

---

# IIC2026

## Visualización de Información

— Hernán F. Valdivieso López —  
(2023 - 1 / Clase 03)

---

# Temas de la clase - SVG avanzado y JS

1. CSS - Prioridad
2. SVG Avanzado
  - a. Otro elemento: Elipses y Line
  - b. Otros atributos de estilo
3. Introducción a Javascript.
  - a. Variables, strings, listas y objetos.
  - b. Control de flujo y loops.
  - c. Funciones y *function arrows*.
  - d. HTML y JS

# CSS - Prioridad

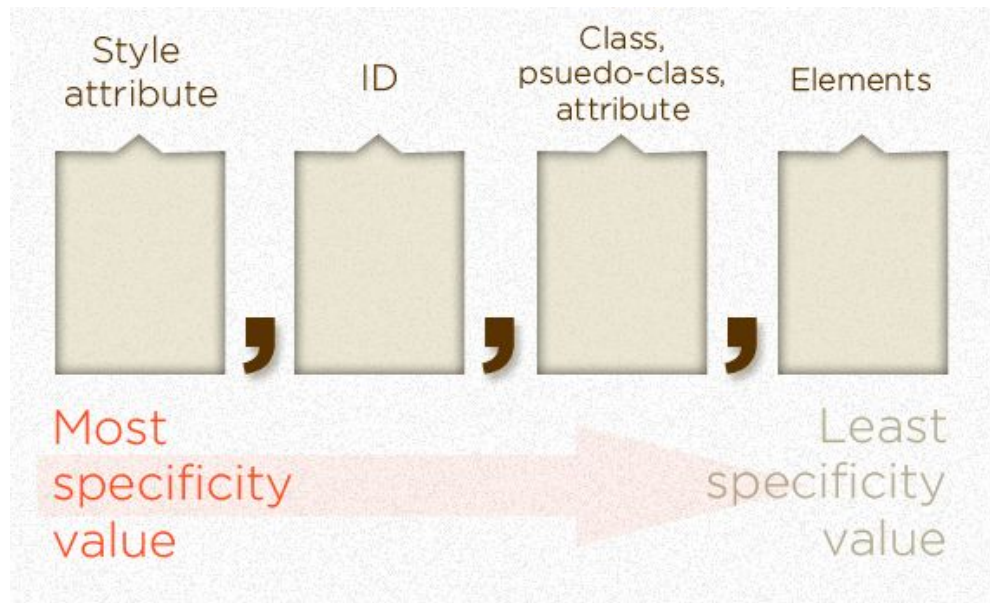
---

# CSS - Prioridad

Llamado también como "especificidad" en css.

Indica la prioridad de entregar estilo a un elemento según el nivel de selector

- Si el elemento tiene un estilo en línea, automáticamente gana. (1, 0, 0, 0)
- El valor de ID aplica (0,1,0,0) puntos.
- Para cada valor de clase aplica (0, 0, 1, 0) puntos.
- Para cada referencia de elemento (por tag) aplica (0, 0, 0, 1) puntos.





