

# Periodic Table of the Loop Macro

<b>simple loop</b>  <pre>(loop   (print     "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>		
<b>do*</b>  <pre>(loop for i below 5   do (print i))</pre>	<p>NAMING &amp; BREAKING OUT OF LOOPS</p>	<p>HASH TABLES</p>	<p>with <b>&lt;VARIABLE</b></p> <pre>(loop with x = (+ 1 2)   repeat 5   do (print x))</pre>
<b>repeat</b>  <pre>(loop repeat 5   do (print     "Prints five times"))</pre>	<b>named</b>  <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>	<b>using</b>  <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>	<b>being</b>  <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>
<b>return</b>  <pre>(loop for i below 10   when (= i 5)   return   'leaving-early   do (print i))</pre>	<b>return-from</b>  <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>	<b>the</b> <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>	<b>each</b>  <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>
<b>initially</b>  <pre>(loop initially   (print     'loop-begin)   for x below 3   do (print x))</pre>	<b>while</b>  <pre>(loop for i in '(0 2 4 555 6)   while (evenp i)   do (print i))</pre>	<b>hash-keys</b> <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>	<b>hash-key</b>  <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>
<b>finally</b>  <pre>(loop for x below 3   do (print x)   finally   (print 'loop-end))</pre>	<b>until</b>  <pre>(loop for i   from 0   do (print i)   until (&gt; i 3))</pre>	<b>hash-values</b> <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>	<b>hash-value</b>  <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><i>"FOR" LOOPING</i></p>				<p><i>BUILDING CONDITIONS</i>      <i>EXTRACTING A RESULT</i></p>	
<b>for</b> <i>SAME</i> <pre>(loop for i   from 0   do (print i)   when (= i 5)   return   'ruchini)</pre>	<b>as</b> <pre>(loop as x   from 5   to 10   collect x)</pre>			<b>if</b> <pre>(loop for i   below 5   if (oddp i)   do (print i))</pre>	<b>count*</b> <pre>(loop   for i   in '(1 1 1 1)   count i)</pre>
<b>in</b> <pre>(loop   for i   in '(100 20 3)   sum i)</pre>	<b>on</b> <pre>(loop for x   on '(1 3 5)   do (print x))</pre>	<b>across</b> <pre>(loop for i   across   #(100 20 3)   sum i)</pre> <p><i>FOR ↑ ARRAYS</i></p>	<b>into</b> <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>	<b>when</b> <pre>(loop for i   below 4   when (oddp i)   do (print i)   do (print "yup"))</pre>	<b>sum*</b> <pre>(loop   for i below 5   sum i)</pre>
<b>by</b> <pre>(loop for i   from 6   to 8 by 2   sum i)</pre> <p><i>CONTROL INCREMENTS OF A FOR LOOP</i></p>	<b>from</b> <pre>(loop for i   from 6   to 8   sum i)</pre>	<b>to</b> <pre>(loop for i   from 6   to 8   sum i)</pre>	<b>always</b> <pre>(loop for i   in '(0 2 4 6)   always (evenp i))</pre> <p><i>CHECK COLLECTION FOR TRUTH OF A CONDITION</i></p>	<b>unless</b> <pre>(loop for i   below 4   unless (oddp i)   do (print i))</pre>	<b>minimize*</b> <pre>(loop   for i   in '(3 2 1 2 3)   minimize i)</pre>
<b>then</b> <pre>(loop repeat 5   for x = 10.0   then (/ x 2)   collect x)</pre>	<b>upfrom</b> <pre>(loop for i   upfrom 6   to 8   sum i)</pre>	<b>upto</b> <pre>(loop for i   from 6   upto 8   sum i)</pre>	<b>never</b> <pre>(loop for i   in '(0 2 4 6)   never (oddp i))</pre>	<b>and</b> <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>	<b>maximize*</b> <pre>(loop   for i   in '(1 2 3 2 1)   maximize i)</pre>
	<b>downfrom</b> <pre>(loop for i   downfrom 10   to 7   do (print i))</pre>	<b>downto</b> <pre>(loop for i   from 10   downto 7   do     (print i))</pre>	<b>thereis</b> <pre>(loop for i   in '(0 2 4 555 6)   thereis (oddp i))</pre>	<b>else</b> <pre>(loop for i   below 5   if (oddp i)   do (print i)   else   do     (print "w00t"))</pre>	<b>append*</b> <pre>(loop for i   below 5   append   (list 'Z i))</pre>
				<b>end</b> <pre>(loop for i   below 4   when (oddp i)   do (print i)   end   do (print "yup"))</pre>	<b>nconc*</b> <pre>(loop for i   below 5   nconc   (list 'Z i))</pre>