

# The Wizard's Adventure Game <sup>1</sup>

Eric Bailey

October 14, 2017 <sup>2</sup>

In this game, you are a wizard's apprentice.  
You'll explore the wizard's house.

## Contents

<i>Setting the Scene</i>	2
<i>Describing the Location</i>	2
<i>Describing the Paths</i>	3
<i>Describing Multiple Paths at Once</i>	4
<i>Describing Objects at a Specific Location</i>	4
<i>Describing Visible Objects</i>	5
<i>Describing It All</i>	5
<i>Walking Around in Our World</i>	6
<i>Picking Up Objects</i>	6
<i>Checking Our Inventory</i>	7
<i>Tests</i>	7
<i>lol.wizard5 (Private Parts)</i>	8
<i>lol.wizard5 (Public API)</i>	8
<i>Running the Tests</i>	9
<i>Full Listing</i>	10
<i>Chunks</i>	12
<i>Index</i>	12

<sup>1</sup>

Conrad Barski. *Land of Lisp: Learn to Program in Lisp, One Game at a Time!*, chapter 5, pages 67–84. No Starch Press, 2010. ISBN 9781593273491. URL <http://landoflisp.com>

<sup>2</sup> Last updated October 17, 2017

```
1  <* 1>≡
    (in-package :cl-user)
    (defpackage lol.wizard5
      (:use :cl)
      (:export :look
               :walk
               :pickup
               :inventory))
    (in-package :lol.wizard5)
```

*<define the global variables 2d>*

This definition is continued in  
chunks 2–7.

Root chunk (not used in this document).

Defines:

*lol.wizard5*, used in chunks 7 and 8.

Uses inventory 7b, look 5f, pickup 6l,  
and walk 6g.

## Setting the Scene

This world consists of only three locations:

- 2a 1.  $\langle \text{The living room } 2a \rangle \equiv$   
     you are in the living room.  
     a wizard is snoring loudly on the couch.  
     This code is used in chunks 2d and 8.
- 2b 2.  $\langle \text{A beautiful garden } 2b \rangle \equiv$   
     you are in a beautiful garden.  
     there is a well in front of you.  
     This code is used in chunks 2d and 8b.
- 2c 3.  $\langle \text{The attic } 2c \rangle \equiv$   
     you are in the attic.  
     there is a giant welding torch in the corner.  
     This code is used in chunk 2d.

*\*nodes\** is simply an *association list* with locations as keys and the previous descriptions as values.

2d  $\langle \text{define the global variables } 2d \rangle \equiv$   
     (defparameter \*nodes\*  
       '((living-room ( $\langle \text{The living room } 2a \rangle$ ))  
         (garden ( $\langle \text{A beautiful garden } 2b \rangle$ ))  
         (attic ( $\langle \text{The attic } 2c \rangle$ ))))

This definition is continued in chunks 3–5.

This code is used in chunk 1.

Defines:

\*nodes\*, used in chunks 5f and 8a.

## Describing the Location

To find the description,  $\langle \text{look up a location } 2e \rangle$  and take the *cadr*. Preferring the *functional programming* style, pass *nodes* as an argument, instead of referencing *\*nodes\** directly.

2e  $\langle \text{look up a location } 2e \rangle \equiv$   
     (assoc location nodes)

This code is used in chunk 2f.

2f  $\langle *1 \rangle + \equiv$   
     (defun describe-location (location nodes)  
       (cadr ( $\langle \text{look up a location } 2e \rangle$ )))

Defines:

describe-location, used in chunks 5f and 8a.

## Describing the Paths

3a `<garden door 3a>≡`  
`THERE IS A DOOR GOING WEST FROM HERE.`  
 This code is used in chunks 7c and 8a.

3b `<living-room paths 3b>≡`  
`(garden west door)`  
 This definition is continued in chunk 3d.  
 This code is used in chunk 3g.

