Rules of computation in Racket

Look at the expression

- If it's a **constant** (i.e. a **number** or **string**), it's its own value
- If it's a **variable name** (i.e. a word, hyphenated phrase, etc.), look its value up in the dictionary
- Parentheses? (Check if it's one of the **special cases**)
- Otherwise it's a **function call**
  - (*func-expression* $arg\text{-}expression_1$ … $arg\text{-}expression_n$)
  - Execute all the subexpressions (*func-expression* and $arg\text{-}expression_1$ through $arg\text{-}expression_n$)
  - Call the value of *func-expression*, passing it the values of $arg\text{-}expression_1$ through $arg\text{-}expression_n$ as inputs
  - Use its output as the value of the expression

Special Cases: Rules of computation in Racket

If it starts with **define***

`(define` *name value-expression*`)`

- Run *value-expression*
- Assign its value to *name* in the dictionary

If it starts with **λ** or **lambda***:

`(λ (`$name_1$ … $name_{last}$`)` *result-expression*`)`

- Make a function
  - That names its inputs $name_1$ … $name_{last}$
  - And returns the value of *result-expression* (presumably using those names)

*__Sussman Form__ is a combination of these two forms

If it starts with the the word **local**

`(local [(define` $name_1$ $value_1$`)`
`…`
`(define` $name_{last}$ $value_{last}$`)]`
result-expression`)`

- Run the *value* expressions
- Substitutes their values for their respective *names* in *result-expression*
- Runs the substituted *result-expression*
- Returns its value

If it starts with **if**:

`(if test consequent alternative)`

- Run `test`
- If it returns **true**, run `consequent` and return its value
- Otherwise run `alternative` and return its value

If it starts with **define-struct**

`(define-struct typename (properties…))`

- Creates a new type of data called `typename`
- Defines a constructor: `make-typename`
- Defines accessors: `typename-property-1`…
- Defines predicate: `typename?`

If it starts with **cond**:

`(cond [`$test_1$ $result_1$`]`
`[ … ]`
`[else` $result_{last}$`])`

- Runs tests in order, first one to pass (`#true`) return corresponding `result`

If it starts with **require**:

`(require some-library)`

- Loads function definitions for later use from `some-library`

If it starts with **check-expect**:

`(check-expect thing-1 thing-2)`

- Compares two `things` and "passes" if the two things are equivalent.