Introduction to Clojure

Guide to baking for the intrepid robot

PurelyFunctional.tv

Baking Commands

Name of ingredient; you can also grab the :cup.

`(grab :egg)`	Pick something up.
`(squeeze)`	Squeeze whatever you are holding.
`(release)`	Release what you are holding.
`(scoop :flour)`	Fill cup in hand with an ingredient.
`(add-to-bowl)`	Add what you are holding to the bowl.
`(mix)`	Mix the ingredients in the bowl.
`(pour-into-pan)`	Pour the bowl into the pan.
`(bake-pan 30)`	Put the pan in the oven.
`(cool-pan)`	Put the pan on the cooling racks.
	Returns the id of the cooling rack.

Number of minutes to bake.

Fetching commands

Name of location to go to.

`(go-to :pantry)`	Change locations in the bakery.
`(load-up :egg)`	Pick up an ingredient as cargo.
`(unload :egg)`	Put down an ingredient from cargo.

Name of ingredient.

Filling orders

`(get-morning-orders)`	Get a list of orders.
`(delivery receipt)`	Give delivery bot a receipt for baked goods.

Receipt contains order id, cooling rack ids, and delivery address.

General commands

`(status)`	Print out the status of the bakery.
`(start-over)`	Reset the state of the simulator.
`(bakery-help)`	Print out a list of commands.

Ingredients

`:egg`	squeezed	`:fridge`
`:butter`	simple	`:fridge`
`:milk`	scooped	`:fridge`
`:flour`	scooped	`:pantry`
`:cocoa`	scooped	`:pantry`
`:sugar`	scooped	`:pantry`

Locations

`:fridge`	`:egg, :milk, :butter`
`:pantry`	`:flour, :cocoa, :sugar`
`:prep-area`	where baking is done

Simple ingredients

Simple ingredients are simply grabbed then added to the bowl.

```
(grab :butter)
(add-to-bowl)
```

Squeezed ingredients

Squeezed ingredients must be squeezed after grabbing them, then added to the bowl.

```
(grab :egg)
(squeeze)
(add-to-bowl)
```

Scooped ingredients

Scooped ingredients use the cup. Grab the cup, scoop, then add to bowl. Be sure to release the cup after you're done.

```
(grab :cup)
(scoop :flour)
(add-to-bowl)
(release)
```

Cake recipe

2 cups flour
2 eggs
1 cup milk
1 cup sugar

1. mix all ingredients
2. bake in pan for 25 minutes
3. let cool

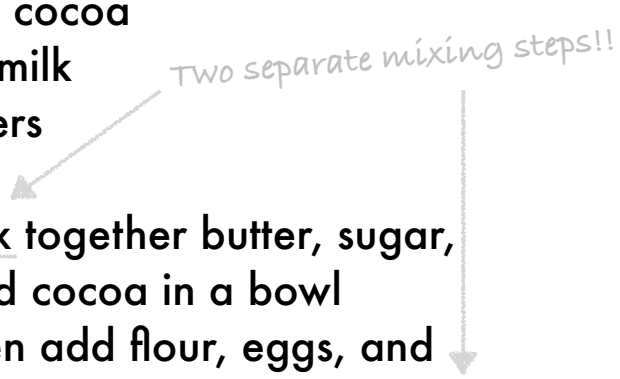
Cookie recipe

1 egg
1 cup flour
1 cup sugar
1 stick butter

1. mix in bowl
2. bake in pan 30 minutes
3. let cool

Brownie recipe

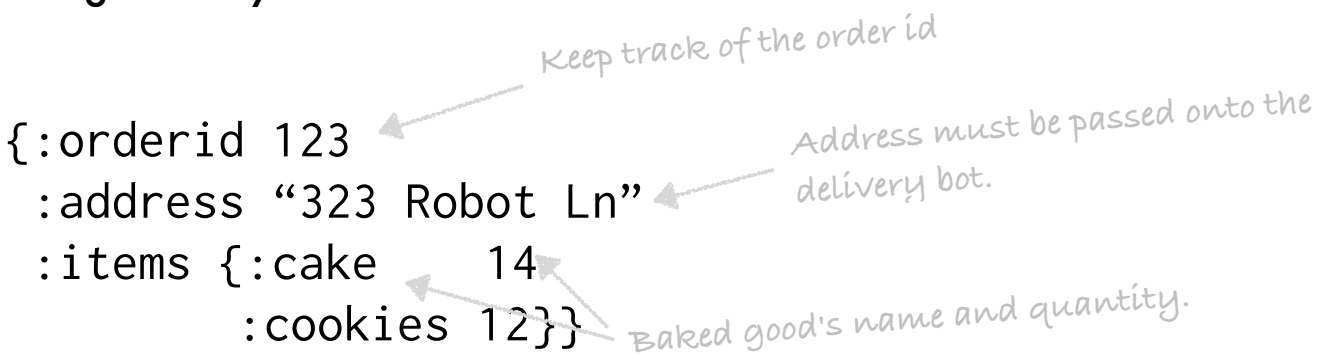
2 cups flour
2 eggs
1 cup sugar
2 cups cocoa
1 cup milk
2 butters



