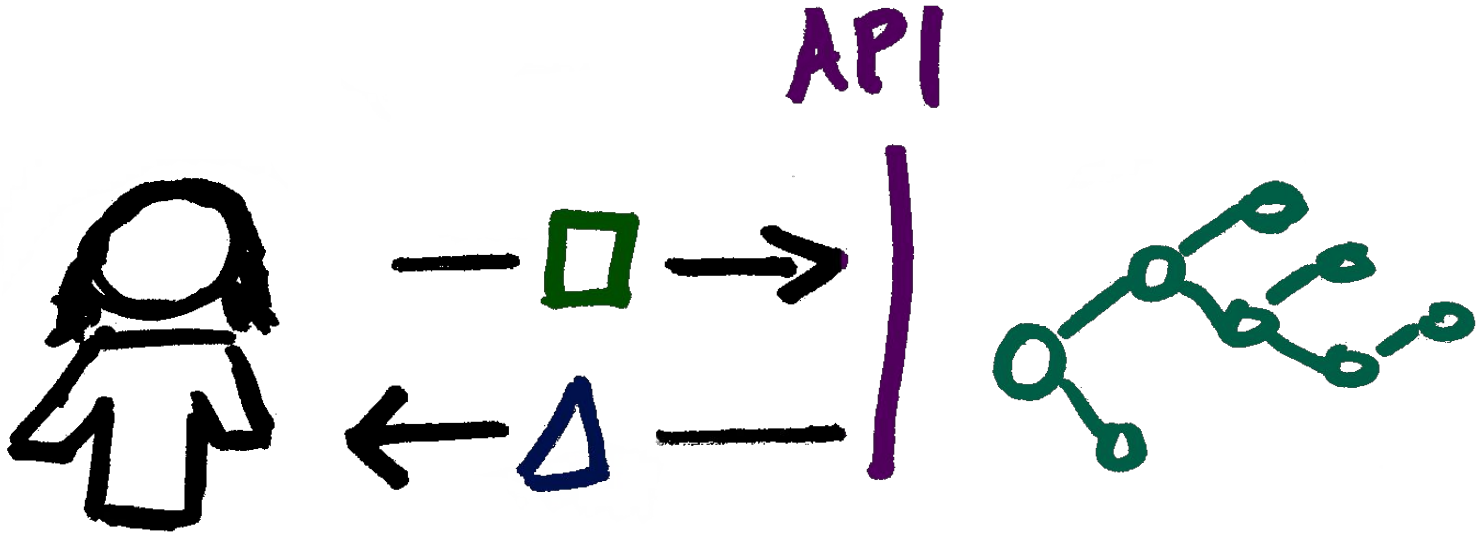
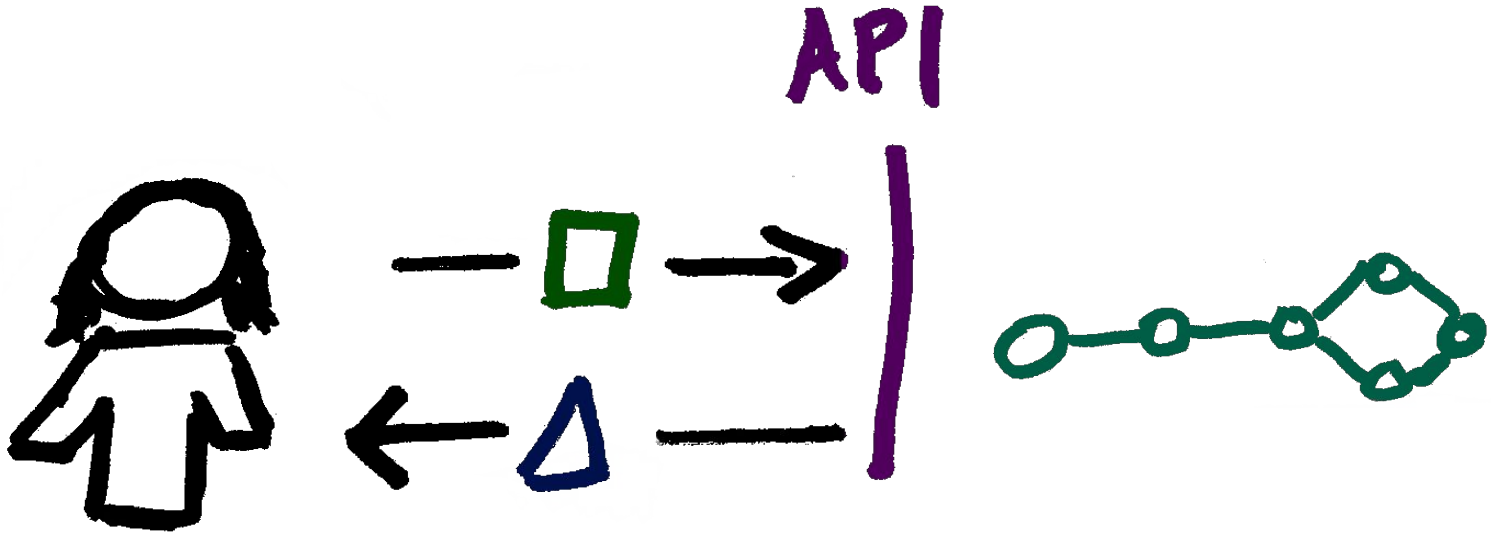
Refactoring Refactored with Ramda

