

**| bars make cores**

**|\_** spec alas (map term tome)  
 produces a door (a core with sample)  
**|%** (unit term) (map term tome)  
 produces a core (battery and payload)  
**|@** (unit term) (map term tome)  
 produces a wet core (battery and payload)  
**|:** [hoon hoon]  
 produces a gate with a custom sample  
**|.** hoon  
 produces a trap (a core with one arm)  
**|-** hoon  
 produces a trap (a core with one arm) and evaluates it  
**|^** hoon (map term tome)  
 produces a core whose battery includes a \$ arm and computes the latter  
**|~** [spec value]  
 produces an iron gate  
**|\*** [spec value]  
 produces a wet gate (a one-armed core with sample)  
**|=** [spec value]  
 produces a dry gate (a one-armed core with sample)  
**|?** hoon  
 produces a lead trap  
**|\$** (lest term) spec  
 produces a mold

---

**\$ bucs form molds**

**\$@** [spec spec]  
 structure that normalizes a union tagged by head atom  
**\$\_** hoon  
 structure that normalizes to an example \_foo  
**\$:** (list spec)  
 forms a cell type (tuple) [a=foo b=bar c=baz]  
**\$\$** (list spec)  
 structure that recognizes a union tagged by head atom (e.g., a list of named parameters)  
**\$<** [spec spec]  
 structure from filter (excluding)  
**\$>** [spec spec]  
 structure from filter (requiring)  
**\$|** [spec hoon]  
 structure with verification  
**\$&** [spec hoon]  
 repaired structure  
**\$\$** hoon  
 structure that normalizes a union tagged by head depth (cell)  
**\$~** [hoon spec]  
 defines a custom type default value  
**\$-** [spec spec]  
 structure that normalizes to an example gate  
**\$=** [skin spec]  
 structure that wraps a face around another structure foo=bar  
**\$?** (list spec)  
 forms a type from a union of other types ?(\$foo \$bar \$baz)

**\$.** [spec (map term spec)]  
structure as read-write core  
**\$;** hoon  
manual structure

---

**% cens put the fun in function**

**%\_** [wing (list (pair wing hoon))]  
resolves a wing with changes, preserving type  
**%.** [hoon hoon]  
calls a gate, inverted  
**%^** [hoon hoon hoon hoon]  
calls a gate with triple sample  
**%+** [hoon hoon hoon]  
calls a gate with a cell sample  
**%-** [hoon hoon]  
calls a gate (fun arg)  
**%:** [hoon (list hoon)]  
calls a gate with many arguments  
**%~** [wing hoon hoon]  
evaluates an arm in a door ~(arm core arg)  
**%\*** [wing hoon (list (pair winghoon))]  
evaluates an expression, then resolves a wing with changes  
**%=** [wing (list (pair wing hoon))]  
resolves a wing with changes foo(x 1, y 2, z 3)

---

**: cols make cells**

**:\_** [hoon hoon]  
constructs a cell, inverted  
**:^** [hoon hoon hoon hoon]  
constructs a cell, 4-tuple [a b c d]  
**:+** [hoon hoon hoon]  
constructs a cell, 3-tuple [a b c]  
**:-** [hoon hoon]  
constructs a cell, 2-tuple [a b], a^b (a^b^c)  
**:~** (list hoon)  
constructs a null-terminated list ~[a b c]  
**:\*** (list hoon)  
constructs an n-tuple [a b c d e ...]  
**::** marks a comment (digraph, not rune)

---

**. dots nock**

**.+** atom  
increments an atom using Nock 4 +(42)  
**.\*** [hoon hoon]  
evaluates using Nock 2  
**.=** [hoon hoon]  
tests for equality using Nock 5 =(a b)  
**.?** hoon  
tests for cell or atom using Nock 3  
**.^** [spec hoon]  
loads from namespace using Nock 12

---

**-/= terminators terminate**

**--** terminates core expression (digraph, not rune)  
**==** terminates running series of Hoon expressions (digraph, not rune)

