CS 596 Functional Programming and Design
Fall Semester, 2014
Doc 19 File I/O, Seesaw
Nov 13, 2014

Lazy Sequences

```
(def fibonacci-seq
   (concat [1 2] (lazy-cat (map + (drop 1 fibonacci-seq) fibonacci-seq ))))

(defn sdsu-fibonacci-even
  [n]
   (if (<= n 1)
     0
     (reduce + (filter even? (filter #(<= % n) fibonacci-seq)))))
```

(sdsu-fibonacci-even 2)	Exception
(sdsu-fibonacci-even 2)	2

Lazy Sequences

cache the sequence when evaluated

Only cache up to the exception

File I/O

I/O

	to *out*	to String
For Reader	pr	pr-str
with newline	prn	prn-str
For Humans	print	print-str
with newline	println	println-str

```
(pr-str [:a :b :c] {:a 1})
                    "[:a :b :c] {:a 1}"
```

	from *in*	From string
	read	read-string
	read-line	

read-string with evaluate code

```
(read-string "#=(eval (def x 3))")                    #'user/x
x                                                              3
```

Turning off eval in read

```
(binding [*read-eval* false]
  (read-string "#=(eval (def y 3))"))
```

Exception

Writing & Reading Clojure Data

```
(spit "data.clj" (pr-str [:a :b :c]))
(read-string (slurp "data.clj"))                [:a :b :c]
```

```
(binding [*read-eval* false]
  (read-string (slurp "data.clj")))
```


From: Luke VanderHart and Ryan Neufeld. “Clojure Cookbook”

read-string

reads one object from string

(read-string "[1 2 3] {:a 2}") [1 2 3]

More Efficient Read/Write Clojure Data

```
(with-open [w (clojure.java.io/writer "data.clj")]
  (binding [*out* w]
    (pr large-data-structure)))
```

```
(with-open [r (java.io.PushbackReader. (clojure.java.io/reader "data.clj"))]
  (binding [*read-eval* false]
    (read r)))
```

with-open

flushes output stream
closes the bindings

```
(with-open [w (clojure.java.io/writer "data.clj")]
  (binding [*out* w]
    (pr large-data-structure)))
```

Alternative

```
(with-open [r (java.io.PushbackReader. (clojure.java.io/reader "data.clj"))]
  (binding [*read-eval* false]
    (read r)))
```

```
(with-open [r (java.io.PushbackReader. (clojure.java.io/reader "data.clj"))]
  (binding [*in* r
            *read-eval* false]
    (read)))
```

Appending

```
(with-open [w (clojure.java.io/writer "data.clj" :append true)]
  (binding [*out* w]
    (pr [1 2 3 ])))
```

Where do the files go

(spit "data.clj" (pr-str [:a :b :c]))

If using lein project

Project directory

If insta-repl

Don’t know

Dealing with files & Directories

Use me.raynes/fs library

:dependencies [me.raynes/fs "1.4.4"]

(:require [me.raynes.fs :as fs])

cwd
chdir
chmod
copy
copy-dir
create
delete
exists?
home
iterate-dir
list-dir
mkdir
parent
rename
size
temp-dir
temp-file

More Seesaw

Containers -Showing More that One Widget

top-bottom-split
left-right-split
border-panel
flow-panel
grid-panel
horizontal-panel
tabbed-panel
vertical-panel

Used in examples

```
(defn display
  [content width height]
  (let [window (seesaw/frame :title "Example"
                             :content content
                             :width width
                             :height height)]
    (seesaw/show! window)))
```

Tabbed-panel

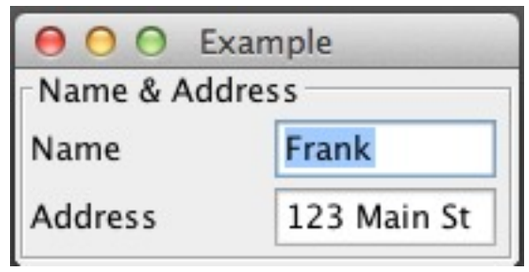
```
(def tabs (seesaw/tabbed-panel
           :placement :top
           :tabs [{:title "A"
                   :content "This is A's Content"}
                  {:title "B"
                   :tip "To be or not to b"
                   :content "This is B's Content"}]))

(display tabs 200 400)
```