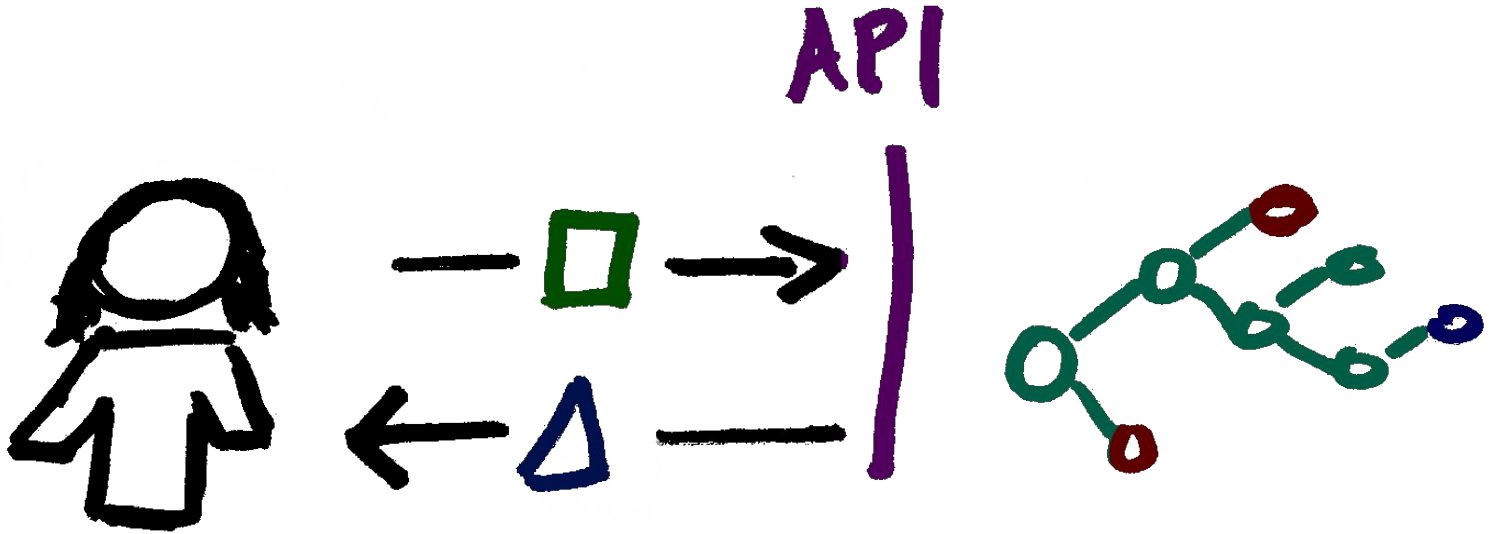
@mjpy3

/mjpy3/Presentations

What is Refactoring?


