hoon

pg1 build: cores & molds

	runes	need ==?		
cores	|%		corp	build a generic dry core. any number of arms.
	|.		trap	build a corp with one arm named $
	|-		loop	build a trap and then automatically "kick" it, i.e. compute its $ arm with %=
	|^		cork	build a corp with an auto-fired $ arm (like a loop but with additional non-auto-fired arms)
	|=		gate	build a trap with a sample
	|:		gasp	build a dry gate (same as |=) but with a custom sample
	|_		door	build a corp (of any # of arms) with a sample (so like a gate, but bigger – multiple arms)
	|~		port	build an iron (contravariant) gate
	|?		mine	build a lead (bivariant) trap
	|@		lava	build a generic wet core. any number of arms.
	|*		gill	build a wet gate (a one-armed core with sample)
	|$		ptri	build a gill specifically for producing a mold. the ptri will contain spec code.
	++		slus	add a normal arm
	+$		jib	add a structure-building arm. the arm will contain spec code.
	+*		pseu	add an alias "pseudo-arm". commonly used to give a face to a faceless core: +* this .
	+|		chap	begins a chapter (a group of arms). used for organization and doccords.
	--		p'hep	terminate your list of arms (and thus your core) in a multi-arm core.
pulls	%=	==	pull	on an arm: make changes then fire. on a leg: make changes.
	%_	==	herd	(as in shepherding) pull, staying within a fence
	%*	==	draw	(blacksmithing term) fire and then pull
fires	%:	==	fire	fire a core. any number of children.
	%-		fire (2)	fire a core. 2 children.
	%.		swapfire	fire a core. 2 children, swapped order.
	%+		fire (3)	fire a core. 3 children.
	%^		fire (4)	fire a core. 4 children.
	%~		oven	fire in a door
molds	$_		top hat	(pull a rabbit out) discards its sample and just produces a new noun
	$=		liquid latex	assign a face to nouns that pass through your mold, so that you can reference them.
	$:	==	stem cell	shape a cell mold. cytodifferentiate (what shape of cell is required?) w/ spec within the $:
	$?	==	muffin tin	multi-track OR union for atom molds. for more complex nouns, use $@, $^, and/or $%.
	$@		popsicle	dual-track. shape 2 molds: one for single pops (atoms), one for twin-pops (cells)
	$^		brazen squamate	dual-track. shape 2 molds: one for hydras (cell head), one for serpents (single atom head)
	$%	==	stable	multi-track. shape a list of cell molds with cold atom heads.
	$>		star	require the head of a cell mold to fit a mold.
	$<		strike	require the head of a cell mold to not fit a mold.
	$|		piping tip	takes a mold and further restricts it.
	$&		upgrade	upgrades from an old mold to a new mold.
	$~		bundt pan	take any mold and define a custom default value for it.
	$+		facia	new in 2023. assign a label to the actual mold itself for prettyprinting in the dojo.
	$-		presto chango	input a noun fitting one mold to get the bunt of another as output.
casts	^-		cast	cast to a spec. simply write the spec.
	^=		skin	assign a face to a noun, so that you can reference it.
	^+		fence	cast to "whatever this is." cast by writing a noun and letting the system infer its spec.
	^*		bunt	produce the default value.
	^.		belt	fire a core and cast to the inferred spec of its product.
	^~		rockpile	fold constant at compile time if possible
	^&		zinc-mute	transmute a core to zinc (covariant)
	^|		ferr-mute	transmute a core to iron (contravariant)
	^?		plumb-mute	transmute a core to lead (bivariant)

jots	
wing(subwing1 hoon, subwing2 hoon)	
(|.(7)) (sqt 49) (add 5 2) etc.	
(~(arm door doorsample) armsample)	
_%spam	
my-face=@	
[a=@ b=* c=^ d=%7]	
?(%good %evil)	
`spec`hoon	
skin=hoon	
*spec or *hoon	

