Periodic Table of the Loop Macro

simple loop

```
(loop
  (princ
      "type something"
  (force-output)
  (read))
```

AN ASTERISK * MEANS A WORD CAN HAVE AN "ING" FORM (SO "SUM" AND "SUMMING" ARE BOTH LEGAL)

NAMING & BREAKING OUT OF LOOPS

HASH TABLES

CREATE A LOCAL VARIABLE

do*

```
(loop for i below 5
      do (print i))
```

with

```
(loop with x = (+ 1 2)
      repeat 5
      do (print x))
```

repeat

```
(loop repeat 5
   do (print
    "Prints five times"))
```

named

```
(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))))
```

using

```
(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)))
```

being

```
(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))
```

return

```
(loop for i below 10
      when (= i 5)
      return
       'leaving-early
      do (print i))
```

return-from

```
(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))))
```

the

SAME

```
(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))
```

each

```
(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))
```

initially

```
(loop initially
      (print
         'loop-begin)
      for x below 3
      do (print x))
```

while

```
(loop for i in '(0 2 4 555 6)
      while (evenp i)
      do (print i))
```

hash-keys

SAME

```
(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))
```

hash-key

```
(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))
```

finally

```
(loop for x below 3
      do (print x)
      finally
    (print 'loop-end))
```

until

```
(loop for i
      from 0
      do (print i)
      until (> i 3))
```

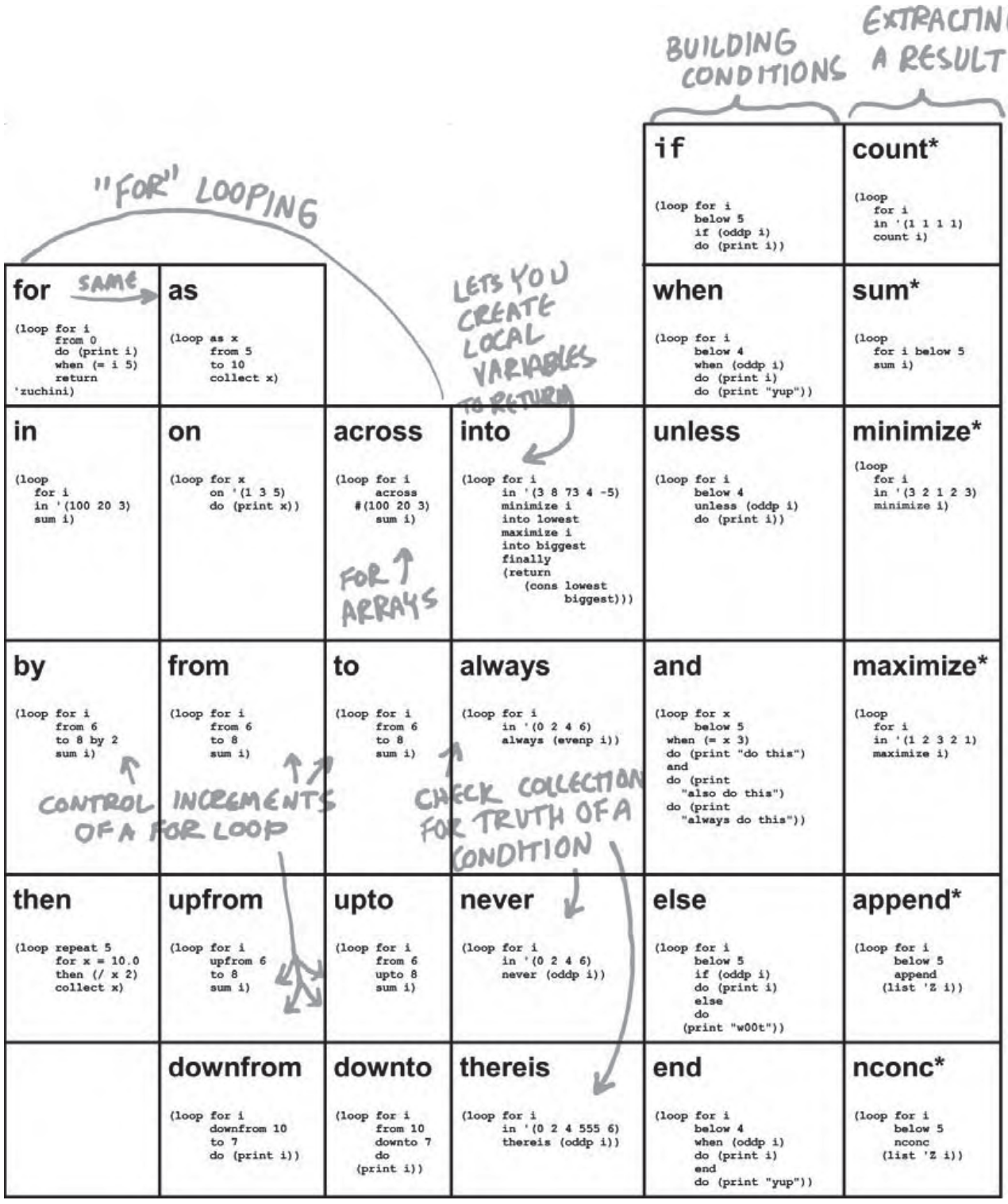
hash-values

SAME

```
(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))
```

hash-value

```
(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))
```


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