```
Inductive day : Type :=
  | monday : day
  | tuesday : day
  | wednesday : day
  | thursday : day
  | friday : day
  | saturday : day
  | sunday : day.
```

```
Definition next_weekday (d:day) : day :=
  match d with
  | monday => tuesday
  | tuesday => wednesday
  | wednesday => thursday
  | thursday => friday
  | friday => monday
  | saturday => monday
  | sunday => monday
  end.
```

```
Example test_next_weekday:
  (next_weekday (next_weekday saturday)) = tuesday.
Proof. simpl. reflexivity. Qed.
```

```
Inductive bool : Type :=
  | true : bool
  | false : bool.
```

```
Definition negb (b:bool) : bool :=
  match b with
  | true => false
  | false => true
  end.
```

```
Definition andb (b1:bool) (b2:bool) : bool :=
  match b1 with
  | true => b2
  | false => false
  end.
```

```
Definition orb (b1:bool) (b2:bool) : bool :=
  match b1 with
  | true => true
  | false => b2
  end.
```

```
Definition admit {T: Type} : T. Admitted.
```

```
Definition nandb (b1:bool) (b2:bool) : bool := negb (andb b1 b2).
```

```
Inductive nat : Type :=
  | O : nat
  | S : nat -> nat.
```

```
Definition pred (n:nat) : nat :=
  match n with
  | O    => O
  | S n' => n'
  end.
```

```
Definition minustwo (n:nat) : nat :=
  match n with
  | O         => O
  | S O       => O
  | S (S n') => n'
  end.
```

```
Fixpoint evenb (n:nat) : bool :=
  match n with
  | O        => true
  | S O      => false
  | S (S n') => evenb n'
  end.
```

```
Definition oddb (n:nat) : bool := negb (evenb n).
```

```
Fixpoint plus (n:nat) (m:nat) : nat :=
  match n with
  | O    => m
  | S n' => S (plus n' m)
  end.
```

```
Fixpoint mult (n m:nat) : nat :=
  match n with
  | O    => O
  | S n' => plus m (mult n' m)
  end.
```

```
Fixpoint minus (n m:nat) : nat :=
  match n, m with
  | O , _      => O
  | S _ , O    => n
  | S n', S m' => minus n' m'
  end.
```

```
Fixpoint exp (base power:nat) : nat :=
  match power with
    | O   => S O
    | S p => mult base (exp base p)
  end.
```

```
Fixpoint factorial (n:nat) : nat :=
  match n with
  | O    => S O
  | S n' => mult n (factorial n')
  end.
```

```
Fixpoint beq_nat (n m:nat) : bool :=
  match n with
  | O    => match m with
             | O    => true
             | S m' => false
            end
  | S n' => match m with
              | O    => false
              | S m' => beq_nat n' m'
             end
  end.
```

```
Fixpoint ble_nat (n m : nat) : bool :=
  (ble_nat (S n) m).
```

```
Theorem plus_0_r :
  forall n:nat, n + 0 = n.
Theorem plus_0_l :
  forall n:nat, 0 + n = n.
```

```
Theorem foo : forall n m:nat,
  n = m ->
  n + n = m + m.
```

- It means it is its own inverse.

```
Theorem negb_involutive : forall b:bool,
  negb (negb b) = b.
```