Contenido

[Html 12](#_Toc72854144)

[CSS 16](#_Toc72854145)

[Regla en css: 16](#_Toc72854146)

[Selectores de tipo 16](#_Toc72854147)

[Selectores de id y clase 16](#_Toc72854148)

[Selectores descendientes 16](#_Toc72854149)

[Textos 17](#_Toc72854150)

[Familia tipográfica: 17](#_Toc72854151)

[Tamaño de Fuente: 17](#_Toc72854152)

[Estilo de fuentes: 17](#_Toc72854153)

[Peso de fuente: 17](#_Toc72854154)

[Variantes de fuente: 18](#_Toc72854155)

[Color de letra: 18](#_Toc72854156)

[Alinear texto horizontalmente: 18](#_Toc72854157)

[Alinear texto verticalmente: 18](#_Toc72854158)

[decoration: 18](#_Toc72854159)

[Texto identado: 19](#_Toc72854160)

[Sombras de texto: 19](#_Toc72854161)

[Transformación de texto: 19](#_Toc72854162)

[Espacio entre letras de un texto: 19](#_Toc72854163)

[Espacio entre palabras de un texto: 20](#_Toc72854164)

[Espacio entre renglones de un texto: 20](#_Toc72854165)

[Propiedad word-wrap: 20](#_Toc72854166)

[Regla @font-face: 20](#_Toc72854167)

[Ancho y alto 21](#_Toc72854168)

[Propiedades relleno, borde y margen 21](#_Toc72854169)

[Relleno: 21](#_Toc72854170)

[Borde: 22](#_Toc72854171)

[Margen: 23](#_Toc72854172)

[Fondo 23](#_Toc72854173)

[Estilo para las listas 24](#_Toc72854174)

[Estilos para tablas 24](#_Toc72854175)

[Enlaces 25](#_Toc72854176)

[Pseudo selectores para maquetar enlaces con CSS 25](#_Toc72854177)

[Puntero del raton 25](#_Toc72854178)

[La propiedad display 27](#_Toc72854179)

[La propiedad visivility 27](#_Toc72854180)

[Posicionando elementos con CSS 27](#_Toc72854181)

[Propiedades de posicionamiento CSS 27](#_Toc72854182)

[Propiedad position y formas de posicionamiento 28](#_Toc72854183)

[Propiedad float 28](#_Toc72854184)

[Propiedad clear 28](#_Toc72854185)

[Propiedad overflow 28](#_Toc72854186)

[Propiedad Z-Index 29](#_Toc72854187)

[Esquinas redondeadas 29](#_Toc72854188)

[Sombras 29](#_Toc72854189)

[Degradados en fondos 30](#_Toc72854190)

[Imágenes de fondo 31](#_Toc72854191)

[La propiedad opacity 32](#_Toc72854192)

[pseudoclases y psudoelementos 32](#_Toc72854193)

[La propiedad content: 33](#_Toc72854194)

[Transiciones 33](#_Toc72854195)

[Propiedades de transición: 33](#_Toc72854196)

[Filtros 34](#_Toc72854197)

[JavaScript 35](#_Toc72854198)

[Basico 35](#_Toc72854199)

[Clase Date 39](#_Toc72854200)

[Clase array 39](#_Toc72854201)

[Diferentes formas de crear Arrays: 39](#_Toc72854202)

[Arrays densos: 40](#_Toc72854203)

[Arrays no densos o dispersos: 41](#_Toc72854204)

[Metodos push y pop: 42](#_Toc72854205)

[Métodos unshift y shift: 43](#_Toc72854206)

[Instrucción delete: 43](#_Toc72854207)

[Métodos sort y reverse: 43](#_Toc72854208)

[Método splice: 44](#_Toc72854209)

[Método slice: 47](#_Toc72854210)

[Método Join: 48](#_Toc72854211)

[Método concat: 49](#_Toc72854212)

[Método toString : 49](#_Toc72854213)

[Distintas formas de recorrer Arrays: 49](#_Toc72854214)

[Clase Math 50](#_Toc72854215)

[Clase String 51](#_Toc72854216)

[Expresiones regulares: 52](#_Toc72854217)

[Formularios y Eventos 58](#_Toc72854218)

[GetElemntById: 58](#_Toc72854219)

[Evento onChange(), SELECT, selectedIndex y options: 59](#_Toc72854220)

[Propiedad Checked: 60](#_Toc72854221)

[Eventos 62](#_Toc72854222)

[onFocus y onBlur: 62](#_Toc72854223)

[onMouseOver y onMouseOut: 63](#_Toc72854224)

[OnLoad: 64](#_Toc72854225)

[Load y DOMContentLoaded: 64](#_Toc72854226)

[Click y dblclick: 65](#_Toc72854227)

[Mousedown y mouseup: 66](#_Toc72854228)

[Mouseover y mouseout: 67](#_Toc72854229)

[Mousemove: 68](#_Toc72854230)

[keydown, keyup y keypress: 69](#_Toc72854231)

[Change: 70](#_Toc72854232)

[Focus y blur: 71](#_Toc72854233)

[Parámetro del método asociado al addEventListener: 72](#_Toc72854234)

[Submit: 75](#_Toc72854235)

[Objeto Window 77](#_Toc72854236)

[Propiedad location: 79](#_Toc72854237)

[Propiedad history: 79](#_Toc72854238)

[Propiedad Screen: 81](#_Toc72854239)

[Propiedad navigator: 81](#_Toc72854240)

[POO en JS. 82](#_Toc72854241)

[Definición de varias clases: 84](#_Toc72854242)

[Vectores con componentes de tipo objeto: 86](#_Toc72854243)

[Creación de objetos literales: 87](#_Toc72854244)

[Funciones 89](#_Toc72854245)

[Objeto arguments: 89](#_Toc72854246)

[Variables locales y variables globales: 90](#_Toc72854247)

[Funciones anidadas: 90](#_Toc72854248)

[Parámetros de tipo función: 91](#_Toc72854249)

[Función isNaN: 92](#_Toc72854250)

[Función isFinite: 93](#_Toc72854251)

[Funciones setInterval, clearInterval, setTimeout y clearTimeout: 95](#_Toc72854252)

[ES6 97](#_Toc72854253)

[Definición de constantes y variables: 97](#_Toc72854254)

[Plantillas de cadenas de caracteres: 101](#_Toc72854255)

[Funciones con parámetros por defecto: 102](#_Toc72854256)

[Funciones con parámetros Rest: 103](#_Toc72854257)

[Operador Spread (operador de propagación): 105](#_Toc72854258)

[Arrow functions (funciones flecha): 105](#_Toc72854259)

[Nombres de métodos abreviados: 107](#_Toc72854260)

[Declaración de clases: 108](#_Toc72854261)

[Herencia: 110](#_Toc72854262)

[Definición de getters y setters: 112](#_Toc72854263)

[Definición de métodos estáticos: 115](#_Toc72854264)

[Estructura repetitiva 'for of': 116](#_Toc72854265)

[Módulos: 118](#_Toc72854266)

[Map: 124](#_Toc72854267)

[Set: 128](#_Toc72854268)

[WeakMap y WeakSet : 130](#_Toc72854269)

[Array: 131](#_Toc72854270)

[JQuery 142](#_Toc72854271)

[Selección de un elemento del documento mediante el id: 144](#_Toc72854272)

[Selección de elementos por el tipo de elementos: 146](#_Toc72854273)

[Selección de elementos utilizando los selectores CSS: 148](#_Toc72854274)

[Selección de elementos utilizando las clases CSS definidas: 150](#_Toc72854275)

[Método text(), text(valor): 151](#_Toc72854276)

[Métodos attr(nombre de propiedad), attr(nombre de propiedad,valor) y removeAttr(nombre de propiedad): 153](#_Toc72854277)

[Métodos addClass y removeClass: 156](#_Toc72854278)

[Método html() y html(valor): 159](#_Toc72854279)

[Administración de eventos con jQuery: 160](#_Toc72854280)

[Eventos mouseover y mouseout: 163](#_Toc72854281)

[Evento hover: 164](#_Toc72854282)

[Evento mousemove: 165](#_Toc72854283)

[Eventos mousedown y mouseup: 166](#_Toc72854284)

[Evento dblclick: 167](#_Toc72854285)

[Evento focus: 169](#_Toc72854286)

[Evento blur: 170](#_Toc72854287)

[Manipulación de los elementos del DOM: 172](#_Toc72854288)

[Efectos con los métodos show y hide: 176](#_Toc72854289)

[Efectos con los métodos fadeIn y fadeOut: 179](#_Toc72854290)

[Efecto con el método fadeTo: 183](#_Toc72854291)

[Efecto con el método toggle: 186](#_Toc72854292)

[Iteración por los elementos (each): 188](#_Toc72854293)

[Bootstrap 190](#_Toc72854294)

[Cree la primera página web con Bootstrap 191](#_Toc72854295)

[Cuadrículas Bootstrap 192](#_Toc72854296)

[Sistema de cuadrícula Bootstrap: 192](#_Toc72854297)

[Clases de cuadrícula: 193](#_Toc72854298)

[Reglas del sistema de cuadrícula: 193](#_Toc72854299)

[Estructura básica de una cuadrícula de Bootstrap: 193](#_Toc72854300)

[Tipografia: 194](#_Toc72854301)

[Configuración predeterminada de Bootstrap: 194](#_Toc72854302)

[Bootstrap frente a valores predeterminados del navegador: 194](#_Toc72854303)

[Colores y fondos contextuales: 194](#_Toc72854304)

[Tablas de Bootstrap 195](#_Toc72854305)

[Tabla básica de Bootstrap: 195](#_Toc72854306)

[Striped Rows: 195](#_Toc72854307)

[Bordered Table: 195](#_Toc72854308)

[Hover Rows: 196](#_Toc72854309)

[Condensed Table: 196](#_Toc72854310)

[Clases contextuales: 196](#_Toc72854311)

[Responsive Tables: 197](#_Toc72854312)

[Imágenes de Bootstrap 197](#_Toc72854313)

[Esquinas redondeadas: 197](#_Toc72854314)

[Circulo: 197](#_Toc72854315)

[Miniatura : 197](#_Toc72854316)

[Responsive Images : 197](#_Toc72854317)

[Galería de imágenes: 198](#_Toc72854318)

[Incrustaciones receptivas: 199](#_Toc72854319)

[Bootstrap Jumbotron y encabezado de página 199](#_Toc72854320)

[Creando un Jumbotron: 199](#_Toc72854321)

[Contenedor Jumbotron: 199](#_Toc72854322)

[Encabezado de página de ejemplo: 200](#_Toc72854323)

[Wells 200](#_Toc72854324)

[Alertas de Bootstrap 200](#_Toc72854325)

[Enlaces de alerta: 201](#_Toc72854326)

[Alertas de cierre: 201](#_Toc72854327)

[Alertas animadas: 202](#_Toc72854328)

[Botones Bootstrap 202](#_Toc72854329)

[Estilos de botones: 202](#_Toc72854330)

[Tamaños de botones: 203](#_Toc72854331)

[Botones de nivel de bloque: 203](#_Toc72854332)

[Botones activos / desactivados: 203](#_Toc72854333)

[Grupos de botones de Bootstrap 204](#_Toc72854334)

[Grupos de botones verticales: 204](#_Toc72854335)

[Grupos de botones justificados: 204](#_Toc72854336)

[Grupos de botones anidados y menús desplegables: 205](#_Toc72854337)

[Menú desplegable de botones divididos: 205](#_Toc72854338)

[Glifos de Bootstrap 206](#_Toc72854339)

[Insignias y etiquetas de Bootstrap 207](#_Toc72854340)

[Insignias: 207](#_Toc72854341)

[Etiquetas: 207](#_Toc72854342)

[Barras de progreso de Bootstrap 207](#_Toc72854343)

[Barra de progreso con etiqueta: 208](#_Toc72854344)

[Barras de progreso de colores: 208](#_Toc72854345)

[Barras de progreso a rayas: 209](#_Toc72854346)

[Barra de progreso animada: 210](#_Toc72854347)

[Barras de progreso apiladas: 210](#_Toc72854348)

[Paginación Bootstrap 211](#_Toc72854349)

[Enlace deshabilitado: 212](#_Toc72854350)

[Tamaño de la paginación: 212](#_Toc72854351)

[Migas de pan: 212](#_Toc72854352)

[Pager: 213](#_Toc72854353)

[Grupos de listas de Bootstrap 213](#_Toc72854354)

[Lista de grupo con insignias: 214](#_Toc72854355)

[Grupo de lista con elementos vinculados: 214](#_Toc72854356)

[Clases contextuales: 215](#_Toc72854357)

[Contenido personalizado: 215](#_Toc72854358)

[Paneles Bootstrap 216](#_Toc72854359)

[Grupo de paneles: 217](#_Toc72854360)

[Paneles con clases contextuales: 217](#_Toc72854361)

[Desplegables Bootstrap: 218](#_Toc72854362)

[Encabezado desplegable: 219](#_Toc72854363)

[Deshabilitar y elementos activos: 219](#_Toc72854364)

[Dropup: 220](#_Toc72854365)

[Plegables de Bootstrap 220](#_Toc72854366)

[Panel plegable: 220](#_Toc72854367)

[Grupo de lista contraíble: 221](#_Toc72854368)

[Acordeón: 222](#_Toc72854369)

[Píldoras y pestañas Bootstrap 223](#_Toc72854370)

[Menús: 223](#_Toc72854371)

[Pestañas: 224](#_Toc72854372)

[Pestañas con menú desplegable: 224](#_Toc72854373)

[Pastillas: 225](#_Toc72854374)

[Pastillas verticals: 225](#_Toc72854375)

[Pastillas verticales seguidas: 226](#_Toc72854376)

[Pastillas con menú desplegable: 226](#_Toc72854377)

