# OBSERVABLE SEQUENCES

## AN INTRODUCTION

# INTRODUCTIONS

- @tyronetudehope
- Dev @ Continuon
- Interested in FRP

# OBSERVABLE SEQUENCES?

- A sequence of ongoing events ordered in time - Andre Staltz
- Henceforth, Event Streams

**React to events asynchronously in a declarative and composable manner**

# WORKING W/ STREAMS

- Define a producer
- Add some operations
- Subscribe

# WHERE CAN THEY BE USED?

- **Everywhere!**
- User interactions
- FS events
- Arrays
- Single value
- Nothing

# CATS AND DOGS

# OPERATORS

- Transform the stream to emit a new sequence of events
- Akin to prototype methods of an Array object

# MAP

- Similar to Array.prototype.map
- Transforms values through a transformation function

# CLICK COORDINATES

```
clicks$
.map(ev => ({
    x: ev.x,
    y: ev.y
  })
)
```

Example

# FILTER

- Similar to Array.prototype.filter
- Values passed to a predicate
- Only emits values where the function returns true

# OUTSIDE OF BOUNDS

```
clicks$
.map(toCoordinates)
.filter(coords =>
  coords.x > 100 || coords.y > 100
)
```

Example

# CREATE

- Given some producer, create a stream
- Handles putting events onto the stream

# VIEWPORT

```
const createResizeStream = () => {
  const getDimentions = el => ({ width: el.innerWidth, height: el.innerHeight })

  return xs.create({
    start: listener =>
      window.addEventListener('resize', ev =>
        listener.next(getDimentions(ev.target)))
  })
}

const viewport$ = createResizeStream()
```

Example

# COMBINE (COMBINELATEST)

- Merges two or more streams
- Emits latest events from each stream together
- xstream emits an array
- RxJS lets you optionally specify a transformation

# CLICK BELOW MIDDLE

```
xs.combine(click$, viewport$)
  .map(([clickEvent, viewport]) => ({
    x: clickEvent.x,
    y: clickEvent.y,
    width: viewport.width,
    height: viewport.height
  }))
  .filter(hemisphere(-1))
```

Example

# MERGE

- Emit values concurrently from multiple streams

# GETTING DIRECTION

```
const left$ = viewportClick$
  .filter(hemisphere(-1))
  .mapTo(-1)
const right$ = viewportClick$
  .filter(hemisphere(1))
  .mapTo(1)

const direction$ = xs.merge(left$, right$)
```

Example

# REDUCE/SCAN/FOLD

- Similar to Array.prototype.reduce
- Combines emitted values using an accumulator function
- Emits the results of the accumulator

# CHANGING LANES

```
const playerPosition$ = direction$
  .fold((pos, side) =>
    side < 0 ? moveLeft(pos, side) : moveRight(pos, side),
    1 // Start in the centre lane
  )
```

Example

# CHALLENGES

- Restrict user movement to 500ms intervals
- sampleCombine/withLatestFrom
- Used a sync$ to control events emitted by both player and enemies

# LET'S PLAY A GAME

**Cats and Dogs**

# THANK YOU #CTJS

## AMA