1. mix together butter, sugar, and cocoa in a bowl
2. then add flour, eggs, and milk to same bowl and mix
3. bake in pan for 35 minutes
4. let cool

Orders

Orders come in in the morning. You can get the morning orders with `(get-morning-orders)` or `(get-morning-orders-day3)`. They are given to you as a list. Each order looks like this:

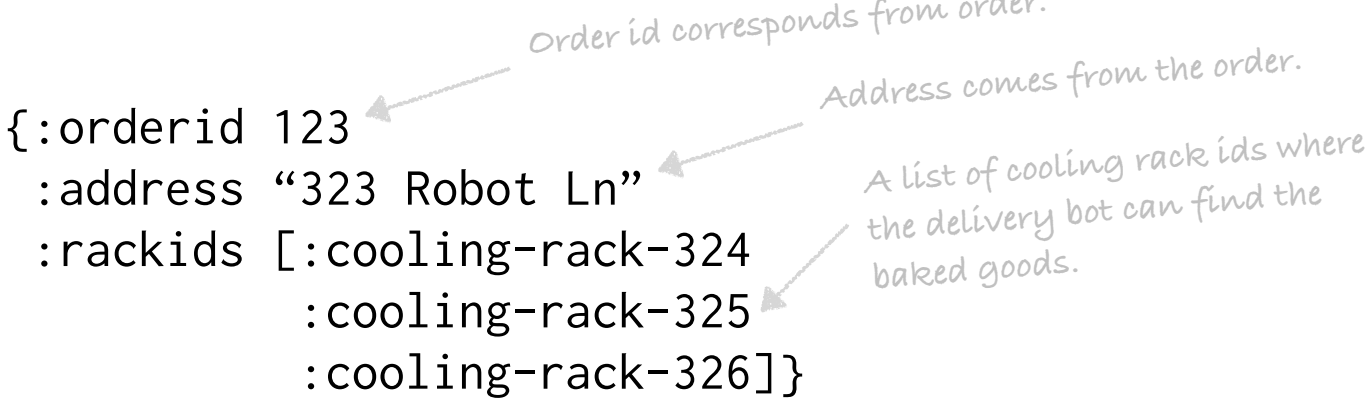



```
{:orderid 123
 :address “323 Robot Ln”
 :items {:cake    14
         :cookies 12}}
```

As a baker, you are responsible for managing the orders you receive. That means fetching the required ingredients, baking all of the goods, and giving a receipt to the delivery bot.

Receipts

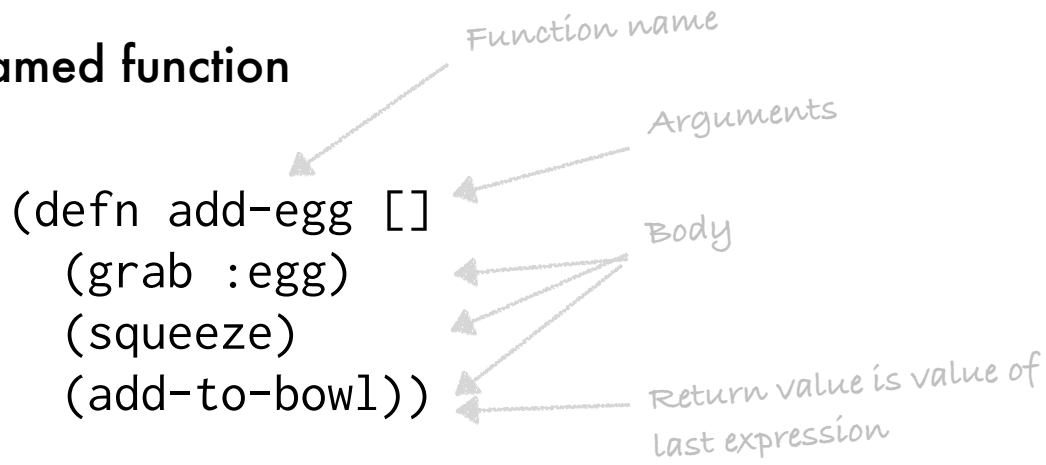
Receipts should be created for each order and given to the delivery bot for prompt delivery. Please ensure that you create one receipt for each order. It confuses the delivery bot to have multiple receipts per order.



