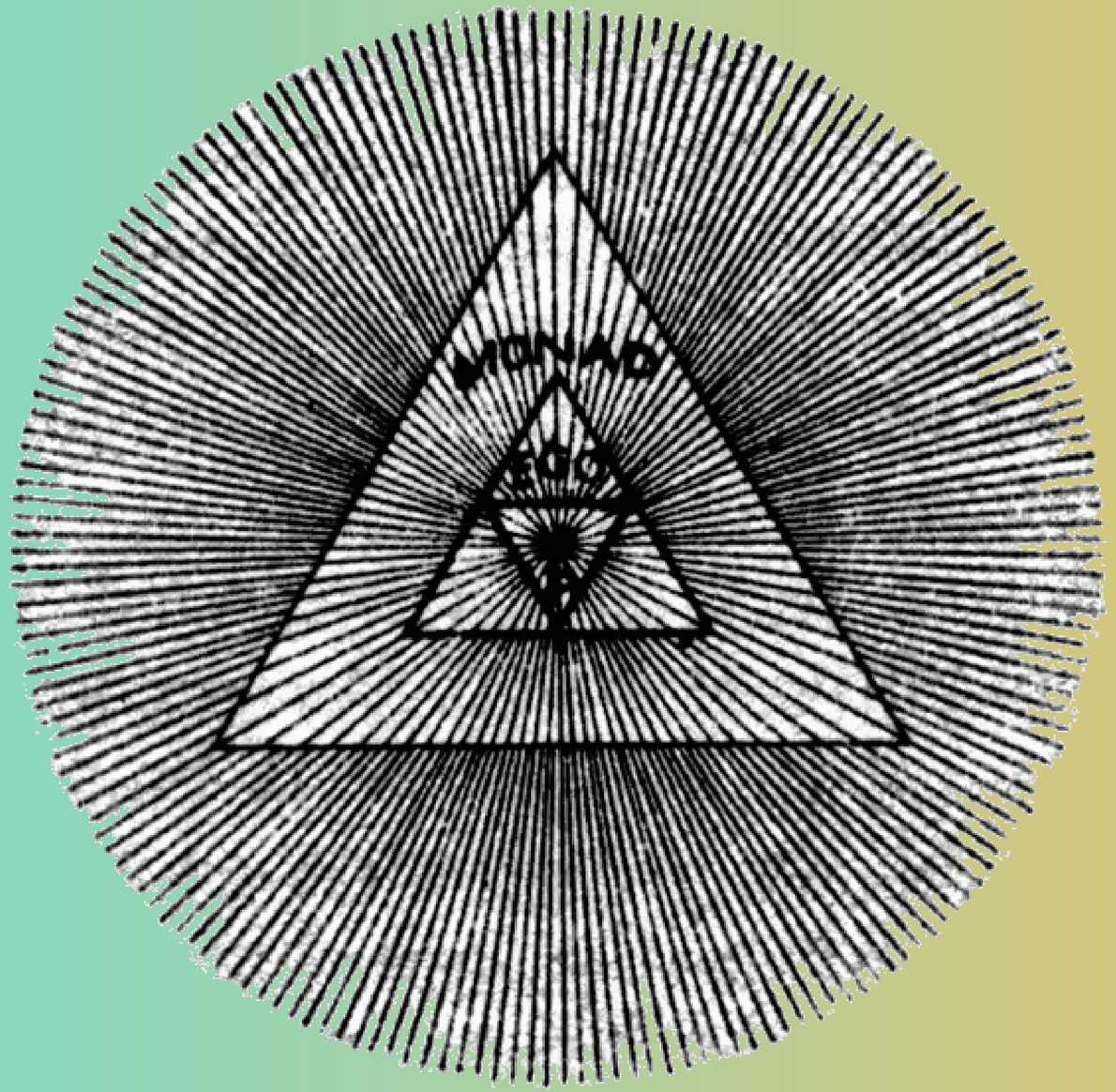
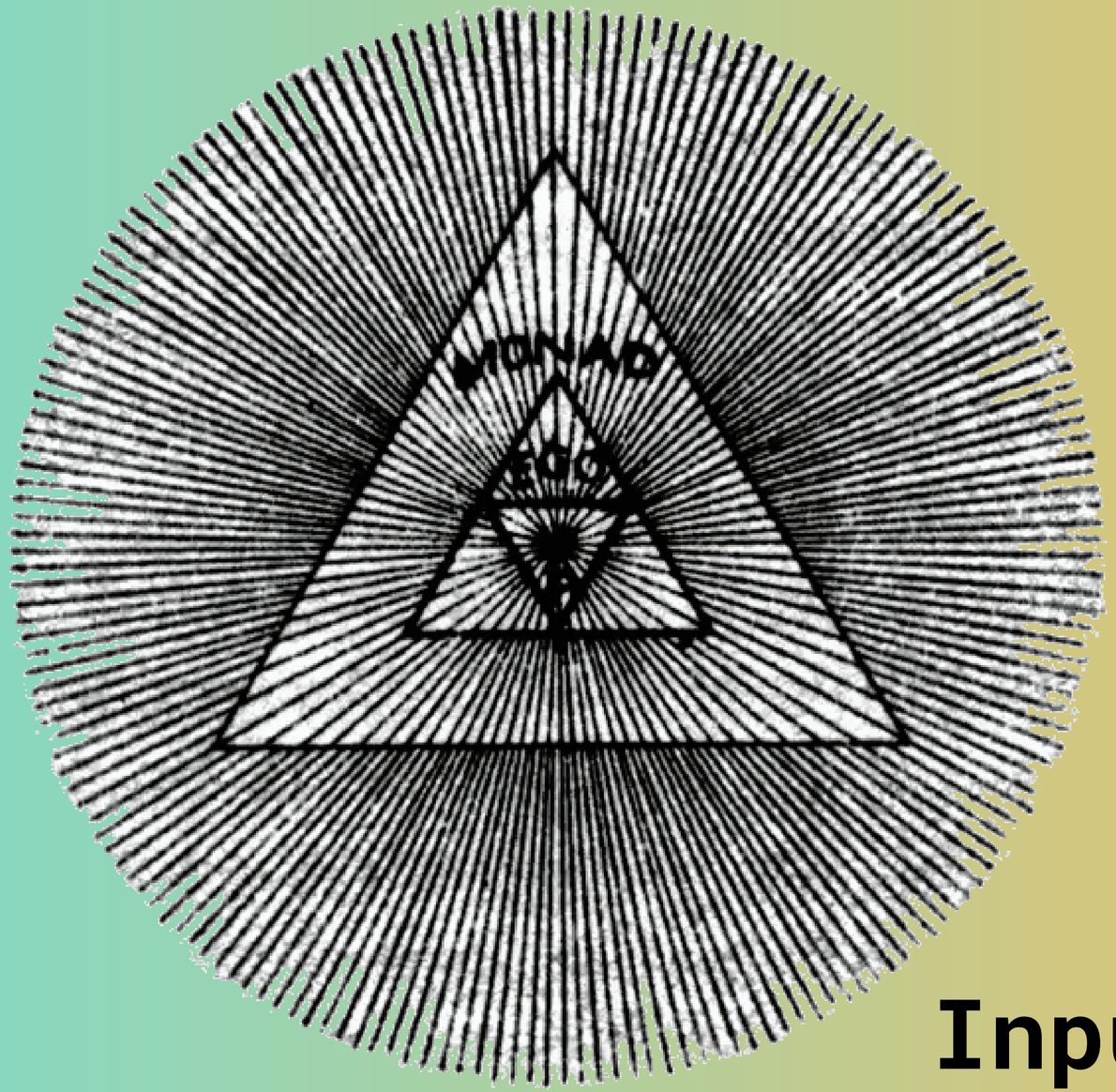
unctional
paradigm