Thursday, November 13, 14

display is from previous slide

grid-panel

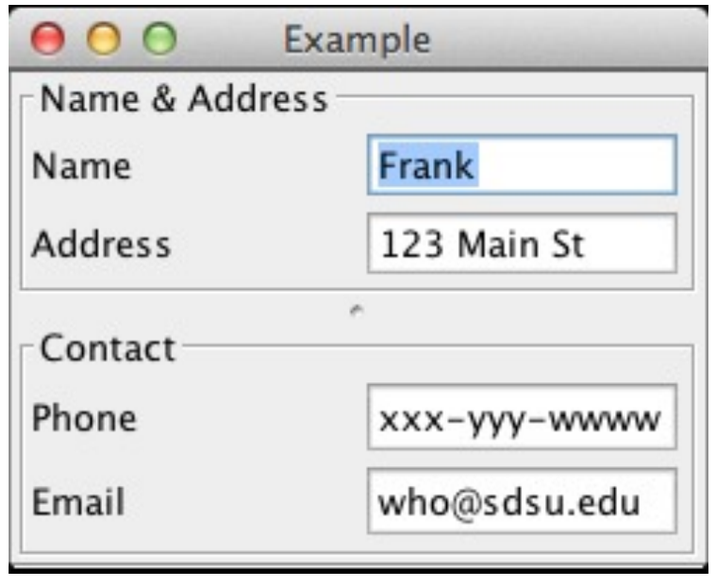
```
(def grid-address (seesaw/grid-panel
  :border "Name & Address"
  :columns 2
  :items ["Name" (seesaw/text :text "Frank" :id :name )
          "Address" (seesaw/text "123 Main St")]))
(display grid-address 200 100)
```




```
(def grid-contact (seesaw/grid-panel
  :border "Contact"
  :columns 2
  :items ["Phone" (seesaw/text :text "xxx-yyy-wwww" :id :phone )
          "Email" (seesaw/text :text "who@sdsu.edu" :tip "Enter your email")]))
```

Nesting panels

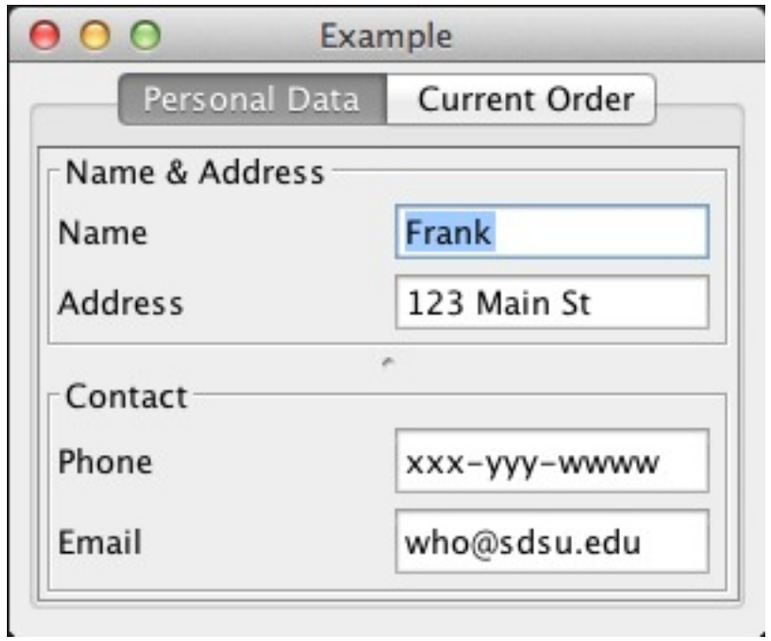
```
(def split-panel (seesaw/top-bottom-split
                   grid-address
                   grid-contact))
(display split-panel 250 200)
```



grid-address & grid-contact from previous slide

More Nesting

```
(def order-tab (seesaw/tabbed-panel
          :placement :top
          :tabs [{:title "Personal Data"
                  :content split-panel}
                 {:title "Current Order"
                  :tip "To be or not to b"
                  :content "Add content here"}]))
```



