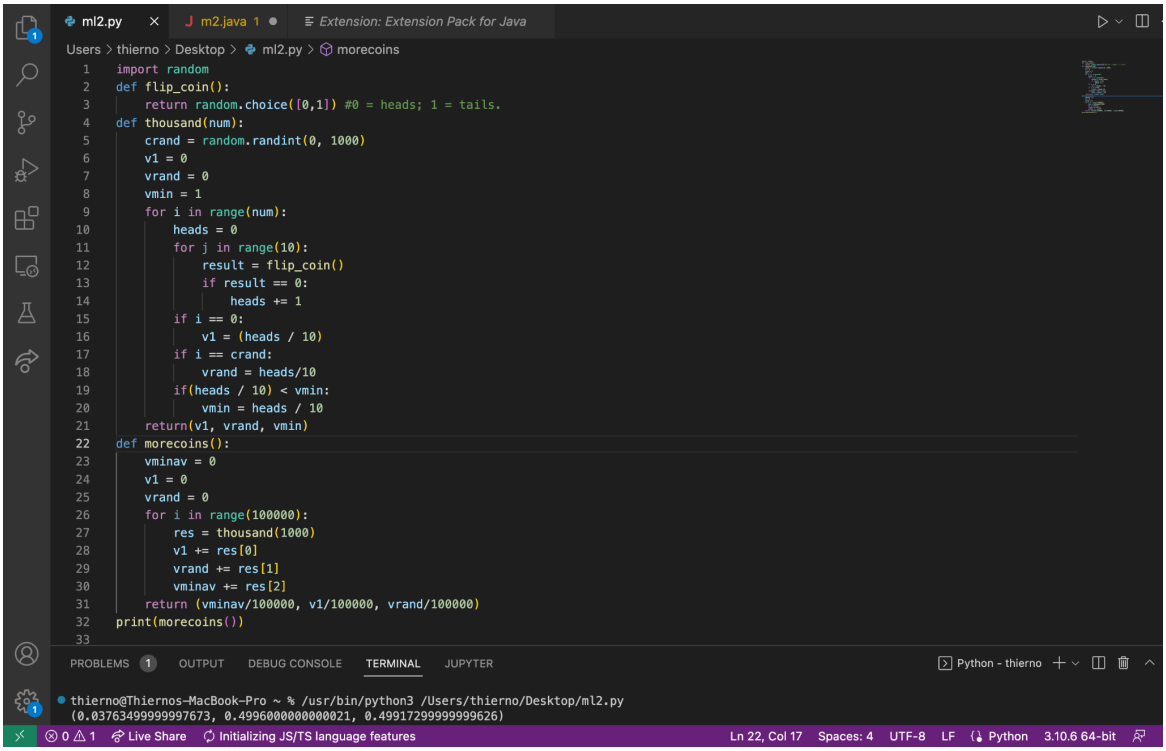
Closest answer = choice - answer that is closest to zero

1.

a.



b. Closest answer = choice - answer closest to zero thus:

c. [a] $0 - 0.0377 = -0.0376$

d. [b] $0.01 - 0.0377 = -0.0276$

e. [c] $0.1 - 0.0377 = 0.0624$

f. [d] $0.5 - 0.0377 = 0.4624$

g. [e] $0.67 - 0.0377 = 6324$

h. Hence the correct answer is b as it's closest to zero.

2. We know that we have a fair coin thus mu should be 0.5 for the single bin Hoeffding Inequality hence looking at the probability distributions above from question one's code, we see that C1 and the random coin are closest approximations of mu hence they satisfy the single bin Hoeffding Inequality. As we have a fair coin the probability of getting heads should be 0.50 for all coins but only C1 and the random coin distributions approximate that as they are about 0.5 each thus both satisfying the inequality.

a. [a] Wrong because it does not match the description above.

