Symbols

Lisp was invented to do symbolic processing.

A key Racket value is the symbol.

The symbol cat is written (quote cat) or 'cat.

Symbols are values and so evaluate to themselves.

> 'cat
'cat

; 'thing is just an abbreviation for (quote thing)
> (quote cat)
'cat

Symbols similar to strings, except they’re atomic; we don’t do character manipulations on them.

S-Expressions

Lisp pioneered symbolic expressions, a.k.a. s-expressions, a parenthesized notation for representing trees as nested lists (compare to other tree notations, like XML or JSON).

Example:
'((this is (a nested)) list (that (represents a) tree))

Atoms

The leaves of an s-expression are atomic (indivisible) and so are called atoms. In Racket, atoms include numbers, booleans, and strings in addition to symbols.

Example: '((251 #f) ("foo bar" baz))
Quotation with Atoms and Lists

A quoted atom \((\text{quote } \text{atom})\) (abbreviated \(\text{'atom}\)) denotes the atom. For atoms that are not symbols, \((\text{quote } \text{atom})\) desugars to \(\text{atom}\). For example:

- \((\text{quote } 251)\) desugars to 251
- \((\text{quote } \#t)\) desugars to \#t
- \((\text{quote } "\text{Hi there!}\)\) desugars to "Hi there!"

A quoted parenthesized structure \((\text{quote } (...)\) (abbreviated \(\text{'(...)\)}) denotes a list, according to the following desugaring:

\(((\text{quote } \text{sexp}_1 \ldots \text{sexp}_n)\) desugars to \((\text{list } (\text{quote } \text{sexp}_1) \ldots (\text{quote } \text{sexp}_n))\)

Example: What is the desugaring of the following:

\('(17 \text{ foo } \#f) "\text{bar}\) (\text{list } + (\text{quote } \text{quux}))\)

Functions on s-expression trees

Write the following functions that take an s-expression tree as their only arg:

1. \((\text{sexp-num-atoms } \text{sexp})\) returns the number of atoms (leaves) in the s-expression tree \(\text{sexp}\).

   \[\begin{align*}
   > \text{(sexp-num-atoms tr)} \\
   11
   \end{align*}\]

2. \((\text{sexp-atoms } \text{sexp})\) returns a list of the atoms (leaves) encountered in a left-to-right depth first search of the s-expression tree \(\text{sexp}\).

   \[\begin{align*}
   > \text{(sexp-atoms tr)} \\
   '(\text{a b c d e f g h i j k})
   \end{align*}\]

3. \((\text{sexp-height } \text{sexp})\) returns the height of the s-expression tree \(\text{sexp}\).

   \[\begin{align*}
   > \text{(sexp-height tr)} \\
   4
   \end{align*}\]

A sample s-expression

We will do some exercises with this sample s-expression:

\[(\text{define tr } '((\text{a b c d}) \text{ e } (((\text{f g h}) \text{ i j k}))))\]

Draw the tree associated with this s-expression.

An s-expression Read-Eval-Print Loop (REPL)

\[(\text{define (sexp-repl)}\)
\[(\begin{align*}
\text{(begin (display "Please enter an s-expression:"))} \\
\text{(let \{[(\text{sexp (read))}] \text{; read prompts user for sexp}} \\
\text{\text{if } (\text{eq? sexp 'quit)} \text{'done}} \\
\text{\text{\begin{align*}
\text{(begin (display (list 'sexp-num-atoms:} \\
\text{(sexp-num-atoms sexp)))}} \\
\text{(newline)} \\
\text{(display (list 'sexp-atoms:} \\
\text{(sexp-atoms sexp)))}} \\
\text{(newline)} \\
\text{(display (list 'sexp-height:} \\
\text{(sexp-height sexp)))}} \\
\text{(newline)} \\
\text{(sexp-repl))))))))
\end{align*}\)
\)]
On to Metaprogramming

A *metaprogram* is a program that manipulates another program, such as an interpreter, compiler, type checker, assembler, etc.

In a metaprogram, how could we represent a Racket definition like this?

```
(define avg (lambda (a b) (/ (+ a b) 2)))
```