[Centradas: 227](#_Toc72854378)

[Pestañas dinámicas / alternables: 227](#_Toc72854379)

[Pastillas alternas / dinámicas: 228](#_Toc72854380)

[Barra de navegación Bootstrap 228](#_Toc72854381)

[Barra de navegación invertida: 229](#_Toc72854382)

[Barra de navegación con menú desplegable: 229](#_Toc72854383)

[Barra de navegación alineada a la derecha: 230](#_Toc72854384)

[Botones de la barra de navegación: 231](#_Toc72854385)

[Formularios de barra de navegación: 231](#_Toc72854386)

[Texto de la barra de navegación: 232](#_Toc72854387)

[Barra de navegación fija: 232](#_Toc72854388)

[Colapsar la barra de navegación: 233](#_Toc72854389)

[Formularios de Bootstrap 235](#_Toc72854390)

[Forma vertical de Bootstrap (predeterminado): 235](#_Toc72854391)

[Formulario en línea de Bootstrap: 236](#_Toc72854392)

[Forma horizontal Bootstrap: 237](#_Toc72854393)

[Entradas de formulario Bootstrap 238](#_Toc72854394)

[Bootstrap Input: 238](#_Toc72854395)

[Bootstrap Textarea: 239](#_Toc72854396)

[Bootstrap Checkboxes: 239](#_Toc72854397)

[Bootstrap Radio Buttons: 240](#_Toc72854398)

[Bootstrap Select List: 240](#_Toc72854399)

[Control estático: 240](#_Toc72854400)

[Grupos de entrada de Bootstrap: 241](#_Toc72854401)

[Estados de control de formulario de Bootstrap: 242](#_Toc72854402)

[Tamaño de inputs de Bootstrap 244](#_Toc72854403)

[Tamaño de inputs en formularios: 244](#_Toc72854404)

[Tamaño de altura: 244](#_Toc72854405)

[Tamaño de columna: 245](#_Toc72854406)

[texto de ayuda: 245](#_Toc72854407)

[Bootstrap Media Objects 246](#_Toc72854408)

[Objetos multimedia: 246](#_Toc72854409)

[Objeto multimedia básico: 246](#_Toc72854410)

[Alineación superior, media o inferior: 247](#_Toc72854411)

[Anidar objetos multimedia: 249](#_Toc72854412)

[Bootstrap Carousel 250](#_Toc72854413)

[Agregar subtítulos a las diapositivas: 252](#_Toc72854414)

[Modal Bootstrap 253](#_Toc72854415)

[Tamaño modal: 255](#_Toc72854416)

[Bootstrap Tooltip 255](#_Toc72854417)

[Bootstrap Popover 256](#_Toc72854418)

[Bootstrap Scrollspy 257](#_Toc72854419)

[Bootstrap Affix 258](#_Toc72854420)

[Scrollspy & Affix: 259](#_Toc72854421)

[Filtros Bootstrap 260](#_Toc72854422)

[Tablas de filtros: 260](#_Toc72854423)

[Listas de filtros: 260](#_Toc72854424)

[Filtrar menús desplegables: 261](#_Toc72854425)

[PHP 261](#_Toc72854426)

[Sintaxis PHP 262](#_Toc72854427)

[Comentarios en PHP: 263](#_Toc72854428)

[Variables PHP: 264](#_Toc72854429)

[Alcance de las variables de PHP: 265](#_Toc72854430)

[Alcance global y local: 265](#_Toc72854431)

[PHP La palabra clave global: 266](#_Toc72854432)

[PHP La palabra clave static: 267](#_Toc72854433)

[Declaraciones echo y print en PHP: 267](#_Toc72854434)

[Tipos de datos PHP 268](#_Toc72854435)

[Strings PHP 271](#_Toc72854436)

[Matemáticas PHP 272](#_Toc72854437)

[Constantes PHP 272](#_Toc72854438)

[Matrices constantes PHP: 273](#_Toc72854439)

[Operadores PHP 273](#_Toc72854440)

[Declaraciones condicionales de PHP 276](#_Toc72854441)

[PHP - La declaración if: 276](#_Toc72854442)

[PHP - La declaración if ... else: 276](#_Toc72854443)

[PHP - La declaración if ... elseif ... else: 277](#_Toc72854444)

[La declaración de switch de PHP 277](#_Toc72854445)

[Bucles PHP 277](#_Toc72854446)

[While: 277](#_Toc72854447)

[Do while: 278](#_Toc72854448)

[For: 278](#_Toc72854449)

[Foreach: 278](#_Toc72854450)

[Break & Continue: 279](#_Toc72854451)

[Funciones PHP 279](#_Toc72854452)

[Argumentos de funciones PHP: 280](#_Toc72854453)

[Valor de argumento predeterminado de PHP: 281](#_Toc72854454)

[Devolución de valores: 281](#_Toc72854455)

[Pasar argumentos por referencia: 282](#_Toc72854456)

[Matrices PHP 283](#_Toc72854457)

[Matrices indexadas PHP: 283](#_Toc72854458)

[Matrices asociativas PHP: 284](#_Toc72854459)

[Matrices multidimensionales: 284](#_Toc72854460)

[Matrices bidimensionales: 285](#_Toc72854461)

[Funciones de ordenación para matrices 286](#_Toc72854462)

[Variables globales de PHP – Superglobales 287](#_Toc72854463)

[Expresiones regulares PHP 291](#_Toc72854464)

[Funciones de expresión regular: 291](#_Toc72854465)

[Manejo de formularios PHP 292](#_Toc72854466)

[GET VS POST: 293](#_Toc72854467)

[Validación de formularios PHP 294](#_Toc72854468)

[Nota importante sobre la seguridad de los formularios PHP: 295](#_Toc72854469)

[¿Cómo evitar las vulnerabilidades de $ \_SERVER ["PHP\_SELF"]? 296](#_Toc72854470)

[Validar datos de formulario con PHP: 296](#_Toc72854471)

[Formularios PHP - Campos obligatorios 298](#_Toc72854472)

[PHP - Mostrar los mensajes de error: 299](#_Toc72854473)

[Formularios PHP : validar correo electrónico y URL 299](#_Toc72854474)

[Fecha y hora de PHP 301](#_Toc72854475)

[Crear una fecha con mktime (): 303](#_Toc72854476)

[Crear una fecha a partir de una cadena con strtotime (): 303](#_Toc72854477)

[PHP Incluir archivos 303](#_Toc72854478)

[PHP incluye vs.requiere: 305](#_Toc72854479)

[Manejo de archivos PHP 305](#_Toc72854480)

# Html

|  |  |
| --- | --- |
| Etiquetas | Atributos |
| * <html>: documento html * <head>: cabecera de la página * <title>: título de la página en la cabecera * <body>: contenido de la página | * Id =” ”: Para identificar un elemento único. * class =” ”: Identificador múltiple. * align =” ”: Alineación de contenido. * border =” ”: Para darle borde al contenido. * style =” ”: Para darle un estilo al contenido. * background-color =” ”: Para color de fondo. * height =” ”: Para determinar altura. * width =” ”: Para determinar ancho. * disable=” ”: inhabilitar. * on click=””: click que llama a una función de JS. |
| * <header.- contiene la cabecera * <nav>.- contiene una barra de navegación o lista de hipervínculos. * <aside>.- puede ser un sidebar o un bloque de contenido secundario * <article>.- el contenido principal * <section>.- secciones de la pagina * <Footer.- el pie de página. |
| * <b>: Para hacer texto en negrita. * <i>: Texto en cursiva. * <p>: Para elementos de párrafo. * <br>: Indica un salto de línea * <h1>: Encabezado o título principal. * <h2>, <h3>, <h4>, <h5>, <h6>: indica niveles de encabezado, subtítulos. * <strong>: Textos importantes (negrita) * <em>: enfatizar texto * <u>: Para texto subrayado. * <big>: Texto grande. * <small>: Para texto pequeño. * <sub>: Para texto subíndice. * <sup>: Para texto superíndice. * <ins>: Para texto insertado. * <del>: Para texto borrado. * <mark>: Para texto marcado. * <font>: estilos de letra * <blockquote>: citas. * <code>: codigo. | * zise=” ”: tamaño. * face=” “: fuente. * color=” “:fuente. |
| * <a>: Para crear enlaces. * <link>: invoca un archivo (css por ejemplo) * <script src="funciones.js"></script>: invoca un archivo JS. | * href =” ”: El atributo href contiene el valor de la url de destino o dirección de archivo. * target="\_blank": para que el enlace se habrá en una nueva ventana. * rel=” “: define la relación entre el documento HTML y el documento CSS invocado, el valor “stylesheet” indica que es un estilo preferido, mientras que “slternate stylesheet” define un estilo alterno. |
| * <img>: Para insertar imágenes. | * src=“ ”: Para indicar la ubicación o dirección de la imagen. * alt=” “:contiene el texto alternativo, este texto se mostrará si por alguna razón no carga la imagen (Descripción de la imagen) * height =” ”: Para determinar altura. * width =” ”: Para determinar ancho. * Border=” “:border crea un borde para la imagen el valor de este atributo se puede expresar en píxeles o en otra unidad de medida compatible. * vspce= “ “:coloca un espacio en la parte superior e inferior de la imagen y su valor se puede expresar en píxeles u otra unidad. |
| * <ol>: Lista ordenada * <li>: elementos de la lista * <ul>: Para listas sin ordenar * <dl>.- Esta etiqueta se utiliza para crear una lista de definiciones si lo deseamos. * <dt>.- Esta etiqueta contiene el término que queremos definir. * <dd>.- Esta etiqueta contiene la definición del término que le precede. | * style =” ”: listas con viñetas.   + disc: este es el valor predeterminado.   + circle: muestra marcador es un círculo hueco.   + square: muestra marcador es un cuadrado.   + decimal: marcador es un número decimal, que comienza con 1.   + lower-roman: marcador es un número romano en minúscula (i, ii, iii, iv, v, etc.)   + upper-roman: el marcador es un número romano en mayúscula (I, II, III, IV, V, etc.)   + lower-greek   + lower-latin   + upper-latin   + lower-alpha   + upper-alpha   + list-style-image: imagen.   + none. |
| * <table>: para crear tablas. * <tr>: Filas * <td>:Columnas * <th>: Encabezado * <caption>: titulo * <thead>: cabecera. * <tbody>: cuerpo. * <tfood>: pie. | * border=” ”: borde de la tabla * rowspan=” ”: combinar celdas de distintas filas. * colspan=” ”: combinar celdas de distintas columnas |
| * <form>: Para crear formularios. * <input>: campo de formulario. * <textarea>: área de texto amplia. * <select>: campo desplegable. * <option>: opciones de select. * <button>: botón * <label>: texto. * <fieldset>: agrupar elementos. * <legend>: nombre del fieldset. | * type=” ”:   + text: para campos de texto   + password: para campos de contraseña   + checkbox: para casillas de verificación   + radio: para casillas de selección   + submit: para botones de envío   + reset: para botones de resetear   + file: para campos de selección de archivo   + color: crea un campo con selector de color, de acuerdo al sistema operativo, aparece una paleta para escoger un color.   + date: se creará un campo de fecha, en ella se pueden colocar el día, mes y año.   + datetime: permite ingresa información de hora, minuto y segundo.   + datetime-local: genera un campo para fecha y hora, podemos colocar en ella el día, mes, año y hora.   + email: para información de correo electrónico.   + month: genera un campo para ingresar un mes del año.   + number: para recopilar datos numéricos.   + range: crea una barra con un botón desplazable para elegir un rango, por defecto el rango es de 0 a 100.   + search: para generar una caja de búsqueda.   + tel: para recopilar información de número telefónico.   + time: crea un campo para ingresar hora y minuto.   + url: para información de dirección URL.   + week: crea un campo para elegir la semana del año. * name=” ”:nombre del formulario o input. * action=” ”: será la URL de la página que queremos que cargue después de que se envíe la información de un formulario. * method=””: puede tomar como valores a GET o POST y esto indica el método http que será utilizado cuando se envíe la información. * rows=” “: filas de un textarea. * cols=” “: columnas de un textarea. * maxlength=” ”: longitud máxima. * autocomplete: cuando se establece autocomplete activado, el navegador trata de autocompletar los campos que el usuario está completando, basándose en su actividad en el navegador. * required: para campos requeridos, sirve para establecer de que el campo debe ser completado obligatoriamente, en caso contrario no será posible enviar la información del formulario. * placeholder: generalmente se usa para colocar en su valor un breve texto que indique al usuario el tipo de información que debe coloca en el campo. Se suele usar en elementos <input> y <textarea> * autofocus: sirve para enfocar automáticamente el campo deseado cuando el formulario se carga. * size: para establecer el tamaño de los campos, un número de caracteres para campos de texto. * readonly: sirve para que la información de un campo no pueda ser modificada, solo de lectura. * disabled: este atributo hace que un campo o control quede deshabilitado, de tal manera que no se puede hacer nada con ella. |
| * <audio>: sirve para inserta un reproductor de sonido. * <video>: insertar un reproductor de vídeo. * <source>: insertar diferentes formatos de audio y video. | * src=” ”: nombre del archivo. * controls: reproductor con sus iconos para reproducir el sonido, controlar el avance y volumen, pausar, etc. * type=” “: especificar el tipo de archivo. * autoplay: hará que el archivo se reproduzca automáticamente al cargarse la página, sin consultar al usuario. * loop: hará que el archivo se reproduzca indefinidamente una vez que se inicia, es decir al finalizar, empezara a reproducirse automáticamente de nuevo. |
| * <progress>: crear una barra de progreso. | * Min: su valor será el valor mínimo de la barra de progreso. * Max: su valor será el valor máximo de la barra de progreso. * Value: su valor será un valor entre el valor mínimo y el valor máximo y se reflejara en la barra de progreso. |
| * <div>: contenedor en bloque. * <span>: contenedor en línea. * <hr>: Es una etiqueta que crea una línea horizontal o separador * <center>: centrar * <!-- esto es un comentario --> |  |

# CSS

## Regla en css:

1. El selector: el selector es quien indica a que elemento HTML se aplica el código CSS (el estilo)
2. La propiedad: es la característica que se desea definir, por ejemplo: el color.
3. Valor: es el valor que tomará la propiedad, por ejemplo: verde.
4. Punto y coma (;): una declaración siempre se debe terminar con un punto y coma simple (;)

## Selectores de tipo

Los selectores de tipo en CSS se refieren a los electores que apuntan a un tipo de elemento html como puede ser:

* Los títulos: h1
* Los subtitulos: h2
* Párrafos: p
* Los enlaces: a
* Tablas: table

## Selectores de id y clase

Este tipo de selectores en CSS apuntan a un elemento HTML identificado, es decir a un elemento HTML que tenga el atributo id o class.

