25 Key Runes

1 C	hoon hoon	=/ a	1	<rest-of-hoon></rest-of-hoon>
define a	variable of value noon with name skin			
= spec	noon a gate (one-armed core with battery hoon and sa	= a= nple sp	=@rs ec)	(add:rs a .1.0)
I- hoon	a Bate (one armed core with battery noon and sa		uu)	
produce	a trap (one-armed core with battery hoon and arm	n \$) and	l kick	it
_ spec	: alas (map term tome)			
produce	a door (a generalized gate, a core with sample bu	t many	arms)	which accepts sample spec
% (uni	t term) (map term tome)			
produce	a generic core (cell of [battery payload])			
x- hoon	hoon	(add 1	L 1)	
call a gat	te hoon (one-armed core) with sample hoon	•	•	
%~ wing	hoon hoon	`@t`~((x ne	0xf)
evaluate	an arm wing in a door (resolve the wing as a gat	e and ca	ll it)	
%= wing	(list (pair wing hoon)	\$(cour	nt +(count))
resolve a	wing wing with changes; frequently used with \$	to itera	te a ti	rap forward as a loop
++ term	hoon			•
produce	a normal arm with name term and content hoon			
+\$ term	spec			
produce	a structure arm (type definition) with name terr	and mo	old sp	ec
\$= skin	spec	foo=ba	ΞZ	
assign a	name skin to a hoon ("wrap a face around a hoo	n")		
\$_ hoon	· •	foo		
normali	ze structure to example			
^- spec	hoon	^- @ı	a bu	
typecast	armligitly			
	explicitly			
^+ hoon	hoon	^+ .1	L a	
^+ hoon typecast	hoon by example	^+ .1	La	
<pre>^+ hoon typecast ?: hoon</pre>	by example	^+ .1	l a th:rs	a .0) a (sub:rs .0 a))
<pre>^+ hoon typecast ?: hoon branch c</pre>	hoon by example hoon hoon onditionally on test; if hoon then hoon else hoon	^+ .1 ?:((gt	l a th:rs	a .0) a (sub:rs .0 a))
<pre>^+ hoon typecast ?: hoon branch c ?. hoon</pre>	hoon by example hoon onditionally on test; if hoon then hoon hoon hoon	^+ .1 ?:((gt ?.((gt	La th:rs th:rs	a .0) a (sub:rs .0 a)) a .0) (sub:rs .0 a) a)
<pre>^+ hoon typecast ?: hoon branch c ?. hoon reversed</pre>	hoon by example <u>hoon</u> hoon onditionally on test; if hoon then <u>hoon</u> else hoon <u>hoon</u> hoon conditionality; branch conditionally on test; if ho	^+ .1 ?:((gt ?.((gt	La th:rs th:rs a hoor	a .0) a (sub:rs .0 a)) a .0) (sub:rs .0 a) a) else <u>hoon</u>
<pre>^+ hoon typecast ?: hoon branch c ?. hoon reversed ?= spec</pre>	hoon by example hoon hoon onditionally on test; if hoon then hoon else hoon hoon hoon conditionality; branch conditionally on test; if ho wing	^+ .1 ?:((gt ?.((gt	L a th:rs th:rs t h: rs	a .0) a (sub:rs .0 a)) a .0) (sub:rs .0 a) a) a else <u>hoon</u>
<pre>^+ hoon typecast ?: hoon branch c ?. hoon reversed ?= spec test patte</pre>	hoon by example hoon hoon onditionally on test; if hoon then hoon else hoon hoon hoon conditionality; branch conditionally on test; if ho wing ern match, whether wing is type spec	^+ .1 ?:((gt ?.((gt	L a th:rs th:rs a hoon	a .0) a (sub:rs .0 a)) a .0) (sub:rs .0 a) a) a else <u>hoon</u>
<pre>^+ hoon typecast ?: hoon branch c ?. hoon reversed ?= spec test patte ?> hoon</pre>	hoon by example hoon hoon onditionally on test; if hoon then hoon else hoon hoon hoon conditionality; branch conditionally on test; if how wing ern match, whether wing is type spec hoon	^+ .1 ?:((gt ?.((gt	l a th:rs th:rs t hoon	a .0) a (sub:rs .0 a)) a .0) (sub:rs .0 a) a) else <u>hoon</u>
<pre>^+ hoon typecast ?: hoon branch c ?. hoon reversed ?= spec test patte ?> hoon assert pc</pre>	hoon by example hoon hoon onditionally on test; if hoon then hoon else hoon hoon hoon conditionality; branch conditionally on test; if ho wing ern match, whether wing is type spec hoon ositively that hoon and hoon match	^+ .1 ?:((gt ?.((gt	L a th:rs th:rs	a .0) a (sub:rs .0 a)) a .0) (sub:rs .0 a) a) a else <u>hoon</u>
