RACKET

Data Structures: Lists

Genesis of Racket

- 1960: LISP. The second programming language, after FORTRAN
- 1975: Scheme.
- 1995: Racket.

Basic Operations in Racket

- Racket is an interpreted language (think of a genie: you ask questions, it answers them.)
- Racket expressions are formed using pre-order notation.
- Examples:

```
(+ 3 4)        -> 7
(- 7 2)        -> 5
```

Fundamental Data Structure in Racket

- In Racket, data (and programs) are represented as lists.
- There are no data types in Racket.
- Examples:
  - `'(one two 3 42)`
  - `'(+ 3 4)`
- Lists containing data are prepended by a quote: '
- No quote, no data.

Operations on Lists

- Accessors:
  - **`car`**: obtains the head of the list.
  - **`cdr`**: obtains the list after the head of the list
- Constructor:
  - **`cons`**: places the first argument at the head of the list provided by the second argument

car

- Syntax: *(car <arg>)*
- *car* stands for Contents of Address Register
- Examples:

```
(car '(1 2 3))              -> 1
(car '(a b c))              -> a
(car '((x y) z (d)))        -> (x y)
(car '(a))                  -> a
(car '())                   -> An error
```

cdr

- Syntax: *(cdr <arg>)*
- *cdr* stands for Contents of Displacement Register
- Examples:

```
(cdr '(1 2 3))            -> (2 3)
(cdr '(a b c))            -> (b c)
(cdr '((x y) z (d)))      -> (z (d))
(cdr '(a))                -> '()
(cdr '())                 -> An error
```

cons

- Syntax: *(cons <arg1> <arg2>)*
- *cons* stands for Construct
- Examples:

```
(cons 1 '(2 3))           -> (1 2 3)
(cons 'a '(b c))          -> (a b c)
(cons '(x y) '(z (d)))    -> ((x y) z (d))
(cons 'a '())             -> (a)
(cons '() '())            -> (())
```

Proper Lists

- So what is a list anyway?
- Consider: ‘(a b c)
- Recall from CSSE 220: A list is a bunch of nodes linked together:

- Each node contains an element and a pointer to the next element.
- The last node contains a null pointer, in Racket an empty list.

Proper Lists

- *cons* is a constructor for lists
- The list: 

  would be constructed like so:

```
(cons ‘a (cons ‘b (cons ‘c ‘())))
```

Proper Lists

You may use "list?" to verify whether something is a list:

(list? 3)
(list? 'a)
(list? '(a b))
(list? '(a . b))

Improper Lists: Pairs

- What if we were to do:

```
(cons 'a 'b)   -> (a . b)
```

- We call this an *improper list*
- It is improper because it ends in an element rather than an empty list.
- Improper lists are called "*dotted pairs*."
- The list '(a b c) could be written as:

  (a . (b . (c . ())))

Improper Lists: Pairs

You may use “pair?” to verify whether something is a list:

(pair? 3)
(pair? ‘a)
(pair? ‘(a b))
(pair? ’(a . b))

Easy Lists

- To make a proper list quickly:

```
(list ‘a ‘b ‘c ‘x 1)   -> (a b c x 1)
```

Numbers

- Numbers are just that.
- You can test whether something is a number with the "number?" predicate:
  - (number? 4)
  - (number? 'a)
  - (number? '(a b))

Symbols

- Symbols are single, quoted items.
- If a symbol were to be unquoted, it would be an identifier.
- You can test whether something is a symbol with the "symbol?" predicate:
  - (symbol? 4)
  - (symbol? 'a)
  - (symbol? '(a b))

Class Exercise

- Let’s write some simple, recursive procedures for lists:
  - length
  - member