

What is a seq?

Anything that can be viewed as a list, regardless of its actual implementation.

What makes a collection seq-able?

Supporting `first`, `rest`, `cons`, as described in
`clojure.lang.ISeq`.

How can you get an actual seq from a seqable?

How do you move through a seq?

```
(seq coll)
```

```
(next aseq) ;; aka (seq (rest aseq))
```

Functions on collections result in ...

... seqs.

Where are custom seqs defined for most seq-ables?

In an inner class of the collection. This leads to mangled names like:

```
clojure.lang.SomeCollection$Seq
```

How can maps be viewed as seqs?

The key/value pairs are the elements.

How can you get a reliable order of traversal
for maps and sets?

Use sorted sets and sorted maps.

```
(sorted-set & elements)
```

```
(sorted-map & elements)
```

Describe `conj` and `into`.

They are like `cons` and `concat`, but they add the elements in the position most efficient for the underlying representation.

```
(conj coll element & elements)  
(into to-coll from-coll)
```


All Clojure sequences are ... and most are ...

... immutable ... lazy.

What's an easy way to insert commas
between words?

```
(interpose separator coll)
user=> (apply str (interpose ", " ["a" "b" "c"]))
"a, b, c"
```

clojure.contrib.str-util's `str-join` wraps the common
`apply str` **pattern**.

What's the difference between `set/vec` and `hash-set/vector`?

`vec` and `set` take a single collection argument, `hash-set` and `vector` take variable elements.

What are the common sequence factory methods that take a variable number of elements as arguments?

list, vector, hash-set, hash-map.

What are the common sequence filtering functions?

```
(take-while pred coll)
(drop-while pred coll)
(split-at index coll)
(split-with pred coll)
```

How can you create a new predicate that reverses the truth conditions of an existing one?

(complement pred)

What are the common sequence predicates?

```
(every? pred coll)  
(some pred coll)  
(not-every? pred coll)  
(not-any? pred coll)
```

What are the common sequence transforming functions?

```
(map f coll)
(reduce f val? coll)
(sort comp? coll)
(sort-by keyfn comp? coll) ; keyfn is for *getting* the keys
```


Give the signature for seq comprehensions.

How are they implemented?

```
(for [binding-form coll-expr filter-expr? ...] expr)
```

There can be many binding-form/coll-expr pairs in a row.

Seq comprehensions are a macro.

Which filter expressions are available for seq comprehensions?

- `:when expr` which is much like a standard filter/guard.
- `:while expr` which stops the comprehension as soon as the predicate fails.

Force the evaluation of a lazy sequence.

(doall coll) ; stores result of traversal and returns
(dorun coll) ; doesn't store, returns nil

What can be treated as a seq, Clojure collections aside?

- Java collections, arrays, and strings
- Regexp
- File hierarchies
- Streams
- XML trees
- Database results

Which structure-specific functions do lists support?

```
(peek coll)
```

```
(peek coll)
```

`peek` returns the first element, `pop` is like `rest` but throws an exception on an empty sequence.

Which structure-specific functions do vectors support?

`peek` which returns the last element, and `pop` which returns the "init" of a vector.

`(get vector index)`

`(vector index)`

`(subvec vector start end?)`

What's the difference between `take/drop`
and `subvec`?

`take/drop` work on any sequence, but `subvec` is much faster for vectors.

Which structure-specific functions do maps support for querying contents?

(keys map)

(vals map)

(get map key not-found?)

(a-map element) ; test for membership

(a-keyword map) ; test for membership

What's the trouble with looking up a key in a map?

How can you avoid this problem?

You can't know if a result of `nil` indicates the key was not in the map or if it was present, mapped to `nil`.

You can get around the problem with:

```
(contains? map key)
```

```
(get map key not-found?)
```

What structure-specific functions do maps support for creating new maps?

```
(assoc map key val & more-kvs)
(dissoc map key & more-keys)
(select-keys map key-seq)
(merge map1 map2) ; map2 wins if both keys exist
(merge-with merge-fn & maps)
```

Unlike other structure-specific functions, those
for sets must be ...

... imported to be used unqualified.

```
(use 'clojure.set)
```

Which set-theoretic operations do sets support?

unions, intersection, difference.

How can database results be viewed as seq-able.

There are two important correspondences between relational algebra, databases, and the clojure type system.

relation = table = set-like

tuple = row = map-like

So in clojure you might have a set of maps (i.e., a relation).

Which database-like functions does Clojure support?

Straight sets:

`(select pred set)`

Relations (sets of maps):

`(rename relation rename-map)`

`(project relation keys)`

`(join rel1 rel2 keymap?)`