



# COMPUEDUCACIÓN

---

## REACT JS



# BIENVENIDA





# DATOS



**Ing. Mario Mendoza Sedeño**  
[mario.sedeno@gmail.com](mailto:mario.sedeno@gmail.com)



# AGENDA

---

- ✓ INTERNET
- ✓ HTML
- ✓ PROTOTIPADO
- ✓ CSS
- ✓ JAVASCRIPT
- ✓ NODE JS
- ✓ REACT JS
- ✓ REDUX JS





# INTERNET





# INTERNET



## HISTORIA

Comenzó hace tres décadas, cuando la comunidad científica buscaba con *esfuerzo* una forma rápida y efectiva de compartir información, conocimientos y éxito.

**El surgimiento de los ordenadores** propició el origen de la plataforma abierta donde intercambiaban documentos estructurados de forma **fiable** y **universal**.

En este contexto, a finales de los años 60 nació la **ARPAnet** (Advanced Research Projects Agency) que puso a disposición de los científicos una red análoga llamada **NSFnet**, creada por la **NSF** (National Science Foundation).



# INTERNET



Esta red permitió la comunicación entre muchas universidades y desarrolló un nuevo sistema de comunicación para desarrollar protocolos llamado "**comutación de paquetes**".

En la década de los 70 apareció el *Protocolo de Control de Transmisión/Protocolo de Internet (TCP/IP)*, en el que basaban los servicios de Internet y los mensajes de correo electrónico.

Los estándares desarrollados en ese periodo pasaron en los años 80 a la **Agencia de Comunicación de Defensa del Departamento de Defensa de los Estados Unidos**, que se convirtió en su guardián hasta que se pasaron al *Internet Architecture Board*.



# INTERNET



Comenzó a vislumbrarse la posibilidad de **conectar todas las redes existentes en el mundo**, pero para conseguirlo era necesario crear una forma estándar de almacenar los datos que pudiera verse desde cualquier plataforma informática.

Así nació **HTML** (HyperText Markup Language o Lenguaje de Etiquetas de Hipertexto), que se convertiría en el estándar de diseño Web en los años posteriores.'

También se desarrollaron otras especificaciones como **URL** e HiperTexto Transfer Protocolo (**HTTP**) o Protocolo de transferencia de Hipertexto publicadas en el primer servidor y que lograron una amplia difusión.



# INTERNET



Andreessen y otros investigadores fundaron la **Netscape Communication Corporation** que produjo la primera versión de este navegador, y Microsoft, para no quedar a la zaga, lanzó **Microsoft Internet Explorer**, dando inicio a la llamada *batalla de los navegadores* por el dominio del mercado, fenómeno que fomentó la aparición arbitraria de *formas no estándar del HTML*.

Para definir una dirección futura, Tim Berners-Lee creó el **World Wide Web Consortium (W3C)** en **1994**, **www.w3.org**, que desde entonces interviene como un foro neutral donde empresas y organizaciones pueden discutir y ponerse de acuerdo sobre nuevos protocolos informáticos.

G

# HTML



**HTML**



## Hypertext Markup Language (HTML)

- Las páginas no son elementos aislados, sino que están unidas a otras mediante los links o enlaces hipertexto (imagen, vídeo, script, entre otros.), es decir, no se incrustan directamente en el código de la página, sino que se hace una referencia a la ubicación de dicho elemento mediante texto.
- Está compuesto por TAGS: <html> ... </html>

\* [HTML Tags reference](#)



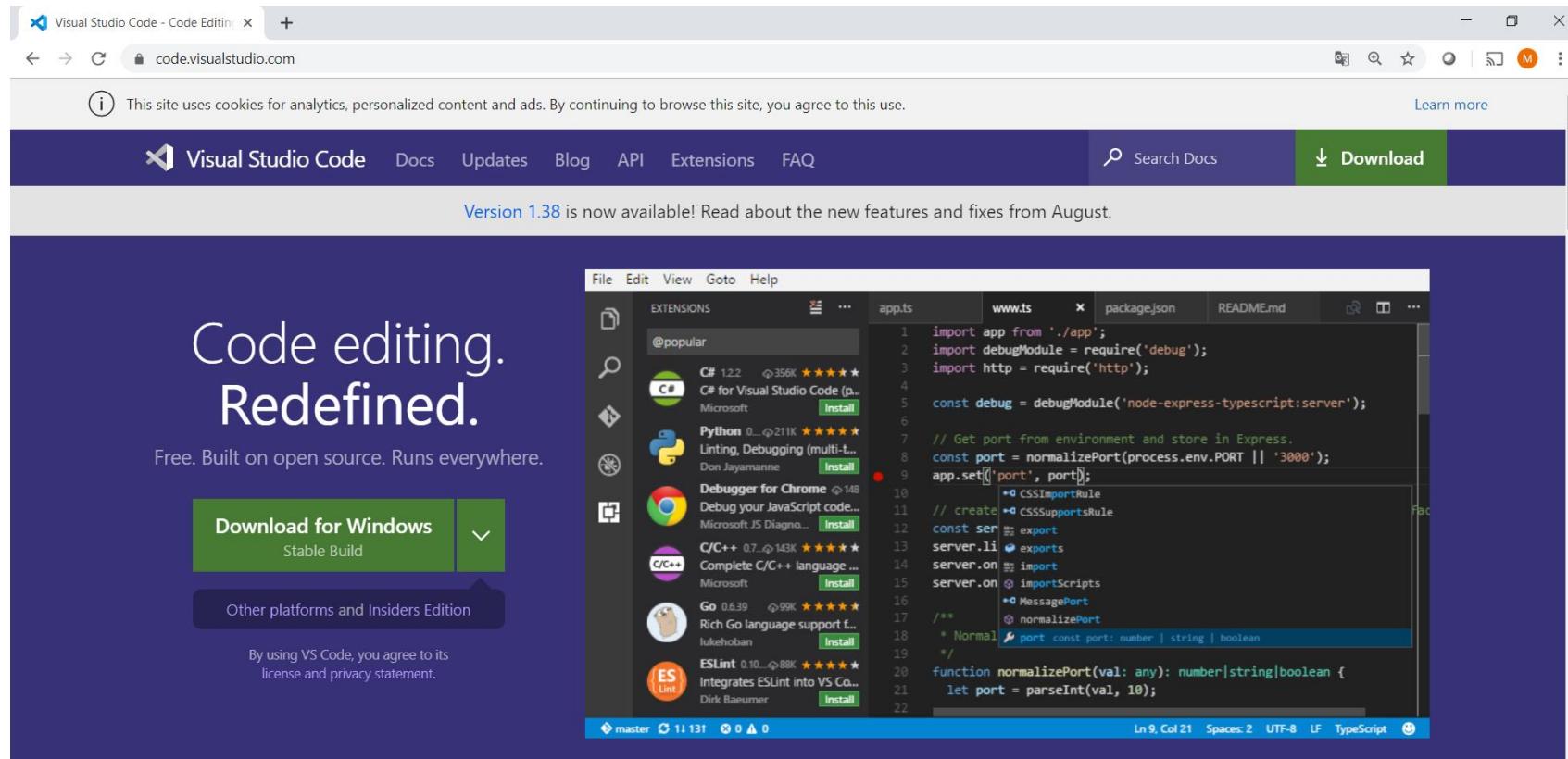
# HTML



## Versiones

Versión	Año
HTML	1991
HTML 2.0	1995
HTML 3.2	1997
HTML 4.01	1999
XHTML	2000
HTML5	2014

# Instalación Editor de Código





# HTML



## Elementos principales:

```
1. <!doctype html>
2. <html>
3.   <head>
4.     <meta charset="uft-8"/>
5.     <link rel="stylesheet" type="text/css" href="theme.css">
6.     <title>Hola Mundo en HTML</title>
7.   </head>
8.   <body>
9.     <h1>Hola Mundo</h1>
10.    <p>Mi primera página en HTML.</p>
11.    <a href="http://www.gruposalinas.com.mx/">Grupo Salinas</a>
12.  </body>
13. </html>
```

## DOCTYPE

<! DOCTYPE > debe ser lo primero en un documento HTML, no es una etiqueta es una instrucción que nos indica en qué versión de HTML está escrita la página.

### HTML 5

```
<!DOCTYPE html>
```

### HTML 4.01

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">
```

# Etiquetas(tags)

`<!--...-->` Comentario

`<html>` Documento raíz

`<head>` Encabezados

`<meta>` Definición de metadatos

`<title>` Título

`<link>` Enlace a recurso externo

`<body>` Cuerpo del documento

`<h1..6>` Encabezado de texto

`<p>` Párrafo

`<a>` Enlace a otro documento

`<img>` Enlace externo de imagen

`<div>` Secciones en documento

`<br>` Genera un salto de línea

`<hr>` Línea horizontal

`<small>` Texto más pequeño

`<strong>` Énfasis

`<ul><ol><li>` Listas

`<table><tr><td>` Tablas

`<u><i><b>` Subrayado, itálica, bold

## Referencia

<https://developer.mozilla.org/es/docs/Web/HTML/Elemento>

# HTML Atributos

- class
- colspan, rowspan
- cols
- data-\*
- disabled
- enctype
- method
- width, height
- id
- maxlength
- readonly
- selected
- src
- style
- title
- <div class="model">
- <td colspan="2">, <tr rowspan="2">
- <textarea cols="2">
- <section data-title="Sección 1">
- <input type="text" disabled >
- <form enctype="multipart/form-data">
- <form method="POST">
- 
- <input id="nombre" name="nombre">
- <input type="text" maxlength="20">
- <input type="number" readonly >
- <option selected>
- 
- <div style="background-color:red">
- <a title="Contacto">

# Ejemplo

```
<html>
  <head>
    <meta charset="UTF-8">
    <meta name="description" content="Ejemplo HTML">
    <meta name="keywords"
content="HTML,CSS,XML,JavaScript">
    <meta name="author" content="John Doe">
    <title>Titulo de la página</title>
  </head>
  <body>

    <h1>Lista no ordenada</h1>
    <ul>
      <li>Elemento 1</li>
      <li>Elemento 2</li>
    </ul>
    normal <strong>strong</strong> <small>small</small> <br>
    
    <hr>
    <ol>
      <li><a href="#c1">Contenido 1</a></li>
      <li><a href="#c2">Contenido 2</a></li>
    </ol>
    <br/>
    <h3 id="c1">Contenido 1</h3>
    <p>.... </p>
    <h3 id="c2">Contenido 2</h3>
    <p> .... </p>
  </body>
</html>
```

# Caracteres especiales

&ampnbsp Non-breaking space

&amp; & (ampersand)

&lt; <

&gt; >

&copy; © (copyright symbol)

&reg; ® (Reg. trademark symbol)

&deg; ° (degree)

&quot; " (Quotation mark)

&ndash; – (n-dash)

&aacute; á

&Aacute; Á

&acirc; â

&auml; ä

&ntilde; ñ

## Ejemplo

<p>

El c&iacute;rculo que se  
mostr&oacute; a un lado del ni&ntilde;o  
med&iacute;a 360&deg;

</p>

Práctica Hacer la siguiente página

## **Curriculum Vitae de Bruce Wayne**

### **Datos personales**

- Nombre completo: **Bruce Wayne**
- Fecha de nacimiento: **1/5/1939**
- Lugar de nacimiento: **Gotham City**

### **Formación académica**

- 1956-1961: **Universidad del Espantapájaros**
- 1952-1956: **Instituto de Dos Caras**
- 1944-1952: **Escuela Primaria del Joker**

### **Experiencia laboral**

- 1975-1985: **En el paro**
- 1965-1975: **Cazavillanos y demás chusma**
- 1962-1965: **Aprendiz de superhéroe**

# Tablas HTML

Una tabla básica puede ser declarada usando los elementos:

**table** .- Contenedor principal.

**thead** .- Agrupa el contenido del encabezado de una tabla.

**tbody**.- Agrupa el contenido del cuerpo de una tabla.

**tr**.-Representa las filas contenedoras de las celdas.

**td**.- Representa a las celdas.

**th**.- Representa a las celdas de encabezado.

# Tablas HTML

Extender filas o columnas

- **ROWSPAN**: extiende una celda a varias filas. Formato:

ROWSPAN=número de filas

- **COLSPAN**: extiende una celda a varias columnas. Formato:

COLSPAN=número de columnas

# Ejemplo tabla

```
<!DOCTYPE html>
<html>
<head>
  <title>Clima</title>
</head>
<body>
  <table class="egt" width="100%">
    <thead>
      <tr>
        <th>Hoy</th>
        <th>Mañana</th>
        <th>Sábado</th>
      </tr>
    </thead>
```

```
  <tbody>
    <tr>
      <td>Soleado</td>
      <td>Mayormente soleado</td>
      <td>Parcialmente nublado</td>
    </tr>
    <tr>
      <td>19°C</td>
      <td>17°C</td>
      <td>12°C</td>
    </tr>
    <tr>
      <td>E13 km/h</td>
      <td>E11 km/h</td>
      <td>S16 km/h</td>
    </tr>
  </tbody>
</table>
</body>
</html>
```

## HTML 5

Se adicionaron tags para una mejor estructura del documento.

- <article>, <aside>, <details>
- <header>, <main>, <footer>
- <dialog>
- <nav>
- <section>
- <summary>

[https://www.w3schools.com/html/html5\\_new\\_elements.asp](https://www.w3schools.com/html/html5_new_elements.asp)

# HTML 5

Se adicionaron tags para elementos multimedia

- <audio>
- <embed>
- <source>
- <track>
- <video>

[https://www.w3schools.com/html/html5\\_new\\_elements.asp](https://www.w3schools.com/html/html5_new_elements.asp)

# Ejemplo HTML5

```
<!DOCTYPE html>
<html lang="es">
  <head>
    <meta charset="utf-8" />
    <title>Titulo</title>
  </head>
  <body>
    <div id="centrado">
      <header>
        <hgroup>
          <h1>TÍTULO</h1>
          <h2>DESCRIPCIÓN DE LA WEB</h2>
        </hgroup>
        <div id="logotipo">LOGOTIPO</div>
      </header>
      <div id="contenido">
        <nav>MENU DE NAVEGACION</nav>
        <aside>ANUNCIO DEL PDF</aside>
        <section id="izquierda">
          <h2>Últimas Novedades</h2>
          <article>NOVEDAD UNO</article>
          <article>NOVEDAD DOS</article>
        </section>
        <section id="centro">
          <h2>Temas Populares</h2>
          <article>POPULAR A</article>
          <article>POPULAR B</article>
        </section>
      </div>
      <footer>
        <section id="copyright">Copyright </section>
        <section id="contacto">Contacto</section>
      </footer>
    </div>
  </body>
</html>
```

# Ejemplo HTML5 Video/Audio

```
<video width="400" controls>
    <source src="https://www.w3schools.com/html/mov_bbb.mp4" type="video/mp4">
    <source src="https://www.w3schools.com/html/mov_bbb.ogg" type="video/ogg">
        No tiene soporte para video HTML5
</video>
<audio controls>
    <source src="https://www.w3schools.com/html/horse.ogg" type="audio/ogg">
    <source src="https://www.w3schools.com/html/horse.mp3" type="audio/mpeg">
        No tiene soporte para el elemento audio HTML5
</audio>
```

# HTML 5

Se adicionaron tags para gráficos HTML 5

- <canvas>
- <svg>

[https://www.w3schools.com/html/html5\\_new\\_elements.asp](https://www.w3schools.com/html/html5_new_elements.asp)

# Ejemplo HTML5 Gráficos

```
<canvas id="myCanvas">Your browser does not support the HTML5 canvas tag.</canvas>
```

```
<script>
```

```
var c = document.getElementById("myCanvas");
```

```
var ctx = c.getContext("2d");
```

```
ctx.fillStyle = "#FF0000";
```

```
ctx.fillRect(0, 0, 80, 100);
```

```
</script>
```

```
<svg width="300" height="200">
```

```
  <polygon points="100,10 40,198 190,78 10,78 160,198"
```

```
    style="fill:lime;stroke:purple;stroke-width:5;fill-rule:evenodd;" />
```

```
Sorry, your browser does not support inline SVG.
```

```
</svg>
```

## Formularios

- Los formularios o también llamados forms, se utilizan para la entrada y envío de datos (el procesamiento de los mismos es aparte) desde una página HTML.
- En un formulario podremos solicitar diferentes datos (campos) cada uno de los cuales quedará asociado a una variable. Una vez introducidos los valores en los campos, el contenido de éstos será enviado a la dirección (URL) donde esté el programa que pueda procesar las variables.

## Estructura Formulario

```
<form action="insertar.html" method="POST">
  <input type="text" name="tarea" />
  <textarea name="descripcion"></textarea>
  <select name="tipo">
    <option selected>Nueva</option>
    <option>Actualizar</option>
    <option>Otra</option>
  </select>
  <input type="submit" value="Enviar" />
</form>
```

- **form.** Etiqueta de formulario
- **action.** Acción a realizar o programa que procesara los datos
- **method.** Método HTTP (GET o POST) que se utilizará al enviar los datos.
- **enctype.** Especifica cómo se deben codificar los datos del form al enviarlos (application/x-www-form-urlencoded)
- **input, textarea, select.** Elementos de entrada de variables.
- **type.** Tipo de entrada: texto, número, fecha, opción, etc.

# Tipos de <INPUT>

- text
- password
- checkbox
- radio
- button
- image
- file
- hidden
- submit
- reset
- Texto
- Contraseña
- Caja de selección
- Opción
- Botón
- Imagen como para envío de formulario
- Selección de archivo
- Oculto
- Envío de formulario
- Resetear o limpiar formulario

# Tipos de <INPUT> HTML5

- color
- date
- datetime-local
- email
- month
- number
- range
- search
- tel
- time
- url
- week
- Color picker
- Control fecha
- Control fecha y hora
- Email
- Control mes y año.
- Número
- Rango de valores
- Texto para su búsqueda
- Número de teléfono
- Hora
- Campo para una URL
- Semana del año

## Atributos de <INPUT>

- accept
- align
- alt
- autocomplete
- autofocus
- checked
- dirname
- disabled
- form
- height
- list
- max
- maxlength
- formtarget
- min
- multiple
- name
- pattern
- placeholder
- readonly
- required
- size
- src
- step
- type
- value
- width

[https://www.w3schools.com/tags/tag\\_input.asp](https://www.w3schools.com/tags/tag_input.asp)

## Form Archivos

Para envío de archivos, se debe incorporar el enctype, y método POST, para poder enviar los paquetes completos del archivo.

