| 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`
produces a trap (a core with one arm)

|:   `[hoon hoon]`
produces a gate with a custom sample

|-   `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

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

$_   `hoon`
structure that normalizes to an example   `_foo`

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

$*   `hoon`
bunt (irregular form is *)

$^   `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 spec]`
structure from filter (requiring)

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

$$ `[spec (map term spec)]`
structure from recursion

$| `[spec hoon]`
        structure with verification
$.   `[spec (map term spec)]`
        structure as read–write core
$+   `[stud spec]`
        standard structure
$;   `hoon`
        manual structure
$/   `[spec (map term spec)]`
        structure as write-only core
**$`**   `[spec (map term spec)]`
        structure as read-only core
$&   `[spec hoon]`
        repaired structure
$!   `[spec (map term spec)]`
        structure as opaque core
  %   **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`
:~   `(list hoon)`
        constructs a null-terminated list   `~[a b c]`
:*   `(list hoon)`
        constructs an n-tuple   `[a b c d e …]`
::
        marks a comment

· **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

^ **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

`^&` `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`
produces example type value

`^:` `spec`
produces a 'factory' gate for a type

~ **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)`
`;<`   `[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
`;+`
       (<u>Sail</u>) makes a single XML node
`;*`
       (<u>Sail</u>) makes a list of XML nodes from Hoon expression
`;=`   `marl:hoot`
       (<u>Sail</u>) makes a list of XML nodes
`;/`   `hoon`
       (<u>Sail</u>) yields tape as XML element
   `=` **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   `foo:bar`
`=>`   `[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
`=+`   `[hoon hoon]`
       combines a new noun with the subject
`=~`   `(list hoon)`
       composes many expressions

? wuts test

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

?: `[hoon hoon hoon]`
branches on a boolean test

?. `[hoon hoon hoon]`
branches on a boolean test, inverted

?< `[hoon hoon]`
negative assertion

?> `[hoon hoo]`
positive assertion

?- `[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

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

!> `hoon`
wraps a noun in its type

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

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

!! `~`
crashes

!< `hoon`
lift dynamic value into static context

/   fases ford
/$ slams a gate on extra arguments
/| takes a series of horns and produces the first one (L-to-R) that succeeds; if none succeed, produces stack traces from arguments
/=   runs a horn (usually produced by another Ford rune), takes the result of that horn, and wraps a face around it
/.   produces a null-terminated list from a sequence of horns, terminated by ==
/,   acts as switch statement, picking a branch to evaluate based on whether the current path matches the path in the switch statement
/&   pass a horn through multiple marks
/_   *unfiltered*: takes a horn, producing new horn mapping supplied horn over list of files in current directory; *filtered*: runs a horn on each file matching aura
/~   produces a horn that evaluates a twig and places the product in the subject
/:   takes a path and a horn, and evaluates the horn with the current path set to the supplied path
/^   takes a mold and a horn, and casts the result of the horn to the mold
/!   produces a mark
/+   accepts a filename and loads that filename from the `lib` directory
/-   accepts a filename and loads that filename from the `sur` directory
//   parses relative path as a hoon twig, and adds the resulting twig to the subject
/;   takes a twig and a horn; the twig should evaluate to a gate, which is then slammed with the result of the horn as its sample
/#   takes a horn and produces a cell of the dependency hash of the result of the horn, and the result itself
/%   forwards extra arguments to enclosed renderers
/?   parses `%zune` version
-/=   terminators terminate
--   terminates core expression
==   terminates running series of Hoon expressions
+   luses change
+|
       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]]              [%a [%b %c]]            .:[%a [%b %c]]   [%a [%b %c]]

+2:[%a [%b %c]]   %a                                                -:[%a [%b %c]]   %a

+3:[%a [%b %c]]   [%b %c]                                           +:[%a [%b %c]]   [%b %c]

+4:[%a [%b %c]]   %ride failed                                      -<:[%a [%b %c]]  %ride failed

+6:[%a [%b %c]]   %b                                                +<:[%a [%b %c]]  %b

+7:[%a [%b %c]]   %c                                                +>:[%a [%b %c]]  %c
```
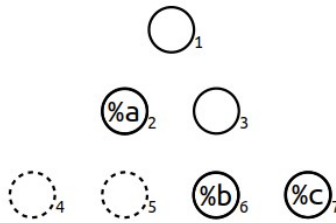
```
        ·  current subject                         ^face  face in outer core
        +  +:.                                      ..arm  core in which ++arm is defined
        -  -:.
       +>  +>:.
```

```
        ~  0 (nil)              eny  entropy                    `a  [~ a]
%.y     &  yes/true             now  current time              ~[a b c]  [a b c ~]
%.n     |  no/false                                            [a b c]~  [[a b c] ~]
```

```
    <[1 2 3]>  renders list as a tape          ?=($hoon %hoon) %.y
    >[1 2 3]<  renders list as a tank          ?=($hoon %loon) %.n
```

```
      -:!>  type spear, use as -:!>(.3.14)
```

@p notation

@c	Unicode codepoints	`~-~45fed.`
@d	Date	
@da	Date, absolute	`~2020.12.25..7.15.0..1ef5`
@dr	Date, relative	`~d71.h19.m26.s24..9d55`
@n	Nil	
@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`
@sw	Signed base-64	`--0w2.04AfS.G8xqc`
@sx	Signed hexadecimal	`--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`
@ud	Unsigned decimal	`8.675.309`
@uv	Unsigned base-32	`0v88nvd`
@uw	Unsigned base-64	`0wx5~J`
@ux	Unsigned hexadecimal	`0x84.5fed`