3c `<attic ladder 3c>≡`  
`THERE IS A LADDER GOING UPSTAIRS FROM HERE.`  
 This code is used in chunk 7c.

3d `<living-room paths 3b>+≡`  
`(attic upstairs ladder)`  
 This code is used in chunk 3g.

3e `<garden path 3e>≡`  
`(living-room east door)`  
 This code is used in chunk 3g.

3f `<attic path 3f>≡`  
`(living-room downstairs ladder)`  
 This code is used in chunk 3g.

To describe such a symbolic path, take the means (`caddr`) and direction (`cadr`) and return a descriptive list.

3h `<* 1>+≡`  
`(defun describe-path (edge)`  
 `'(there is a ,(caddr edge) going ,(cadr edge) from here.))`

Defines:  
`describe-path`, used in chunk 8a.

From the `living-room`, you can move to the `garden` by going `west` through the `door`.

Or to the `attic` by going `upstairs` via the `ladder`.

From the `garden`, you can move to the `living-room` by going `east` through the `door`.

From the `attic`, you can move to the `living-room` by going `downstairs` via the `ladder`.

3g `<define the global variables 2d>+≡`  
`(defparameter *edges*`  
 `'((living-room <living-room paths 3b>)`  
 `(garden <garden path 3e>)`  
 `(attic <attic path 3f>)))`

This code is used in chunk 1.

Defines:  
`*edges*`, used in chunks 5f, 6a,  
 and 8a.

## Describing Multiple Paths at Once

To describe multiple paths:

4a 1. *⟨Find the relevant edges. 4a⟩*≡  
       (cdr (assoc location edges))

This code is used in chunk 4d.

4b 2. *⟨Convert the edges to descriptions. 4b⟩*≡  
       mapcar #'describe-path

This code is used in chunk 4d.

4c 3. *⟨Join the descriptions. 4c⟩*≡  
       apply #'append

This code is used in chunks 4d and 5d.

4d *⟨\* 1⟩*+≡  
       (defun describe-paths (location edges)  
         (*⟨Join the descriptions. 4c⟩* (*⟨Convert the edges to descriptions. 4b⟩* (*⟨Find the relevant edges. 4a⟩*))))

Defines:

describe-paths, used in chunks 5f and 8a.

## Describing Objects at a Specific Location

4e *⟨define the global variables 2d⟩*+≡  
       (defparameter \*objects\* '(whiskey bucket frog chain))

      (defparameter \*object-locations\*  
         '((whiskey living-room)  
           (bucket living-room)  
           (chain garden)  
           (frog garden)))

This code is used in chunk 1.

Defines:

\*object-locations\*, used in chunks 5–8.

\*objects\*, used in chunks 5–8.

4g *⟨\* 1⟩*+≡  
       (defun objects-at (loc objs obj-locs)  
         (labels (*⟨at-loc-p 4f⟩*)  
           (remove-if-not #'at-loc-p objs)))

4f *⟨at-loc-p 4f⟩*≡  
       (at-loc-p (obj)  
                 (eq (cadr (assoc obj obj-locs)) loc))

This code is used in chunk 4g.

Defines:

objects-at, used in chunks 5–8.

## Describing Visible Objects

To describe the objects visible at a given location:

- 5a 1. *Find the objects at the current location. 5a* ≡  
 (objects-at loc objs obj-loc)

This code is used in chunk 5d.

Uses objects-at 4g.

- 5b 2. *Convert the objects to descriptions. 5b* ≡  
 mapcar #'describe-obj

This code is used in chunk 5d.

3. *Join the descriptions. 4c*

- 5c *describe-obj 5c* ≡  
 (describe-obj (obj)  
 '(you see a ,obj on the floor.))

This code is used in chunk 5d.

- 5d *\* 1* + ≡  
 (defun describe-objects (loc objs obj-loc)  
 (labels ((describe-obj 5c)  
 ((Join the descriptions. 4c)  
 ((Convert the objects to descriptions. 5b)  
 (Find the objects at the current location. 5a))))))

Defines:

describe-objects, used in chunks 5f and 8.

