- What should our alg. do?
  - output the solution as the "BestSoFar"
  - keep going until interrupted (by the user pressing ENTER)
  - After being interrupted, return the route that gave the current BSF
  - implement TSP as an anytime alg.

- Input type 1:
  - A file with N rows & two columns (x & y)
  - 1st row is the landing/recharge pad
  - Assume we will never see a value greater than 10,000 meters
  - Units are in meters
  - keep going until user hits ENTER

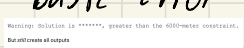
- Input type 2:
  - user hits enter & output is returned w/ route for drone
  - no upper bound on time
  - most interruptions in 5-20 sec.

- Output: (3)
  - D, total dist. of the route, only nearest integer
  - R, route, a list of length N+1, starting & ending w/ '1'
    ex: 1 3 5 2 4 1
    - should be written to the desktop, N+1 rows, 1 column. Name of file should be input file_SOLUTION_D.txt
  - M, an image, showing a visual of the route (Jpeg, PNG, BMP), need high contrast & readable colors, no need for #'s but landing pad must be a diff. color, must be ≥10 pixel buffer between any point an edge, output should be written to the desktop

RandomNN.py

- Compute classic NN, & set as BSF
- While not interrupted, do this modified NN search
  - Find 2 closest nodes, but pick the longer one w/ 1/10 probability
  - If needed, update BSF

What does the alg. need?

- 1st we need to perform classic NN, but how do we do this? What data structures do we need? Definitely need sets. How do we keep track of the nodes we've already visited? prolly another set.
  - set this solution as the BSF, but how? Global variable?
  - then output the distance of this route/solution to terminal
- While not interrupted, keep performing a modified NN search. What is modified? the probability of choosing the closest next node (w/ 9/10 probability) or choosing the 2nd closest next node (w/ 1/10 Prob.)
  - If this leads to a better solution, update BSF & output new dist. to terminal
- If user interrupts alg. (by pressing "ENTER") write solution to disk as "nameofInputfile_SOLUTION_totaldistance.txt"
- Output "Route written to disk as name_of_location so: " to terminal
- How exactly do we form the route tho? number each point/loc. & then push to a queue & the order in which they are popped off is the route?? Maybe.
- How do we calculate the distances? create an N^2 size dist. matrix, ith row jth column is the dist. between locations i & j, only need upper or lower triangle
- How to optimize? Early abandoning. If sum of dist. so far for the current solution is already greater than BSF, stop & start over with diff. node visited 1st
- Need to have basic error handling, if dist. is > 6000 m. when user hits ENTER echo to screen: 

, file doesn't ~~exist~~, file in wrong format, N > 256; give a helpful message & then abort
- Implement some sort of Plateau handling. Ex: If the solution dist. doesn't decrease after 5 subsequent attempts, or for 1.5-2 min. then abort and output the BSF & state that no better solution can be found

What are the parts of the RandomNN main function?

- classic NN alg. ✓
- modified NN alg. ✓
- function to create N^2 size dist. Matrix that returns only 1 triangle ✓
- early abandoning function (our optimizer) ✓
- Plateau function ✓
- basic error handling section (at the top, should be very first lines of code) ✓

What will be passed as parameters to the main function?

- the input file as: input_file = ... ✓
- the starting alg. as: starting_alg = ... ✓
- the modified/second alg. as: second_alg = ... ✓
- func. to create dist. matrix as: calculate_dist = ... ✓
- optimizer as: optimizer = ...

Why are we doing it this way? Allows for extensibility & maintainability. We can modify diff. parts easily in the future if needed. If we find a better way to calculate the distances we can easily modify it. Optimizer can always be changed too & this is a general practice to keep it as a parameter.

What is not passed as a parameter?

- Plateau function as: handle_plateau(...) (not too sure if this is how we'll do it.)
- basic error handling