+ lus • Arms

+| label a chapter (produces no arm)

+$ [term spec] produce a structure arm (type definition)

+- [term hoon] produce a (normal) arm

+* [term term spec] define deferred expression (within a door)

| bar • Cores

|$ (lest term) spec produce a mold

|_ spec alas (map term tome) produce a door (a core with sample)

|: [hoon hoon] produce a gate with a custom sample

|% (unit term) (map term tome) produce a core (battery and payload)

|. hoon produce a trap (a core with one arm)

|^ hoon (map term tome) produce a core with a $ arm and compute the latter

|- hoon produce a trap (a core with one arm) and evaluate it

|~ [spec value] produce an iron gate

|* [spec value] produce a wet gate (a one-armed core with sample)

|= [spec value] produce a dry gate (a one-armed core with sample)

|@ (unit term) (map term tome) produce a wet core (battery and payload)

|? hoon produce a lead trap

$ buc • Structures

$| [spec hoon] _foo structure with verification

$_ hoon structure that normalizes to an example

$% (list spec) structure that recognizes a union tagged by head atom

$: (list spec) [a=foo b=bar c=baz] form a cell type (tuple)

$< [spec spec] structure from filter (excluding)

$> [spec spec] structure from filter (requiring)

$- [spec spec] structure that normalizes to an example gate

$^ hoon structure that normalizes a union tagged by head depth

$@ [spec hoon] repaired structure (using normalizing gate)

$~ [hoon spec] define a custom type default value

$@ [spec spec] structure that normalizes a union tagged by head atom

$= [skin spec] foo=bar structure that wraps a face around another structure

$? (list spec) ?(%foo %bar %baz) form a type from a union of other types

% cen • Calls

%_ [wing (list (pair wing hoon))] resolve a wing with changes, preserving type

%: [hoon (list hoon)] call a gate with many arguments

%. [hoon hoon] call a gate, inverted

%- [hoon hoon] (gat smp) call a gate

%^ [hoon hoon hoon hoon] call a gate with triple sample

%+ [hoon hoon hoon] call a gate with a cell sample

%~ [wing hoon hoon] ~(arm cr smp) evaluate an arm in a door

%* [wing hoon (list (pair winghoon))] evaluate an expression, then resolves a wing with changes

%= [wing (list (pair wing hoon))] ↵ resolve wing with changes foo(bar 1, baz 2)

: col • Cells

:_ [hoon hoon]
construct a cell, inverted

:- [hoon hoon] [foo bar] foo^bar
construct a cell, 2-tuple

:^ [hoon hoon hoon hoon] ↵
construct a cell, 4-tuple [foo bar baz qux]

:+ [hoon hoon hoon]
construct a cell, 3-tuple

:~ (list hoon) ~[foo bar baz]
construct a null-terminated list

:* (list hoon) [foo bar baz ...]
construct an n-tuple

:: mark a comment (digraph, not rune)

. dot • Nock

.^ [spec hoon]
load from namespace using Nock 12 (scry or peek)

.+ atom +(foo)
increment an atom using Nock 4

.* [hoon hoon]
evaluate using Nock 2

.= [hoon hoon] =(foo bar)
test for equality using Nock 5

.? hoon
test for cell or atom using Nock 3

/ fas • Imports (↔ford arm of %clay)

/$ %from %to
import mark conversion gate from /mar

/% %mark
import mark definition from /mar

/- foo, *bar, baz=qux
import a file from /sur (* no face, = specified face)

/+ foo, *bar, baz=qux
import a file from /lib (* no face, = specified face)

/= clay-raw /sys/vane/clay
import results of user-specified path with face

/* myfile %hoon /gen/myfile/hoon
import the contents of a file in the desk converted to a mark (build-time static data)

/~ face type /path
import contents of dir as face=(map @ta type)

/? pin version number (not enforced)

^ ket • Casts

^| hoon
convert a gold core to an iron core (invariant)

^: spec ,foo
produce a 'factory' gate for a type (switch from regular parsing to spec/type parsing)

^. [hoon hoon]
typecast on value

^- [spec hoon] `foo`bar
typecast by explicit type label

^+ [hoon hoon]
typecast by inferred type (a fence)

^& hoon
convert a core to a zinc core (covariant)

^~ hoon
fold constant at compile time

^* spec *foo
bunt, produces default mold value

^= [skin hoon] foo=bar
bind name to a value

^? hoon
convert a core to a lead core (bivariant)

; mic macros

;: [hoon (list hoon)] :(gat foo bar baz)
call a binary function as an *n*-ary function

;/ hoon
(Sail) yield tape as XML element

;< [spec hoon hoon hoon]
glue a pipeline together (monadic bind)

;+
(Sail) make a single XML node

;; [spec hoon]
normalize with a mold, asserting fixpoint

;~ [hoon (list hoon)]
glue a pipeline together with a product-sample adapter (monadic bind)