pg2

branch: trees & tricks

cells	:* ==	encell	from any number of items, make cells, nesting to the right.	[head tail] head^tail ._head_tail__
	:-	encell (2)	make a single cell – a pair of 2 nouns. order matters. 1st called the head, 2nd tail.	
	:_	swapcell	make a single cell, but writing the two nouns in swapped order.	
	:+	encell (3)	encell 3 nouns	
	:^	encell (4)	encell 4 nouns	
	:~ ==	tack a null	encell any # of elements then add a ~ (not a proper list until cast)	~[head tail]
trees	=>	rappel	compose two hoon expressions	
	=<	prusik	compose two hoon expressions, swapped order	(sub +2 20):[(mul 9 3) .]
	=~ ==	zipline	compose many hoon expressions	
	=+	tack	add a new noun to the subject as the head	
	=-	back	add a new noun to the subject as the tail	
	=/	pin	add be-faced noun to subject as the head	
	=;	fin	add be-faced noun to subject as the tail	
	=|	tap	add a typed bunt to the subject as the head	
	=.	peg	change a leg	
	=: ==	pegs	change multiple legs	
	=^	pin&peg	=/ and =. pin a noun & change leg	
	=?	peg if	change a leg if a statement is true	
	=*	aka	define an alias	
	=,	bridge	define a bridge / expose a namespace	
clay	/=	sign for a parcel	import a file	
	/~	unload a van	import a directory	
	/+	check out a book	import a file from the library directory, /lib	
	/-	make a withdrawal	import a file from the structure-building (think: bucs) directory, /sur	
	/%	memorize an entry	import a mark definition from the mark dictionary, mar/	
	/$	rosetta stone	import mark conversion gate from mar/	
	/*	translation	import a file converted to a mark	
	/?	thermometer	pin a version number	
root logic	.*	boil	run code (core or not) w/ Nock 2	
	.?	cell?	is it a cell? Nock 3	
	.+	increment	increment. Nock 4	
	.=	same?	is it equal? Nock 5	
	.^	scry	just read, no event, not logged. "Nock 12"	
basic logic	?:	if	if,then,else: branch on true or false. do the 2nd child if the 1st is true, the 3rd if false.	
	?.	ifn't	if, swapped order (if,else,then), or swapped polarity (if not, then, else)	
	?>	must be	1st child is true, otherwise crash. (if, then, crash)	
	?<	mustn't be	1st child is false, otherwise crash. (if, crash, else)	
	?| ==	or	if any of its children produce "true", it produces true.	|()
	?& ==	and	only if all of its children produce "true" does it produces true.	&()
	?!	not	the child produces true or false and the ?! will produce the opposite.	!hoon
	?^	a cell?	do the 2nd child if the 1st produces a cell, the 3rd if not.	
	?@	an atom?	do the 2nd child if the 1st produces an atom, the 3rd if not.	
	?~	null?	do the 2nd child if the 1st produces a null, the 3rd if not.	
	?- ==	menu	switch against a union (no default)	
	?+ ==	toppings	switch against a union with a default	
fish	?=	tie a fly	dry fish. fish for whether a noun fits a dry mold.	
	?#	hook a worm	promised rune that will wet fish. fish for whether a noun fits a wet mold.	

pg3

high logic

`;:`	==	daycare	allow a gate that accepts two children to instead accept any number of children :()
`;<`		bucket brigade	glue a pipeline together (monadic bind)
`;~`	==	supply line	glue a pipeline together with a product-sample adapter (monadic bind)
`;;`		stone's cast	a cast, but more forceful

sail

`;+`		dinghy	(Sail) make a single XML node
`;=`	==	navy	(Sail) make a list of XML nodes
`;*`		shipyard	(Sail) make XML nodes from marl-making Hoon
`;/`		bottle	(Sail) turn a tape into an XML element

dojo prints

`~&`		print	print (used for debugging)
`~?`		print?	print conditionally (used for debugging)
`~|`		rats!!	print in stack trace if failure
`~!`		deer!!	print type in stack trace if failure (e.g. what caused this crash? need/have)
`~_`		tale!!	print in stack trace if failure, user-formatted

helpful hints

`~%`		jet	register a jet
`~/`		flight	register a jet with registered context
`~+`		dog-ear	cache a computation
`~>`		how?	raw hint, applied to computation ("forward")
`~<`		what?	raw hint, applied to product ("backward")
`~=`		dup?	detect duplicate
`~$`		hits:	profiler hit counter

wild

`!:`		tron	turn on stack trace
`!.`		troff	turn off stack trace
`!?`		thermostat	restrict Hoon Kelvin version
`!=`		formulate	assemble the Nock formula for a Hoon expression
`!,`		gene sequencer	emit gene (AST) of expression (use as !,(*hoon expression))
`!;`		amphora	raw vase
`!>`		vase	wrap a noun in its type
`!<`		club	de-vase (unsafely), passing through a mold (or crashing). lift the noun out of its vase and turn it back into its normal statically-typed self.
`!@`		wiff	evaluate conditional on existence of wing

stop

`==`		stet	ends those few runes with an indefinite number of children. most runes need no terminator.
`::`		comment	ends code parsing on that line to insert a comment. code parsing resumes on the new line. you can't end a hoon file with a comment.
`!!`		crash	just crash. stop computing and produce no product.