<pre>^+ hoon typecast ?: hoon branch c ?. hoon reversed ?= spec test patte ?> hoon assert pc .^ spec</pre>	hoon by example hoon hoon onditionally on test; if hoon then hoon else hoon hoon hoon conditionality; branch conditionally on test; if ho wing ern match, whether wing is type spec hoon positively that hoon and hoon match hoon	^+ .1 ?:((gt ?.((gt pon ther	th:rs th:rs th:rs thoon	a .0) a (sub:rs .0 a)) a .0) (sub:rs .0 a) a) a else <u>hoon</u>
<pre>^+ hoon typecast ?: hoon branch c ?. hoon reversed ?= spec test patte ?> hoon assert pc .^ spec scry into</pre>	hoon by example hoon hoon onditionally on test; if hoon then hoon else hoon hoon hoon conditionality; branch conditionally on test; if ho wing ern match, whether wing is type spec hoon ositively that hoon and hoon match hoon	^+ .((gt ?.((gt ?.((gt pon ther .^(arc mold sj	L a th:rs th:rs hoon	a .0) a (sub:rs .0 a)) a .0) (sub:rs .0 a) a) else <u>hoon</u> y %) o the result
<pre>^+ hoon typecast ?: hoon branch c ?. hoon reversed ?= spec test patte ?> hoon assert pc .^ spec scry into :- hoon construct</pre>	hoon by example hoon hoon onditionally on test; if hoon then hoon else hoon hoon hoon conditionality; branch conditionally on test; if how wing ern match, whether wing is type spec hoon positively that hoon and hoon match hoon o vane namespace per instruction hoon and apply hoon t a cell (2-tuple); see also n-tuple constructor :*	^+ .1 ?:((gt ?.((gt pon then .^(arc mold si :- %s	th:rs th:rs th:rs thoon th %c pec to say	a .0) a (sub:rs .0 a)) a .0) (sub:rs .0 a) a) a else <u>hoon</u> y %) o the result foo
<pre>^+ hoon typecast ?: hoon branch c ?. hoon reversed ?= spec test patte ?> hoon assert pc .^ spec scry intc :- hoon construc .< mold</pre>	hoon by example hoon hoon onditionally on test; if hoon then hoon else hoon hoon hoon conditionality; branch conditionally on test; if ho wing ern match, whether wing is type spec hoon positively that hoon and hoon match hoon vane namespace per instruction hoon and apply hoon t a cell (2-tuple); see also <i>n</i> -tuple constructor :*	^+ .1 ?:((gt ?.((gt pon ther .^(arc mold sj :- %s	th:rs th:rs th:rs thoon th %c pec to say	a .0) a (sub:rs .0 a)) a .0) (sub:rs .0 a) a) d else <u>hoon</u> y %) o the result foo
<pre>^+ hoon typecast ?: hoon branch c ?. hoon reversed ?= spec test patte ?> hoon assert pc .^ spec scry intc :- hoon construc ;< mold monadic</pre>	hoon by example hoon hoon onditionally on test; if hoon then hoon else hoon hoon hoon conditionality; branch conditionally on test; if howing ern match, whether wing is type spec hoon ositively that hoon and hoon match hoon o vane namespace per instruction hoon and apply hoon t a cell (2-tuple); see also <i>n</i> -tuple constructor :* hoon hoon bind defer completion of hoon until after hoon hoon	^+ ?:((gt ?.((gt oon then .^(arc mold sj :- %s	th:rs th:rs	a .0) a (sub:rs .0 a)) a .0) (sub:rs .0 a) a) else <u>hoon</u> y %) o the result foo
<pre>^+ hoon typecast ?: hoon branch c ?. hoon reversed ?= spec test patte ?> hoon assert pc .^ spec scry intc :- hoon construcc ;< mold monadic /+ nath</pre>	hoon by example hoon hoon onditionally on test; if hoon then hoon else hoon hoon hoon conditionality; branch conditionally on test; if ho wing ern match, whether wing is type spec hoon ositively that hoon and hoon match hoon o vane namespace per instruction hoon and apply hoon t a cell (2-tuple); see also <i>n</i> -tuple constructor :* hoon hoon bind, defer completion of hoon until after hoon hoon	^+ .1 ?:((gt ?.((gt oon then .^(arc mold sj :- %s	th:rs th:rs hoon th %c pec to say ved; t	a .0) a (sub:rs .0 a)) a .0) (sub:rs .0 a) a) a else <u>hoon</u> y %) o the result foo noon is an adapter