**a**

1 x element selector

Sith power: 0,0,1



**p a**

2 x element selectors

Sith power: 0,0,2



**.foo**

1 x class selector \*

Sith power: 0,1,0



**a.foo**

1 x element selector  
1 x class selector

Sith power: 0,1,1



**p a.foo**

2 x element selectors  
1 x class selector

Sith power: 0,1,2



**.foo .bar**

2 x class selectors

Sith power: 0,2,0



**p.foo a.bar**

2 x element selectors  
2 x class selectors

Sith power: 0,2,2



**#foo**

1 x id selector

Sith power: 1,0,0



**p a.foo**

2 x element selectors  
1 x class selector

Sith power: 0,1,2



**.foo .bar**

2 x class selectors

Sith power: 0,2,0



**p.foo a.bar**

2 x element selectors  
2 x class selectors

Sith power: 0,2,2



**#foo**

1 x id selector

Sith power: 1,0,0



**a#foo**

1 x element selector  
1 x id selector

Sith power: 1,0,1



**.foo a#bar**

1 x element selector  
1 x class selector  
1 x id selector

Sith power: 1,1,1



**.foo .foo #foo**

2 x class selectors  
1 x id selector

Sith power: 1,2,0



**style**

1 x style attribute

Sith power: 1,0,0,0



**p a.foo**

2 x element selectors  
1 x class selector

Sith power: 0,1,2



**a#foo**

1 x element selector  
1 x id selector

Sith power: 1,0,1



**No lo hagan**

**!important**



**#foo**

1 x id selector

Sith power: 1,0,0



**style**

1 x style attribute

Sith power: 1,0,0,0

# CSS - Prioridad

Lecturas recomendadas:

- [Jerarquía CSS - Cómo evitar el uso de !important en CSS - Cómo sobrescribir a un important CSS](#)
- [Especificidad - CSS | MDN](#)
- (Imagen mostrada antes) <https://stuffandnonsense.co.uk/archives/images/css-specificity-wars.png>

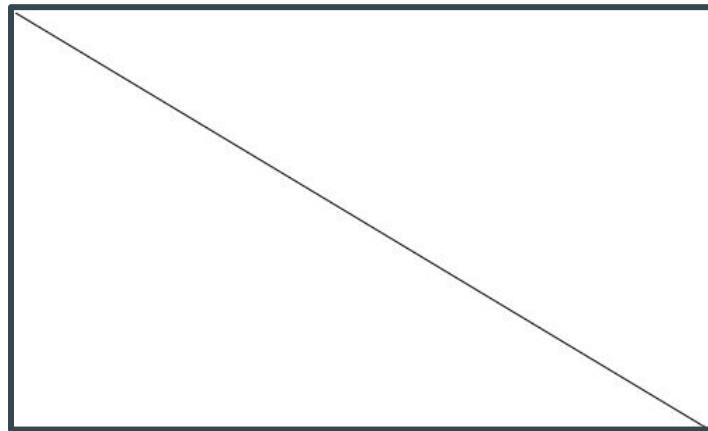


# SVG Avanzado

# SVG Avanzado

```
<svg width="500" height="300">  
  <line x1="0" y1="0" x2="500" y2="300" stroke="black"/>  
</svg>
```

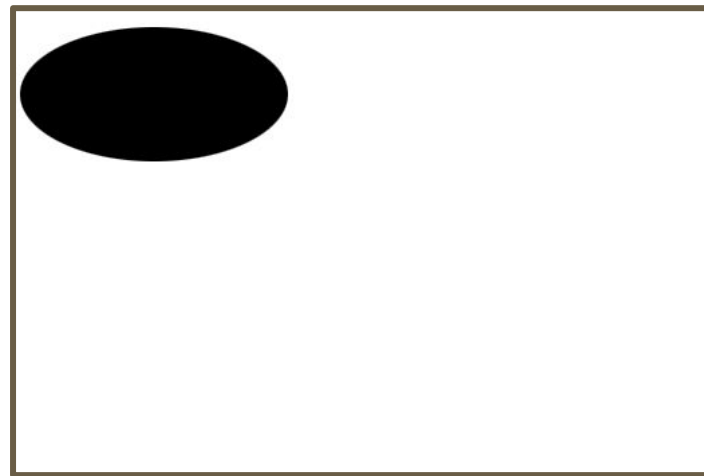
- Tag `<line/>` para crear una **línea**.
- `x1="0"` → Indica el inicio de la recta en eje X
- `y1="0"` → Indica el inicio de la recta en eje Y
- `x2="500"` → Indica el fin de la recta en eje X
- `y2="300"` → Indica el fin de la recta en eje Y
- `stroke="black"` → Color de la recta. **Sin darle un color a la línea, esta no se verá.**



# SVG Avanzado

```
<svg width="500" height="300">  
  <ellipse cx="100" cy="50" rx="100" ry="50" />  
</svg>
```

- Tag `<ellipse/>` para crear una **Elipse**.
- `cx="100"` → Indica centro en eje X
- `cy="50"` → Indica centro en eje Y
- `rx="50"` → Indica radio del círculo en eje X
- `ry="100"` → Indica radio del círculo en eje Y