API

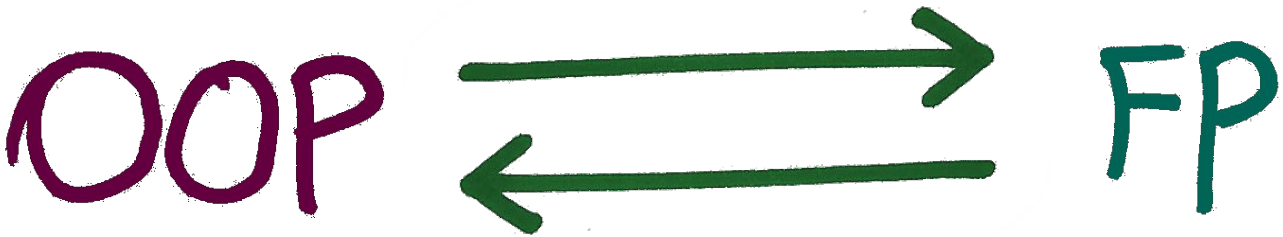

API

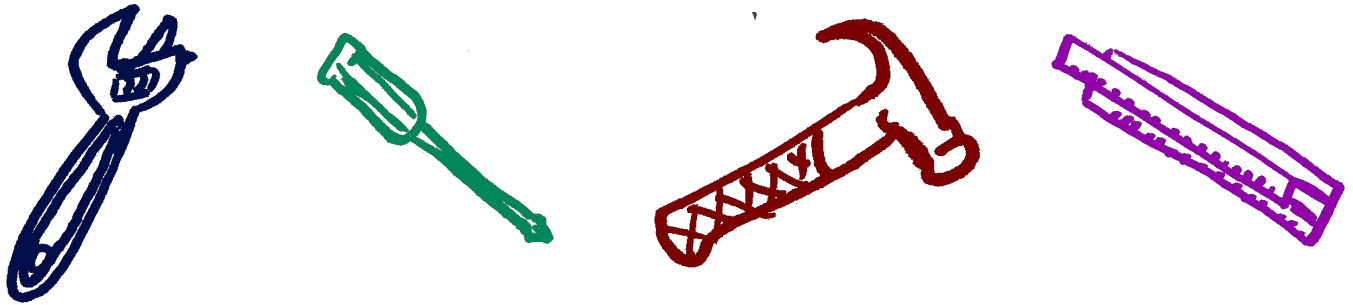

API

OOP

OOP

FP


OOP
FP



What is Ramda?

auto-curried

(a, b, c, d) -> e

auto-curried

~~(a,b,c,d) -> e~~

a -> (b -> (c -> (d -> e)))

auto-curried

~~(a,b,c,d) -> e~~

a -> b -> c -> d -> e

data-last

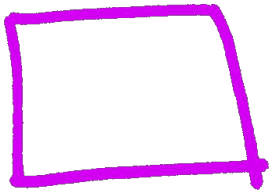
```
values.map(fn)
```

data-last

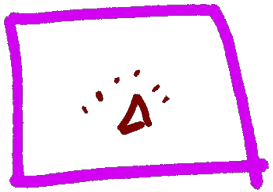
~~values.map(fn)~~

R.map(fn, values)