**^ kets cast**

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

**^.** [hoon hoon]  
typecasts on value

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

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

**^&** hoon  
converts a core to a zinc core (covariant)

**^~** hoon  
folds constant at compile time

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

**^?** hoon  
converts a core to a lead core (bivariant)

**^\*** spec  
bunt, produces default mold value \*foo

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

---

**~ sigs hint**

**~|** [hoon hoon]  
prints in stack trace if failure

**~\$** [term hoon]  
profiler hit counter

**~\_** [hoon hoon]  
prints in stack trace, user-formatted

**~%** [chum hoon tyre hoon]  
registers jet

**~/** [chum hoon]  
registers 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]  
caches a computation

**~&** [@ud hoon hoon]  
prints (used for debugging)

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

**~=** [hoon hoon]  
detects duplicate

**~!** [hoon hoon]  
prints type if compilation failure

---

**; mics make**

**;;** [hoon (list hoon)]  
calls a binary function as an \$n\$-ary function :(fun a b c d)

**;/** hoon  
(Sail) yields tape as XML element

```

;< [spec hoon hoon hoon]
   glues a pipeline together (monadic bind)
;~ [hoon (list hoon)]
   glues a pipeline together with a product-sample adapter (monadic bind)
;; [spec hoon]
   normalizes with a mold, asserting fixpoint
;+
   (Sail) makes a single XML node
;*
   (Sail) makes a list of XML nodes from Hoon expression
;= marl:hoot
   (Sail) makes a list of XML nodes


---


= tises alter
=| [spec hoon]
   combines default type value with the subject
=. [wing hoon hoon]
   changes one leg in the subject
=? [wing hoon hoon hoon]
   changes one leg in the subject conditionally
=^ [skin wing hoon hoon]
   pins the head of a pair; changes a leg with the tail
=: [(list (pair wing hoon)) hoon]
   changes multiple legs in the subject
=/ [skin hoon hoon]
   combines a named noun with the subject
; [skin hoon hoon]
   combines a named noun with the subject, inverted
=< [hoon hoon]
   composes two expressions, inverted
=> [hoon hoon]
   composes two expressions
=- [hoon hoon]
   combines a new noun with the subject
=* [(pair term (unit spec)) hoon hoon]
   defines an alias
=, [hoon hoon]
   exposes namespace (defines a bridge)
=+ [hoon hoon]
   combines a new noun with the subject
=~ (list hoon)
   composes many expressions


---


? wuts test
?| (list hoon)
   logical OR (loobean)
?: [hoon hoon hoon]
   branches on a boolean test
?. [hoon hoon hoon]
   branches on a boolean test, inverted
?< [hoon hoon]
   negative assertion
?> [hoon hoon]
   positive assertion

```

foo:bar

|(foo bar baz)

?- [wing (list (pair spec hoon))]  
 switches against a union, no default  
 ?^ [wing hoon hoon]  
 branches on whether a wing of the subject is a cell  
 ?= [spec wing]  
 tests pattern match  
 ?# [skin wing]  
 tests pattern match  
 ?+ [wing hoon (list (pair spec hoon))]  
 switches against a union, with default  
 ?& (list hoon)  
 logical AND (loobean) &(foo bar baz)  
 ?@ [wing hoon hoon]  
 branches on whether a wing of the subject is an atom  
 ?~ [wing hoon hoon]  
 branches on whether a wing of the subject is null  
 ?! hoon  
 logical NOT (loobean) !foo

---

! **zaps run wild**  
 !:  
 turns on stack trace  
 !.  
 turns off stack trace  
 !, [\*hoon hoon]  
 emits AST of expression (use as !, \*hoon expression)  
 !; [hoon hoon]  
 emits the type for an expression using the type of type  
 !> hoon  
 wraps a noun in its type  
 !< hoon  
 lift dynamic value into static context  
 !@ [(list wing) hoon hoon]  
 != hoon  
 makes the Nock formula for a Hoon expression  
 !? [\$@(@ { @ @}) hoon]  
 restricts Hoon version  
 !! ~  
 crashes

---

/ **fases file (+ford arm of %clay)**  
 /? foo  
 pin a version number  
 /- foo, \*bar, baz=qux  
 imports a file from the sur directory (\* pinned with no face, = with specified face)  
 /+ foo, \*bar, baz=qux  
 imports a file from the lib directory (\* pinned with no face, = with specified face)  
 /= clay-raw /sys/vane/clay  
 imports results of user-specified path wrapped in face  
 /% %mark  
 imports mark definition from mar/  
 /\$ %from %to  
 imports mark conversion gate from mar/

```

/* myfile %hoon /gen/myfile/hoon
   imports the contents of a file in the desk converted to a mark (build-time static data)
/~ face type /path
   imports contents of a directory under face=(map @ta type)

+ luses arm cores
+|
   labels a chapter (produces no arm)
+$ [term spec]
   produces a structure arm (type definition)
++ [term hoon]
   produces a (normal) arm
+* [term term spec]
   produces a type constructor arm

