Core Transformation Functions

Ken Q Pu, Faculty of Science, Ontario Tech University

Comprehensive Guide

https://clojure.org/api/cheatsheet

Ken Q Pu, Faculty of Science, Ontario Tech University

Collections

```
(count <sequence>)
⇒ number
```

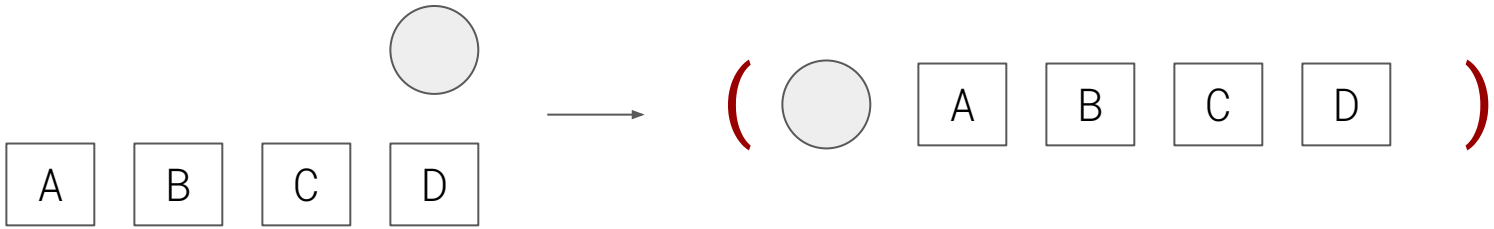
Clojure supports infinite streams as sequences, so make sure that <sequence> is finite to ensure termination.

"Adding" to sequence

Ken Q Pu, Faculty of Science, Ontario Tech University

Collections

```
(cons <element> <collection>)
⇒ <collection>
```

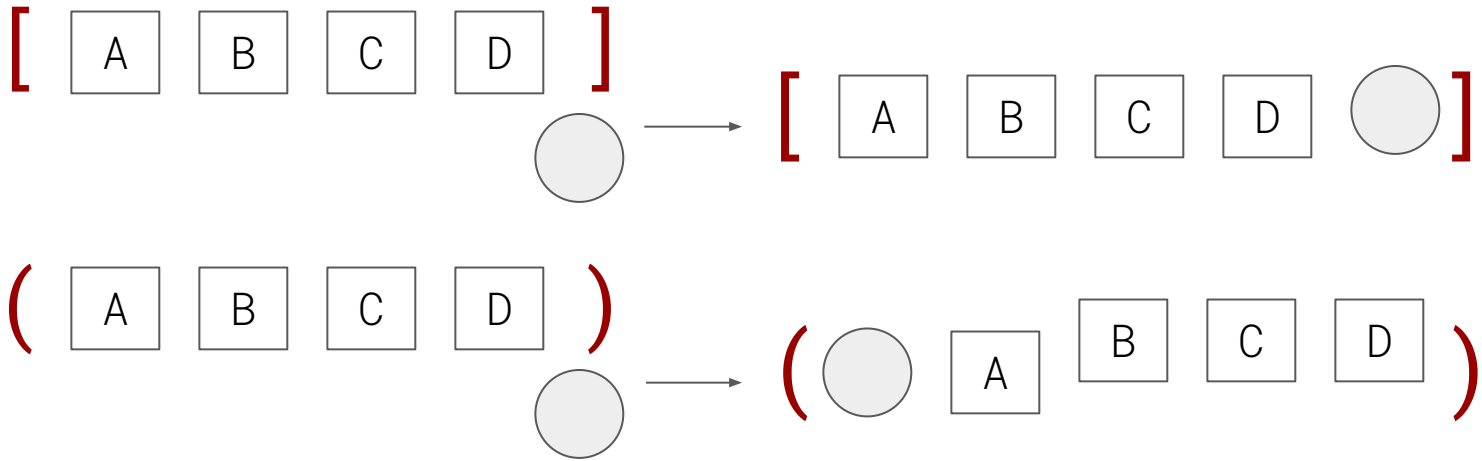


Always returns a list

Collections

```
(conj <collection> <element>)
⇒ <collection>
```

Add the <element> into <collection> in the most efficient way.