# SVG Avanzado - opacity

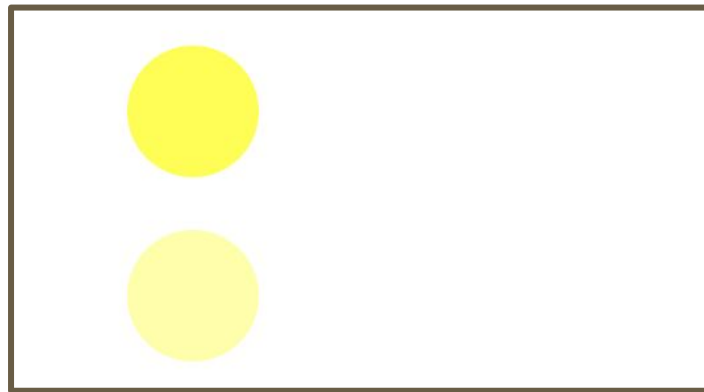
```
<svg width="500" height="300">  
  <circle cx="150" cy="100" r="50" fill="yellow"/>  
  <circle cx="150" cy="100" r="50" fill="yellow" opacity="0.5"/>  
</svg>
```

Como su nombre indica, el atributo **opacity** permite cambiar la opacidad (transparencia) de las figuras.

- El valor va entre 0 y 1.
- 0 = nada opaco, es decir, 100% transparente.
- 1 = totalmente opaco, es decir, 0% transparente.

Muy útil más adelante cuando queremos destacar un elemento.

Les bajamos la opacidad a los demás elementos sin intervenir en su color.

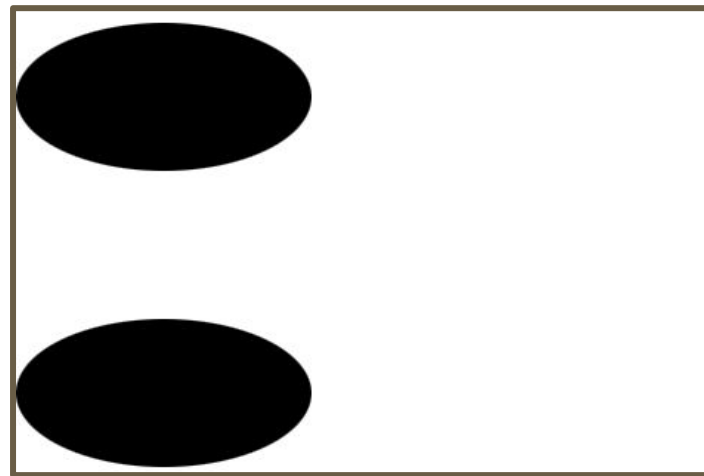


# SVG Avanzado - transform

```
<svg width="500" height="300">  
  <ellipse cx="100" cy="50" rx="100" ry="50" />  
  <ellipse cx="100" cy="50" rx="100" ry="50" transform="translate(0, 200)" />  
</svg>
```

Como su nombre indica, el atributo **transform** permite aplicar transformaciones a las figuras.

- **translate**: permite mover trasladar un elemento.
  - El primer elemento es la traslación en X
  - El segundo elemento es la traslación en Y