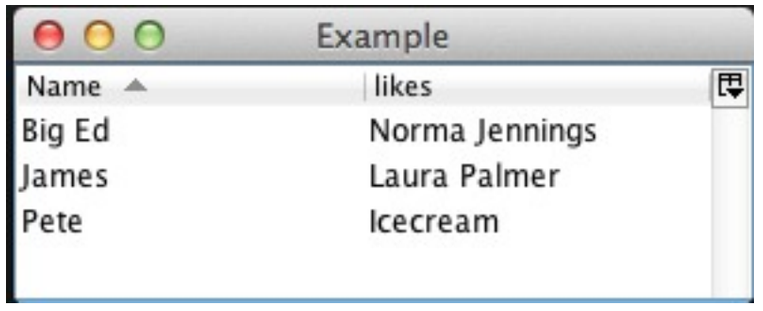

```
(display order-tab 300 250)
```

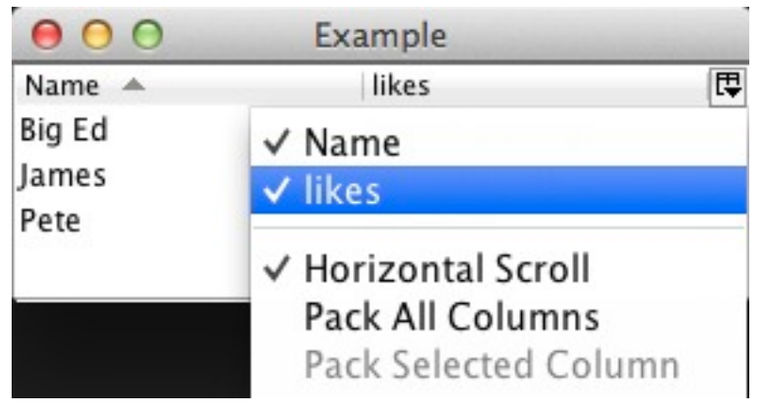

split-panel is from previous slide

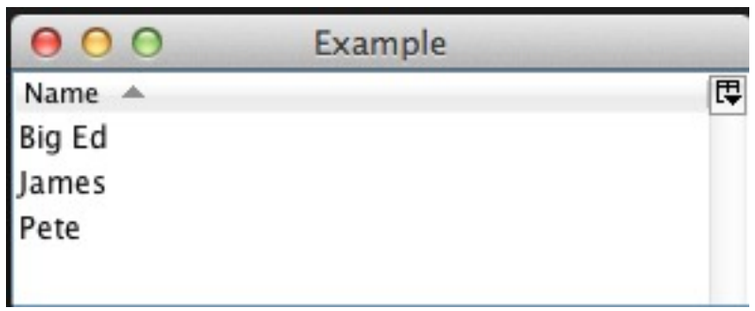
table-x

```
(:require [seesaw.core :as seesaw]
          [seesaw.swingx :as swingx])

(def table (swingx/table-x
  :horizontal-scroll-enabled? true
  :model [
    :columns [{:key :name :text "Name"} :likes]
    :rows [["Pete" "Icecream"]
           {:likes "Laura Palmer" :name "James"}
           {:name "Big Ed" :likes "Norma Jennings"}]]))
```

(display (seesaw/**scrollable** table) 300 200)



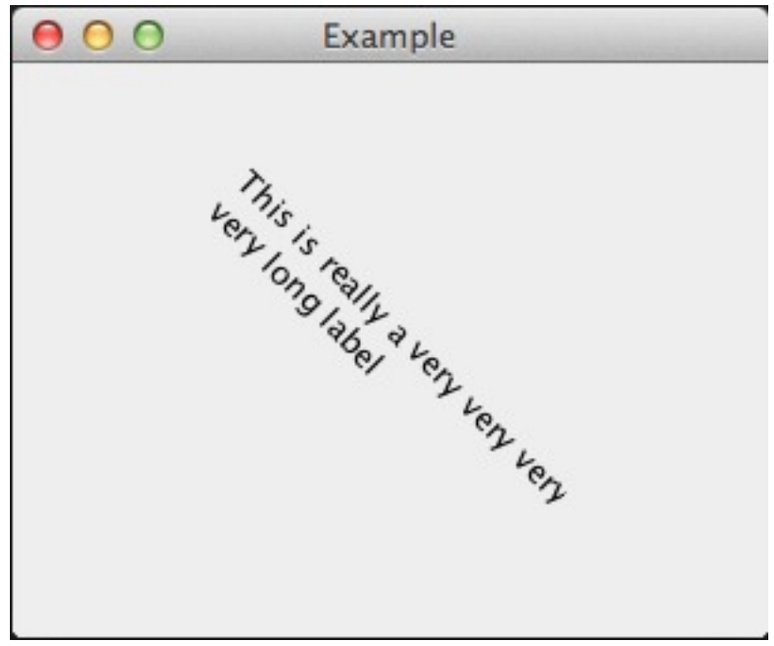




Hyperlink


```
(:require [seesaw.core :as seesaw]
          [seesaw.swingx :as swingx])

(def link
  (swingx/hyperlink :text "Click Me" :uri "http://google.com"))

(display link 200 50)
```