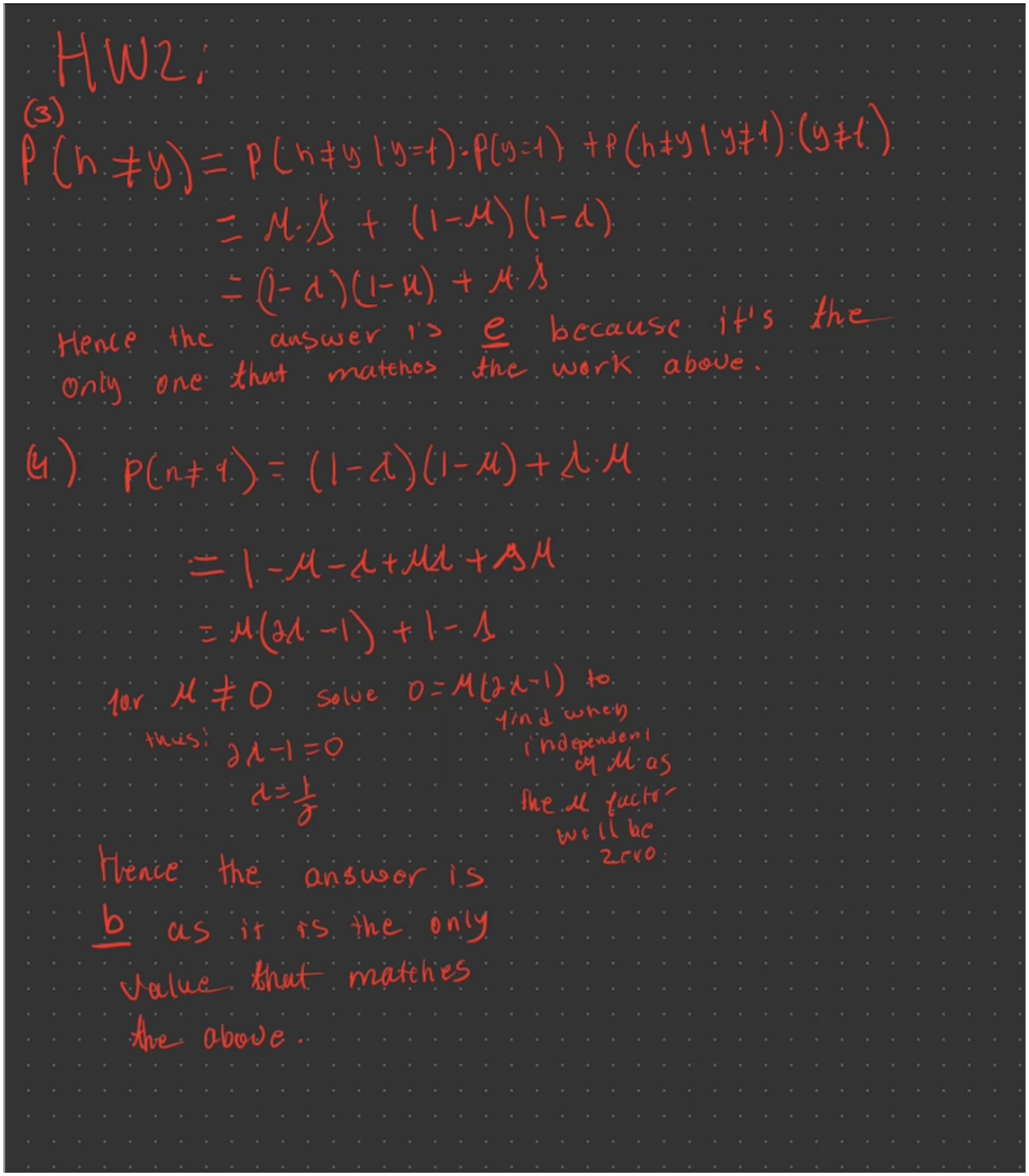
b. [b] Wrong because it does not match the description above.

c. [c] Wrong because it does not match the description above.

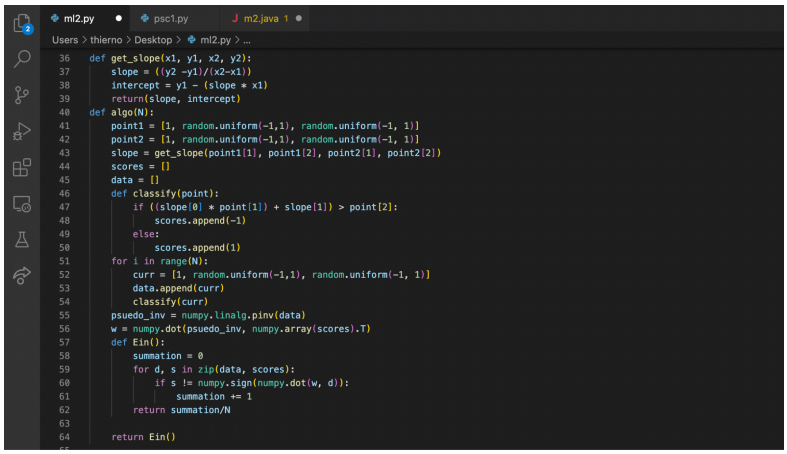
d. [d] Correct because it matches the description above.

e. [e] Wrong because it does not match the description above.

