Raju Gandhi

BEING PROACTIVELY REACTIVE

@LOOSELYTYPED

REACTIVE PROGRAMMING?

WITH FP PRINCIPLES APPLIED

STREAMS
ASYNC
PUSH

FUNCTIONAL REACTIVE

MAP
REDUCE
FILTER

FRP

FRP

“WHY I CANNOT SAY FRP BUT I JUST DID”

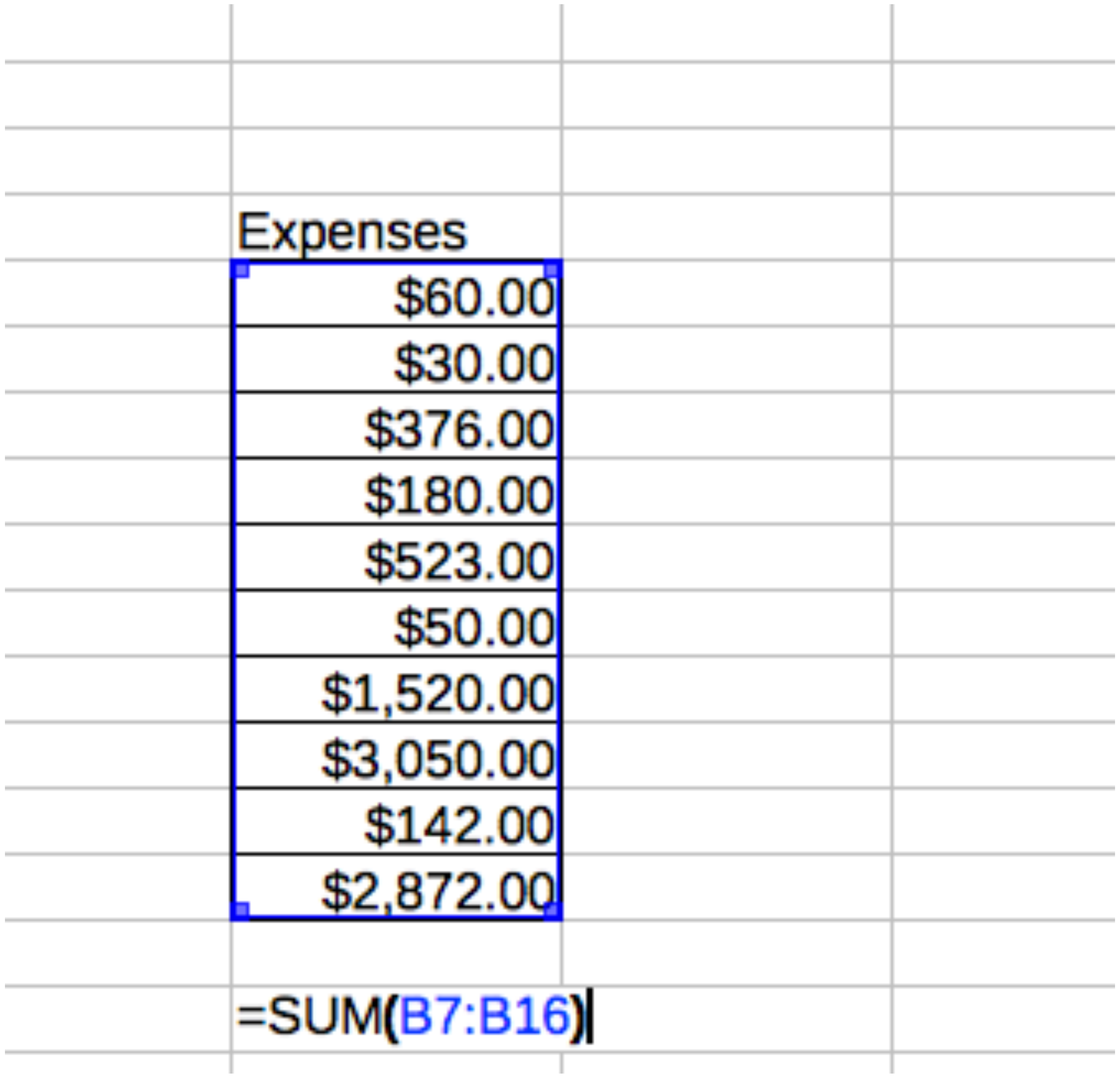
By Andre Staltz

PROMISES?

PROMISES VS OBSERVABLES

- Promises
  - return a single value
  - are eager
  - cannot be “cancelled”

DATA FLOW PROGRAMMING



Expenses
$60.00
$30.00
$376.00
$180.00
$523.00
$50.00
$1,520.00
$3,050.00
$142.00
$2,872.00
=SUM(B7:B16)

OBSERVER PATTERN

```
class Publisher {
  constructor() {
    this.listeners = [];
  }

  addListener(listener) {
    this.listeners.push(listener);
  }

  removeListener(listener) {
    const index = this.listeners.indexOf(listener);
    this.listeners.splice(index, 1);
  }

  notify(msg) {
    this.listeners.forEach((n) => {
      n.send(msg);
    });
  }
};
```

```
class Listener {
  constructor(id) {
    this.id = id;
  }

  send(msg) {
    console.log(`${this.id} received msg: ${msg}`);
  }
}

const publisher = new Publisher();
publisher.addListener(new Listener(1));
publisher.addListener(new Listener(2));

publisher.notify("Broadcasting!");
```

