# PE08: Programming Exercise

## Documentation reference

Mertz, J. (n.d.). Documenting Python Code: A Complete Guide.

## Instructions

- `treasure_hunter_main.py`
- `treasure_hunter.py`

### Description

Given a hunters and treasures locations from the `treasure_hunter_main.py`, complete the functions in the separate `TreasureHunterClass` class within `treasure_hunter.py`.
The main object of the `TreasureHunterClass` is to find the maximum treasures with given hunters.
Note that `treasure_hunter_main.py` with the `main` method and starter file "`treasure_hunter.py`" with class has already been provided.
Keep in mind to always comment and document your class and methods.

A `treasure` + `hunter` map `array` of size `n` is constructed with the following specifications:

1. Each element in the array contains either a `hunter(H)` or a `treasure(T)`.
2. Each hunter can find only one treasure.
3. A hunter cannot catch a treasure that is more than `K` units away from the hunter.

### Expected result

1. `treasure_hunter_main.py`
    - This is the Main python file that is already provided and contains the main and test procedure, which calls methods implemented on "`treasure_hunter.py`"
2. `treasure_hunter.py`
3. This class contains such methods as `init`, `hunt_treasure`.
    1. Please keep in mind the following notes for each method during implementation:
        - `init()`: this method has already been provided.
        - `Hunt_treasure(arr, n, k)`: the method returns the maximum number of treasures that can be found with given hunters on the map.
        - `Parameters`: treasure hunt map `arr`, array length `n`, hunter coverage units `k`

### Output

Describe how the greedy algorithms can be used with an example problem.

### Example Problem

I've Attached a Notebook with an example of the greedy algorithm. Utilizing the traveling salesmen problem and the Minimum Spanning Tree. Both can be used with a greedy algorithm.
Both TSP and MST are NP-hard problems, meaning that there is no known efficient algorithm to solve them optimally for all inputs. Greedy algorithms provide approximate solutions that may not always be optimal, but they are relatively simple and efficient.