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 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 define special form

Programs in Racket are combinations of constant value and function definitions.

The define special form is the mechanism by which we can add entries to the global environment (GE). Define can be used to add names for constant values and functions.

 Constants are necessary to use in a program because it is much harder to maintain code that has "magic numbers" repeated throughout.

To maintain code is to keep it running when the problem description or the values of the "magic numbers" change. It is much easier to change the value in a single define statement than to change it throughout the code.

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. Eg,

\( (\text{define TURTLE-WT } (* \text{BEAK-LENGTH})) \)

This statement writes the name TURTLE-WT in the GE and assumes the value of BEAK-LENGTH is already in the GE so the 2nd argument can be evaluated to simplest form.
**Evaluation and Evaluation**

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

Evaluation of an expression involves using the default rule to reduce the expression to simplest form.

Again, only the second argument is evaluated to simplest form (in this example, it is an expression).

```
(define TREE-AGE (* NUM-RINGS 50))
```

This statement writes the name TREE-AGE in the GE and assumes the value of NUM-RINGS is already defined. If the value of NUM-RINGS is 10, the value of tree-age is 500.

Important to remember: Only the last argument of a define special form is evaluated. The second argument is written to the GE, a side-effect.

---

**Writing function names & values to GE (defining functions)**

A function is written to the GE when the `define` keyword is followed immediately by an open-parenthesis containing ≥ 2 names. The first name is the name of the new function, the other names are parameter names. This parenthesis is closed and followed by the body of the function, usually in its own set of parentheses.

Ex: `(define (horse-hands ht wd hoof-ht) (- (* ht wd) hoof-ht))`

---

**Function Basics**

A function can have any number of inputs, but can only have one output. However, the function can also create side-effects.

What is the output when this define statement is entered and evaluated in DrRacket?

Nothing! The define special form returns void.

However, if you try calling the function horse-hands with 3 numeric arguments, i.e.

(horse-hands 23 4 2)

you will get a numeric return value. The define special form returns void, but has the important side-effect of writing a name and the associated function in the GE.
Side-Effects of Functions

Side effects include:
• Writing a name in the Global Environment.
• Printing to the interactions window.
• Changing the value of a constant or parameter.
• ...

Taking away the "syntactic sugar"

The lambda special form is the way Racket programs actually store a value for a function name. For example the definition of the function horse-hands is really evaluated like this:

```
(define horse-hands
  (lambda (ht wd hoof-h) (- (* ht wd) hoof-h)))
```

The lambda special form is really the only way to define a function. The special form has 2 arguments: the first is a list of parameters and the second is the body of the function (usually another list).

```
(l lambda (x) (* x x))
```

Since lambda literally means "function", we can use an unnamed lambda as follows:

```
((lambda (x) (* x x)) 5)
```

Here, 5 is the input value of the unnamed lambda expression.

Naming Functions

The define special form is used to write a name and the lambda special form to the GE:

```
(define cube
  (lambda (z) (* z z z)))
```

has exactly the same meaning as

```
(define (cube z) (* z z z))
```

Both these expressions write the name cube in the GE and store a lambda expression as the value.

Calling Defined Functions

To use the cube function after it has been written to the GE, you would use the default evaluation rule:

```
(cube 50) ➔ 125000
```

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

Note about semicolons:

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

Using comments in your code is essential and every function should be preceded by a contract, parameter names and types, and the purpose of the function, all written in comments.

There are other ways to comment code, but using ; is the standard method.
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. They are essential.