Programming Paradigms

CT331 Week 6 Lecture 2

Finlay Smith

Finlay.smith@universityofgalway.ie

Lisp - Lists

A list is an ordered group of data

Lists are displayed between parentheses using the quote character.

List elements are separated by a space.

The list syntax is shorthand for an s-expression

List of numbers:

```
'(1 2 3)
```

List of strings:

```
'("this" "that" "the other")
```

List of mixed data types:

```
'(1 2 "three" 4)
```

Lisp - Car and Cdr

Lisp uses nested lists (basically linked lists):

Access the first element of a list using the `car` function.

Access the rest of the list using the `cdr` function.

`cdr` is just like `element->rest` in our C linked list.

```
> (car `(1 2 3))
1

> (cdr `(1 2 3))
`(2 3)

> (car (cdr `(1 2 3)))
2
```

Lisp - C*r

There exists a shorthand for a combination of cars and cdrs (up to 4 operations usually but depends on Scheme environment), where * is a or d or a combination (if supported).

Example:

write sequence of cars and cdrs to extract:

- d from list (a b c d e f)
- a from list ((a b) 3 (c d))
- b from list ((a b) 3 (c d))
- d from list ((a b) 3 (c d))

Lisp - Lists are Cons Pairs

Lists are really just cons pairs where the second element is another list or `empty`.

`empty` is a special word - similar to NULL.

```
> (cons 2 empty)
'(2)
```

```
> (cons 1 (cons 2 empty))
'(1 2)
```

Lisp - Lists are Cons Pairs

What do the following return:

- `(car (cons ’x ’(y z a b)))`
- `(cdr (cons ’a ’(x y z w)))`

Lisp - Lists are Cons Pairs

What do the following return:

- `(car (cons ’x ’(y z a b)))`
- `(cdr (cons ’a ’(x y z w)))`

i.e. the first argument to cons is the car of the resultant list and the second argument is the cdr of the resultant list.

Note: the built-in functions `list` and `append` provide a more convenient way to create lists

Lisp - Lists are Cons Pairs

What do the following return:

- `(cons (cdr ’(a b c)) (cdr ’(b c d)))`
- `(cons (car ’(a b c)) (cdr ’(b c d)))`
- `(cons ’(car ’(a b)) ’(and orange))`

Lisp - Define

Define binds a variable to some data.

Format:

```
(define variable value)
```

Lisp - Define

Used for user-defined functions

Format:

```
(define (function_name parameter-list)
      Function-body
)
```

Note: User defined fns. can be used within other user defined fns. as long as the functions are defined before they are invoked.

Lisp - Define

```
(define (sumabs num1 num2)
        (+ (abs num1) (abs num2))
)
```

Calculates the absolute addition of two numbers where the function abs returns the absolute value of a number.

```
> (sumabs 2 -3)
5
```

Note: No return statement.

Lisp - Define

What’s wrong with this?

```
(define sumabs (num1 num2)
    (+ (abs num1) (abs num2))
)
```

Lisp - Define

Define a function `secondel` which returns the second element of a list such that:

- `(secondel ’(a b c d)) returns b`
- `(secondel ’(a (b c d ) e)) returns (b c d)`

Define a function thirdel which returns the third element of a list

Define a function fourthel which returns the fourth element of a list

Lisp - List function

Constructs a list from components

format:

```
(list el-1 el-2 el-n)
```

These components can be symbols, numbers or lists

Lisp - List function

- `>(list ’a ’b ’c ’d ’e ’f)`
- `>(list ’(1) ’(a b c))`
- `>(list ’a (car ’(b c d)))`
- `>(list (cdr ’(x y z)) (cdr ’(b c d)) ’f)`

Lisp - List function

- `>(list ’a ’b ’c ’d ’e ’f)`

  `(a b c d e f)`
- `>(list ’(1) ’(a b c))`
- `>(list ’a (car ’(b c d)))`
- `>(list (cdr ’(x y z)) (cdr ’(b c d)) ’f)`

Lisp - List function

-
  ```
  >(list 'a 'b 'c 'd 'e 'f)
  (a b c d e f)
  ```

-
  ```
  >(list '(1) '(a b c))
  ((1) (a b c))
  ```

-
  ```
  >(list 'a (car '(b c d)))
  ```

-
  ```
  >(list (cdr '(x y z)) (cdr '(b c d)) 'f)
  ```

Lisp - List function

- `>(list ’a ’b ’c ’d ’e ’f)`

  `(a b c d e f)`
- `>(list ’(1) ’(a b c))`

  `((1) (a b c))`
- `>(list ’a (car ’(b c d)))`

  `(a b)`
- `>(list (cdr ’(x y z)) (cdr ’(b c d)) ’f)`

Lisp - List function

-
  ```
  >(list 'a 'b 'c 'd 'e 'f)
  (a b c d e f)
  ```

-
  ```
  >(list '(1) '(a b c))
  ((1) (a b c))
  ```

-
  ```
  >(list 'a (car '(b c d)))
  (a b)
  ```

-
  ```
  >(list (cdr '(x y z)) (cdr '(b c d)) 'f)
  ( (y z) (c d) f)
  ```

Cons vs list

What’s the different between:

```
(cons `(1) `(a))
```

And

```
(list `(1) `(a))
```

Lisp - append

Collects components from several lists into one list

format:

```
(append list1 list2 ... listn)
```

Note: arguments must be lists

Lisp - append

- `(append ’(mr) ’(john) ’(jones))`
- `(append ’((3 2)) ’( ) ’((( 1 2 3))) )`
- `(append 4 ’(3) )`
- `(append ’(3 2) ’(1 2 3))`

Lisp - append

- `(append ’(mr) ’(john) ’(jones))`

  `(mr john jones)`
- `(append ’((3 2)) ’( ) ’((( 1 2 3))) )`
- `(append 4 ’(3) )`
- `(append ’(3 2) ’(1 2 3))`

Lisp - append

- `(append ’(mr) ’(john) ’(jones))`

  `(mr john jones)`

- `(append ’((3 2)) ’( ) ’((( 1 2 3))) )`

  `((3 2) (( 1 2 3)) )`

- `(append 4 ’(3) )`

- `(append ’(3 2) ’(1 2 3))`

Lisp - append

- `(append ’(mr) ’(john) ’(jones))`

  `(mr john jones)`
- `(append ’((3 2)) ’( ) ’((( 1 2 3))) )`

  `((3 2) (( 1 2 3)) )`
- `(append 4 ’(3) )`

  `error`
- `(append ’(3 2) ’(1 2 3))`

Lisp - append

- (append ’(mr) ’(john) ’(jones))

  (mr john jones)
- (append ’((3 2)) ’( ) ’(((1 2 3))))

  ((3 2) ((1 2 3)))
- (append 4 ’(3))

  error
- (append ’(3 2) ’(1 2 3))

  (3 2 1 2 3)