]
```

```
NOT(result)

     1
     0
     0
     1
```

QUESTION 1

Implement Perceptron learning algorithm for classification of following points {P0(-1,-1,-1) , P1(-1,-1,1) , P2(-1,1,-1) , P3(-1,1,1) ,P4(1,-1,-1) , P5(1,-1,1) , P6(1,1,-1) , P7(1,1,1) } in to two classes:

C1={P7 (1,1,1)}

C2={P0(-1,-1,-1) , P1(-1,-1,1) , P2(-1,1,-1) , P3(-1,1,1) ,P4(1,-1,-1) , P5(1,-1,1) , P6(1,1,-1) }

```
import numpy as np

data_points = {
    'p0': np.array([-1,-1,-1]),
    'p1': np.array([-1,-1,1]),
    'p2': np.array([-1,1,-1]),
    'p3': np.array([-1,1,1]),
    'p4': np.array([1,-1,-1]),
    'p5': np.array([1,-1,1]),
    'p6': np.array([1,1,-1]),
    'p7': np.array([1,1,1])
}


c1 = ['p7']
c2 = ['p0','p1','p2','p3','p4','p5','p6'] #positive class
```

```
weights = np.zeros(len(data_points['p0']))
bias = 0
print(weights)
```

```
[0. 0. 0.]
```

Learning Rate

```
#define learning rate
learning_rate = 1
```

```
#define number of iterations
epochs = 100
```

```
for epoch in range(epochs):
  for point in c2:
    x = data_points[point]
    y = 1

    #calculate prediction
    summ = np.dot(weights, x) + bias

    #update weights and bias on prediction
    if summ <= 0:
      weights += learning_rate * x
      bias += learning_rate
```

```
weights
```

```
array([-1., -1., -1.])
```

```
new_point = np.array([1,1,1])
summ = np.dot(weights, new_point) + bias
```

```
#classify new point
if summ > 0:
  print(f'The new point {new_point} belongs to class c2')
else:
  print(f'The new point {new_point} belongs to class c1')
```

```
The new point [1 1 1] belongs to class c1
```

QUESTION 1B Write a python program to find the number of linearly separable problems out of total binary classification problems on {P0(-1,-1,-1), P1(-1,-1,1) , P2(-1,1,-1) , P3(-1,1,1) ,P4(1,-1,-1), P5(1,-1,1) , P6(1,1,-1) , P7(1,1,1) }.

```
import numpy as np
from itertools import combinations

# Define the points
points = {
    'p0': np.array([-1,-1,-1]),
    'p1': np.array([-1,-1,1]),
    'p2': np.array([-1,1,-1]),
    'p3': np.array([-1,1,1]),
    'p4': np.array([1,-1,-1]),
    'p5': np.array([1,-1,1]),
    'p6': np.array([1,1,-1]),
    'p7': np.array([1,1,1])
}

# Define the Perceptron learning algorithm
def perceptron_learning_algorithm(c1, c2, epochs=100, learning_rate=1):
    weights = np.zeros(len(points['p0']))
    bias = 0
    for epoch in range(epochs):
        for point in c2:
            x = points[point]
            y = 1
            summ = np.dot(weights, x) + bias
            if summ <= 0:
                weights += learning_rate * x
                bias += learning_rate
    return weights, bias

# Count the number of linearly separable problems
count = 0
total = 0
for r in range(1, len(points)):
    for c1 in combinations(points.keys(), r):
        c1 = list(c1)
        c2 = [p for p in points.keys() if p not in c1]
        weights, bias = perceptron_learning_algorithm(c1, c2)
        if all(np.dot(weights, points[p]) + bias > 0 for p in c2) and all(np.dot(weights, points[p]) + bias <= 0 for p in c1):
            count += 1
        total += 1

print(f'Number of linearly separable problems: {count}')
print(f'Total binary classification problems: {total}')
```

```
Number of linearly separable problems: 9
Total binary classification problems: 254
```