To run the program, enter the directory with the sbt.build file (in this case the `CS476_Homework_1` directory).

`sbt compile` to compile
`sbt run` to run the program
`sbt test` to run the Scalatests

"Frizzy" is the language implementation for some operations in Fuzzy Set Theory. Specifically, it allows for the creation of Fuzzy values and the sets of said Fuzzy values. Some operations that can be performed using "Frizzy" are `Union, Intersection, XOR, AlphaCut,` and `Complement.` The general semantics of the language are described below.

Given an initial setup of 3 sets (which are strings mapped to Fuzzy values):

```
// Create fuzzy sets A, B, and C
val universe: Set[Element] = Set("x1", "x2", "x3", "x4")
val setA = Map("x1" -> 0.2,  "x2" -> 0.8,  "x3" -> 0.5, "x4" -> 1.0)
val setB = Map("x1" -> 0.5,  "x2" -> 0.6,  "x3" -> 0.3, "x4" -> 0.7)
val setC = Map("x1" -> 0.75, "x2" -> 0.75, "x3" -> 0.75, "x4" -> 0.75)
```

You can Assign a variable `"x"` to setA:

```
eval(Assign("x", FuzzySetInstance(setC)))
```

You must use eval([body of expression]) to have anything evaluated.

`Assign` is an operation that binds a variable (the first parameter) to the result of the body. `FuzzySetInstance` is the operation used to create an instance of a Fuzzy set given an actual set (in this case setC, a map).

```
eval(Assign("x", FuzzySetInstance(setA)))
eval(Assign("y", FuzzySetInstance(setB)))
eval(Assign("unionResult", Union(Variable("x"), Variable("y"))))
```

In this case, the variable "unionResult" stores the union of setA and setB.

This operation can be done within a named scope:

```
eval(CreateScope("MyScope",
 Perform(List(
```

```
    Assign("x", FuzzySetInstance(setA)),
    Assign("y", FuzzySetInstance(setB)),
    Assign("unionResult", Union(Variable("x"), Variable("y")))
    // Additional operations can be added in the list
    // the last operation's (end of list) result will be returned
  ))
))
```

Perform is used when one wants to perform a sequence of operations at once. The operation takes in a List of operations. In this case, the list of operations is 2 assignments, followed by a union of the two sets that were assigned.

The value of “unionResult” can be accessed using SummonScope (which returns the value of the last operation in the list passed into its Perform operation).

```
val something = eval(SummonScope("MyScope"))
println(s"Result of the scope: $something")
```

“`something`” will print out the union of the two sets.

You can also create an “anonymous” scope, which cannot be invoked after it’s been executed in the control-flow of the program. `BeginScope` marks the beginning of a set of operations that will be performed within their own scope, and `EndScope` marks the end of the temporary scope.

```
val setOuter = Map("x1" -> 0.5)
val setInner = Map("x1" -> 0.8)

eval(Assign("A", FuzzySetInstance(setOuter)))

eval(BeginScope("InnerScope"))
eval(Assign("A", FuzzySetInstance(setInner)))
val innerResult = eval(Variable("A"))
innerResult shouldEqual setInner
eval(EndScope("InnerScope"))

val outerResult = eval(Variable("A"))
```

`outerResult` should be equal to `setOuter`, because the inner scope has finished executing, and therefore the values in that scope should be obscured from the outer scope.

The language does not implement all of the logic operations found in the Fuzzy set theory. The tests found in the tests folder test variable assignment/access, Union/XOR/Intersect, “anonymous” scoping, creation/invocation of scopes, variable shadowing/obscuring, complement operation, and alpha cut operation.