Lab Assigment 1: Varying Amounts of Noise

In this project, I want you to augment the randomized response query (the one we just wrote) to allow for varying amounts of randomness to be added. Specifically, I want you to bias the coin flip to be higher or lower and then run the same experiment.

Note - this one is a bit tricker than you might expect. You need to both adjust the likelihood of the first coin flip AND the de-skewing at the end (where we create the "augmented_result" variable).

```
import torch
```

Function to create original dataset with true results.

```
def create_db(entries):
  return torch.rand(entries) > 0.5
```

Defining `mean` function which will calculate mean value of given dataset.

```
# create a new query function, which finds the mean rather than the sum
def query_mean(data):
    return data.float().mean()
```

Following command torch.rand(size) > dropout_probability will return number from 0 to 1 with specified probabilty. By adjusting dropout_probability we can set probability of getting 0 (tail in our case).

```
def flip_coin(tail_probability):
    coin = torch.rand(1) > tail_probability
    return coin
```

Defining function to add noise to ariginal dataset. Noise added by calling `flip_coin` function.

```
def creste_noise_list(data_org, tail_prob):
  noise_list = list()
  for j in range (len(data_org)):
    temp = flip_coin(tail_prob)
    if(temp == 1):
      noise_list.append(data_org[j])
    elif(temp == 0):
      noise_list.append(flip_coin(tail_prob))
  noise_list_res = torch.FloatTensor(noise_list)
  return noise_list_res
```

Formula to calculate probability: **P (Yes) = P (Yes | Head) * P (Head) + P (Yes | Tail) * P (Tail)**. In ordre to calculate thre result from our outcome we need to find `P(Yes | Head)`

```
tail_probabilty_value = 0.80
orig_db_result = create_db(10000)
```

```
noise_db_result = creste_noise_list(orig_db_result, tail_probabilty_value)


result_mean_org = query_mean(orig_db_result)


result_mean_noise = query_mean(noise_db_result)
#print(result_mean_noise)

truth = (result_mean_noise - (tail_probabilty_value * (1 - tail_probabilty_value))) / tail_pr
```

Outputs are following:

```
print('Head probability: ', 1-tail_probabilty_value, '. Tail probability: ', tail_probabilty_
print(orig_db_result)
print(noise_db_result)
print('Original dataset: ', result_mean_org)
print('Truth: ', truth)
```

```
Head probability:  0.19999999999999996 . Tail probability:  0.8
tensor([1, 1, 0,  ..., 1, 1, 1], dtype=torch.uint8)
tensor([1., 0., 0.,  ..., 0., 0., 0.])
Original dataset:  tensor(0.4992)
Truth:  tensor(0.1357)
```