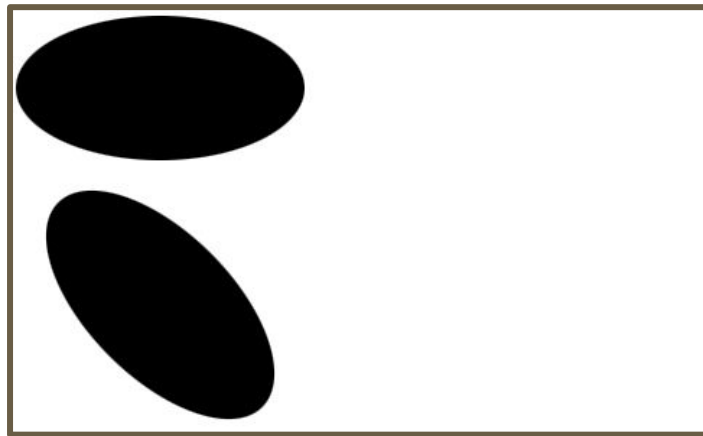


# SVG Avanzado - transform

```
<svg width="500" height="300">  
  <ellipse cx="100" cy="50" rx="100" ry="50" />  
  <ellipse cx="100" cy="50" rx="100" ry="50" transform="translate(0, 150) rotate(45 100 50)" />  
</svg>
```

Como su nombre indica, el atributo **transform** permite aplicar transformaciones a las figuras.

- **translate**: permite mover trasladar un elemento.
  - El primer elemento es la traslación en X
  - El segundo elemento es la traslación en Y
- **rotate**: permite rotar un elemento
  - El primer elemento son los grados de rotación
  - El segundo elemento es el punto de origen en X de la rotación.
  - El tercer elemento es el punto de origen en Y de la rotación



**Spoiler:** esto será muy útil cuando veamos los `<g>` 🐵

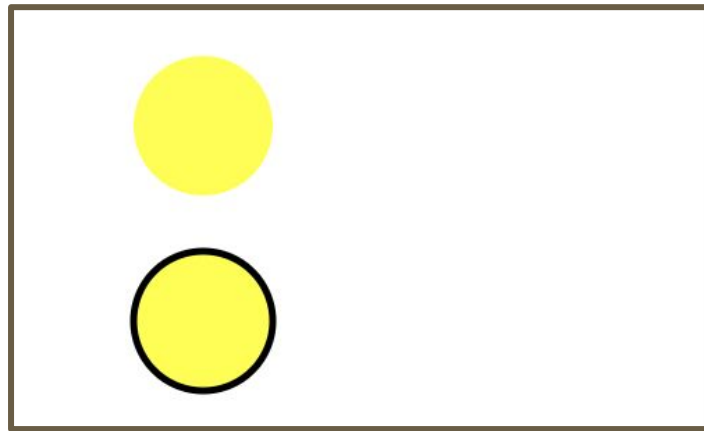
# SVG Avanzado - stroke y stroke-width

```
<svg width="500" height="300">  
  <circle cx="150" cy="100" r="50" fill="yellow"/>  
  <circle cx="150" cy="100" r="50" fill="yellow" stroke="black" stroke-width="5px"/>  
</svg>
```

Como sus nombres indican, estos atributos sirven para definir el borde y ancho del borde de las figuras.

- **stroke**: define el color del borde.
- **stroke-width**: define el ancho del borde.

⚠ Si stroke-width es 10px en un círculo. El círculo será 5px más grande y el relleno será comido en 5px. Es decir, **el borde crece equitativamente para afuera y para dentro**.



# SVG Avanzado - superposición de elementos

```
<svg width="500" height="300">  
  <circle cx="150" cy="100" r="50" fill="yellow"/>  
  <circle cx="150" cy="100" r="50" fill="magenta"/>  
</svg>
```

¿Qué círculo veremos?

- Opción A: círculo amarillo
- Opción B: círculo magenta



# SVG Avanzado - superposición de elementos

```
<svg width="500" height="300">  
  <circle cx="150" cy="100" r="50" fill="yellow"/>  
  <circle cx="150" cy="100" r="50" fill="magenta"/>  
</svg>
```

¿Qué círculo veremos?

- Opción A: círculo amarillo
- **Opción B: círculo magenta**

Un elemento más "abajo" en el SVG estará por encima de los primeros elementos

# SVG Avanzado - superposición de elementos

¿Cómo se formó esta imagen?

1. Círculo grande cuyo *fill* "rgb(200, 300, 10)"
2. 2 Elipses de color naranja cuyo rx es 100 y ry es 50.
  - a. Una se rotó 45 grados desde su centro.
  - b. La otra se rotó 90 grados desde su centro.
3. Un círculo más chico de cuyo *fill* es "transparent" y con un borde negro de 5 pixeles. Se aplicó una opacidad de 0.5
4. 2 líneas para hacer la cruz del mismo color del círculo grande.