## Describing It All

- 5f *\* 1* + ≡  
 (defun look ()  
 (append (describe-location \*location\* \*nodes\*)  
 (describe-paths \*location\* \*edges\*)  
 (describe-objects \*location\* \*objects\* \*object-locations\*)))

N.B. The **look** function is **not** functional,  
 since it reads global variables.

- 5e *define the global variables 2d* + ≡  
 (defparameter \*location\* 'living-room)

Defines:

look, used in chunks 1, 6f, and 8b.

Uses \*edges\* 3g, \*location\* 5e, \*nodes\* 2d, \*object-locations\* 4e, \*objects\* 4e,  
 describe-location 2f, describe-objects 5d, and describe-paths 4d.

This code is used in chunk 1.

Defines:

\*location\*, used in chunks 5 and 6.

## Walking Around in Our World

Given a **direction**, *<locate the path marked with the appropriate direction 6c>* and *<try to go in that direction 6f>*. Since the **direction** will be there, *<match against the cadr of each path 6b>*.

```
6c <locate the path marked with the appropriate direction 6c>≡
    (find direction
      <look up the available walkings paths 6a>
      <match against the cadr of each path 6b>)
```

This code is used in chunk 6g.

If such a path is found, *<adjust the player's position 6d>*, otherwise *<admonish the player 6e>*.

```
6f <try to go in that direction 6f>≡
    (if next
      (progn <adjust the player's position 6d>
              (look))
      <admonish the player 6e>)
```

This code is used in chunk 6g.  
Uses look 5f.

```
6g (* 1)+≡
    (defun walk (direction)
      (let ((next <locate the path marked with the appropriate direction 6c>))
        <try to go in that direction 6f>))
```

Defines:

walk, used in chunks 1 and 8b.

## Picking Up Objects

To determine if *<the object is on the floor 6h>*,

```
6i <get the list of objects here 6i>≡
    (objects-at *location* *objects* *object-locations*)
```

This code is used in chunk 6h.  
Uses \*location\* 5e, \*object-locations\* 4e, \*objects\* 4e, and objects-at 4g.

... and check if **object** is a **member**. If so...

```
6j <pick it up 6j>≡
    (push (list object 'body) *object-locations*)
    '(you are now carrying the ,object)
```

This code is used in chunk 6l.  
Uses \*object-locations\* 4e.

Otherwise...

```
6k <you cannot get that. 6k>≡
    '(you cannot get that.)
```

This code is used in chunks 6l and 8b.

```
6a <look up the available walkings paths 6a>≡
    (cdr (assoc *location* *edges*))
```

This code is used in chunk 6c.  
Uses \*edges\* 3g and \*location\* 5e.

```
6b <match against the cadr of each path 6b>≡
    :key #'cadr
```

This code is used in chunk 6c.

```
6d <adjust the player's position 6d>≡
    (setf *location* (car next))
```

This code is used in chunk 6f.  
Uses \*location\* 5e.

```
6e <admonish the player 6e>≡
    '(you cannot go that way.)
```

This code is used in chunks 6f and 8b.

```
6h <the object is on the floor 6h>≡
    (member object <get the list of objects here 6i>)
```

This code is used in chunk 6l.

```
6l (* 1)+≡
    (defun pickup (object)
      (if <the object is on the floor 6h>
        (progn <pick it up 6j>)
        <you cannot get that. 6k>))
```

Defines:

pickup, used in chunks 1 and 8b.

## Checking Our Inventory

To check our inventory, we *<retrieve the list of carried objects 7a>* and prepend (a.k.a. `cons`) the symbol `items-`.

**7a** *<retrieve the list of carried objects 7a>*≡  
`(objects-at 'body *objects* *object-locations*)`  
 This code is used in chunk **7b**.  
 Uses `*object-locations*` **4e**, `*objects*` **4e**, and `objects-at` **4g**.

