Exercises

Exercise 1

Create a generic `SortedList` type, where elements are sorted on insertion

Exercise 2

Take back your expression evaluator from last class

Use lifetime annotations and references inside your evaluator instead of boxes. Assess the limitations of this approach.

Don't use this version for further questions!

Exercise 3

Take back your expression evaluator from last class

- Make it dynamically typed:
  - Add a boolean type to the values that can be represented by the evaluator
  - Add true & false literal
  - Add relational operators (and, or)
  - TIP: You need to declare a new enum type for values, and eval needs to return this enum type.
- Add a `let` expression and a `ref` expression:
  - `Let` allows the user to declare a local binding (binding a value to a name)
  - `Ref` allows the user to reference a binding, returning its value
  - TIP: You'll need to use chained hash maps to represent scopes, binding names to values

Exercise 4

- Organise your evaluator into a module
- Provide constructor functions for the user to be able to construct `Expr`s (since the fields won't be visible anymore)

Exercise 5

Use https://docs.rs/lexpr/latest/lexpr/ to make a simple parser for your expression evaluator. make a function that takes a string and returns an expression, like:

```
(let a 12 (+ a (if true 15 18)))
```

Exercise 6

- Reimplement your evaluator using an `Expr` trait, and trait objects, rather than an enum.
- Showcase the extensibility by adding a new expression in another module: `Print` will print the value of an expression and return it

Exercise 7 (bonus)

Add a

```
enum ExprType { Bool, Int }
```

```
fn check_types(&self) -> Result<ExprType, 'static str>
```

function to the expression evaluator, which will return the type of an expression before evaluation, *without* evaluating it.