**Desafío:** Intente programar ese código.

Está subida la solución pero intente hacerlo por su cuenta primero 😊.

# Links recomendados

- [SVG Tutorial](#) (w3schools)
- [Tutorial de SVG](#) (mozilla)
- [Transforms on SVG Elements | CSS-Tricks](#)

# Introducción a Javascript

---

# Introducción a Javascript

- Lenguaje de programación de la web.
- Es un lenguaje de alto nivel, interpretado y multiparadigma.
- Puede ejecutarse en un navegador web y afectar el contenido de una página.
- Lo utilizaremos como lenguaje para construir visualizaciones a partir de datos.

# Introducción a Javascript II

- Lenguaje en constante evolución.
- Su estándar se llama ECMAScript (ES).
- Consideraremos desde ES2015 o ES6 en adelante para este curso.

# Introducción a Javascript - Variables

## Variable (let)

```
let miVariable = 1;  
miVariable = 4;
```

## Variable (var) → Ya no se ocupa

```
var miVariable = 1;  
miVariable = 4; // Funciona, pero ya no se ocupa
```

## Constante (const)

```
const miVariableConstante = 1;  
miVariableConstante = 4; // ERROR
```

# Introducción a Javascript - String y Arrays

## String

```
const nombre = "Hernán";  
const saludo = "¡Hola " + nombre + "!";  
const saludo2 = `¡Hola ${nombre}!`;
```

## Arrays

```
let arreglo = ["awa", "ewe", "iwi"];  
const variable = arreglo[0]; // "awa"  
arreglo.push("uwu");  
const largo = arreglo.length; // 4
```



# Introducción a Javascript - Objetos

## Objetos

```
let miObjeto = {  
  title: "Spy x Family",  
  year: 2019,  
};
```

Obtener dato del objeto

```
const titulo = miObjeto["title"]; // "Spy x Family"  
const year = miObjeto.year; // 2019
```

Editar dato del objeto

```
miObjeto["genre"] = ["Comedy", "Spy"];  
miObjeto.author = "Tatsuya Endō";
```

# Introducción a Javascript - Formato

**Uso de llaves. No importa la indentación.**

```
if (miVariable > 5) {  
  miVariable = miVariable / 2;  
}
```

```
for (let i = 0; i < 10; i++) {  
  miVariable += i;  
}
```

```
function miFuncion(arg1, arg2) {  
  return arg1 + 2 * args2;  
}
```

# Introducción a Javascript - Control de flujo

```
if (miVariable > 5 && miVariableConstante == 1) {  
    // se ejecuta si ambas cumplen  
    miVariable = miVariable / 2;  
} else if (miVariable > 5 || miVariableConstante == 1) {  
    // se ejecuta si alguna cumple  
    miVariable -= 2;  
} else {  
    miVariable = 0;  
}
```

# Introducción a Javascript - Loops (while)

```
let valor = 0;  
let contador = 0;  
while (contador < 10) {  
  valor += 2;  
  contador += 1;  
}
```

```
let valor = 0;  
for (let contador = 0; contador < 10; contador += 1) {  
  valor += 2;  
}
```

# Introducción a Javascript - Loops (for)

```
let valor = 0;  
for (let contador = 0; contador < 10; contador += 1) {  
  valor += 2;  
}
```

```
const arreglo = [1, 2, 3];  
let suma = 0;  
for (const element of arreglo) {  
  suma += 0;  
}
```

```
const objeto = { a: 1, b: 2, c: 3 };  
let suma = 0;  
for (const propiedad in objeto) {  
  suma += objeto[propiedad]  
}
```

# Introducción a Javascript - Funciones

## Opción 1

```
function sumar10(numero) {  
  return numero + 10;  
}  
sumar10(6); // 16  
sumar10(sumar10(6)); // 26
```