#valor de identificación {propiedades;}

## Selectores descendientes

Los selectores descendientes en CSS sirven para apuntar elementos que se encuentran dentro otros elementos mayores, de esta manera se puede colocar un estilo específico a un elemento que se encuentra contenido en otro elemento HTML mayor.

#id .class elemento {propiedades;}

/\*Comentario css\*/

## Textos

### Familia tipográfica:

font-family: podemos definir el tipo de letra o familia tipográfica para cualquier elemento de texto que tengamos, estos pueden ser por ejemplo: un párrafo, títulos, subtítulo, enlaces, etc.

* p {font-family: cursive;}
* h1 {font-family: georgia;}
* h2 {font-family: fantasy;}
* a {font-family: new times roman;}
* h3 {font-family: monospace;}

Es importante colocar más de una familia tipográfica dentro de los valores de la propiedad font-family, esto con el objetivo de asegurar un estilo específico:

* p {font-family: arial, helvetica, roboto, "new times roman", lato;}

### Tamaño de Fuente:

font-size: indica el tamaño de fuente para nuestros elementos de texto. Podemos establecer el tamaño de letra con palabras clave o indicando una medida exacta con sus respectivas unidades.

|  |  |  |
| --- | --- | --- |
| * p {font-size: small;} * p {font-size:medium;} * p {font-size: large;} * p {font-size: x-large;} | * h1{font-size: 32px;} * p {font-size: 8px;} * p {font-size: 10px;} * p {font-size: 15px;} | * h2{font-size: 2em;} |

### Estilo de fuentes:

font-style: nos permite establecer un texto en cursiva, los valores que puede tomar esta propiedad son:

* p {font-style: normal;} el texto se muestra normal, sin efectos.
* p {font-style: italic;} el texto se muestra en cursiva.
* p {font-style: oblique;} texto en oblicua, es parecido a italic.

### Peso de fuente:

font-weight: nos permite establecer el ancho de un texto, podemos elegir entre textos con grosores desde muy delgados hasta muy anchos independientemente de otras propiedades.

* p {font-weight: lighter;} este valor produce un texto ligero o delgado.
* p {font-weight: normal;} Normal define el grosor de texto por defecto, sin modificar el grosor.
* p {font-weight: bold;} establece texto grueso o en negrita.
* p {font-weight: bolder;} produce un texto bastante grueso.
* p {font-weight: 100;}
* p {font-weight: 500;}
* p{font-weight: 900;}

### Variantes de fuente:

font-variant: sirve para cambiar el texto a mayúsculas pequeñas.

* P {font-variant: normal;} no aplica ningún cambio.
* p {font-variant: small-caps;} establece que las letras deben ser mayúsculas pequeñas.
* p {Font-variant: inherit;} Hereda el estilo del elemento padre que lo contiene.

### Color de letra:

|  |  |  |
| --- | --- | --- |
| p {color:red;}  p {color:blue;}  p {color:yellow} | p {color: #ff0000;}  p {color:#0000ff;}  p {color:#ffff00;} | p {color: rgb(255, 0, 0);}  p {color: rgb(0, 0, 255);}  p {color: rgb:(255,255,0);} |

### Alinear texto horizontalmente:

text-align: es la que nos permite determinar la alineación de los elementos de texto.

* p {text-align: left;} indica que el texto debe ser alineado a la izquierda.
* p {text-align: right;} indica que el texto debe ser alineado a la derecha.
* p {text-align: center;} establece texto con alineación al centro.
* p {text-align: justify;} hace que el texto se justifique, quedando las lineas estiradas de principio a fin para que todos queden iguales.

### Alinear texto verticalmente:

vertica-align: se emplea en general para establecer la alineación vertical de un elemento HTML dentro de otro.

* p {vertical-align: top;} Indica que el texto se ubica en la parte superior.
* p {vertical-align: middle;} establece que el texto se ubica en al medio.
* p {vertical-align: bottom;} Ubica al texto en la parte inferior.
* span {vertical-align: baseline;} alinea el elemento padre con la linea del elemento hijo.
* span {vertical-align: sub;} Alinea la línea base del elemento seleccionado con la línea subíndice del elemento padre.
* span {vertical-align: super;} Alinea la línea base del elemento seleccionado con la línea superíndice del elemento padre.

### decoration:

text-decoration: nos permite decorar nuestros elementos de texto, básicamente se trata de agregar una línea decorativa ya sea debajo, encima o al medio del texto en cuestión.

* p {text-decoration: none;} evita que se aplique algún estilo de decoración de texto.
* p {text-decoration: inherit;} hereda el estilo del elemento padre que lo contiene.
* p {text-decoration: overline;} decora con una línea horizontal encima del texto.
* p {text-decoration: underline;} decora con una línea horizontal debajo del texto.
* p {text-decoration: line-through;} este valor hace que se dibuje una línea horizontal atravesando al texto.

### Texto identado:

text-indent: sirve para establecer cuanto el espacio que se debe dejar horizontalmente para empezar con la primera línea de un texto.

* p {text-indent: 70px;}

### Sombras de texto:

text-shadow y en sus valores colocamos las medidas y color de sombra deseado.

* Distancia horizontal: Es el primero en ser definido, se expresa con un valor numérico y su unidad de medida. Establece cuánto será la sombra en dirección horizontal, si el valor es positivo la sombra se dirige a la derecha y si es negativo a la izquierda.
* Distancia vertical: Es el segundo valor que debemos definir y también con un valor numérico con unidades de medida. Establece cuánto será la sombra en dirección vertical, si el valor es positivo, la sombra se dirige hacia abajo y si es negativo hacia arriba.
* Desenfoque: Determina el radio de desenfoque, este valor también es numérico y hará que la sombra tenga un aspecto borroso, de tal manera que aparente una sombra.
* color de sombra: definimos un color para la sombra.
* h3 {text-shadow: 7px 4px 5px blue;}
* h1 {

text-shadow:

3px 3px 1px red,

5px 5px 2px yellow,

8px 8px 3px blue;

}

### Transformación de texto:

text-transform: sirve para transformar nuestros elementos de texto en varios formatos, ya sea a minúsculas, mayúsculas, sola las iniciales, etc.

* p {text-transform: none;} es el valor por defecto y evita cualquier transformación.
* p {text-transform: capitalize;} Convierte en mayúscula solo a las primeras letras de cada palabra escrita.
* p {text-transform: uppercase;} Indica que todas las palabras del texto en cuestión sean transformadas a mayúsculas.
* p {text-transform: lowercase;} Este valor hará que todas las palabras se conviertan en minúsculas.

### Espacio entre letras de un texto:

Letter-spacing: Es la propiedad de CSS que nos permite controlar el espacio entre letras de un elemento de texto, pueden ser: normales, amplios, estrechos y además pueden ser expresados con valor numérico y su unidad de medida.

* p {letter-spacing: normal;}
* p {letter-spacing: 4px;}
* p {letter-spacing: -2px;}

### Espacio entre palabras de un texto:

Word-spacing: Es la propiedad de CSS que nos permite controlar el espacio entre palabras dentro de un texto.

* p.normal {word-spacing: normal;}
* p.posit {word-spacing: 8px;}
* p.nega {word-spacing: -2px;}

### Espacio entre renglones de un texto:

Line-height: Es la propiedad de CSS que nos permite controlar el espacio entre renglones dentro de un texto.

* p.normal { line-height: normal;}
* p.posit { line-height: 8px;}
* p.nega { line-height: -2px;}

### Propiedad word-wrap:

sirve para controlar palabras largas que pueden exceder un contenedor, podemos hacer que la palabra larga se interrumpa y continúe en la siguiente línea, de tal manera que queda ajustada dentro del contenedor o podemos dejar tal cual

* #texto {word-wrap:normal;} hace que la palabra. no se altere y quede como por defecto.
* #texto {word-wrap:break-word;} hará que las palabras largas que sobrepasa un contenedor se ajuste y quede totalmente dentro de dicho contenedor, para ello se romperán dichas palabras si es necesario para continuar en la siguiente línea de texto.

### Regla @font-face:

@font-face es una regla de CSS que permite establecer una fuente tipográfica o también conocida como tipografía o tipo de letra para los elementos de texto que se desean. En palabras simples, @font-face es la regla que debemos usar si queremos emplear algún tipo de letra en especial, que no esté disponible en el dispositivo del usuario por defecto.

* @font-face {

font-family: nombrepersonalizado;

src:url();

font-style: cursive;

font-weight:700;

}

p {

font-size:30px;

font-family:nombrepersonalizado;

}

* Propiedades:
  + font-stretch: nos permite especificar como se alarga el texto, es decir la distancia entre letras y el valor predeterminado es el normal, no altera el texto.
    - Normal: normal
    - Condensado: condensed
    - ultra-condensado: ultra-condensed
    - extra-condensada: extra-condensed
    - semi-condensado: semi-condensed
    - expandido: expanded
    - semi-expandido: semi-expanded
    - extra-expandida: extra-expanded
    - ultra-expandido: ultra-expanded
  + font-weight: es bastante conocida, pues nos permite definir el peso de la fuente, sus valores pueden ser normal, bold y números que van del 100 al 900.

Un valor de bold hace que el texto se vuelva en negrita y los valores numéricos son proporcionales. Un valor de 100 crea un texto delgado, a medida que va creciendo 900 es el texto más grueso. Valores intermedio crear efectos proporcionales.

* + Font-style: es otra bastante conocida, nos permite definir un estilo, el valor por defecto que toma es el normal, podemos elegir entre itálico y oblicuo, estos harán que el texto adquiera una ligera y sutil inclinación.
    - Normal: normal
    - itálico: italic
    - oblicuo: oblique
  + unicode-range: Esta es una propiedad opcional que sirve para establecer el rango específico de caracteres que puede ser usada por la fuente establecida, definir esta propiedad hará que ciertos caracteres estén disponibles cuando sea necesario en la página. Si dentro del contenido existe algún carácter dentro del rango establecido, entonces la fuente se descarga, en caso contrario, es decir si en el contenido no se emplea algún carácter del rango establecido, entonces la fuente no se descarga innecesariamente.

## Ancho y alto

p {width: 500px;}: Nos permite establecer el ancho de cualquier elemento HTML, ya sea un contenido específico o un contenedor.

p {height: 500px;}: Indicar la altura de cualquier elemento HTML, ya sea un elemento tipo bloque o de línea.

* min-width (Ancho mínimo)
* min-height (Altura mínima)
* max-width (Ancho máximo)
* max-height (Altura máxima)

## Propiedades relleno, borde y margen

### Relleno:

* .imagen {padding: 50px}: Poner relleno uniforme a un elemento de HTML.
* .elemento {padding: 40px 20px 65px 80px;}: Poner rellenos diferentes a cada lado del elemento. El primero define la dimensión del relleno superior, el segundo la dimensión del lado derecho, el tercero de la parte inferior y el cuarto del lado izquierdo.
* padding-top: margen superior
* padding-bottom: margen inferior
* padding-left: margen izquierdo
* padding-right: margen derecho

### Borde:

* p {border:7px solid blue;}: Definir un borde completo en CSS. En su valor se define el ancho, estilo y color, en ese orden, separado por espacios.
* p {border-width: 3px;}: border-width establece el ancho de borde uniforme para un elemento.
* p {border-width: 7px 8px 4px 5px;}: definir bordes con anchos diferentes en una única propiedad con cuatro valores, el primer valor indica el ancho del borde superior, seguido del derecho, inferior e izquierdo.
* border-bottom-width (Ancho de borde inferior)
* border-top-width (Ancho de borde superior)
* border-left-width (Ancho de borde izquierdo)
* border-right-width (Ancho de borde derecho)
* p {border-color: blue;}: Nos permite establecer el color uniforme de borde de un elemento.
* p { border-color:red blue yellow green;}: Establecer colores diferentes para borde.
* border-top-color (color de borde superior)
* border-right-color (color de borde derecho)
* border-bottom-color (color de borde inferior)
* border-left-color (color de borde izquierdo)
* p {border-style: solid;}: Establecer un estilo uniforme en todos los bordes de un elemento.
* border-top-style (superior)
* border-right-style (derecho)
* border-bottom-style (inferior)
* border-left-style (izquierdo)

#### Valores de border-style CSS:

* hidden.- Oculta el borde.
* none.- evita que se muestre el borde y es el valor por defecto.
* solid.- muestra un borde sólido, linea recta y continua.
* dotted.- el borde se muestra con puntos redondeados.
* dashed.- muestra un borde formado por segmentos de línea, como si fueran guiones.
* double.- estilo con doble linea.
* groove.- muestra una línea con apariencia tallada.
* ridge.- este muestra un borde de estilo extruido.
* inset.- estilo de borde le da una apariencia de incrustado al elemento.
* outset.- hace que el elemento parezca en relieve.

