|  |  |
| --- | --- |
| TEA PRIMITIVE | SEMANTICS |
| N: | |  |  | | --- | --- | | NAME | Number | | PURPOSE | Generate a Random Number using some criteria | | SYNTAX  & SEMANTICS | |  | | --- | | n: | | Return a random whole number n in the range (0, 9) inclusive | | n:N1:N2:N3:GLUE  n:LIMIT:LLIMIT:SIZE:GLUE | | Same as n:, but using range (0,LIMIT) or rather n: invoked with arguments behaves as such:  The first argument, N1 sets the upper limit, so that the number generated is from the range 0 <= n < N1  With the second argument too, we also control the lower limit so that the number generated lies in the range N2 <= n < N1  The third argument tells the n: command to return utmost N3 numbers in the specified range. Finally, the last argument, GLUE, specifies how to glue the generated numbers. By default, the glue used is the SINGLE SPACE CHARACTER. | | n!: | | Same as n: | | n!:LIMIT | | Same as n:LIMIT | | n\*:vNAME  n\*:vLIMIT:vLLIMIT:vSIZE:vGLUE | | Same as n!:LIMIT or the longer form, n:LIMIT:LLIMIT:SIZE:GLUE but referencing values stored in vaults | |  | | NOTES | Much as TEA is generally considered to be a Text Processing language by definition, and yet, with the N: TEA command space, we find primitive utilities in TEA, that make numerical transforms and computations somewhat possible---given TEA is essentially a string, and not number processing language. Merely by having an inbuilt mechanism to generate random numbers within TEA programs, many interesting and useful mathematical, or rather, numerical processing problems become readily solvable.  The most basic Random Number Generator (RNG) possible, is simply implemented using the following minimalist TEA program:  N:  # (=6) even just n!: would similarly work  The parameterized version of the N-command space instructions can be illustrated with the following basic, but very potent example – perhaps an example for how to generate random but correct IP addresses  n!:256:0:4:.  # (=1.125.0.74) perhaps magical address to some \*special\* Internet asset?!  To appreciate this multi-parameter form of the Number command, consider that generating a random number in the range N2 <= n < N1 for N2 > 9 could be computed easily by computing a number X in the range 0 <= X < (N1 - N2), which is simpler to generate, then merely computing (N2 + X) --- refer to [1]  A pointer-based/vault-based version of the same exact above program is the following:  v:vL:256|v:vLL:0|v:vS:4|v:vG:.|n\*:vL:vLL:vS:vG  That will generate pseudo-IP addresses such as 43.28.26.154, 0.204.15.129, etc. |  | |