Ken Q Pu, Faculty of Science, Ontario Tech University

"Deleting" from sequence

Ken Q Pu, Faculty of Science, Ontario Tech University

Collections

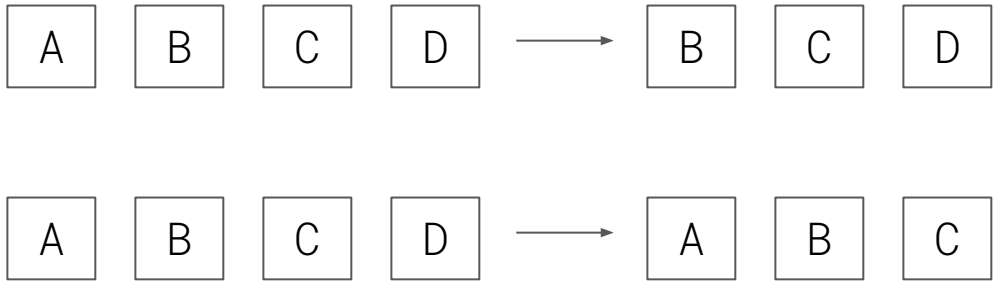
```
(rest <collection>)
⇒ <collection>
```

Collections

```
(pop <collection>)
⇒ <collection>
```

Removes an element in the most efficient way possible.

For vectors, remove the last element.
For lists, remove the first element.



"Adding" to hashmap

Ken Q Pu, Faculty of Science, Ontario Tech University

Hashmaps

```
(assoc <hashmap> <key> <value> <key> <value>)
⇒ <hashmap>
```

```
(assoc {:name "Ken" :course "CSCI 3055U"}
  :name "Ken Pu"
  :role "instructor")

=>

{:name   "Ken Pu"
 :course "CSCI 3055U"
 :role   "instructor"}
```

Inserts key value pairs into the input hashmap. If the key already exists, the previous value is overwritten by the new value.

Hashmaps

(assoc-in <*hashmap*> [k_1 k_2 ...] <value>)
⇒ <*hashmap*>

```
(def person {:name "Albert Einstein"
             :address {:city "Princeton"
                       :state "New Jersey"}})

(assoc-in person
 [:address :country] "United States")

=>
{:name "Albert Einstein"
 :address {:city "Princeton"
           :state "New Jersey"
           :country "United State"}}
```

Hashmaps

(assoc-in <*hashmap*> [k_1 k_2 ...] <value>)
⇒ <*hashmap*>

```
(def person {:name "Albert Einstein"
             :address {:city "Princeton"
                       :state "New Jersey"}})

(assoc-in person
 [:address :country] "United States")

(assoc person
 :address (assoc (person :address)
            :country "United States"))
```

assoc-in is not strictly required as it can be replaced by nested assoc forms.

But the latter is much more verbose.