### Margen:

* p. {margin: 30px;}: poner márgenes de igual medida en todos los lados de un contenido.
* img {margin: 7px 8px 9px 14px;}: Establecer márgenes con diferentes dimensiones a cada lado de los elementos HTML. La primera corresponde al margen superior, sigue el margen derecho, luego inferior y finalmente el izquierdo.
* p.elemento {margin; 20px 30px;}: margen vertical indica el valor del margen superior e inferior, mientras que el margen horizontal influye sobre el margen derecho e izquierdo.
* p.elemento {margin: 10px 15px 14px;}: el primero corresponde al margen superior, el segundo al margen horizontal, esto incluye al derecho e izquierdo, el tercer valor corresponde al margen de abajo.
* margin-top (margen superior)
* margin-right (margen derecho)
* margin-bottom (margen inferior)
* margin-left (margen izquierdo)
* margin: auto: elemento centrado.

## Fondo

* div {background-color:#abbcff;}: Color de fondo hexadecimal.
* p {background-color: purple;}: Color de fondo palabra clave.
* p {background-color: rgb(239, 127, 26); }: Color de fondo RGB.
* div {background-image: url("fondo.jpg");}: Imagen de fondo.
* div {background-image: url("gatito.png"), url("https://unnsitioimg.net/imagenes/gatito.png");}: varias imágenes de fondo.
* body {background-image: url("fondo.jpg"); background-repeat: repeat-x;}: La propiedad background-repeat nos permite determinar la dirección en que se repite la imagen de fondo o si deseas que no se repita.
  + repeat-x.- solo repite en dirección x u horizontalmente.
  + repeat-y.- solo repite en dirección y o verticalmente.
  + no-repeat.- no se repite en ninguna dirección.
* La propiedad background-repeat puede ser heredada a los elementos hijo siempre y cuando se establece el valor heredar en dichos elementos hijo (inherit).
* div {background-image: url("fondo.jpg"); background-repeat: no-repeat; background-attachment: scroll;}: La propiedad background-attachment nos permite definir si la imagen de fondo se queda fija o se mueve en la pantalla cuando el usuario se desplace sobre ella o haga scroll.
  + scroll.- hace que la imagen de fondo se mueva en la pantalla junto con el elemento que lo contiene.
  + fixed.- la imagen de fondo queda fija mientras el bloque se mueve con el resto del contenido.
* El valor de background-attachment también puede ser heredado de un elemento padre a un elemento hijo, para ello se emplea inherit como valor para elemento hijo.

## Estilo para las listas

* ul {list-style-type: circle;}: nos permite establecer el tipo de estilo que queremos para los marcadores de la lista.
  + disc.- indica un marcador en forma de disco.
  + circle.- marcador con forma de círculo.
  + square.- tipo de estilo cuadrado.
  + decimal.- Este valor enumera la lista empezando por 1.
  + decimal-leading-zero.- Empieza la numeración con un 01.
  + lower-roman.- lista con números romanos en minúscula (i, ii, iii, iv, v, etc.).
  + upper-roman.- números romanos en mayúscula (I, II, III, IV…).
  + lower-greek.- letras griegas en minúscula alfa/α, beta/β, gamma/γ, …
  + lower-latin.- indica letras ASCII en minúscula (a, b, c, … z).
  + upper-latin.- letras ASCII en mayúscula (A, B, C, … Z).
  + armenian.- aplica numeración armenia tradicional (ayb/ayp, ben/pen, gim/keem, …).
  + georgian.- numeración georgiana tradicional (an, ban, gan, …, he, tan, in, in-an, …).
  + none.- no aplica ningún estilo.
* ul {list-style-image:url("sonrisa.jpg");} nos permite establecer una imagen como viñeta para los elementos de una lista.
* ul {list-style-position:inside;} nos permite establecer la posición de la caja de viñetas.
  + inside.- indica que la lista está adentro del flujo de contenido.
  + outside: este es el valor por defecto e indica que la lista está fuera del flujo de contenido.

## Estilos para tablas

* table {border-collapse: separate;} nos permite colapsar esos bordes separados en un solo borde o mantenerlos separados.
  + inherit.- establece que se debe heredar el estilo del elemento padre que contiene a la tabla.
  + separate.- indica que los bordes deben ser separados.
  + collapse.- hace que los bordes sean colapsados en uno solo.
* table {border-collapse: separate; border-spacing: 30px;} border-spacing nos permite establecer el espacio entre celdas adyacentes de una tabla y solo funciona cuando los bordes de celdas están separadas. Podemos definir espacios uniformes en todos los lados al expresar un solo valor numérico.
* table {border-collapse: separate; border-spacing: 70px 15px;} en este caso el primero establece el espaciado horizontal y el segundo el espaciado vertical.
* caption {caption-side:bottom;} está pensada para posicionar la leyenda (descripción del contenido) de las tablas con CSS.
  + top.- Coloca la leyenda en la parte superior.
  + bottom.- Indica que la leyenda va en la parte inferior.
* table {border-collapse:separate; empty-cells: hide;} empty-cells nos permite establecer si se muestran o no los bordes de celdas que no poseen contenido dentro de ella.
  + Show.- indica que los bordes de celdas sin contenido se muestren.
  + Hide.- indica que los bordes de celdas vacías no se muestren.
* table { table-layout: auto; width: 10px;} table-layout sirve para determinar el ancho de las columnas de una tabla, nos referimos a la dimensión horizontal de celdas.
  + Auto.- este es el valor por defecto, indica que el ancho se ajusta al contenido automáticamente, es decir será proporcional a la cantidad de contenido que poseen las celdas.
  + Fixed.- indica que el ancho de columna es fijo y no se ajusta al contenido que contienen las celdas, por ello es posible que el contenido se salga de una celda, visualmente invadiendo a otras.
* Tr: nth-child(even){

Background-color: ;

} nth-child() sirve para aplicar estilos a las filas pares o impares (odd, even).

## Enlaces

### Pseudo selectores para maquetar enlaces con CSS

* a: link.- este pseudo selector sirve para apuntar a enlaces normales de la página web (no accedidos).
* a:visited.- mediante este pseudo selector apuntaremos a los enlaces visitados o accedidos por el usuario.
* a:hover.- nos permite establecer el estilo para enlaces cuando el cursor del ratón se encuentra sobre dichos enlaces.
* a:active.- se refiere a aquellos enlaces en las que se han hecho clic y están activos.

Para que estos pseudo selectores no entren en conflicto al momento de ser procesados, debemos tener presente que a:hover puede ir después de a:link y a:visited. Por otro lado, a:active será establecida después de a:hover.

* a.sin {text-decoration: none;} Eliminar el subrayado por defecto con el que aparecen los enlaces

## Puntero del raton

Personalizar el puntero del mouse con CSS es tan simple como identificar el elemento o zona sobre la cual queremos que un tipo de puntero específico se aplique y establecer mediante CSS cual será dicha forma o tipo de cursor.

* p.pointer {cursor:pointer;}
  + defaul.- es el puntero por defecto, con forma de flecha.
  + help.- cursor de ayuda, con forma de flecha acompañado de un signo de interrogación.
  + crosshair.- puntero en forma de cruz.
  + move.- cursor en forma de cruz pero con flechas en cada extremo.
  + pointer.- en forma de manito con el índice señalando.
  + auto.- el navegador determina automáticamente que tipo de cursor mostrar de acuerdo al tipo de elemento que se presente.
  + none.- el cursor desaparece.
  + context-menu.- indica la presencia de un menú contextual.
  + help.- puntero acompañado de un signo de interrogación, indica que hay ayuda disponible.
  + pointer.- posee forma de mano y generalmente es empleado cuando se pasa sobre enlaces.
  + progress.- este puntero con forma circular, indica que un programa está en progreso o está ocupado en segundo plano pero el usuario aún puede interactuar con la interfaz.
  + wait.- También posee la forma circular, ligeramente mayor al anterior el programa se encuentra trabajando y ocupado (en ocasiones con forma de un reloj de arena).
  + Cell.- el cursor celda posee forma de cruz e indica que se puede seleccionar celdas.
  + Crosshair.- cursor cruzado, generalmente para indicar la selección en un mapa de bits
  + Text.- es el conocido cursor que nos permite seleccionar texto.
  + vertical-text.- su forma es similar al cursor de texto pero dispuesto horizontalmente, indica que se puede seleccionar texto vertical.
  + Alias.- este cursor se puede emplear para índica que se debe crear un alias o acceso directo
  + Copy.- este cursor presenta la flecha acompañado de un signo (+) y puede servir para indicar que un elemento puede ser copiado.
  + Move.- posee forma de cruz con flechas, sirve para elementos que pueden ser movidos con el cursor.
  + no-drop.- este tiene la forma de círculo rojo con raja diagonal, indica que no se permite una caída en la ubicación actual
  + not-allowed.- similar al anterior, pero indica que no se puede realizar algo.
  + all-scroll.- indica que se trata de un elemento que se puede desplazar en cualquier dirección.
  + col-resize.- cursor para modificar las dimensiones de las columnas horizontalmente.
  + e-resize.- cursor dimensionar el borde derecho de un cuadro.
  + ew-resize.- cursor para cambiar dimensiones a izquierda o derecha.
  + n-resize.- puntero para indicar que se puede cambiar el tamaño del borde superior de un cuadro.
  + ne-resize.- cursor que indica que es posible cambiar el tamaño de la esquina superior derecha de un cuadro.
  + nesw-resize.- cursor para cambiar dimensiones en las direcciones superior, derecha, inferior o izquierda.
  + ns-resize.- para cambiar dimensión vertical, es decir superior o inferior.
  + nw-resize.- indica que es posible cambiar el tamaño de la esquina superior izquierda de un cuadro.
  + nwse-resize.- cursor para cambiar el tamaño en las direcciones superior izquierda o inferior derecha
  + row-resize.- para modificar dimensiones de filas en dirección vertical.
  + s-resize.- cursor para cambiar tamaño de borde inferior de un cuadro.
  + se-resize.- cursor para cambiar el tamaño de la esquina inferior derecha de un elemento.
  + sw-resize.- cursor para cambiar el tamaño de la esquina inferior izquierda de un elemento.
  + w-resize.- cursor para modificar las dimensiones del borde izquierdo de un cuadro.

## La propiedad display

La propiedad display nos permite elegir como se muestran los elementos en la página, podemos indicar que se muestren como elementos de línea o como elementos de bloque, además podemos indicar que no se muestre.

Por defecto, cada elemento de una página web se comporta como una caja.

* Elementos de línea.- estos ocupan solo lo necesario, por lo cual pueden continuar en una misma linea sin hacer salto de línea; ejemplo de ellos son: enlaces, imágenes, elementos en span, textos sub y super, etc.
* Elemento de bloque.- estos elementos ocupan el ancho máximo disponible, por lo cual realizan salto de línea al presentarse en la estructura de la página; ejemplo de ellos tenemos a: párrafos, bloques div, tablas, formularios, listas, etc.
  + inline.- establece que el elemento debe comportarse como un elemento de línea.
  + block.- indica que el elemento debe comportarse como un elemento de bloque.
  + none.- indica que el elemento no debe ser mostrado.
  + inherit.- hereda del elemento padre.
  + initial.- indica que se cumpla la forma inicial o predeterminada.

## La propiedad visivility

La propiedad visivility de CSS nos permite establecer si un elemento es visible o invisible, de acuerdo al valor que elijamos establecer:

* + visible.- indica que el elemento es mostrado.
  + hidden.- establece que el elemento se esconde pero los demás elementos de la página se comportan como si dicho elemento estuviese presente ahí, respetan sus dimensiones. Es como si el elemento solo fuera invisible.
  + collapse.- pensada para controlar la visibilidad de elementos de tabla, filas y columnas.

## Posicionando elementos con CSS

### Propiedades de posicionamiento CSS

* top.- establece la distancia superior, del elemento respecto a otro.
* left.- establece la distancia por el lado derecho del elemento respecto a otro.
* bottom.- distancia inferior de un elemento respecto a otro.
* right.- distancia por el lado izquierdo de un elemento respecto a otro.

### Propiedad position y formas de posicionamiento

La propiedad position nos permite establecer la forma en la que será posicionado un determinado elemento.

* + estatic.- la forma por defecto, obedece al flujo normal de la página.
  + relative.- establece que la posición de un elemento depende de otro (El elemento existe entre los demas).
  + absolute.- indica que la posición de un elemento no depende de otro (El elemento es como si no existiera para los demás).
  + fixed.- permite fijar un elemento en una posición determinada (posición fija de la PANTALLA, no de la pagina.).
  + inherit.- hereda el estilo de posición del elemento padre.

### Propiedad float

La propiedad float establece que un elemento debe flotar a un lado de otro, ya sea al lado derecho izquierdo del contenedor en el que se encuentre, de tal manera que un elemento quede flotando en un lado con los demás elementos de texto y de linea alrededor de ella como si los estuvieran envolviendo.

* + left.- el elemento flota hacia la izquierda dentro del bloque que lo contiene.
  + right.- indica que el elemento flota a la derecha de su bloque contenedor.
  + none.- indica que el elemento no flota en ningún sentido y es el valor por defecto.

### Propiedad clear

Como ya hemos visto, después de establecer la propiedad float para un elemento determinado, los elementos que siguen suelen envolver a dicho elemento flotante.

La propiedad clear nos permite limpiar o despejar elementos que se encuentran a lado de un elemento flotante.

* + left.- indica que desplace al elemento hasta donde no haya un flotante a la izquierda.
  + right.- indica que desplace al elemento hasta donde no haya un flotante a la derecha.
  + both.- indica que desplace al elemento hasta donde no haya flotantes a la derecha ni a la izquierda.

## Propiedad overflow

Cuando por alguna razón el elemento hijo sobrepasa los bordes de su elemento padre, Overflow permite mantener visible el excedente, ocultar, o crear barras de desplazamiento sobre el contenedor del que excede.

