introducing
managed effects

@hkgumbs

8th Light

@hkgumbs

@hkgumbs

```
let a = [ “Hello” ]
a[2]
```

```
let a = [ “Hello” ]
a[2]
```

@hkgumbs

```
conf =
     { place = “CodeMash”
     , time = 8
     , topic = Nothing
     }
```

```
conf =
     { place = “CodeMash”
     , time = 8
     , topic = Nothing
     }
```

```
conf =
    { place = “CodeMash”
    , time = 8
    , topic = Just “elm”
    }
```

```
addConf calendar =
    conf :: calendar
```

```
let a = [ “Hello” ]
var b = a
b.append(“World”)

a != b
```

/// Arrays, like all variable-size collections in the standard library, use copy-on-write optimization.

```
let a = [ “Hello” ]
var b = a
b.append(“World”)

a != b
```

```
update :
    msg -> model -> model
```

```
program fs =
    Native.VDom.program fs
```

```
var program = mkProgram();
```

```
update :
    msg -> model -> model
```

compile target
effect management

compile target

effect management

```
type Maybe a
     = Just a
     | Nothing
```

compile target
effect management

Effects are interactions
with external state
  - Richard (Elm in Action)

```
    elm
     +
js effects  >  compiled js
```

elm
+
js effects

> compiled js

@hkgumbs

Elm has a DSL for scripting effects called JS

- wheatBread (Reddit)

Elm has a DSL for scripting effects called JS

- Evan (creator of Elm)

```
elm
  +
???
```

```
>  compiled js
```

```
  js logic
     +
js effects
```

> webapp

introducing
managed effects

@hkgumbs

```
Http.get json “google.cal”
  |> Task.attempt addConf
```

```
addConf maybeCalendar =
    case maybeCalendar of
        Just cal ->
            conf :: cal
        Nothing ->
            [ conf ]
```

```
function addMeetup(cal){
  return cal
    ? [meetup].concat(cal)
    : [meetup];
}
```

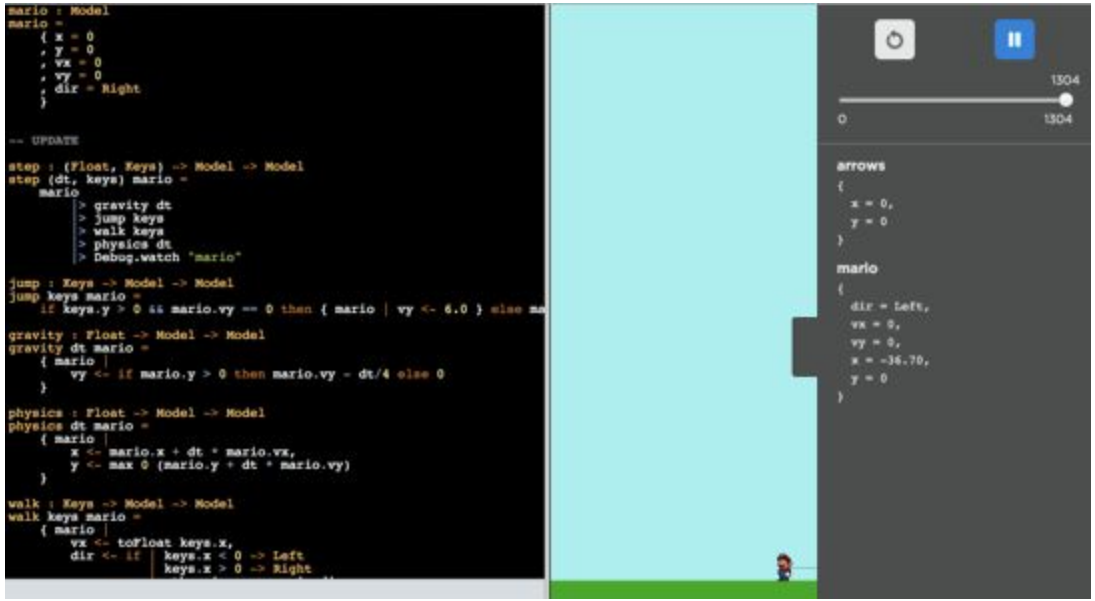
```
function get(json, url){
  return {
    action: “HTTP_GET”,
    data: {
      json: json, url: url
    }
  };
}
```

```
main(effects, {
   init: [],
   update: update,
   view: view
});
```

```
function effects(task, f){
  switch (task.action){
    case “HTTP_GET”:
      $.get(...).then(f);
  }
}
```

¯_(ツ)_/¯

@hkgumbs


mario : Model
mario =
{ x = 0
, y = 0
, vx = 0
, vy = 0
, dir = Right
}
-- UPDATE
step : (Float, Keys) -> Model -> Model
step (dt, keys) mario =
mario
|> gravity dt
|> jump keys
|> walk keys
|> physics dt
|> Debug.watch "mario"
jump : Keys -> Model -> Model
jump keys mario =
if keys.y > 0 && mario.vy == 0 then { mario | vy <- 6.0 } else ma
gravity : Float -> Model -> Model
gravity dt mario =
{ mario |
vy <- if mario.y > 0 then mario.vy - dt/4 else 0
}
physics : Float -> Model -> Model
physics dt mario =
{ mario |
x <- mario.x + dt * mario.vx,
y <- max 0 (mario.y + dt * mario.vy)
}
walk : Keys -> Model -> Model
walk keys mario =
{ mario |
vx <- toFloat keys.x,
dir <- if | keys.x < 0 -> Left
keys.x > 0 -> Right
1304
0
1304
arrows
{
x = 0,
y = 0
}
mario
{
dir = Left,
vx = 0,
vy = 0,
x = -36.70,
y = 0
}

@hkgumbs

```
function debugFx(task, f){
  var msg = // real effect
  history.append(msg);
  f(msg);
}
```

```
function savedFx(task, f){
  f(history[i++]);
}
```

```
function get(json, url){
  return {
    action: “HTTP_GET”,
    data: {
      json: json, url: url
    }
  };
}
```

introducing
managed effects

@hkgumbs