GRUPO 10

JAVASCRIPT FUNCIONAL

DISEÑO AVANZADO DE APLICACIONES WEB

RAFAELA KARA - CRISTOBAL MUÑOZ - FLAVIO TARSETTI

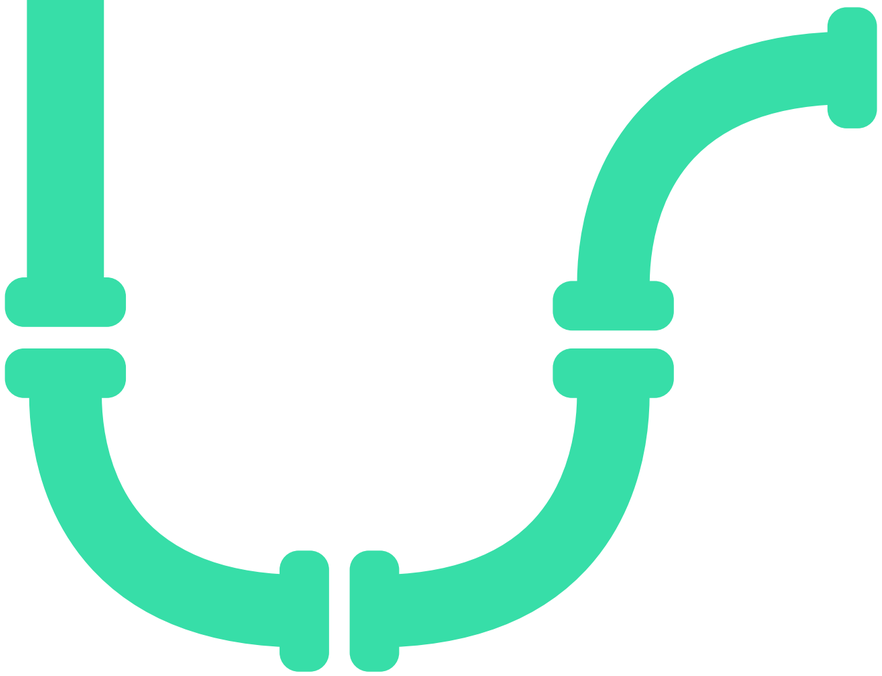
ELEMENTOS IMPLEMENTADOS

- Headings (h1-h6)
- Lists (unordered-ordered)
- Paragraphs
- Italic and bold text
- Links
- Images
- Code

SUPUESTOS

- No hay listas dentro de listas.
- Las listas ordenadas comienzan con "1."
- Para texto en cursiva, negrita o ambas se asume que está bien definido los * al inicio y final.
- Debe haber un espacio luego de definir headings (#) y listas enumeradas (1. o *).
- El * se utiliza para crear listas o poner énfasis en el texto, no como un símbolo de texto.
- Los enlaces para links o imagenes no pueden contener _.

FLUJO INICIAL



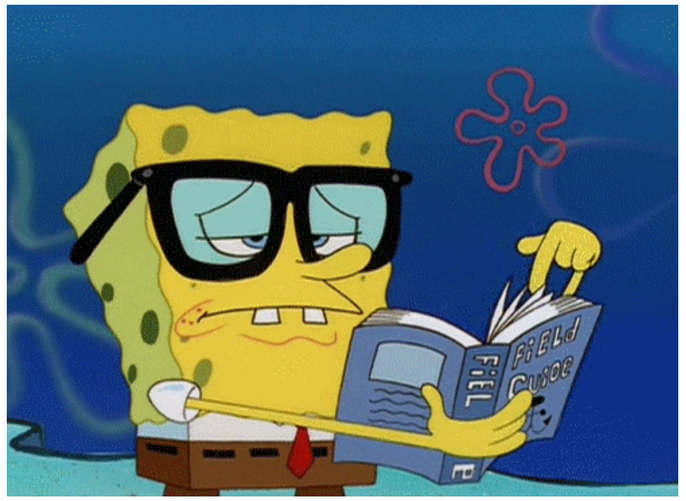
1. **Preparar input**
2. **Recorrer inicio de cada línea y ver si comienza con #, * o no. Si tiene # contar cantidad**
3. **Recorrer cada línea y buscar elementos dentro.**
4. **Crear html**

