




Periodic Table of the Loop Macro

simple loop <pre>(loop (princ "type something" (force-output) (read)))</pre>	<p>AN ASTERISK* MEANS A WORD CAN HAVE AN "ING" FORM (SO "SUM" AND "SUMMING" ARE BOTH LEGAL)</p> <p>CREATE A LOCAL</p>		
do* <pre>(loop for i below 5 do (print i))</pre>	<p>NAMING & BREAKING OUT OF LOOPS</p>	<p>HASH TABLES</p>	<p>with <VARIABLE</p> <pre>(loop with x = (+ 1 2) repeat 5 do (print x))</pre>
repeat <pre>(loop repeat 5 do (print "Prints five times"))</pre>	named <pre>(loop named outer for i below 10 do (progn (print "outer") (loop named inner for x below i do (print "**inner") when (= x 2) do (return-from outer 'kicked-out-all-the-way))))</pre>	using <pre>(defparameter salary (make-hash-table)) (setf (gethash 'bob salary) 80) (setf (gethash 'john salary) 90) (loop for person being each hash-key of salary using (hash-value amt) do (print (cons person amt)))</pre>	being <pre>(defparameter salary (make-hash-table)) (setf (gethash 'bob salary) 80) (setf (gethash 'john salary) 90) (loop for person being each hash-key of salary do (print person))</pre>
return <pre>(loop for i below 10 when (= i 5) return 'leaving-early do (print i))</pre>	return-from <pre>(loop named outer for i below 10 do (progn (print "outer") (loop named inner for x below i do (print "**inner") when (= x 2) do (return-from outer 'kicked-out-all-the-way))))</pre>	the <i>SAME</i>  <pre>(defparameter salary (make-hash-table)) (setf (gethash 'bob salary) 80) (setf (gethash 'john salary) 90) (loop for person being the hash-keys of salary do (print person))</pre>	each <pre>(defparameter salary (make-hash-table)) (setf (gethash 'bob salary) 80) (setf (gethash 'john salary) 90) (loop for person being each hash-key of salary do (print person))</pre>
initially <pre>(loop initially (print 'loop-begin) for x below 3 do (print x))</pre>	while <pre>(loop for i in '(0 2 4 555 6) while (evenp i) do (print i))</pre>	hash-keys <i>SAME</i>  <pre>(defparameter salary (make-hash-table)) (setf (gethash 'bob salary) 80) (setf (gethash 'john salary) 90) (loop for person being the hash-keys of salary do (print person))</pre>	hash-key <pre>(defparameter salary (make-hash-table)) (setf (gethash 'bob salary) 80) (setf (gethash 'john salary) 90) (loop for person being each hash-key of salary do (print person))</pre>
finally <pre>(loop for x below 3 do (print x) finally (print 'loop-end))</pre>	until <pre>(loop for i from 0 do (print i) until (> i 3))</pre>	hash-values <i>SAME</i>  <pre>(defparameter salary (make-hash-table)) (setf (gethash 'bob salary) 80) (setf (gethash 'john salary) 90) (loop for amt being the hash-values of salary do (print amt))</pre>	hash-value <pre>(defparameter salary (make-hash-table)) (setf (gethash 'bob salary) 80) (setf (gethash 'john salary) 90) (loop for amt being each hash-value of salary do (print amt))</pre>

<p>"FOR" LOOPING</p>				<p>BUILDING CONDITIONS</p>		<p>EXTRACTING A RESULT</p>	
<p>for <i>SAME</i> as</p> <pre>(loop for i from 0 do (print i) when (= i 5) return 'zuchini)</pre>	<p>as</p> <pre>(loop as x from 5 to 10 collect x)</pre>			<p>if</p> <pre>(loop for i below 5 if (oddp i) do (print i))</pre>	<p>count*</p> <pre>(loop for i in '(1 1 1 1) count i)</pre>		
<p>in <i>SAME</i> on</p> <pre>(loop for i in '(100 20 3) sum i)</pre>	<p>on</p> <pre>(loop for x on '(1 3 5) do (print x))</pre>	<p>across</p> <pre>(loop for i across #(100 20 3) sum i)</pre> <p><i>FOR ↑ ARRAYS</i></p>	<p>into</p> <pre>(loop for i in '(3 8 73 4 -5) minimize i into lowest maximize i into biggest finally (return (cons lowest biggest)))</pre> <p><i>LETS YOU CREATE LOCAL VARIABLES TO RETURN</i></p>	<p>when</p> <pre>(loop for i below 4 when (oddp i) do (print i) do (print "yup"))</pre>	<p>sum*</p> <pre>(loop for i below 5 sum i)</pre>		
<p>by</p> <pre>(loop for i from 6 to 8 by 2 sum i)</pre> <p><i>CONTROL INCREMENTS OF A FOR LOOP</i></p>	<p>from</p> <pre>(loop for i from 6 to 8 sum i)</pre>	<p>to</p> <pre>(loop for i from 6 to 8 sum i)</pre>	<p>always</p> <pre>(loop for i in '(0 2 4 6) always (evenp i))</pre> <p><i>CHECK COLLECTION FOR TRUTH OF A CONDITION</i></p>	<p>unless</p> <pre>(loop for i below 4 unless (oddp i) do (print i))</pre>	<p>minimize*</p> <pre>(loop for i in '(3 2 1 2 3) minimize i)</pre>		
<p>then</p> <pre>(loop repeat 5 for x = 10.0 then (/ x 2) collect x)</pre>	<p>upfrom</p> <pre>(loop for i upfrom 6 to 8 sum i)</pre>	<p>upto</p> <pre>(loop for i from 6 upto 8 sum i)</pre>	<p>never</p> <pre>(loop for i in '(0 2 4 6) never (oddp i))</pre>	<p>and</p> <pre>(loop for x below 5 when (= x 3) do (print "do this") and do (print "also do this") do (print "always do this"))</pre>	<p>maximize*</p> <pre>(loop for i in '(1 2 3 2 1) maximize i)</pre>		
	<p>downfrom</p> <pre>(loop for i downfrom 10 to 7 do (print i))</pre>	<p>downto</p> <pre>(loop for i from 10 downto 7 do (print i))</pre>	<p>thereis</p> <pre>(loop for i in '(0 2 4 555 6) thereis (oddp i))</pre>	<p>else</p> <pre>(loop for i below 5 if (oddp i) do (print i) else do (print "w00t"))</pre>	<p>append*</p> <pre>(loop for i below 5 append (list 'Z i))</pre>		
				<p>end</p> <pre>(loop for i below 4 when (oddp i) do (print i) end do (print "yup"))</pre>	<p>nconc*</p> <pre>(loop for i below 5 nconc (list 'Z i))</pre>		