<pre>^+ hoon typecast ?: hoon branch c ?. hoon reversed ?= spec test patte ?> hoon assert pc .^ spec scry intc :- hoon construc ;< mold monadic /+ path imports</pre>	hoon by example hoon hoon onditionally on test; if hoon then hoon else hoon hoon hoon conditionality; branch conditionally on test; if how wing ern match, whether wing is type spec hoon bitively that hoon and hoon match hoon vane namespace per instruction hoon and apply hoon t a cell (2-tuple); see also <i>n</i> -tuple constructor :* hoon hoon bind, defer completion of hoon until after hoon hoon a file from lib/nath (* ninned with no face = with the face = with th	^+ .1 ?:((gt ?.((gt pon then .^(arc mold s :- %s as resol /+ ge th speci	th:rs th:rs th:rs th:rs thoon thoon thoon thoon thoon thoon thoon	a .0) a (sub:rs .0 a)) a .0) (sub:rs .0 a) a) a else hoon y %) o the result foo noon is an adapter tors ace)
<pre>^+ hoon typecast ?: hoon branch c ?. hoon reversed ?= spec test patte ?> hoon assert pc .^ spec scry inte :- hoon construc ;< mold monadic /+ path imports ~& hoon</pre>	hoon by example hoon hoon onditionally on test; if hoon then hoon else hoon hoon hoon conditionality; branch conditionally on test; if ho wing ern match, whether wing is type spec hoon ositively that hoon and hoon match hoon o vane namespace per instruction hoon and apply hoon t a cell (2-tuple); see also <i>n</i> -tuple constructor :* hoon hoon bind, defer completion of hoon until after hoon h a file from lib/path (* pinned with no face, = with	^+ ?:((g1 ?.((g1 ?.((g1 pon then mold si :- %s as resol /+ ge th speci ~& [1	th:rs th:rs hoon th %c pec to say ved; h enera fied fa	a .0) a (sub:rs .0 a)) a .0) (sub:rs .0 a) a) else <u>hoon</u> y %) o the result foo noon is an adapter tors ace) bar> <baz>1</baz>
<pre>^+ hoon typecast ?: hoon branch c ?. hoon reversed ?= spec test patte ?> hoon assert pc .^ spec scry intc :- hoon construc ;< mold monadic /+ path imports ~& hoon side effect</pre>	hoon by example hoon hoon onditionally on test; if hoon then hoon else hoon hoon hoon conditionality; branch conditionally on test; if how wing ern match, whether wing is type spec hoon ositively that hoon and hoon match hoon o vane namespace per instruction hoon and apply hoon t a cell (2-tuple); see also <i>n</i> -tuple constructor :* hoon hoon bind, defer completion of hoon until after hoon hoon a file from lib/path (* pinned with no face, = with the stders	^+ ?:((gt ?.((gt oon ther .^(arc mold sj :- %s as resol /+ ge th speci ~& [1	th:rs th:rs th:rs hoon th %c pec to say ved; h enera fied fi foo <	a .0) a (sub:rs .0 a)) a .0) (sub:rs .0 a) a) else <u>hoon</u> y %) o the result foo noon is an adapter tors ace) bar> <baz>]</baz>
<pre>^+ hoon typecast ?: hoon branch c ?. hoon reversed ?= spec test patte ?> hoon assert pc .^ spec scry intc :- hoon construc ;< mold monadic /+ path imports ~& hoon side effec !> hoon</pre>	hoon by example hoon hoon onditionally on test; if hoon then hoon else hoon hoon hoon conditionality; branch conditionally on test; if howing ern match, whether wing is type spec hoon bositively that hoon point boon bind, defer completion of hoon bind, defer completion of hoon bind, defer completion of hoon to stderr ct: output value of hoon to stderr	<pre>^+ .1 ?:((g1 ?.((g1 pon then .^(arc mold si i- %s as resol /+ ge th speci ~& [1</pre>	th:rs th:rs hoon th %c pec to say ved; h enera fied fa foo <	a .0) a (sub:rs .0 a)) a .0) (sub:rs .0 a) a) a else <u>hoon</u> y %) o the result foo noon is an adapter tors ace) bar> <baz>]</baz>
