Clojure

Quién soy

María Clara Juanita De La Cuesta

Ingeniera electrónica de la UPB, experiencia en hardware, firmware y ahora ¡en webapps!

Me gustan los gatos y los libros y el maquillaje.

Actualmente trabajo en Barista Ventures.

Programación funcional VS POO

input-> functions-> outputs **VS** objects actions and interactions

En PF los objetos son inmutables, las funciones no tienen efectos fuera de su ámbito y dadas las mismas entradas, siempre retornarán las mismas salidas.

Sintaxis de clojure

```
(operator operand1 operand2 ... operandn)
```

Todas las formas se escriben igual y los paréntesis son llamadas a funciones. Los programas se dividen en *namespaces*.

Tipos de datos

(reader forms)

LITERALS

Números: -23 3447.98 16/7

Cadenas (strings): “Hola”

Caracteres: \h \space \tab

Booleanos: true false

Nil: nil (Falso lógico)

Claves (Keywords): :name

ESTRUCTURAS

Listas: `‘(1 2 3 4)` `(list 1 2 3 4)` **Mapas:** `{:name ["maria" "clara"] 4 85}`

Vectores: `[1 2 3]` `(vector 1 2 3)` **Colecciones (sets):** `#{:a 6 #{:hi “hi”}}`

Los diferentes elementos pueden ser de cualquier tipo y pueden estar mezclados y anidados.

Para Clojure, todo son *sequencias.*

Control de flujo

Condicionales
if - do - when

```
(if "anything other than nil or
false is considered true"
  "A string is considered true"
  "A string is not considered
true")

(if nil
  "nil is considered true"
  "nil is not considered true")

(if (get {:a 1} :b)
  "expressions which evaluate to
nil are considered true"
  "expressions which evaluate to
nil are not considered true")
```

FUNCIONES

Definición

```
(defn do-something
    "Función que no hace nada"
  ([one two] "hm, ok, will do"))
```

Retorna lo último que se evaluó.

Definición *arity* y &

```
(defn do-something
    "Función que no hace nada"
  ([] "nothing")
  ([one] "easy!")
  ([one two] "hm, ok, will do")
  ([one two & more] "oh, no, so many!"))
```

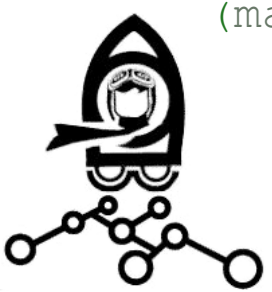
Funciones anónimas

```
(map f c1 c2 c3 & colls)
```

```
(map (fn [x] (* x x)) (1 2 3 4))

(map #(* % %) (range 1 10))

(map #(str "Hola " % "!" ) ["Clara" "Marian"] ["Cami" "Clau"])
```

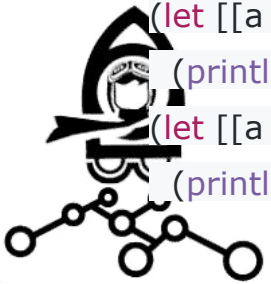


Destructuring

```
(let [bindings*] expresions*)
```

```
(def my-vector [:a :b :c :d])
(def my-nested-vector [:a :b :c :d [:x :y :z]])

(let [[a b c d] my-vector]
  (println a b c d))
(let [[a _ _ d [x y z]] my-nested-vector]
  (println a d x y z))
(let [[a b & the-rest :as all] my-vector]
  (println a b the-rest all))
```



```
(def my-hashmap {:a "A" :b "B" :c "C" :d "D"})
(def my-nested-hashmap {:a "A" :b "B" :c "C" :d "D"
:q {:x "X" :y "Y" :z "Z"}})

(let [{a :a d :d} my-hashmap]
  (println a d))
(let [{a :a, b :b, {x :x, y :y} :q} my-nested-hashmap]
  (println  b  y))
(let [{a :a, not-found :not-found, b :b} my-hashmap]
  (println a not-found b))
(let [{:keys [a b], {:keys [x y]} :q}
my-nested-hashmap]
  (println a b x y))
```

Ejercicios y más funciones

https://github.com/Juanadelacuesta/clojure-pioneras.git

https://docs.google.com/document/d/1FHF9Js_sIIaTnXSnTvB4HANBGHdLcJHs5n-AtwkrsIk/edit?usp=sharing

https://repl.it/languages/clojure