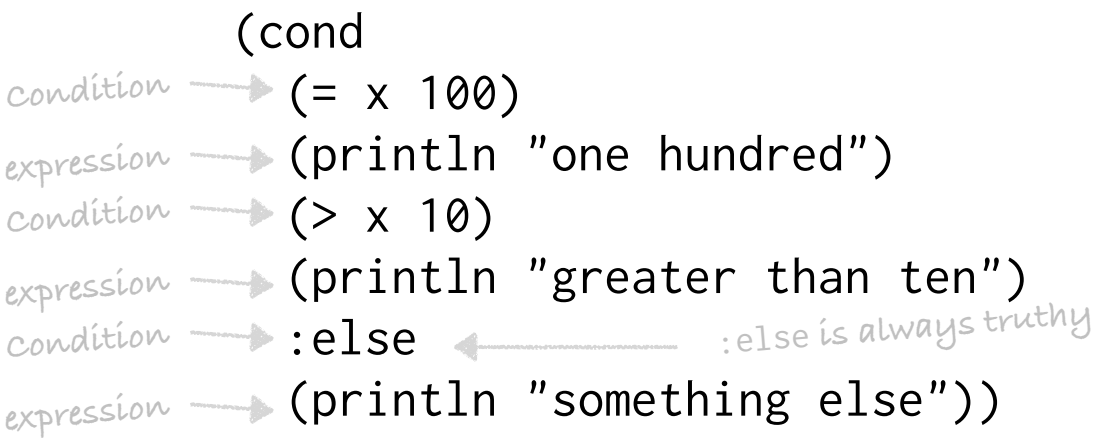

```
{:orderid 123
 :address “323 Robot Ln”
 :rackids [:cooling-rack-324
           :cooling-rack-325
           :cooling-rack-326]}
```

Clojure guide

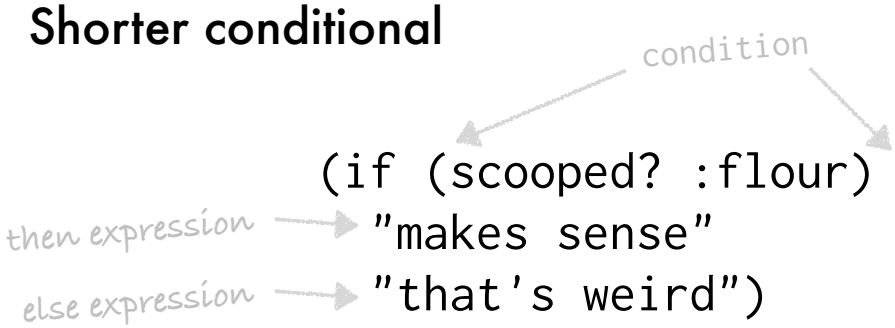
Defining a named function



Conditional



Shorter conditional



Do things in sequence

```
(do
  (println "first step")
  (println "second step"))
```

Body

Do things a certain number of times

Name of iteration count

Number of times to iterate

```
(dotimes [i 10]
  (println i))
```

Body

One function, different number of arguments

Function name

```
(defn my-function
  ([]
    "zero-argument version")
  ([a]
    "one-argument version")
  ([a b]
    "two-argument version")
  ([a b c d]
    "four-argument version"))
```

