How are variables created?

```
let name = expr
```

Create a variable with limited, lexical scope.

```
let name = expr1 in expr2
```

The `name` binding is only active in `expr2`.

Apparent re-assignments are actually what?

Instances of shadowing, creating a hole in the previous binding's scope.

What is the anonymous function syntax?

```
fun param1 ... paramN -> expr
```

How can parens be avoided, in general?

By using the `begin` and `end` keywords.

What is the syntax of function application?

As in Haskell, give the function name followed by the arguments, space delimited.

```
funcname arg1 ... argN
```

How are types written?

Which way do they associate?

As in Haskell:

```
param1Type -> ... -> paramNType -> resultType
```

They associate right, so the following two are equivalent:

```
X -> Y -> Z
X -> (Y -> Z)
```

How are functions partially applied?

Simly leave off arguments, at the end of the parameter list. The result will be a new function.

What syntactic sugar is provided for declaring named functions?

```
let name param1 ... paramN = expr
```