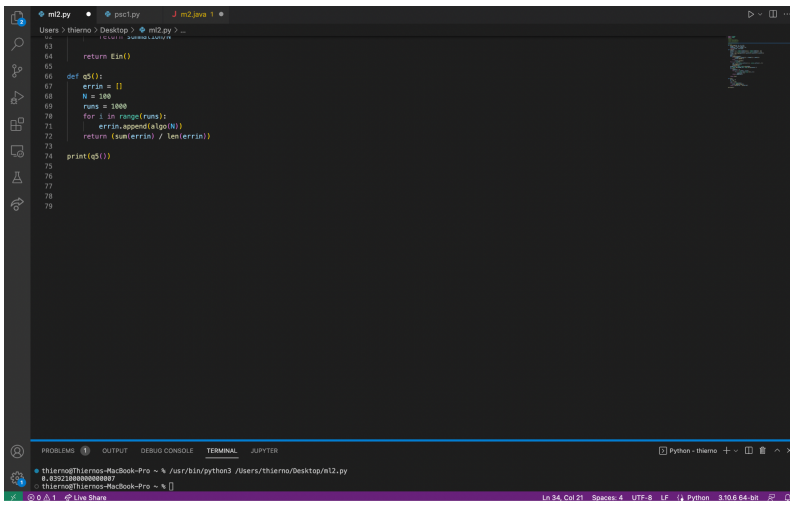
3. For notation purposes let Lambda = L in this question. We know that 3 and 4:



5:



```python
def get_slope(x1, y1, x2, y2):
    slope = ((y2 -y1)/(x2-x1))
    intercept = y1 - (slope * x1)
    return(slope, intercept)
def algo(N):
    point1 = [1, random.uniform(-1,1), random.uniform(-1, 1)]
    point2 = [1, random.uniform(-1,1), random.uniform(-1, 1)]
    slope = get_slope(point1[1], point1[2], point2[1], point2[2])
    scores = []
    data = []
    def classify(point):
        if ((slope[0] * point[1]) + slope[1]) > point[2]:
            scores.append(-1)
        else:
            scores.append(1)
    for i in range(N):
        curr = [1, random.uniform(-1,1), random.uniform(-1, 1)]
        data.append(curr)
        classify(curr)
    psuedo_inv = numpy.linalg.pinv(data)
    w = numpy.dot(psuedo_inv, numpy.array(scores).T)
    def Ein():
        summation = 0
        for d, s in zip(data, scores):
            if s != numpy.sign(numpy.dot(w, d)):
                summation += 1
        return summation/N

    return Ein()
```

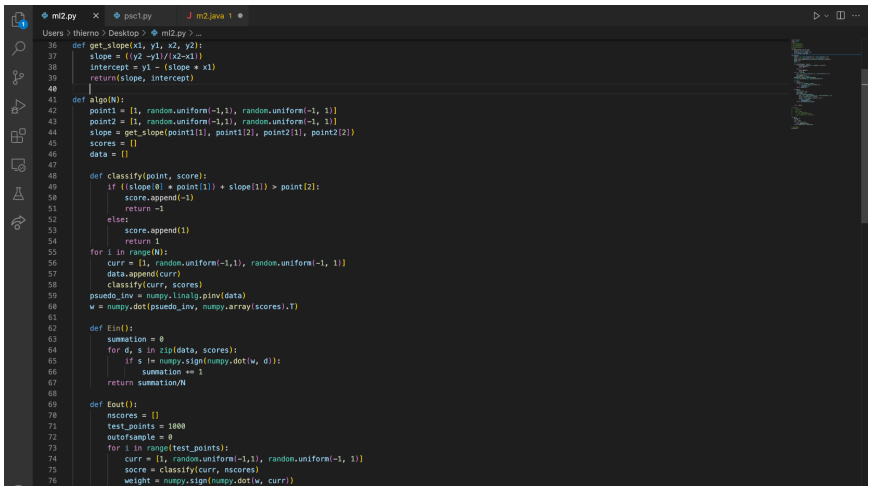


```python
    return Ein()

def q5():
    errin = []
    N = 100
    runs = 1000
    for i in range(runs):
        errin.append(algo(N))
    return (sum(errin) / len(errin))

print(q5())
```

