We start with the module header:

```
module Agda where
open import Relation.Binary.PropositionalEquality
open ≡-Reasoning
open import Data.Function using (_∘_)
```

This is the identity function:

```
id : {A : Set} → A → A
id {A} x = x
```

Natural numbers (Peano style), addition and predecessor:

```
data ℕ : Set where
  zero : ℕ
  suc  : ℕ → ℕ

_+_ : ℕ → ℕ → ℕ
zero  + n = n
suc m + n = suc (m + n)

pred : ℕ → ℕ
pred zero    = zero
pred (suc n) = n
```

Vectors:

```
data Vec (A : Set) : ℕ → Set where
  []  : Vec A zero
  _::_ : forall {n} → A → Vec A n → Vec A (suc n)
```

The empty type ⊥ and its eliminator:

```
data ⊥ : Set where

⊥-elim : {whatever : Set} → ⊥ → whatever
⊥-elim ()
```

Decidable propositions and relations:

```
Rel : Set → Set1
Rel a = a → a → Set

infix 3 ¬_

¬_ : Set → Set
¬ P = P → ⊥

data Dec (P : Set) : Set where
  yes : (p : P) → Dec P
```

```
  no  : (¬p : ¬ P) → Dec P
Decidable : {a : Set} → Rel a → Set
Decidable _∼_ = forall x y → Dec (x ∼ y)
```

Decidable equality on natural numbers:

```
zero≢suc : forall {n} → ¬ zero ≡ suc n
zero≢suc ()

_≟_ : Decidable {ℕ} _≡_
zero  ≟ zero                 = yes ≡-refl
suc m ≟ suc n   with m ≟ n
suc m ≟ suc .m | yes ≡-refl = yes ≡-refl
suc m ≟ suc n  | no prf     = no (prf ∘ ≡-cong pred)
zero  ≟ suc n                = no (⊥-elim ∘ zero≢suc)
suc m ≟ zero                 = no (⊥-elim ∘ zero≢suc ∘ ≡-sym)
```

Associative and commutative operations:

```
Op₂ : Set
Op₂ = ℕ → ℕ → ℕ

Associative : Op₂ → Set
Associative _•_ = forall x y z → ((x • y) • z) ≡ (x • (y • z))

Commutative : Op₂ → Set
Commutative _•_ = forall x y → (x • y) ≡ (y • x)
```

Commutativity of addition:

```
m+1+n≡1+m+n : forall m n → m + suc n ≡ suc (m + n)
m+1+n≡1+m+n zero    n = byDef
m+1+n≡1+m+n (suc m) n = ≡-cong suc (m+1+n≡1+m+n m n)

n+0≡n : forall n → n + zero ≡ n
n+0≡n zero    = byDef
n+0≡n (suc n) = ≡-cong suc (n+0≡n n)

+-comm : Commutative _+_
+-comm zero    n = ≡-sym (n+0≡n n)
+-comm (suc m) n =
  begin
    suc m + n
  ≡⟨ byDef ⟩
    suc (m + n)
  ≡⟨ ≡-cong suc (+-comm m n) ⟩
    suc (n + m)
```

```
≡⟨ ≡-sym (m+1+n≡1+m+n n m) ⟩
  n + suc m
∎
```