## Syntax Reference

### Program Structure

An Eve program

```plaintext
# # An Eve program

The following is a block of Eve code.
...
  search // search a database
  ...
  bind // modify a database
  ...
```

Eve programs are documents with blocks of code interspersed. The prose of the document is CommonMark compatible, with blocks of Eve code contained in code fences. In every block of Eve code you search for data in a database, and change data based on what you found.

### Record

```plaintext
// addresses of people who are 30 years old
people = [tag: "person" age: 30 address]
```

The predominant data structure in Eve is a record. In every block, you search for records by supplying a pattern of attributes and values. All records matching that pattern are returned.

```plaintext
person.age = 30
[#person brother: [name: "Ryan"]]
```

You can access attributes with dot notation.

```plaintext
// These are equivalent
[#person]
[tag: "person"]
```

Records can be nested to find more complex patterns.

### Actions

```plaintext
search
  people = [#person age: 30 address]
```

Eve has three actions: search, bind, and commit. Search tells Eve to find records in a database. Bind and commit only execute when all records are found.

```plaintext
commit
  [#Chris age: 30]
```

Commit tells Eve to persist the subsequent records, even if their supporting data are removed.

```plaintext
search
  [#time hours]
bind @browser
  [#div text: "It is {{hours}} o'clock"]
```

Bind tells Eve to update subsequent records as their supporting data change. This is how Eve reacts to changes in data.

### Not

```plaintext
// people who are not employees
person = [#person]
not(person = [#employee])
```

You can check for the absence of conditions using not. In this case, we're specifying that the person is not also tagged employee.

### Equivalence

```plaintext
// Pairs of people with the same age, because age is used in both records
person = [#person age]
person2 = [#person age]
```

Eve doesn't have assignment, only equivalence. Records can be joined by using an attribute in two different records.
//Something that's never true
x = 10
x = 100

This will always fail. x is not first 10 and then 100. Instead this says that 10 = 100, which will never be true.

// People older than 30
[#person age > 30]
// The same as above
[#person age]
age > 30
// Also the same as above
people = [#person]
people.age > 30

Three ways to filter attributes.
- Filter an attribute directly in a record.
- Filter an attribute outside of a record.
- Use dot notation to access an attribute on a record.

### If-Then

```
guest = if p = [#person] then p
   if [#person spouse] then spouse
```

if allows you to do conditional equivalence. Here we're stating that guest is equivalent to all the people and the spouses of those people.

The second example uses else to make the options exclusive (only the first matching clause will be taken) and does multiple returns.

### Functions and Aggregates

// The sin function being used with degree input
x = sin[degrees: data]

// The sin function being used with radian input
x = sin[radians: data * \( \pi / 180 \)]

Functions take a set and return a set. They operate element-wise on their input, akin to the `map()` function in other languages. Arguments are explicitly defined when the function is called, so they can be written in any order.

```
total-employees = count[given: employees]
department-budgets = sum[given: salary, per: department]
```

Aggregates are functions that collapse a set to a single value. Examples include sum, count, or max. Aggregates are akin to the `reduce()` or `fold()` function in other languages.

### Update Operators

```
search
cris = [#Chris]

commit
  cris.age := 30
  cris.favorite-food += "pizza"
  cris.favorite-color -= "blue"
  cris <- [eye-color: "green", hair-color: "brown"]
  cris := none
```

Eve has four operators that update records: add, set, remove, and merge.
- Add (+=) - adds value to attribute
- Set (=) - sets the value of attribute
- Remove (-=) - removes value from attribute
- Merge (<-) - merges one record into another

Using the set operator with the `none` keyword removes the record from the database entirely.

### Databases

// Actions can be performed on any number of databases
search @db
bind @db1, @db2

```
search @db1
  [#data data-sources]
bind data-sources
  [#new-record]
```

Databases contain facts. You can perform actions on one or more databases. If no database is specified, the action is performed on a default local database.

Databases are first-class citizens that can be used like any other value. You can apply actions to databases specified by values.