SOLUCIÓN

```
let filename = process.argv[2]

fs.readFile(filename, 'utf-8', (err, data) {
    const html = _.chain(data)
       ...
       ...
       ...
       .value()

    fw.writeFile(filename.replace('.md', '.html'), html)
};
```



LECTURA DE ARCHIVO MARKDOWN

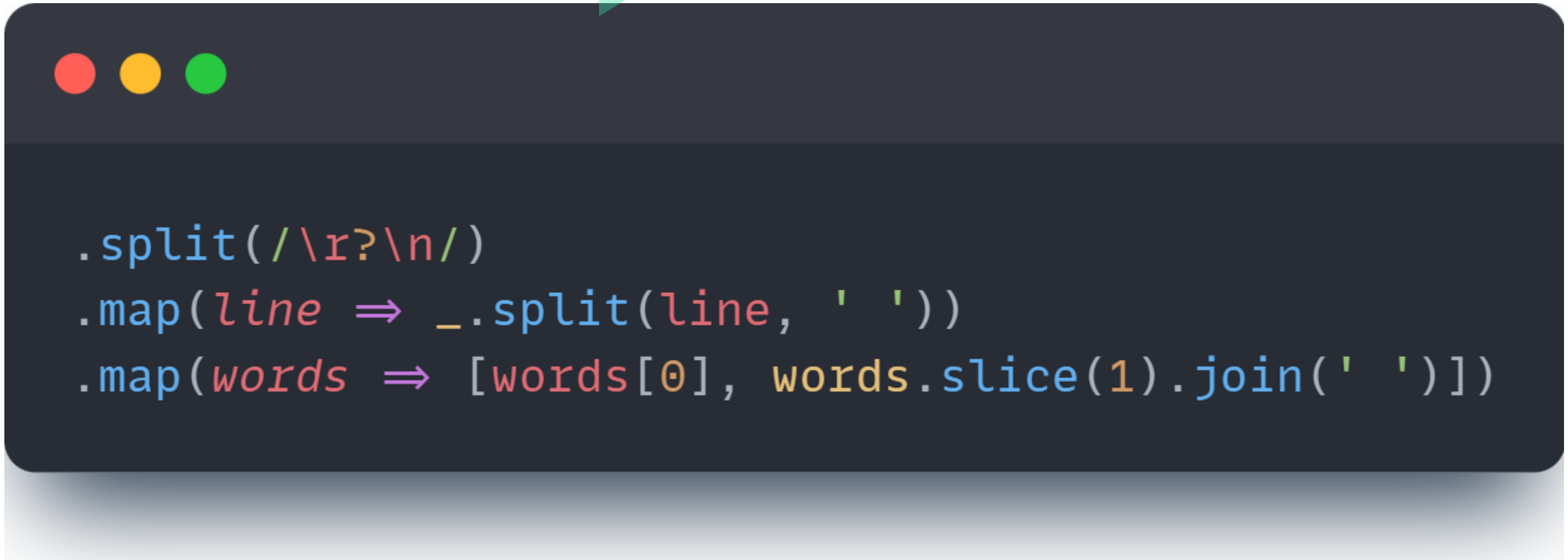
CHAIN DE FUNCIONES

ESCRITURA DE ARCHIVO HTML

- **Separación en lista de líneas**
- **Separación de línea en palabras**
- **Obtención de primera palabra**

EJEMPLO

```
> '# Esto es un título\nhola *como* estas'
> ['#', 'Esto', 'es', 'un', 'título'],['hola, *como*, estas]
> ['#', 'Esto es un título'],['hola', '*como* estas']
```



```
const tagMap = new Map([
    [/^#$/, 'h1'],
    [/^##$/, 'h2'],
    [/^###$/, 'h3'],
    [/^####$/, 'h4'],
    [/^#####$/, 'h5'],
    [/^######$/, 'h6'],
    [/^[*+-]$/, 'li'],
    [/^\d+\./, 'numero'],
]);
```

- **getRegex:** Toma primer elemento de la línea e intenta hacer **match** con las expresiones regulares de **tagMap**
- **addTag:** Si hace match, transforma línea de markdown a html, agregando el **tag** de **tagMap**