label-x

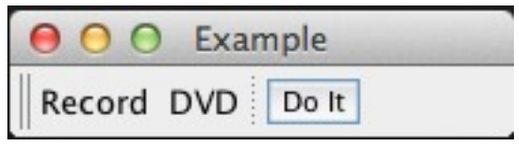


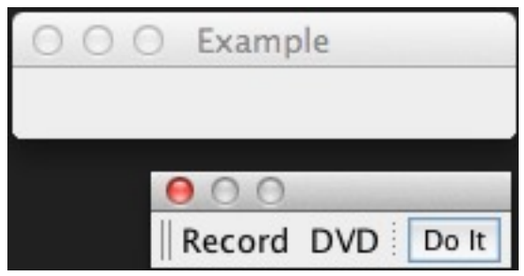

```
(:require [seesaw.core :as seesaw]
          [seesaw.swingx :as swingx])
```

```
(def label
  (swingx/label-x :text        "This is really a very very very very very very long label"
          :wrap-lines? true
          :text-rotation    (Math/toRadians 45.0)))
```

```
(display label 300 250)
```

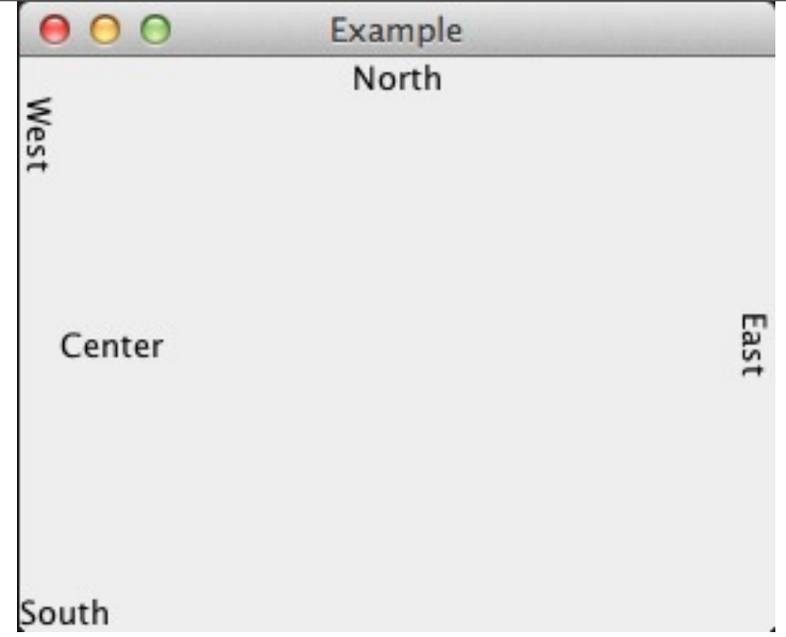
toolbar





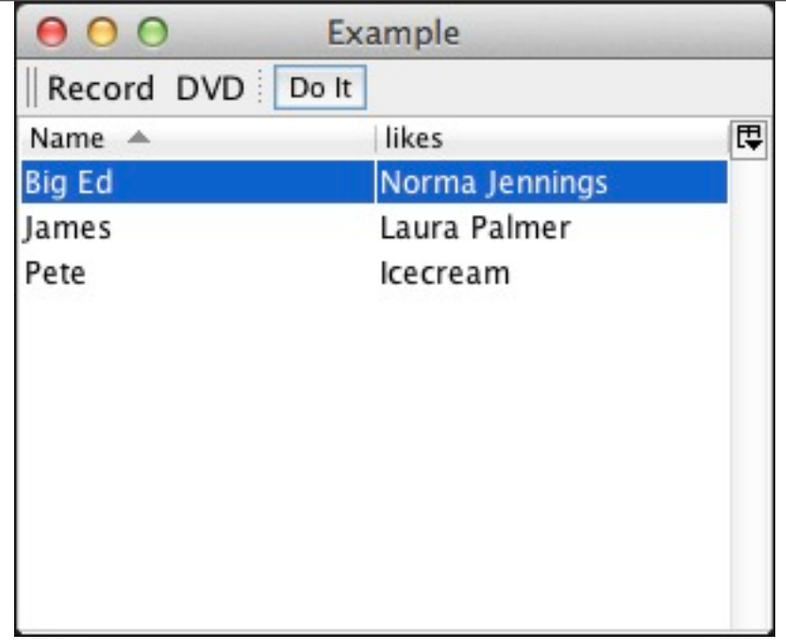

```
(def toolbar
  (seesaw/toolbar
   :floatable? true
   :orientation :horizontal
   :items ["Record"
           "  "
          "DVD"
          :separator
          (seesaw/action :handler #(seesaw/alert % "Done") :name "Do It")]))

(display toolbar 200 50)
```

border-panel




```
(def border
  (seesaw/border-panel
   :north (seesaw/label :text "North" :halign :center)