* + visible.- es el valor por defecto y hace que todo sea visible, aunque el contenido esté desbordado.
  + hidden.- hace que el contenido excedente se oculte.
  + scroll.- El contenido ingresa al contenedor sin desbordarse, pero aparecen las barras de desplazamiento horizontal y vertical para visualizar el contenido.
  + auto.- automáticamente aparecen las barras de desplazamiento siempre y cuando sean necesarias, el contenido no se desborda, ingresa en su contenedor.
  + inherit.- hereda el estilo del elemento padre.

## Propiedad Z-Index

Z-index es una propiedad que está pensada para ordenar elementos en el eje Z, prácticamente es el eje que sale del plano de la pantalla; en palabras sencillas, Z-index sirve para poner elementos uno sobre otros.

Cabe recalcar que z-index actúa sobre elementos posicionados, es decir, que posean la propiedad position y además cuyo valor sea distinto de static. z-index no actúa sobre elementos no posicionados.

* + auto.- La caja no establece un nuevo contexto de apilamiento. El nivel de apilamiento de la caja generada es el mismo que el de la caja padre.
  + Entero.- pude ser un número entero.
  + inherit.- hereda del elemento padre.

## Esquinas redondeadas

* #redondeado {border-radius: 20px;} nos permite establecer un radio para redondear uniformemente las esquinas de un elemento.
* Crear un círculo con CSS: Crear un círculo usando la propiedad border-radius es muy fácil, solo debemos asegurar que el elemento que queremos convertir en círculo tenga las dimensiones de sus lados iguales, es decir debe tener la forma de un cuadrado, de tal manera que si establecemos esquinas redondeadas a la mitad de la dimensión de dicho lado, el cuadrado quedará en forma de círculo perfecto.
* #circulo {border-radius: 20px 70px;} el primer valor redondea las esquinas superior izquierdo e inferior derecho, mientras que el segundo valor se aplica a las esquinas superior derecho e inferior izquierdode un elemento.
* #circulo {border-radius: 20px 40px 60px 80px;} superior izquierdo, superior derecho, inferior derecho, inferior izquierdo.
  + superior izquierda: border-top-left-radius: 10px;
  + Superior derecha: border-top-right-radius: 10px;
  + inferior derecha: border-bottom-right-radius: 10px;
  + inferior izquierda: border-bottom-left-radius: 10px;

## Sombras

box-shadow nos permite poner sombra a las cajas con CSS.

* .bloque {box-shadow: 10px 10px 6px 3px #474747;}
  + Sombra X.- específica el tamaño de sombra a lo largo del eje X, se trata de un valor numérico con su respectiva unidad de medida longitudinal, en nuestro caso pueden ser px, em, etc. los valores positivos crean sombras hacia la derecha y valores negativos hacia la izquierda.
  + Sombra en Y.- especifica la dimensión de la sombra a lo largo del eje Y, es decir verticalmente, es un valor numérico con su unidad de medida longitudinal, ya sea px, em, etc. valores positivos crean sombras hacia abajo y valores negativos sombras hacia arriba del elemento.
  + Sombra X.- específica el tamaño de sombra a lo largo del eje X, se trata de un valor numérico con su respectiva unidad de medida longitudinal, en nuestro caso pueden ser px, em, etc. los valores positivos crean sombras hacia la derecha y valores negativos hacia la izquierda.
  + Sombra en Y.- especifica la dimensión de la sombra a lo largo del eje Y, es decir verticalmente, es un valor numérico con su unidad de medida longitudinal, ya sea px, em, etc. valores positivos crean sombras hacia abajo y valores negativos sombras hacia arriba del elemento.
  + Color de sombra.- el color puede especificarse con palabras clave como: red, blue, Green, yellow, etc. también con valores hexadecimales o con los ya conocidos y admitidos.
  + Color de sombra.- el color puede especificarse con palabras clave como: red, blue, Green, yellow, etc. también con valores hexadecimales o con los ya conocidos y admitidos.
* .bloque {box-shadow: inset 10px 10px 6px 3px #474747;} inset aplica sombra interna.

.bloque {box-shadow:

10px 10px 1px 3px #ff2317,

20px 20px 2px 4px #ffe417,

30px 30px 3px 5px #23ff17,

40px 40px 4px 6px #174dff;

} caja con muchas sombras.

## Degradados en fondos

* .lineal {background-image: linear-gradient(red, yellow, blue, green);} Degradado lineal de colores de arriba abajo.
* .lineal {background-image: linear-gradient(red 70%, yellow);} % de abarcado de color.
* Direccion:
  + Left: el degradado se genera de izquierda a derecha, por ejemplo: linear-gradient(left, blue, yellow);
  + Right.- el degradado se genera de derecha a izquierda, por ejemplo: linear-gradient(right, blue, yellow);
  + Top: hará que el degrado se genere de arriba hacia abajo. Ejemplo: linear-gradient(top, blue, yellow);
  + Bottom: hace que el degradado se genere de abajo hacia arriba. Ejemplo: linear-gradient(bottom, blue, yellow);
  + bottom left .- Degradado desde la esquina inferior izquierda.
  + top left.- Degradado desde la esquina superior izquierda.
  + bottom right.- Degradado desde la esquina inferior derecha.
  + top right.- Degradado desde la esquina superior izquierda.
  + También podemos especificar un ángulo que determine la dirección del degradado.
* .lineal {background-image: -webkit-repeating-linear-gradient(left, red 100px, yellow 200px);} Repetición: Es posible establecer la repetición de un determinado degradado, para ello haremos uso de la función repeating-linear-gradient(); es decir simplemente agregamos la palabra repeating antes del degradado.
* .radial {background-image: radial-gradient(red, yellow, blue, green);} Los degradados radiales son aquellos que parten de un centro y se extienden en dirección radial, es decir en todas direcciones. Por defecto es una elipse pero se puede cambiar a un circulo agregando circle antes de los valores de colores. También se puede establecer los % de abarcado de color.
* Posición del degradado:
  + Para desplazar el centro del degradado horizontalmente, es decir a lo largo del eje X, podemos hacer uso de las palabras clave: left (izquierda), center (centro), right (derecha).
  + Para desplazar la posición del centro verticalmente, es decir a lo largo del eje Y podemos emplear las siguientes palabras clave: top (arriba), center (centro), bottom (abajo).
  + bottom left .- Origen en la esquina inferior izquierda.
  + top left.- Se origina en la esquina superior izquierda.
  + bottom right.- Degradado con centro en esquina inferior derecha.
  + top right.- Degradado con centro en la esquina superior izquierda.

## Imágenes de fondo

* div.fondoimg {background-image: url("https://cdn.pixabay.com/photo/2017/02/05/00/19/web-design-2038872\_960\_720.jpg");} nos permite poner imágenes de fondo a un elemento determinado. Es simple de usar, solo declaramos y en su valor colocamos url(""); dentro de las comillas la ubicación del archivo, ya sea local o URL.
* div.fondoimg {background-image: url("https://cdn.pixabay.com/photo/2017/02/05/00/19/web-design-2038872\_960\_720.jpg"); background-size: 450px 200px;} background-size nos permite especificar las dimensiones del fondo de un elemento; ahora tenemos el control para determinar el tamaño de una imagen de fondo, podemos especificar de varias maneras, de acuerdo al valor que designemos.
  + Contain.- hace que el fondo se ajuste al contenedor, si el contenedor es más grande que la imagen de fondo, entonces dicha imagen aumenta su tamaño para ajustarse al contenedor, y en caso contrario, disminuye para ajustarse al contenedor, pero no la imagen siempre se muestra completa (sin recorte), aunque no alcance a cubrir por una diferencia de relación de aspecto; por defecto la imagen se repetirá para cubrir espacios vacíos, pero podemos controlar. Mediante la propiedad background-repeat, cuyo valor puede ser no-repeat o repeat.
  + Cover.- en este caso la imagen cubre el contenedor, para ello se escala y si sus dimensiones no guardan relación con las del contendor, entonces se recortará para poder cubrir (aunque sea recortada), el objetivo de la imagen es cubrir y no mostrarse completa.
* div.fondoimg {background: red; background-clip: content-box;} bacground-clip nos permite especificar hasta donde se extiende el fondo de un elemento, ya sea que estemos trabajando con un color, una imagen, degradado, etc. podemos definir si dicho fondo solo cubre el área de contenido o incluye el área de relleno o extenderse hasta el borde.
  + contend-box.- este valor hace el fondo cubra solo el contenido específicamente,
  + padding-box.- con este valor el fondo cubrirá el contenido más el área de relleno del elemento.
  + border-box.- mediante este valor podemos hacer que el fondo cubra hasta el borde del elemento.
* div.fondoimg {background-image: url("https://cdn.pixabay.com/photo/2020/04/30/15/34/code-5113374\_\_340.jpg"), url("https://cdn.pixabay.com/photo/2018/02/16/10/23/web-3157323\_\_340.jpg"), url("https://cdn.pixabay.com/photo/2017/08/10/08/47/code-2620118\_\_340.jpg"), url("https://cdn.pixabay.com/photo/2018/11/04/00/02/web-page-3793072\_\_340.png"); background-position: left top, right top, bottom left, bottom right; background-size: 200px 150px; background-repeat:no-repeat;} para poner varias imagenes, solo separamos con comas (,). Por ejemplo: background: url(""), url(""), url("");

Una vez especificadas las imágenes, ahora establecemos una dimensión en caso de ser necesario; adicionalmente declaramos las propiedades background-position para establecer posiciones, background-repeat para indicar si las imágenes se repiten o no.

## La propiedad opacity

Nos permite ponerle opacidad a un elemento cualquiera, o si prefieres llamarlo transparencia.

* div.rojo {opacity:0.3;}
* img.transparente {opacity:0.4;}
* p.transparente {opacity: 0.4;}

## pseudoclases y psudoelementos

Las pseudoclases son selectores que básicamente nos permiten apuntar a un elemento específico que cumple una característica determinada.

* div p:first-child {color: red;font-size: 20px;} :first-child podemos establecer un determinado estilo para el primer elemento hijo de un elemento padre.
* div p:last-child {color: red; font-size: 20px;} :last-child nos permite dar estilo al último elemento hijo de un determinado elemento padre.

Los pseudo-elementos están pensados para establecer un estilo a partes específicas de un elemento, por ejemplo: la primera línea de un texto, la primera letra de un texto, etc.

* p::first-line {font-size: 30px; font-weight: bold; color: green; font-style: italic;} ::first-line sirve para establecer un estilo a la primera línea de texto de un elemento seleccionado.
* p::first-letter { font-size: 70px; font-weight: bold; color: green; font-style: italic;} ::first-letter permite establecer un estilo para la primera letra del texto de u elemento de texto.
* p::selection {color: orange; background: #047a0c;} ::selection nos permite dar estilo a un elemento seleccionado por el usuario.

### La propiedad content:

content nos permite establecer el contenido que queremos insertar antes o después de un elemento mediante código CSS; para ello usamos los pseudoelementos ::before o ::after.

* + string: básicamente se trata de un texto entre comillas dobles
  + url: permite establecer una dirección URL
* .ejem::before {content: "➥ Un pedazo de texto:"; color: red; font-size:20px;} ::before permite agregar o insertar un contenido antes de un elemento mediante CSS.
* li::after { content: "🍎"; } ::after sirve para insertar o agregar contenido después de un elemento HTML especificado en CSS mediante el selector correspondiente.

## Transiciones

Una transición es el evento que sucede entre el estado inicial y final, es decir el cambio de estado, dicho cambio puede ser de alguna característica o propiedad de un elemente HTML en nuestro caso, este cambio lo haremos mediante código CSS.

Una transición se ejecuta cuando ocurre un evento determinado, e nuestro caso específico trabajaremos cuando el ratón pasa por encima de un elemento; es decir aprenderemos a aplicar transiciones cuando el puntero del ratón pasa o se detiene sobre un elemento específico.

Recordemos que para determinar un estilo cuando el puntero del ratón pasa sobre un elemento, utilizamos una pseudoclase llamada :hover, por ende estableceremos dentro de ella los estados finales de la transición.

Poner una transición a cualquier elemento es muy simple, mediante el selector apuntaremos al elemento que queremos aplicar la transición, y mediante las propiedades de transición determinamos cuál es la propiedad a cambiar, la duración de dicha transición, su función de tiempo, retraso, etc.

La sintaxis tendría la siguiente forma: selector {propiedades: valores;}

### Propiedades de transición:

Las propiedades de transición, se escriben como cualquier propiedad de CSS con su respectivo valor, a continuación conoceremos cuáles son y los valores que pueden tomar estos.

* **Transition-property:** La propiedad transition-property nos permite establecer la propiedad a ser traicionada, es decir al que se aplica la transición, por lo tanto el valor para la propiedad transition-property es una propiedad CSS.

Los valores que puede tomar transition-property puede ser cualquier propiedad CSS, por ejemplo si queremos que la transición sea del ancho de un elemento, entonces el valor de transition-property sería width (ancho) y la sintaxis quedaría de la siguiente manera: transition-property: width;

Por poner otro ejemplo, si la propiedad a cambiar fuera el color de letra de un elemento de texto, entonces la declaración quedaría de la siguiente manera: transition-property: color;

También es posible establecer varias propiedades, simplemente debemos separarlas con comas (,); por ejemplo si queremos cambiar el ancho y color de fondo seria: transition-property: width, background;

* **Transition-duration:** Mediante la propiedad transition-duration podemos indicar el tiempo de duración de la transición desde que inicia hasta que culmina. Su valor es un valor numérico con su respectiva unidad de tiempo, generalmente suele ser segundos (s) o milisegundos (ms).

Por ejemplo si queremos una duración de 10 segundos, la sintaxis quedaría de la siguiente manera: transition-duration: 10s;

La propiedad transition-duration es necesaria para que se ejecute la transición, su valor por defecto es 0 y si no se establece no ocurrirá ninguna transición.