Arguments

Body

Arguments

Body

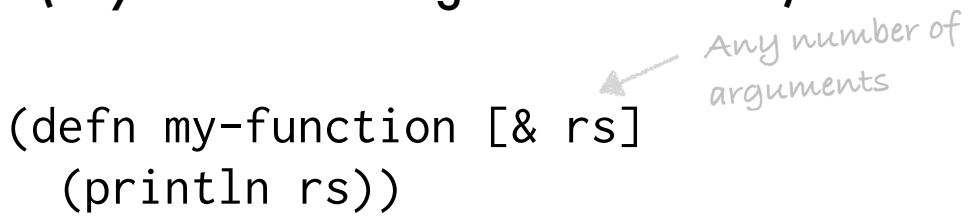
Arguments

Body

Arguments

Body

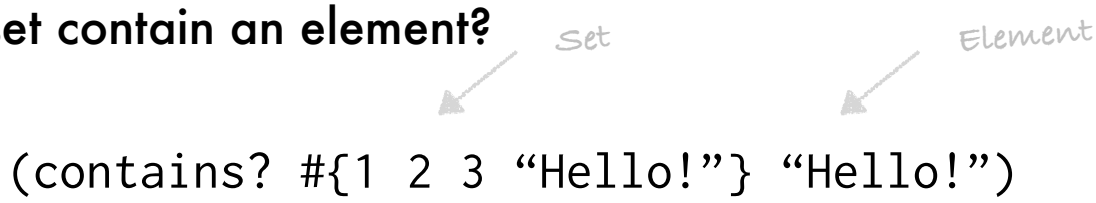
Rest argument (any number of arguments in a list)



Apply a function to a list of arguments

Sets

Does a set contain an element?



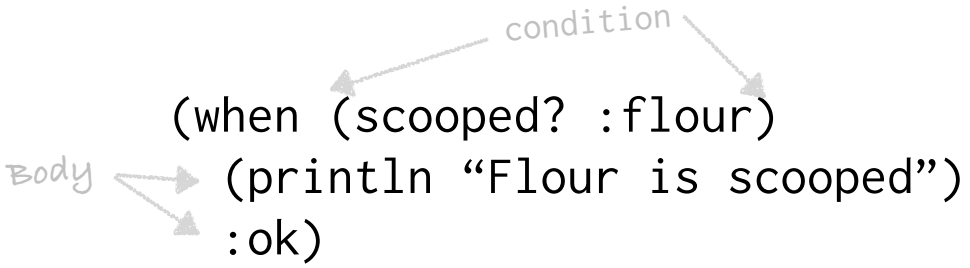
Define a global variable

Maps (hash maps)

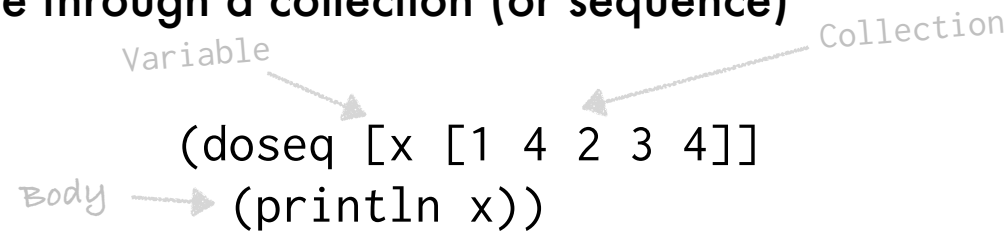
Get the value given a key

Check if a map contains a given key

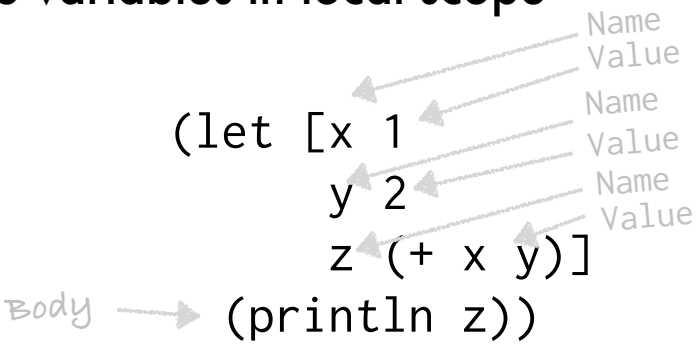
An even shorter conditional, often used for effects



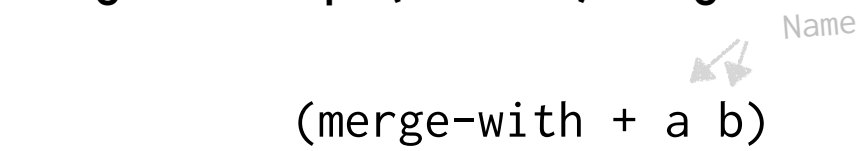
Iterate through a collection (or sequence)



