|  |  |
| --- | --- |
| TEA PRIMITIVE | SEMANTICS |
| P: | |  |  | | --- | --- | | NAME | Permutate | | PURPOSE | Generate unique permutations of things | | SYNTAX  & SEMANTICS | |  | | --- | | p: | | Return the AI expanded to utmost 100 unique  permutations of its elements (which essentially means returning anagrams of the AI), glued together | | p:VALUE  p:VALUE:GLUE  p:VALUE:GLUE:LIMIT  p:VALUE:GLUE:LIMIT:LLIMIT | | Same as p: but using VALUE instead of AI. Also, if the second argument is also provided, it becomes the GLUE string used for joining the permutations generated when constructing the IO. The default or if GLUE isn’t specified, is the SINGLE SPACE CHARACTER. The last argument is expected to be a number LIMIT, to limit how many permutations to return at most. If LLIMIT is provided, then exactly restrict the output to between LLIMIT as lower limit and LIMIT as upper. If LLIMIT = LIMIT, return exactly LIMIT permutations if they can be generated. | | p!: | | Return a random string of utmost 100 characters. | | p!:SIZE  p!:SIZE:ALPHABET  p!:SIZE:ALPHABET:GLUE | | Return a random string of exactly SIZE characters. If the second parameter, the string ALPHABET, is provided, then use only the characters in the provided string for generating the random words, otherwise, will use the full Latin alphabet extended with the SINGLE SPACE CHAR. GLUE serves to join the generated strings if provided, otherwise is the default. | | p\*: vVALUE  p\*: vVALUE:vGLUE  p\*: vVALUE:vGLUE:vLIMIT:vLLIMIT | | Permutates provided string. Same as p:VALUE:GLUE:LIMIT , but using the string parameters stored in the vaults with the specified names, vVALUE as the VALUE, vGLUE as glue, vLIMIT as limit. Only the first, parameter is mandatory. | | p\*!: vSIZE  p\*!: vSIZE:vALPHABET  p\*!: vSIZE:vALPHABET:vGLUE | | Same as p!:SIZE:ALPHABET:GLUE but operating on values stored in vaults. | |  | | NOTES | A basic illustration of this P: power can be demonstrated by the following example TEA program that returns all possible unique combinations of the first 3 English alphabet letters:  I!:{abc} | v:vA | v:vGLUE:- | p\*:vA:vGLUE  # (=”abc-acb-bac-bca-cab-cba”)  Perhaps, it is important to stress that since mathematically it is known that a string of N characters has at most N! (N-factorial) possible permutations or rather anagrams, then, without enforcing a hard limit such as the default 100, it can become very expensive to run the p: command especially on large values. Thus, where necessary, ensure to specify the LIMIT parameter when invoking p: commands.  P: maps a string to a set of its anagrams. This also means, one could do what P: commands do, with clever, or rather creative use of the TEA fundamental anagram command “**a!:**”. This interesting fact is illustrated by the following application of TEA:  **EXPERIMENTS|< 13:35:35 $>\*** awk -e "BEGIN{do{n++; system(\"tttt -i abc -c a!:\");}while(n<=100)}" | sort | uniq  Returns:    abc  acb  bac  bca  cab  cba  It produces similar output an invocation of p: would have given for the same input “abc”, however, it merely leverages the minimalist TEA program “a!:” and some clever use of the Awk programming language.  **NOTE**: The **P!:** command space is the only utility in TEA, with which one might elegantly generate random text. This is the equivalent of generating random numbers, for words. A utility many, if not most programming languages never provide a primitive solution for.  The most basic Random String Generator (RSG) possible, is simply implemented using the following minimalist TEA program:  P!:  # could return “fnudgzwhh ztnttwhehb iptkerl chwuljtiw”  While, the following basic TEA program is guaranteed to generate useful random strings of exactly 10 characters:  P!:10  # could return “ssrmykqzyz”, “dfooctwrid”, “gdo yoqqlt”, etc  Finally, an involved vault accessing example of p: is:  v:vSIZE:5|v:vALPH:"aeiou"|v:vGLUE:-|p\*!:vSIZE:vALPH:vGLUE  Which should return random string of exactly 5 characters, based on the provided alphabet "aeiou" glued with "-", e.g.  #aueoi-oueia-eoaiu-eoaui-eaiou-ueaoi-aouei-uoeai-iouea-ieauo |  |  | |