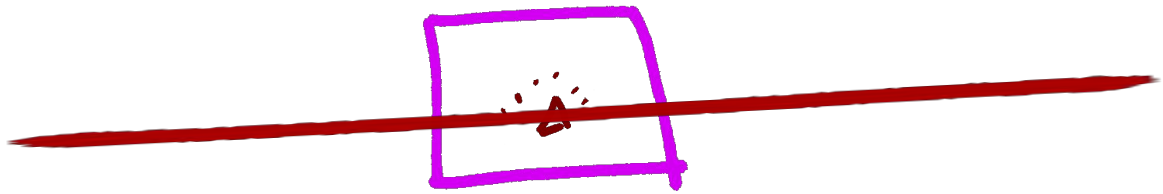
Immutability



Immutability



Immutability



JS-Aware

lodash/underscore

JS-Aware

lodash/underscore

promises

JS-Aware

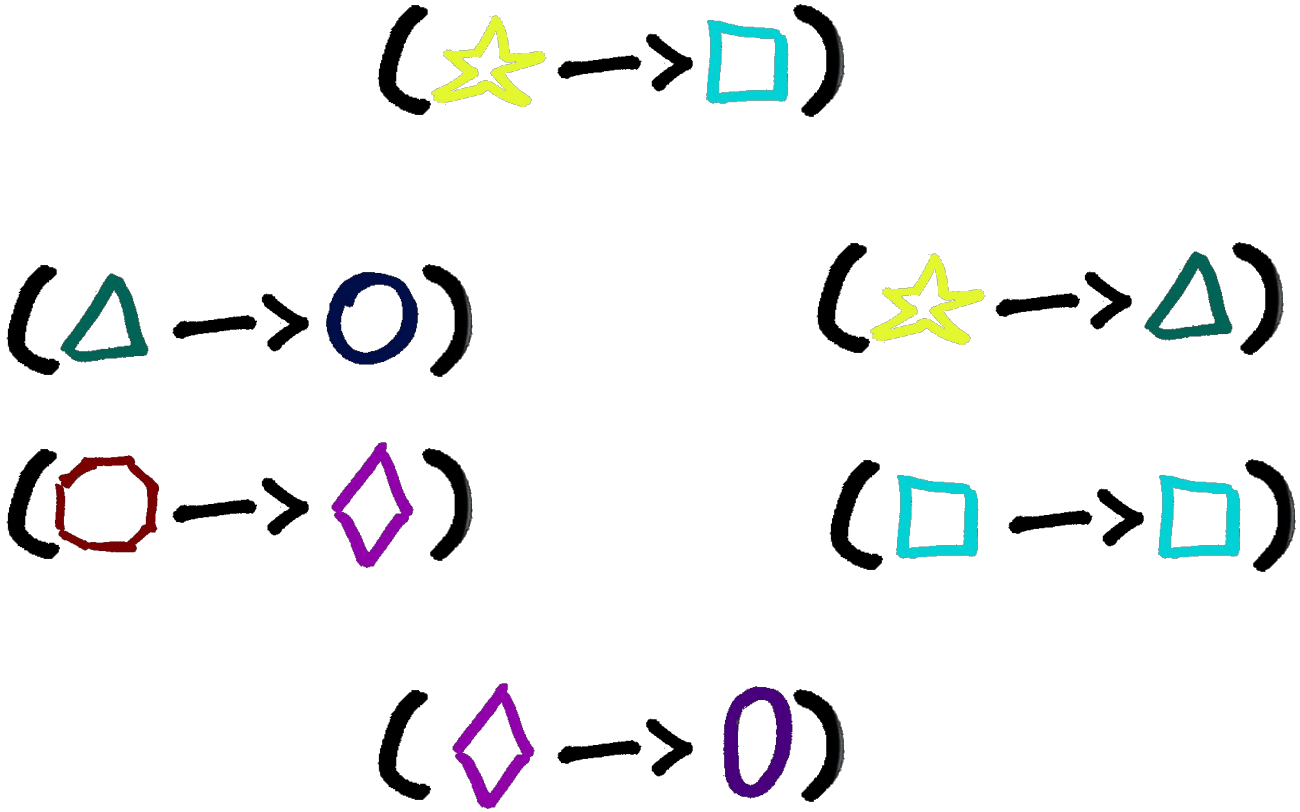
lodash/underscore

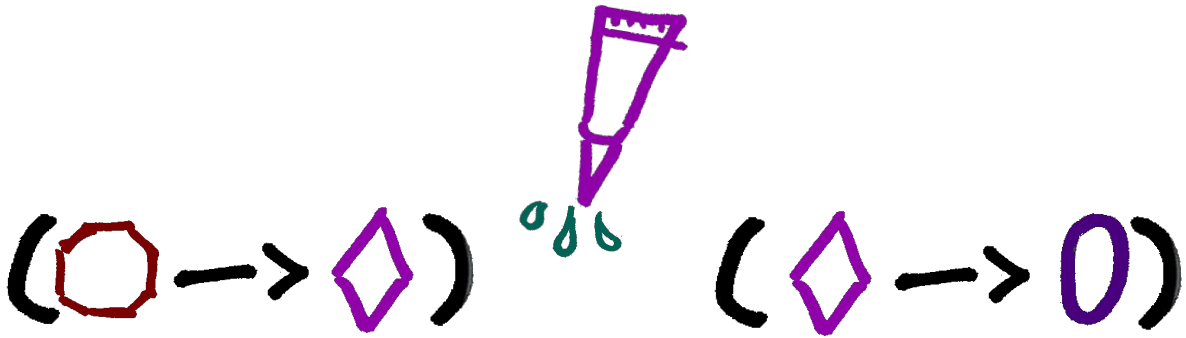
promises