```
<form action="enviar.html" enctype="multipart/form-data">  
....  
    <input type="file" name="recibo" />  
....  
</form>
```

# Práctica

Realizar el siguiente HTML en tipo con las opciones de Noticia y Aviso

## Registro

Por favor llene el formulario para crear un artículo.

---

Título:

Autor:

Tipo:

Fecha publicación:

Texto:



# PROTOTIPADO





# PROTOTIPADO



## Prototipado

El prototipado se realiza tras haber completado nuestra fase de **investigación** para definir el público objetivo, la creación de la **arquitectura de información** de nuestro sitio o app y la definición de los **contenidos** que vamos a albergar.

Con todos estos datos ya podemos empezar a tener una base sólida para concretar, **de menor a mayor definición**, nuestro producto.

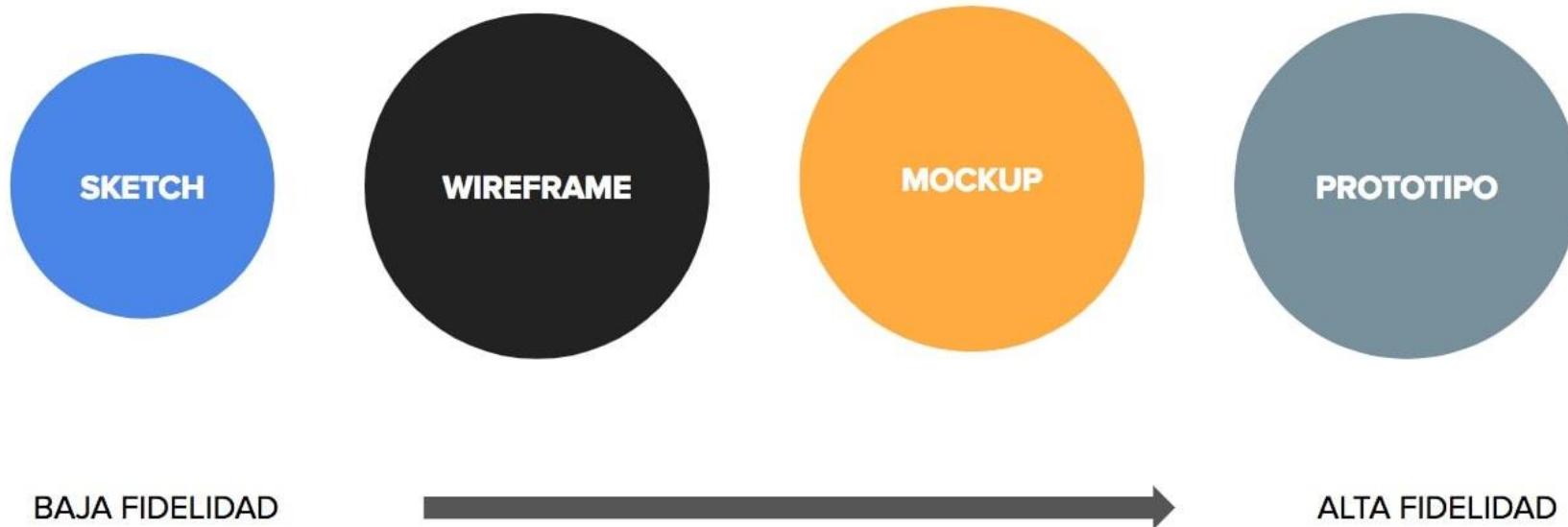


# PROTOTIPADO



## Fases

Generalmente, el prototipado se compone de cuatro fases en las que iremos incrementando la fidelidad del producto: **sketching**, **wireframing**, **mockups** y **prototipo funcional**.





# PROTOTIPADO



## SKETCHING

De una forma sencilla, los sketches nos permiten **dibujar rápidamente pantallas, secciones y relaciones entre éstas**. Es una fase donde no se requiere fidelidad y permite probar ideas y establecer composiciones con rapidez para dejar cimentada una base.

Con varios sketches realizamos el flujo de la aplicación o de la web y generamos el esquema de navegación. Sin costo tenemos todas las pantallas del sitio, con las que podemos jugar y generar feedback por parte del equipo.



# PROTOTIPADO



## SKETCHING Características

- Papel y lápiz: no necesitamos más para realizar sketches.
- Formas básicas: líneas, cuadrados, rectángulos y círculos.
- Un sólo color: no necesitamos más para dejar plasmado en papel un elemento de la web.
- Baratos y desechables: no se requiere ninguna herramienta costosa para realizarlos.

Ejemplo: [https://youtu.be/iVFTBj\\_BYy0](https://youtu.be/iVFTBj_BYy0)



# PROTOTIPADO



## WIREFRAMING

Subimos el nivel de fidelidad y comenzamos con los wireframes. Entran en juego el **layout y los elementos de la web** (cajas de texto, botones, formularios, enlaces, secciones, imágenes, etc), los cuales ya tendrán un nivel de detalle lo suficientemente alto como para poder generar un diseño fiel en la fase siguiente.

Los wireframes son lo primero que el cliente ve, ya no son solo requerimientos sino algo tangible, y además en un tiempo muy corto desde el comienzo del proyecto. Gracias a esto comenzamos a **recibir feedback muy pronto**, por lo que cambiar, deshacer y hacer no supone un gran trabajo.



The image shows a prototype of a social media application interface. At the top, there is a navigation bar with links: Home, Profile, Find People, Settings, Help, and Sign out. Below the navigation bar, the main content area features a large header with a placeholder for a profile picture (a red X inside a square) and the text "What's happening?". To the right of this header is a count of 140. Below the header is a large, empty rectangular input field for posting a tweet. Underneath this input field, a message reads: "Latest: Trying out this w i reframing prototyping tool called FlairBuilder. http://www.flairbuilder.com about 1 hour ago". To the right of this message is a "Tweet" button. The main content area also includes a "Home" section with three recent tweets from a user named "nickf". Each tweet consists of a small placeholder image (red X), the user's name, the tweet text ("Having a great day today. I think this calls for some cupcakes or something."), and the timestamp ("less than 20 seconds ago via web"). To the right of the main content area is a sidebar for the user "robenslin". The sidebar displays the user's profile picture (red X), their name "robenslin", and the number of tweets they have posted (3,983). It also shows their follower count (599), the number of people they are following (853), and the number of accounts listed (43). Below this information is a brief description of the user: "oneforty-essentials n. a collection of must have Twitter apps.". The sidebar also contains links to various user statistics: Home, @robenslin, Direct Messages (73), Favourites, Retweets, Saved Searches, and Lists. Each of these links is preceded by a small placeholder image.



# PROTOTIPADO



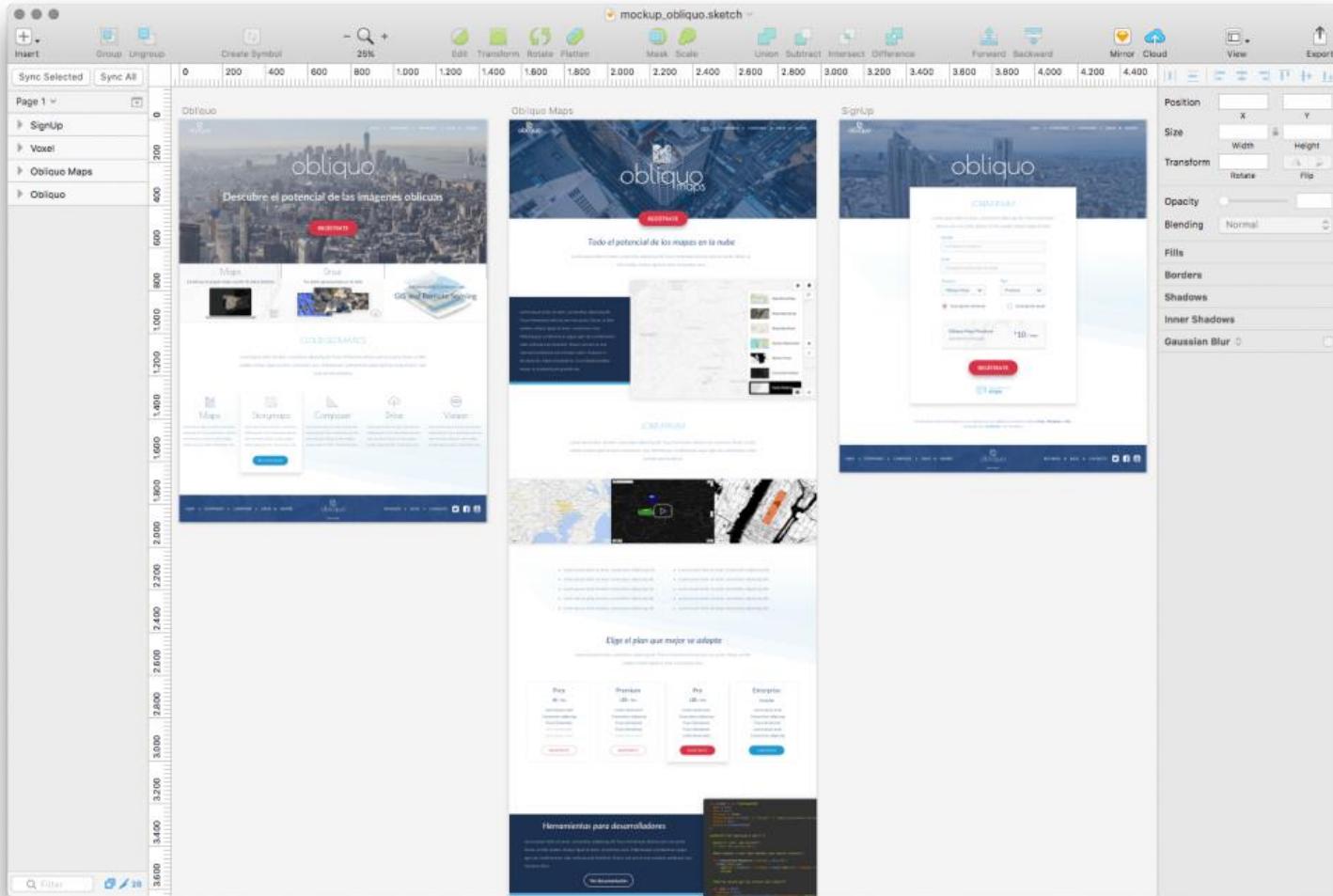
## MOCKUP

Volvemos a subir el nivel de fidelidad y es momento de realizar los mockups. En esta fase toma protagonismo el **look & feel de la web**, introduciendo logotipos, colores, gráficos, tipografías, etc. Es lo que normalmente se considera diseño web, el cual el equipo de desarrollo convertirá en el producto final.

A esta fase debemos llegar con todo bien definido. Tan bien definido como para que no deba cambiar la estructura y composición de elementos de la web ya que este trabajo se ha realizado y refinado en las dos fases anteriores.

# G

# PROTOTIPADO





# PROTOTIPADO



## PROTOTIPO

Tras tener finalizado el mockup toca darle vida y mostrar las interacciones y flujos como si de una web o app real se tratara. Esto nos permite representar de forma fidedigna cómo va a funcionar y se va a comportar nuestra web o aplicación móvil.



# PROTOTIPADO



## HERRAMIENTAS

- Papel y lápiz para sketchs.
- Balsamiq Mockups para realizar los wireframes.
- Sketch para hacer los mockups y Sketch Cloud para tener los mockups online y que tanto el equipo como el cliente puedan dar feedback de manera sencilla.
- Marvel App para realizar el prototipo dinámico.

G

CSS



CSS



# Introducción

## CSS. Cascade Style Sheets

- Es el lenguaje que describe el estilo de un elemento HTML en un documento y cómo deben ser desplegados en la pantalla, impresora o en otro tipo de media.
- Puede controlar la plantilla de múltiples páginas Web a la vez
- Se pueden definir las hojas de estilo en el documento HTML o en un archivo externo.
- Ahorra mucho trabajo.

# Sintaxis

Selector

h1

Declaration

{ color:blue; font-size:12px; }

Declaration

Property      Value

Property      Value

Selector. Puede ser Etiqueta, clase, identificador, o combinación de todos.

Ejemplo:

```
p {  
    color: red;  
    text-align: center;  
}
```

# CSS en línea

```
<!DOCTYPE html>
<html>
<body>

<h1 style="background-color:dodgerblue;">Hello World</h1>

<p style="background-color:Tomato;">
Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat.
</p>

</body>
</html>
```

# Selectores

```
<h1>Título</h1>
```

```
h1 { color: red; }
```

```
<span class="alert">Error al guardar</span>
```

```
.alert {  
    background-color: #ff0000,  
    color: black;  
}
```

```
<label id="nombre">Don Simón</label>
```

```
#nombre {  
    font-family: "Times New Roman", Times,  
    serif;  
}
```

# Agrupación y Cascada

- Agrupación de selectores

```
h1, h2, p {  
    text-align: center;  
    color: red;  
}
```

- Cascada

```
p.alerta span {  
    text-align: center;  
    color: red;  
}
```

# CSS más utilizados

Property	Values	Used with Elements
color	#RRGGBB (Red, Green, Blue hex values)	any element that contains text
text-align	left   right   center   justify	block elements h1..h6, p, li, etc.
text-decoration	none   underline   overline   line-through   blink   inherit	mostly with a (anchor) elements
text-transformation	none   capitalize   uppercase   lowercase	any element that contains text
line-height	% or px	block elements h1..h6, p, li, etc.
letter-spacing	normal or px value	any element that contains text
font-family	font or font-family [ , font or font-family ... ]	any element that contains text
font-size	px or em value	any element that contains text
font-style	normal   italic   oblique	any element that contains text
font-weight	normal   bold	any element that contains text
background-color	#RRGGBB (Red, Green, Blue hex values)	any element with a background
background-image	url("[image url]")	mostly with body
background-repeat	repeat   repeat-x   repeat-y   no-repeat	mostly with body
background-position	left   center   right   top   center   bottom	mostly with body
list-style-type	disc   square   circle	ul
list-style-type	decimal   lower-roman   upper-roman   lower-alpha   upper-alpha ol	

# CSS bordes

La propiedad border permite definir de golpe todos los bordes en una única regla de la hoja de estilos. Se puede utilizar border para definir el o los valores siguientes: **border-width**, **border-style**, **border-color**.

Sintaxis:

```
border: [border-width || border-style || border-color] ;
```

Ejemplo:

```
border: 1px solid #000; □
```

# CSS modelo de caja

La propiedad **height** especifica la altura del área de contenido de un elemento y **width** su ancho.

Ejemplo: `height: 100px; width:100px;`

El área de padding es el espacio entre el contenido del elemento y su borde.

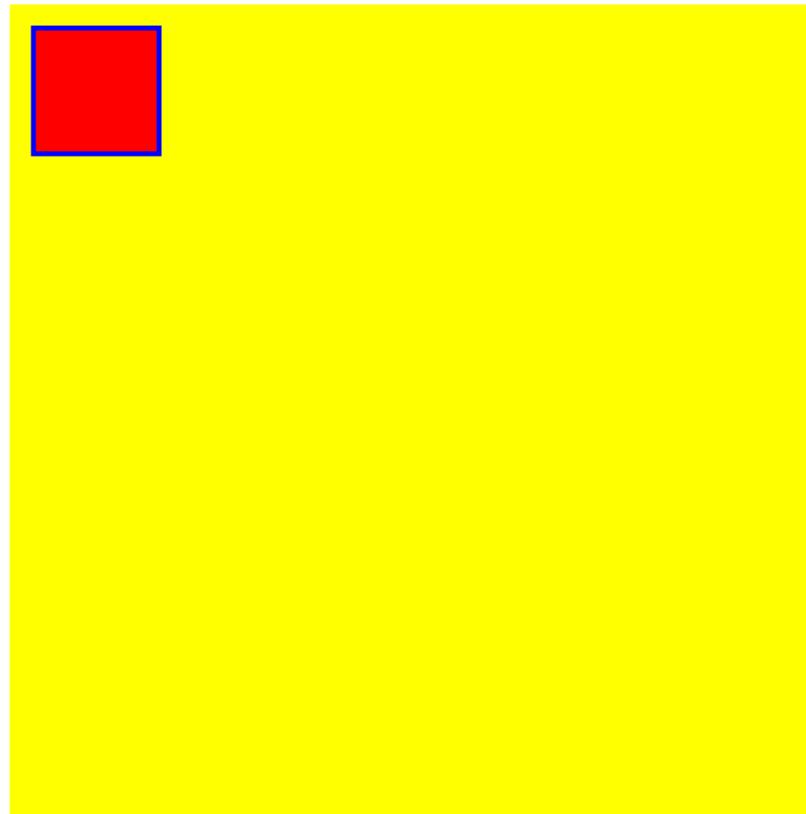
Ejemplo: `padding: 10px;`

La propiedad margin establece el margen para los cuatro lados.

Ejemplo: `margin: 5px;`

# Práctica

Realizar el siguiente HTML (Utilizar position:absolute;)



## CSS float

La propiedad **float** especifica si un elemento debe salir del flujo normal y aparecer a la izquierda o a la derecha de su contenedor, donde los elementos de texto y los en línea aparecerán a su alrededor.

Sintaxis:

**float: left | right | none**

Ejemplo:

`float: left;`

# CSS pseudo-clase

Una **pseudoclase** CSS es una palabra clave que se añade a los selectores y que especifica un estado especial del elemento seleccionado. Por ejemplo, `:hover`

```
<!DOCTYPE html>
<html>
<head>
    <style>
        p:hover {
            background-color: #F89B4D;
        }
    </style>
</head>
<body>
    <p>Lorem ipsum dolor sit amet, c</p>
</body>
</html>
```

# CSS pseudo-classes

- [`:active`](#)
- [`:checked`](#)
- [`:default`](#)
- [`:dir\(\)`](#)
- [`:disabled`](#)
- [`:empty`](#)
- [`:enabled`](#)
- [`:first`](#)
- [`:first-child`](#)
- [`:first-of-type`](#)
- [`:fullscreen`](#)
- [`:focus`](#)
- [`:hover`](#)
- [`:indeterminate`](#)
- [`:in-range`](#)
- [`:invalid`](#)
- [`:lang\(\)`](#)
- [`:last-child`](#)
- [`:last-of-type`](#)
- [`:left`](#)
- [`:link`](#)
- [`:not\(\)`](#)
- [`:nth-child\(\)`](#)
- [`:nth-last-child\(\)`](#)
- [`:nth-last-of-type\(\)`](#)
- [`:nth-of-type\(\)`](#)
- [`:only-child`](#)
- [`:only-of-type`](#)
- [`:optional`](#)
- [`:out-of-range`](#)
- [`:read-only`](#)
- [`:read-write`](#)
- [`:required`](#)
- [`:right`](#)
- [`:root`](#)
- [`:scope`](#)
- [`:target`](#)
- [`:valid`](#)
- [`:visited`](#)

# CSS pseudo-elementos

Al igual que las pseudo-classes, los **pseudo-elementos** se añaden a los selectores, pero en cambio, no describen un estado especial sino que, permiten añadir estilos a una parte concreta del documento.

<p>El estilo solo será aplicado a la primer línea de este párrafo. Después de esto , todo el texto tendrá un estilo normal</p>

```
::first-line { color: blue; text-transform: uppercase; }
```

# CSS pseudo-elementos

- [::after](#)
- [::before](#)
- [::first-letter](#)
- [::first-line](#)
- [::selection](#)
- [::backdrop](#)
- [::placeholder](#)
- [::marker](#)
- [::spelling-error](#)
- [::grammar-error](#)

# CSS Diseño responsivo

Página web que responde diferente en diversos dispositivos, ya sea por tamaño o por alguna condición.

## VIEWPORT.

```
<meta name="viewport"  
      content="width=device-width, initial-scale=1.0">
```



Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat. Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan et iusto odio dignissim qui blandit praesent luptatum zzril delenit augue duis dolore te feugait nulla facilisi. Nam liber tempor cum soluta nobis eleifend option congue nihil imperdiet domino

# CSS Diseño responsivo

**Media Queries.** Nos sirven para modificar una página web o aplicación en función del tipo de dispositivo (como una impresora o una pantalla) o de características y parámetros específicos (como la resolución de la pantalla o el ancho del viewport del navegador).

```
body { background-color: red; }
@media (min-width: 640px) { // A partir de 640px en adelante
    body { background-color:green; }
}
@media only screen and (max-width: 800px) { ... }
```

[https://developer.mozilla.org/es/docs/CSS/Media\\_queries](https://developer.mozilla.org/es/docs/CSS/Media_queries)

# Ejemplo Media Query

```
<!DOCTYPE html>
<html>
<head>
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<style>
body {
    background-color: lightgreen;
}
@media only screen and (max-width: 600px) {
    body {
        background-color: lightblue;
    }
}
</style>
</head>
<body>
<p>Redimensiona la ventana, cuando el ancho del documento sea 600 pixeles o menos el color de fondo será lightblue de otra manera será lightgreen</p>
</body>
</html>
```

# Ejemplo Responsivo

## Chania

[The Flight](#)[The City](#)[The Island](#)[The Food](#)

### The City

Chania is the capital of the Chania region on the island of Crete. The city can be divided in two parts, the old town and the modern city.

#### What?

Chania is a city on the island of Crete.

#### Where?

Crete is a Greek island in the Mediterranean Sea.

#### How?

You can reach Chania airport from all over Europe.

Resize the browser window to see how the content respond to the resizing.

## Chania

[The Flight](#)[The City](#)[The Island](#)[The Food](#)

## The City

Chania is the capital of the Chania region on the island of Crete. The city can be divided in two parts, the old town and the modern city.

#### What?

Chania is a city on the island of Crete.

#### Where?

Crete is a Greek island in the Mediterranean Sea.

#### How?

You can reach Chania airport from all over Europe.

# Preprocesadores CSS

Un preprocesador CSS es un programa que te permite generar CSS a partir de la syntaxis única del preprocesador.

Añadiran algunas características que no existen en CSS puro, como variables, mixins, selectores anidados, entre otros. Estas características hacen la estructura de CSS más legible y fácil de mantener.

Para utilizar un preprocesador CSS debes instalar un compilador CSS en tu web.

# Preprocesadores CSS

Estos son algunos de los preprocesadores CSS más populares:

- [SASS](#)
- [LESS](#)
- [Stylus](#)
- [PostCSS](#)

# Preprocesadores CSS

Ejemplo variables SASS:

```
$myFont: Helvetica, sans-serif;  
$myColor: red;  
$myFontSize: 18px;  
$myWidth: 680px;  
  
body { font-family: $myFont; font-size: $myFontSize; color: $myColor; }  
#container { width: $myWidth; }
```

[https://www.w3schools.com/sass/showsass.asp?filename=demo\\_sass\\_var1](https://www.w3schools.com/sass/showsass.asp?filename=demo_sass_var1)

# Práctica

## Aplicar estilos al formulario de la práctica de forms

### Registro

Por favor llene el formulario para crear un artículo.

Titulo:	eeweerqwer
Autor:	Ingrese el autor
Tipo:	Noticia ▾
Fecha publicación:	dd/mm/aaaa
Texto:	

**Guardar** **Limpiar** **Cancelar**

Abajo del formulario anterior generar el html del listado de artículos para que se vea de la siguiente manera

Noticia Aviso Agregar

## Frameworks CSS

Crear CSS consistente, conciso y efectivo puede ser bastante trabajo. Hay tantas cosas a considerar como capacidad de respuesta, accesibilidad y estructura. ¡Esto es exactamente por qué existe CSS Frameworks!

- [Bulma](#)
- [Tailwind CSS](#)
- [Bootstrap 4](#)
- [Semantic UI](#)
- [Foundation](#)
- [Materialize CSS](#)

<https://scotch.io/bar-talk/6-popular-css-frameworks-to-use-in-2019>



# JAVA SCRIPT



JavaScript



# Javascript



## Historia

- Creado por Brendan Eich en 1995 en Netscape Communications
- Inicialmente iba a llamarse LiveScript, pero fue renombrado para tratar de aprovechar la popularidad del lenguaje Java de Sun Microsystems.
- Netscape presentó el lenguaje a Ecma Internacional, un organismo europeo de normalización, lo que dió lugar a la primera edición de ECMAScript en 1997.



# Javascript



## Definición

- JavaScript es un lenguaje dinámico orientado a objetos. Tiene tipos y operadores, objetos básicos y métodos. Su sintaxis viene de dos lenguajes, que son: Java y C.
- Es un lenguaje de programación diseñado originalmente para HTML y Web
- JavaScript no tiene un concepto de entrada o salida. Está diseñado para funcionar como un lenguaje de script dentro de un entorno y depende de los mecanismos de éste para la comunicación con el mundo exterior.
- Una de las principales diferencias es que JavaScript no tiene **Clases**, en cambio, la funcionalidad de Clase se consigue mediante prototipos de objetos.
- Otra diferencia es que las funciones son **Objetos**



# Javascript



## Versiones

Ver	Official Name
1	ECMAScript 1 (1997)
2	ECMAScript 2 (1998)
3	ECMAScript 3 (1999)
4	ECMAScript 4
5	ECMAScript 5 (2009)
5.1	ECMAScript 5.1 (2011)
6	ECMAScript 2015
7	ECMAScript 2016
8	ECMAScript 2017
9	ECMAScript 2018

Referencia:

<https://www.ecma-international.org/publications/standards/Ecma-262.htm>

<https://www.ecma-international.org/publications/standards/Ecma-262-arch.htm>

<http://kangax.github.io/compat-table/es5/>



# Javascript



## Tipos de Datos

**Numéricos.**- Se utilizan para almacenar valores numéricos enteros o decimales. Los números en JavaScript son "de doble precisión de 64 bits en formato IEEE 754", de acuerdo con la especificación.

```
let num = 16; // variable tipo entero
let total = 224.65; // variable tipo decimal
```

Hay que ser un poco cuidadoso con cosas como:

$$0.1 + 0.2 == 0.3000000000000004$$

## Tipos de Datos (Cadenas de texto)

**Cadenas de texto.**- Se utilizan para almacenar caracteres, palabras y/o frases de texto, se encierra el valor entre comillas dobles o simples.

```
let cadena1 = "texto 1";
let cadena2 = 'texto 2';
```

Las cadenas de caracteres en JavaScript son secuencias de Caracteres Unicode, con cada carácter representado por un número de 16 bits.

Para saber la longitud de una cadena, se debe acceder a su propiedad **length**.

```
> "hola".length
4
```

## Tipos de Datos (Cadenas de texto)

También se pueden usar cadenas como si fueran objetos. Ellas también tienen métodos:

```
> "hola".charAt(0)
h

> "hola, mundo".replace("hola", "adiós")
adiós, mundo

> "hola".toUpperCase()
HOLA
```



# Javascript



## Tipos de Datos

**lógico.**- Una variable de tipo lógico (boolean) almacena un tipo especial de valor que solamente puede tomar dos valores: **true** o **false**.

```
let mostrarTabla = true;
```

**undefined.**- Cualquier variable no definida tiene como valor un tipo **undefined**.

```
let variableSinDefinir;
console.log(variableSinDefinir);
```

**null.**- El valor null es un literal de Javascript que representa un valor nulo o "vacío"

```
let foo = null;
```



# Javascript



## Tipos de Datos

**Arrays.**- Un array es una colección de variables, que pueden ser todas del mismo tipo o cada una de un tipo diferente.

```
let arregloVacio = [];
let arreglo = new Array(2);
let dias = ["Lunes", "Martes", "Miércoles", "Jueves", "Viernes", "Sábado", "Domingo"];
let d = new Array("Lunes", "Martes");
```

**Date.**- Es un objeto utilizado para trabajar con fechas y horas.

```
let d = new Date();
let d2 = new Date('Nov 03, 2018');
```



# Javascript



## Tipos de Datos

**NaN.**- Un valor representando un Not-A-Number (No es Un Número), es utilizado para indicar una condición de error

```
var test = Number.parseInt('a');
console.log(test);
```



# JavaScript



## Parsers

Se puede convertir una cadena de texto a un entero utilizando la función **parseInt()**. Esta función toma la base para la conversión como un segundo argumento opcional, que siempre se debería proporcionar.

```
> parseInt("123", 10)  
123
```

```
> parseInt("010", 10)  
10
```



# JavaScript



## Parsers

Si no se proporciona el valor base, se podrían obtener resultados sorpresivos en navegadores antiguos (anteriores a 2013).

```
> parseInt("010")
8
```

Esto sucede porque la función **parseInt** decidió tratar la cadena de texto como un octal por la presencia del 0 inicial.



# JavaScript



## Parsers

Si se quiere convertir un número binario a entero, sólo se cambia la base.

```
> parseInt("11", 2)  
3
```

También se pueden cambiar números flotantes usando la función **parseFloat()** que utiliza la base 10.



# JavaScript



## Palabras reservadas

arguments, break, case, catch, class, const, continue, debugger, default, delete, do, else, enum, export, extends, false, finally, for, function, if, implements, import, in, instanceof, interface, let, new, null, package, private, protected, public, return, static, super, switch, this, throw, true, try, typeof, var, void, while.

Infinity, NaN, undefined.



# JavaScript



## Variabls

Una variable es un elemento que se emplea para almacenar y hacer referencia a otro valor. Las variables en JavaScript se crean mediante la palabra reservada var ó let.

```
var numero_1 = 3;
```

```
let x = 20;
```



# JavaScript



## Variablos

El nombre de una variable debe cumplir las siguientes normas:

- Sólo puede estar formado por letras, números y los símbolos \$ (dólar) y \_ (guión bajo).
- El primer carácter no puede ser un número.

```
// Ejemplos validos
let _$letra;
número
let $$$otroNumero;
punto
```

```
// Ejemplos inválidos
let 1numero; // Empieza por un
              .
let .numero; // Empieza con un
```



# JavaScript



## Comentarios

El comentario de una sola línea inician con `//`, cualquier texto entre `//` y el fin de linea sera ignorado

```
// Ejemplo comentario una sola linea
```

Los comentarios multilínea inician con `/*` y terminan con `*/`, cualquier texto entre `/*` y `*/` será ignorado.

```
/*
Ejemplo de comentario
multilínea
*/
```



# JavaScript



## Operadores Aritméticos

Operador	Descripción
+	Suma
-	Resta
*	Multiplicación
**	Exponenciación( <a href="#">ES2016</a> )
/	División
%	Modulo (Resto después de división)
++	Incremento
--	Decremento



# JavaScript



## Operadores Asignación

Operador	Ejemplo	Igual a
=	$x = y$	$x = y$
$+=$	$x += y$	$x = x + y$
$-=$	$x -= y$	$x = x - y$
$*=$	$x *= y$	$x = x * y$
$/=$	$x /= y$	$x = x / y$
$%=$	$x %= y$	$x = x \% y$
$**=$	$x **= y$	$x = x ** y$



# JavaScript



## Sintaxis

**var** x; // Declaración, ";" es opcional al final de la línea  
// Identificadores: \_a , \$b , x , n0

// Asignación de variables  
x = 3 + y;

// Llamado a función con parámetros, aunque la función no tenga parámetros, se pueden enviar  
foo(x, y);  
// Llamando una función del objeto  
obj.bar(3);

// Condiciones y sentencias de control// Si `x` es igual a 0? === (sin casting) no es lo mismo == (con casting)  
**if** (x === 0) { x = 123;  
}



# JavaScript



## Sintaxis

```
for (var i=0; i < arr.length; i++) {  
    console.log(arr[i]);  
}
```

```
// Definición de funciones  
function baz(a, b) {  
    return a + b;  
}
```

```
// Tipos e instancias  
typeof true; // 'boolean'  
'test' instanceof String //true  
  
// Boolean  
'', false, 0, undefined, null, NaN ---> false
```



# JavaScript



## Práctica

Realizar las funciones que calculen el área y perímetro de cada una de las siguientes figuras su nombre será así:  
**calcularAreaFigura** y **calcularPerimetroFigura** ejecutar funciones parecido a esto:

```
console.log('areaCuadrado = ' + calcularAreaCuadrado(5));
```

### Cuadrado

Área: longitudLado<sup>2</sup>

Perímetro: longitudLado \* 4

### Rectángulo

Área: ancho \* alto

Perímetro: 2 \* ancho + 2 \* alto

### Círculo

Área:  $\pi * radio^2$

Perímetro:  $2 * \pi * radio$

### Elipse

Área:  $\pi * semiEjeMayor * semiEjeMenor$

Perímetro:  $2 * \pi * ((semiEjeMayor^2 + semiEjeMenor^2) / 2)$



# JavaScript



## Arreglos Métodos

### [filter\(\)](#)

Crea una nueva matriz unidimensional con todos los elementos de esta matriz unidimensional para los cuales la función de filtrado provista devuelve true.