## Opción 2

```
const sumar10 = function(numero) {  
  return numero + 10;  
}  
sumar10(6); // 16  
sumar10(sumar10(6)); // 26
```

# Introducción a Javascript - *Functions Arrows*

```
const sumar10 = (numero) => numero + 10;  
sumar10(6); // 16
```

```
const algoMasComplicado = (arg1, arg2, arg3) => {  
  // ...  
  return valorResultado;  
};  
algoMasComplicado(1, 2, 3);
```

# Introducción a Javascript - 🧐🧐🧐

```
const sumar10 = (numero) => numero + 10;
```

```
sumar10("a")           // 'a10'
```

```
sumar10("1")           // '110'
```

```
sumar10([])            // '10'
```

```
sumar10([2])           // '210'
```

```
sumar10([2, 2, 7])     // '2,2,710'
```

```
sumar10(+ "2")         // 12
```

```
sumar10(- "1")         // 9
```

```
sumar10(- "a")         // NaN
```

Aunque Javascript no diga nada, tener cuidado con no mezclar diferentes tipos de datos



# Introducción a Javascript - HTML y JS

```
// programa_1.js  
let arreglo = [];  
for (let numero = 0; numero <= 20; numero += 2) {  
    arreglo.push(numero);  
}  
console.log(arreglo);  
for (numero of arreglo) {  
    console.log(numero);  
}
```

# Introducción a Javascript - HTML y JS

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Ejemplo con JS 1</title>
  </head>
  <body>
    <script src='programa_1.js' charset='utf-8'></script>
  </body>
</html>
```

# Document Object Model (DOM)

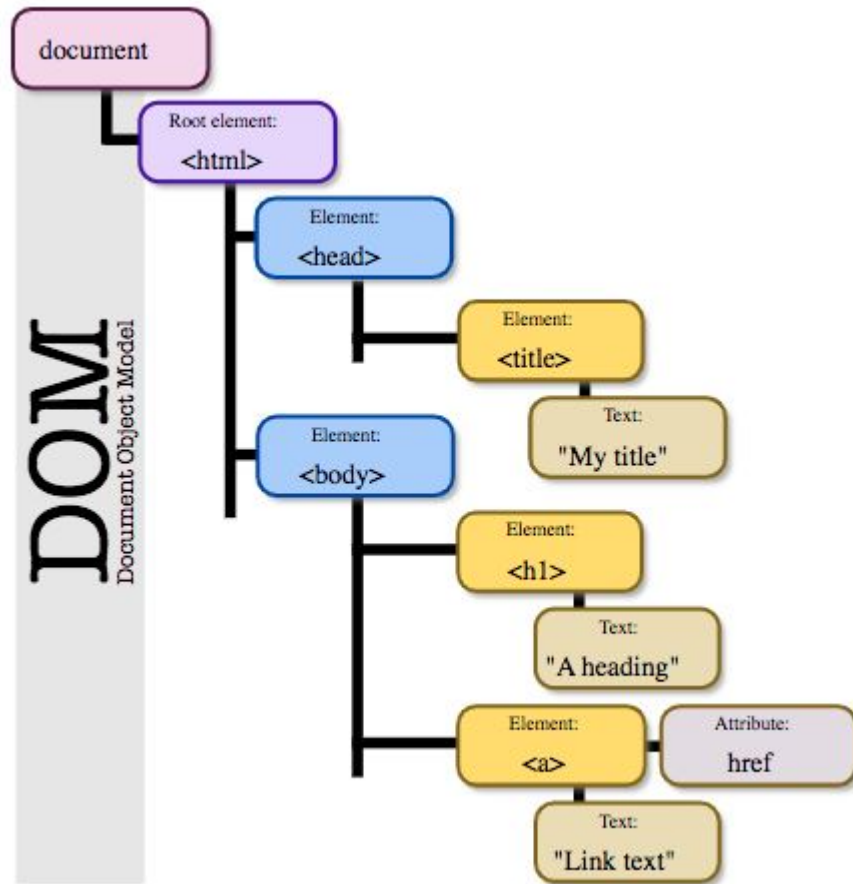
---

# Document Object Model (DOM)

- Forma de representar el contenido de un documento HTML como objeto de programación.

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Title</title>
  </head>

  <body>
    <h1>A heading</h1>
    <a href="...">Link text</a>
  </body>
</html>
```