Grupo 2


Learn to program
Make recursive function
Y Combinator
Learn to program
Make recursive function
No exit condition
No exit condition





Input/Output

IOMonad.js

```
export class IOMonad {
  #effect;

  constructor(effect) {
    this.#effect = effect;
  }

  static of(value) {
    return new IOMonad(() => value);
  }

  map(f) {
    return new IOMonad(() => f(this.eval()));
  }

  chain(f) {
    return new IOMonad(() => f(this.eval()).eval());
  }

  eval() {
    return this.#effect();
  }
}
```

demo.js

```
// given input will be: [1, 2, 3, 4]
const monad = new IOMonad(() => prompt());

// curry reduce function
const combine = (nums) => _.reduce(
  nums,
  (acc, nxt) => acc + nxt,
  0,
);

// sum will be 1 + 2 + 3 + 4 = 10
const sum = monad
  .map(JSON.parse) // returns a new IOMonad
  .map(combine)    // returns a new IOMonad
  .eval();         // prompt will be executed here
```

purity.js

```
const pureF1 = (f1, f2, f3) => {
  const monad = new IOMonad(() => prompt());

  return monad
    .map(f1)
    .map(f2)
    .map(f3);
};

const pureF2 = () => {
  const monad = new IOMonad(() => prompt());

  return monad.map(JSON.parse);
};

const impureF = () => {
  const m1 = pureF1(transform1, transform2, transform3);
  const m2 = pureF2();

  m1.eval();
  m2.eval();
}
```

Extra: sistema de tipos ayuda mucho

IOMonad.js

```
export class IOMonad {
  #effect;

  constructor(effect) {
    this.#effect = effect;
  }

  static of(value) {
    return new IOMonad(() => value);
  }

  map(f) {
    return new IOMonad(() => f(this.eval()));
  }

  chain(f) {
    return new IOMonad(() => f(this.eval()).eval());
  }

  eval() {
    return this.#effect();
  }
}
```

¿Qué es 'f'?

IOMonad.js

```js
/**
 * @template T
 * @callback Effect
 * @returns {T}
 */

/**
 * @template T
 */
export class IOMonad {
  /** @type {Effect<T>} */
  #effect;

  /**
   * @param {Effect<T>} effect
   */
  constructor(effect) {
    this.#effect = effect;
  }

  /**
   * @template A
   * @param {A} value
   * @returns {IOMonad<A>}
   */
  static of(value) {
    return new IOMonad(() => value);
  }

  /**
   * @template A
   * @param {(value: T) => A} f
   * @returns {IOMonad<A>}
   */
  map(f) {
    return new IOMonad(() => f(this.eval()));
  }

  /**
   * @template A
   * @param {(value: T) => IOMonad<A>} f
   * @returns {IOMonad<A>}
   */
  chain(f) {
    return new IOMonad(() => f(this.eval()).eval());
  }

  /**
   * @returns {T}
   */
  eval() {
    return this.#effect();
  }
}
```

IOMonad.ts

```ts
export type Effect<Type> = () => Type;

export class IOMonad<T> {
  #effect: Effect<T>;

  constructor(effect: Effect<T>) {
    this.#effect = effect;
  }

  static of<A>(value: A): IOMonad<A> {
    return new IOMonad(() => value);
  }

  map<A>(f: (value: T) => A): IOMonad<A> {
    return new IOMonad(() => f(this.eval()));
  }

  chain<A>(f: (value: T) => IOMonad<A>): IOMonad<A> {
    return new IOMonad(() => f(this.eval()).eval());
  }

  eval(): T {
    return this.#effect();
  }
}
```