1 Introduction to Special Forms

There are many ‘lists’ in Racket which do not follow the standard rules of evaluation for lists. In these situations, there are specific rules that must be followed. The different cases, or special forms in Racket that we will be covering for now in CS17 include the following:

- lambda
- quote
- cond
- if
- and
- or

If the first element of the list is one of the symbols listed above, you know that you are dealing with a special form.

2 Lambda

Lambda expressions are lists which have three arguments. The first is the symbol lambda, the second is called the argument list, and the third is called the body. Lambda expressions evaluate to procedures.
2.1 Quiz

What are the values of the following expressions?

((lambda (x) (+ x 2)) ((lambda (y) (* y 2)) 3))

Answer: 8

((lambda (f) (f 3 2)) +)

Answer: 5

3 Quote

Sometimes, rather than evaluating expressions, we want to use the expressions themselves as our data objects. For instance, when it comes to symbols, sometimes rather than evaluating the symbol, we want to use the symbol itself as the data object, or value. How can we make an expression in Racket whose value is a symbol? The answer is by using quote form. By wrapping a given expression with the quote operator, that expression will not be evaluated and the expression itself will be treated as the data object. These are examples of quote form expressions and what they evaluate to:

- (quote rainy) \Rightarrow \text{the symbol rainy}
- (quote cold) \Rightarrow \text{the symbol cold}
- (quote (rainy cold)) \Rightarrow \text{the list (rainy cold)}
- (quote (+ rainy cold)) \Rightarrow \text{the list (+ rainy cold)}
- (quote (+ 2 3)) \Rightarrow \text{the list (+ 2 3)}

To start out, we will be mostly using the quote operator on symbols.

Something to remember is that just because something looks like an expression, doesn’t mean it will get evaluated. Conversely, just because something doesn’t look like an expression doesn’t mean it won’t get evaluated.
4  Cond

To evaluate a list whose first item is cond,

- Each of the remaining items (2, 3, ...n) must be a two-element list (first element is called the question the second element is called the answer).
- Evaluate the first element of item 2; the value must be a boolean. If it is #true, the value of the cond is the value of the second element.
- Evaluate the first element of item 3; the value must be a boolean. If it is #true, the value of the cond is the value of the second element.
- ..... 

Key point:  After a #true value is found in a question, the later question and answer expressions are not evaluated.

If none of the question expressions evaluates to true, BLARGH! (there will be an error)

Often, the last question is just #true to catch any remaining cases.

5  And and Or

These are Boolean operations: the inputs are Boolean and the output is Boolean. The form of and and or expressions is as follows:

(or  expr-1 expr-2 ... expr-n)
(and  expr-1 expr-2 ... expr-n)

The following is the ‘truth table’ for and:

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>and</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
</tr>
<tr>
<td>FALSE</td>
<td>TRUE</td>
<td>FALSE</td>
</tr>
<tr>
<td>TRUE</td>
<td>FALSE</td>
<td>FALSE</td>
</tr>
<tr>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
</tr>
</tbody>
</table>

The following is the ‘truth table’ for or:

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>and</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
</tr>
<tr>
<td>FALSE</td>
<td>TRUE</td>
<td>TRUE</td>
</tr>
<tr>
<td>TRUE</td>
<td>FALSE</td>
<td>TRUE</td>
</tr>
<tr>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
</tr>
</tbody>
</table>
Please let us know if you find any mistakes, inconsistencies, or confusing language in this or any other CS 17 document by filling out the anonymous feedback form: [http://cs.brown.edu/courses/csci0170/feedback](http://cs.brown.edu/courses/csci0170/feedback)