```
let numbers = [45, 4, 9, 16, 25];
let mayor18 = numbers.filter((value, index, array) => {
  return value > 18
});
console.log(mayor18);
```



# JavaScript



## Arreglos Métodos forEach()

Invoca a una función por cada elemento en la matriz unidimensional.

```
let txt = '';
let numbers = [45, 4, 9, 16, 25];
numbers.forEach((value, index, array) => {
  txt += value + '|';
});
console.log(txt);
```



## Arreglos Métodos [map\(\)](#)

Crea una nueva matriz unidimensional con los resultados de la invocación de una función provista sobre cada elemento en esta matriz unidimensional.

```
let frutas = ['Fresa', 'Sandía', 'Melón', 'Mango'];
let opciones = frutas.map((value, index, array) => {
    return { value: index + 1, label: value}
});
console.log(opciones);
```



# JavaScript



## Arreglos Métodos [some\(\)](#)

Retorna verdadero true si al menos un elemento en esta matriz unidimensional satisface la función de pruebas provista.

```
let numbers = [45, 4, 9, 16, 25];
let menor10 = numbers.some((value, index, array) => {
  return value < 10
});
console.log(menor10);
```

Referencia:

[https://www.w3schools.com/js/js\\_array\\_methods.asp](https://www.w3schools.com/js/js_array_methods.asp)



## Arreglos Métodos reduce()

Ejecuta una función proporcionada para cada valor de la matriz (de izquierda a derecha), el método reduce la matriz a un solo valor

```
let numbers = [175,50,25];
let total = numbers.reduce((total, currentValue, currentIndex, array) => {
  return total - currentValue;
});
console.log(total);
```

Referencia:

[https://www.w3schools.com/js/js\\_array\\_methods.asp](https://www.w3schools.com/js/js_array_methods.asp)



# JavaScript



## Práctica

Del siguiente arreglo obtener un arreglo con objetos {label:'asentamiento', value:'indice'} de los asentamientos que sean de la Ciudad de México.

```
let asentamientos = [{cp: '06030', asentamiento: 'Tabacalera'},  
  {cp: '55010', asentamiento: 'El diamante'},  
  {cp: '55070', asentamiento: 'Villas del Sol'},  
  {cp: '06200', asentamiento: 'Morelos'},  
  {cp: '06350', asentamiento: 'Buenavista'},  
  {cp: '55020', asentamiento: '10 de Abril'}];
```



# JavaScript



## Alcance

El alcance de una variable puede ser de dos tipos, **global** y **local**.

### Global

Una variable cuyo scope es global se puede acceder desde cualquier parte del código. Ejemplo:

```
var a = 1;
function global() {
    console.log(a);
}
global();
console.log(a);
```



# JavaScript



## Alcance

Local

Una variable local solo se puede acceder desde la función que la contiene.

Ejemplo:

```
function local() {  
    var a = 2;  
    console.log(a);  
}  
local();  
console.log(a);
```



# JavaScript



## Alcance

La instrucción **let** declara una variable de alcance local con ámbito de bloque.

```
if (true) {  
    let x = 0 console.log(x);  
}  
console.log(x);
```



# JavaScript



## IIFE Pattern

### Immediately-Invoked Function Expression (IIFE)

A veces queremos simular un bloque de código, por ejemplo, para evitar que una variable se convierta en global. El patrón para hacerlo se llama IIFE. Por ejemplo:

```
(function () { // open IIFE
    var tmp = ...; // not a global variable
}()); // close IIFE
```

En el ejemplo se puede ver una expresión de función que se llama de inmediato. Los paréntesis evitan que se interprete como una declaración de función; sólo las expresiones de función pueden ser invocadas inmediatamente. El cuerpo de la función introduce un nuevo alcance y evita que **tmp** se vuelva global.



# JavaScript



## Callback

Son funciones que se pasan como parámetros de otras funciones y que se ejecutan dentro de éstas.

```
function funcion1(dato, callback) {
    callback(dato);
}
funcion1(1, function (x) { alert(x); });
//-----
function funcion1(seconds, callback) {
    setTimeout(callback, 1000 * seconds)
}
funcion1(2, function () { alert('Pasaron algunos segundos'); ));
```



# JavaScript



## Función Flecha

Las funciones de flecha nos permiten escribir una sintaxis de función más corta, y siempre son anónimas.

Estas funciones son funciones no relacionadas con métodos y no pueden ser usadas como constructores.

```
saludo = function() {  
    return "Hola Mundo";  
}  
console.log(saludo());
```

```
saludo = () => {  
    return "Hola Mundo";  
}  
console.log(saludo());
```



# JavaScript



## Closures

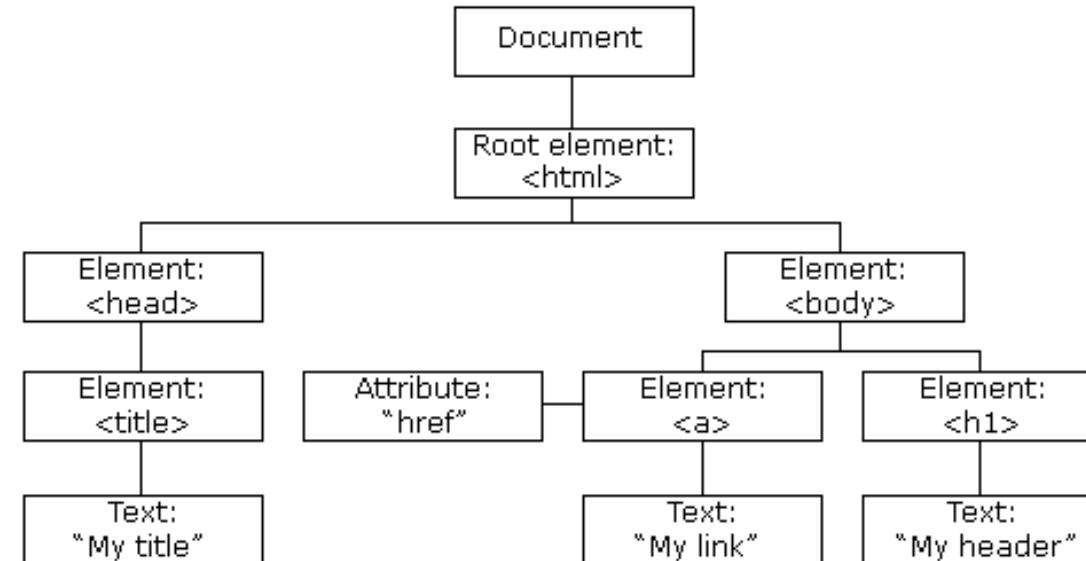
Closure es una función con conexión a las variables de sus ámbitos circundantes, lo que devuelve **createIncrementor** es un closure. Cada función está conectada a las variables de las funciones que la rodean, incluso después de que deja el alcance que la creó. Por ejemplo:

```
function createIncrementor(start) {
  return function () { // (1)
    start++;
    return start;
  }
}
var inc = createIncrementor(5);
inc();
> 6
inc()
> 7
```

La función que comienza en la línea (1) deja el contexto en el que se creó, pero permanece conectada a una versión en vivo de la variable **start**

## DOM de HTML (Modelo de Objetos del Documento)

El DOM es un API para documentos [HTML](#) y [XML](#). Proporciona una representación estructural del documento, permitiendo la modificación de su contenido o su presentación visual.





# JavaScript



## DOM de HTML

- En el DOM, todos los elementos HTML se definen como objetos .
- La interfaz de programación son las propiedades y métodos de cada objeto.
- Una propiedad es un valor que puede obtener o establecer (como cambiar el contenido de un elemento HTML).
- Un método es una acción que puede hacer (como agregar o eliminar un elemento HTML).



# JavaScript



## Encontrar Elementos

El atributo id nos puede permitir acceder a un nodo del DOM con dicho valor de atributo, para poder hacerlo utilizamos la siguiente función:

- **getElementById**

Ejemplo:

Para el siguiente elemento:

```
<div id='div_id'>div con id='div_id'</div>
```

Lo podríamos obtener de la siguiente forma:

```
document.getElementById("div_id");
```



# JavaScript



## Encontrar Elementos

El nombre de un tag nos puede permitir acceder a los elementos del DOM con ese nombre, para poder hacerlo utilizamos la siguiente función:

- `getElementsByName`

Ejemplo:

Para el siguiente elemento:

```
<p>parrafo1</p>
<p>parrafo2</p>
```

|

Lo podríamos obtener de la siguiente forma:

```
document.getElementsByName ("p");
```



# JavaScript



## Encontrar Elementos

Podemos buscar todos los elementos que tengan un mismo nombre de clase, para poder hacerlo utilizamos la siguiente función:

- `getElementsByClassName`

Ejemplo:

Para el siguiente elemento:

```
<p class="item">parrafo1</p>
<p>parrafo2</p>
<p class="item">parrafo3</p>
```

|

Lo podríamos obtener de la siguiente forma:

```
document.getElementsByClassName("item");
```



# JavaScript



## Encontrar Elementos por selectores CSS

Los selectores CSS son patrones que se utilizan para seleccionar los elementos a los que se desea aplicar estilo, en Javascript podemos obtener elementos usando la misma sintaxis con las siguientes funciones:

- querySelector
- querySelectorAll



# JavaScript



## Encontrar Elementos por selectores CSS

**querySelector** podemos obtener el primer elemento del DOM que haga match con el selector que le pasamos. Por ejemplo:

```
<ul id="ejemplo1">
  <li id="uno">uno</li>
  <li id="dos">dos</li>
</ul>

document.querySelector('#ejemplo1 li');
// nos trae <li id="uno">uno</li>
```



# JavaScript



## Encontrar Elementos por selectores CSS

**querySelectorAll** funciona de la misma manera que **querySelector**, pero nos trae todos los elementos que coincidan con la condición dada.

```
<ul id="ejemplo2">
  <li id="uno">uno</li>
  <li id="dos">dos</li>
</ul>
```

```
document.querySelectorAll('#ejemplo2 li');
```

```
/*
```

Nos trae un array con los elementos que cumplan la condición anterior:

```
[ <li id="uno">uno</li>, <li id="dos">dos</li> ]
```



# JavaScript



## Ventanas emergentes

**Alert.**- A menudo se usa un cuadro de alerta si desea informar algo al usuario.

```
alert("Soy un alert!");
```

**Confirm.**- Un cuadro de confirmación se usa a menudo si desea que el usuario verifique o acepte algo.

```
var txt = '';
if (confirm("Presione un botón!")) {
    txt = "Presionaste Aceptar!";
} else {
    txt = "Presionaste Cancelar!";
}
console.log(txt);
```



## Ventanas emergentes

**Prompt** .- A menudo se usa un cuadro de solicitud si desea que el usuario ingrese un valor antes de ingresar una página.

```
var txt;
var person = prompt("Por favor escribe tu nombre:", "Rebeca");
if (person == null || person == "") {
    txt = "El usuario canceló.";
} else {
    txt = "Hola " + person + "! Como estas hoy?";
}
console.log(txt);
```



# JavaScript



## Eventos

Un evento es una acción determinada que Javascript puede detectar, estos pueden haber sido provocados por la acción del usuario o por el navegador.

- **addEventListener** Registra un evento a un objeto en específico, el objeto específico puede ser un simple [elemento](#) en un archivo, el mismo [documento](#), una [ventana](#) o un [XMLHttpRequest](#).
- **removeEventListener** Elimina el controlador de eventos que se ha adjuntado con el método addEventListener.



# JavaScript



## Eventos

- click
- dblclick
- mousedown
- onmouseup
- mouseover
- mousemove
- mouseout
- keypress
- keydown
- keyup
- focus
- blur
- load
- unload
- change
- select

Referencia:

<https://developer.mozilla.org/es/docs/Web/Events>



# JavaScript



## Eventos (click)

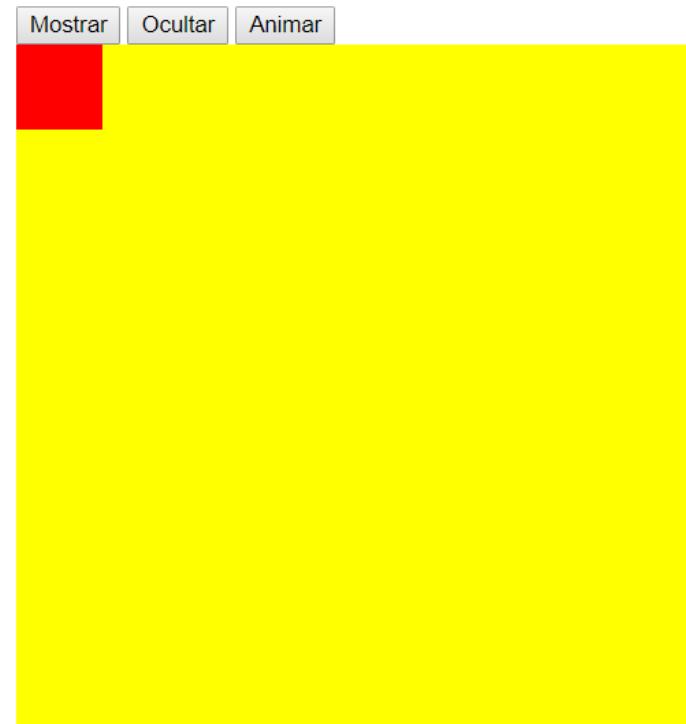
```
<button id="btn1">Click Me!</button>

let miFuncion = function() {
    console.log("has pulsado el botón 1");
}
document.getElementById("btn1").addEventListener("click", miFuncion );

document.getElementById("btn1").removeEventListener("click", miFuncion );
```

## Práctica

Realizar el html y javascript para que al dar click al botón mostrar muestre el cuadro amarillo, al dar click en ocultar oculte y al dar click en animar mueva el cuadro rojo





# JavaScript



## Crear Elementos

El método **createElement(tagName)** crea un elemento HTML especificado por su tagName

```
let ul = document.getElementById("contenedor");
let li = document.createElement("li");
ul.appendChild(li);
```

El método **createTextNode()** crea un nuevo nodo de texto.

```
let newtext = document.createTextNode("nuevoTexto"),
p1 = document.getElementById("p1");
p1.appendChild(newtext);
```



# JavaScript



## Práctica

Realizar el html y javascript para que en un input se pueda escribir el texto de una fruta y cada vez que se dé click en agregar haga otro registro en la lista

Fruta  Agregar

- Fresa
- Melón

Esta página dice

El campo esta vacío

Aceptar



# JavaScript



## Plantillas de cadena de texto

Son literales de texto que habilitan el uso de expresiones incrustadas. Es posible utilizar cadenas de texto de más de una línea, y funcionalidades de interpolación de cadenas de texto con ellas.

Se delimitan con el carácter de comillas o tildes invertidas (` `)

```
console.log(`línea 1 de la cadena de texto  
línea 2 de la cadena de texto`);
```



# JavaScript



## Plantillas de cadena de texto

Interpolación de expresiones `cadena de texto \${expresión} texto`

- Cadenas de texto normales.

```
let a = 5,  
    b = 10;  
  
console.log("Quince es " + (a + b) + " y\nno " + (2 * a + b) + ".");
```

- Plantillas de cadena de texto.