ITERATOR PATTERN

```javascript
const iterable = {
  upperLimit: 10,
  [Symbol.iterator]: function() {
    var that = this;
    return {
      cur: 0,
      next: function() {
        if(this.cur < that.upperLimit) {
          let ret = this.cur;
          this.cur++;
          return {value: ret, done: false};
        }
        this.cur = 0;
        return {done:true};
      }
    };
  }
};

for(let i of iterable) console.log(i); // 0,1,2,…,9
```

OBSERVABLE

OBSERVER + ITERATOR

==

RX

DUALITY

Iterable	Observable
pull	**push**
next()	next(ev)
!hasNext()	completed()
throws Exception	error(e)

SEQUENCE / TIME

```
[1,2,3,4,5]
  .map((n) => n * 2)
  .filter((n) => n % 2 == 0)
  .reduce(((acc, n) =>  acc + n), 0);
```

```
<Observable>
  .map((n) => n * 2)
  .filter((n) => n % 2 == 0)
  .reduce(((acc, n) =>  acc + n), 0);
```

OBSERVABLE

	SINGLE	MULTIPLE
SYNC	FUNCTION	ITERABLE
ASYNC	PROMISE	OBSERVABLE

CREATING OBSERVABLES

EVERYTHING

- Iterables
- Events (mouse move, clicks, keyboard)
- Async (Promises)
- Creating operators

```
// emit a single value (or .return())
Rx.Observable.of(42);

// repeat a value n times
// see also doWhile and while
Rx.Observable.of(42).repeat(3);

// from a range
Rx.Observable.range(1, 5);

// over a interval starting in 5
// seconds, then every 1 second
Rx.Observable.timer(5000, 1000);
```

```
// from an array, map, set or string
// ANY iterable
Rx.Observable
  .from([16,3,56,33,89,45,21])
  .filter(n => n > 25)
  .take(3)
  .subscribe({
    next: n => console.log(n),
    error: e => console.log(e),
    complete: () => console.log('Done!')
  });
```

```
// NEED rx.dom.js !!!
var mouses = Rx.Observable
  .fromEvent(document.getElementsByTagName('body')[0],
'mouseover');

var cancel = Rx.Observable
  .fromEvent(document.getElementById('cancel'), 'click');

mouses
  .takeUntil(cancel)
  .subscribe({
    onNext: n => console.log(n),
    onError: e => console.log(e),
    onComplete: () => console.log('Done!')
   });
```

```
// READ A DIR USING NODE
const fs = require('fs');
const readDir = Rx.Observable
                .bindNodeCallback(fs.readdir)('/Users/raju');

readDir
  .flatMap((n) => Rx.Observable.from(n))
  .subscribe({
    next: n => console.log(`file is ${n}`),
    error: e => console.log(e),
    complete: () => console.log('Done!')
   });
```

THE PLAYERS

Producer

```
Rx.Observable
  .from([16,3,56,33,89,45,21])
  .filter(n => n > 25)
  .take(3)
  .subscribe({
    next: n => console.log(n),
    error: e => console.log(e),
    complete: () => console.log('Done!')
  });
```

```
Rx.Observable
  .from([16,3,56,33,89,45,21])
  .filter(n => n > 35)
  .take(3)
  .subscribe({
    next: n => console.log(n),
    error: e => console.log(e),
    complete: () => console.log('Done!')
  });
```

Consumer

```
Rx.Observable
  .from([16,3,56,33,89,45,21])
  .filter(n => n > 25)
  .take(3)
  .subscribe({
    next: n => console.log(n),
    error: e => console.log(e),
    complete: () => console.log('Done!')
  });
```

Pipeline

```
Rx.Observable
  .from([16,3,56,33,89,45,21])
  .filter(n => n > 25)
  .take(3)
  .subscribe({
    next: n => console.log(n),
    error: e => console.log(e),
    complete: () => console.log('Done!')
  });
```

OPERATORS

OPERATORS

- map, filter
- reduce, scan
- flatMap/switchMap

ERROR HANDLING

ERRORS

- catch
- retry
- retryWhen

```
Rx.Observable.from([1,2,'a','b'])
  .map((n) => {
    if(Number.parseInt(n)) {
      return n;
    } else {
       throw Error('Oops');
    }
  })
  .catch((e) => Rx.Observable.of({
     error: `An error ${e} occurred`
  }))
  .subscribe({
    next: n => console.log(n),
    // onError is no longer relevant!
    error: e => console.log(e),
    complete: () => console.log('Done!')
   });

// next 1
// next 2
// next { error: 'An error Error: Oops occurred' }
```

```
(function() {
  let count = 0;

  Rx.Observable.range(1, 3)
  .map((n) => {
    if(count < 2) {
      count++;
      console.log(`${n}th time`);
      throw new Error("Error!!");
    } else {
      return n;
    }
  })
  .retry(3)
  .subscribe({
    next: (n) => console.log('Raju Next: ', n),
    error: (err) => console.log('Raju Error', err),
    complete: () => console.log("Raju Done")
  });
})();
```

HOT & COLD OBSERVABLES

HOT VS COLD OBSERVABLES

- Hot
  - are like “live” events
  - subscribers could “miss” events (that occurred before they subscribed)
  - broadcast
- Cold
  - are like “video” recording
  - all subscribers get all the events (from the beginning) upon subscription
  - send events asynchronously

GENERAL GOTCHAS

GOTCHAS

- There are 2 **separate** projects - Reactive-Extensions/RxJS and ReactiveX/rxjs (rxjs is to replace RxJS)
- ES7 Observable proposal
- Rx implementations in multiple languages

RESOURCES

RESOURCES

- LearnRx by Jafar Husain (Netflix)
- RxJS docs
- Jafar Husain: Async Programming in ES7 | JSConf US 2015

THANK YOU!