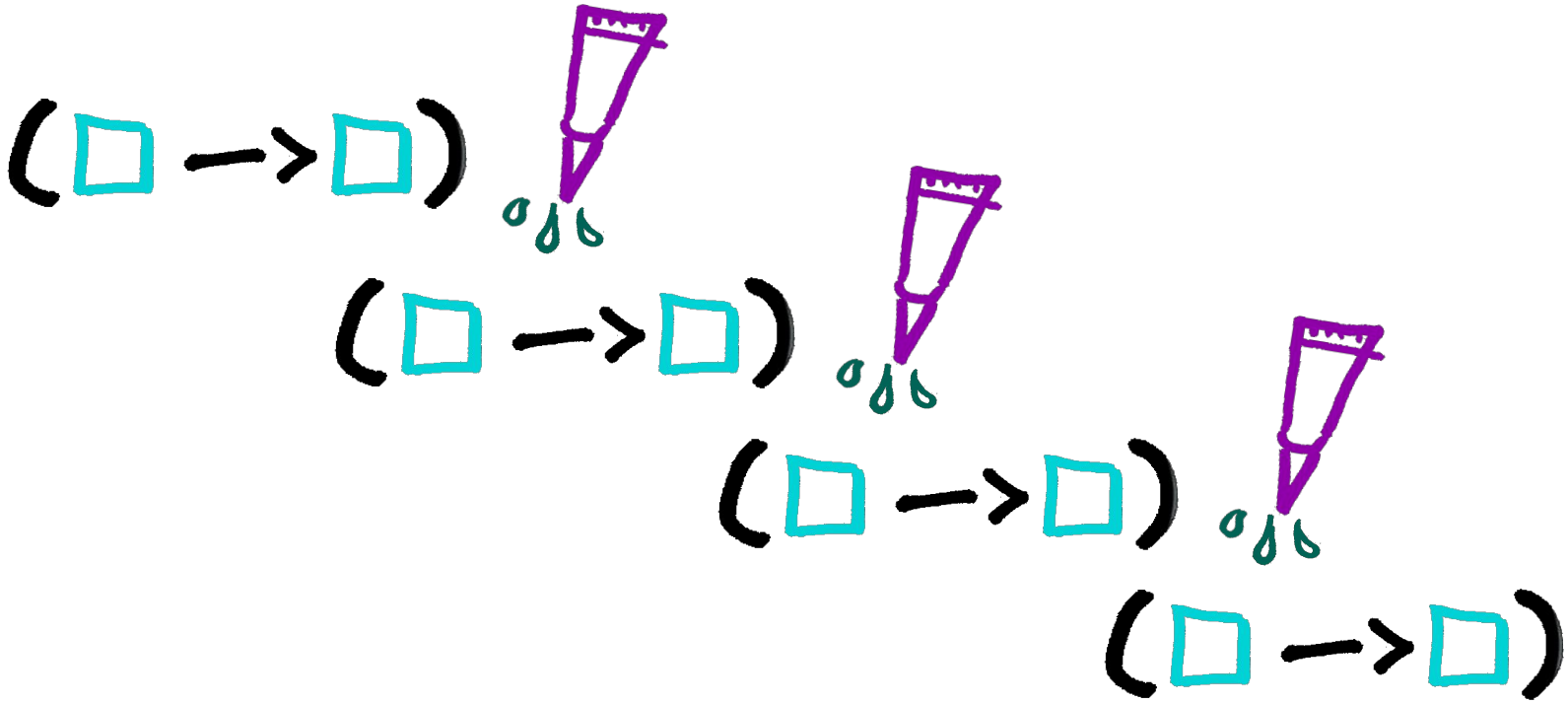
Defaults


"WHERE'S THE BEEF?"

compose & pipe







```
f3(f2(f1(x)))
```

```
R.compose(f3, f2, f1)
```

```
R.pipe(f1, f2, f3)
```

Example

Decisions

OOP has expressions Statements

FP has expressions

expressions

R.when

R.ifElse

R.cond

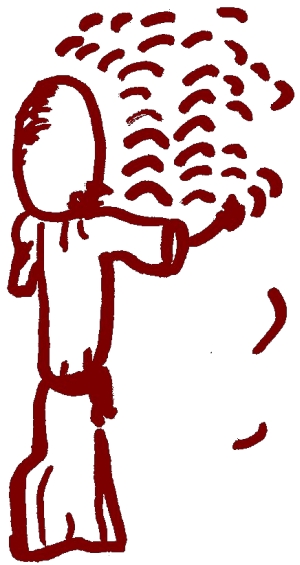
