{:title “Basics of Clojure - Mapping (Part 2)” :layout :post :date “2016-09-17” :tags [“Clojure” “Guide” “Code”]}

Link to Previous Guide: Basics of Clojure (Part 1)

Today we’ll explore how to do some mapping in clojure.

Lets start off with writing a simple function that will add one to whatever argument given.

```
(defn add [x] (+ x 1)) ;; function that adds one to the argument given.
```

Now lets make something called a vector which will contain numbers that we want to manipulate later on.

```
(def numbers [1 2 3]) ;; this is how you create a persistant vector.

(type numbers) ;; shows you that the numbers is of type persistantvector.
```

Now we can map the function add across the vector to each number using map like so.

```
(map add numbers) ;; This will essentially add 1 to each number in the vector.

;; (2 3 4)
```

Notice how the end result is a LazySeq and not a vector. I’m not going to get into why that happens but you can do some research into the reason why.

```
(type (map add numbers)) ;; This is of type LazySeq.

(vec (map add numbers)) ;; turns the LazySeq into a vector if it really bothers you.

(type (vec (map add numebrs))) ;; Shows that the result from using vec on the lazyseq retur
```

You could also do it like the following to achieve the same result as previous.

```
(map (fn [x] (+ x 1)) numbers) ;; this time you’re just using an anonymous function.
```

Now lets say we want the second power of all numbers from 1 to 100? How would we go about such as task? Well lets start off with a function that raises an argument to the second power.

```
(defn powertwo [x] (* x x)) ;; raises it to the second power since its multiplying by itsel

(powertwo 7) ;; should give you 49 to see that it works.
```

Now we don't wanna write out 1 to 100 so we'll just create a lazyseq for it by the following using range.

```
(range 101) ;; it gives us all the numbers 0 to 100. However we don't really want 0 so we ca

(rest (range 101)) ;; this gives us a list so add vec to it.

(vec (rest (range 101))) ;; there we go.

(def numbers (rest (range 101))) ;; now we have that binded to numbers.
```

Now we can use map to use our powertwo function over all then numbers.

```
(map powertwo numbers) ;; now all the numbers are raised to the second power.
```

Thats about it and finally as a bonus I'll show you another function reduce. That will give you the sum of all the numbers in the vector!

```
(reduce + (map powertwo numbers)) ;; BONUS
```