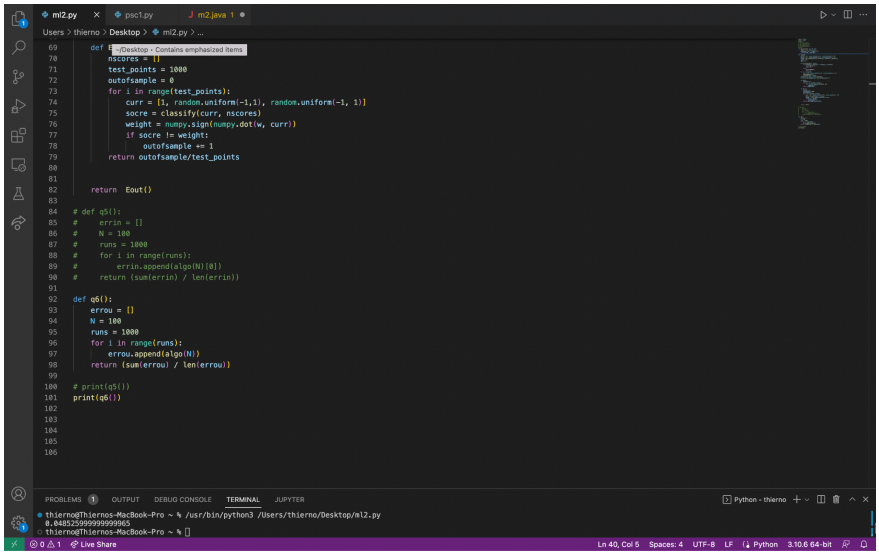
```
thierno@Thiernos-MacBook-Pro ~ % /usr/bin/python3 /Users/thierno/Desktop/ml2.py
0.0392100000000007
thierno@Thiernos-MacBook-Pro ~ %
```

We get an Ein = 0.0392
[a] 0 - Ein = −0.0392
[b] 0.001 - Ein = −0.0382
[c] 0.01 - Ein = −0.0292
[d] 0.1 - Ein = 0.0608
[e] 0.5 - Ein = 0.4608
Hence the answer is **C** because it is closest to zero.

6.



```python
def get_slope(x1, y1, x2, y2):
    slope = ((y2 -y1)/(x2-x1))
    intercept = y1 - (slope * x1)
    return(slope, intercept)

def algo(N):
    point1 = [1, random.uniform(-1,1), random.uniform(-1, 1)]
    point2 = [1, random.uniform(-1,1), random.uniform(-1, 1)]
    slope = get_slope(point1[1], point1[2], point2[1], point2[2])
    scores = []
    data = []

    def classify(point, score):
        if ((slope[0] * point[1]) + slope[1]) > point[2]:
            score.append(-1)
            return -1
        else:
            score.append(1)
            return 1
    for i in range(N):
        curr = [1, random.uniform(-1,1), random.uniform(-1, 1)]
        data.append(curr)
        classify(curr, scores)
    psuedo_inv = numpy.linalg.pinv(data)
    w = numpy.dot(psuedo_inv, numpy.array(scores).T)

    def Ein():
        summation = 0
        for d, s in zip(data, scores):
            if s != numpy.sign(numpy.dot(w, d)):
                summation += 1
        return summation/N

    def Eout():
        nscores = []
        test_points = 1000
        outofsample = 0
        for i in range(test_points):
            curr = [1, random.uniform(-1,1), random.uniform(-1, 1)]
            socre = classify(curr, nscores)
            weight = numpy.sign(numpy.dot(w, curr))
```



```python
    def E  ~/Desktop - Contains emphasized items
        nscores = []
        test_points = 1000
        outofsample = 0
        for i in range(test_points):
            curr = [1, random.uniform(-1,1), random.uniform(-1, 1)]
            socre = classify(curr, nscores)
            weight = numpy.sign(numpy.dot(w, curr))
            if socre != weight:
                outofsample += 1
        return outofsample/test_points


    return  Eout()

# def q5():
#     errin = []
#     N = 100
#     runs = 1000
#     for i in range(runs):
#         errin.append(algo(N)[0])
#     return (sum(errin) / len(errin))

def q6():
    errou = []
    N = 100
    runs = 1000
    for i in range(runs):
        errou.append(algo(N))
    return (sum(errou) / len(errou))

# print(q5())
print(q6())
```

```
thierno@Thiernos-MacBook-Pro ~ % /usr/bin/python3 /Users/thierno/Desktop/ml2.py
0.048529999999999965
thierno@Thiernos-MacBook-Pro ~ %
```