<pre>^+ hoon typecast ?: hoon branch c ?. hoon reversed ?= spec test patta ?> hoon assert pc .^ spec scry intc :- hoon construct ;< mold monadic /+ path imports ~& hoon side effec !> hoon</pre>	hoon by example hoon hoon onditionally on test; if hoon then hoon else hoon hoon hoon conditionality; branch conditionally on test; if ho wing ern match, whether wing is type spec hoon ositively that hoon and hoon match hoon o vane namespace per instruction hoon and apply hoon t a cell (2-tuple); see also <i>n</i> -tuple constructor :* hoon hoon bind, defer completion of hoon until after hoon h a file from lib/path (* pinned with no face, = with a file from lib/path (* pinned with no face, = with ct: output value of hoon to stderr	^+ .1 ?:((g1 ?.((g1 ?.((g1 pon ther .^(arc mold s :- %s as resol /+ ge th speci ~& [1	th:rs th:rs th:rs th:rs thoon thoon ch %c pec to say ved; f enera fied fa foo <	a .0) a (sub:rs .0 a)) a .0) (sub:rs .0 a) a) a else hoon y %) o the result foo noon is an adapter tors ace) bar> <baz>]</baz>
<pre>^+ hoon typecast ?: hoon branch c ?. hoon reversed ?= spec test patte ?> hoon assert pc .^ spec scry intc ;< mold monadic /+ path imports ~& hoon side effec !> hoon wrap a n !!</pre>	hoon by example hoon hoon onditionally on test; if hoon then hoon else hoon hoon hoon conditionality; branch conditionally on test; if ho wing ern match, whether wing is type spec hoon ositively that hoon and hoon match hoon o vane namespace per instruction hoon and apply hoon t a cell (2-tuple); see also <i>n</i> -tuple constructor :* hoon hoon bind, defer completion of hoon until after hoon h a file from lib/path (* pinned with no face, = wi ct: output value of hoon to stderr hoon in its type; frequently used as the "type	^+ ?:((g1 ?.((g1 ?.((g1 pon ther mold si :- %s as resol /+ ge th speci ~& [1	th:rs th:rs hoon th %c pec to say ved; h enera fied fa foo < -:!>	a .0) a (sub:rs .0 a)) a .0) (sub:rs .0 a) a) a else <u>hoon</u> y %) o the result foo noon is an adapter tors ace) bar> <baz>]</baz>
<pre>^+ hoon typecast ?: hoon branch c ?. hoon reversed ?= spec test patte ?> hoon assert pc .^ spec scry intc :- hoon construcc ;< mold monadic /+ path imports ~& hoon side effec !> hoon wrap a m !! crash (no</pre>	hoon by example hoon hoon onditionally on test; if hoon then hoon else hoon hoon hoon conditionality; branch conditionally on test; if howing ern match, whether wing is type spec hoon ositively that hoon and hoon match hoon ovane namespace per instruction hoon and apply hoon t a cell (2-tuple); see also <i>n</i> -tuple constructor :* hoon hoon bind, defer completion of hoon until after hoon hoon a file from lib/path (* pinned with no face, = with a file from lib/path (* pinned with no face, = with ct: output value of hoon to stderr hoon in its type; frequently used as the "type o children); useful for stubbing out branches in definition bind, defer completion of hoon to stderr	<pre>^+ .f ?:((g1 ?.((g1 ?.((g1 ?.((g1 oon ther oon ther mold si :- %s as resol /+ ge th speci ~& [1 e spear" vvelopm</pre>	th:rs th:rs hoon th %c pec to say ved; h enera fied fa foo < -:!> ent	a .0) a (sub:rs .0 a)) a .0) (sub:rs .0 a) a) a else <u>hoon</u> y %) o the result foo noon is an adapter tors ace) bar> <baz>]</baz>