An outline of things that should be in the standard library:

Data structures

- Tuples. These are somewhat “free” from cells, upon choosing an associativity direction (right) for cons.
- Unit/Option/Maybe/\forall T. T + 1

/whatever you call it

- List. This is also somewhat “free” from cells, upon choosing a terminator (0

).

- Lazy streams. (Very slightly) involved to set up, easy to use.
- Sets (balanced as trees)
- Maps (balanced as above, but by key)

Arithmetics

- Unsigned, natural-number arithmetic.
- Both crash-on-underflow and unit-valued subtract.
- The least surprising thing with regard to division (and have rem

, and probably divrem

giving a pair)

- Both crash-on-underflow and unit-valued division (by zero)
- Both crash-on-underflow and unit-valued subtract.
- The least surprising thing with regard to division (and have rem

, and probably divrem

giving a pair)

- Both crash-on-underflow and unit-valued division (by zero)
- Signed, integer arithmetic
- As above, but subtract-underflow is no longer an issue.
- As above, but subtract-underflow is no longer an issue.
- Probably not floating-point.
- Wildly complex out of proportion to its usefulness. Worth keeping in mind as a far-future feature (we might want to perform some light numerical calculus operations)
- Wildly complex out of proportion to its usefulness. Worth keeping in mind as a far-future feature (we might want to perform some light numerical calculus operations)
- Decimals and Rationals
- Wildly useful out of proportion to its complexity; worth having now.
- by “decimals” I mean fixed-point decimals under the usual simple precision rules
- Rationals: p/q, p integers
- positive rationals over naturals? maybe
- positive rationals over naturals? maybe
- Wildly useful out of proportion to its complexity; worth having now.

- by “decimals” I mean fixed-point decimals under the usual simple precision rules
- Rationals: p/q, p integers
- positive rationals over naturals? maybe
- positive rationals over naturals? maybe
- Timestamps
- Absolute and Relative time; this is so you can e.g. add “one hour” to “now”.
- This is actually just a format/pretty-printing convenience over naturals. (We can easily choose time 0 to be older than the universe; e.g. Hoon’s time 0 is midnight, January 1, 292,277,024,401 BC, chosen to place January 1, 2000 in about the middle of the 64-bit time range.)
- Pick exactly one calendar (again, this is just pretty-printing.)
- Absolute and Relative time; this is so you can e.g. add “one hour” to “now”.
- This is actually just a format/pretty-printing convenience over naturals. (We can easily choose time 0 to be older than the universe; e.g. Hoon’s time 0 is midnight, January 1, 292,277,024,401 BC, chosen to place January 1, 2000 in about the middle of the 64-bit time range.)
- Pick exactly one calendar (again, this is just pretty-printing.)
- Binaries, bit and byte fiddling, and so on.
- This is only vaguely arithmetic, but includes things that feel arithmetic like bitwise xor.
- This is only vaguely arithmetic, but includes things that feel arithmetic like bitwise xor.

Library Functions

- Support for all the data structures mentioned above.
- Unit: monadic apply, need, just, &c.
- List: map, filter, &c.
- Lazy stream: as above.
- Set: insert, member?, union, intersect, delete, difference, &c.
- Map: as above, but values add complexity over just keys
- e.g., map union needs to decide what to do with duplicate keys (so map union takes 2 maps and also a function to decide what to do with duplicate keys).
- e.g., map union needs to decide what to do with duplicate keys (so map union takes 2 maps and also a function to decide what to do with duplicate keys).
- Unit: monadic apply, need, just, &c.
- List: map, filter, &c.
- Lazy stream: as above.
- Set: insert, member?, union, intersect, delete, difference, &c.
- Map: as above, but values add complexity over just keys
- e.g., map union needs to decide what to do with duplicate keys (so map union takes 2 maps and also a function to decide what to do with duplicate keys).
- e.g., map union needs to decide what to do with duplicate keys (so map union takes 2 maps and also a function to decide what to do with duplicate keys).
- Cryptography primitives
- Cryptographic hashes
- SHA2: generic cryptography hash, useful everywhere

- SHA2: generic cryptography hash. useful everywhere
- others? poseidon?
- SHA2: generic cryptography hash. useful everywhere
- others? poseidon?
- Ed25519 for signatures, DH
- AES? (as a generic block cipher)
- block cipher modes over it? GCM?
- block cipher modes over it? GCM?
- As I understand it, Poseidon somehow works for encryption as well
- All I know is that someone mentioned it and the Poseidon website has a page on it
- All I know is that someone mentioned it and the Poseidon website has a page on it
- KDFs, perhaps?
- argon, e.g.; deliberately-slow hashes
- argon, e.g.; deliberately-slow hashes
- Simple constructions over these? (e.g. hmac)
- Generic interfaces
- for e.g. “signer”, “hasher”
- for e.g. “signer”, “hasher”
- PRNGs fit in here as well.
- Cryptographic hashes
- SHA2: generic cryptography hash. useful everywhere
- others? poseidon?
- SHA2: generic cryptography hash. useful everywhere
- others? poseidon?
- Ed25519 for signatures, DH
- AES? (as a generic block cipher)
- block cipher modes over it? GCM?
- block cipher modes over it? GCM?
- As I understand it, Poseidon somehow works for encryption as well
- All I know is that someone mentioned it and the Poseidon website has a page on it
- All I know is that someone mentioned it and the Poseidon website has a page on it
- KDFs, perhaps?
- argon, e.g.; deliberately-slow hashes
- argon, e.g.; deliberately-slow hashes
- Simple constructions over these? (e.g. hmac)
- Generic interfaces
- for e.g. “signer”, “hasher”
- for e.g. “signer”, “hasher”

- PRNGs fit in here as well.
- Low-level Nock support infrastructure.
- Mug (fast hash-table style hash, used for equality)
- also includes set/map balancing hashes
- also includes set/map balancing hashes
- Jam, cue
- Type serialization?
- This depends on future work to know what they look like.
- This depends on future work to know what they look like.
- Debugging affordances (to the extent they’re in stdlib and not keyword magic)
- Reflection, e.g., self-virtualization
- It’s easy to write Nock in Nock and proceed to jet it with the original Nock interpreter itself.
- It’s easy to write Nock in Nock and proceed to jet it with the original Nock interpreter itself.
- Mug (fast hash-table style hash, used for equality)
- also includes set/map balancing hashes
- also includes set/map balancing hashes
- Jam, cue
- Type serialization?
- This depends on future work to know what they look like.
- This depends on future work to know what they look like.
- Debugging affordances (to the extent they’re in stdlib and not keyword magic)
- Reflection, e.g., self-virtualization
- It’s easy to write Nock in Nock and proceed to jet it with the original Nock interpreter itself.
- It’s easy to write Nock in Nock and proceed to jet it with the original Nock interpreter itself.
- Binary processing
- This means e.g. strings, and other binary data.
- Can we compile elixir binary patterns to this in our macros? I believe yes.
- This means e.g. strings, and other binary data.
- Can we compile elixir binary patterns to this in our macros? I believe yes.
- Core conveniences
- Higher-order functions
- curry, compose, &c.
- curry, compose, &c.
- Core reflection
- may require runtime types
- may require runtime types
- Higher-order functions

- curry, compose, &c.
- curry, compose, &c.
- Core reflection
- may require runtime types
- may require runtime types