Special Form Evaluation

Programs in Racket are combinations of function and constant definitions and the use of special forms. The `define` special form is the mechanism by which you can add names to the global environment (GE).

Each special form has its own, special evaluation rule (its own semantics). For `define`, the syntax is:

```
(define name expression)
```

The `define` special form

( define S E ) → [ list with define S E ⇒ void ] → void

1. it writes name s in the GE with the value void;
2. it evaluates E to its primitive type e;
3. it inserts e as the value of s; and
4. it returns nothing (void).

The only part of a `define` special form that is evaluated is the second argument. There is no output other than void. The side-effect is to insert a new entry into the GE.

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 with a value of void, and
3. the 2nd argument is evaluated and written into the GE as the value of name.

( define STARTERS (+ (* 5 2) 3))

would write the name STARTERS and the value 13 in the GE. Only primitive types are written as values (no expressions).

The define special form

The keyword `define` is used as shown below to name constants in the GE:

```
(define INNINGS 9)
(define STRIKE-LIMIT 3)
(define STARTERS 10)
```

By many programmers' convention, constant names are written in all capital letters. Function names do not follow this convention.

The define special form

define is always followed by exactly 2 arguments. The first is a name that is not evaluated, but is written in the GE, and the second is an expression that evaluates to a primitive data type.

```
(define TAX-RATE 0.25)
(define INCOME 94000)
(define FED-TAX (* TAX-RATE INCOME))
```

Previous definitions can be used in ones that come later (lower).
Defining Global Constants
You will define far more names for functions than you do for constant values, but one universal rule for programmers is:

*Keep numerals and other values out of your code.*

When you have a number (other than 0 or 1) that is used inside a function definition, give it a name in the GE and use the named constant inside your code. This is particularly true if the number is used more than once in the code.

---

Evaluation vs Look-up
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 in simplest primitive form. In the cases we've seen, they are numbers.

---

Syntax for Defining Functions
There are 2 ways to define functions (they have the same result):

1. a simplified version of define without lambda to avoid complexity.
2. a define statement that uses the LAMBDA special form.

The lambda special form is a function.

---

Function Definition Form 1
1. (define followed by (  
2. The name to be written in the GE  
3. A sequence of parameters, and )  
4. An expression that evaluates to a primitive type and )

(define (horse-info shoulder-ht ear-ht shoe-size)  
  (- (+ shoulder-ht ear-ht) shoe-size))

---

Function Definition Form 2
1. (define  
2. The name to be written in the GE.  
3. (lambda  
4. A sequence of parameters in ()  
5. An expression that evaluates to a primitive type and )

(define horse-info  
  (lambda (shoulder-ht ear-ht shoe-size)  
    (- (+ shoulder-ht ear-ht) shoe-size)))
Calling Functions

We can apply the function defined on the last 2 slides using the default evaluation or application rule:

\[(\text{horse-info } 63 72 2)\]

When the default evaluation rule is applied, horse-info is looked up in the GE, and the numbers (input arguments) are matched sequentially to the parameter names.

\(\text{define horse-info} \)

\(\lambda \text{ (shoulder-ht ear-ht shoe-size)} (\cdot (\text{+ shoulder-ht ear-ht) shoe-size})))\)

Calling a Defined Function

To use the cube1 function after it has been written to the GE, you use the default evaluation rule: type a left parenthesis, followed by the name of the function, followed by appropriate values for the argument(s):

\((\text{cube1 } 50) ;; \rightarrow 125000\)

Calling the cube1 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.

Every function should be preceded by a contract, parameter names and types, and the purpose of the function, all written in comments.
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 Headers

One type of comment is essential for all programs: Function headers. These comments give the function name, usage, types of inputs, type of output, and any side effects.

;;; Name:   cube1
;;; Usage:  (cube1  n1) -> number
;;; Input:  n1 is a number.
;;; Output: n1 * n1 * n1  (n1 cubed)

Evaluation of a DEFINE expression

When a left parenthesis occurs before the keyword define there are always exactly 2 arguments that follow the keyword. Neither the keyword define nor the first argument is evaluated. Only the second argument is evaluated to simplest form.

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 - Method 1

A function is written to the GE when the define keyword is followed immediately by an uncommented 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)
     (* 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)
  (- (+ shoulder-ht ear-ht) shoe-ht))
```

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 produces an output, but define has only a side-effect and void output.*

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...).