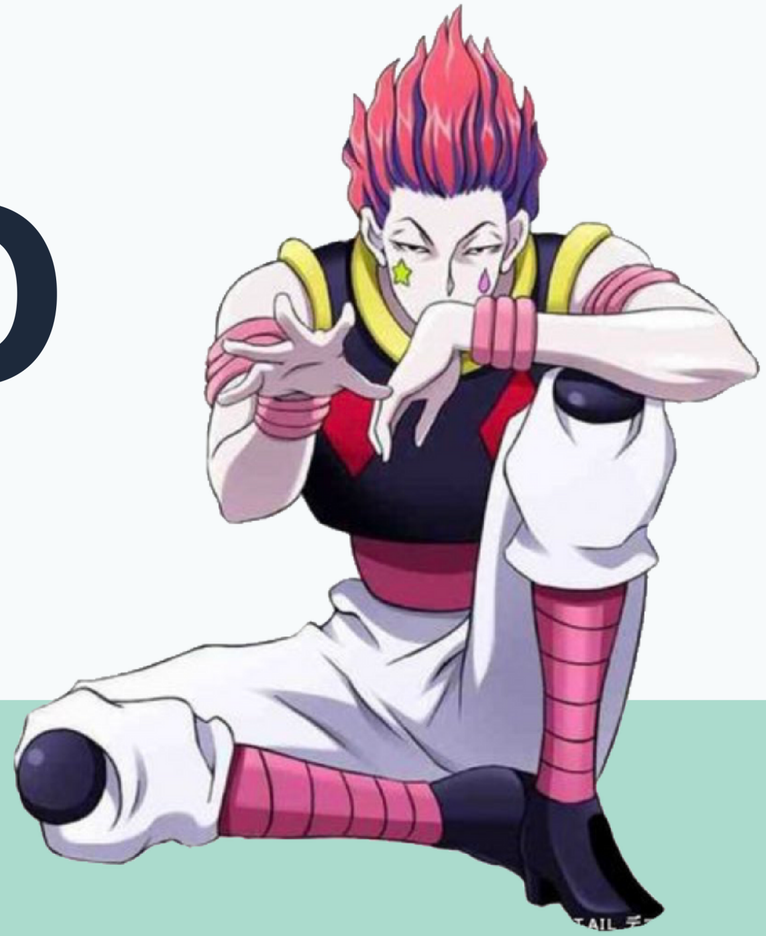
JS Funcional

Grupo 4

Supuestos

- **¿?** y **¡!** se consideran como caracteres cualquiera.
- Cada párrafo y frase terminan con un punto.
- El texto a transformar está "relativamente bien escrito".
- No hay puntos seguidos del estilo "...".
- El texto termina con un punto final, sin espacios ni saltos de línea después del punto.

DEMO



Funciones

```
// Cada frase debe comenzar con n espacios en blanco (después de un punto seguido)
const addSpacesFollowedDot = (text, n) =>
  cleanText2(text)
    .split(". ")
    .map((sentence, index) =>
      index > 0
        ? (sentence = "." + " ".repeat(n) + sentence)
        : (sentence = sentence)
    )
    .join("");

// Cada párrafo debe estar separado por n líneas (después de un punto aparte)
const addLinesSeparateDot = (text, n) =>
  cleanText1(text)
    .split(".\n")
    .map((sentence, index) =>
      index > 0
        ? (sentence = "." + "\n".repeat(n + 1) + sentence)
        : (sentence = sentence)
    )
    .join("");
```

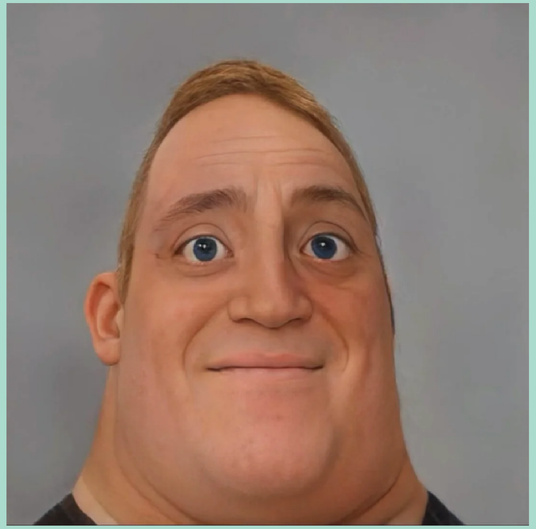
Funciones

```
// El ancho del texto debe ser a lo más n (sin cortar palabras)
const addMaxWidth = (text, n) => {
  const textWidthReducer = (accumulator, new_word) => {
    if (new_word === "\n") {
      return [...accumulator, ""];
    } else {
      return _.last(accumulator).length + new_word.length > n
        ? [...accumulator, new_word]
        : [..._.dropRight(accumulator), _.last(accumulator) + new_word];
    }
  };

  const wrappText = _.flow([
    (text) => splitStringWithSingleSpaces(text),
    (words) => _.reduce(words, textWidthReducer, [""]),
  ]);

  return wrappText(text).join("\n");
};
```

Funciones

```
// Cada párrafo debe tener n espacios de sangría
const addIndentation = (text, n) =>
  text
    .split(".\n")
    .map((paragraph, index) =>
      index > 0
        ? paragraph.replace(/^\n*/, "$&" + " ".repeat(n))
        : " ".repeat(n) + paragraph
    )
    .join(".\n");
```