```
let a = 5,  
    b = 10;  
  
console.log(`Quince es ${a + b} y\nno ${2 * a + b}.`);
```



# JavaScript



## Ejemplo

```
<div id="frutaSeleccionada">Seleccione una fruta</div>

<select id="frutas">
</select>

let frutas = [{id:1,nombre:'Mango'}, {id:2,nombre:'Sand&iacute;a'}, {id:3,nombre:'Fresa'},
{id:4,nombre:'Mel&oacute;n'}, {id:5, nombre:'Naranja'}];
let selectFrutas = document.getElementById("frutas");
let options = '';
frutas.forEach((e,i,a) => { options += `<option value='${e.id}'>${e.nombre}</option>` ; });
selectFrutas.innerHTML = options;
selectFrutas.addEventListener("change", (e) => {
    let el = e.target;
    let divFrutaSeleccionada = document.getElementById("frutaSeleccionada");
    let text = el.options[el.selectedIndex].innerHTML;
    divFrutaSeleccionada.innerHTML = `valor= <b>${el.value}</b>, etiqueta:<b>${text}</b>`;
});
```



# JavaScript



## Práctica

Colocar 2 select's uno de departamento y otro de categoría, cada vez que se cambie el departamento se deben mostrar las categorías de ese departamento. La información de los select's será llenada de manera dinámica a partir del siguiente arreglo:

```
let categorias = [
    {id: 1, nombre:'Laptops',departamento:'Electrónica'},
    {id: 2, nombre:'Pantallas',departamento:'Electrónica'},
    {id: 3, nombre:'Zapatillas',departamento:'Mujer'},
    {id: 4, nombre:'Peluches',departamento:'Niño'},
    {id: 5, nombre:'Carriolas',departamento:'Niño'},
    {id: 6, nombre:'Sillas',departamento:'Hogar'}
];
```



# JavaScript



## Práctica

Sobre el ejercicio anterior llenar una tabla de manera dinámica, los registros de esta tabla serán filtrados con los select's del ejercicio anterior. Los datos con los que se llenará la tabla son:

```
let productos = [
    {descripcion:'MacBook Air 13.3", Intel Core i5 dual core, RAM 8 GB, SSD 128 GB, Gris Espacial', precio:18899, marca:'Apple', categoriaId:1},
    {descripcion:'Laptop Pavilion 15.6", AMD Ryzen 5 2500U, RAM 12 GB, DD 1 TB, SSD 128 GB, Azul', precio:17099, marca:'HP', categoriaId:1},
    {descripcion:'Bundle Laptop, Inspiron 5481, 14", Core I3, RAM 4 GB, DD 1 TB, Plata + Mochila', precio:9899, marca:'Dell', categoriaId:1},
    {descripcion:'Pantalla 55" LED UHD 4K Series 7', precio:12811, marca:'Samsung', categoriaId:2},
    {descripcion:'Pantalla Lg 86P Tv Ai Thinq Uhd 4K86UK6570PUA', precio:12811, marca:'LG', categoriaId:2},
    {descripcion:'Zapatillas tacón de aguja', precio:2399, marca:'STEVE MADDEN', categoriaId:3},
    {descripcion:'Sandalias con tacón alto', precio:2199, marca:'WESTIES', categoriaId:3},
    {descripcion:'Oso Gigante', precio:15399, marca:'DOUDOU ET COMPAGNIE', categoriaId:4},
    {descripcion:'Carriola LiteWay 3', precio:3199, marca:'CHICCO', categoriaId:5}
];
```



# JavaScript



## Clases y objetos

Javascript genera objetos basados en prototipos

### ECMAScript 5 y anteriores

```
function Carro(marca) {  
    this.marca = marca;  
}  
  
console.log(new Carro("Tesla"));
```

### ECMAScript 6

```
class Carro {  
    constructor(marca) {  
        this.marca = marca;  
    }  
}  
  
console.log(new  
Carro("Tesla"));
```



# JavaScript



## Métodos

### ECMAScript 5 y anteriores

```
function Carro(marca) {  
    this.marca = marca;  
}  
Carro.prototype.imprime = function() {  
    alert(this.marca);  
};  
new Carro("Tesla").imprime();
```

### ECMAScript 6

```
class Carro {  
    constructor(marca) {  
        this.marca = marca;  
    }  
    imprime () {  
        alert(this.marca);  
    }  
}  
new Carro("Tesla").imprime();
```



# JavaScript



## Notación JSON

```
let carro = {
    marca: 'Tesla',
    imprime() {
        alert(this.marca);
    }
}
carro.imprime();
```



# JavaScript



## Subclases

La palabra clave **extends** es usada en declaraciones de clase o expresiones de clase para crear una clase hija.

```
class Animal {  
    constructor(nombre) {  
        this.nombre = nombre;  
    }  
    hablar() {  
        console.log(this.nombre + ' hace un ruido.');//  
    }  
}  
  
class Perro extends Animal {  
    hablar() {  
        console.log(this.nombre + ' ladra.');//  
    }  
}
```



# JavaScript



## Métodos estáticos

Los métodos estáticos son llamados sin instanciar su clase. Son habitualmente utilizados para crear funciones para una aplicación.

```
class Carro {  
    constructor() {  
    }  
    static imprime () {  
        alert('metodo estatico');  
    }  
}  
Carro.imprime();
```



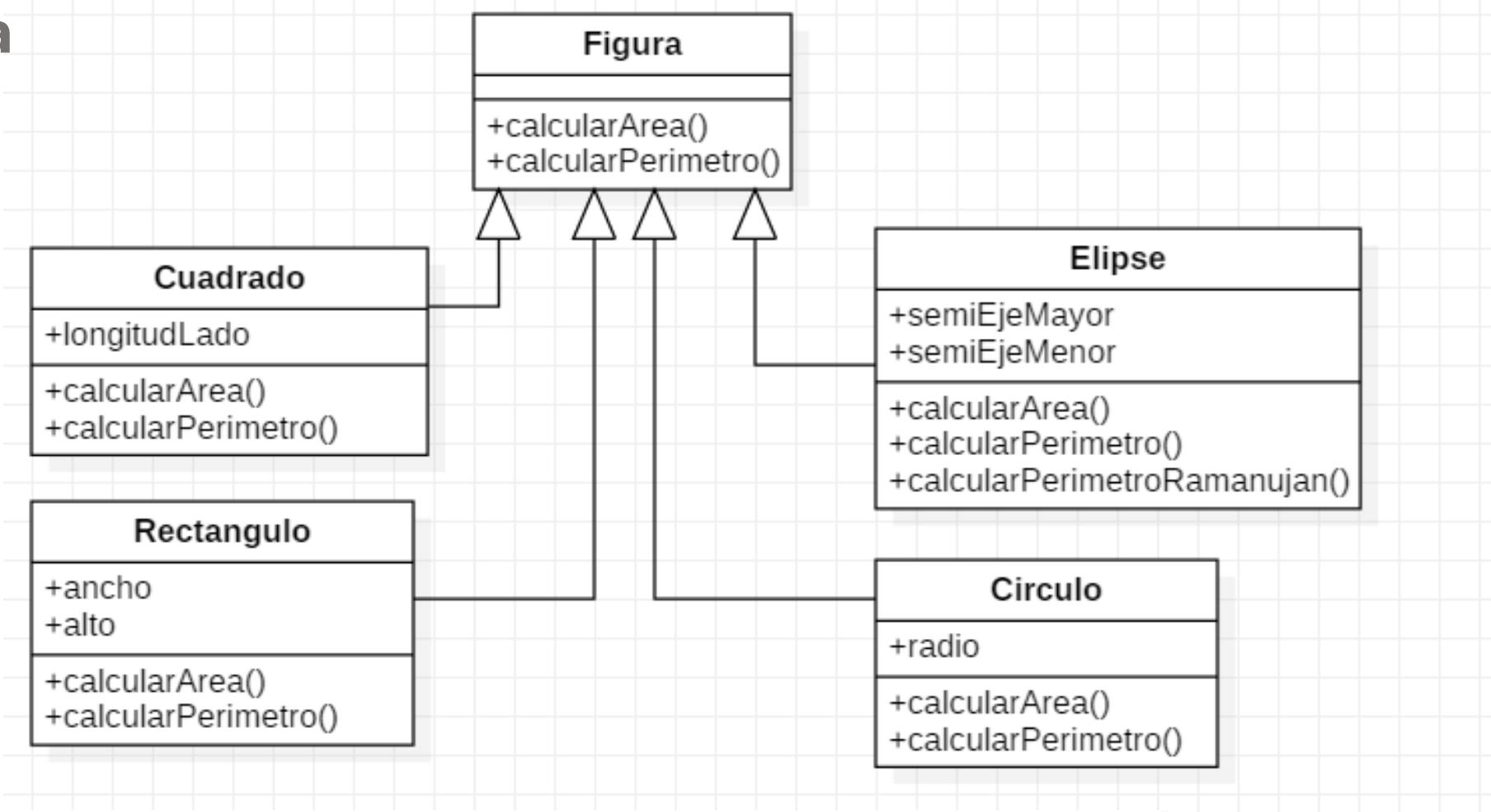
# JavaScript



## Setter Getter en ECMAScript

```
class Carro {  
    constructor(marca) {  
        this.marca = marca;  
    }  
    get marca() {  
        return this._marca;  
    }  
    set marca(value) {  
        this._marca = value;  
    }  
}  
alert((new Carro("Tesla")).marca);
```

## Práctica



*superclase* permite generalizar atributos y métodos evitando la repetición de código y unificando procesos



# JavaScript



## Forms validación

Todos los elemento **<input>** pueden ser validados usando el atributo **pattern**. Este atributo espera como valor una **Expresión Regular**, sensible a uso de mayúsculas. Si el valor del elemento no es vacío y no coincide con la expresión regular especificada en el atributo **pattern**, el elemento es considerado inválido.



# JavaScript



## Ejemplo

```
input:invalid { border: 1px solid red; }  
input:valid { border: 1px solid green; }
```

```
<form>  
<label for="choose">¿Preferirías un plátano o una cereza?</label>  
<input id="choose" name="i_like" pattern="plátano|cereza"> <button>Enviar</button>  
</form>
```

## Forms required

Si un elemento requiere un valor antes de que el formulario sea enviado, puedes marcar al elemento usando el atributo **required**. Cuando este atributo es verdadero, no se permite que el campo esté vacío. Ejemplo:

```
<input id="choose" name="i_like" pattern="plátano|cereza"  
required>
```

**Todos los tipos de elemento que pueden recibir entrada del usuario (`<textarea>`, `<select>`, etc.) soportan el atributo **required**, pero cabe aclarar que el elemento `<textarea>` no soporta el atributo **pattern**.**



## Forms otras restricciones

Todos los campos de texto (`<input>` or `<textarea>`) pueden ser restringidos en tamaño usando el atributo **maxlength**. Un campo es inválido si su valor es mayor al valor del atributo **maxlength**. La mayoría de las ocasiones, sin embargo, los navegadores no permiten que el usuario escriba un valor más largo del indicado.

Para campos numéricos, los atributos **min** y **max** también proveen una restricción de validación. Si el valor del campo es menor que el del atributo **min** o mayor que el del atributo **max**, el campo será inválido.



# JavaScript



## Forms propiedades de la API de validación.

`validity.patternMismatch`

Devuelve `true` si el valor del elemento no cumple el patrón indicado; de lo contrario, devuelve `false`.

`validity.valid`

Devuelve `true` si el valor del elemento no tiene problemas de validación; de lo contrario, devuelve `false`.

`validity.rangeOverflow`

Devuelve `true` si el valor del elemento es mayor al máximo permitido; de lo contrario, devuelve `false`.

`validity.rangeUnderflow`

Devuelve `true` si el valor del elemento es menor al mínimo permitido; de lo contrario, devuelve `false`.

`validity.tooLong`

Devuelve `true` si el valor del elemento es mayor al indicado como longitud máxima; de lo contrario, devuelve `false`.



# JavaScript



## Forms submit

```
<form novalidate>
    <label for="mail">Quisiera que me proporcionaras tu dirección de correo electrónico</label>
    <input type="email" id="mail" name="mail">
    <span class="error"></span>
    <button>Enviar</button>
</form>
form.addEventListener("submit", function (event) {
    // Cada vez que el usuario intenta enviar los datos, verificamos si el campo de correo es válido.
    if (!email.validity.valid) {
        // Si el campo no es válido, mostramos un mensaje de error.
        error.innerHTML = "¡Yo esperaba una dirección de correo";
        error.className = "error active";
        // Y prevenimos que el formulario sea enviado, cancelando el evento
        event.preventDefault();
    }
});
```



# JavaScript



## Promesas

Las promesas se utilizan para la realización de tareas asíncronas, como por ejemplo la obtención de respuesta a una petición HTTP.

Una promesa puede tener 4 estados:

- **Pendiente**: Es su estado inicial, no se ha cumplido ni rechazado.
- **Cumplida**: La promesa se ha resuelto satisfactoriamente.
- **Rechazada**: La promesa se ha completado con un error.
- **Arreglada**: La promesa ya no está pendiente. O bien se ha cumplido, o bien se ha rechazado.



# JavaScript



## Crear Promesa

```
new Promise(function(resolve, reject) { ...  
} );
```

Cuando creamos una **promise**, le pasamos una función en cuyo interior deberían producirse las operaciones asíncronas, que recibe 2 argumentos:

- **Resolve**: Es la función que llamaremos si queremos resolver satisfactoriamente la promesa.
- **Reject**: Es la función que llamaremos si queremos rechazar la promesa.



## Consumir Promesa

Para consumir una promesa tenemos 2 métodos:

- **catch(onRejected)**: Añade un callback que se ejecutará si la promise es rechazada, y devuelve la promise actualizada.
- **then(onFulfilled, onRejected)**: Añade un callback para caso de éxito, y otro para caso de error y devuelve la promise actualizada.



# JavaScript



## Ejemplo

```
let p = new Promise(function(resolve, reject){  
    let t = setTimeout(function(){  
        resolve("bien");  
        //reject("rechazado");  
    }, 3000);  
});
```

```
p.then(function(value) {  
    console.log("todo bien " + value);  
},  
function(value) {  
    console.log("algo salio mal " + value);  
});
```



# JavaScript



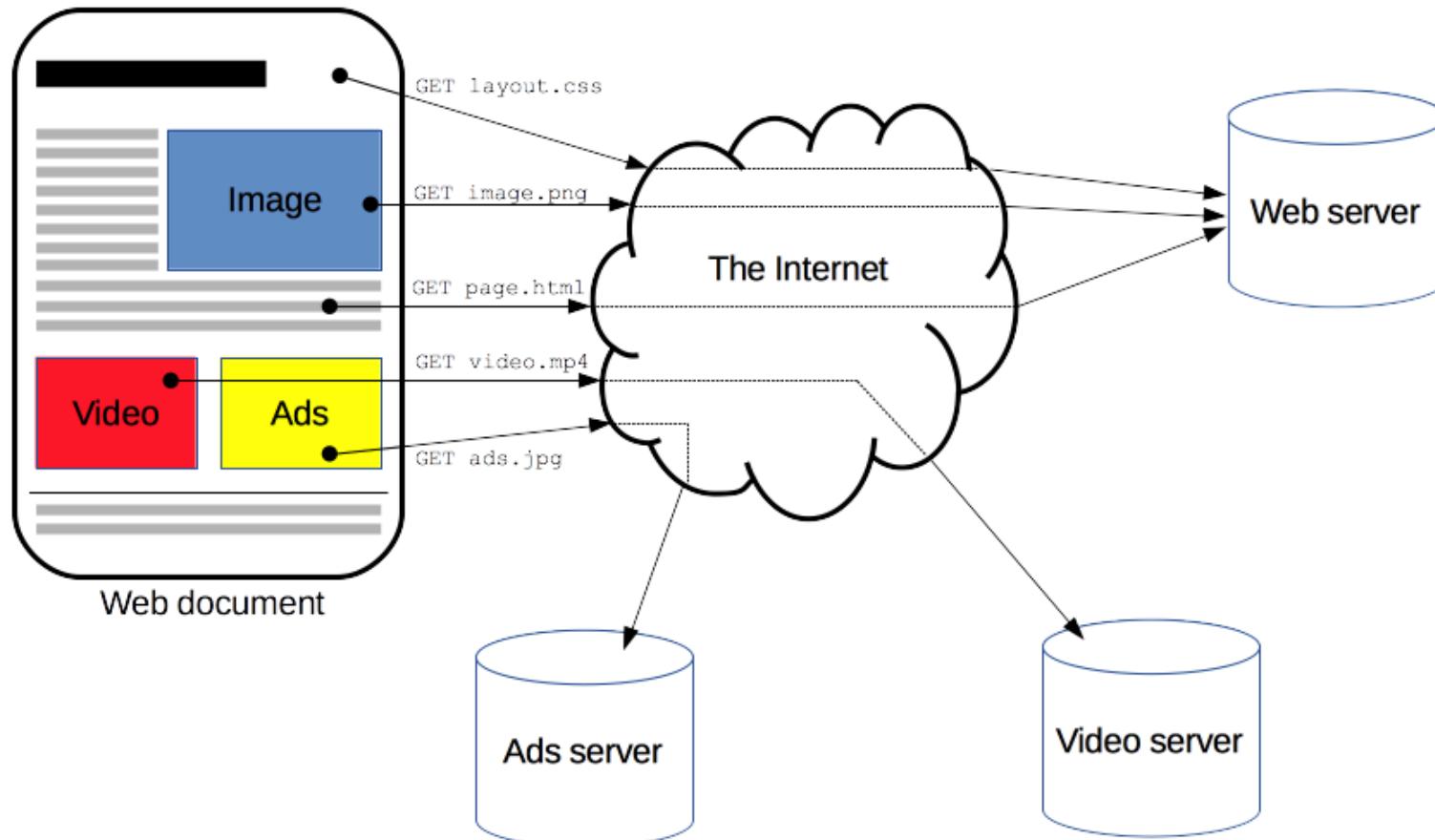
## Encadenamiento

Una necesidad común es el ejecutar dos o más operaciones asíncronas seguidas, donde cada operación posterior se inicia cuando la operación previa tiene éxito, con el resultado del paso previo. Logramos esto creando una cadena de objetos **promises**.

