Day 11

Santa's Snowy Algorithm

The Story

"LISTEN UP, YOU MAGNIFICENT, STRESSED-OUT CODERS!" Santa bellowed. "Tomorrow, we test the sleigh again. And this time, it WILL work. No more Florida nonsense. Palm trees are NOT part of the Christmas aesthetic."

The Story

He pointed dramatically at Bernard and Pepper. "You two are coming with me. If the sleigh glitches mid-flight again, I want live debugging happening in real time. No excuses".

The Story

The elves whispered nervously. Everyone still remembered the "Florida Incident"—last week's failed test landing in a snowless golf course. Santa didn't appreciate the HOA complaints.

The Story

"This time," Santa continued, pacing, "Snowball compiled every scrap of data we need for each location. Top-notch metrics. But metrics mean nothing without an algorithm. Your job is to write a function to find the snowball-densest landing spot".

Your Mission

Snowball has provided you with a `Vec<Location>`, now the other elves need to write a function to find the most dense area with snow.

Here is what you need to to:

- Write a `new()` associated function for the `Location` struct that takes `x: f64`, `y: f64`, `z: f64`, `area: f64`, and `snow`.

Your Mission

- The `snow` parameter must be able to accept all `SnowKg`, `SnowLb` and `Snowball` types.
- Implement a method for the `Location` struct named `density()` that gets the density of snow in the location.
- Finish the `find_best_location` function which takes a `Vec<Location>` and returns a `Result<Location, Box<dyn Error>>`.

That's it! 🎅

Hints

If you're stuck or need a starting point, here are some hints to help you along the way!

Hints

- When we implemented the `From<T>` trait in the previous challenge, Rust automatically implemented the `Into<T>` trait for us. This means that if you implement `From<T>` for a type, you can convert that type to `T` using the `into()` method. e.g. `let snow = snow.into();`.

Hints

- For the `snow` parameter, you can accept anything that implements the `Into<Snowball>` trait. e.g. `new(x: f64, y: f64, z: f64, area: f64, snow: impl Into<Snowball>) -> Self`. Here `impl Into<Snowball>` means any type that implements the `Into<Snowball>` trait.

Hints

- Convert the `snow` parameter to a `Snowball` type using the `into()` method. e.g. `let snow = snow.into();`.
- Implement the `density()` method for the `Location` struct. The density of snow is calculated by dividing the snow weight by the area of the location. Make sure you handle the division by zero case.

Hints

- Use `into_iter` to convert the `Vec<Location>` to a consuming iterator. e.g. `locations.into_iter()` .

Hints

Use the `max_by()` method to find the location with the highest density of snow. e.g.

```
locations
    .into_iter()
    .max_by(|a, b| {
        a.density()
            .partial_cmp(&b.density())
            .unwrap_or(Ordering::Equal)
    })
```

Hints

- Import `Ordering::Equal` using use `std::cmp::Ordering`.
- Use the `ok_or()` method to convert the `Option` returned by `max_by()` to a `Result`. e.g. `ok_or("No locations found".into())`.