JS Funcional

Grupo 2:
Ariel Camhi
Nicolás Rosenberg
Dan Ustilovsky

Fundamentos programación funcional

- Funciones como unidad de composición
- Control de flujo vía recursión
- Objetos inmutables

Funciones como unidad de composición

```
const applySpecialPlay = (name) => {
    ...
};
const printWinner = (playerName) => {
    ...
};
const ingresar_jugada = (name, score, shots) => {
    ...
};
const init_game = (players) => {
    ...
};
const Y = (f) => ((x) => x(x))((x) => f((y) =>
x(x)(y)));
const gameLogicGen = f => ((playersPoints) => {
    ...
);
const play_game = (players) => {
    ...
};
```

Control de flujo vía recursión

```
const gameLogic = (playersPoints) => {
  [actual, ...rest] = playersPoints;
  [playerName, playerScore] = actual;
  const newPlay = JSON.parse(
    readline.question(`${playerName} ingrese su jugada >`)
  );
  const playersStatus = [
    ...rest,
    [playerName, ingresar_jugada(playerName, playerScore,
newPlay)],
  ];
  playersStatus.some((x) => x[1] === 0)
    ? printWinner(playerName)
    : gameLogic(playersStatus);
};
```

Y Generator

```
const Y = (f) => ((x) => x(x))((x) => f((y) => x(x)(y)));

const gameLogicGen = f => ((playersPoints) => {
  [actual, ...rest] = playersPoints;
  [playerName, playerScore] = actual;
  const newPlay = JSON.parse(readline.question(`${playerName} ingrese su jugada >`));
  const playersStatus = [
    ...rest,
    [playerName, ingresar_jugada(playerName, playerScore, newPlay)],
  ];
  playersStatus.some((x) => x[1] === 0)
    ? printWinner(playerName)
    : f(playersStatus);
});

const play_game = (players) => {
  const playersPoints = init_game(players);
  console.log(`Juego inicializado con los jugadores ${players.join(", ")}.`);
  Y(gameLogicGen)(playersPoints);
};
```

Objetos inmutables

```
const gameLogicGen = f => ((playersPoints) => {
 [actual, ...rest] = playersPoints;
 [playerName, playerScore] = actual;
 const newPlay =
JSON.parse(readline.question(`${playerName} ingrese su
jugada >`));
 const playersStatus = [...rest,
    [playerName, ingresar_jugada(playerName, playerScore,
newPlay)],
 ];
 playersStatus.some((x) => x[1] === 0)
    ? printWinner(playerName)
    : f(playersStatus);
});
```

Currying

```
const applyDB = (play) => {
  return play === "DB" ? [1, 50] : play;
};

const applySB = (play) => {
  return play === "SB" ? [1, 25] : play;
};
```

Currying

```
const applySpecialPlay = (name, points, play) => {
  return play === name ? [1, points] : play;
};
```

Currying

```
const applySpecialPlay = (name) => {
 return (points) => {
   return (play) => {
     return play === name ? [1, points] : play;
   };
 };
};

const applyDB = applySpecialPlay("DB")(50);
const applySB = applySpecialPlay("SB")(25);
```

Chaining

```
const _ = require("lodash");
...
const ingresar_jugada = (name, score, shots) => {
 const bullAppliedShots = shots.map((shot) => applySB(applyDB(shot)));
 const multipliedPoints = bullAppliedShots.map((shot) => shot[0] * shot[1]);
 const turnScore = multipliedPoints.reduce((x, y) => x + y);
 const updatedScore = Math.abs(score - turnScore);
 console.log(`${name} queda con ${updatedScore} puntos.`);
 return updatedScore
};
```

Chaining

```
const _ = require("lodash");
...
const ingresar_jugada = (name, score, shots) => {
 const turnScore = _
    .chain(shots)
    .map((shot) => applySB(applyDB(shot)))
    .map((shot) => shot[0] * shot[1])
    .reduce((accumulator, currentValue) => accumulator + currentValue)
    .value();
 const updatedScore = Math.abs(score - turnScore);
 console.log(`${name} queda con ${updatedScore} puntos.`);
 return updatedScore
};
```