term *directly* in the goal.

Use associativity and commutativity to prove `add_right_comm`. You don't need induction. `add_assoc` moves brackets around, and `add_comm` moves variables around.

Remember that you can do more targetted rewrites by adding explicit variables as inputs to theorems. For example `rw [add_comm ...`

Goal:

`a + b + c = a + c + b`

```
rw[add_assoc a b c]
```

Retry

Active Goal

Objects:

a b c : ℕ

Goal:

`a + (b + c) = a + c + b`

```
rw[add_comm b c]
```

Retry

Active Goal

Objects:

a b c : ℕ

Goal:

`a + (c + b) = a + c + b`

```
rw[<-add_assoc a c b]
```

Retry

Active Goal

Objects:

a b c : ℕ

Goal:

`a + c + b = a + c + b`

```
rfl
```

Retry

level completed! 🎉

add_assoc

`(a b c : ℕ) : a + b + c = a + (b + c)`

`add_assoc a b c` is a proof that `(a + b) + c = a + (b + c)`. Note that in Lean `(a + b) + c` prints as `a + b + c`, because the notation for addition is defined to be left associative.