Conditionals and Value Definitions
Conditional Expressions

To express choosing between two alternatives, Scala has a conditional expression `if-else`.

It looks like a `if-else` in Java, but is used for expressions, not statements.

Example:

```scala
def abs(x: Int) = if (x >= 0) x else -x
```

`x >= 0` is a *predicate*, of type `Boolean`. 
Boolean Expressions

Boolean expressions \( b \) can be composed of

- `true`  `false`     // Constants
- `!b`     // Negation
- `b && b` // Conjunction
- `b || b` // Disjunction

and of the usual comparison operations:

- `e <= e`, `e >= e`, `e < e`, `e > e`, `e == e`, `e != e`
Here are reduction rules for Boolean expressions (\(e\) is an arbitrary expression):

\[
\begin{align*}
!true & \rightarrow \ false \\
!false & \rightarrow \ true \\
true \ & \& \ e & \rightarrow \ e \\
false \ & \& \ e & \rightarrow \ false \\
true \ & || \ e & \rightarrow \ true \\
false \ & || \ e & \rightarrow \ e
\end{align*}
\]

Note that \&\& and || do not always need their right operand to be evaluated.

We say, these expressions use “short-circuit evaluation”.
Exercise: Formulate rewrite rules for if-else
We have seen that function parameters can be passed by value or be passed by name.

The same distinction applies to definitions.

The `def` form is “by-name”, its right hand side is evaluated on each use.

There is also a `val` form, which is “by-value”. Example:

```
val x = 2
val y = square(x)
```

The right-hand side of a `val` definition is evaluated at the point of the definition itself.

Afterwards, the name refers to the value.

For instance, `y` above refers to 4, not `square(2)`.
Value Definitions and Termination

The difference between `val` and `def` becomes apparent when the right hand side does not terminate. Given

```haskell
def loop : Boolean = loop
```

A definition

```haskell
def x = loop
```

is OK, but a definition

```haskell
val x = loop
```

will lead to an infinite loop.
Write functions `and` and `or` such that for all argument expressions `x` and `y`:

```
and(x, y)   ==   x && y
or(x, y)    ==   x || y
```

(do not use `||` and `&&` in your implementation)

What are good operands to test that the equalities hold?