PROGRAMACIÓN FUNCIONAL

Grupo 6

Cada frase debe comenzar con *n* espacios en blanco (después de un punto seguido)

```
const blankSpaces = (text, n) => {
    const newText = text
        .replace(/\. +/g, (match) => '.' + ' '.repeat(n))
        .replace(/ +\n/g, '\n');

    return newText
}
```

Cada párrafo debe estar separado por *n* líneas (después de un punto aparte)

```
const paragraphSpacing = (text, n) => {
    const newText = text
        .replace(/\.\s+\n+/g, (match) => '.' + '\n'.repeat(n + 1))

    return newText
}
```

El ancho del texto debe ser a lo más *n* (sin cortar palabras)

```javascript
const maxWidth = (text, n) => {
    let newText = ''
    let word = ''
    let lineCount = 0
    Array.from(text).forEach(char => {
        if (char != '\n'){
            lineCount += 1;
        }
        if (char == ' ' || char == '.') {
            if (lineCount >= n){
                newText += '\n';
                lineCount = word.length + 1;
            }
            newText += word + char;
            word = '';
        }
        else if (char == '\n') {
            lineCount = 0;
            newText += char;
        }
        else {
            word += char;
        }
    })
    return newText
}
```

```javascript
const limitWidth = (text, n) => {
    const paragraphDistance = text.match(/\n+/g)[0]
    const newText = text
        .split(/\n+/g)
        .map(paragraph => {
            const words = paragraph.split(' ')
            let newParagraph = ''
            let line = ''
            words.forEach((word) => {
                if (line.length + 1 + word.length < n) {
                    line = line != '' ? `${line} ${word}` : word
                } else {
                    newParagraph += line + '\n'
                    line = word
                }
            })
            newParagraph += line
            return newParagraph
        })
        .join(paragraphDistance)

    return newText
}
```

Cada párrafo debe tener *n* espacios de sangría

```
const sangria = (text, n) => {
    let newText = ' '.repeat(n)
    Array.from(text).forEach(char => {
        if (newText[newText.length-2]=='.' && newText[newText.length-1]=='\n' && char == '\n') {
            newText += char + ' '.repeat(n);
            return
        }
        newText += char;
    })
    return newText
}
```

```
const identation = (text, n) => text.replace(/^/gm, ' '.repeat(n))
```

Se ignoran los párrafos que tienen menos de *n* frases

```
const ignoreParagraphLess = (text, n) => {
    const paragraphDistance = text.match(/\n+/g)[0]
    const newText = text
        .split(/\n+/g)
        .filter(paragraph => countPhrases(paragraph) >= n)
        .join(paragraphDistance)

    return newText
}
```

Se ignoran los párrafos que tienen más de *n* frases

```
const ignoreParagraphMore = (text, n) => {
    const paragraphDistance = text.match(/\n+/g)[0]
    const newText = text
        .split(/\n+/g)
        .filter(paragraph => countPhrases(paragraph) <= n)
        .join(paragraphDistance)

    return newText
}
```

```javascript
const countPhrases = (paragraph) => {
    const numPhrases = paragraph
        .split(/\. +|\? +|\! +/g)
        .filter(phrase => phrase.trim() != '')
        .length

    return numPhrases
}
```

Cada frase debe aparecer en párrafo aparte

```
// Nota: Se asume que una frase termina con un punto, interrogación o
exclamación.
const paragraphPerPhrase = (text) => {
    const paragraphDistance = text.match(/\n+/g)[0]
    const newText = text
        .replace(/\. +|\? +|\! +/g, (match) => match[0] + paragraphDistance)

    return newText
}
```

Solo las primeras *n* frases de cada párrafo

```javascript
const maxFrases = (text, n) => {
    let newText = ''
    let frasesCount = 0
    Array.from(text).forEach(char => {
        if (esParrafo(newText, char)) {
            frasesCount = 0;
        }
        if (frasesCount >= n) {
            return
        }
        if (".?!".includes(char)) {
            frasesCount += 1;
        }
        newText += char;
    })
    return newText
}
```

```javascript
const maxPhrases = (text, n) => {
    const paragraphDistance = text.match(/\n+/g)[0]
    const newText = text
        .split(/\n+/g)
        .map(paragraph => {
            const newParagraph = separateParagraph(paragraph).slice(0, n)
            return newParagraph.join(' ')
        })
        .join(paragraphDistance)

    return newText
}
```

```
const separateParagraph = (paragraph) => {
    paragraph = paragraph.split(/(\. +|\? +|\! +)/)
    const newParagraph = []
    paragraph.forEach((phrase) => {
        if (phrase.match(/\. +|\? +|\! +/)) {
            newParagraph[newParagraph.length - 1] = newParagraph[newParagraph.length - 1] + phrase.trim()
        } else {
            newParagraph.push(phrase)
        }
    })
    return newParagraph
}
```

TEXTOS