Scala - Exercises 3 - Pattern Matching

Warm-up

1. Use a pattern matching function that satisfies the following

```
If number is < 0 return "<0"
    0 to 18 raturn "0<=number<=18"
    19 to 35 return "19<=number<=35"
      36 to 65 return "36<=number<=54"
      Over 65 return ">65"
```

2. Write a function that takes a List and if it's empty returns 0, if it contains three elements, return the third, otherwise return the first
3. Write a function that gives the length of a list, Gives the size of a map, list or vector or gives -1 otherwise
4. Re-write the balanced parenthesis function to use pattern matching along with recursion. The function is:

   Checks whether a string containing only parenthesis, e.g. () characters is balanced. A string is balanced, for example:

   1. () - true
   2. ((())) - true
   3. (())()()(())((())) - true
   4. (() - false
   5.)(- false

   The signature of the function is:

   ```
   def hasValidParenthesis(s:String): Boolean
   ```

5. Using pattern matching, write a function swap that swaps the first two elements of an array provided its length is at least two.

Now we're ready to go

```
sealed trait Expr
case class Number(n:Int) extends Expr
case class Sum(e1: Expr, e2: Expr) extends Expr
```

A sealed trait only allows for it to be extended in the same file as it resides. It also, allows for the compiler to know when there are any cases missing in your match expression.

Above, we have a Number, used to simply represent an integer and a Sum class representing summing wo expressions (Expr). **Note:** Sum is a recursive definition as it can contain any Expr including other Expr

2. Create a method show for the above that outputs a string representation of the expression

```
show(Number(2)) = "2"
show(Sum(Number(2), Number(3))) = "2 + 3"
show(Sum(Sum(Number(2), Number(3)), Number(4)) = "2 + 3 + 4"
```

The definition of the function is:

```
def show(expr:Expr): String
```

3. Add an extra case class to Expr for multiplication (called Prod). In this case, you'll need to ensure that parenthesis go in the correct places where necessary

```
show(Prod(Sum(Number(2), Number(3)), Number(4)) = "(2 + 3) * 4"
show(Sum(Prod(Number(2), Number(3)), Number(4)) = "2 * 3 * 4"
```

4. Finally, add another case for Division (called Div) again adding the appropriate parenthesis where necessary

```
show(Div(Prod(Sum(Number(2), Number(3)), Number(4)), Number(5)) = "((2 +
3) * 4)/5"
```

The sealed trait below again defines an expression, in this case however we have:

- a Variable representing an algerbraic variable in a maths e.g. x or y
- a Number representing a constant number
- a UnaryOperation representing an operation acting on a single Expression, e.g. negation, sqrt
- A BinaryOperation representing an operation acting on two expressions, e.g. +, -, *, /

```
sealed trait Expression
  case class Variable(name:String) extends Expression
  case class Number(number: Double) extends Expression
  case class UnaryOperation(operator: String, argument: Expression)
extends Expression
  case class BinaryOperation(operator: String, left: Expression, right:
Expression) extends Expression
```

5. Write a function that simplifies double negation, addiing 0 and multiplying by 1. The signature of the function is

```
def simplify(expr: Expression): Expression
```

6. Add to your simplify function, a function that converts adding a number to itself to multiplying by 2.