Eout = 0.0485

[a] 0 - Eout = -0.0485

[b] 0.001 - Eout = -0.0475

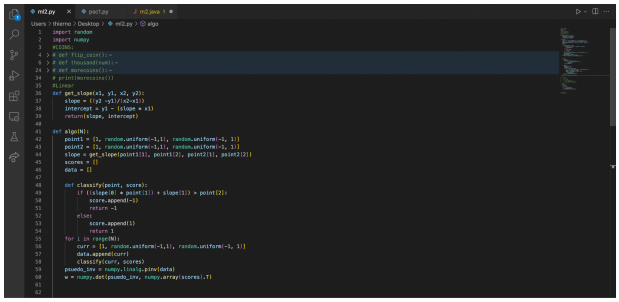
[c] 0.01 - Eout = -0.0385

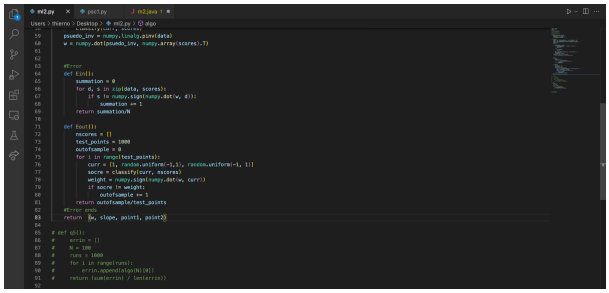
[d] 0.1 - Eout = 0.0515

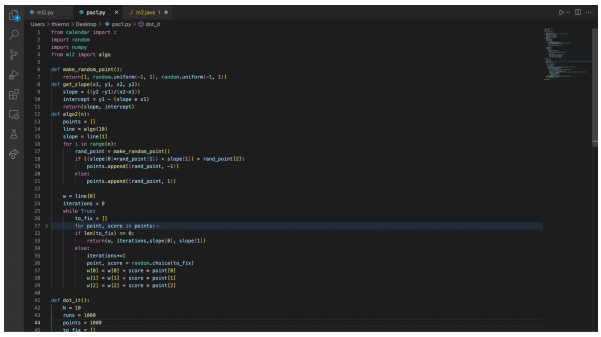
[e] 0.5 - Eout = 0.4515

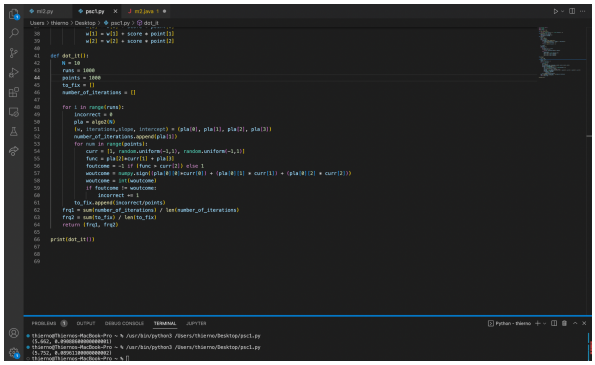
Hence the correct answer according to the above code is **c** as it is closest.

7.