```
let p = new Promise(function(resolve, reject){  
    let t = setTimeout(function(){  
        resolve(1);  
        //reject("rechazado");  
    }, 3000);  
});
```

```
p.then(function(value) {  
    console.log("A. " + value);  
    return ++value;  
}).then(function(value) {  
    console.log("B. " + value);  
}).catch(function(value) {  
    console.log("algo salio mal " + value);  
});
```

## HTTP





# JavaScript



## Métodos HTTP

HTTP define un conjunto de métodos de petición para indicar la acción que se desea realizar para un recurso determinado. Aunque estos también pueden ser sustantivos, estos métodos de solicitud a veces son llamados HTTP verbs.

### GET

El método GET solicita una representación de un recurso específico. Las peticiones que usan el método GET sólo deben recuperar datos.

### POST

El método POST se utiliza para enviar una entidad a un recurso en específico, causando a menudo un cambio en el estado o efectos secundarios en el servidor.

### PUT

El modo PUT reemplaza todas las representaciones actuales del recurso de destino con la carga útil de la petición.



# JavaScript



## Métodos HTTP

### **DELETE**

El método DELETE borra un recurso en específico.

### **OPTIONS**

El método OPTIONS es utilizado para describir las opciones de comunicación para el recurso de destino.

Referencia:

<https://developer.mozilla.org/es/docs/Web/HTTP/Methods>



# JavaScript



## API fetch

El API fetch es un nuevo estándar que da una alternativa para interactuar por HTTP, con un diseño moderno, basado en promesas.

```
fetch('https://httpbin.org/ip')
  .then(function(response) {
    return response.text();
  })
  .then(function(data) {
    console.log('data = ', data);
  })
  .catch(function(err) {
    console.error(err);
  });
});
```

En los ejemplos utilizaremos los servicios de <https://httpbin.org/>



# JavaScript



## API fetch form data

### Opciones

```
fetch('https://httpbin.org/post', {
  method: 'POST',
  headers: {
    'Content-Type': 'application/x-www-form-
urlencoded'
  },
  body: 'a=1&b=2'
})
.then(function(response) {
  console.log('response =', response);
  return response.json();
})
```

```
.then(function(data) {
  console.log('data = ', data);
})
.catch(function(err) {
  console.error(err);
});
```

<https://developer.mozilla.org/en-US/docs/Web/API/WindowOrWorkerGlobalScope/fetch>



# JavaScript



## API fetch Opciones

- **method**: método a utilizar
- **headers**: cabeceras que se deben enviar (ver objeto **Headers**).
- **body**: cuerpo que se envía al servidor, que puede ser una cadena, un objeto **Blob**, **BufferSource**, **FormData** o **URLSearchParams**.
- **mode**: modo de la solicitud: ‘cors’, ‘no-cors’, ‘same-origin’, ‘navigate’.
- **credentials**: credenciales utilizadas: ‘omit’, ‘same-origin’, ‘include’.
- **cache**: forma de utilización de la caché: ‘default’, ‘no-store’, ‘reload’, ‘no-cache’, ‘force-cache’, ‘only-if-cached’.
- **redirect**: forma de gestionar la redirección: ‘follow’, ‘error’, ‘manual’.
- **referrer**: valor utilizado como referrer: ‘client’, ‘no-referrer’ una URL.
- **referrerPolicy**: especifica el valor de la cabecera referer: ‘no-referrer’, ‘no-referrer-when-downgrade’, ‘origin’, ‘origin-when-cross-origin’, ‘unsafe-url’.
- **integrity**: valor de integridad de la solicitud



# JavaScript



## API fetch JSON

```
fetch('https://httpbin.org/post',{
    method: 'POST',
    headers: {
        'Content-Type': 'application/json'
    },
    body: JSON.stringify({ "a": 1, "b": 2}),
    cache: 'no-cache'
})
.then(function(response) {
    return response.json();
})
.then(function(data) {
    console.log('data = ', data);
})
.catch(function(err) {
    console.error(err);
});
```

<https://developer.mozilla.org/en-US/docs/Web/API/WindowOrWorkerGlobalScope/fetch>



# JavaScript



## API fetch response

En la función `then` vamos a recibir un objeto `Response`. Este objeto contiene la respuesta que da el servidor.

```
fetch('https://httpbin.org/ip')
  .then(function(response) {
    console.log('response.body =', response.body);
    console.log('response.bodyUsed =', response.bodyUsed);
    console.log('response.headers =', response.headers);
    console.log('response.ok =', response.ok);
    console.log('response.status =', response.status);
    console.log('response.statusText =', response.statusText);
    console.log('response.type =', response.type);
    console.log('response.url =', response.url);
    return response.json();
  })
  .then(function(data) {
    console.log('data = ', data);
  })
  .catch(function(err) {
    console.error(err);
  });
}
```



# JavaScript



## API fetch response

El contenido del body no está disponible directamente en el objeto Response, tenemos que llamar uno de los métodos disponibles para que nos devuelva una promesa donde recibiremos el valor

- `response.text()` para que nos devuelva el contenido en formato texto
- `response.json()` para que lo devuelva como objeto Javascript
- `response.arrayBuffer()` para obtenerlo como ArrayBuffer
- `response.blob()` como valor que podemos manejar con `URL.createObjectURL()`
- `response.formData()` para obtenerlos como `FormData`



# JavaScript



## API fetch request

Una forma alternativa de configurar el comportamiento de **fetch()** es crear un objeto **Request**.

```
var request = new Request('https://httpbin.org/get', {
    method: 'GET',
    mode: 'cors',
    credentials: 'omit'
});
console.log('request =', request);
fetch(request)
.then(function(response) {
    console.log('response =', response);
    return response.text();
})
.then(function(data) {
    console.log('data = ', data);
})
.catch(function(err) {
    console.error(err);
});
```



# JavaScript



## API fetch FormData

```
var dataSend = new FormData();
dataSend.append('a', '1');
dataSend.append('b', '2');
var request = new Request('https://httpbin.org/put', {
    method: 'PUT',
    body: dataSend
});
console.log('request =', request);
for (var k of dataSend.keys()) {
    console.log('dataSend.get("' + k + '") =', dataSend.get(k));
}
fetch(request)
    .then(function(response) {
        console.log('response =', response);
        return response.text();
    })
    .then(function(data) {
        console.log('data = ', data);
    })
    .catch(function(err) {
        console.error(err);
    });
}
```



# JavaScript



## Práctica

Realizar ABC de artículos como se muestra en la siguiente imagen

### Artículos

#### Registro

Por favor llene el formulario para crear un artículo.

Titulo:	Ingrese el título
Autor:	Ingrese el autor
Tipo:	Seleccione una opción
Fecha publicación:	dd/mm/aaaa
Texto:	   

**Guardar** **Limpiar** **Cancelar**

**Noticia** **Aviso** **Agregar**

#### Comida Visita

Autor: Gerencia  
Tipo: Aviso

Fecha publicación: 2019-09-15

Se comunica a todo el personal que el próximo lunes 30 de abril del año en curso, se presentará en oficinas centrales de esta Empresa, el Presidente del Consorcio Mundial TAKAWE, por lo que en su honor se ofrecerá una comida en el Centro Libanes de la Cd. de México a partir de las 15:00 horas.La bienvenida estará a cargo de nuestro Director de la Planta Nogales, Sr. Oko Na-gazawa, por lo cual, les rogamos su puntual asistencia. Los boletos serán personales y podrán recogerlos en la Gerencia Administrativa, así como los viáticos correspondientes, a partir de hoy.

**Eliminar** **Editar**



# Node JS





# Node JS



## ¿Qué es NodeJS?

Es un entorno que trabaja en tiempo de ejecución, de código abierto, multiplataforma, que permite a los desarrolladores crear toda clase de herramientas de lado servidor y aplicaciones en JavaScript.

Como tal, el entorno omite las APIs de JavaScript específicas del explorador web y añade soporte para APIs de sistema operativo más tradicionales que incluyen HTTP y bibliotecas de sistemas de ficheros.



# Node JS



## Instalación

Para instalar nodejs lo bajamos de la página oficial y ejecutamos su instalable

The screenshot shows the Node.js website at [nodejs.org/es/](https://nodejs.org/es/). The page has a dark header with the Node.js logo and navigation links for INICIO, ACERCA, DESCARGAS, DOCUMENTACIÓN, PARTICIPA, SEGURIDAD, NOTICIAS, and FUNDACIÓN. The FUNDACIÓN link is highlighted with a green background. Below the header, a text block states: "Node.js® es un entorno de ejecución para JavaScript construido con el motor de JavaScript V8 de Chrome." Two download buttons are visible: "Descargar para Windows (x64)" with options for "10.16.3 LTS" (Recommended for most) and "12.10.0 Actual" (Latest features). At the bottom, there are links for "Otras Descargas | Cambios | Documentación del API" and "Otras Descargas | Cambios | Documentación del API". A note at the bottom says: "Ó revise la Agenda de LTS."



# Node JS



## Hola Mundo

Puedes crear de forma sencilla un servidor web básico para responder cualquier petición simplemente usando el paquete HTTP de Node, como se muestra abajo. Este, creará un servidor y escuchará cualquier clase de peticiones en la URL `http://127.0.0.1:8000/`; cuando se reciba una petición, se responderá enviando en texto la respuesta: "Hola Mundo!".

```
// Se carga el módulo de HTTP
var http = require("http");

// Creación del servidor HTTP, y se define la escucha
// de peticiones en el puerto 8000
http.createServer(function(request, response) {

    // Se define la cabecera HTTP, con el estado HTTP (OK: 200) y el tipo de contenido
    response.writeHead(200, {'Content-Type': 'text/plain'});

    // Se responde, en el cuerpo de la respuesta con el mensaje "Hello World"
    response.end('Hola Mundo!\n');

}).listen(8000);

// Se escribe la URL para el acceso al servidor
console.log('Servidor en la url http://127.0.0.1:8000/');
```



# Node JS



## Iniciar Servidor

```
Símbolo del sistema - node ejemplo1_HolaMundo.js
C:\proyectos\yoliti\curso-reactjs\4.REACTJS\ejemplo1_holaMundoNode>node ejemplo1_HolaMundo.js
Servidor en la url http://127.0.0.1:8000/
```



# Node JS



## Terminal REPL

Node.js viene con un entorno virtual llamado REPL (también conocido como Node shell). REPL significa Read-Eval-Print-Loop. Es una forma rápida y fácil de probar el código simple Node.js / JavaScript.

Para iniciar REPL (Node shell), abra el símbolo del sistema (en Windows) o terminal (en Mac o UNIX / Linux) y escriba **node**

```
C:\>node  
>
```



# Node JS



## Terminal REPL

Ahora puede probar prácticamente cualquier expresión Node.js / JavaScript en REPL. Por ejemplo, si escribe "10 + 20", mostrará el resultado 30 inmediatamente en una nueva línea.

```
C:\>node
> 10 + 20
30
>
```



# Node JS



## Terminal REPL

Puede definir variables y Si necesita escribir una expresión o función de JavaScript de varias líneas, simplemente presione **Enter** cuando desee escribir algo en la siguiente línea como continuación de su código.

El terminal REPL mostrará tres puntos (...), lo que significa que puede continuar en la línea siguiente. Escriba .break para salir del modo de continuidad.

```
C:\>node
> function hola() {
...   console.log('hola mundo');
...
undefined
> hola();
hola mundo
undefined
>
```



# Node JS



## Terminal REPL

Comando REPL	Descripción
.help	Mostrar ayuda sobre todos los comandos
teclas de tabulación	Mostrar la lista de todos los comandos.
Teclas arriba / abajo	Ver los comandos anteriores aplicados en REPL.
.save nombrearchivo	Guarde la sesión actual de Node REPL en un archivo.
.load nombrearchivo	Cargue el archivo especificado en la sesión actual de Node REPL.
Ctrl + C	Terminar el comando actual.
Ctrl + C (dos veces)	Salga de REPL.
Ctrl + D	Salga de REPL.
.break	Salir de la expresión multilínea.
.clear	Alias .break



# Node JS



## NPM

Node Package Manager o simplemente npm es un gestor de paquetes, gracias a él podremos tener cualquier librería disponible con solo una línea de código, npm nos ayudará a administrar nuestros módulos, distribuir paquetes y agregar dependencias de una manera sencilla.

[www.npmjs.com](http://www.npmjs.com) aloja miles de paquetes gratuitos para descargar y usar.

Un paquete en Node.js contiene todos los archivos que necesita para un módulo. Los módulos son bibliotecas de JavaScript que puede incluir en su proyecto.



# Node JS



## NPM Paquetes locales vs globales

- ✓ Los **paquetes locales** se instalan en el directorio donde se ejecuta **npm install <package-name>** y se colocan dentro del directorio **node\_modules**
- ✓ Los paquetes globales se colocan en un solo lugar en su sistema (depende de su configuración del paquete), independientemente de dónde se ejecute **npm install -g <package-name>**



# Node JS

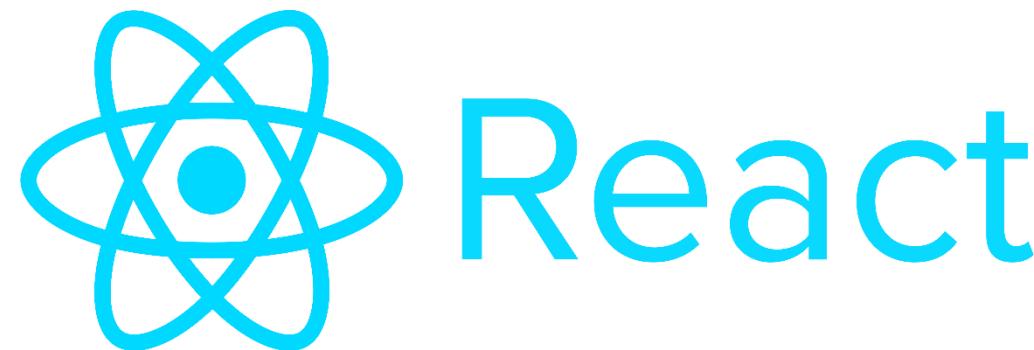


## Ejemplo instalación paquete

```
C:\proyectos\yoliti\curso-reactjs>mkdir test  
C:\proyectos\yoliti\curso-reactjs>cd test  
C:\proyectos\yoliti\curso-reactjs\test>npm install upper-case
```



# React JS





# React JS



## ¿Qué es React?

React es una librería de JavaScript declarativa, eficiente y flexible para construir interfaces de usuario. Permite componer IUs complejas de pequeñas y aisladas piezas de código llamadas “componentes”.

Dentro del patrón MVC React representa la V (Vista)

Su principal desarrollador es





# React JS



## Puntos fuertes

- ✓ Gran Ecosistema
- ✓ Estabilidad y Alta Retrocompatibilidad
- ✓ Performance



# React JS



## Herramientas

### React Developer Tools for Chrome

<https://chrome.google.com/webstore/detail/react-developer-tools/fmkadmapgofadoplbjbjfkapdkoienihi?hl=es>

The screenshot shows the React Developer Tools extension for Google Chrome. At the top, there are two toggle switches: 'Permitir acceso a URL de archivo' (Allow access to file URL) and 'Recopilar errores' (Collect errors), both of which are turned on (blue). Below this is a toolbar with icons for Elements, Console, Sources, Network, Application, Performance, Memory, Audits, Security, Components (which is selected and highlighted in blue), and Profiler. A search bar at the top left contains the text 'Search (text or /regex/)'. The main area displays component information for a component named 'Football'. It shows the 'props' section with a prop named 'prop1' set to 'test', and the 'context' section with a value of 'Object (empty)'. There are also small icons for copy, paste, and refresh.



# React JS



## Hola Mundo

<https://es.reactjs.org/redirect-to-codepen/hello-world>

```
ReactDOM.render(  
  <h1>Hello, world!</h1>,  
  document.getElementById('root')  
);
```



# React JS



## JSX

Es una extensión de la sintaxis de JavaScript. Recomendamos usarlo con React para describir cómo debería ser la interfaz de usuario. JSX puede recordarte a un lenguaje de plantillas, pero viene con todo el poder de JavaScript.

- ✓ Escribir nuestros propios componentes con capitalización. `<App />`
- ✓ Escribir los componentes built-in (HTML) con minúsculas. `<img />`

```
const element = <h1>Hello, world!</h1>;
```



# React JS



## Expresiones en JSX

En JSX puedes poner cualquier expresión de JavaScript dentro de llaves {}. Por ejemplo 2 + 2, user.firstName, o formatName(user)

```
const name = 'Josh Perez';
const element = <h1>Hello, {name}</h1>

ReactDOM.render(
  element,
  document.getElementById('root')
);
```

```
function formatName(user) {
  return user.firstName + ' ' + user.lastName;
}

const user = {
  firstName: 'Harper',
  lastName: 'Perez'
};

const element = (
  <h1>
    Hello, {formatName(user)}!
  </h1>
);

ReactDOM.render(
  element,
  document.getElementById('root')
);
```

<https://es.reactjs.org/redirect-to-codepen/introducing-jsx>



# React JS



## JSX también es una expresión

Esto significa que puedes usar JSX dentro de declaraciones **if** y bucles **for**, asignarlo a variables, aceptarlo como argumento, y retornarlo desde dentro de funciones:

```
function getGreeting(user) {
  if (user) {
    return <h1>Hello, {formatName(user)}!</h1>;
  }
  return <h1>Hello, Stranger.</h1>;
}
```



# React JS



## JSX atributos

Puedes utilizar comillas para especificar strings literales como atributos:

```
const element = <div tabIndex="0"></div>;
```

puedes usar llaves para insertar una expresión JavaScript en un atributo:

```
const element = <img src={user.avatarUrl}></img>;
```

### Advertencias:

- ✓ Debes utilizar comillas (para los valores de los strings) o llaves (para las expresiones), pero no ambas en el mismo atributo.
- ✓ Dado que JSX es más cercano a JavaScript que a HTML, React DOM usa la convención de nomenclatura camelCase en vez de nombres de atributos HTML.

Por ejemplo, class se vuelve className en JSX, y tabindex se vuelve tabIndex.



# React JS



## JSX especificando hijos

Si una etiqueta está vacía, puedes cerrarla inmediatamente con />, como en XML:

```
const element = <img src={user.avatarUrl} />;
```

Las etiquetas de Javascript pueden contener hijos, el código html debe estar envuelto en un solo elemento de nivel superior:

```
const element = (
  <div>
    <h1>Hello!</h1>
    <h2>Good to see you here.</h2>
  </div>
);
```



# React JS



## JSX representa objetos

Babel compila JSX a llamadas de `React.createElement()`  
Estos dos ejemplos son idénticos:

```
const element = (
  <h1 className="greeting">
    Hello, world!
  </h1>
);
```

```
const element = React.createElement(
  'h1',
  {className: 'greeting'},
  'Hello, world!'
);
```



# React JS



## Renderizando elementos

A diferencia de los elementos del DOM de los navegadores, los elementos de React son objetos planos, y su creación es de bajo costo. React DOM se encarga de actualizar el DOM para igualar los elementos de React.