```
R.when =
  pred => trueFn => x =>
    pred(x) ? trueFn(x) : x;
```

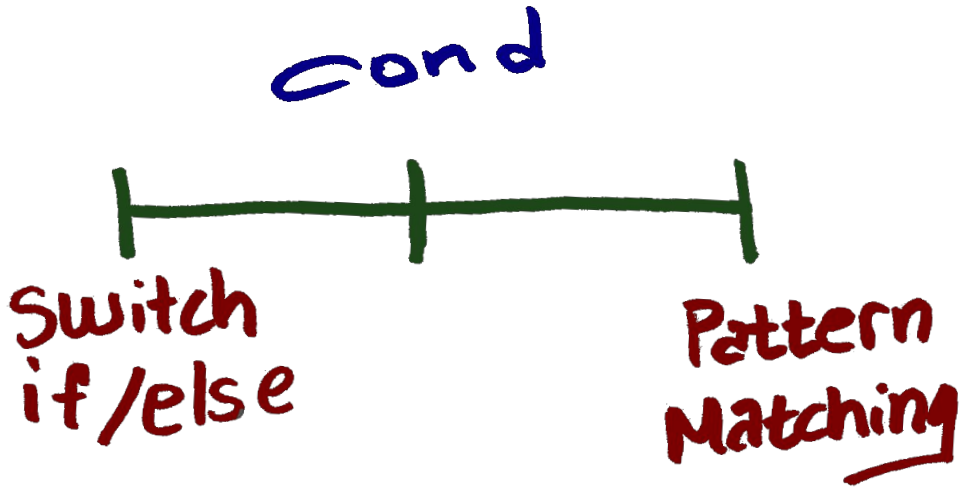
```
R.ifElse =
  pred => trueFn => falseFn => x =>
    pred(x) ? trueFn(x) : falseFn(x);
```

cond

cond

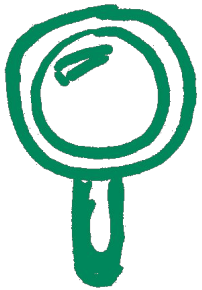
(remember? lisp)