Iteration av = 5.752

[a] 1 - av = −4.752

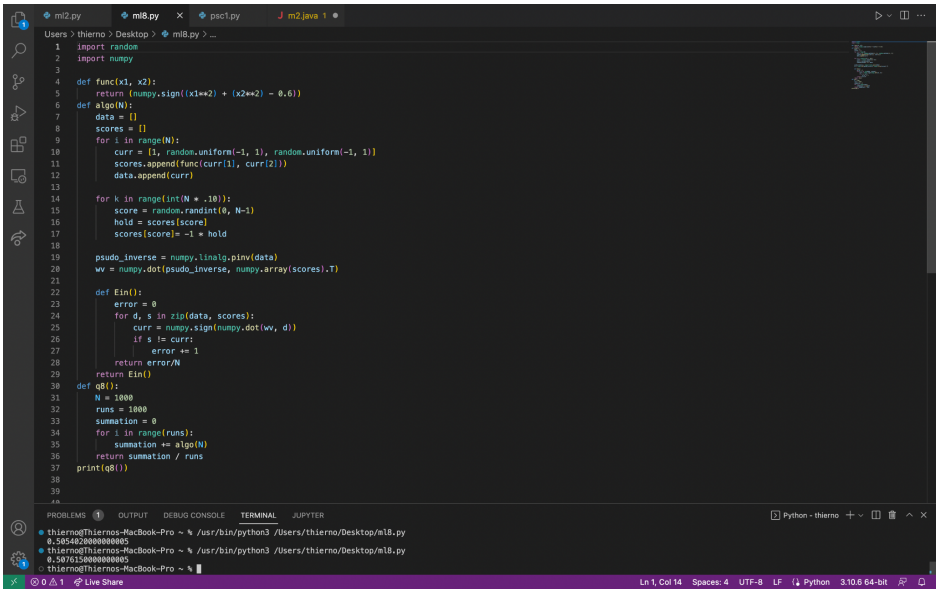
[b] 15 - av = 9.248

[c] 300 - av = 294.248

[d] 5000 - av = 4994.248

[e] 10000 - av = 9994.248

Hence based on the code then **a** is the correct answer because it is closest to zero.

8. 

```
import random
import numpy

def func(x1, x2):
    return (numpy.sign((x1**2) + (x2**2) - 0.6))
def algo(N):
    data = []
    scores = []
    for i in range(N):
        curr = [1, random.uniform(-1, 1), random.uniform(-1, 1)]
        scores.append(func(curr[1], curr[2]))
        data.append(curr)

    for k in range(int(N * .10)):
        score = random.randint(0, N-1)
        hold = scores[score]
        scores[score]= -1 * hold

    psudo_inverse = numpy.linalg.pinv(data)
    wv = numpy.dot(psudo_inverse, numpy.array(scores).T)

    def Ein():
        error = 0
        for d, s in zip(data, scores):
            curr = numpy.sign(numpy.dot(wv, d))
            if s != curr:
                error += 1
        return error/N
    return Ein()
def q8():
    N = 1000
    runs = 1000
    summation = 0
    for i in range(runs):
        summation += algo(N)
    return summation / runs
print(q8())
```

```
thierno@Thiernos-MacBook-Pro ~ % /usr/bin/python3 /Users/thierno/Desktop/ml8.py
0.5054020000000005
thierno@Thiernos-MacBook-Pro ~ % /usr/bin/python3 /Users/thierno/Desktop/ml8.py
0.5076150000000005
thierno@Thiernos-MacBook-Pro ~ %
```