```
<div id="root"></div>
```

```
const element = <h1>Hello, world</h1>;
ReactDOM.render(element, document.getElementById('root'));
```



# React JS



## Actualizando el elemento renderizado

Los elementos de React son inmutables. Una vez creas un elemento, no puedes cambiar sus hijos o atributos. Un elemento es como un fotograma solitario en una película: este representa la interfaz de usuario en cierto punto en el tiempo.

Con nuestro conocimiento hasta este punto, la única manera de actualizar la interfaz de usuario es creando un nuevo elemento, y pasarlo a ReactDOM.render().



# React JS



## React solo actualiza lo que es necesario

React DOM compara el elemento y su hijos con el elemento anterior, y solo aplica las actualizaciones del DOM que son necesarias para que el DOM esté en el estado deseado.

Ejemplo:

```
function tick(){
  const element = (
    <div>
      <h1>Hola, mundo!</h1>
      <h2>Son las {new Date().toLocaleTimeString()}</h2>
    </div>
  );
  ReactDOM.render(element, document.getElementById('root'))
}

setInterval(tick, 1000);
```



# React JS



## Practica

Realizar el siguiente componente en reactjs

### Etiquetas por categoria

La siguiente tabla muestra algunas etiquetas html por categorias

20:41:48

Categoría	Etiquetas
Formulario	form
	button
	input
	select
	textarea
Tabla	table
	tr
	th
	td
	caption
Texto	b
	em
	i
	strong
	sub
	sup



# React JS



## Componentes

Un componente en React simplemente es una pieza de UI (user interface), que tiene una funcionalidad independiente definida.

Un componente se puede definir con una función de JavaScript o una clase ES6

```
function Welcome(props) {  
  return <h1>Hello, {props.name}</h1>;  
}
```

```
class Welcome extends React.Component {  
  render() {  
    return <h1>Hello, {this.props.name}</h1>;  
  }  
}
```



# React JS



## Componentes

Al crear un componente React, el nombre del componente debe comenzar con una letra mayúscula.

El componente creado como **clase** tiene que incluir la declaración **extends React.Component**, esta declaración crea una herencia para **React.Component** y le da acceso al componente a las funciones de **React.Component**, también requiere un método **render()**, este método devuelve HTML.

```
class Carro extends React.Component {
  render() {
    return <h2>Hola, Soy un Carro!</h2>;
  }
}
ReactDOM.render(<Carro />, document.getElementById('root'));
```



# React JS



## Composición de componentes

Los componentes pueden referirse a otros componentes en su salida. Un botón, un cuadro de diálogo, un formulario, una pantalla: en aplicaciones de React, todos son expresados comúnmente como componentes.

```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}

function App() {
  return (
    <div>
      <Welcome name="Sara" />
      <Welcome name="Cahal" />
      <Welcome name="Edite" />
    </div>
  );
}

ReactDOM.render(
  <App />,
  document.getElementById('root')
);
```



# React JS



## Composición de componentes

### Ejemplo

```
class Car extends React.Component {
  render() {
    return <h2>I am a Car!</h2>;
  }
}

class Garage extends React.Component {
  render() {
    return (
      <div>
        <h1>Who lives in my Garage?</h1>
        <Car />
      </div>
    );
  }
}

ReactDOM.render(<Garage />, document.getElementById('root'));
```



# React JS



## Entorno de desarrollo local

**Create React App** configura tu ambiente de desarrollo de forma que puedas usar las últimas características de Javascript, es un ambiente cómodo para **aprender React**. Para ello internamente usa [Babel](#) y [webpack](#)

```
npx create-react-app my-app  
cd my-app  
npm start
```

**npx.**- Es una herramienta de cli que nos permite ejecutar paquetes de npm de forma mucho más sencilla.

**npm start .** - Nos permite iniciar el servidor de node



# React JS



## Cadena de herramientas

Una cadena de herramientas para construir Javascript generalmente consiste de:

- ✓ Un **gestor de paquetes** como [Yarn](#) o [npm](#). Este te permite aprovechar el vasto ecosistema de paquetes de terceros, e instalarlos o actualizarlos de una manera fácil.
- ✓ Un **empaquetador** como [webpack](#) o [Parcel](#). Este te permite escribir código modular y empaquetarlo junto en paquetes más pequeños que optimizan el tiempo de carga.
- ✓ Un **compilador** como [Babel](#). Este te permite escribir Javascript moderno que aún así funciona en navegadores más antiguos.

Referencia: <https://blog.usejournal.com/creating-a-react-app-from-scratch-f3c693b84658>



# React JS



## Componentes en archivos

React se trata de reutilizar código, por lo que puedes tener tus componentes en archivos separados.

Ten en cuenta que el archivo debe comenzar con **import React**, y debe terminar con la declaración **export default NombreClase**:

```
4.REACTJS > ejemplo7-componente-archivos > src > components > car > JS Car.js > .
 1 import React from 'react'
 2
 3 class Car extends React.Component {
 4   render() {
 5     return <h2>I am a Car!</h2>;
 6   }
 7 }
 8 export default Car;
```



# React JS



## Props

Las props son la manera que un componente superior manda información a componentes inferiores (el flujo de datos es unidireccional) y son de solo lectura.

Las props de reactjs son como argumentos de función en JavaScript y atributos en HTML.  
Para enviar props a un componente, use la misma sintaxis que los atributos HTML

```
class Car extends React.Component {
  render() {
    return <div>
      <h2>I am a {this.props.color} {this.props.brand}!</h2>
      <p>Model: {this.props.moreInfo.model}, Year: {this.props.moreInfo.year}</p>
    </div>
  }
}

class Garage extends React.Component {
  render() {
    let color = 'Blue';
    let moreInfo = {model: 'Mustang', year: 1967};
    return (
      <div>
        <h1>Who lives in my garage?</h1>
        <Car brand="Ford" color={color} moreInfo={moreInfo}/>
      </div>
    );
  }
}

//ReactDOM.render(<Car brand="Ford" />, document.getElementById('root'));
ReactDOM.render(<Garage />, document.getElementById('root'));
```



# React JS



## Props

Ya sea que declares un componente como una función o como una clase, este nunca debe modificar sus props.

```
class Car extends React.Component {
  render() {
    this.props.color = 'green';
    return <div>
      <h2>I am a & {this.props.color} & {this.props.brand}!</h2>
      <p>Model: & {this.props.moreInfo.model}, Year: & {this.props.moreInfo.year}</p>
    </div>
  }
}
```



# React JS



## Props en el constructor

Si el componente tiene un constructor, las props siempre deben pasarse al constructor y también al React.Component a través del método super().

```
class Car extends React.Component {
  constructor(props) {
    super(props);
  }
  render() {
    //this.props.color = 'green';
    return <div>
      <h2>I am a &lt;{this.props.color}&gt; {this.props.brand}!</h2>
      <p>Model: &lt;{this.props.moreInfo.model}&gt;, Year: &lt;{this.props.moreInfo.year}&gt;</p>
    </div>
  }
}
```



# React JS



## State

El **state** es el estado de un componente, es similar a las props, pero es privado y está completamente controlado por el componente.

- ✓ Se inicializa en el constructor.
- ✓ Puede contener tantas propiedades como desee.
- ✓ Puede consultar el objeto **state** en cualquier parte del componente utilizando la sintaxis: **this.state.propertyname**.
- ✓ Para cambiar un valor en el objeto **state**, use el método **this.setState()**
- ✓ Cuando se cambia un valor en el objeto **state**, el componente se volverá a pintar, lo que significa que la salida cambiará de acuerdo con los nuevos valores.



# React JS



## State

Ejemplo:

```
class Car extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      brand: "Ford",
      model: "Mustang",
      color: props.color,
      year: 1964
    };
  }
  changeColor = () => {
    this.setState({color: "blue"});
  }
  render() {
    return <div>
      <h1>My &{this.state.brand}</h1>
      <p>
        It is a &{this.state.color} &{this.state.model} from &{this.state.year}.
      </p>
      <button
        type="button"
        onClick={this.changeColor}>
        Change color
      </button>
    </div>;
  }
}
ReactDOM.render(<Car color="red"/>, document.getElementById('root'));
```



# React JS



## Ciclo de vida

Cada componente en React tiene un ciclo de vida que puede monitorear y manipular durante sus tres fases principales

- ✓ Montaje
- ✓ Actualización
- ✓ Desmontaje .



# React JS



## Ciclo de vida (Montaje)

Al montar un componente, **React** llama a los siguientes 4 métodos, en este orden:

1. constructor()
2. getDerivedStateFromProps()
3. render()
4. componentDidMount()



## Ciclo de vida (Montaje)

### ✓ **constructor**

El constructor() se llama antes que nada, cuando se inicia el componente, y es el lugar natural para configurar el **state** inicial y otros valores iniciales.

El método constructor() se llama con los **props**, como argumentos, y siempre debe comenzar llamando al **super(props)** antes que nada, esto iniciará el método constructor del padre y permitirá que el componente herede los métodos de su padre ( **React.Component** ).

### ✓ **getDerivedStateFromProps**

El método **getDerivedStateFromProps()** se llama justo antes de representar los elementos en el DOM, se llama cada vez que el componente padre se vuelve a renderizar, sin importar si las props son «diferentes» a las anteriores.

Este es el lugar natural para establecer el objeto de state basado en los props iniciales. Toma el **state** como argumento y devuelve un objeto con cambios en el **state**.



# React JS



## Ciclo de vida (Montaje)

### ✓ **render**

El método **render()** es requerido y es el método que genera HTML en el DOM.

### ✓ **componentDidMount**

Se llama al método **componentDidMount()** después de representar el componente.

Aquí es donde ejecuta las declaraciones que requieren que el componente ya esté ubicado en el DOM.



# React JS



## Ciclo de vida (Montaje)

```
class Header extends React.Component {
  constructor(props) {
    super(props);
    this.state = {favoritecolor: "red"};
    console.log('constructor');
  }
  static getDerivedStateFromProps(props, state) {
    console.log('getDerivedStateFromProps');
    return {favoritecolor: props.favcol};
  };
  componentDidMount() {
    console.log('componentDidMount');
    setTimeout(() => {
      this.setState({favoritecolor: "green"})
    }, 1000);
  };
  render() {
    console.log('render');
    return (
      <h1>My Favorite Color is {this.state.favoritecolor}</h1>
    );
  }
}

ReactDOM.render(<Header favcol="yellow"/>, document.getElementById('root'));
```



# React JS



## Ciclo de vida (Actualización)

Al actualizar un componente, **React** llama a los siguientes 5 métodos, en este orden:

1. `getDerivedStateFromProps()`
2. `shouldComponentUpdate()`
3. `render()`
4. `getSnapshotBeforeUpdate()`
5. `componentDidUpdate()`



# React JS



## Ciclo de vida (Actualización)

### ✓ **getDerivedStateFromProps**

También en la actualización se llama a este método y es el primer que se llama cuando se actualiza un componente.

### ✓ **shouldComponentUpdate**

En el método **shouldComponentUpdate()** puede devolver un valor booleano que especifica si React debe continuar con el render o no.

### ✓ **render**

El método **render()** , por supuesto, se llama cuando se actualiza un componente, tiene que volver a pintar el HTML en el DOM, con los nuevos cambios.



# React JS



## Ciclo de vida (Actualización)

### ✓ **getSnapshotBeforeUpdate**

En el método **getSnapshotBeforeUpdate()** tiene acceso a los props y el state antes de la actualización, lo que significa que incluso después de la actualización, puede verificar cuáles eran los valores antes de la actualización.

Si el método **getSnapshotBeforeUpdate()** está presente, también debe incluir el método **componentDidUpdate()** , de lo contrario obtendrá un error.

### ✓ **componentDidUpdate**

El método **componentDidUpdate** se llama después de que el componente se actualiza en el DOM.



# React JS



## Ciclo de vida (Actualización)

```
class Header extends React.Component {
  constructor(props) {
    super(props);
    this.state = {favoritecolor: "red"};
  }
  static getDerivedStateFromProps(props, state) {
    console.log('getDerivedStateFromProps');
    return {};
  }
  componentDidMount() {
    setTimeout(() => {
      this.setState({favoritecolor: "yellow"})
    }, 1000)
  }
  shouldComponentUpdate() {
    console.log('shouldComponentUpdate');
    return true; //false;
  }
  getSnapshotBeforeUpdate(prevProps, prevState) {
    console.log('getSnapshotBeforeUpdate');
    document.getElementById("div1").innerHTML =
      "Before the update, the favorite was " + prevState.favoritecolor;
    return 100;
  }
  componentDidUpdate(prevProps, prevState, number) {
    console.log('componentDidUpdate ' + number);
    document.getElementById("div2").innerHTML =
      "The updated favorite is " + this.state.favoritecolor;
  }
  render() {
    console.log('render');
    return (
      <div>
        <h1>My Favorite Color is {this.state.favoritecolor}</h1>
        <div id="div1"></div>
        <div id="div2"></div>
      </div>
    );
  }
}

ReactDOM.render(<Header />, document.getElementById('root'));
```



# React JS



## Ciclo de vida (Desmontaje)

La siguiente fase del ciclo de vida es cuando un componente se elimina del DOM, **React** llama al siguiente método:

1. `componentWillUnmount()`



# React JS



## Ciclo de vida (Desmontaje)

### ✓ **componentWillUnmount**

Se llama al método `componentWillUnmount` cuando el componente se elimina del DOM.

```
class Container extends React.Component {
  constructor(props) {
    super(props);
    this.state = {show: true};
  }
  delHeader = () => {
    this.setState({show: false});
  }
  render() {
    let myheader;
    if (this.state.show) {
      myheader = <Child />;
    }
    return (
      <div>
        {myheader}
        <button type="button" onClick={this.delHeader}>Delete Header</button>
      </div>
    );
  }
}

class Child extends React.Component {
  componentWillMount() {
    alert("The component named Header is about to be unmounted.");
  }
  render() {
    return (
      <h1>Hello World!</h1>
    );
  }
}

ReactDOM.render(<Container />, document.getElementById('root'));
```



# React JS



## Eventos

Los eventos de React se nombran usando camelCase, en vez de minúsculas.

Con JSX pasas una función como el manejador del evento dentro de llaves, en vez de un string. Una buena práctica es colocar el controlador de eventos como método en la clase de componente.

```
class Football extends React.Component {
  shoot() {
    alert("Great Shot!");
  }
  render() {
    return (
      <button onClick={this.shoot}>Take the shot!</button>
    );
  }
}

ReactDOM.render(<Football />, document.getElementById('root'));
```



# React JS



## Eventos **this**

Para los métodos en React, la palabra clave **this** debería representar el componente que posee el método.

Es por eso que debes usar las funciones de flecha. Con las funciones de flecha, **this** siempre representará el objeto que definió la función de flecha.