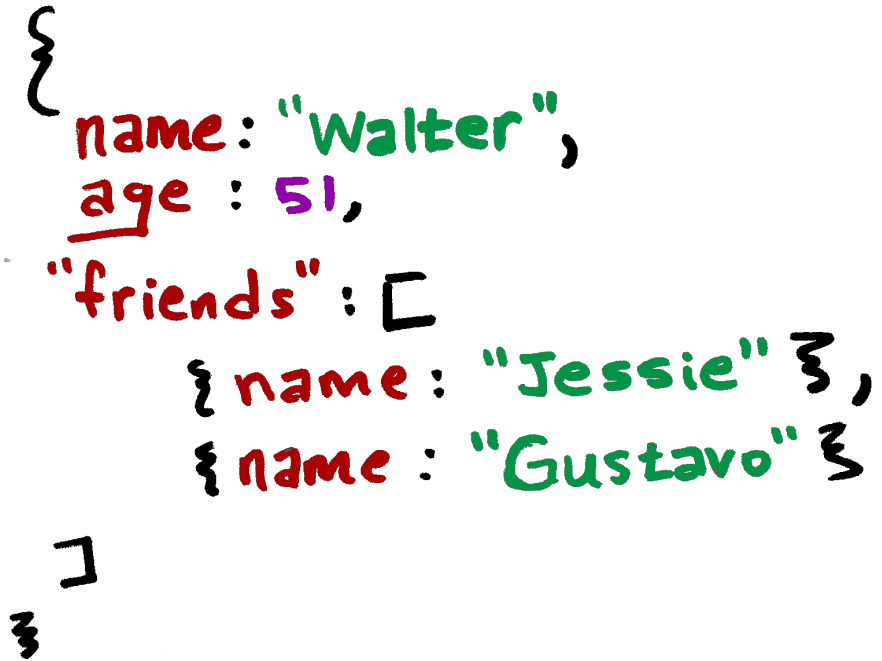


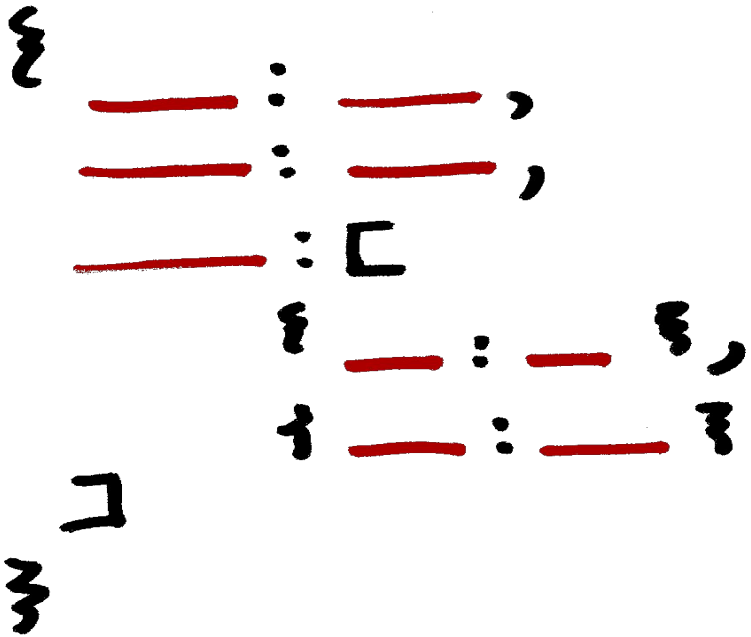

cond
Switch
if/else
Pattern
Matching

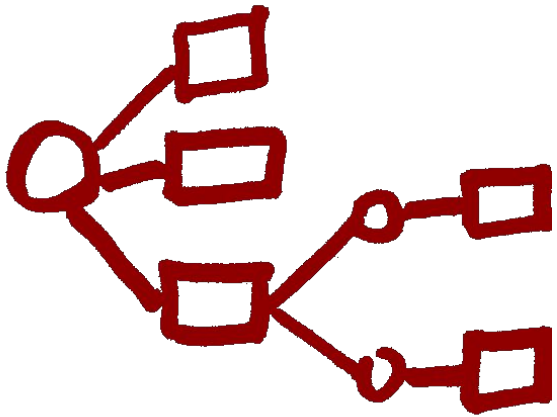
Example

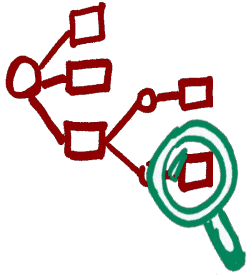
Lenses



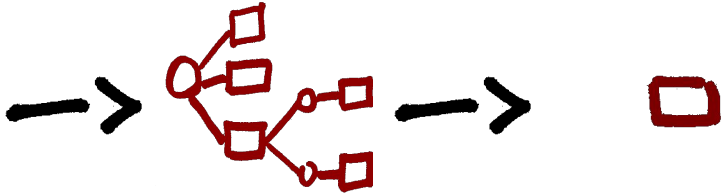

{
name: "Walter",
age : 51,
"friends": [
{name: "Jessie"},
{name : "Gustavo"}
]
}

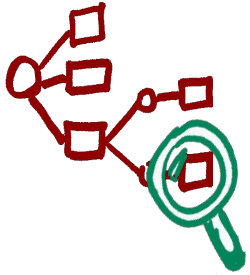




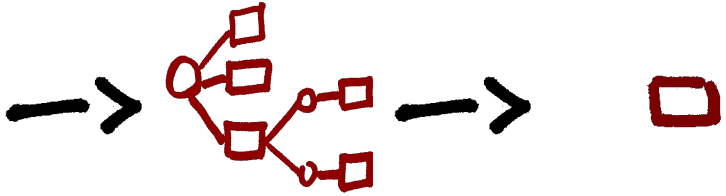


view