Funciones

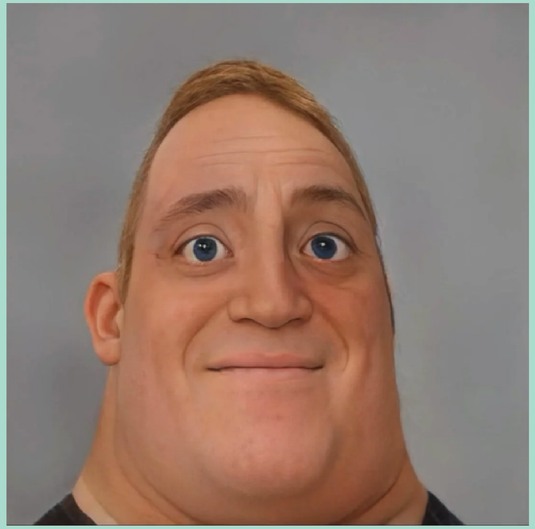


```
// Se ignoran los párrafos según la función comparadora
const ignoreParagraphs = (comparator_function, text, n) =>
  _.chain(text)
    .split(".\n")
    .map((paragraph, index) =>
      index < text.split(".\n").length - 1
        ? paragraph + ".\n"
        : paragraph
      )
    .map((paragraph) => paragraph.split(/(?=[\.])/))
    .filter((paragraph_splitted) =>
      comparator_function(paragraph_splitted.length, n)
    )
    .map((paragraph_splitted) => paragraph_splitted.join(""))
    .join("")
    .value();

const ignoreParagraphsCurried = _.curry(ignoreParagraphs);
```

Funciones

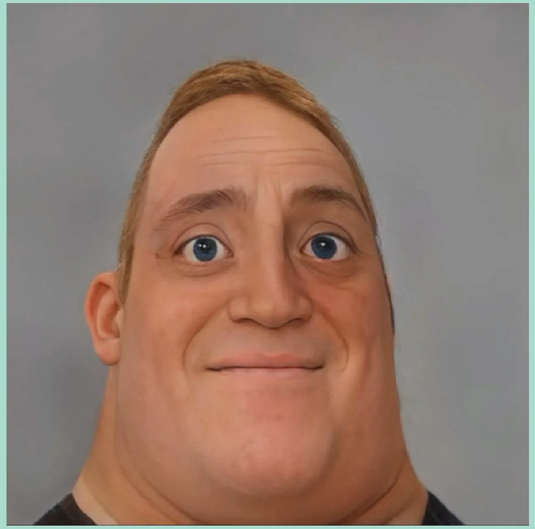
```
const ignoreParagraphsCurried = _.curry(ignoreParagraphs);

// Se ignoran los párrafos que tienen menos de n frases
const ignoreShortParagraphs = ignoreParagraphsCurried(
  (paragraph_length, n) => paragraph_length - 1 >= n
);

// Se ignoran los párrafos que tienen más de n frases
const ignoreLongParagraphs = ignoreParagraphsCurried(
  (paragraph_length, n) => paragraph_length - 1 <= n
);
```

Funciones



```
// Cada frase debe aparecer en párrafo aparte
const addNewParagraphEachLine = (text) =>
  _.chain(text)
    .split("\n")
    .join(" ")
    .split(". ")
    .filter((paragraph) => paragraph != "")
    .map((paragraph) => paragraph.replace(".", ""))
    .map((paragraph) => paragraph + ".")
    .join("\n")
    .value();
```

Funciones



```
// Solo las primeras n frases de cada párrafo
const firstPhrasesEachParagraph = (text, n) =>
  _.chain(text)
    .split(".\n")
    .map((paragraph) =>
      paragraph
        .split(".", n)
        .filter((paragraph) => paragraph != "")
        .join(".")
    )
    .filter((paragraph) => paragraph != "")
    .map((paragraph) => paragraph + ".")
    .join("\n")
    .value();
```

Compose Function

```
// inspired by: const S = f => g => x => f(x)(g(x))
const composeFunction = (functions) => (text, n_array) =>
  functions.reduce(
    (acc, f, current_index) => f(acc, n_array[current_index]),
    text  // initial value
  );
```

```
const buttonClick = () => {
  const text = document.getElementById("text").value;
  const result = document.getElementById("result");
  const { functions_selected, n_for_functions } = getFunctionsAndNSelected();
  const transformText = composeFunction(functions_selected);
  result.innerHTML = transformText(text, n_for_functions);
};
```

Conclusiones

Trabajar en grupo se hace más sencillo

Código más compacto

Más sencillo de testear

Alta modularización y poco acoplamiento

Alta Mantenibilidad

JS Funcional

Grupo 4

```
// Trim spaces of every "/n" and replace "\n...\n" with a single "/n"
const cleanText1 = (text) =>
  text.replace(/ +\n/g, "\n").replace(/\n +/g, "\n").replace(/\n+/g, "\n"
);

// If theres a dot followed by a lot of spaces, replace it with a dot
// followed by one space
const cleanText2 = (text) =>
  text.replace(/\.([^\n])/g, ". $1").replace(/\. +/g, ". ");

const splitStringWithSingleSpaces = (string) =>
  string
    .split(/(\s+)/)
    .map((word) => (word.match(/\s+/) ? word.split("") : word))
    .flat();
```