Default Evaluation Rule
The default evaluation rule for non-empty lists is used every time a non-quoted, non-empty list expression is evaluated. By this rule, for an expression such as 

\[(d_1 \ d_2 \ d_3 \ \ldots \ \ d_n)\]

\(each\) of the expressions \(d_1\ldots d_n\) are evaluated. \(d_1\) must evaluate to a function, and the results of evaluating \(d_2\ldots d_n\) are arguments to function \(d_1\).

The define special form
Programs in Racket are combinations of function and value definitions (constants).

The \emph{define} special form is the mechanism by which you can add entries to the global environment (GE). Define can be used to add names for functions and values (constants) to the GE.

Evaluation of a DEFINE expression
When a left parenthesis occurs before the keyword \emph{define} there are always exactly 2 arguments that follow the keyword. Neither the keyword \emph{define} nor the first argument is evaluated. Only the second argument is evaluated to simplest form.

<table>
<thead>
<tr>
<th>(define</th>
<th>TURTLE-WT</th>
<th>(* 50 BEAK-LENGTH) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>sp form</td>
<td>constant name</td>
<td>value</td>
</tr>
<tr>
<td>(not eval)</td>
<td>1st argument (not evaluated)</td>
<td>2nd argument (evaluated)</td>
</tr>
</tbody>
</table>

Like all other programming languages, Racket has what are called KEYWORDS. These are character sequences that are a part of the language syntax. Keywords in Racket are called \textsc{special forms} because the evaluation of each one is slightly different, and none of them are the same as the default evaluation rule.

Using a special form looks like using the default evaluation rule but special forms are each evaluated in their own way.

The special form \texttt{define} is used as shown below to name constants in the GE:

\[(\text{define INNINGS 9})\]
\[(\text{define STRIKE-LIMIT 3})\]
\[(\text{define STARTERS 10})\]
\[(\text{define BEAK-LENGTH 8})\]
The define expression is processed as follows:
1. `define` is read but not evaluated,
2. the 1st argument is a name for a function or value and is not evaluated but is written in the name column of the GE, and
3. the 2nd argument is evaluated and written in the value column of the GE.

`(define STARTERS (+ (* 5 2) 3))` would write the name STARTERS and the value 13 in the GE

**Evaluation vs Look-up**
Evaluation of an expression involves using the default rule to reduce the expression to simplest form.

Evaluation of a name that is defined in the GE involves looking up the name in the table. All values written in the GE are already evaluated to simplest form. In the cases we've seen, they are numbers.

When defining a name for a number or other value, only the second argument is evaluated to simplest form.

`(define NUM-RINGS 100)`
`(define TREE-AGE (* NUM-RINGS 10))` The first statement writes the name NUM-RINGS in the GE. The second statement looks up the value of NUM-RINGS and then evaluates the expression (* NUM-RINGS 10), setting the value 1000 for TREE-AGE in the GE.

**Defining Functions**
There are 2 ways we will define functions. The first is a simplified version to avoid complexity for beginners.

The second form gives the actual value of the function as it is stored in the GE.

**Defining Functions - Method 1**
A function is written to the GE when the `define` keyword is followed immediately by an unquoted open-parenthesis containing >= 1 name. The name to the right of define is the name of the function and the other names are parameter names. This parenthesis is closed and followed by the body of the function in its own set of parentheses.

Ex: `define (room-area side1 side2 height)`

`(define (room-area side1 side2 height)`

**Defining Functions – Method 1**
Ex: red indicates function name,
blue indicates parameter names
green indicates body of function and return value

`(define (horse-info shoulder-ht ear-ht shoe-ht)`

`(define (horse-info shoulder-ht ear-ht shoe-ht)`
Defining Functions – Method 2

The lambda special form is the way Racket programs actually store a function. For example the definition of the function room-area is really evaluated to the following before it is executed:

```
(define room-area
  (lambda (side1 side2 height)
    (* side1 side2 height)))
```

In either of the cases for function definition, the define special form returns void.

Calling Functions

Calling either version of the function horse-info requires 3 numeric arguments:

```
(horse-info 63 72 2)
```

you will get a numeric return value, in this case 133.

Define + Lambda

The define special form returns void, but has the important side-effect of writing a name and the associated lambda (function) in the GE.

Calling a function can produce an output, a side-effect alone, or both an output and a side-effect.

Side-Effects of Functions

Side effects include, e.g.:

- Writing a name in the Global Environment.
- Printing to the interactions window.
- Changing the value of a constant or parameter defined in the Global Environment (later...).

Naming Functions

In summary, the define special form is used to write a name associated with a lambda special form in the GE:

```
(define cube1
  (lambda (z) (* z z z)))
```

has exactly the same meaning as

```
(define (cube2 z) (* z z z))
```

The first expression writes the name cube1 and the second writes the name cube2 in the GE as lambda expressions.
Calling a Defined Function

To use the cube function after it has been written to the GE, you type a left parenthesis, followed by the name of the function, followed by values for the argument(s):

(cube 50) ;;; -> 125000

Calling this function has the output of 125000 but no side-effects.

Commenting Code

All characters typed to the right of a semicolon (;) in DrRacket are called comments.

Comments are usually colored brown and no automatic indentation takes place inside a comment.

Comments inside functions

When writing a function, it is a good idea to include comments, either between lines of code or to the right of the code.

Comments are not evaluated. They are meant to make the purpose of the code clear to anyone reading it. Comments are an essential part of every program.

Length of Lines in Code

When you are writing a program, it is essential to write the code and the comments such that no typed characters extend beyond 80 columns.

Lines that extend beyond 80 columns tend to "wrap around" to the next line when they are printed.

DrRacket allows you to set a line in the code for the maximum line width you will allow.

(Preferences → Editing → General Editing)

Function Contracts

Function contracts will be displayed in this class to the interactions window. The contract explains the type of inputs, type of output (purpose of function), and any side effects.

(display "
  Contract: (add3 n1 n2 n3) --> number
  Input: n1, n2, and n3 are numbers.
  Output: n1 + n2 + n3, a number
")