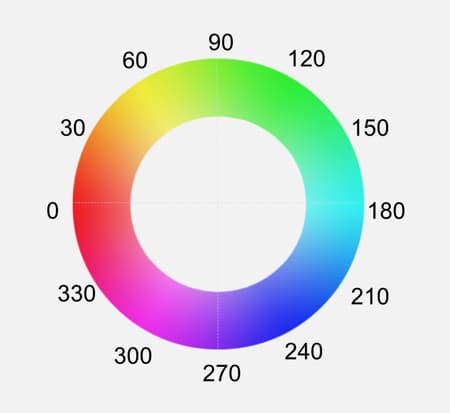
* **Transition-timing-function:** Mediante la propiedad transition-timing-function podemos definir la función de tiempo, básicamente se trata de elegir el ritmo en el tiempo de la transición, podemos elegir entre ritmos uniformes, acelerados, lentos y sus combinaciones.
  + **Ease:** Cuando transition-timing-function toma el valor ease la transición empieza con ritmo lento y luego acelera para culminar.
  + **Ease-in:** El valor ease-in hará que la transición inicie lentamente, luego se torna acelerado y finalmente se detiene espontáneamente.
  + **Ease-out:** Cuando la propiedad transition-timing-function toma el valor ease-out, la transición es rápida al principio, pero luego se desacelera hasta que al final se detiene, es decir comienza rápido y se detiene lentamente.
  + **Ease-in-out:** El valor ease-in-out hará que la transición sea al principio lento y luego rápida, sin embargo no es igual al resultado producido por ease, ya que ease-in-out produce una transición más sutil.
  + **Linear:** Cuando transition-timing-function toma el valor linear, la transición sucede con ritmo o velocidad constante, por lo tanto la transición es uniforme en el tiempo.
  + **Cubic-bezier ():** El valor cubic-bezier() nos permite crear un ritmo de transición personalizado, mediante cuatro valores numéricos que van del 0 al 1 y separadas por comas (,)

**Transition-delay:** La propiedad transition-delay permite definir el retraso de la transición, básicamente se trata del tiempo que la ejecución de la transición se retrasa antes de empezar. Es decir, en caso queremos que la transición se inicie en un tiempo específico, podemos establecer dicho momento en segundos (s) o también milisegundos (ms). La declaración quedaría de la siguiente forma: transition-delay: 3s;

## Filtros

Colocar un filtro es muy sencillo, simplemente apuntamos al elemento y definimos la propiedad filter, en este caso también estableceremos una función de filtro.

* #gatito {width:400px; filter: grayscale(100%);} nos permite aplicar un filtro de escala de grises a una imagen.
* #gatito {width:400px; filter: invert(100%);} se utiliza para invertir los colores de una imagen en cierta medida controlada.
* #gatito {width:400px; filter: sepia(100%);} Este tipo de filtro es bastante conocido, se trata de un color marrón- rojizo de débil saturación.
* #gatito {width:400px; filter: hue-rotate(120deg);} nos permite rotar el tono de color de una imagen.



* #gatito {width:400px; filter: saturate(30%);} controlar la saturación de colores de una imagen.
* #gatito {width:400px; filter: contrast(150%);} controlar el contraste de los colores de la imagen.
* #gatito {width:400px; filter: opacity(50%);} controlar la opacidad de un elemento como una imagen.
* #gatito {width:400px; filter: brightness(40%);} permite ajustar el brillo de una imagen.
* #gatito {width:400px; filter: blur(4px);} permite manejar el desenfoque de la imagen.
* #gatito {width:400px; filter: drop-shadow(7px 10px 2px green);} nos permite crear sombras alrededor de una imagen. Ancho, altura, desenfoque y color.
* #gatito {width:400px; filter: drop-shadow(10px 8px 2px blue) blur(1px) grayscale(50%);} Con una sola declaración de filtro implementaremos varias funciones.

# JavaScript

## Basico

* document.write(‘hola mundo’); Escribir en el html.
* Var nombre de variable= valor de variable; JS es débilmente tipado.
* Nombre de variable = prompt(); input de la variable.
* var dia, mes, anio; se pueden declarar varias variables en una misma línea.
* var edad=20; variable inicializada.
* nombre=prompt('Ingrese su nombre:',''); Entrada de datos por teclado. <variable que recibe el dato>=prompt(<mensaje a mostrar en la ventana>,<valor inicial a mostrar en la ventana>);
* ParseInt() ; parseo a entero. Se puede hacer antes del prompt(): parseInt(prompt('Ingrese numero:',''));
* CONDICIONALES IGUAL A JAVA.

if (<condición> &&<condición> )

{

<Instruccion(es)>

}

else if (<condición> ||<condición>)

{

<Instruccion(es)>

}

else

{

<Instruccion(es)>

}

* OPERADOR CONDICIONAL ?:

El operador condicional ?: remplaza en algunos casos a estructuras if/else sencillas con el objetivo de escribir un código más compacto o corto.

**If/else:**

var mayor;

if (v1>v2)

mayor=v1;

else

mayor=v2;

document.write('El mayor de los dos valores es:'+mayor);

**?:**

var mayor;

v1>v2 ? mayor=v1 : mayor=v2;

document.write('El mayor de los dos valores es:'+mayor);

* SWITCH IGUAL A JAVA

switch (valor) {

case 1: document.write('uno');

break;

case 2: document.write('dos');

break;

case 3: document.write('tres');

break;

case 4: document.write('cuatro');

break;

case 5: document.write('cinco');

break;

default:document.write('debe ingresar un valor comprendido entre 1 y 5.');

* WHILE IGUAL A JAVA

var x;

x=1;

while (x<=100)

{

document.write(x);

document.write('<br>');

x++;

}

* DO WHILE IGUAL A JAVA

do {

valor=prompt('Ingrese un valor entre 0 y 999:','');

valor=parseInt(valor);

document.write('El valor '+valor+' tiene ');

if (valor<10)

{

document.write('Tiene 1 digitos');

}

else

{

if (valor<100)

{

document.write('Tiene 2 digitos');

}

else

{

document.write('Tiene 3 digitos');

}

}

document.write('<br>');

} while(valor!=0);

* FOR IGUAL A JAVA

for (<Inicialización> ; <Condición> ; <Incremento o Decremento>)

{

<Instrucciones>

}

* FUNCIONES:

function <nombre de función>(argumento1, argumento2, ..., argumento n)

{

<código de la función>

<return>

}

* alert(): mensaje del explorador
* var nombre=document.getElementById('nombre').value; recibir elementos del campos html
* OPERADORES DE IGUALDAD Y DESIGUALDAD:

El operador == primero hace la conversión a un mismo tipo de dato para verificar si son iguales, en cambio el operador === llamado operador de igualdad estricta compara los valores sin hacer conversiones. Esto hace que cuando utilizamos el operador === retorne siempre false en caso que las variables que comparamos sean de distinto tipo.

El siguiente if se verifica verdadero ya que con el operador == primero se transforman los dos valores al mismo tipo de datos previos a verificar su igualdad:

var v1='55';

if (v1==55)

document.write('son iguales');

En cambio el siguiente if utilizando el operador de igualdad estricta se verifica false ya que son variables o valores de distinto tipo:

var v1='55';

if (v1===55)

document.write('son iguales');

Los operadores != y !== son los opuestos de == y ===.

El operador != retorna true si son distintos previo conversión a un mismo tipo de dato y el operador !== retorna true si los valores son distintos sin hacer una conversión previa, teniendo en cuenta que para tipos de datos distintos es true el valor retornado.

El siguiente if se verifica false:

var v1='55';

if (v1!=55)

document.write('son distintos');

En cambio utilizando el operador !== se verifica verdadero ya que son tipos de datos desiguales:

var v1='55';

if (v1!==55)

document.write('son distintos');

El operador de igualdad estricto === analiza los operandos de la siguiente forma:

Si los dos valores son de diferente tipo luego el resultado es false.

Si los dos valores almacen null luego el resultado es true. Lo mismo si los dos valores almacenan undefined.

Si uno de los valores almacena el valor NaN luego la condición se verifica como false.

El operador de igualdad == analiza los operandos de la siguiente forma:

Si los dos operandos no son del mismo tipo el operador == primero verifica si uno de los valores es null y el otro undefined luego retorna true, si uno es un número y el otro un string convierte a string el número y luego compara, si uno de los valores es true convierte este a 1 y luego compara, si uno de los valores es false convierte este a 0 (cero) y luego compara.

Algunas comparaciones:

if (true==1) //verdadero

if (true===1) //falso

if (false==0) //verdadero

if (false===0) //falso

## Clase Date

Esta clase nos permitirá manejar fechas y horas. Se invoca así:

* fecha = new Date();//creación de un objeto de la clase Date
* fecha = new Date(año, mes, dia);
* fecha = new Date(año, mes, dia, hora, minuto, segundo);

El objeto Date dispone, entre otros, de los siguientes métodos:

* getFullYear()
* setFullYear(año)
* getMonth()
* setMonth(mes)
* getDate()
* setDate(dia)
* getHours()
* setHours(horas)
* getMinutes()
* setMinutes(minutos)
* getSeconds()
* setSeconds(segundos)
* getDay()

## Clase array

* Definir vector:

var sueldos;

sueldos=new Array(5);

* Cargar:

var f;

for(f=0;f<sueldos.length;f++)

{

var v;

v=prompt('Ingrese sueldo:','');

sueldos[f]=parseInt(v);

}

* Leer:

for(f=0;f<sueldos.length;f++)

{

document.write(sueldos[f]+'<br>');

}

### Diferentes formas de crear Arrays:

Tenemos muchas formas de inicializar un array en Javascript según nuestra situación particular, veamos con ejemplos diferentes formas:

* Creación de un array sin elementos:

var vector1=new Array();

* Otra sintaxis para crear un array sin elementos:

var vector2=[];

* Creación de un array indicando la cantidad de componentes iniciales que podrá almacenar:

var vector3=new Array(5);

* Creación e inicialización llamando al constructor Array y pasando como parámetros los valores a almacenar en las componentes:

var vector4=new Array(1,70,'juan');

* Creación e inicialización de un array utilizando los corchetes:

var vector5=[1,70,'juan'];

### Arrays densos:

Todos los vectores donde almacenamos en forma contigua todos sus elementos son llamados array densos.

Almacenar elementos siguiendo esta premisa tiene como ventaja poder acceder a sus elementos mediante una estructura repetitiva disponiendo el contador del for como subíndice.

Una propiedad de suma utilidad trabajando los vectores sin dejar subíndices sin utilizar es la propiedad length. La propiedad length almacena la cantidad de componentes que tiene el vector.

<script>

var vec=[];

var valor;

var indice=0;

do {

valor=prompt('Ingrese un valor entero (0 para finalizar)','');

valor=parseInt(valor);

if (valor!=0)

{

vec[indice]=valor;

indice++;

}

} while (valor!=0);

var f;

var suma=0;

for(f=0;f<vec.length;f++)

{

suma=suma+vec[f];

}

document.write('Se ingresaron '+vec.length+' valores<br>');

for(f=0;f<vec.length;f++)

{

document.write(vec[f]+'<br>');

}

document.write('La suma de los valores ingresados es:'+suma);

</script>

### Arrays no densos o dispersos:

Javascript permite crear array e inicializar componentes no contiguas.

Por ejemplo podemos crear un vector e inicializar las componentes con subíndice 5 y 10:

* var vec=[];
* vec[5]=100;
* vec[10]=200;

Este tipo de array se los llama array dispersos ya que no tenemos todas las componentes contiguas ocupadas.

<script>

var vec=[];

vec[5]=100;

vec[10]=200;

document.write('Atributo length:'+vec.length+'<br>');

var f;

for(f=0;f<vec.length;f++)

{

document.write(vec[f]+'<br>');

}

</script>

Si ejecutamos este programa la salida en la página será:

Atributo length: 11

undefined

undefined

undefined

undefined

undefined

100

undefined

undefined

undefined

undefined

200

### Metodos push y pop:

Push añade el valor al final del vector:

var vec=[];

vec.push(10);

vec.push(20);

document.write(vec.length); //imprime 2

Automáticamente cuando llamamos al método push el valor que le pasamos en el parámetro se almacena en el vector y se incrementa el atributo length.

Podemos inclusive llamar al método push pasando más de 1 parámetro:

var vec=[];

vec.push(10,20);

document.write(vec.length); //imprime 2

El método inverso llamado pop extrae el último elemento del Array y decrementa en uno el atributo length:

var vec=[];

vec.push(10,20,30,40);

document.write(vec.length+'<br>'); //imprime 4

vec.pop();

document.write(vec.length+'<br>'); //imprime 3

document.write(vec.pop()+'<br>'); //imprime un 30

document.write(vec.length+'<br>'); //imprime 2

El método pop() además de eliminar el último elemento del vector retorna el valor almacenado en dicha componente.

Si llamamos al método pop y el vector está vacío retorna el valor undefined.

### Métodos unshift y shift:

Así como el método push inserta un elemento al final del vector el método unshift inserta un elemento al principio del vector y desplaza el resto una posición.

El método shift extrae el primer elemento del vector y desplaza hacia delante el resto de elementos del vector.

### Instrucción delete:

Javascript tiene una instrucción que permite eliminar un elemento de un vector llamada delete.

Al comando delete le pasamos el nombre del vector y la posición que queremos borrar:

delete vec[3];

Con el comando anterior estamos eliminando la componente de la posición 3 del vector. Cuando se elimina una componente no se modifica la propiedad length y el vector se convierte en no denso. Si luego tratamos de acceder a dicha posición el resultado es el valor undefined.