**7b** *<\*1>*+≡  
`(defun inventory ()`  
 `(cons 'items- <retrieve the list of carried objects 7a>))`  
 Defines:  
`inventory`, used in chunks **1** and **8b**.

## Tests

**7i** *<test/wizard5.lisp 7i>*≡  
`(in-package :lol.wizard5)`  
  
`(prove:plan 2)`  
  
*<Test the private functions in lol.wizard5 8a>*  
  
*<Test the exported functions in lol.wizard5. 8b>*  
  
`(prove:finalize)`  
 Root chunk (not used in this document).  
 Uses `lol.wizard5 1`.

**7c** *<living-room path descriptions 7c>*≡  
`<garden door 3a>`  
`<attic ladder 3c>`  
 This code is used in chunk **8**.

**7d** *<living-room object descriptions 7d>*≡  
`YOU SEE A WHISKEY ON THE FLOOR.`  
`YOU SEE A BUCKET ON THE FLOOR.`  
 This code is used in chunk **8**.

**7e** *<garden path description 7e>*≡  
`THERE IS A DOOR GOING EAST FROM HERE.`  
 This code is used in chunk **8b**.

**7f** *<garden object descriptions 7f>*≡  
`YOU SEE A FROG ON THE FLOOR.`  
`YOU SEE A CHAIN ON THE FLOOR.`  
 This code is used in chunk **8b**.

**7g** *<You've got whiskey! 7g>*≡  
`'(YOU ARE NOW CARRYING THE WHISKEY)`  
 This code is used in chunk **8b**.

**7h** *<All you have is whiskey. 7h>*≡  
`'(ITEMS- WHISKEY)`  
 This code is used in chunk **8b**.

**lol.wizard5** (*Private Parts*)

```

8a <Test the private functions in lol.wizard5 8a>≡
    (prove:subtest "lol.wizard5 (Private Parts)"
      (prove:is (describe-location 'living-room *nodes*)
        '(<The living room 2a>))
      (prove:is (describe-path '(garden west door))
        '(<garden door 3a>))
      (prove:is (describe-paths 'living-room *edges*)
        '(<living-room path descriptions 7c>))
      (prove:is (describe-objects 'living-room *objects* *object-locations*)
        '(<living-room object descriptions 7d>))
      (prove:is (objects-at 'living-room *objects* *object-locations*)
        '(WHISKEY BUCKET)))

```

This code is used in chunk 7i.

Uses \*edges\* 3g, \*nodes\* 2d, \*object-locations\* 4e, \*objects\* 4e, describe-location 2f, describe-objects 5d, describe-path 3h, describe-paths 4d, lol.wizard5 1, and objects-at 4g.

**lol.wizard5** (*Public API*)

```

8b <Test the exported functions in lol.wizard5. 8b>≡
    (prove:subtest "lol.wizard5 (Public API)"
      (prove:is (look)
        '(<The living room 2a>
          <living-room path descriptions 7c>
          <living-room object descriptions 7d>)))
      (prove:subtest "Pick up the whiskey"
        (prove:is (pickup 'whiskey)
          '(<You've got whiskey! 7g>))
        (prove:is (objects-at 'living-room *objects* *object-locations*)
          '(BUCKET))
        (prove:is (describe-objects 'living-room *objects* *object-locations*)
          '(YOU SEE A BUCKET ON THE FLOOR.)))
      (prove:is (pickup 'the-pace)
        '(<you cannot get that. 6k>))
      (prove:is (walk 'west)
        '(<A beautiful garden 2b>
          <garden path description 7e>
          <garden object descriptions 7f>)))
      (prove:is (walk 'south)
        '(<admonish the player 6e>))
      (prove:is (inventory)
        '(<All you have is whiskey. 7h>)))

```

This code is used in chunk 7i.