```
class Football extends React.Component {
  constructor(props) {
    super(props)
    this.shoot2 = this.shoot2.bind(this)
  }
  shoot() {
    console.log(this); // undefined
  }
  shoot1 = () => {
    console.log(this.props.prop1);
    /*
    The 'this' keyword refers to the component object
    */
  }
  shoot2() {
    console.log(this.props.prop1);
  }
  render() {
    return (
      <button onClick={this.shoot}>Take the shot!</button>
    );
  }
}

ReactDOM.render(<Football prop1="test"/>, document.getElementById('root'));
```



# React JS



## Eventos pasando parámetros

Si desea enviar parámetros a un controlador de eventos, tiene dos opciones:

1. Hacer una **función de flecha** anónima
2. Hacer **bind** del controlador de eventos a **this**



# React JS



## Eventos pasando parámetros

Ejemplo

```
class Football extends React.Component {
  constructor(props) {
    super(props)
  }
  shoot = (value) => {
    console.log(value);
  }
  shoot2() {
    console.log(value);
  }
  render() {
    return (
      <div>
        <button onClick={() => this.shoot("Goal")}>Take the shot!</button>
        <button onClick={this.shoot.bind(this, "Goal")}>Take the shot 2!</button>
      </div>
    );
  }
}

ReactDOM.render(<Football prop1="test"/>, document.getElementById('root'));
```



# React JS



## React Event Object

Para tener acceso al objeto React Event que activó la función, se tienen 2 maneras

```
class Football extends React.Component {
  constructor(props) {
    super(props)
  }
  shoot = (value, event) => {
    console.log(value);
    console.log('type: ' + event.type, event);
  }
  shoot2(value, event) {
    console.log(value);
    console.log('type: ' + event.type, event);
  }
  render() {
    return (
      <div>
        <button onClick={() => this.shoot("Goal", ev)}>Take the shot!</button>
        <button onClick={this.shoot2.bind(this, "Goal")}>Take the shot 2!</button>
      </div>
    );
  }
}

ReactDOM.render(<Football prop1="test"/>, document.getElementById('root'));
```



# React JS

Practica





# React JS



## Formularios

En HTML, los elementos de formularios como los `<input>`, `<textarea>` y el `<select>` normalmente mantienen sus propios estados y los actualizan de acuerdo a la interacción del usuario. En React, el estado mutable es mantenido normalmente en la propiedad `state` de los componentes, y solo se actualiza con `setState()`.

En React un formulario se agrega como cualquier otro elemento:

```
class MyForm extends React.Component {
  render() {
    return (
      <form>
        <h1>Hello</h1>
        <div>
          <label for="name">Enter your name:</label>
          <input id="name"
                 type="text"
            />
        </div>
      </form>
    );
  }
}
ReactDOM.render(<MyForm />, document.getElementById('root'));
```



# React JS



## Formularios

El manejo de formularios se trata de cómo maneja los datos cuando cambia de valor o se envía. En HTML, el DOM generalmente maneja los datos del formulario, en React, los datos del formulario generalmente son manejados por los componentes.

Cuando los datos son manejados por los componentes, todos los datos se almacenan en el **state** componente, puede controlar la acción de envío agregando un controlador de eventos en el atributo onSubmit:

```
class MyForm extends React.Component {
  constructor(props) {
    super(props);
    this.state = { username: '' };
  }
  mySubmitHandler = (event) => {
    event.preventDefault();
    alert("You are submitting " + this.state.username);
  }
  myChangeHandler = (event) => {
    this.setState({username: event.target.value});
  }
  render() {
    return (
      <form onSubmit={this.mySubmitHandler}>
        <h1>Hello {this.state.username}</h1>
        <p>Enter your name, and submit:</p>
        <input
          type='text'
          onChange={this.myChangeHandler}
          defaultValue={this.props.username}>
        />
        <input
          type='submit'
        />
      </form>
    );
  }
}

ReactDOM.render(<MyForm username="test"/>, document.getElementById('root'));
```



# React JS



## Formularios Múltiples Campos

Puede controlar los valores de más de un campo de entrada agregando un atributo de **name** a cada elemento, cuando inicializa el estado en el constructor, use los nombres de campo.

Para acceder a los campos en el controlador de eventos, use la sintaxis **event.target.name** y **event.target.value** y para actualizar el estado en el método **this.setState** , use corchetes [notación de corchetes] alrededor del nombre de la propiedad.

```
class MyForm extends React.Component {
  constructor(props) {
    super(props);
    this.state = { username: '' };
  }
  mySubmitHandler = (event) => {
    event.preventDefault();
    alert("You are submitting " + this.state.username);
  }
  myChangeHandler = (event) => {
    this.setState({username: event.target.value});
  }
  render() {
    return (
      <form onSubmit={this.mySubmitHandler}>
        <h1>Hello, {this.state.username}</h1>
        <p>Enter your name, and submit:</p>
        <input
          type='text'
          onChange={this.myChangeHandler}
          defaultValue={this.props.username}>
        </input>
        <input
          type='submit'
        />
      </form>
    );
  }
}

ReactDOM.render(<MyForm username="test"/>, document.getElementById('root'));
```



# React JS



## Formularios Validación

Puede validar la entrada del formulario cuando el usuario está escribiendo o puede esperar hasta que se envíe el formulario.

```
myChangeHandler = (event) => {
  let nam = event.target.name;
  let val = event.target.value;
  this.data[nam].value = val;
  this.data[nam].touch = true;
  if(nam === 'age' && !Number(val)) {
    this.data.age.valid = false;
    this.data.age.message = "Your age must be a number";
  } else {
    this.data.age.valid = true;
    this.data.age.message = "";
  }
  this.setState({ageMessage: this.data.age.message});
  this.setState({[nam]: val});
}

render() {
  const mystyle = {
    color: "red"
  };
  const hidestyle = {
    display: "none"
  };
  return (
    <form onSubmit={this.mySubmitHandler}>
      <h1>Hello {this.state.username} {this.state.age}</h1>
      <div>
        <label htmlFor="username">Enter your name:</label>
        <input
          id="username"
          type="text"
          name='username'
          onChange={this.myChangeHandler}
        />
        <p style={(this.state.usernameMessage !== '') ? mystyle : hidestyle}>{this.state.usernameMessage}</p>
      </div>
      <div>
        <label htmlFor="age">Enter your age:</label>
        <input id="age" type="text" name='age' onChange={this.myChangeHandler} />
        <p style={(this.state.ageMessage !== '') ? mystyle : hidestyle}>{this.state.ageMessage}</p>
      </div>
      <br/>
      <input type='submit' />
    </form>
  );
}
```



# React JS



## Listas y Keys

Puedes hacer colecciones de elementos e incluirlos en JSX usando llaves {}.

Debajo, recorreremos el array numbers usando la función map() de Javascript. Devolvemos un elemento <li> por cada ítem . Finalmente, asignamos el array de elementos resultante a listItems:

```
const numbers = [1, 2, 3, 4, 5];
const listItems = numbers.map((number) =>
  <li>{number}</li>
);
```



# React JS



## Listas y Keys

Las keys ayudan a React a identificar que ítems han cambiado, son agregados, o son eliminados. Las keys deben ser dadas a los elementos dentro del array para darle a los elementos una identidad estable:

```
const numbers = [1, 2, 3, 4, 5];
const listItems = numbers.map((number) =>
  <li key={number.toString()}>
    {number}
  </li>
);
```



# React JS



## Composición

A veces pensamos en componentes como “casos concretos” de otros componentes. Por ejemplo, podríamos decir que un WelcomeDialog es un caso concreto de Dialog.

En React, esto también se consigue por composición, en la que un componente más “específico” renderiza uno más “genérico” y lo configura con props:

```
function Dialog(props) {
  return (
    <FancyBorder color="blue">
      <h1 className="Dialog-title">
        {props.title}
      </h1>
      <p className="Dialog-message">
        {props.message}
      </p>
    </FancyBorder>
  );
}

function WelcomeDialog() {
  return (
    <Dialog
      title="Welcome"
      message="Thank you for visiting our spacecraft!" />
  );
}
```



# React JS



## Herencia

En Facebook usamos React en miles de componentes, y no hemos hallado ningún caso de uso en el que recomendariamos crear jerarquías de herencia de componentes.



# React JS



## React sin JSX

JSX no es un requisito para usar React. Usar React sin JSX es especialmente conveniente cuando no quieras configurar herramientas de compilación en tu entorno de desarrollo.

Cada elemento JSX es solamente una mejora sintáctica para llamar a

**React.createElement(component, props, ...children).**

Por lo tanto, cualquier cosa que se pueda hacer con JSX se puede hacer con Javascript puro.



# React JS

## Ejemplo sin JSX

### Con JSX

```
class Hello extends React.Component {
  render() {
    return <div>Hello {this.props.toWhat}</div>;
  }
}

ReactDOM.render(
  <Hello toWhat="World" />,
  document.getElementById('root')
);
```

### Sin JSX

```
class Hello extends React.Component {
  render() {
    return React.createElement('div', null, `Hello ${this.props.toWhat}`);
  }
}

ReactDOM.render(
  React.createElement(Hello, {toWhat: 'World'}, null),
  document.getElementById('root')
);
```



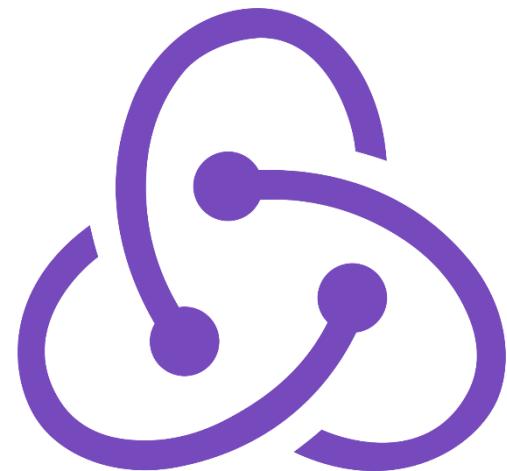
# React JS

Practica





# Redux.js





# Redux.js



## Introducción

Redux es una biblioteca JavaScript de código abierto para administrar el estado de la aplicación . Se usa más comúnmente con bibliotecas como React o Angular para construir interfaces de usuario.



# Redux.js



## Instalación

Para instalar la versión estable:

```
npm i -S redux
```

Para usar la conexión a React y las herramientas de desarrollo:

```
npm i -S react-redux
```

```
npm i -D redux-devtools
```



# Redux.js



## Acciones

Las acciones son un bloque de información que envia datos desde tu aplicación a tu store. Son la única fuente de información para el store. Las envias al store usando **store.dispatch()**.

Las acciones son objetos planos de JavaScript. Una acción debe tener una propiedad **type** que indica el tipo de acción a realizar. Los tipos normalmente son definidos como strings constantes. Una vez que tu aplicación sea suficientemente grande, quizas quieras moverlos a un módulo separado.

```
{  
  type: ADD_TODO,  
  text: 'Build my first Redux app'  
}
```



# Redux.js



## Reducers

Las acciones describen que algo pasó, pero no especifican cómo cambió el estado de la aplicación en respuesta. Esto es trabajo de los reducers.

El reducer es una función pura que toma el estado anterior y una acción, y devuelve un nuevo estado. Sólo evalúa el siguiente estado. Debe ser completamente predecible: invocarla con las mismas entradas muchas veces debe producir las mismas salidas. No debe realizar ningún efecto alterno como las llamadas al API o las transiciones del router. Esto debe suceder antes de que se envíe la acción.

```
(previousState, action) => newState
```



# Redux.js



## Reducers

Se llama reducer porque es el tipo de función que pasarías a `Array.prototype.reduce(reducer, ?initialValue)`. Es muy importante que los reducer se mantengan puros. Cosas que nunca deberías hacer dentro de un reducer:

- ✓ Modificar sus argumentos.
- ✓ Realizar tareas con efectos secundarios como llamas a un API o transiciones de rutas.
- ✓ Llamar una función no pura, por ejemplo `Date.now()` o `Math.random()`.



# Redux.js



## Store

Definimos las acciones que representan los hechos sobre "**lo que pasó**" y los reductores son los que actualizan el estado de acuerdo a esas acciones.

El **Store** es el objeto que los reúne. El **store** tiene las siguientes responsabilidades:

- ✓ Contiene el estado de la aplicación;
- ✓ Permite el acceso al estado via `getState()`;
- ✓ Permite que el estado sea actualizado via `dispatch(action)`;
- ✓ Registra los listeners via `subscribe(listener)`;
- ✓ Maneja la anulación del registro de los listeners via el retorno de la función de `subscribe(listener)`.



# Redux.js



## Flujo de datos

La arquitectura Redux gira en torno a un flujo de datos estrictamente unidireccional.

Esto significa que todos los datos de una aplicación siguen el mismo patrón de ciclo de duración, haciendo que la lógica de tu aplicación sea más predecible y más fácil de entender. También fomenta la normalización de los datos, de modo que no termines con múltiples copias independientes de la misma data sin que se entere una de la otra.



# Redux.js



## Flujo de datos

El ciclo de duración de la data en cualquier aplicación Redux sigue estos 4 pasos:

1. **Haces una llamada a `store.dispatch(action)`.** Una acción es un simple objeto describiendo que pasó. Por ejemplo:

```
{ type: 'LIKE_ARTICLE', articleId: 42 }  
{ type: 'FETCH_USER_SUCCESS', response: { id: 3, name: 'María' } }  
{ type: 'ADD_TODO', text: 'Leer la documentación de Redux.' }
```

Piense en una acción como un fragmento muy breve de noticias. "A María le gustó el artículo 42." o "'Leer la documentación de Redux.' fue añadido a la lista de asuntos pendientes."

Puedes invocar `store.dispatch(action)` desde cualquier lugar en tu aplicación, incluyendo componentes y XHR callbacks, o incluso en intervalos programados.



# Redux.js



## Flujo de datos

### 2. El store en Redux invoca a la función reductora que le indicaste.

El store pasará dos argumentos al reductor: el árbol de estado actual y la acción. Por ejemplo, en el caso de la aplicación de asuntos pendientes, el reductor raíz podría recibir algo como esto:

```
// El estado actual de aplicación (listado de asuntos pendientes y un filtro)
let previousState = {
  visibleTodoFilter: 'SHOW_ALL',
  todos: [
    {
      text: 'Leer la documentación.',
      complete: false
    }
  ]
}

// La acción que se está realizando (agregando un asunto)
let action = {
  type: 'ADD_TODO',
  text: 'Entendiendo el flujo.'
}

// Tu reductor devuelve el siguiente estado de aplicación
let nextState = todoApp(previousState, action)
```



# Redux.js



## Flujo de datos

### 3. El reductor raíz puede combinar la salida de múltiples reductores en un único árbol de estado.

Como se estructura el reductor raíz queda completamente a tu discreción. Redux provee una función `combineReducers()` que ayuda, a "dividir" el reductor raíz en funciones separadas donde cada una maneja una porción del árbol de estado.

Así es como funciona `combineReducers()`. Digamos que usted tiene dos reductores, uno para una lista de asuntos y otro para la configuración del filtro asunto actualmente seleccionado:

```
function todos(state = [], action) {
  // Calcularlo de alguna manera...
  return nextState
}

function visibleTodoFilter(state = 'SHOW_ALL', action) {
  // Calcularlo de alguna manera...
  return nextState
}

let todoApp = combineReducers({
  todos,
  visibleTodoFilter
})
```



# Redux.js



## Flujo de datos

4. **El store en Redux guarda por completo el árbol de estado devuelto por el reductor raíz.**

¡Este nuevo árbol es ahora el siguiente estado de tu aplicación! Cada listener registrado usando `store.subscribe(listener)` será ahora invocado; los listeners podrán invocar `store.getState()` para obtener el estado actual.

Ahora, la interfaz de usuario puede actualizarse para reflejar el nuevo estado. Si utilizas herramientas como React Redux, este es el momento donde invocas `component.setState(newState)`.



# Redux.js



## Uso con react

Redux no tiene relación alguna con React. Puedes escribir aplicaciones Redux con React, Angular, Ember, jQuery o vanilla JavaScript.

Dicho esto, Redux funciona especialmente bien con librerías como [React](#) y [Deku](#) porque te permiten describir la interfaz de usuario como una función de estado, y Redux emite actualizaciones de estado en respuesta a acciones.

[React Redux](#) no está incluido en Redux de manera predeterminada. Debe instalarlo explícitamente:

```
npm install --save react-redux
```



# Redux.js



## Uso con react

Para asociar React con Redux se recurre a la idea de **separación de presentación y componentes contenedores**.

	Componentes de Presentación	Componentes Contenedores
<b>Propósito</b>	Como se ven las cosas ( <i>markup</i> , estilos)	Como funcionan las cosas (búsqueda de datos, actualizaciones de estado)
<b>Pertinente a Redux</b>	No	Yes
<b>Para leer datos</b>	Lee datos de los <i>props</i>	Se suscribe al estado en Redux
<b>Para manipular datos</b>	Invoca llamada de retorno (callback) desde los <i>props</i>	Envia acciones a Redux
<b>Son escritas</b>	Manualmente	Usualmente generados por React Redux



# Redux.js

Practica





marketing@compueducacion.mx



/CompuEducacion



01 (55) 5283 8260



www.CompuEducación.mx



Plaza Polanco, Jaime Balmes No. 11.  
Edificio B Piso 7  
Col. Los Morales Polanco