Ein = 0.508

[a] 0 - Ein = -0.508

[b] 0.1 - Ein = -0.408

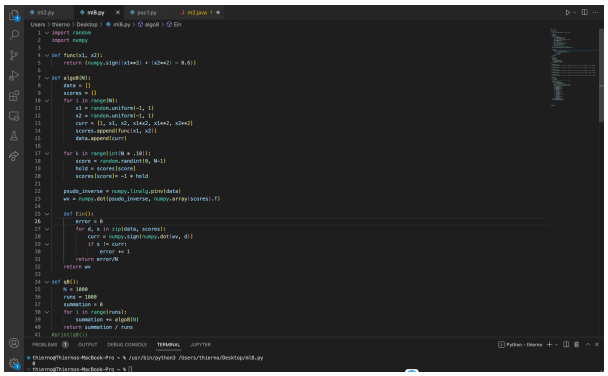
[c] 0.3 - Ein = -0.208

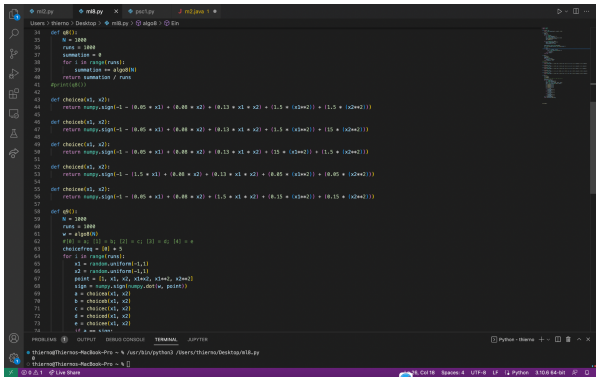
[d] 0.5 - Ein = −0.008

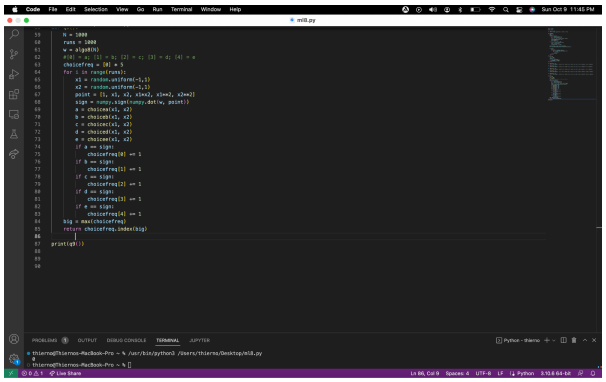
[e] 0.8 - Ein = 0.292

Hence according to the code **d** is the correct answer because it is closest to zero.

9.

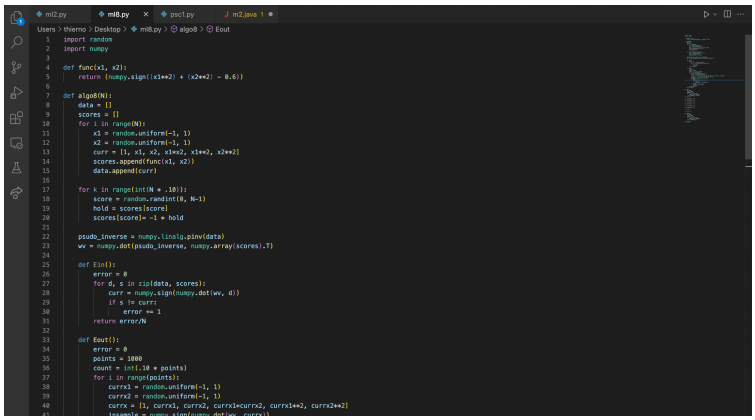


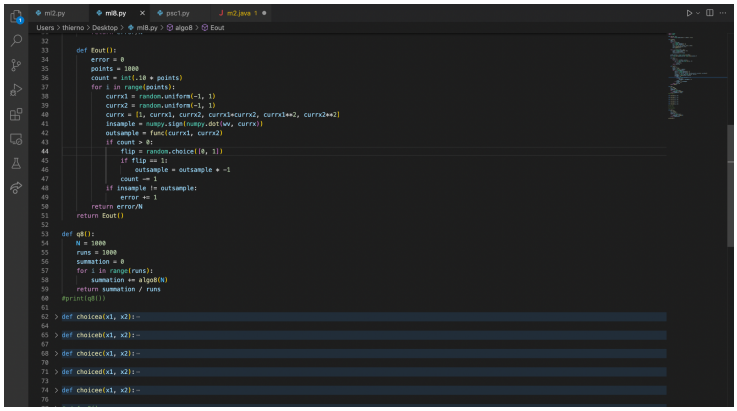


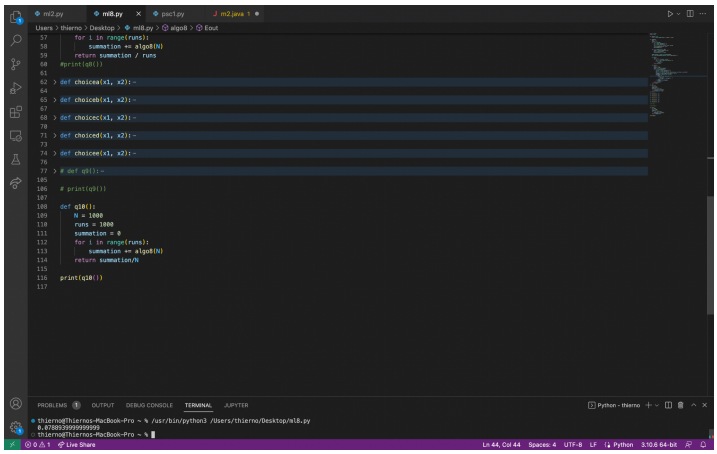


Max frequency is at index 0 according to the code above hence the answer choice is **a** according to frequency sort in the code of which g(x) best matches.

10.







Eout from the above code is 0.0789

[a] 0 - Eout = - 0.0789

[b] 0.1 - Eout = 0.0211

[c] 0.3 - Eout = 0.2211

[d] 0.5 - Eout = 0.4211

[e] 0.8 - Eout = 0.7211

Hence the answer is **b** as it is closest to zero