`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
```

Using a semicolon after every expression.

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

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

```
while expr1 do
  expr2
done
```

- 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.

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

```
type newName = existing type
```