   :south "South"
   :center "Center"
   :east (swingx/label-x
          :halign :center
          :text "East"
          :text-rotation    (Math/toRadians 90.0))
     :west (swingx/label-x
          :text "West"
          :text-rotation    (Math/toRadians 90.0))))

(display border 300 250)
```

Nesting Containers again




```
(def border
  (seesaw/border-panel
   :north toolbar
   :center (seesaw/scrollable table) ))

(display border 300 250)
```


toolbar and table are defined on previous slides

User-data

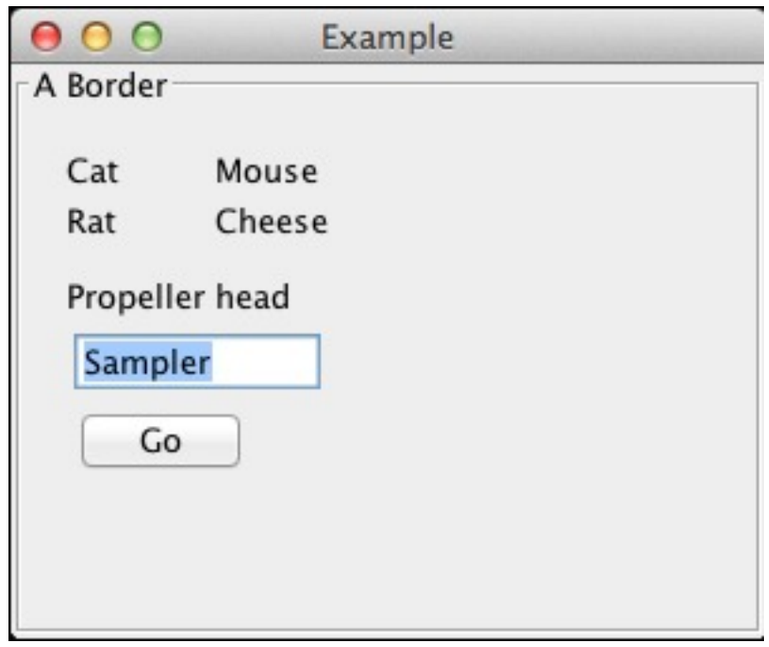
Attaching data to widgets

```
(seesaw/label :text "HI!" :user-data 99)
(seesaw/button :text "Foo" :user-data [ 1 2 3])
(seesaw/button :text "Bar" :user-data #(seesaw/alert "Hi"))
```

Accessing the data

```
(seesaw/user-data (seesaw/label :text "HI!" :user-data 99))
```

