The AND special form
The AND special form is one of the logical operators. The arguments to this form are booleans. The AND form inputs are evaluated only until one evaluates to false and then the return is false.
Another way to say this is that an AND evaluates to true iff all its inputs are true. No more evaluation occurs after a false expression is found (i.e., short-circuit evaluation).

For the following uses of and, evaluate the result and underline the parts of the expression that are not evaluated:

\[(\text{and} \ (< \ 5 \ 6) \ (> \ 9 \ 8)) \rightarrow (\text{and} \ #t \ #t) \rightarrow #t\]

\[(\text{and} \ (> \ x \ 1) \ (\text{zero?} \ (\text{remainder} \ x \ 2))) \rightarrow\]

if \(x = 2\), evaluates to \((\text{and} \ #t \ #t) \rightarrow #t\)
if \(x = 1\), evaluates to \((\text{and} \ #f) \rightarrow #f\) and the expression with \text{zero?} is not evaluated

The OR special form
The OR special form is one of the logical operators. The arguments to this form are booleans. The OR form inputs are evaluated only until one evaluates to true and then the return is true.
Another way to say this is that an OR evaluates to false iff all its inputs are false. No more evaluation occurs after a true expression is found (i.e., short-circuit evaluation).

For the following uses of or, evaluate the result and underline the parts of the expression that are not evaluated:

\[(\text{or} \ (< \ 5 \ 6) \ (> \ 9 \ 8)) \rightarrow (\text{or} \ #t) \rightarrow #t\]

\[(> \ 9 \ 8) \text{ is not evaluated.}\]

\[(\text{or} \ (> \ x \ 1) \ (\text{zero?} \ (\text{remainder} \ x \ 2))) \rightarrow\]

if \(x = 2\), evaluates to \((\text{or} \ #t) \rightarrow #t\) and the \text{zero?} expression is not evaluated
if \(x = 1\), evaluates to \((\text{or} \ #f \ #f) \rightarrow #f\)

The NOT function
The not function takes 1 boolean input and returns the negation of its input, for example:

\[\text{(not} \ (> = 8 \ 7 \ 6 \ 5 \ 3)) \rightarrow \text{(not} \ #t) \rightarrow #f\]

\[\text{(not} \ (< \ 3 \ 4 \ 5 \ 5 \ 3 \ 7)) \rightarrow \text{(not} \ #f) \rightarrow #t\]
The IF special form

The IF special form is the basic 2-way decision statement. It is written like this:

\[
\text{(if (> x p) (+ p 1) (- x 2))}
\]

There are always three inputs to the if. The first is a predicate expression. The second is the result returned if the predicate evaluates to #t and the third is the result returned in the predicate evaluates to #f. What would the if above return if \( x = 8 \) and \( p = 4 \)?

When explaining the evaluation of an if statement, I often refer to the "true part" (the second input) and the "false part" (the third input). This special form should really only be used for two-way decisions. The if can also have only a true part and no false part.

There is a better way to do multi-way decisions, and that is the COND special form.

The COND special form

The COND special form is the multi-way decision statement. The general form is:

\[
\text{(cond [(< n 10) 5.0] [(> n 90) 'A ] [(< n 20) 5] [(> n 80) 'B ] [(< n 30) true] [else 'C ]})
\]

Each part of the cond inside []'s is called a clause. Each clause can have only 2 parts, a question and an answer. Only 1 cond clause answer is ever returned.

Example COND expressions:

For the first condition that evaluates to true, Scheme evaluates the corresponding answer, and the value of the answer is the value of the entire cond-expression. If the last condition is else and all other conditions fail, the answer for the cond is the value of the last answer expression.

Eg, Write a function that takes one argument, an integer representing a student’s score on an exam. For a score \( \geq 90 \), the return is 'A, for a score \( \geq 80 \), the return is 'B, >=70 is 'C, >= 60 is 'D, else 'F

The order of questions is important. For example, what would be the result if we wrote the cond statements as follows:

\[
\text{(cond [(< n 30) 5.0] [(> n 70) 'A ] [(< n 20) 5] [(> n 80) 'B ] [(< n 10) true] [else 'C ]})
\]