;*
(Sail) make list of XML nodes from Hoon expression

;= marl : hoot
(Sail) make a list of XML nodes

~ sig · Hints

~| [hoon hoon]
print in stack trace if failure

~$ [term hoon]
profiler hit counter

~_ [hoon hoon]
print in stack trace, user-formatted

~% [chum hoon tyre hoon]
register jet

~/ [chum hoon]
register jet with registered context

~< [$@(term [term hoon]) hoon]
raw hint, applied to product ("backward")

~> [$@(term [term hoon]) hoon]
raw hint, applied to computation ("forward")

~& [@ hoon]
cache computation

~% [@ud hoon hoon]
print (used for debugging)

~= [hoon hoon]
detect duplicate

~? [@ud hoon hoon hoon]
print conditionally (used for debugging)

~! [hoon hoon]
print type if compilation failure

= tis · Subject

=| [spec hoon]
combine default type value with the subject

=: [(list (pair wing hoon)) hoon]
change multiple legs in the subject

=, [hoon hoon]
expose namespace (defines a bridge)

=. [wing hoon hoon]
change one leg in the subject

=/ [skin hoon hoon]
combine a named noun with the subject

=< [hoon hoon] foo:bar
compose two expressions, inverted

=> [hoon hoon]
compose two expressions

=- [hoon hoon]
combine a new noun with the subject

=^ [skin wing hoon hoon]
pin the head of a pair; changes a leg with the tail

=+ [hoon hoon]
combine a new noun with the subject

=; [skin hoon hoon]
combine a named noun with the subject, inverted

=~ (list hoon)
compose many expressions

=* [(pair term (unit spec)) hoon hoon]
define an alias

=? [wing hoon hoon hoon]
change one leg in the subject conditionally

? wut conditionals

?| (list hoon) |(foo bar baz ...)
logical OR (loobean)

?: [hoon hoon hoon]
branch on a boolean test

?. [hoon hoon hoon]
branch on a boolean test, inverted

?< [hoon hoon]
assert false

?> [hoon hoon]
assert true

?- [wing (list (pair spec hoon))]
switch against type union, no default

?^ [wing hoon hoon]
branch on whether a wing of the subject is a cell

?+ [wing hoon (list (pair spec hoon))]
switch against a union, with default

?& (list hoon) &(foo bar baz ...)
logical AND (loobean)

?@ [wing hoon hoon]
branch on whether a wing of the subject is an atom

?~ [wing hoon hoon]
branch on whether a wing of the subject is null

?= [spec wing]
test pattern match

?! hoon !foo
logical NOT (loobean)

Terminators

== terminate running series of expressions
(digraph, not rune)

-- terminate core expression
(digraph, not rune)

! zap · Wild

!: `hoon`
turn on stack trace

!, `[*hoon hoon]`
emit AST of expression – `!,(*hoon expression)`

!. `hoon`
turn off stack trace

!< `hoon`
lift dynamic value into static context

!> `hoon`
wrap a noun in its type

!; `[hoon hoon]`
emit the type for an expression using the type of type

!@ `[(list wing) hoon hoon]`
evaluate conditional on existence of wing

!= `hoon`
make the Nock formula for a Hoon expression

!? `[$@(@ {@ @}) hoon]`
restrict Hoon Kelvin version

!! `~`
crash

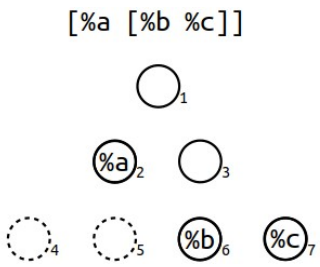
Nock 4K

A noun is an atom or a cell. An atom is a natural number. A cell is an ordered pair of nouns. Reduce by the first matching pattern; variables match any noun.

```
nock(a)              *a
[a b c]              [a [b c]]

?[a b]               0
?a                   1
+[a b]               +[a b]
+a                   1 + a
=[a a]               0
=[a b]               1

/[1 a]               a
/[2 a b]             a
/[3 a b]             b
/[(a + a) b]         /[2 /[a b]]
/[(a + a + 1) b]     /[3 /[a b]]
/a                   /a

#[1 a b]             a
#[(a + a) b c]       #[a [b /[(a + a + 1) c]] c]
#[(a + a + 1) b c]   #[a [/[(a + a) c] b] c]
#a                   #a

*[a [b c] d]        [*[a b c] *[a d]]

*[a 0 b]             /[b a]                                 slot operator (tree address)
*[a 1 b]             b                                      constant
*[a 2 b c]           *[*[a b] *[a c]]                       evaluate
*[a 3 b]             ?*[a b]                                test for atom
*[a 4 b]             +*[a b]                                increment
*[a 5 b c]           =[*[a b] *[a c]]                       distribution

*[a 6 b c d]         *[a *[[c d] 0 *[[2 3] 0 *[a 4 4 b]]]]  if-then-else
*[a 7 b c]           *[*[a b] c]                            compose
*[a 8 b c]           *[[*[a b] a] c]                        extend
*[a 9 b c]           *[*[a c] 2 [0 1] 0 b]                  invoke
*[a 10 [b c] d]      #[b *[a c] *[a d]]                     edit noun

*[a 11 [b c] d]      *[[*[a c] *[a d]] 0 3]                 hint
*[a 11 b c]          *[a c]

*a                   *a                                     interpret
```