Midge layout



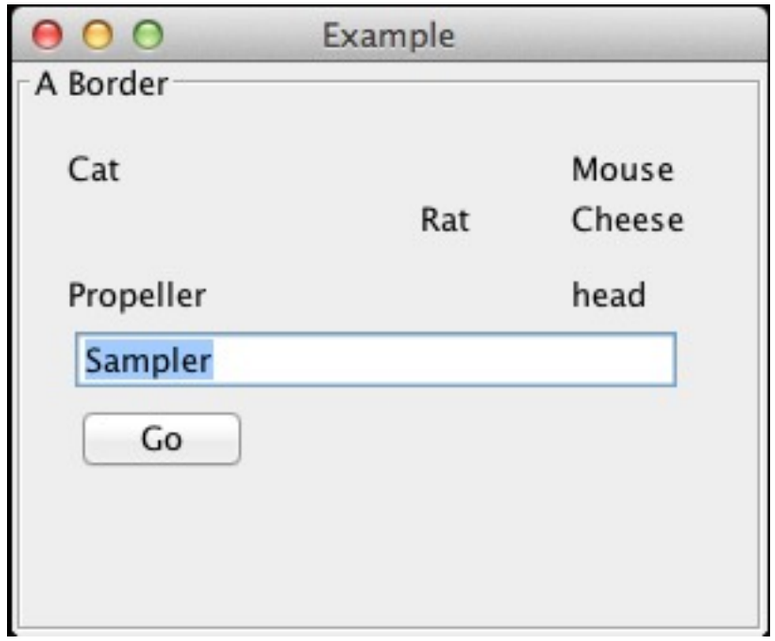

```
(def mig-example (mig/mig-panel
      :border "A Border"
       :constraints ["wrap 2"]
       :items [["Cat"]      ["Mouse"]
               ["Rat"]      ["Cheese"]
               [ "Propeller"  "gaptop 10"] ["head"]
               [(seesaw/text :text "Sampler") "grow, span"]
               [(seesaw/action :name "Go" :handler #(seesaw/alert % "Gone")) "span 2"]
                ]))

(display toolbar 200 50)
(display mig-example 300 250)
```

mig-panel

(mig-panel & opts)

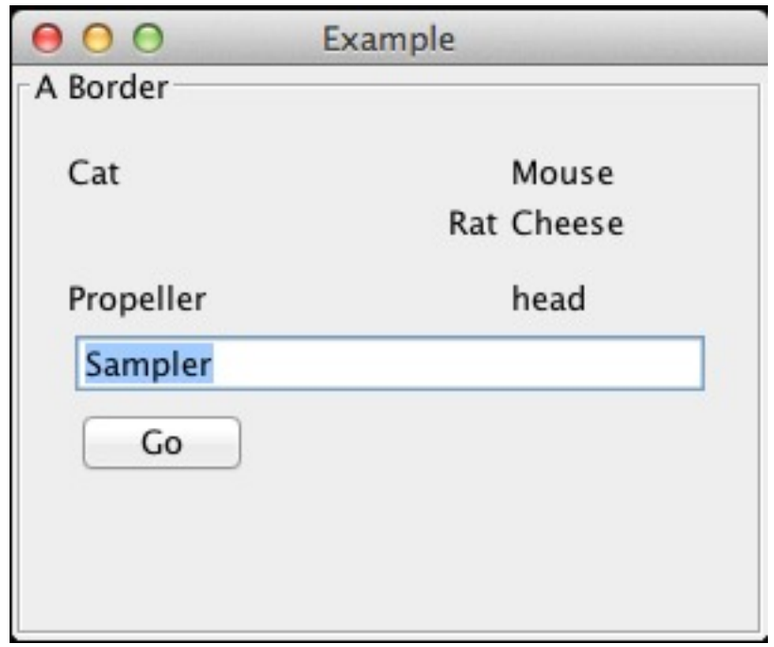
:constraints ["layout constraints" "column constraints" "row constraints"]

Column widths - px




```
(def mig-example (mig/mig-panel
      :border "A Border"
       :constraints ["wrap 2" "[160]40[30]"]
       :items [["Cat"]      ["Mouse"]
               ["Rat" "align right"]      ["Cheese"]
               [ "Propeller"  "gaptop 10"] ["head"]
               [(seesaw/text :text "Sampler") "grow, span"]
               [(seesaw/action :name "Go" :handler #(seesaw/alert % "Gone")) "span 2"]
                ]))

(display toolbar 200 50)
(display mig-example 300 250)
```