EJEMPLO

```
> ['#', 'Esto es un título'],['hola', '*como* estas']
> '<h1>Esto es un título</h1>', '<p>hola *como* estas</p>'
```

```
.map(line ⇒ getRegex(line[0]) ? addTag(line) :
      line[0] ≠ '' ? `<p>${line.slice().join(' ')}</p>`: '')
```

```
function replace(string, pattern, line) {
    return _.replace(line, pattern, string)
}
const curriedReplace = _.curry(replace)
const replaceBoldCursive = curriedReplace(`<em><strong>$1</strong></em>`)
const replaceBold = curriedReplace(`<strong>$1</strong>`)
const replaceCursive = curriedReplace(`<em>$1</em>`)
```

- Currificamos función para reemplazar patrones *inline* con su tag html
- Función currificada para cada tipo de énfasis

```
const pipelineBold = pipe([
    replaceBold(/\*\*(.*?)\*\*/g),
    replaceBold(/__(.*?)__/g),
])
```

- Pipeline incluyendo cada variación de caracteres

EJEMPLO

```
> '<p>hola **como** _estas_</p>'
> '<p>hola <strong>como</strong> <em>estas</em></p>
```

```
.map(line ⇒ pipelineBold(line))
```

- **Transformación de código, imágenes e hipervínculos mediante expresiones regulares**

EJEMPLO IMAGEN

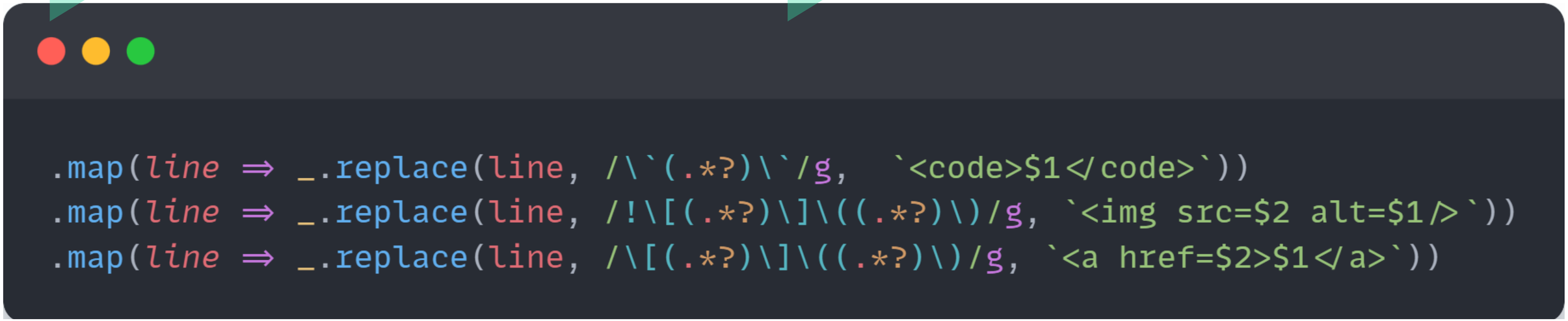
```
> '<p>![imagen](www.imagen.com)</p>'
> '<p><img src=www.imagen.com alt=imagen /></p>'
```

EJEMPLO CÓDIGO

```
> '<p>Abrir `archivo`</p>'
> '<p>Abrir <code>archivo</code></p>
```