Uses \*object-locations\* 4e, \*objects\* 4e, describe-objects 5d, inventory 7b, lol.wizard5 1, look 5f, objects-at 4g, pickup 6l, and walk 6g.



## Running the Tests

## Describe prove

9c `<Run the system tests. 9c>≡`  
`(prove:run-test-system :lol-test)`

This code is used in chunk 9d.

9d `<Run the system tests and exit. 9d>≡`  
`(uiop:quit (if <Run the system tests. 9c> 0 1))`

This code is used in chunk 9j.

## Describe nix-shell shebang

9e `<script header 9e>≡`  
`#!/usr/bin/env nix-shell`  
`#! nix-shell -i sh -p sbcl`

This definition is continued in chunk 9.

This code is used in chunk 9j.

9j `<bin/runtests 9j>≡`  
`<script header 9e>`  
`-eval "<Load the test package. 9h>" \`  
`-eval "<Run the system tests and exit. 9d>"`

`<script footer 9i>`

Root chunk (not used in this document).

`$ ./bin/runtests`

`✓ 2 tests completed (0ms)`

Summary:

All 1 file passed.

9k `<init.lisp 9k>≡`  
`#-quicklisp`  
`(let ((quicklisp-init (merge-pathnames "quicklisp/setup.lisp"`  
`(user-homedir-pathname))))`  
`(when (probe-file quicklisp-init)`  
`(load quicklisp-init)))`

`(push (concatenate 'string (sb-posix:getcwd) "/")`  
`asdf:*central-registry*)`

Root chunk (not used in this document).

9a `<Set the exit status. 9a>≡`  
`(if (null failures) 0 1)`

Root chunk (not used in this document).

9b `<Exit with an appropriate status code. 9b>≡`  
`(sb-posix:exit status)`

Root chunk (not used in this document).

Run sbcl quietly:

`<script header 9e>+≡`  
`sbcl -noinform -non-interactive \`

This code is used in chunk 9j.

Load `<init.lisp 9k>` as the user initialization file:

9g `<script header 9e>+≡`  
`-userinit init.lisp \`

This code is used in chunk 9j.

9h `<Load the test package. 9h>≡`  
`(asdf:load-system :lol-test)`

This code is used in chunk 9j.

9i `<script footer 9i>≡`  
`# Local Variables:`  
`# mode: sh`  
`# End:`

This code is used in chunk 9j.

*Full Listing*

```

11 (defparameter *nodes*
12   '((living-room (you are in the living room.
13                 a wizard is snoring loudly on the couch.))
14     (garden      (you are in a beautiful garden.
15                 there is a well in front of you.))
16     (attic       (you are in the attic.
17                 there is a giant welding torch in the corner.))))
18
19 (defparameter *edges*
20   '((living-room (garden west door)
21                 (attic upstairs ladder))
22     (garden      (living-room east door))
23     (attic       (living-room downstairs ladder))))
24
25 (defparameter *objects* '(whiskey bucket frog chain))
26
27 (defparameter *object-locations*
28   '((whiskey living-room)
29     (bucket living-room)
30     (chain garden)
31     (frog garden)))
32
33 (defparameter *location* 'living-room)
34
35
36 (defun describe-location (location nodes)
37   (cadr (assoc location nodes)))
38
39
40 (defun describe-path (edge)
41   `(there is a ,(caddr edge) going ,(caddr edge) from here.))
42
43
44 (defun describe-paths (location edges)
45   (apply #'append (mapcar #'describe-path (cdr (assoc location edges)))))
46
47
48 (defun objects-at (loc objs obj-locs)
49   (labels ((at-loc-p (obj)
50             (eq (cadr (assoc obj obj-locs)) loc)))
51     (remove-if-not #'at-loc-p objs)))

