We started by defining a semigroup class, which declares a binary operation to be associative. From here, we build up through monoids, which introduce identities, to groups, which introduce inverses. Note the double equals in these definitions is notation for an arbitrary equivalence relation over the group's carrier set.

```
Infix "==" := equiv (at level 60, no associativity).
Class Semigroup := {
  semigroup_assoc:
    forall (a b c: Carrier),
      a <o> b <o> c == a <o> (b <o> c);
}.
Class Monoid := {
  monoid_semigroup :> Semigroup equiv op;
  monoid_ident_l:
    forall (a: Carrier), ident <o> a == a;
  monoid_ident_r:
    forall (a: Carrier), a <o> ident == a;
}.
Class Group := {
  group_monoid :> Monoid equiv op ident;
  group_inv_l:
    forall (a: Carrier), inv a <o> a == ident;
  group_inv_r:
    forall (a: Carrier), a <o> inv a == ident;
}.
```

The lines like `monoid_semigroup :> Semigroup equiv op;` simply coerce the monoid typeclass into a semigroup.

While we found later on that we did not need quotients, it is worth remarking that we got quotients working rather nicely in Coq using typeclasses. An algebraic quotient are an equivalence relation on the algebraic structure which preserve that structure. For example, quotient groups are formed by taking a subgroup and making every element of that subgroup equivalent to the identity. With `P` being the predicate for the subgroup, there are two ways to make an equivalence relation from this description.

```
Definition left_congru (a b: Carrier) :=
  P (inv a <o> b).
Definition right_congru (a b: Carrier) :=
  P (a <o> inv b).
```

When these two relations coincide, then we can prove that the equivalence relation(s) actually preserve the group structure. Subgroups which have this property are called *normal subgroups*.

```
Let normal_subgroup_congru_coincide :=
  forall (a b: Carrier),
    left_congru op inv P a b <->
    right_congru op inv P a b.

Theorem quotient_normal_subgroup_group:
  normal_subgroup_congru_coincide ->
  Group (left_congru op inv P) op ident inv.
```

It is because of quotients that we used equivalence relations to define the components of group structure. If we were to use the regular Leibniz equality, this would make it very difficult to say that a quotient group is another group. But by having the definition of a group depend upon an arbitrary equivalence relation, we enable our theory to state that a quotient group is simply a group with under a different equivalence. We didn't loose much as we could still rely upon Coq's setoid rewrite tactics. Setoids are types equipped with an equivalence relation.

Moving onwards, we then defined structures for rings, which have two binary operations: a commutative plus $+$ and a non-commutative times $\cdot$, and structures for commutative rings, where multiplication is commutative and has an identity. Here we define *ideals* which are normal subgroups under addition and are absorbing with regards to multiplication, i.e. ra is in the ideal whenever a is in the ideal and r is any element of the ring. Again, we defined quotient rings but found them unnecessary in the end. Maximal ideals were defined next. These are ideals that are proper subsets of the ring while having no larger ideal except for the ring itself. Below is the definition in Coq, which uses `P` as the predicate for the ideal.

```
Definition maximal_ideal :=
  exists (r: Carrier), (not (P r) /\
    forall (Q: Carrier -> Prop)
        (Q_proper: Proper (equiv ==> iff) Q)
        (Q_ideal: Ideal add zero minus mul Q),
      (forall (r: Carrier), P r -> Q r) ->
      (forall (r: Carrier), Q r) \/
        (forall (r: Carrier), Q r -> P r)).
```

We could then define a local ring, which is a ring with a single maximal ideal.

```
Definition local_ring :=
  exists (P: Carrier -> Prop)
      (P_proper: Proper (equiv ==> iff) P)
      (P_ideal: Ideal add zero minus mul P),
    maximal_ideal P /\
    (forall (Q: Carrier -> Prop)
        (Q_proper: Proper (equiv ==> iff) Q)
        (Q_ideal: Ideal add zero minus mul Q),
      maximal_ideal Q -> forall (r: Carrier), P r <-> Q r).
```

Here we had to include an axiom that in commutative ring, any non-unit x is contained in some maximal ideal. This was made into an axiom as the standard mathematical argument is a proof with potentially infinitely many steps. The standard argument goes as follows.

> Set $I_1 := (x)$ to be the principal ideal for x, i.e. the ideal generated by the single element x. If I_1 is not maximal, then there exists a strictly larger ideal I_2, i.e. $x \in I_1 \subsetneq I_2$. If I_2 is not maximal, then there exists another strictly larger ideal I_3. Continue these arguments infinitely many times if necessary in order to get an infinite chain of ideals containing x.
>
> $$x \in I_1 \subsetneq I_2 \subsetneq I_3 \subsetneq \cdots \subsetneq R$$
>
> It is a simple matter to show that $\bigcup_{k=1}^{\infty} I_k$ is also an ideal containing in x. So by Zorn's lemma, there exists a maximal ideal in R which contains x.

Zorn's lemma is equivalent to the axiom of choice. So we had to add at least one axiom here in order to proceed. By adding the axiom that we did, not only will we encapsulate an infinite argument, but we will also avoid the need for the axiom of choice.

Also, we used classical logic to prove that $1-x$ is a unit where x is a non-unit in any local ring. The proof used the rule $\neg\neg P \to P$ in order to do a proof by contradiction.

The next structure defined were modules over rings which generalize vector spaces over fields. We needed to capture the notion of linear combinations of coefficients and module vectors. This was done by dependently typed vectors, i.e. lists parameterized by their length. Because there is an overload of the term "vector", we will use that term to refer to module vectors, and use the term "list" to mean length parameterized lists. As we don't use the simpler kind of lists, this avoids any name collisions. A finitely generated module is like the vector space $\mathbf{R}^n$ in that there are finitely many vectors which can generate all other vectors, for $\mathbf{R}^n$ one such collection of generators are $\mathbf{e}_1 = \begin{pmatrix}1 & 0 & 0 & \cdots & 0\end{pmatrix}^T$, $\mathbf{e}_2 = \begin{pmatrix}0 & 1 & 0 & \cdots & 0\end{pmatrix}^T, \ldots, \mathbf{e}_n = \begin{pmatrix}0 & 0 & 0 & \cdots & 1\end{pmatrix}^T$. In our code, `M` is the type of module elements, `R` is the type of ring elements which act as coefficients, and `t A n` is a list whose elements are of type `A` and whose length is `n`.

```
Definition finitely_generated {n: nat}(basis: t M n) :=
  forall (vector: M),
    exists (coeffs: t R n),
      vector =M= linear_combin coeffs basis.
```