My Problem Setup:
In this homework I am using lists as my states. Each state has integers in range of 1-8 that each integer represent the row number of the queen and the index represent the column number +1. I am calling random.randint(1,8) to generate random numbers between 1 and 8 inclusive inside a list with length of 8 :

```
Temp = [random.randint(1, 8) for I in range(8)]
```

Example of state: [1,2,3,4,5,6,7,8]

```
o o o o o o o X
o o o o o o X o
o o o o o X o o
o o o o X o o o
o o o X o o o o
o o X o o o o o
o X o o o o o o
X o o o o o o o
```

My objective function is called attacks which counts the number of attacks in each state. I am counting horizontal attacks which is a case if we have duplicates of integers in our list, and diagonal attacks if "index(i) - index(j) = i - j" is true.

I am having two different functions for creating neighbors for each state.
1 - "neighbors_min" : is only used for steepest ascent hill climbing algorithm which would create all 56 neighbors of each state along with their objective function value. I will then find state with minimum attacks and if I have more than one states with minimum attacks, I will randomly select one.
2 - "neighbors_random" : is used in first choice hill climbing and simulated annealing algorithms. Instead of creating all neighbors, this function will only create and return one random neighbor.

My Algorithms:

* Steepest Ascent:
1- Get the initial state as a list from input and set it to current state

2- if current is a solution, return current
2- get the desired neighbor state from “neighbors_min”
3- if value(neighbor) is better than value(current state), then current = neighbor otherwise returns current (which can

* First Choice:
1- Get the initial state as a list from input and set it to current state
2- if current is a solution, return current
3- get a random neighbor
4- if value(neighbor) is better than value(current state), then current = neighbor otherwise choose another neighbor
>> I have implemented different versions of this function that will be explained vastly in analysis section.

* Simulated Annealing:
1- Get the initial state and temperature from input. Set initial state as current state
2- if current is a solution, return current
3- Determine the temperature and randomly select a neighbor
4- deltaE = value(current) - value(neighbor)
5- if deltaE>0 meaning value of neighbor is better than current, current = neighbor
5- or if the probability of exp(deltaE/T) > random(0,1), then current = neighbor
6- if none of the above if statements were true return current

Analysis:

* Steepest Ascent:
10 sample for number of steps: 5,5,3,3,4,5,3,3,5,4
Average number of steps: 4

In general, this hasn’t been a good algorithm to get to a solution. I ran 1000s of random initial states and it was failing more than 90% of the time meaning it gets stuck in local maximum or plateau easily. But choosing the best neighbor based on their objective function makes this algorithm to get to a solution in less than 10 steps.

* First Choice:
For this algorithm, I have used different methods to get out of the loop and return current.

1- Return current if value(neighbor) = value(current) using a list of visited states (equals=True, visited=True, max_num=False)
10 sample for number of steps: 553, 6596, 102, 1264, 145, 2121, 10418, 7308, 2204, 6
Average number of steps: 3071

2- Return current if value(neighbor) = value(current) with no list of visited states (equals=True, visited=False, max_num=False)
10 sample for number of steps: 2, 4, 6577, 384, 313, 6962, 8441, 1662, 20, 4749
Average number of steps: 2911

3- Using a maximum number of iterations with no list of visited states (equals=False, visited=False, max_num=True, max_n=number)

Max number=50	Average number of steps: 29
Max number=100	Average number of steps: 62
Max number=300	Average number of steps: 160

4- Using a maximum number of iterations using a list of visited states (equals=False, visited=True, max_num=True, max_n=number)

Max number=100	Average number of steps: 54

In general, first choice algorithm just like steepest ascent is not an ideal algorithm and its even slightly worse than steepest ascent in terms of average number of steps it usually takes to get to a solution. It takes bigger number of steps than steepest ascent but I think because of using random variables, it will let us to jump out of local maxima or plateaus.

For both Steepest Ascent and First Choice, they were failing most of the time so I had to run thousands of initial states to collect some steps in order to get an average.

* Simulated Annealing:
I have used several temperatures and cooling functions for my Simulated Annealing algorithm and bellow is some details for all I've tried:

1- T -= i , where T is my temperature and i is the number of iterations.
I started with initial temperature set to 200 and then increased it and tested up to 2000 and what I have seen is that the higher the initial temperature, the more chance I will have to get to a result. At temp=200 basically the results weren't significantly better than steepest ascent or first choice and I was getting to a solution less than 30% of the time.
Average number of steps: 12

2- T -= 1 , for each iteration, temperature was decreased by 1.
This method took longer than the last method to run but it was better and with higher temperatures that I ran it with, there was more chance to get to a solution. With initial temperatures set to 500 and higher, I got to a solution almost 50% of the time. Also the number of steps that it took to get to a solution is higher than method 1 but the chance of not getting stuck in local max or plateau was also higher.
Average number of steps: 119

3- T *= alpha , alpha = 0.2, 0.5, 0.8
In this method the smaller alpha worked the best and with really high temperatures like 2000 and higher, there was a great chance to get to a solution more than 50% of the time. With alpha = 0.8 , it was slightly better than method 1 and worse than method 2. It also took bigger steps in order to get to a result
Average number of steps: 516

4- T = T0 + 0.8^i where T0 is my initial temperature and i is the number of iterations.
5- T = T0/(1 + log(1+i))

My last two methods have been great and are having similar symptoms that I thought I should combine them together.
These methods are great in terms of getting to a solution or better say

getting to a global maximum. They all get to a solution 100% of the time but they all take longer than the last methods and take huge number of steps which is good for local search since we don’t care about the path-cost. All these two can be good potential cooling functions for simulated annealing with initial temperatures even as low as 50.
But the tradeoff here compare to the first few methods is that although we always get to a solution but we also have to take big number of steps, in the other hands, the first 3 methods don’t always get to a solution and there is a high chance that they get stuck in local maximum or plateau but they take smaller amount of steps.

Initial temperature = 10Average number of steps: 142505
Initial temperature = 500 Average number of steps: 100071

In conclusion, Simulated Annealing is the best in this list of local search algorithms. If we set our initial temperature high enough and use a good cooling function that decreases slowly and has a very low descending slope, our chances to get to a solution get higher and higher.

On the other hand, I have found Steepest Ascent and First Choice not very practical/optimal algorithms at all since they find a solution less than 10% of the times I attempted to run random initial states. But when they do find a solution, they usually find it in smaller number of steps.