"Delete" from hashmap

Ken Q Pu, Faculty of Science, Ontario Tech University

Hashmaps

```
(dissoc <hashmap> <key1> <key2> ...)
⇒ <hashmap>
```

Removes one or more keys from the input hashmap

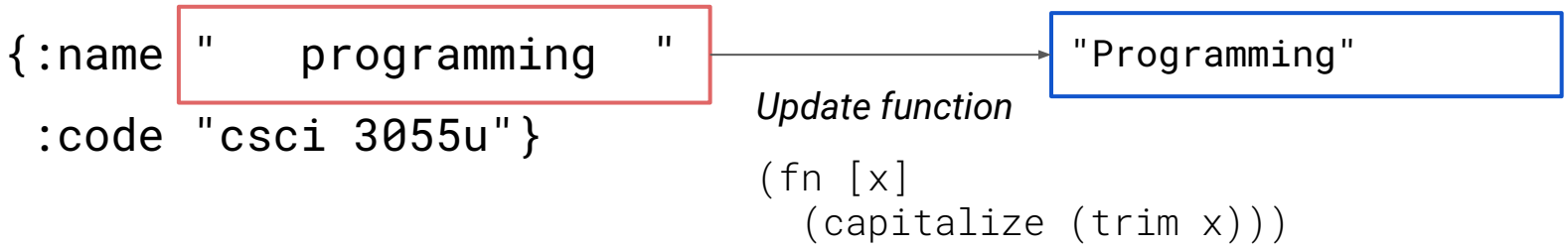
"Update" hashmap

Ken Q Pu, Faculty of Science, Ontario Tech University

Hashmaps

```
(update <hashmap> <key> <fn> <args...>)
⇒ <hashmap>
```

update is a functional way of changing a value in a hashmap.




```
(update course-info
 :name (fn [x] (capitalize (trim x))))
```

Hashmaps

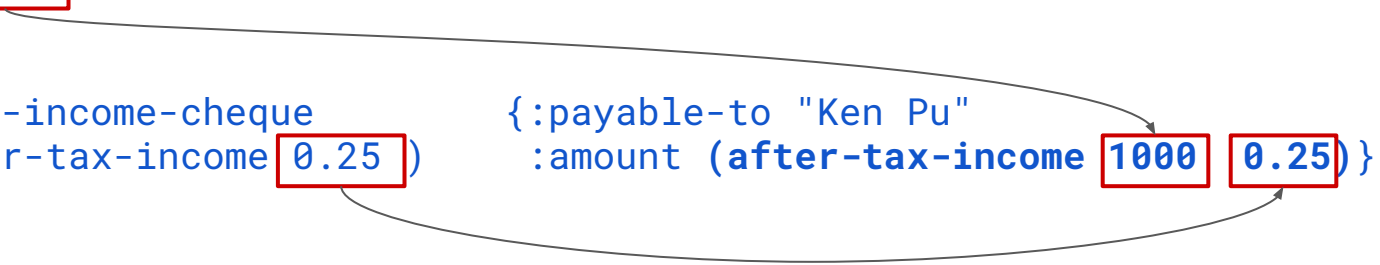
```
(update <hashmap> <key> <fn> <args...>)
⇒ <hashmap>
```

```
(def before-income-cheque
  {:payable-to "Ken Pu"
   :amount 1000 })

(update before-income-cheque
  :amount after-tax-income 0.25 )
```

```
(def after-tax-income [income tax-rate]
  (- income (* income tax-rate)))

{:payable-to "Ken Pu"
 :amount (after-tax-income 1000 0.25)}
```



Ken Q Pu, Faculty of Science, Ontario Tech University

Hashmaps

```
(update-in <hashmap> [k1 k2 ...] <fn> <args...>)
⇒ <hashmap>
```

This is the natural generalization to update nested hashmaps.

Vectors

Vector is

- an iterable sequence like lists
- an indexed structure like hashmaps

```
(conj <vector> <element>)

(assoc <vector> <index> <element>)

(update <vector> <index> <update-fn>)
```

Example of hybrid structure

```
(def course-outline
 {:name "CSCI 3055U"
  :topics [ {:name "Lambda Calculus"
             :weeks 2}
            {:name "Clojure"
             :weeks 5}
            {:name "Kotlin"
             :weeks 3} ]})
```

We will introduce threading forms to build data processing pipelines easily.

```
(update-in course-outline
  [:topics 1 :weeks]
  dec)
```

```
(update-in course-outline
  [:topics 2 :weeks]
  inc)
```

```
(let [a (update-in course-outline
          [:topics 1 :weeks]
          dec)]
 (update-in a
  [:topics 2 :weeks]
  inc))
```