Senior Code Python developer to the data science team

Must have

- Minimum 3 years hands-on expertise in core python
- Strong expertise in RDBMS & PSQL
- Sound in data structure and algorithm design.
- Hands-on experience in DB design & Architecture with a focus on performance.
- Good analytical skills and problem-solving skills
- Strong OOPS Concepts
- Debugging and performance analysis.
- In-depth knowledge of Source Code Repository and experience working with Git.
- Strong understanding of the software development life cycle and best practices and experience of working in an Agile development environment and SCRUM
- Strong leadership and communication skills

Good to Have :

- Knowledge in big data, Hadoop, Spark, Ambari and Kafka
- Good understating of Mathematical libraries, Machine learning, deep learning like Mathplotlib, NumPy, Pandas, etc
- Hands-on experience in VMware and server configuration
- Knowledge in docker
- Knowledge in ML frameworks e.g.: Tensorflow
- Minimum 3 years experience in Software development.

Machine Test

What is expected :

- Write a very brief logical implementation or flow chart
- Why you choose to use any modules if any
- Share the Code preferably in GIT with readme
- Brief documentation and readme
- Proof of execution screenshot or video

Task 1

Given "latitude_longitude_details.csv" as a list of latitude and longitudes from Point A to Point B

1. Write a python code to find the latitude and longitude coordinates that are out of line and automatically fix the same to form a continuous path.
2. From the given terrain list with kilometeres, write a python script to generate DB of each latitude and longitude pair with matching terrain information (NB: take the starting latitude and longitude and 0 KM and end as)
3. Write Query to list all the points with terrain “road” in it without “civil station”

Task 2 (Optional)

1. Generate a set of points which are 25 meters to the left and right of the given latitude and longitude, Use multi-threaded/ multi-processing to optimize the execution with the reasoning of the same
2. Design a data structure that can, **efficiently** with respect to time used, store and check if the total of any three successively added elements is equal to a given total.

   For example, *MovingTotal()* creates an empty container with no existing totals. *append([1, 2, 3, 4])* appends elements *[1, 2, 3, 4]*, which means that there are two existing totals (1 + 2 + 3 = 6 and 2 + 3 + 4 = 9). *append([5])* appends element 5 and creates an additional total from *[3, 4, 5]*. There would now be three totals (1 + 2 + 3 = 6, 2 + 3 + 4 = 9, and 3 + 4 + 5 = 12). At this point *contains(6)*, *contains(9)*, and *contains(12)* should return *True*, while *contains(7)* should return *False*.

```
class MovingTotal:
    def append(self, numbers):
        """
        :param numbers: (list) The list of numbers.
        """
        pass
    def contains(self, total):
        """
        :param total: (int) The total to check for.
        :returns: (bool) If MovingTotal contains the total.
        """
        return None

if __name__ == "__main__":
    movingtotal = MovingTotal()

    movingtotal.append([1, 2, 3, 4])
    print(movingtotal.contains(6))
    print(movingtotal.contains(9))
    print(movingtotal.contains(12))
    print(movingtotal.contains(7))

    movingtotal.append([5])
    print(movingtotal.contains(6))
    print(movingtotal.contains(9))
    print(movingtotal.contains(12))
    print(movingtotal.contains(7))
```