```

```

54 (defun describe-objects (loc objs obj-loc)
55   (labels ((describe-obj (obj)
56             `(you see a ,obj on the floor.)))
57     (apply #'append
58            (mapcar #'describe-obj
59                     (objects-at loc objs obj-loc)))))
60
61
62 (defun look ()
63   (append (describe-location *location* *nodes*)
64           (describe-paths *location* *edges*)
65           (describe-objects *location* *objects* *object-locations*)))
66
67
68 (defun walk (direction)
69   (let ((next (find direction
70                     (cdr (assoc *location* *edges*))
71                     :key #'cadr)))
72     (if next
73         (progn (setf *location* (car next))
74                (look))
75         '(you cannot go that way.)))
76
77
78 (defun pickup (object)
79   (if (member object (objects-at *location* *objects* *object-locations*))
80       (progn (push (list object 'body) *object-locations*)
81              `(you are now carrying the ,object))
82       '(you cannot get that.)))
83
84
85 (defun inventory ()
86   (cons 'items- (objects-at 'body *objects* *object-locations*)))

```

## Chunks

(\* 1) [1](#), [2f](#), [3h](#), [4d](#), [4g](#), [5d](#), [5f](#), [6g](#), [6l](#), [7b](#)  
 <A beautiful garden [2b](#)> [2b](#), [2d](#), [8b](#)  
 <adjust the player's position [6d](#)> [6d](#), [6f](#)  
 <admonish the player [6e](#)> [6e](#), [6f](#), [8b](#)  
 <All you have is whiskey. [7h](#)> [7h](#), [8b](#)  
 <at-loc-p [4f](#)> [4f](#), [4g](#)  
 <attic ladder [3c](#)> [3c](#), [7c](#)  
 <attic path [3f](#)> [3f](#), [3g](#)  
 <bin/runtests [9j](#)> [9j](#)  
 <Convert the edges to descriptions. [4b](#)> [4b](#), [4d](#)  
 <Convert the objects to descriptions. [5b](#)> [5b](#), [5d](#)  
 <define the global variables [2d](#)> [1](#), [2d](#), [3g](#), [4e](#), [5e](#)  
 <describe-obj [5c](#)> [5c](#), [5d](#)  
 <Exit with an appropriate status code. [9b](#)> [9b](#)  
 <Find the objects at the current location. [5a](#)> [5a](#), [5d](#)  
 <Find the relevant edges. [4a](#)> [4a](#), [4d](#)  
 <garden door [3a](#)> [3a](#), [7c](#), [8a](#)  
 <garden object descriptions [7f](#)> [7f](#), [8b](#)  
 <garden path [3e](#)> [3e](#), [3g](#)  
 <garden path description [7e](#)> [7e](#), [8b](#)  
 <get the list of objects here [6i](#)> [6h](#), [6i](#)  
 <init.lisp [9k](#)> [9k](#)  
 <Join the descriptions. [4c](#)> [4c](#), [4d](#), [5d](#)  
 <living-room object descriptions [7d](#)> [7d](#), [8a](#), [8b](#)  
 <living-room path descriptions [7c](#)> [7c](#), [8a](#), [8b](#)  
 <living-room paths [3b](#)> [3b](#), [3d](#), [3g](#)  
 <Load the test package. [9h](#)> [9h](#), [9j](#)  
 <locate the path marked with the appropriate direction [6c](#)> [6c](#), [6g](#)  
 <look up a location [2e](#)> [2e](#), [2f](#)  
 <look up the available walkings paths [6a](#)> [6a](#), [6c](#)  
 <match against the cadr of each path [6b](#)> [6b](#), [6c](#)  
 <pick it up [6j](#)> [6j](#), [6l](#)  
 <retrieve the list of carried objects [7a](#)> [7a](#), [7b](#)