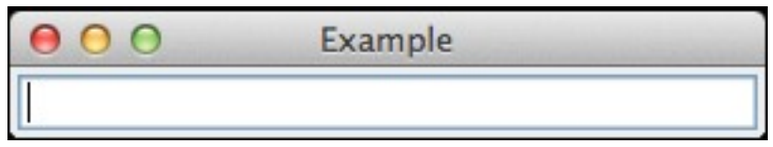
Column widths - percent



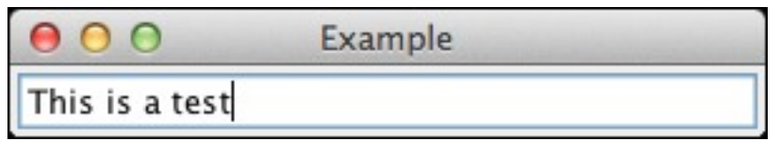

```
(def mig-example (mig/mig-panel
      :border "A Border"
       :constraints ["wrap 2" "[60%]5[30%]"]
       :items [["Cat"]      ["Mouse"]
               ["Rat" "align right"]      ["Cheese"]
               [ "Propeller"  "gaptop 10"] ["head"]
               [(seesaw/text :text "Sampler") "grow, span"]
               [(seesaw/action :name "Go" :handler #(seesaw/alert % "Gone")) "span 2"]
        ]))
```

Bind - Connecting atom to Textfield

```
(:require [seesaw.core :as seesaw]
          [seesaw.bind :as bind])
```

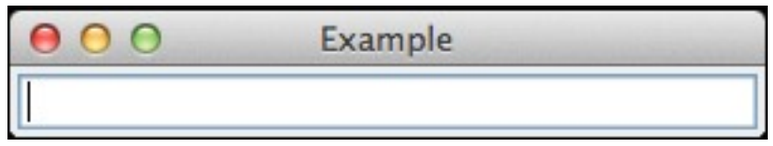



```
(def atom-value (atom "Cat"))
(def sample-text (seesaw/text))
(bind/bind sample-text atom-value)
(display sample-text 300 50)
```

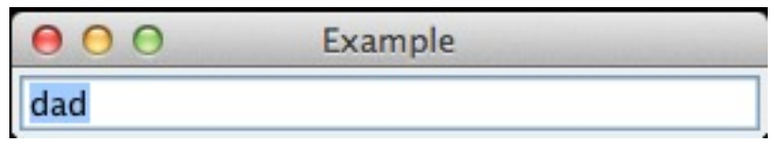


@atom-value "This is a test"

Bind - Connecting Textfield to atom



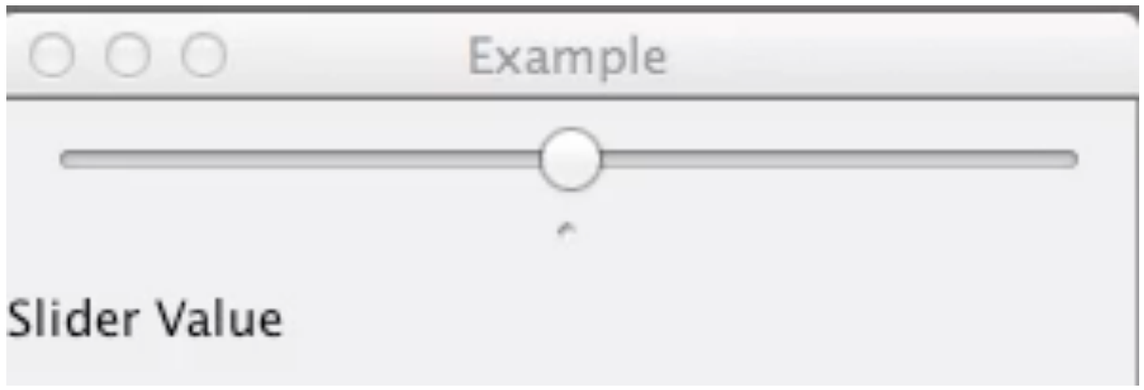

```
(def atom-value (atom "Cat"))
(def sample-text (seesaw/text))
(bind/bind  atom-value (bind/property sample-text :text))
(display sample-text 300 50)

(reset! atom-value "dad")
```



No You can not bind both ways

```
(def atom-value (atom "Cat"))
(def sample-text (seesaw/text))
(bind/bind  atom-value (bind/property sample-text :text))
(bind/bind  sample-text atom-value)
(display sample-text 300 50)
```

Slider Example




```
(def slider (seesaw/slider :min 0 :max 100 :value 50))

(def label (seesaw/label :text "Slider Value"))

(bind/bind slider (bind/transform / 100.0) (bind/property label :text))

(def example (seesaw/top-bottom-split slider label))

(display example 300 100)
```

