Programming Languages

Spring 2019 Lisp Quick Reference Sheet

Dr. Gurka

-----------------------------------------------------------------------------------------------------------------------------------------

Lisp functions (explained with examples below)

* defun, cond
* car, cdr
* cons, list, append
* atom, listp, numberp
* length
* +, -, \*, /, mod
* <, <=, >, >=
* =, equal (not eq or eql – see below)
* and, or, not, null

do not use

* if, loop, I/O functions (& many more)

-----------------------------------------------------------------------------------------------------------------------------------------

*writing a new function*

defun and cond

* defun: define a function, cond: conditional block
* form:

(defun func-name (args)

(cond ( (condition1) (action1) )

( (condition2) (action2) )

: :

(t (action-n)) << fall-through case, often recursive

)

)

* recursive examples:

(defun add-list (lst)

(cond ( (null lst) 0 )

( t (+ (car list) (add-list (cdr lst))))

)

)

(defun add-list-deep (lst)

(cond ( (null lst) 0 )

( (lisp (car lst)) (+ (add-list-deep (car lst))

(add-list-deep (cdr lst)))

( t (+ (car lst)

(add-list-deep (cdr lst))))

)

)

-----------------------------------------------------------------------------------------------------------------------------------------

-----------------------------------------------------------------------------------------------------------------------------------------

*list spliters*

car: the first element in a list

* form: (car some-list), result: an atom or a list
* examples: (car '(a b c)) 🡪 a, (car '( (cat dog) quiz)) 🡪 (cat dog), (car nil) 🡪 nil

cdr: a list of all elements in a original list except the first

* form: (cdr some-list), result: a list
* examples: (cdr '(a b c)) 🡪 (b c), (cdr '( (cat dog) quiz))🡪 (quiz), (cdr nil) 🡪 nil

-----------------------------------------------------------------------------------------------------------------------------------------

*list constructors*

cons

* creates a new list: first argument becomes the new first element of its second argument
* form: (cons thing list), result: a list
* examples:
  + (cons 'cat '(a b c)) 🡪 (cat a b c)
  + (cons '(a b c) '(cat dog)) 🡪 ((a b c) cat dog)

list

* creates a new list whose elements are the arguments, adds an extra level of nesting
* form: (list args), result: a list
* examples:
  + (list nil) 🡪 (nil), (list 'x) 🡪 (x), (list 'cat 'dog) 🡪 (cat dog)
  + (list '(cat dog) nil) 🡪 ( (cat dog) nil)
  + (list '(a b c) '(x y z)) 🡪 ( (a b c) (x y z) )
  + (list 'dog '(1 2) 'cat) 🡪 (dog (1 2) cat)

append

* creates a new list by merging lists, does not add an extra level of nesting
* form: (append lists), result: a list
* examples:
  + (append nil '(a b c)) 🡪 (a b c),
  + (append '(cat dog) nil )🡪 (cat dog)
  + (append '(a b c) '(x y z)) 🡪 (a b c x y z)
  + (append '(a b) '(1 2 3 4) '(cat)) 🡪 (a b 1 2 3 4 cat)

-----------------------------------------------------------------------------------------------------------------------------------------

-----------------------------------------------------------------------------------------------------------------------------------------

*types*

atom: returns true if its argument is an atom, nil otherwise

* form: (atom thing)
* examples: (atom 'a) 🡪 T, (atom '(1 2)) 🡪 nil, (atom nil) 🡪 T

listp: returns true if its argument is a list, nil otherwise

* form: (list thing)
* examples: (listp '(1 2)) 🡪 T, (listp 'a) 🡪 nil, (listp nil) 🡪 T

numberp: returns true if its argument is a number, nil otherwise

* form: (numberp thing)
* examples: (numberp 55) 🡪 T, (numberp 'a) 🡪 nil, (numberp '(2) ) 🡪 nil

-----------------------------------------------------------------------------------------------------------------------------------------

*number functions*

length: number of top-level elements in a list

* form: (length some-list)
* examples: (length '(a b c (1 2) )) 🡪 4, (length nil) 🡪 0

math: +, -, \*, /

* form: (math-op arg1 arg2 … )
* example: ( + 2 3 4 ) 🡪 9
* integer division produces an integer if there is no remainder, a rational number otherwise
  + examples: ( / 10 2) 🡪 5, ( / 10 9 ) 🡪 10/9
* real division works as expected
  + example: ( / 10.0 3) 🡪 3.3333333
* unary operators work as expected
  + example: ( + +5 -4 ) 🡪 1
* use *mod* for integers only

inequality relational operators (numbers)

* functions (4): <, <=, >, >=
* form: (rel-op number1 number2)
* example: (<= 7 5) 🡪 nil

equality

* form: (equal-op arg1 arg2)
* functions (2): =, equal
  + use = for numeric arguments
    - examples: ( = 4 4 ) 🡪 T, ( = 2.0 2 ) 🡪 T, ( = 5 9 ) 🡪 nil
  + use *equal* for atoms or lists of arbitrary depth
    - examples: (equal 'a 'a ) 🡪 T, (equal '(1 2 '(a b c) ) '(1 2 '(a b c) ) ) 🡪 T
* do not use *eq* or *eql* since they vary by implementation

-----------------------------------------------------------------------------------------------------------------------------------------

-----------------------------------------------------------------------------------------------------------------------------------------

*logical operations*

and, or, not, null

* anything non-nil is true (for example, a number or a list)
* *and* and *or*
  + evaluate left to right and short circuit
  + *and* returns the last element if nil was not found earlier, nil otherwise
  + *or* returns the first true element
  + examples: (and 'cat nil 88) 🡪 nil, (and 'a '(1 2 3)) 🡪 (1 2 3), (or nil 'dog) 🡪 dog
* *not* and *null* are equivalent and invert their argument

-----------------------------------------------------------------------------------------------------------------------------------------

misc notes

* nil and () are equivalent, can be used interchangeably, and mean the empty list
* nil is both an atom and a list
* numbers can be integer or real
* consider uppercase and lowercase the same, specifically:
  + nil is the same as NIL, t is the same as T, cat is the same as CAT
* a single quotation mark suppresses evaluation of a list or an atom

examples:

* + '(x y z) does not attempt to evaluate *x* as a function
  + 'cat does not attempt to find the value bound to the atom *cat*
* some of the functions listed have additional special cases; if you use one of those, write a note explaining the special case and give the IDE/interpreter you were using
* LispWorks is one available Lisp environment, it must be installed
* use the *load* function to bring a Lisp program (plain text file) into the environment

-----------------------------------------------------------------------------------------------------------------------------------------