<Run the system tests and exit. [9d](#)> [9d](#), [9j](#)  
 <Run the system tests. [9c](#)> [9c](#), [9d](#)  
 <script footer [9i](#)> [9i](#), [9j](#)  
 <script header [9e](#)> [9e](#), [9f](#), [9g](#), [9j](#)  
 <Set the exit status. [9a](#)> [9a](#)  
 <Test the exported functions in lol.wizard5. [8b](#)> [7i](#), [8b](#)  
 <Test the private functions in lol.wizard5 [8a](#)> [7i](#), [8a](#)  
 <test/wizard5.lisp [7i](#)> [7i](#)  
 <The attic [2c](#)> [2c](#), [2d](#)  
 <The living room [2a](#)> [2a](#), [2d](#), [8a](#), [8b](#)  
 <the object is on the floor [6h](#)> [6h](#), [6l](#)  
 <try to go in that direction [6f](#)> [6f](#), [6g](#)  
 <you cannot get that. [6k](#)> [6k](#), [6l](#), [8b](#)  
 <You've got whiskey! [7g](#)> [7g](#), [8b](#)

## Index

\*edges\*: [3g](#), [5f](#), [6a](#), [8a](#)  
 \*location\*: [5e](#), [5f](#), [6a](#), [6d](#), [6i](#)  
 \*nodes\*: [2d](#), [5f](#), [8a](#)  
 \*object-locations\*: [4e](#), [5f](#), [6i](#), [6j](#), [7a](#), [8a](#), [8b](#)  
 \*objects\*: [4e](#), [5f](#), [6i](#), [7a](#), [8a](#), [8b](#)  
 describe-location: [2f](#), [5f](#), [8a](#)  
 describe-objects: [5d](#), [5f](#), [8a](#), [8b](#)  
 describe-path: [3h](#), [8a](#)  
 describe-paths: [4d](#), [5f](#), [8a](#)  
 inventory: [1](#), [7b](#), [8b](#)  
 lol.wizard5: [1](#), [7i](#), [8a](#), [8b](#)  
 look: [1](#), [5f](#), [6f](#), [8b](#)  
 objects-at: [4g](#), [5a](#), [6i](#), [7a](#), [8a](#), [8b](#)  
 pickup: [1](#), [6l](#), [8b](#)  
 walk: [1](#), [6g](#), [8b](#)

3

## Glossary

*association list* a list of **conses** representing an association of keys with values, where the **car** of each **cons** is the key and the **cdr** is the associated value. 2

**caddr** (**lambda** (**x**) (**car** (**cdr** (**cdr** **x**)))) 3

**cadr** (**lambda** (**x**) (**car** (**cdr** **x**))) 2, 3

**car**

1.
  - a. the first component of a **cons**; the other is the **cdr**.
  - b. the head of a list, or **nil** if the list is the *empty list*.
2. the *object* that is held in the **car**. "The function **car** returns the **car** of a **cons**."

13

**cdr**

1.
  - a. the second component of a **cons**; the other is the **car**.
  - b. the tail of a list, or **nil** if the list is the *empty list*.
2. the *object* that is held in the **cdr**. "The function **cdr** returns the **cdr** of a **cons**."

13

**cons**

1. a compound data *object* made up of a **car** and a **cdr**.
2. to create such an *object*.
3. to create any *object* or to allocate storage.

13

*empty list* the list containing no elements. 13

**nil** represents both boolean **false** and the *empty list*. Alternatively notated as **()** to emphasize its use as an *empty list*. 13

*object* any Lisp datum. 13

3

Kent M. Pitman. CLHS: Glossary. [http://www.lispworks.com/documentation/HyperSpec/Body/26\\_a.htm](http://www.lispworks.com/documentation/HyperSpec/Body/26_a.htm), April 2005. Accessed: 2017-10-17

## References

Conrad Barski. *Land of Lisp: Learn to Program in Lisp, One Game at a Time!*, chapter 5, pages 67–84. No Starch Press, 2010. ISBN 9781593273491. URL <http://landoflisp.com>.

Kent M. Pitman. CLHS: Glossary. [http://www.lispworks.com/documentation/HyperSpec/Body/26\\_a.htm](http://www.lispworks.com/documentation/HyperSpec/Body/26_a.htm), April 2005. Accessed: 2017-10-17.