Declarative Programming

Describes what computation should be performed and not how to compute it

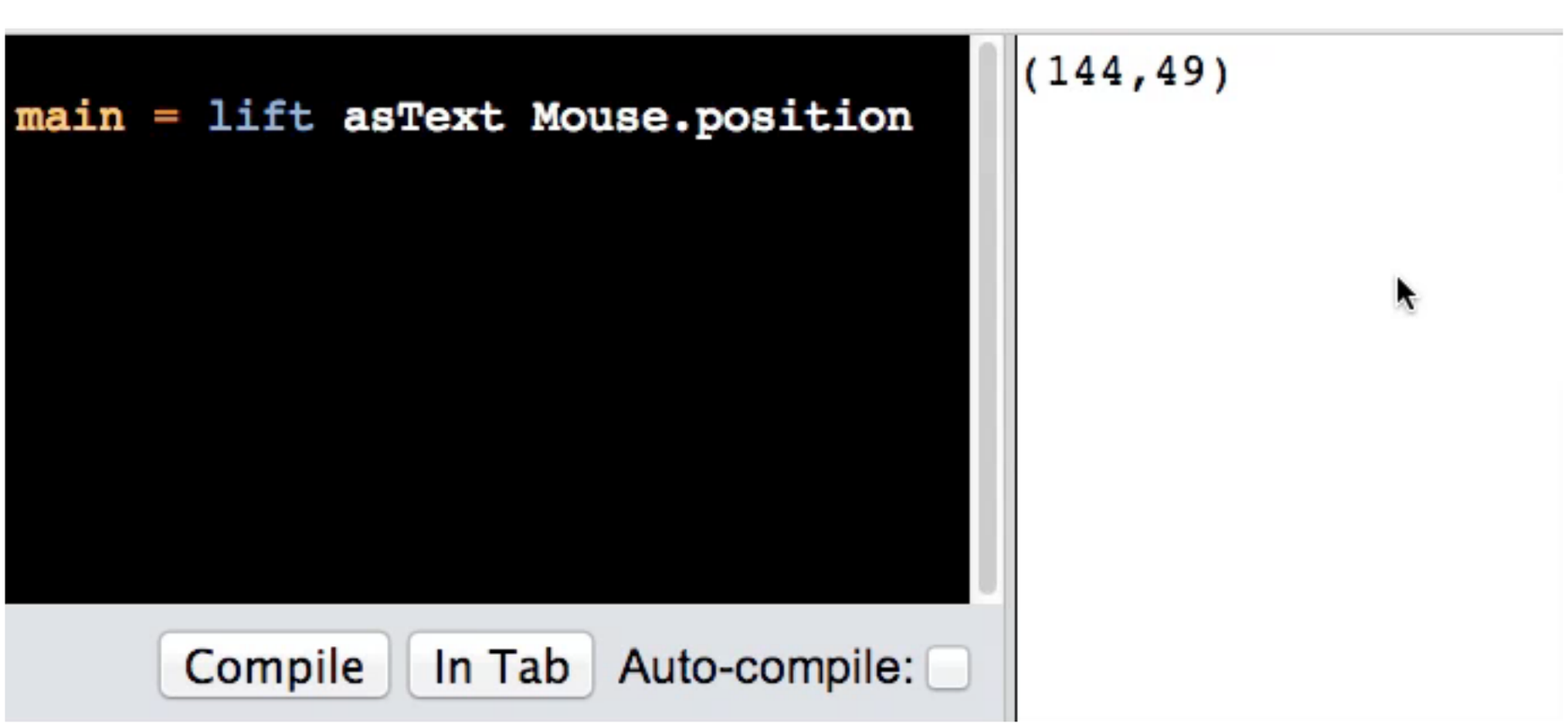
```
sum = 0;
for (int k = 0; k< array.length; k++)
      sum = sum + array[k];
```

```
(reduce + array)
```

Reactive Programming

datatypes that represent a value 'over time'

Spreadsheets
Elm
Meteor.js



Thursday, November 13, 14

http://stackoverflow.com/questions/1028250/what-is-functional-reactive-programming
http://elm-lang.org/
http://en.wikipedia.org/wiki/Reactive_programming

React from Facebook

http://facebook.github.io/react/

Javascript UI library

- one-way reactive data flow

Om

- ClojureScript library on top of React