Curriculum

ThoughtWorks®

AGENDA

1. **6:30pm** Intro
2. **7:00pm** Clojure
3. **8:30pm** Time to experiment & Mingling

Before we start...

1. There is a Code of conduct

tl;dr Be nice and support each other

2. Who are we?
3. How is this going to work?
4. Cheat sheet

The Game

Open the project.clj in IntelliJ as Project

Our project has the following structure:

- `starter-project`
  - `resources` → Sprites, Textures, ...
  - `src` → Source Code
    - `core` → Game Loop, screen rendering
    - `entities` → Animations and texture loader
    - `launcher` → Game launcher (initialize graphic comp.)
    - `player` → Player specific functions
    - `utils` → Utility functions (physics, ...)

What is a function?

Functions is all we do

What is a function?

```
(bake milk flour eggs sugar)
```

```
(bake milk flour eggs)
```

Functional programming ≙ Writing functions all the time

Functions is all we do

Defining a function?

```
(defn bake
   [milk flour eggs sugar]
   (cake))

(defn bake
   ([milk flour eggs sugar] (cake))
   ([milk flour eggs]       (pancake)))
```

Functional programming ≙ Writing functions all the time

Calculate squares

1. Define a function square ...
   a. ... that takes one number
   b. ... and returns its square

2. Examples:
   a. (square 0) ; => will result in 0
   b. (square 1) ; => will result in 1
   c. (square 3) ; => will result in 9

*Hints: defn, **

Calculate squares

1. Define a function square ...
   a. ... that takes one number
   b. ... and returns its square

2. Examples:
   a. (square 0) ; => will result in 0
   b. (square 1) ; => will result in 1
   c. (square 3) ; => will result in 9

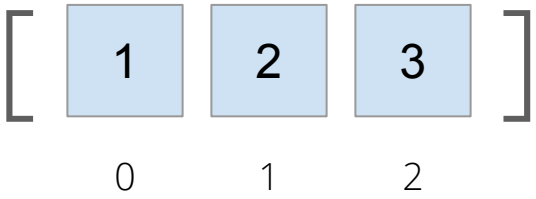
Possible solution:

```
(defn square[base-number]
 (* base-number base-number))
```

Collections

Vectors

Vectors are an indexed, sequential data structure that hold data in each compartment



Creating a vector:

```
[1 2 3]
(vector 1 2 3)
(conj [1 2] 3)
(cons 1 [2 3])
```

How to access values in vectors

```
(first [1 2 3]) ; => 1

(second [1 2 3]) ; => 2

(nth [1 2 3] 1) ; => 2 - index starts at 0

(rest [1 2 3]) ; => [2 3] - vector without first element
```

Maps

Collection of key and value pairs. They are unordered and contain no duplicate keys.

```
{
   :recipe-name    “Delicious Cookies”
   :duration-in-s 30
}
```

Creating a map:

```
{:width 30 :height 40}
(assoc {:width 30} :height 40)
(merge {:width 30} {:height 40})
```

How to access values in maps

```
(get {:width 30 :height 40} :height) ; => 40

(:height {:width 30 :height 40}) ; => 40

(get-in
   {
     :name "Triangle"
     :size {:width 30 :height 40}
   }
   [:size :height]) ; => 40
```

Update values in maps

```
(update {:eggs 0 :flour 1} :eggs inc)
; => {:eggs 1 :flour 1}, (inc) increases the old value

(update {:eggs 0 :flour 1} :eggs + 5)
; => {:eggs 5 :flour 1}

(update-in {:name "Cookies"
            :ingredients {:eggs 0 :flour 1}}
           [:ingredients :eggs] + 5)
; => {:name "Cookies" :ingredients {:eggs 5 :flour 1}}
```

Create your character!

1. Modify the function (create) in *src/super_koalio/player.clj*
   a. that returns a map with our player attributes

```
:x 20 :y 10 :width 1 :height 1.5
```

Hints: defn, assoc

Create your character!

1. Modify the function (create) in *src/super_koalio/player.clj*
   a. that takes the filename of a picture as first argument (“x.png”)
   b. that creates a texture using (create-texture) in *src/springbake/player.clj*
   c. that associates the texture with the attributes

```
:x 20 :y 10 :width 1 :height 1.5
```

Possible solution:

```
(defn create []
    {:x       20
    :y       10
    :width   1
    :height 1.5})
```

Conditionals

Flow so hard

`if` consists of a `condition`, a `then`, and an `else`

```
(if (contains? cake eggs))
    (println "Not vegan!")
    (println "The cake is vegan"))
```

condition

then

else

Flow so hard

`when` is an `if` with only a `then` branch

```
(when (contains? cake milk)
    (println "Not vegan!"))
```

condition

then

Flow so hard

`cond` is like an if but with multiple `conditions`

```
(cond
  (= y 1)  "Solution is 1!"      ; condition then
  (= y 2)  "Solution is 2!"      ; condition then
  (= y 3)  "Solution is 3!"      ; condition then
  :else "OMG I don't know what to do!")  ; condition then
```

The world is bigger than that

Make your character move sideways according to the buttons:

- Left arrow (left)
- Right arrow (right)

Modify the function (move-sideways) in *src/super_koalio/player.clj* to:

1. Return -15 when your character should move left
2. Return 15 when your character should move right
3. Return 0 in all other cases

Hints: cond, key-pressed?

Keys are: `:dpad-left`, `:dpad-right`

The world is bigger than that

Possible solution:

```
(defn move-sideways []
 (cond
   (key-pressed? :dpad-left) -15)
   (key-pressed? :dpad-right) 15)
   :else 0))
```

Take a leap!

Make our player jump! Our (jump) function returns the height the player can jump.

Modify (jump) so that:

1. When hitting the `:dpad-up` the player should jump 55
2. Return 0 if nothing happened

Hints: if, key-pressed?

Take a leap!

Make our player jump! Our (jump) function returns the height the player can jump.

Modify (jump) so that:

1. When hitting the `:dpad-up` the player should jump 55
2. Return 0 if nothing happened

```
(defn jump [entity]
 (if (key-pressed? :dpad-left)
    55
    0))
```

Don’t cheat though!

Right now it is possible to jump as many times as you want while hitting up.

Modify (jump) so that the player can only jump:

1. when the `:dpad-up` is pressed AND
2. it is possible to jump - (can-jump?)
3. return 0 if nothing happened

Hints: and, can-jump?

Don’t cheat though!

Right now it is possible to jump as many times as you want while hitting up.

Modify (jump) so that the player can only jump:

1. when the `:dpad-up` is pressed AND
2. it is possible to jump - (can-jump?)
3. return 0 if nothing happened

```
(defn jump [entity]
 (if (and (can-jump? entity)
          (key-pressed? :dpad-left))
    55
    0))
```

Try to beat the game!

Resources

clojure.org | clojurebridge.org | play-clj

How to reach us

dschruhl@thoughtworks.com |
michael.calvert@thoughtworks.com