What are the three primary functions for working with ref cells?

Give their type signatures?

`ref` for creating a ref cell. `:=` for re-assigning. `!` for extracting the value.

```
val ref  : 'a -> 'a ref
val (:=) : 'a ref -> 'a -> unit
val (!)  : 'a ref -> 'a
```

How can sequentially evaluated expressions be composed into one expression?

Using a semicolon after every expression.

Give the syntax of `for`.

```
for ident := expr1 to expr2 do
  expr3
done

for ident := expr1 downto expr2 do
  expr3
done
```

Give the `while` syntax.

```
while expr1 do
  expr2
done
```

What are the "deeper" problems with functional purity?

- It can be difficult to construct cyclic data structures.
The structure can be created using only values that already exist, so it itself can't be one of them.

- There may be no purely functional algorithm with equivalent performance to the best imperative algorithms for solving some problems.

Function application and ref cells are not ...

... values, i.e., immutable values.

Create a type alias.

```
type newName = existing type
```