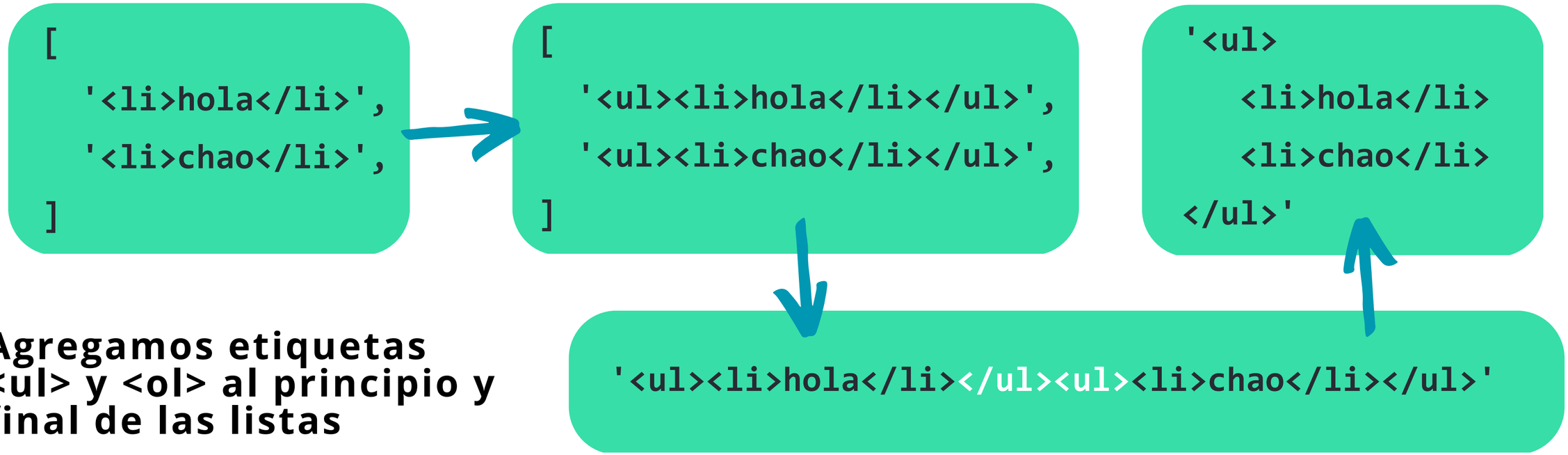
EJEMPLO HIPERVÍNCULO

```
> '<p>[link](www.link.com)</p>'
> '<p><a href=www.link.com>link</a></p>'
```



```
.map(line ⇒ _.replace(line, /\`(.*?)\`/g,  `<code>$1</code>`))
.map(line ⇒ _.replace(line, /!\[(.*?)\]\((.*?)\)/g, `<img src=$2 alt=$1/>`))
.map(line ⇒ _.replace(line, /\[(.*?)\]\((.*?)\)/g, `<a href=$2>$1</a>`))
```

```
.map(line ⇒ _.replace(line, /<li>(.*?)<\/li>/g, `<ul>\n\t<li>$1</li>\n</ul>`))
.map(line ⇒ _.replace(line, /<numero>(.*?)<\/numero>/g, `<ol>\n\t<li>$1</li>\n</ol>`))
.join('\n')
.replace(/<\/ul>\n<ul>\n/g, '')
.replace(/<\/ol>\n<ol>\n/g, '')
.replace(/\n+/g, '\n')
```



- **Agregamos etiquetas <ul> y <ol> al principio y final de las listas**

DIFICULTADES

```
const fixedListLines = _.chain(regularLines)
    .map(line => line.startsWith('<li>') ? line : '')
    .map((line, idx, array) => line.startsWith('<li>') && (array[idx - 1] === '' || array[idx - 1] === undefined) ? `<ul>\n\t${line}` : line)
    .map((line, idx, array) => line.startsWith('<li>') && (array[idx + 1] === '' || array[idx + 1] === undefined) ? `${line}\n</ul>` : line)
    .map(line => line.startsWith('<li>') ? `\t${line}` : line)
    .map((line, idx) => line === '' ? splitLines[idx] : line)
    .value()
const fixedNumberLines = _.chain(fixedListLines)
    .map(line => _.replace(line, /<numero>(.*?)<\/numero>/,  `<li>$1</li>`))
    .map(line => line.startsWith('<li>') ? line : '')
    .map((line, idx, array) => line.startsWith('<li>') && (array[idx - 1] === '' || array[idx - 1] === undefined) ? `<ol>\n\t${line}` : line)
    .map((line, idx, array) => line.startsWith('<li>') && (array[idx + 1] === '' || array[idx + 1] === undefined) ? `${line}\n</ol>` : line)
    .map(line => line.startsWith('<li>') ? `\t${line}` : line)
    .map((line, idx) => line === '' ? listLines[idx] : line)
    .join('\n')
    .replace(/\n+/g, '\n')
    .value()
```

Currying

Expresiones regulares

```
.map(line => _.replace(line, /<li>(.*?)<\/li>/g, `<ul>\n\t<li>$1</li>\n</ul>`))
.map(line => _.replace(line, /<numero>(.*?)<\/numero>/g, `<ol>\n\t<li>$1</li>\n</ol>`))
.join('\n')
.replace(/<\/ul>\n<ul>\n/g, '')
.replace(/<\/ol>\n<ol>\n/g, '')
.replace(/\n+/g, '\n')
```

DEMO