The IF special form (Ch. 11)
The IF special form is the basic 2-way decision statement. It is written like this:

```
(if (> x p) (+ p 1) (- x 2))
```

There are usually 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 if the predicate evaluates to #f**.

What would this if return if  \( x = 8 \) and \( p = 4 \)?

---

The Non-Strict Truth
In a Racket if statement, anything that is not explicitly evaluated to #t is considered to be #t.

So the following if expression would be valid:

```
(if 'we-have-no-bananas "yes" "no") => "yes"
```

```
(if 8 'number 'not-number) => number
```

(note that a quoted symbol is evaluated to a nonquoted sequence of characters—the quote is stripped)

---

The IF special form
The semantics of the IF special form specify that the **predicate expression** is always evaluated to #t or #f. The **true-part** and the **false-part** evaluate to valid Racket expressions of any type. That type is the return value of the if.

```
(if predicate-expression
    true-part
    false-part)
```

---

The IF special form
The semantics of the IF special form specify that the **predicate expression** is always evaluated. After that only one of the **true-part** or the **false-part** is evaluated, never both.

```
(if (> x p) (+ p 1) (- x 2))
```

---

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). While you can do multi-way decisions using a series of nested IF statements, the meaning of the statement may become obscured.

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

---

<table>
<thead>
<tr>
<th>CMPU101</th>
<th>Problem Solving and Abstraction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Special forms IF, WHEN, and COND (Chapter 11 and section 13.1) Special forms AND and OR (Ch. 13)</td>
<td></td>
</tr>
</tbody>
</table>
The IF special form
The syntax of IF may allow it to include a result to return only if the predicate evaluates to #t:
(if (> x p) (+ p 1))

(* x p)

In this case, there are only two inputs to the if. The first is a predicate expression. The second is the result returned if the predicate evaluates to #t. No matter what the evaluated value of predicate is, execution continues on the (* x p) statement after the if.

The WHEN special form (Ch. 11)
The WHEN special form is like the 2-argument IF statement. The difference is that all arguments are evaluated if the cond-expr evaluates to #t.

(when cond-expr arg1 arg2 ... argn)

The when special form may have any number of expressions for the argi's. It will return only the value of the last expression, argn.

The COND special form (Sect. 13.1)
The COND special form is the multi-way decision statement. The general form is:

(cond (predicatea return) ... (predicatemb return) [else return])

The dots indicate that the cond may have an arbitrary number of clauses (including only 2).

The cond special form is the only place you'll see the else keyword used in Racket.

Example COND expressions:

(cond
  (>= n 90) 'A
  (>= n 80) 'B
  (>= n 70) 'C
  else 'F)

The order of clauses is important. For example, what would be returned if we used the cond on the left for input n=0, n=90, or n=25?
What would be returned in the cond on the right for an input of n=90, n=80, or n=40?

(cond
  (<= n 30) 5
  (<= n 70) 'A
  (<= n 80) 'B
  (<= n 100) 1
  else 'C)

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

Eg, Write a function that takes one argument, an integer representing a student's score on an exam. For a score >= 90, the return is 'A, for a score >= 80, the return is 'B, for a score >=70, the return is 'C, for a score >= 60 the return is 'D, else 'F
The AND special form (Sect. 13.2)

The AND special form is one of the logical operators. The arguments to this form are any number of booleans. The AND form inputs are evaluated from left to right 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 (ie, short-circuit evaluation).

The OR special form

The OR special form is one of the logical operators. The arguments to this form are any number of 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 (ie, short-circuit evaluation).

The NOT function

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

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

\[
\text{(not (and #t #f)) } \rightarrow (\text{not #f}) \rightarrow \text{#t}
\]