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 the if above return if \( x = 8 \) and \( p = 4 \)?

The Non-Strict Truth
In a Racket if statement, anything that is not explicitly evaluated to #f 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))
```

The IF special form can contain only one expression (the return value) in each of the true-part and the false-part
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))
```

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. 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
  (cond [(predicate) return] [(predicate) return] ...
        ([predicate) 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 90) 'A ]
  [(>= n 80) 'B] [(>= n 80) 'B]
  [(>= n 70) 'C]) [else 'C ])
```

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

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.

```
(cond
  [(< n 30) 5] [(< n 30) 5 ]
  [(< n 20) 50 ] [(< n 20) 50 ]
  [(< n 10) 1 ]] [else 'C ])
```

The order of questions 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 90) 'A] [(< n 90) 'B]
  [(< n 80) 'A] [(< n 80) 'B]
  [(< n 70) 'C]) [else 'D ])
```

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, If a number is less than 0, the return is 'negative, for a number greater than or equal to 0, the return is 'positive. For a number greater than or equal to 10, the return is 'large, for a number less than 10, the return is 'small, else 'normal.

```
(cond
  [(< n 10) 'negative] [(< n 10) 'positive ]
  [(< n 5) 5 ] [(< n 5) 5 ]
  [(< n 10) 1 ]] [else 'normal ])
```

The order of questions is important. For example, if a number is less than 10, the return is 'normal, for a number less than 5, the return is 'small, for a number less than 10, the return is 'negative. What would be returned in the cond on the right for an input of n=90, n=80, or n=40?
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 (ie, short-circuit evaluation)

The AND special form (Sect. 13.2)

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 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 (ie, short-circuit evaluation)

The OR special form

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

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

\((> 9 \, 8)\) is not evaluated.

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

if \(x \geq 2\), evaluates to \((\text{or } #t) \rightarrow #t\) and the 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:

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

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