# Document Object Model (DOM)

Con la variable `document` podremos acceder a todo tipo información del HTML

```
// Imprimimos el link de donde está el documento  
console.log(document.URL);
```

```
// Obtenemos la lista de hijos del documento  
document.children; // [<html>]
```

```
// Obtenemos elemento HTML con id="uwu"  
const elemento = document.getElementById("uwu");
```

```
// Creamos un elemento cuyo tag es <p>  
const parrafo = document.createElement("p");
```

# Document Object Model (DOM)

```
// Creamos un elemento cuyo tag es <p>
const parrafo = document.createElement("p");

// Creamos un elemento tipo texto
const texto = document.createTextNode("¡Soy un texto >.<!");

// Agregamos el texto al párrafo
parrafo.appendChild(texto);
```

# Document Object Model (DOM) - Ejemplo

```
// programa_2.js
const parrafo = document.createElement("p");
const texto = document.createTextNode("¡Soy un texto >.<!");
parrafo.appendChild(texto);

const elementoRaiz = document.getElementById("raiz");
elementoRaiz.appendChild(parrafo);
```

# Document Object Model (DOM) - Ejemplo

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Ejemplo con JS 2</title>
  </head>

  <body id="raiz">
    <script src='programa_2.js' charset='utf-8'></script>
  </body>
</html>
```



# Document Object Model (DOM) - Clases y ID

```
// Creamos un elemento con tag <p>  
const elemento = document.createElement("p");
```

```
// Definimos la clase del elemento  
elemento.className = "importante";
```

```
// Agregamos una clase al elemento  
elemento.classList.add("otra clase");
```

```
// Removemos una clase al elemento  
elemento.classList.remove("otra clase");
```

```
// Definimos una único id a elemento  
elemento.id = "principal";
```

# Document Object Model (DOM) - Eventos

```
// Creamos un elemento con tag <h1>
```

```
const elemento = document.createElement("h1");
```

```
// Conectamos la ocurrencia de un evento en el elemento a una función
```

```
elemento.addEventListener(evento, funcion);
```

🤔 ¿Qué evento puede ser?

- Click sobre el elemento ([click](#)).
- Pasar el mouse sobre el elemento ([mouseover](#)).
- Copiar el elemento ([copy](#)).
- Hacer scroll en el elemento ([scroll](#)).
- Muchos más: [https://www.w3schools.com/jsref/dom\\_obj\\_event.asp](https://www.w3schools.com/jsref/dom_obj_event.asp)

# Document Object Model (DOM) - Ejemplo de evento

```
// programa_3.js

const raiz = document.getElementById("raiz");
const principal = document.getElementById("principal");

let contador = 0;

principal.addEventListener("click", () => {
  contador += 1;
  const parrafo = document.createElement("p");
  const texto = document.createTextNode(`Cantidad de clics o.o: ${contador}`);
  parrafo.appendChild(texto);

  raiz.appendChild(parrafo);
});
```

# Document Object Model (DOM) - Ejemplo de evento

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Ejemplo con JS 3</title>
  </head>

  <body id="raiz">
    <h1 id="principal">¡Presioname!</h1>
    <script src='programa_3.js' charset='utf-8'></script>
  </body>
</html>
```

# Próximos eventos

## Próxima clase


- Librería D3.js
- Clase **muy importante para el curso**
- Traer *notebook* 

## Ayudantía de mañana

- HTML y SVG. Dibujar con SVG

## Tarea 1

Se sube **este jueves**.

*spoiler:* Van a replicar una imagen compuesta de varias figuras vistas en clases (círculos, elipses, texto, rectángulos). Podrían intentar hacer la pintura de Piet Mondrian  

---

# IIC2026

# Visualización de Información

— Hernán F. Valdivieso López —  
(2023 - 1 / Clase 03)

---