

+ 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]
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

_foo \$^ 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]
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 wing hoon))]
evaluate an expression, then resolves a wing with changes

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

: col · Cells	
:_ [hoon hoon]	:~ (list hoon) ~[foo bar baz]
construct a cell, inverted	construct a null-terminated list
:- [hoon hoon] [foo bar] foo^bar	:* (list hoon) [foo bar baz ...]
construct a cell, 2-tuple	construct an n-tuple
:^ [hoon hoon hoon hoon]	
construct a cell, 4-tuple [foo bar baz qux]	
:+ [hoon hoon hoon]	:: mark a comment (digraph, not rune)
construct a cell, 3-tuple	
. dot · Nock	
.^ [spec hoon]	.=[hoon hoon] =(foo bar)
load from namespace using Nock 12 (screy or peek)	test for equality using Nock 5
.+atom	.?hoon
increment an atom using Nock 4	test for cell or atom using Nock 3
.+atom +(foo)	
.* [hoon hoon]	
evaluate using Nock 2	
/ fas · Imports (++)ford arm of %clay)	
/\$ %from %to	/= clay-raw /sys/vane/clay
import mark conversion gate from /mar	import results of user-specified path with face
/% %mark	/* myfile %hoon /gen/myfile/hoon
import mark definition from /mar	import the contents of a file in the desk converted to a mark (build-time static data)
/- foo, *bar, baz=qux	/~ face type /path
import a file from /sur (* no face, = specified face)	import contents of dir as face=(map @ta type)
/+ foo, *bar, baz=qux	/? pin version number (not enforced)
import a file from /lib (* no face, = specified face)	
^ ket · Casts	
^ hoon	^& hoon
convert a gold core to an iron core (invariant)	convert a core to a zinc core (covariant)
^: spec	^~ hoon
produce a 'factory' gate for a type (switch from regular parsing to spec/type parsing)	fold constant at compile time
^. [hoon hoon]	^^ spec *foo
typecast on value	bunt, produces default mold value
^- [spec hoon]	^=[skin hoon] foo=bar
typecast by explicit type label	bind name to a value
^+ [hoon hoon]	^? hoon
typecast by inferred type (a fence)	convert a core to a lead core (bivariant)
; mic macros	
;; [hoon (list hoon)] : (gat foo bar baz)	;; [spec hoon]
call a binary function as an n-ary function	normalize with a mold, asserting fixpoint
;/ hoon	;;~ [hoon (list hoon)]
(Sail) yield tape as XML element	glue a pipeline together with a product-sample adapter (monadic bind)
;< [spec hoon hoon hoon]	;;*
glue a pipeline together (monadic bind)	(Sail) make list of XML nodes from Hoon expression
;;+	;;= marl:hoot
(Sail) make a single XML node	(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”)

= 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]
 compose two expressions, inverted
 => [hoon hoon]
 compose two expressions

foo:bar

~> [\$@ (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
 =- [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
 foo:bar =* [(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

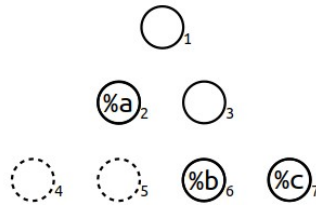
A noun is an atom or a cell. An atom is a natural number. A cell is an ordered pair of nouns. Reduce by the first matching pattern; variables match any noun.

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]] invalid
 +6:[%a [%b %c]] %b
 +7:[%a [%b %c]] %c

[%a [%b %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 nth element

|n tail after nth 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	--0v201.4gvml.245kc
	0123456789abcdefghijklmnopqrstuvwxyz	
@sw	Signed base-64	--0w2.04Afs.G8xqc
	0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ	
@sx	Signed hexadecimal	--0x2004.90fd
	0123456789abcdef	
@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	0c1A1zP1eP5QGefi2DMPTfTL5SLmv7DivfNa
	123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ	
@ud	Unsigned decimal	8.675.309
@ui	Unsigned decimal	0i123456789
@uv	Unsigned base-32	0v88nvd
	0123456789abcdefghijklmnopqrstuvwxyz	
@uw	Unsigned base-64	0wx5~J
	0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ--	
@ux	Unsigned hexadecimal	0x84.5fed
	0123456789abcdef	

Capital letters at the end of auras conventionally indicate the bitwidth in binary powers of two, starting from A = 2⁰.

@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)