Assumptions:

1. The number of balls and bins (N) must be equal and known in advance.

2. The different algorithm polocies have to be feasible for a large number of N.

3. Try to find an algorithm that scale with large N and that can have performance near to the

   deterministic algorithm ( Maximum occupancy 1 ball/bin)

Input paramteters:

1. The starting number of N

        2. The final number of N, in order to measure how the policy choosen scale

Output metrics:

1. The Maximum occupancy in function of N for every policies, that will be compared to the

       theoretical ones.

2. The average occupancy will be 1 ball/bin and the minimal occupancy will be 0 for every

           algorithm that is not deterministic.

Main data structures:

1. The only data structures will be a object that contains a vector of N zeros as intiliazied.

           All the element of the vector will contain from 0 to x balls and every i element will be the

           i+1-th bin.

Main algorithm:

 # First policy is the random dropping

 Initialize the list of metrics

 initialize N\_start and N\_end

 For every N in N\_start and N\_end:

  initialize the object bins with N

For every N:

choose a random bin and add a ball

save the metrics as Maximum Occupancy that refer to this policy and this N in the list

 # For a Random load policy with parameters d

 Initialize the list of metrics

 For every N in N\_start and N\_end:

  initialize the object bins with N

For every N:

choose d random bins

                call the object method --> bins.choose\_min\_busy

                add the ball in the least busy ones

save the metrics as Maximum Occupancy that refer to this policy and this N in the list

Possible extensions:

1. The number of bins and the number of balls are different

        2. Add an additional load balancing policy involving the segmentations of bins: for example if

           you have 100 bins, every time you choose a bin from d random bins you choose it in the first

           25 bins. The second one can be choosen among the 26-th bin and the 50-th and so on so forth until

           starting over from the first segment.

**1. List of all assumptions**

* the average occupancy is always 1 since there N balls and N bins, so it is not necessary to simulate it

**2. List of all the input parameters**

* N, the number of bins and balls

**3. List of all the output metrics**

For each of the dropping policies:

* minimum occupancy
* maximum occupancy

**4. Main data structures**

* class RandomDropping which takes as parameter N and it is able to simulate a random dropping policy with N bins and balls
* class RandomLoadBalancing which takes as parameter N and d and it is able to simulate a random load balancing policy with N bins and balls and d random choices

**5. Main algorithms**

Random dropping policy algorithm:

for each ball:

    randomly choose a bin by generating a number between 0 and N-1

    update the ball count in the chosen bin

compute the minimum and maximum occupancy and return them

Random load balancing algorithm:

for each ball:

    randomly choose d bins by generating d distinct numbers between 0 and N-1

    check the bin with minimum occupancy between the ones selected and update its ball count

compute the minimum and maximum occupancy and return them

**Possible extensions**

* It could be interesting to compare also the execution time of the diffent policies, also with respect to the deterministic algorithm
* Assumption:
* ● Each ball can fall in each bin
* Input parameters:
* ● N number of balls
* ● N numbers of bins
* ● d load balancing
* ● Theoretical formulas
* Output parameters:
* ● average
* ● minimum
* ● maximum occupancy Main data structures:
* ● Dictionary with key = bini values = number of balls inside the bin
* Main algorithm:
* #Initilization of variables
* N = (int)
* creation dictionary -> dict[bin] = 0
* theoretical\_formula = …
* d = [2,4]
* for i in range(N):
* bin = random.randint(n)
* dict[bin] +=1
* calculateAverage(dict)
* minOcc(dict)
* maxOcc(dict)
* for i in range(n):
* bin1 = random.randint(n)
* bin2 = random.randint(n)
* bin = minOcc(bin1,bin2)
* bin += 1
* minOcc(dict) maxOcc(dict)
* for i in range(n):
* bin1 = random.randint(n)
* bin2 = random.randint(n)
* bin3 = random.randint(n)
* bin4 = random.randint(n)
* bin = minOcc(bin1,bin2,bin3,bin4)
* bin += 1
* minOcc(dict)
* maxOcc(dict)
* compareWithTheoreticalFormula()