OUTPUTS

1. Implement a perceptron training algorithm.

a. Compute 2-variables AND, OR, and NOT (1-variable) operations and report the number of steps (number of weight-updates) required for the convergence.

i. AND

```
C:\WINDOWS\system32\cmd.exe

C:\Users\Dushyant Panchal\Desktop\DL_A1>python -W ignore Q1_AND.py
Weights: [-4.  3.  2.]
Total Updates: 18
```

Note: The weights are in the order b, w1, w2

ii. OR

```
C:\WINDOWS\system32\cmd.exe

C:\Users\Dushyant Panchal\Desktop\DL_A1>python -W ignore Q1_OR.py
Weights: [-1.  2.  2.]
Total Updates: 9
```

Note: The weights are in the order b, w1, w2

iii. NOT

```
C:\WINDOWS\system32\cmd.exe

C:\Users\Dushyant Panchal\Desktop\DL_A1>python -W ignore Q1_NOT.py
Weights: [ 1. -2.]
Total Updates: 5
```

Note: The weights are in the order b, w1

2. Using the Madeleine learning algorithm compute the following two functions. Shaded regions are 1, rest are 0. Report the number of neurons for each case.

a. f1 (x1, x2)

```
count of -1/0 =  32  count of 1 =  17
Accuracy =>  60.49382716049383
count of -1/0 =  31  count of 1 =  16
Accuracy =>  58.0246913580247
count of -1/0 =  28  count of 1 =  13
Accuracy =>  50.617283950617285
count of -1/0 =  20  count of 1 =  32
64.19753086419753
```

```
[143] import pickle
      pickle.dump(nn, open("Madaline_best","wb"))
```

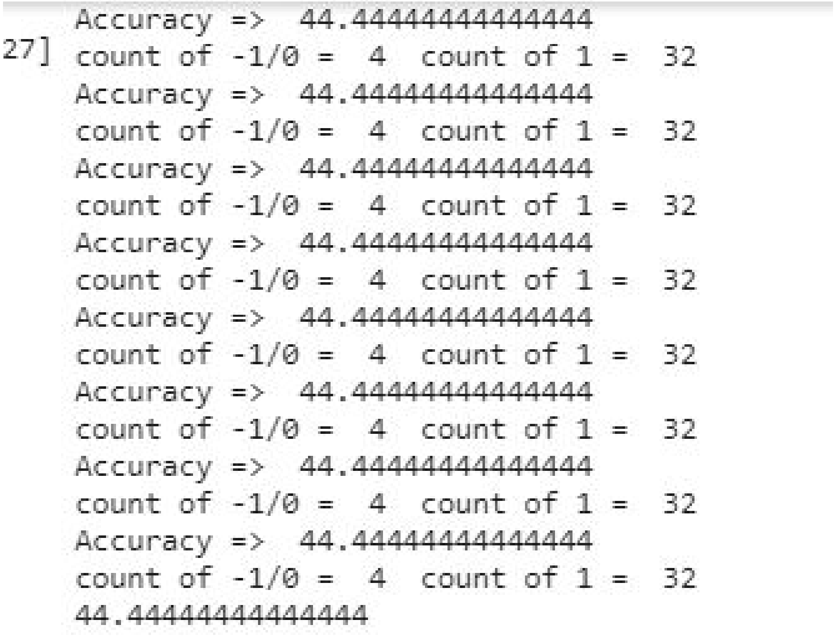
```
best_model = pickle.load(open("Madaline_best","rb"))
best_model.test(x_train,y_train)
```

```
count of -1/0 =  20  count of 1 =  32
64.19753086419753
```

b. f2 (x1, x2)

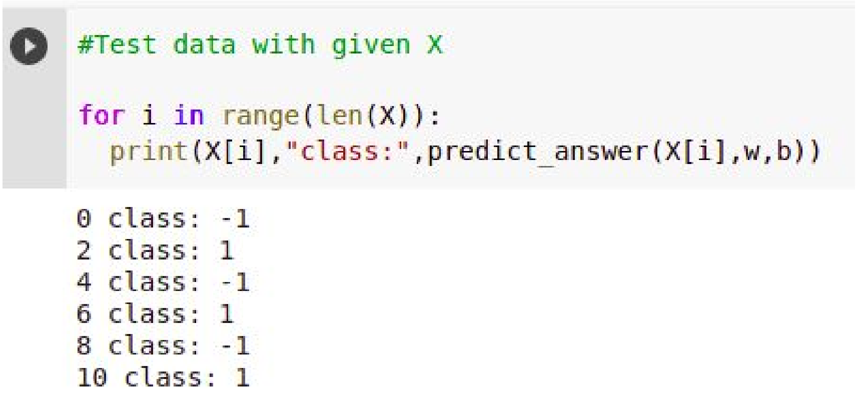
```
Accuracy =>  60.49382716049383
count of -1/0 =  32  count of 1 =  15
Accuracy =>  58.0246913580247
count of -1/0 =  31  count of 1 =  14
Accuracy =>  55.55555555555556
count of -1/0 =  33  count of 1 =  16
Accuracy =>  60.49382716049383
count of -1/0 =  32  count of 1 =  16
Accuracy =>  59.25925925925925
count of -1/0 =  37  count of 1 =  19
Accuracy =>  69.1358024691358
count of -1/0 =  36  count of 1 =  19
Accuracy =>  67.90123456790124
count of -1/0 =  33  count of 1 =  17
Accuracy =>  61.72839506172839
count of -1/0 =  30  count of 1 =  13
Accuracy =>  53.086419753086425
count of -1/0 =  34  count of 1 =  16
Accuracy =>  61.72839506172839
count of -1/0 =  28  count of 1 =  12
Accuracy =>  49.382716049382715
count of -1/0 =  49  count of 1 =  0
60.49382716049383
```

c. In comparison with f1, can you compute f2 in <= 2 more neurons? Justify and implement it.



```
    Accuracy =>  44.44444444444444
27] count of -1/0 =  4  count of 1 =  32
    Accuracy =>  44.44444444444444
    count of -1/0 =  4  count of 1 =  32
    Accuracy =>  44.44444444444444
    count of -1/0 =  4  count of 1 =  32
    Accuracy =>  44.44444444444444
    count of -1/0 =  4  count of 1 =  32
    Accuracy =>  44.44444444444444
    count of -1/0 =  4  count of 1 =  32
    Accuracy =>  44.44444444444444
    count of -1/0 =  4  count of 1 =  32
    Accuracy =>  44.44444444444444
    count of -1/0 =  4  count of 1 =  32
    Accuracy =>  44.44444444444444
    count of -1/0 =  4  count of 1 =  32
    44.44444444444444
```

3. Implement a single-neuron neural network to compute the following function y = f(x). You can use the generalized delta rule for learning. If you think it's not possible, justify your claims properly.



```
#Test data with given X

for i in range(len(X)):
  print(X[i],"class:",predict_answer(X[i],w,b))
```

```
0 class: -1
2 class: 1
4 class: -1
6 class: 1
8 class: -1
10 class: 1
```

```
print(w,b)
```

```
1.5707963267948994 1.570796326794877
```