### Métodos sort y reverse:

Otro método muy útil de la clase Array es sort. La sintaxis más sencilla y por defecto es para ordenar una lista de string:

<script>

var nombres=['marcos','ana','luis','jorge','carlos'];

var f;

document.write('Vector antes de ordenarlo<br>');

for(f=0;f<nombres.length;f++)

{

document.write(nombres[f]+'<br>');

}

nombres.sort();

document.write('Vector después de ordenarlo<br>');

for(f=0;f<nombres.length;f++)

{

document.write(nombres[f]+'<br>');

}

</script>

Para ordenar una lista de enteros se complica el algoritmo ya que debemos pasar al método sort una función anónima indicando como implementar la comparación entre elementos:

<script>

var vec=[100,5,60,3,90];

var f;

document.write('Vector antes de ordenarlo<br>');

for(f=0;f<vec.length;f++)

{

document.write(vec[f]+'<br>');

}

vec.sort(function(v1,v2) {

if (v1>v2)

return 1;

else

return -1;

});

document.write('Vector después de ordenarlo<br>');

for(f=0;f<vec.length;f++)

{

document.write(vec[f]+'<br>');

}

</script>

Como vemos al método sort hemos pasado una función que retorna un 1 o -1:

vec.sort(function(v1,v2) {

if (v1>v2)

return 1;

else

return -1;

});

Los parámetros v1 y v2 son los elementos que se comparan:

Si el contenido de v1 es mayor que v2, se sitúa v2 en un indice menor que v1 (debe retornar un valor menor a cero)

Si el contenido de v1 es menor que v2, se sitúa v1 en un indice menor que v2 (debe retornar un valor mayor a cero)

El siguiente método a analizar en éste concepto es reverse(), como podemos intuir este método invierte el orden de los elementos del vector. Nos puede ser de utilidad si tenemos ordenado un vector en orden ascendente y lo queremos en forma descendente.

### Método splice:

El método splice permite eliminar componentes, insertar componentes o eliminar e insertar componentes en forma simultánea.

#### Borrar:

Para borrar elementos debemos indicar dos parámetros al método splice, el primero indica a partir de qué posición procedemos a borrar componentes y el segundo parámetro indica la cantidad de componentes a borrar.

<script>

var vec=[0,1,2,3,4,5,6,7,8,9];

document.write('Vector inicial<br>');

var f;

for(f=0;f<vec.length;f++)

{

document.write(vec[f]+'<br>');

}

vec.splice(1,3);

document.write('Vector luego de borrar<br>');

var f;

for(f=0;f<vec.length;f++)

{

document.write(vec[f]+'<br>');

}

</script>

Otra cosa que hay que tener en cuenta que el método splice retorna un vector con los elementos borrados, luego si queremos podemos guardar esa referencia:

<script>

var vec=[0,1,2,3,4,5,6,7,8,9];

document.write('Vector inicial<br>');

var f;

for(f=0;f<vec.length;f++)

{

document.write(vec[f]+'<br>');

}

var vec2=vec.splice(1,3);

document.write('Elementos extraídos<br>');

var f;

for(f=0;f<vec2.length;f++)

{

document.write(vec2[f]+'<br>');

}

</script>

Podemos indicar un valor negativo y el posicionamiento se cuenta desde el final del vector, por ejemplo si queremos eliminar los dos últimos elementos del vector luego debemos codificar:

<script>

var vec=[0,1,2,3,4,5,6,7,8,9];

document.write('Vector inicial<br>');

var f;

for(f=0;f<vec.length;f++)

{

document.write(vec[f]+'<br>');

}

vec.splice(-2,2);

document.write('Eliminados los dos últimos elementos<br>');

var f;

for(f=0;f<vec.length;f++)

{

document.write(vec[f]+'<br>');

}

</script>

#### Insertar:

Debemos pasar como mínimo 3 o más parámetros. Los dos primeros cumplen la misma función que cuando borramos (pero en este caso pasamos un cero en el segundo parámetro) y del tercer parámetro en adelante se indican los valores a insertar en el vector.

<script>

var vec=[0,1,2,3,4,5,6,7,8,9];

document.write('Vector inicial<br>');

var f;

for(f=0;f<vec.length;f++)

{

document.write(vec[f]+'<br>');

}

vec.splice(1,0,10,20,30,40);

document.write('Vector luego insertar 4 valores<br>');

var f;

for(f=0;f<vec.length;f++)

{

document.write(vec[f]+'<br>');

}

</script>

#### Borrar e insertar:

La tercer posibilidad que nos presenta este método es en el mismo momento que borramos elementos procedemos a insertar otros.

<script>

var vec=[0,1,2,3,4,5,6,7,8,9];

document.write('Vector inicial<br>');

var f;

for(f=0;f<vec.length;f++)

{

document.write(vec[f]+'<br>');

}

vec.splice(1,3,10,20,30,40,50);

document.write('Vector luego de borrar 3 elementos e insertar otros 5 valores<br>');

var f;

for(f=0;f<vec.length;f++)

{

document.write(vec[f]+'<br>');

}

</script>

### Método slice:

El método slice (traducido al castellano: trozo) retorna un trozo del vector. La sintaxis es:

vec.slice(2,4);

El primer parámetro indica a partir de que elementos debe retornar y el segundo parámetro hasta cual elemento (sin incluir dicha posición)

El método slice no modifica el vector original, sino que retorna otro vector con las componentes indicadas.

<script>

var vec=[10,20,30,40,50,60];

document.write('Vector original<br>');

var f;

for(f=0;f<vec.length;f++)

{

document.write(vec[f]+'-');

}

document.write('<br>');

var vec2=vec.slice(1,4);

document.write('Vector resultante de la llamada a slice<br>');

for(f=0;f<vec2.length;f++)

{

document.write(vec2[f]+'-');

}

</script>

Otras variantes que podemos emplear con el método slice:

* Si queremos extraer desde una determinada posición hasta el final del vector debemos llamar al método slice con un solo parámetro:

var vec=[10,20,30,40,50,60];

var vec2=vec.slice(2); // vec2 almacena [30,40,50,60]

* Si queremos indicar una posición pero comenzando desde el final del vector debemos indicar un valor negativo en el primer parámetro:

var vec=[10,20,30,40,50,60];

var vec2=vec.slice(-2); // vec2 almacena [50,60]

* También podemos indicar el segundo parámetro:

var vec=[10,20,30,40,50,60];

var vec2=vec.slice(-3,-1); // vec2 almacena [40,50]

* Por último podemos indicar el primer parámetro un valor positivo y el segundo un valor negativo indicando una posición desde el final

var vec=[10,20,30,40,50,60];

var vec2=vec.slice(1,-1); // vec2 almacena [20,30,40,50]

### Método Join:

El método join concatena todos los elementos del vector en un único string.

Al método join le pasamos como parámetro el o los caracteres de separación que debe agregar entre los elementos dentro del string que se genera.

<script>

var vec=[10,20,30,40,50];

var cadena=vec.join('-');

document.write(cadena);

</script>

Podemos disponer más de un caracter de separación:

var vec=[10,20,30,40,50];

var cadena=vec.join('---');

document.write(cadena); //10---20---30---40---50

Podemos llamar al método join inclusive sin parámetros, en dicho caso se agrega por defecto el caracter de la coma:

var vec=[10,20,30,40,50];

var cadena=vec.join();

document.write(cadena); //10,20,30,40,50

### Método concat:

El método concat crea un nuevo vector con los datos del objeto original y los datos que le enviamos como parámetro.

<script>

var vec=[10,20,30,40];

var vecnuevo=vec.concat(1,2,3);

document.write('Vector origen<br>');

document.write(vec.join()+'<br>');

document.write('Vector generado<br>');

document.write(vecnuevo.join()+'<br>');

</script>

Otra variante es que podemos pasar como parámetros al metodo concat otros vectores.

<script>

var vec1=[10,20,30,40];

var vec2=[100,200,300,400];

var vecsuma=vec1.concat(vec2);

document.write('Primer vector:');

document.write(vec1.join()+'<br>');

document.write('Segundo vector:');

document.write(vec2.join()+'<br>');

document.write('vectores concatenados:');

document.write(vecsuma.join()+'<br>');

</script>

### Método toString :

El método toString del objeto Array retorna un string con todos los elementos del vector separados por coma.

### Distintas formas de recorrer Arrays:

* Para visitar todas las componentes de un vector hemos visto hasta ahora el ciclo repetitivo for:

<script>

var vec=[10,20,30,40,50];

var f;

for(f=0;f<vec.length;f++)

{

document.write(vec[f]+'-');

}

</script>

* Javascript cuenta con otra variante de for(llamada for in):

<script>

var vec=[10,20,30,40,50];

for(var indice in vec)

{

document.write(indice);

document.write(vec[indice]+'-');

}

</script>

El for se repite tantas veces como elementos tiene el vector y en cada vuelta del ciclo for la variable indice almacena el subíndice de la componente (tengamos en cuenta que en este ejemplo definimos el vector con los valores 10,20,30,40,50 y se almacenaron en los subíndices 0,1,2,3,4) es decir la variable indice almacenará en la primer vuelta el valor 0, en la segunda el valor 1 y así sucesivamente:

for(var indice in vec)

{

document.write(indice);

document.write(vec[indice]+'-');

}

* Es más común utilizar el método 'forEach' de la clase Array para recorrer un arreglo en JavaScript:

<script>

var vec = [10, 20, 30, 40, 50];

vec.forEach(function (componente, indice) {

document.write('Posición:' + indice + ' = ');

document.write(componente + '<br>');

})

</script>

Como vemos llamamos al método 'forEach' y le pasamos una función anónima que contiene dos parámetros que recibirán el valor almacenado en la componente y el índice que se está procesando:

vec.forEach(function (componente, indice) {

document.write('Posición:' + indice + ' = ');

document.write(componente + '<br>');

})

## Clase Math

|  |  |  |  |
| --- | --- | --- | --- |
| **Método** | **Descripción** | **Expresión de ejemplo** | **Resultado** |
| abs | Valor absoluto | Math.abs(-2) | 2 |
| sin, cos, tan | Funciones trigonométricas, reciben el argumento en radianes | Math.cos(Math.PI) | -1 |
| asin, acos, atan | Funciones trigonométricas inversas | Math.asin(1) | 1.57 |
| exp, log | Exponenciación y logaritmo, base E | Math.log(Math.E) | 1 |
| ceil | Devuelve el entero más pequeño mayor o igual al argumento | Math.ceil(-2.7) | -2 |
| floor | Devuelve el entero más grande menor o igual al argumento | Math.floor(-2.7) | -3 |
| round | Devuelve el entero más cercano o igual al argumento | Math.round(-2.7) | -3 |
| min, max | Devuelve el menor (o mayor) de sus dos argumentos | Math.min(2,4) | 2 |
| pow | Exponenciación, siendo el primer argumento la base y el segundo el exponente | Math.pow(2,3) | 8 |
| sqrt | Raíz cuadrada | Math.sqrt(25) | 5 |
| random | Genera un valor aleatorio comprendido entre 0 y 1. | Math.random() | Math.random() |

## Clase String

* CONCATENACIÓN DE CADENAS IGUAL A JAVA (+): var final='La entrada tiene ' + contador + ' caracteres.';
* PROPIEDAD LENGTH IGUAL A JAVA: Retorna la cantidad de caracteres de un objeto String.

var nom='Juan';

document.write(nom.length);

* MÉTODOS IGUALES A JAVA
  + CHARAT(POS): Retorna el carácter del índice especificado. Comienzan a numerarse de la posición cero.

var nombre='juan';

var caracterPrimero=nombre.charAt(0);

* + SUBSTRING (POSINICIAL, POSFINAL): Retorna un String extraída de otro, desde el carácter 'posinicial' hasta el 'posfinal'-1:

cadena3=cadena1.substring(2,5);

* + INDEXOF (SUBCADENA): Devuelve la posición de la subcadena dentro de la cadena, o -1 en caso de no estar.

var nombre='Rodriguez Pablo';

var pos=nombre.indexOf('Pablo');

if (pos!=-1)

document.write ('Está el nombre Pablo en la variable nombre');

* + TOUPPERCASE(): Convierte todos los caracteres del String que invoca el método a mayúsculas:

cadena1=cadena1.toUpperCase();

* + TOLOWERCASE(): Convierte todos los caracteres del String que invoca el método a minúsculas:

cadena1=cadena1.toLowerCase();

### Expresiones regulares:

Una expresión regular es una técnica que nos permite identificar si un string coincide con un patrón determinado. Un ejemplo clásico es determinar si un email está correctamente tipeado (sabemos que comienza con uno o más caracteres alfanuméricos, luego el carácter @ seguido de uno o más caracteres alfanuméricos, luego el carácter '.' y finalmente otros caracteres alfabéticos)

Sin conocer las expresiones regulares se pueden resolver estos tipos de problemas mediante una serie de if que procesan partes del string para comprobar si cumplen en conjunto todas las reglas que necesitamos validar, pero con las expresiones regulares podemos resolver este tipo de problemas con una sintaxis mucho más compacta.

Existe toda una serie de reglas para crear las expresiones regulares que iremos viendo.

El lenguaje JavaScript administra las expresiones regulares mediante el objeto RegExp.

<script>

var valor=prompt('Ingrese un numero entero positivo de 3 dígitos','');

var patron=new RegExp('^[0-9]{3}$');

if (patron.test(valor))

alert('Se ingresó un valor entero positivo de 3 dígitos');

else

alert('No se ingresó un valor entero positivo de 3 dígitos');

</script>

En este primer paso no nos importa la sintaxis dispuesta en el string que le pasamos al constructor del objeto RegExp, sino ver la sintaxis de JavaScript para crear un objeto de RegExp y seguidamente como llamamos al método test que retorna si el string 'valor' contrastado con la expresión regular definida en el objeto 'patron' verifica verdadero o falso:

var patron=new RegExp('^[0-9]{3}$');

if (patron.test(valor))

alert('Se ingresó un valor entero positivo de 3 dígitos');

else

alert('No se ingresó un valor entero positivo de 3 dígitos');

Veremos luego que para definir expresiones regulares intervienen una serie de caracteres ^ $. \* +? =! : | \ / () [] {} que tienen un significado especial en la definición del lenguaje de expresiones regulares, estos tipos de caracteres suelen llamarse metacaracteres.

