**NonEmptyList:**

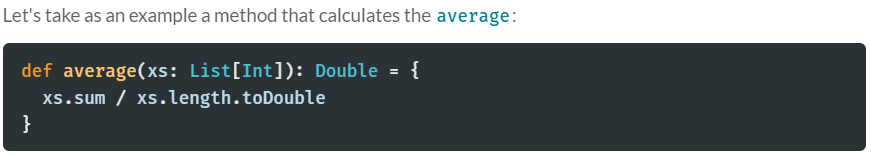
Just a List that can never be empty, so will always contain at least 1 element. Therefore, less error-checking as don’t need to check if NonEmptyList is empty, compared to a normal List.

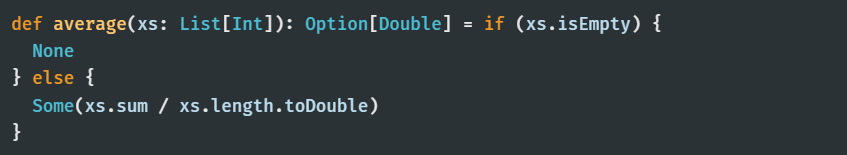
**Usage in Validated and Ior:**

Used in error-reporting structures (e.g. Validated) to store errors, as the NonEmptyList must always contain at least 1 element (therefore 1 error), therefore it is used here as it does not make sense to have an Invalid with no errors: no errors means it is a Valid.

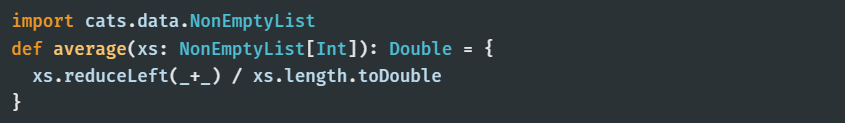
By using NonEmptyList, we explicitly say in the type that: If it is an Invalid, then there is at least one error.

**Avoiding Option by demanding more specific arguments:**

Example method that calculates the average…  


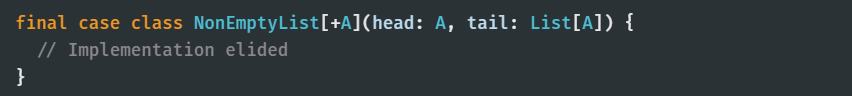
Clearly, this is not a valid definition for empty lists, because division by zero will throw an exception. To fix this, one way is to return an Option instead of a Double right away…

That works and is safe, but this only masks the problem of accepting invalid input. By using Option, we extend the average function with the logic to handle empty lists. Additionally, all callers have to handle the Option cases, maybe over and over again. While better than failing with an exception, this is far from perfect.

Instead, what we would like to express is that average does not make sense at all for an empty list.  
Using Cats NonEmptyList we can assume that the input will never be empty, so we don’t need to deal with the error-handling in the case that the input is empty…  


With that, average is free of any "domain invariant validation" and instead can focus on the actual logic of computing the average of the list. This ties in nicely with the recommendation of shifting your validation to the very borders of your program, where the input enters your system.

**STRUCTURE OF NON-EMPTY-LIST:**



The head of the NonEmptyList will be non-empty. Meanwhile, the tail can have zero or more elements contained in a List.

An important trait of NonEmptyList is totality:  
- For List, both *head* & *tail* operations are partial: they are only well-defined if it has at least one element  
 and can fail if these elements don’t exist in the List.  
  
- NonEmptyList on the other hand, *guarantees* you that operations like *head* and *tail* are defined and wont  
 fail, because constructing an empty NonEmptyList is not possible.