Reaktor Breakpoint 2018 ClojureScript Workshop

Clojure core API: https://clojure.org/api/cheatsheet

Defining variables

```
(def my-var "reaktor breakpoint")
const myVar = 'reaktor breakpoint'
```

Comments

```
;; clojure comment, one ; works too
// javascript comment
```

Functions

```
(defn special-sum-fn [x y]
  (+ x y))
const specialSumFn = (x y) => x + y

(special-sum-fn 5 6) ;; => 11
specialSumFn(5, 6)

(def special-inc-fn (fn [x]
                      (inc x))
const specialIncFn = x => x + 1

(def short-syntax #(+ %1 %2))
const shortSyntax = (x y) => x + y
```

Brackets in Clojure

```
;; (brackets around things) cause
;; them to get evaluated

(defn my-fn []
  (println "breakpoint"))

(my-fn)
"breakpoint"

my-fn
#object[user$my_fn 0x65ab50e5
"user$my_fn@65ab50e5"]
```

Vectors (arrays)

```
(def arr ["foo" "bar" 1 3])
const arr = ['foo', 'bar', 1, 3]

(first arr)
arr[0]

(nth arr 2)
arr[2]
```

Maps (objects)

```
(def my-map {:key1 "val"
             :key2 3241})
const myObj = {
  key1: 'val',
  key2: 3241
}

(:key1 my-map)
myObj.key1

(get my-map :key2)
myObj.key2

;; assign key-pair to map
(def my-map2
  (assoc my-map :reaktor "clojure"))
const myObj2 =
  R.assoc('reaktor', 'clojure', myObj)

;; update existing value in map
;; by applying a function to a value
(def my-map3
  (update my-map :key2 inc))
const myObj3 =
  R.evolve({ key2: R.inc }, myObj)
```

Map

```
(map (fn [num]
       (inc num))
     [1 2 3 4])    ;; => [2 3 4 5]
```

```
[1, 2, 3, 4]
  .map(num => num + 1)

;; shorter version
(map inc [1 2 3 4])
```

Reduce

```
(reduce (fn [accumulator num]
          (+ accumulator num))
        [1 2 3 4])
[1, 2, 3, 4].reduce((accumulator, num)
  => accumulator + num
)

;; shorter version
(reduce + [1 2 3 4])
```

Expression threading

```
(* (/ (+ 5 1) 2) 5)
;; same with threading
(-> 5
    (+ 1)     ;; (+ 5 1) => 6
    (/ 2)     ;; (/ 6 2) => 3
    (* 5))    ;; (* 3 5) => 15

(map (fn [num] (* num 2)) (map inc [1
2 3 4]))
;; same with threading
(->> [1 2 3 4]
     (map inc)
     (map (fn [num]
            (* num 2))))
R.pipe(
  R.map(R.inc),
  R.map(num => num * 2)
)([1, 2, 3, 4])
```

Destructuring vectors (arrays)

```
;; args:
;;   - {:click-count 5}
;;   - [:increase-count [:style-button]]

(defn increase-count
  [state [handler-name [button-id]]]
  ;; handler-name = :increase-count
  ;; button-id = :style-button
  ...
)
```

Destructuring maps (objects)

```
;; args:
;;   - {:click-count    5
;;      :current-page :stats-page}
;;   - [:increase-count [:style-button]]

(defn increase-count
  [{:keys [click-count current-page]}
   params]
  ;; click-count = 5
  ;; current-page = :stats-page
  ...
)
```

Create React components

```
(defn component-fn [props]
  [:div
    [:a {:href "https://www.reaktorbreakpoint.com"}
      "Reaktor Breakpoint"]])
```

```
<div>
  <a href="https://www.reaktorbreakpoint.com">
    Reaktor Breakpoint
  </a>
</div>
```

Dispatch events from React components

```
(defn component-fn []
  [:div
    [:a {:href "https://www.reaktorbreakpoint.com"
         :on-click (fn [event]
                     (re-frame.core/dispatch [:breakpoint-link-clicked]))}
      "Reaktor Breakpoint"]])
```

Handle events dispatched by React components and update app state

```
;; db = application state in re-frame's domain language

(re-frame.core/reg-event-db :breakpoint-link-clicked
  (fn [db]
    (update db :breakpoint-link-click-count inc)))          ;; new application state
                                                            ;; is always returned
```

Subscribe React component to application state

```
;; when using reactive variables, @ must be used to read the value from them
;; react component will rerender only when subscription value changes

(defn click-count-indicator []
  (let [count (re-frame.core/subscribe [:link-click-count])]
    [:div
      [:span (str "link has been clicked " @count "times")]]])

;; the React component above gets value returned by the subscription function

(re-frame.core/reg-sub :link-click-count
  (fn [db]
    (:breakpoint-link-clicked db)))
```