Validation in Scala

1. Validations of members of case class, one validation per member - covered by Daniela in her Scala Exchange 2016 talk.
2. Multiple validations of a single item – explored here.
3. Combine collection of validations into a collection of values if they are all valid or a single combined validation if at least one is invalid – explored here.
4. Combining validations of non-related things - just to know if there is anything wrong – TBD.

2. Multiple validations of a single item.

* BasicValidator performs single validation per case class member.
* AndThenValidator performs 2 validations per case class member
  + It does not accumulate errors, shows first error encountered
* CombineValidator performs 2 validation per class member
  + Accumulates errors correctly
  + Performs undesired value concatenation in successful case
* CombineValidatorImproved
  + Accumulates errors correctly
  + Concatenates values correctly in successful case at the cost of changing the value class as follows:
* **case class** NonCombiningString(value:String) **extends** AnyVal **with** Semigroup[NonCombiningString] {  
   **override def** combine(x: NonCombiningString, y: NonCombiningString): NonCombiningString = x  
  }  
    
  **implicit def** string2NonCombiningString(s:String):NonCombiningString = *NonCombiningString*(s)  
    
  **case class** MyData(email: NonCombiningString, phone: NonCombiningString)

Is there a better way?

3. Convert collection of validations into a collection of values if the are all valid or a single combined failed validation if one or more are invalid.

Implemented in CollectionsValidator, combine seems to be working fine as long as validator returns single element collection of an item rather than an item.

Not ideal because validator needs to be aware of the context in which it will be used, nevertheless, does the job.

4. Combining validations of non-related items – just to know if anything is wrong.

TBD.