Syntax

```
+1:[%a [%b %c]] [%a [%b %c]]
+2:[%a [%b %c]] %a
+3:[%a [%b %c]] [%b %c]
+4:[%a [%b %c]] %ride failed
+6:[%a [%b %c]] %b
+7:[%a [%b %c]] %c
```




```
.:[%a [%b %c]] [%a [%b %c]]
-:[%a [%b %c]] %a
+:[%a [%b %c]] [%b %c]
-<:[%a [%b %c]] invalid
+<:[%a [%b %c]] %b
+>:[%a [%b %c]] %c
```

&n	*n*th element
|n	tail after *n*th element
<[1 2 3]>	renders list as a tape
>[1 2 3]<	renders list as a tank
.	current subject
+	+:.
-	-:.
+>	+>:.
a.b.c	limb search path
~	0 (nil)
%.y	& yes/true/0
%.n	| no/false/1
%a	constant
$	empty term (@tas)
'urbit'	cord, atom @t
"urbit"	tape or list of characters
=wire	shadow type name (in defn)
/path	path name
%	current path

lark syntax equivalents

+1	+5 ->
+2 -	+6 +<
+3 +	+7 +>
+4 -<	+8 -<-

^face	face in outer core (^^face)
..arm	core in which ++arm is defined
,	,. strip the face
-:!>	type spear, use as -:!>(.3.14)

eny	entropy
now	current time
our	ship

```
`a [~ a]
~[a b c] [a b c ~]
[a b c]~ [[a b c] ~]
a/b [%a b]
```

elementary molds

*	noun
@	atom
^	cell
?	loobean
~	null

Aura Notation

Each aura has a characteristic pattern allowing unique identification in its representation. Typically this is indicated by a combination of `~`, `.`, and `-`.

`@`	Empty aura	
`@c`	Unicode codepoint	`~-~45fed.`
`@d`	Date	
`@da`	Date, absolute	`~2020.12.25..7.15.0..1ef5`
`@dr`	Date, relative	`~d71.h19.m26.s24..9d55`
`@f`	Loobean (for compiler, not castable)	`&`
`@i`	Internet address	
`@if`	IPv4 address	`.195.198.143.90`
`@is`	IPv6 address	`.0.0.0.0.0.1c.c3c6.8f5a`
`@n`	Nil (for compiler, not castable)	`~`
`@p`	Phonemic base	`~laszod-dozser-fosrum-fanbyr`
`@q`	Phonemic base, unscrambled (used with Urbit HD wallet)	`.~laszod-dozser-dalteb-hilsyn`
`@r`	IEEE-754 floating-point number	
`@rh`	Floating-point number, half-precision, 16-bit	`.~~3.14`
`@rs`	Floating-point number, single-precision, 32-bit	`.3.141592653589793`
`@rd`	Floating-point number, double-precision, 64-bit	`.~3.141592653589793`
`@rq`	Floating-point number, quadruple-precision, 128-bit	`.~~~3.141592653589793`
`@s`	Integer, signed (sign bit low)	
`@sb`	Signed binary	`--0b10.0000`
`@sd`	Signed decimal	`--1.000`
`@sv`	Signed base-32 `0123456789abcdefghijklmnopqrstuv`	`--0v201.4gvml.245kc`
`@sw`	Signed base-64 `0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ`	`--0w2.04AfS.G8xqc`
`@sx`	Signed hexadecimal `0123456789abcdef`	`--0x2004.90fd`
`@t`	UTF-8 text (cord)	`'urbit'`
`@ta`	ASCII text (knot)	`~.urbit`
`@tas`	ASCII text symbol (term)	`%urbit`
`@u`	Integer, unsigned	
`@ub`	Unsigned binary	`0b10.1011`
`@uc`	Bitcoin address `123456789abcdefghijklmnopqrstuvwxyzABCDEFGHJKLMNPQRSTUVWXYZ`	`0c1A1zP1eP5QGefi2DMPTfTL5SLmv7DivfNa`
`@ud`	Unsigned decimal	`8.675.309`
`@ui`	Unsigned decimal	`0i123456789`
`@uv`	Unsigned base-32 `0123456789abcdefghijklmnopqrstuv`	`0v88nvd`
`@uw`	Unsigned base-64 `0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ-~`	`0wx5~J`
`@ux`	Unsigned hexadecimal `0123456789abcdef`	`0x84.5fed`

Capital letters at the end of auras conventionally indicate the bitwidth in binary powers of two, starting from A = 2^0.

`@tD`	8-bit ASCII text
`@rhE`	half-precision (16-bit) floating-point number
`@uxG`	unsigned 64-bit hexadecimal
`@uvJ`	unsigned 512-bit integer (frequently used for entropy)