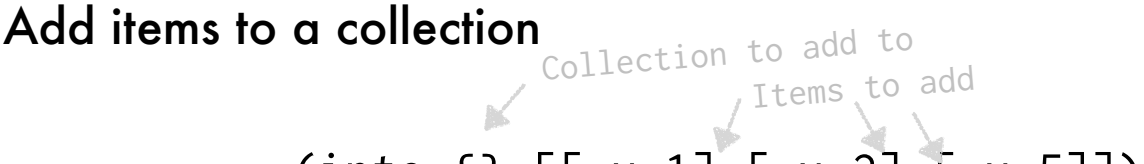
Define variables in local scope



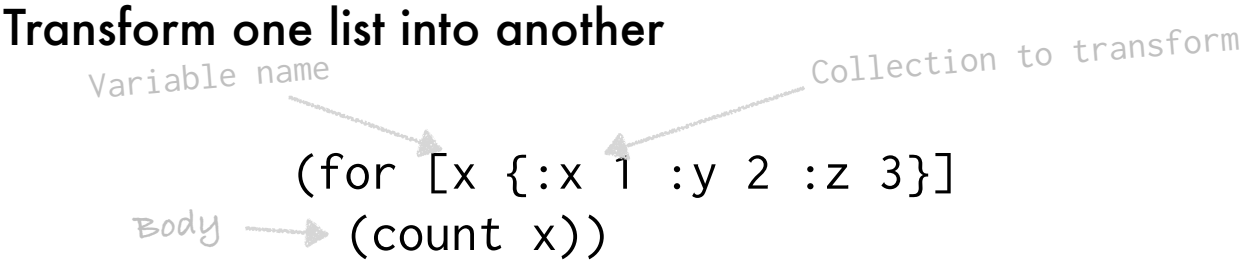
Merge two maps (a and b) using a combining function



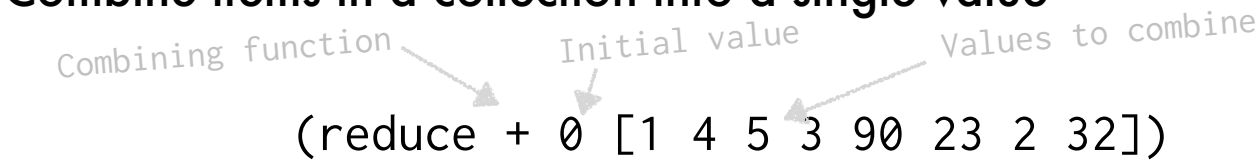
Add items to a collection



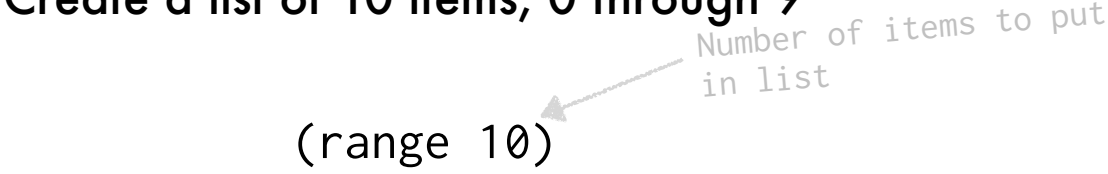
Transform one list into another



Combine items in a collection into a single value

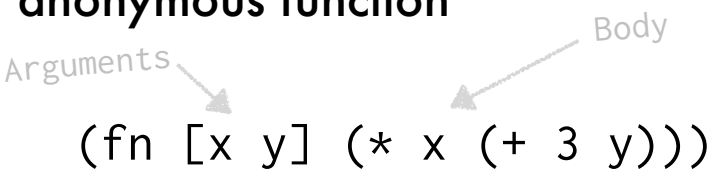


Create a list of 10 items, 0 through 9



Group elements into a hash map based on return values

Create an anonymous function



Leiningen commands

Leiningen is the most popular Clojure project management tool. These commands should be run at the terminal (command prompt).

Start a REPL

```
lein repl
```

Run the `-main` function

```
lein run
```

Get a list of Leiningen directives

```
lein help
```