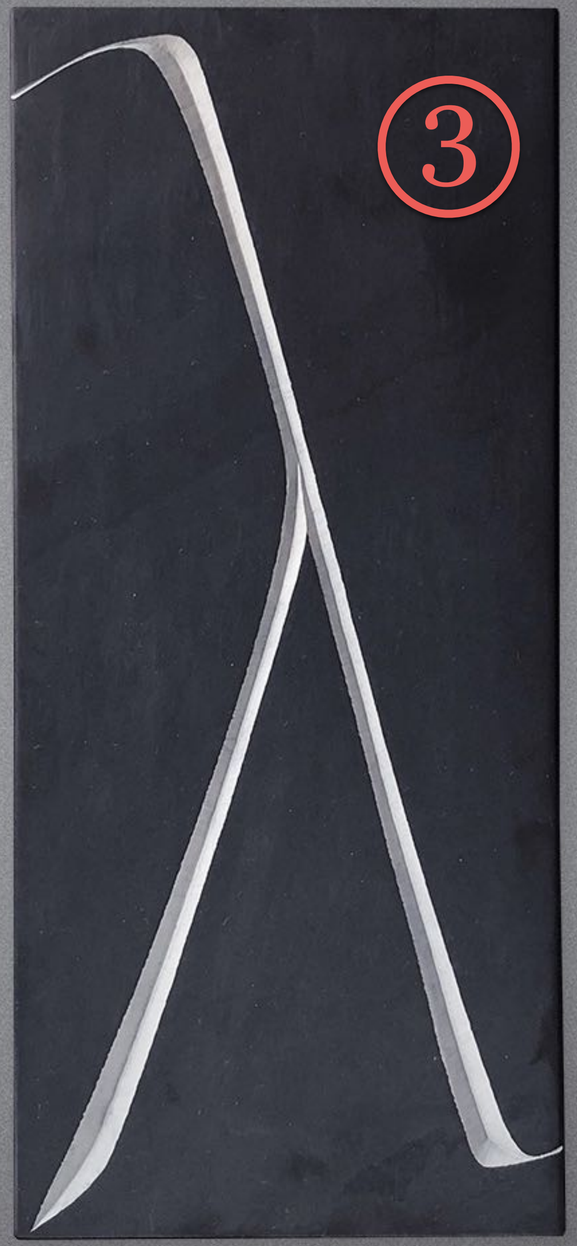


Practical JS stuff alert!

# FP Fundamentals in JS

*Pure Functions · Function Composition · Currying & Partial Application*

# Reminder: FP in a 🌰 Nutshell

- 🎶 Functions everywhere (naturally)
- 🎵 Composition (small pieces → larger constructs)
- 💘 Purity (input → output, no effects)
- 💞 Equational reasoning (call & value interchangeable)
- 💟 First-class & higher-order (code uses & produces code)
- 💗 Currying & partial application (general-purpose → specific)
- 💎 Immutability (foolproof, supports equational reasoning)
- λ Mathematical (lambda calculus, category theory; law-based)

# Pure 💘 Functions

**input → output, no side effects**

# Pure Functions 💘

- **Same input for same output, always**
  - Deterministic (no randomness / unpredictability)
  - Stateless (results do not depend on something that can change)
  - Entirely defined as a map from input(s) (zero or more) to output
- **No "observable" side effects**
  - No changing an object others might have reference to
  - No reassignment of a variable outside function scope
  - No manipulation of the "external world" (files, network, terminal, I/O)
  - No calling other code which does the above

# Game: Pure 💘 or Impure 💔?

# Pure! 💘

```
function increment (number) {
    return number + 1
}
```

same input means same output always

# Impure 💔

```
function grow (person) {
    person.age = person.age + 1
    return person
}
```

mutates object that others
might have or get a reference to

# Impure 💔

```
function yellLog(str) {
    console.log(str + '!')
}
```

has an observable side effect
(logs to the console)

# Pure Function 💘, Impure Body 💔

```
function foobar (rounds) {
    const obj = {}
    for (let i = 0; i < rounds; i++) {
        obj[i] = rounds - i
    }
    return obj
}
```

always creates a new object...
...guaranteed same output for a given input...

**benefits during *use*, but not during *implementation***

# Impure 💔

```
let lucky = 4
function getMore () {
    return lucky + 1
}

log(getMore()) // 5
lucky = 0
log(getMore()) // 1
```

relies on external state,
cannot guarantee same output
for same input

# Impure 💔

```
function luckyNum (min, max) {
    return Math.random() * (max - min) + min
}
```

nondeterministic, cannot
guarantee same output for same input

# Pure! 💘

```
const MAX_VAL = 99
function lowbar (height) {
    return height > MAX_VAL
      ? MAX_VAL
      : height
}
```

same input yields same output...
...but what about external variable?

**MAX_VAL is `const`, only way this func can change is if we edit the code (it cannot change during use).**

# Pure! 💘

```
function secret (message) {
    return function () {
        return message
    }
}
```

secret returns a function.
...is it the always "same" function for a given input?

**"Same" output in terms of purity does not mean same *memory* – just *equivalent value*.**

# Why?

# Pure Functions 💘

- ⍰ **Afford you strong reasoning capabilities**
  - Can move around, invoke anywhere, and nothing will break
  - Do not have to think about how you got to a pure function – *only* inputs and outputs. No need to replay entire program in your head!
- 💯 **Very easy to test**
  - Put stuff in, get something out. If it maps as you intend, it's working.
- 🎵 **Very easy to compose**
  - Glue pure functions to other pure funcs as you wish, they chain together without causing any issues.

# WORKSHOP