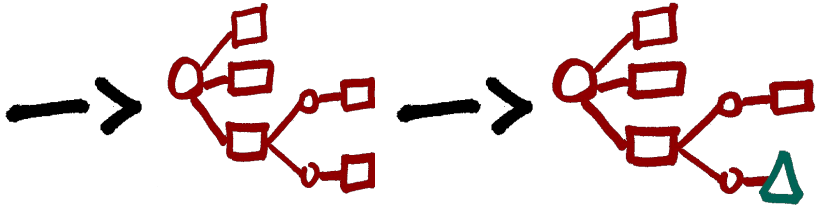


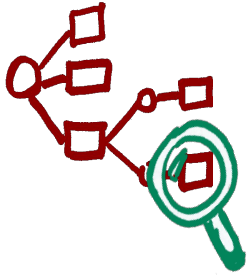


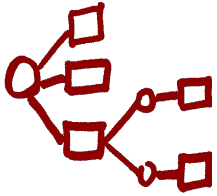
view

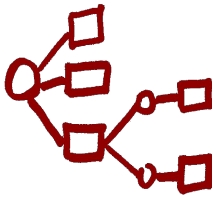
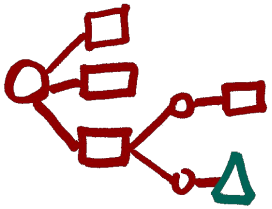


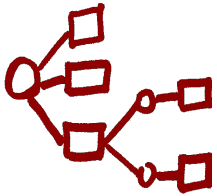
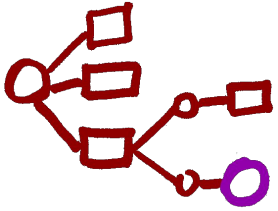
set

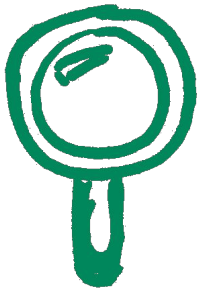
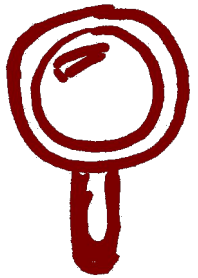




view  ->  -> 

set  ->  ->  -> 

over  -> ( -> ) ->  -> 

compose( , ) -> 

Example