Control (and State changing, etc.)

To If ***** This has changed !!
The official def is found in the control explanation. I will copy it here sometime

To
.name ←. (:exp. ↑ exp)
"lookup the name in current environment (if not there, enter it as most global) and replace BINDING with value of "exp" ".

! name.
"note that the value of the expression on the right "exp" is RETURNed when a rebind is attempted, but when used as QUOTE, it is the name which is RETURNed."

To Eval :exp :globalenv :return :msg.
"There are many ways to EVALuate expressions in Smalltalk. This one allows the user to set up an arbitrary environment for free variable fetches, an arbtrary RETURN process, and an arbitrary MESSAGE environment."Eval" is included here since it is very frequently used in definitions of new control primitives".

To Repeat :howmany .Loopexp.
Code repeat.
Eval Loopexp :global :self EMPTY.
Code again.

"Repeat EVALs its loop expression in the context of its caller."

To Again
"RETURNs control to the caller of its caller--i.e. to a looping control primitive of some kind such as "Repeat" which can decide what to do next".

To Done
"RETURNs control to the caller of (the caller of its caller)--to one level further out than a looping control primitive. This automatically terminates the loop. Eventually "Done" will have an optional argument for passing the RESULT of the loop back".

To Create
"Reschedule caller to be run instead of waiting for a subroutine RETURN".
:call.
"This causes an evaluation of the argument. So it will also be running".

"As seen, "Create" causes a parallel fork in control. Actually, this is what happens naturally in SMALLTALK---the default message discipline is deliberately limited to a subroutine "wait for reply" protocol. "Create" simply prevents the caller from being passivated".

```
To Word

   ,Explain
      ↑"Words are like LISP atoms or ALGOL identifiers. Their basic
      operations have to do with assembly and disassembly of their
      internal structures.
         Words also have a special meaning in the context of
         evaluation. An unquoted instance of a word will be looked
         up (look itself up) when encounted by the EVALuator. So
         cat,first  means  "look up the most local binding of the
         variable "cat" and APPLY it to ,first". But ,cat,first
         means " call routine "," which  RETURNs the word "cat",
         which is APPLYed to ,first, which, as  seen below, will
         RETURN "c" ".
      Numbers are words also, but have many additional operations
      having to do with arithmetic and so are defined as a separate
      class.".

   ,← :value,word
      ↑self.
         "..."
   ,first
      ↑"the first character of the printname of the word".

   ,f
      ↑"same as "first"".

   ,last
      ↑"...the last character of the printname of the word"

   ,l
      ↑"...the same result as for "last". This is just an
      abbreviation."
   ,butfirst
      ↑"Somehow return all but the first character of the string
      representation of the word."

   ,bf
      ↑"...same as butfirst."

   ,butlast
      ↑"Somehow return all but the last character of the string
      representation of the word."

   ,bl
      ↑"...same as butlast."

   ,join :value1,word?
      ↑"This is roughly equivalent to the "cons" of LISP. The word
      will be connected to the list in "value1", and a new list
      reference will be returned."

   ,wjoin :value1,word?
      ↑"This is roughly equivalent to concatenate in SNOBOL. The
      printname of the two words are joined together to produce a
      new word which is returned. ,cat wjoin ,dog      produces
      ,catdog."
   ,word?
      ↑value.

   ,empty?
      ↑EMPTY.

   ,length
      ↑"Somehow calculate the length (in characters) of the
```

```
    number      (including "-" and ".")  ."

.print :
    1"Return a string representation of the object which may be
    displayed. Each class which has instances which have a
    meaningful visual representation will have a meaning for
    .print. This is much simpler than having to inform a global
    print routine about the format of each new class."
```

```
To Number

  .Explain
    ↑"Numbers work in a very intuitive way. The READ program
    recognizes  number literals and creates instances for them in
    storage.The bits  that represent the particular instance of a
    number are stored in  the variable "value" and can be changed
    by assignment as shown.  This might be illegal if it is
    decided that numbers are unique  atoms. The opposite is
    assumed here."

  .← :value.number?
    ↑ self.
      If a "↑" is recognized in the input stream, what follows is
       evaluated and bound to "value" which is applied to number?
      which  returns TRUE if it is. The actual value of the
      number object itself has been changed so that other objects
      which have pointers to "self" will feel the change. This
      might be made illegal.

  .first
    ↑"Somehow return the first character of the number which is
    "-" if negative, is "." if between 0 and 1, and a digit from 0
    to 9 otherwise. It may be reasonable to calculate this value
    rather than keep a string representation of the number
    around."

  .f
    ↑"...the same result as for "first". This is just an
    abbreviation."

  .last
    ↑"Somehow return the last character of the number which is
    "." if greater than 1 and known inexactly, and a digit from 0
    to 9 otherwise. It may be reasonable to calculate this value
    rather than keep a string representation of the number
    around."

  .l
    ↑"...the same result as for "last". This is just an
    abbreviation."

  .butfirst
    ↑"Somehow return all but the first character of the string
    representation of the number."

  .bf
    ↑"...same as butfirst."

  .butlast
    ↑"Somehow return all but the last character of the string
    representation of the number."

  .bl
    ↑"...same as butlast."

  .join :value1.word?
    ↑"This is roughly equivalent to the "cons" of LISP. The word
    will be connected to the list in "value1", and a new list
    reference will  be returned."

  .wjoin :value1.word?
    ↑"This is roughly equivalent to concatenate in SNOBOL. The
    printname of the two words are joined together to produce a
    new word which is returned. .cat wjoin .dog      produces
```

```
    ↓catdog."

↓number? ⇧
    ↑value.
    "Anything not EMPTY will act as TRUE."

↓word? ⇧
    ↑value.

↓empty? ⇧
    ↑EMPTY.

↓length ⇧
    ↑"Somehow calculate the length (in characters) of the
    number     (including "-" and ".")  ."

↓print ⇧
    ↑"Return a string representation of the object which may be
    displayed. Each class which has instances which have a
    meaningful visual representation will have a meaning for
    ↓print. This is much simpler than having to inform a global
    print routine about the format of each new class."


↓= ⇧ :value1↓number? ⇧
    ↑"value if value and value1 are numerically EQUAL, otherwise
    EMPTY.  Note that this allows "a=b=c" to work correctly."

↓≠ ⇧ :value1↓number? ⇧
    ↑"EMPTY if value and value1 are not numerically EQUAL,
    otherwise  value. Note that this allows "a≠b≠c" to work
    correctly."

↓< ⇧ :value1↓number? ⇧
    ↑"value if value is numerically less than value1, otherwise
    EMPTY.  Note that this allows "a<b<c" to work correctly."

↓> ⇧ :value1↓number? ⇧
    ↑"value if value is numerically greater than value1, otherwise
     EMPTY.. Note that this allows "a>b>c" to work correctly."

↓+ ⇧:value1↓number? ⇧
    ↑"value added to value1."

↓- ⇧:value1↓number? ⇧
    ↑"value1 subtracted from value."

↓* ⇧ :number? ⇧
    ↑"value multiplied by value1."

↓/ ⇧ :value1↓number? ⇧
    ↑"value divided by value1."

↓mod ⇧ :value1↓number? ⇧
    ↑"value modulo value1."

↓ip ⇧
    ↑"...the integer part of value."

↓fp ⇧
    ↑"...the fractional part of value."

↓exp ⇧
    ↑"...the exponent (to the base 10) of value."

↓mag ⇧
```

↑ if value < 0 then (0 - value) else value.

<other numeric functions which are stored as attributes>
sin, cos, other trig functions etc.