```

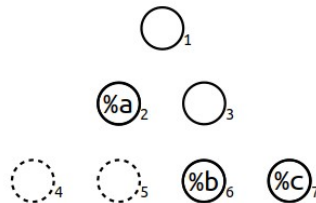
### syntax

```

+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 [%b %c]] %a
+: [%a [%b %c]] [%b %c]
-<: [%a [%b %c]] %ride failed
+<: [%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 (atom)

`^` cell

`?` loobean

`~` null

**@p notation**

|             |  |                                      |
|-------------|--|--------------------------------------|
| <b>@</b>    | Empty aura   |                                      |
| <b>@c</b>   | Unicode codepoint                                      | ~~~45fed.                            |
| <b>@d</b>   | Date   |                                      |
| <b>@da</b>  | Date, absolute   | ~2020.12.25..7.15.0..1ef5            |
| <b>@dr</b>  | Date, relative   | ~d71.h19.m26.s24..9d55               |
| <b>@f</b>   | Loobean (for compiler, not castable)                   | &                                    |
| <b>@i</b>   | Internet address                                       |                                      |
| <b>@if</b>  | IPv4 address   | .195.198.143.90                      |
| <b>@is</b>  | IPv6 address   | .0.0.0.0.0.1c.c3c6.8f5a              |
| <b>@n</b>   | Nil (for compiler, not castable)                       | ~                                    |
| <b>@p</b>   | Phonemic base  | ~laszod-dozser-fosrum-fanbyr         |
| <b>@q</b>   | Phonemic base, unscrambled (used with Urbit HD wallet) | ~laszod-dozser-daltec-hilsyn         |
| <b>@r</b>   | IEEE-754 floating-point number                         |                                      |
| <b>@rh</b>  | Floating-point number, half-precision, 16-bit          | .~~3.14                              |
| <b>@rs</b>  | Floating-point number, single-precision, 32-bit        | .3.141592653589793                   |
| <b>@rd</b>  | Floating-point number, double-precision, 64-bit        | ~3.141592653589793                   |
| <b>@rq</b>  | Floating-point number, quadruple-precision, 128-bit    | ~3.141592653589793                   |
| <b>@s</b>   | Integer, signed (sign bit low)                         |                                      |
| <b>@sb</b>  | Signed binary  | --0b10.0000                          |
| <b>@sd</b>  | Signed decimal   | --1.000                              |
| <b>@sv</b>  | Signed base-32   | --0v201.4gvm1.245kc                  |
| <b>@sw</b>  | Signed base-64   | --0w2.04AfS.G8xqc                    |
| <b>@sx</b>  | Signed hexadecimal                                     | --0x2004.90fd                        |
| <b>@t</b>   | UTF-8 text (cord)                                      | 'urbit'                              |
| <b>@ta</b>  | ASCII text (knot)                                      | ~.urbit                              |
| <b>@tas</b> | ASCII text symbol (term)                               | %urbit                               |
| <b>@u</b>   | Integer, unsigned                                      |                                      |
| <b>@ub</b>  | Unsigned binary  | 0b10.1011                            |
| <b>@uc</b>  | Bitcoin address  | 0c1A1zP1eP5QGefi2DMPTfTL5SLmv7DivfNa |
| <b>@ud</b>  | Unsigned decimal                                       | 8.675.309                            |
| <b>@ui</b>  | Unsigned decimal                                       | 0i123456789                          |
| <b>@uv</b>  | Unsigned base-32                                       | 0v88nvd                              |
| <b>@uw</b>  | Unsigned base-64                                       | 0wx5~J                               |
| <b>@ux</b>  | Unsigned hexadecimal                                   | 0x84.5fed                            |

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

- @ubD** signed single-byte (8-bit) decimal
- @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 eny)

Auras are non-coercive, but conversions may have to go via the empty aura: ^-(@ud ^-(@ 'foo')).

**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                 | +a                                 |                                      |
| +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 (noun at 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                            |