Compose Yourself

Sacramento Web & Mobile Devs
6/27/2019

Setup

- NOT an algorithm challenge
- Solve problem as easily as possible
  - Use language's built in features
- Indulge me. Follow instructions even if they seem silly.
- Please stop when timer sounds

Form Team

- We’re all friends here
- We’re all beginners at some point
- Let’s help each other
- Newer devs partner with more experienced devs

Do Nothing

```
type doNothing = (x: any) => any
```

```
function doNothing(x) {
    return x;
}
```

```
const doNothing = x => x
```

Add 3 to a Number

```
type add3 = (n: number) => number
```

```
function add3(n) {
    return n + 3;
}
```

```
const add3 = n => n + 3
```

Double a Number

```
type doubleIt = (n: number) => number
```

```
function doubleIt(n) {
    return n * 2;
}
```

```
const doubleIt = n => n * 2
```

Uppercase a String

```
type toUpper = (s: string) => string
```

```
function toUpper(s) {
    return s.toUpperCase();
}

const toUpper = s => s.toUpperCase();
```

Append Exclamation to String

```
type emphasize = (s: string) => string
```

```
const emphasize = s => s + '!';
```

Append Question Mark

```
type what = (s: string) => string
```

Append Period

```
type end = (s: string) => string
```

Pattern

- ~~Hard coded data~~
- Use parameter to eliminate duplication
- Makes function more generic
- Introduces Abstraction over the value

Abstraction - Meanings

1. Difficult to understand, obscure, hidden

2. Hide details to simplify thinking

Append Any Character

```
type appendChar = (char: string, s: string) => string
```

Append Question Mark 2 Times

Append Exclamation 3 Times

Append Period 5 Times

Pattern

Add another parameter

Append Any Character N Times

```
type appendNTimes = (char: string, times: number, s: string) => string
```

Prepend Any Character N Times

```
type prependNTimes = (char: string, times: number, s: string) => string
```

```
prependNTimes :: string -> number -> string -> string
```

Prepend ‘W’ 3 times

```
type prependWWW = (s: string) => string

prependWWW :: string -> string
```

```
const prependWWW = prependNTimes(‘W’, 3);
```

Prepend N Times Redux

```
type prepend = (char: string, times: number, s: string) => string
```

```
prepend :: string -> number -> string -> string
```

```
type prepend = char: string => times: number => s: string => string
```

Functions Returning Functions

```
prepend :: char -> number -> string -> string
```

Takes 3 params

```
prepend  :: (char -> number -> string) -> string
```

Takes 1 param & returns function taking 2 params

```
prepend :: (char) -> ((number -> string) -> string)
```

Takes 2 parameters & returns function taking 1 param

```
prepend :: (char) -> (number) -> (string -> string)
```

Prepend In Steps (Part 1)

```
prepend :: char -> number -> string -> string
```

```
const prependW = prepend('W')
```

```
prependW :: number -> string -> string
```

Prepend In Steps (Part 2)

```
prependW :: number -> string -> string
```

```
const prependW3Times = prependW(3)
```

```
const prependW3Times = prepend('W')(3)
```

```
prependW3Times :: string -> string
```

Append Question Mark 2 Times

Add 3 to a Number 5 Times

Double a Number 10 Times

Pattern

- ~~Hard coded behavior~~
- Use parameter to eliminate duplication
- Pass behavior into the function
- Introduces Abstraction over the behavior

Execute a Numeric Function N Times

```
iterate :: (number -> number) -> number -> number -> number
```

```
iterate = (f) -> numTimes -> input -> output
```

Create Function that Executes a Numeric Function 2 Times with an Input Value of 42 (use Iterate)

```
iterate :: (number -> number) -> number -> number -> number

const doubleMeaning = iterate(42, 2);

doubleMeaning :: (number -> number) -> number
```

Reverse String

```
reverse :: string -> string
```

Count Spaces in String

```
spaceCount :: string -> number
```

Is Even?

```
isEven:: number-> bool
```

Make a LOUD! String

```
loud :: string -> string
```

```
const loud = (s: string) => emphasize(toUpper(s));
```

Reverse !DUOL

reverseLoud :: string -> string

```
const reverseLoud = s => reverse(emphasize(toUpper(s)));
```

```
const reverseLoud = s => reverse(loud(s));
```

Append ‘?’, Prepend ‘WWW’, Reverse, Uppercase

```
f :: string -> string
```

```
const f = s => toUpper(reverse(prependWWW(what(s))));
```

Pattern

- Nested function calls
- Use helper function to eliminate nesting

Composition - Meanings

1. Building larger structures out of smaller building blocks

2. “Glueing” functions together

Composition

```
const compose = (...functions) => (data) =>
    functions.reduceRight((composed, func) =>
        func(composed), data)

const pipe = (...functions) => (data) =>
  functions.reduce((piped, func) => func(piped), data)
```

Make a LOUD! String

```
loud :: string -> string
```

```
const loud = (s: string) => emphasize(toUpper(s));
```

```
const loud = compose(appendEx, toUpper);
```

Append ‘?’, Prepend ‘www’, Reverse, Uppercase

```
const f = compose(
  toUpper,
  reverse,
  prependWWW,
  appendQM,
);
```

```
const f = pipe(
  appendQM,
  prependWWW,
  reverse,
  toUpper,
);
```

Composition with Different Types

Does string have even number of spaces?

```
const evenSpaces = compose(isEven, countSpaces);
```

```
const evenSpaces = pipe(countSpaces, isEven);
```

What is Type Signature?

```
evenSpaces :: string -> boolean
```

What Happened to Number?

```
const evenSpaces = compose(isEven, countSpaces);
```

```
countSpaces :: string -> number

isEven ::                number -> boolean


evenSpaces :: string             -> boolean
```

Composition with Array Functions

```
map :: (a -> b) -> [a] -> [b]

filter :: (a -> boolean) -> [a] -> [a]

reduce :: (b -> a -> b) -> b -> [a] -> b
```

Map

```
map :: (a -> b) -> [a] -> [b]
```

Given [1, 2, 3]; Return array with each element doubled and summed with 3.

```
[1, 2, 3].map(doubleIt).map(add3);

[1, 2, 3].map(compose(add3, doubleIt));
```

Filter

```
filter :: (a -> boolean) -> [a] -> [a]
```

[‘hi there’, ‘I like code’, ‘web and mobile dev’]

Return array with elements containing more than two spaces.

```
array.filter(compose(gt2, countSpaces));
```

Put It All Together

[{ name: string, age: number, greeting: string, n: number }]

Return names of everyone older than 50.
Make name uppercase.
Prepend the greeting ‘n’ number of times.
Append ‘n’ number of exclamations.

BONUS Reverse name when age is even WITHOUT using an if statement.

Composition - Next Level

- How would you compose functions when the result of a function is not guaranteed?
- Ex: read name from database, make uppercase
- What happens if name doesn’t exist?