* var patron=/hora/;

Con esta definición especificamos que el patrón a comparar son los caracteres 'hora'. Luego al llamar al método test donde le pasamos el string a comparar verifica si el patrón 'hora' está contenido en alguna posición del string 'oración'.

* Los dos primeros caracteres con un significado especial en una expresión regular son: ^ principio de cadena y $ fin de cadena.

Cuando queremos controlar si un string comienza con un determinado carácter o conjunto de caracteres podemos implementar una expresión regular antecediendo al o los caracteres el símbolo ^.

* Cuando queremos verificar si una cadena finaliza con un determinado carácter o conjunto de caracteres debemos disponer al final el carácter $ y antecederle el o los caracteres a verificar.
* Se emplea [] estos caracteres para encerrar los caracteres permitidos.

Debemos especificar cada uno de los caracteres permitidos encerrados entre los corchetes. Cuando los caracteres son seguidos en la tabla de caracteres podemos utilizar el carácter '-' para indicar un rango de caracteres:

var patron=/^[a-zA-Z]/;

Como solo debemos verificar el primer carácter utilizamos el comando '^' y le sigue los caracteres posibles.

* Ahora veremos que podemos inicializar luego que indicamos entre corchetes el patrón a buscar la cantidad de veces que puede repetirse este patrón.

Son útiles para controlar la cantidad de repeticiones del patrón definido.

Se especifican luego del corchete donde definimos los caracteres permitidos:

Verificar si contiene entre 3 y 6 dígitos:

var patron=/[0-9]{3,6}/;

Verificar si contiene 3:

var patron=/[0-9]{3}/;

Verificar si contiene 3 o más dígitos:

var patron=/[0-9]{3,}/;

* ? busca cero o una ocurrencia en el string

Remplaza a {0,1}

* \* busca cero o más ocurrencias en el string

Remplaza a {0,}

* + busca una o más ocurrencias en el string

Remplaza a {1,}

* \d = [0-9]
* \D = [^0-9]
* \w = [a-zA-Z0-9\_]
* \W = [^a-zA-Z0-9\_]
* \s = espacio en blanco
* alternancia |

Ejemplos de sintaxis para alternancia entre opciones:

[uno|dos|tres]

[a|e]

[hombre|mujer]

<script>

var factura=prompt('Ingrese tipo y nro de factura:','');

var patron=/^[a|b|c]\d+$/;

if (patron.test(factura))

document.write('Se ingresó un tipo y nro de factura correcta');

else

document.write('Se ingresó un tipo y nro de factura incorrecta');

</script>

La expresión regular queda definida como:

var patron=/^[a|b|c]\d+$/;

Es decir debe comenzar con un carácter a,b o c, seguido de uno dígitos.

* metacarácter punto:

El metacarácter punto hace coincidir con cualquier carácter menos el salto de línea.

Por ejemplo si queremos validar el ingreso de cualquier cadena de 5 caracteres que comience con a y finalice con a podemos plantear la siguiente expresión regular:

var patron=/^a...a$/;

Estamos indicando que la cadena debe comenzar con una letra a luego hemos dispuesto tres metacaracteres punto donde se verifican verdadero cualquiera sea el carácter del string y finalmente indicamos que finaliza con la letra a.

<script>

var cadena=prompt('Ingrese una cadena de cinco caracteres que comience y finalice con a','');

var patron=/^a...a$/;

if (patron.test(cadena))

document.write('cadena correcta');

else

document.write('cadena incorrecta');

</script>

Podemos utilizar un cuantificador para no repetir tres veces el metacarácter punto:

var patron=/^a.{3}a$/;

Es importante recordar que siempre que necesitemos utilizar alguno de los metacaracteres como dato a buscar es necesario escaparlo con la barra invertida, por ejemplo si queremos verificar si un número tiene el formato 999.99

<script>

var nro=prompt('Ingrese un número con el formato 999.99','');

var patron=/^\d{3}\.\d{2}$/;

if (patron.test(nro))

document.write('nro correcto');

else

document.write('nro incorrecto');

</script>

* agrupación en subpatrones ()

Los paréntesis abiertos y cerrados pueden tener varios significados en las expresiones regulares. El fundamental es el agrupar una parte de la expresión regular y aplicarle operaciones como |, \*, +, ? etc.

<script>

var hora=prompt('ingrese una hora con el formato hh:mm:ss','');

var patron=/^(0[1-9]|1\d|2[0-3]):([0-5]\d):([0-5]\d)$/;

if (patron.test(hora))

alert('Correcto');

else

alert('Incorrecto');

</script>

Analicemos un poco como hemos agrupado cada uno de las secciones de la hora:

var patron=/^(0[1-9]|1\d|2[0-3]):([0-5]\d):([0-5]\d)$/;

La parte inicial puede tomar un valor entre 0 y 23:

(0[1-9]|1\d|2[0-3])

Utilizamos el metacarácter | para permitir ingresar el cero seguido de un valor entre 1 y 9, o el valor 1 y cualquier dígito, o el dos seguido del dígito 0,1,2, o 3.

Luego debe ingresarse el caracter dos puntos.

La parte de los segundos queda definida por el patrón ([0-5]\d) es decir un valor entre 0 y 5 seguido de cualquier dígito.

Finalmente los segundos es similar a los minutos: ([0-5]\d)

* métodos test y exec del objeto RegExp:

Hasta ahora siempre hemos empleado el método test que retorna un valor true si el string cumple el patrón propuesto o false en caso contrario.

La clase RegExp cuenta con otro método llamado exec que retorna un vector con más datos en caso que el string cumpla el patrón propuesto o null si no cumple el patrón.

<script>

var oracion='saliendo de la casa que esta en la montaña';

var patron1=/la/;

var vector1=patron1.exec(oracion);

document.write(vector1.index);

document.write('<br>');

document.write(vector1.input);

document.write('<br>');

var patron2=/lax/;

var vector2=patron2.exec(oracion);

document.write(vector2);

</script>

Hemos definido en la variable oracion el valor : 'saliendo de la casa que esta en la montaña'. El primer patrón de búsqueda es verificar si contiene los caracteres 'la' la oración:

var oracion='saliendo de la casa que esta en la montaña';

var patron1=/la/;

var vector1=patron1.exec(oracion);

El método exec retornará un valor distinto a null (tener en cuenta que retorna un vector) ya que la palabra 'la' está contenida en la oración.

El vector devuelto por el método exec tiene dos propiedades llamadas index e input:

La propiedad index almacena la posición dentro del string donde se cumple el patrón, en nuestro ejemplo retorna un 12 que es la posición donde se encuentra la l:

saliendo de la casa que esta en la montaña

La propiedad input almacena la cadena que se está analizando.

La segundo parte del problema define un segundo patrón:

var patron2=/lax/;

var vector2=patron2.exec(oracion);

Como no se encuentra en el string la cadena 'lax' el método exec retorna un valor null, es decir no retorna un vector.

* modificadores i (insensitive) g (global) m (multiline)

El objeto RegExp se le puede aplicar tres modificadores que actúan sobre el patrón completo.

* i: especifica que la búsqueda se realiza sin diferenciar entre mayúsculas y minúsculas.
* g: indica que los caracteres a buscar pueden ser encontrados varias veces en la cadena de caracteres.
* m: el modificador m realiza coincidencia de patrones en el modo multilínea. En este modo, si la cadena a buscar contiene varias líneas, el metacaracter ^ y $ deben coincidir con el inicio y final de una línea, además de que coincida con el comienzo y el final de una cadena.

Estos modificadores pueden aplicarse en forma independiente o conjunta a una expresión regular.

* Métodos search, replace, split, match del objeto String.

El objeto String tiene métodos que reciben como parámetro una expresión regular.

El primer método es el search. Este método busca la primer ocurrencia del string según la expresión que le hemos pasado como parámetro. Devuelve un entero indicando la posición donde comienza la cadena encontrada o un -1 si no encuentra el patrón en el string.

<script>

var oracion='La noche está muy oscura y estrellada.';

if (oracion.search(/estrella/)!=-1)

alert('el string estrella está contenido en la variable oracion');

</script>

El segundo método del objeto String que veremos es el replace.

El método replace realiza una búsqueda y reemplazo. Se toma una expresión regular como primer parámetro y un string de reemplazo como su segundo argumento. Si la expresión regular tiene la bandera g, el método replace sustituye todas las coincidencias en la cadena de reemplazo, en caso contrario, se reemplaza sólo la primera coincidencia que encuentre.

<script>

var oracion='uno dos tres uno dos tres uno dos tres';

var cadena=oracion.replace(/uno/g,'1');

alert(cadena); //1 dos tres 1 dos tres 1 dos tres

</script>

El tercer método es el split que tiene por objetivo subdividir un string y retornar un vector con dichas partes. Para la división toma en cuenta la expresión regular pasada como parámetro.

<script>

var cadena='(2,3)-(4,4)-(9,3)';

var vec=cadena.split(/-/);

for(var x=0;x<vec.length;x++)

{

document.write(vec[x]+'<br>');

}

</script>

El último método a analizar del objeto String es el match. Tiene como parámetro una expresión regular y retorna un vector con todos los string que cumplen el patrón especificado (siempre y cuando hallamos activado la bandera 'g' (global))

<script>

var cadena='juan 10 luis 43 pedro 21 carlos 33';

var vec=cadena.match(/\d+/g);

for(var x=0;x<vec.length;x++)

{

document.write(vec[x]+'<br>');

}

</script>

## Formularios y Eventos

El uso de Javascript en los formularios HTML se hace fundamentalmente con el objetivo de validar los datos ingresados. Se hace esta actividad en el cliente (navegador) para desligar de esta actividad al servidor que recibirá los datos ingresados por el usuario.

Esta posibilidad de hacer pequeños programas que se ejecutan en el navegador, evitan intercambios innecesarios entre el cliente y el servidor (navegador y sitio web).

El navegador crea un objeto por cada control visual que aparece dentro de la página. Nosotros podemos acceder posteriormente desde JavaScript a dichos objetos.

El objeto principal es el FORM que contendrá todos los otros objetos: TEXT (editor de líneas), TEXTAREA (editor de varias líneas), etc.

Nuestra actividad en JavaScript es procesar los eventos que generan estos controles.

A los eventos de los objetos HTML se les asocia una función, dicha función se ejecuta cuando se dispara el evento respectivo.

<body>

<script>

var variable=0;

function nombre\_funcion()

{

Contenido de la función…

alert(“”); //si se agrega un alerta

}

</script>

<form>

<input type="button" onClick="nombre\_funcion()" value="incrementar">

</form>

### GetElemntById:

La propiedad "id" es un identificador único para cualquier elemento HTML que luego nos permite desde Javascript acceder a dicho elemento.

El método getElementById nos retorna una referencia del objeto HTML que le pasamos como parámetro, a partir de este objeto accedemos a la propiedad value que almacena el valor ingresado por el operador en el control TEXT.

<body>

<script>

function mostrar()

{

var nom=document.getElementById('nombre').value;

var ed=document.getElementById('edad').value;

alert('Ingresó el nombre:' + nom);

alert('Y la edad:' + ed);

}

</script>

<form>

Ingrese su nombre:

<input type="text" id="nombre"><br>

Ingrese su edad:

<input type="text" id="edad"><br>

<input type="button" value="Confirmar" onClick="mostrar()">

</form>

### Evento onChange(), SELECT, selectedIndex y options:

Select es un objeto visual que podemos disponer en un FORM permite realizar la selección de un string de una lista y tener asociado al mismo un valor no visible. El objetivo fundamental en JavaScript es determinar qué elemento está seleccionado y qué valor tiene asociado. Esto lo hacemos cuando ocurre el evento onChange.

* Para determinar la posición del índice seleccionado en la lista:

document.getElementById('select1').selectedIndex;

* selectedIndex: almacena la posición del string seleccionado de la lista, numerando a partir de cero:
* Para determinar el string seleccionado: document.getElementById('select1').options[document.getElementById('select1').selectedIndex].text;
* Options: con esta expresión tambien accedemos al valor asociado al string: document.getElementById('select1').options[document.getElementById('select1').selectedIndex].value;

<script>

function cambiarColor()

{

var seleccion=document.getElementById('select1');

document.getElementById('text1').value=seleccion.selectedIndex;

document.getElementById('text2').value=seleccion.options[seleccion.selectedIndex].text;

document.getElementById('text3').value=seleccion.options[seleccion.selectedIndex].value;

}

</script>

<form>

<select id="select1" onChange="cambiarColor()">

<option value="ff0000">Rojo</option>

<option value="00ff00">Verde</option>

<option value="0000ff">Azul</option>

</select>

<br>

Número de índice seleccionado del objeto SELECT:<input type="text" id="text1"><br>

Texto seleccionado:<input type="text" id="text2"><br>

Valor asociado:<input type="text" id="text3"><br>

</form>

### Propiedad Checked:

<script>

function contarSeleccionados()

{

var cant=0;

if (document.getElementById('checkbox1').checked)

{

cant++;

}

if (document.getElementById('checkbox2').checked)

{

cant++;

}

if (document.getElementById('checkbox3').checked)

{

cant++;

}

if (document.getElementById('checkbox4').checked)

{

cant++;

}

alert('Conoce ' + cant + ' lenguajes');

}

</script>

<form>

<input type="checkbox" id="checkbox1">JavaScript

<br>

<input type="checkbox" id="checkbox2">PHP

<br>

<input type="checkbox" id="checkbox3">JSP

<br>

<input type="checkbox" id="checkbox4">VB.Net

<br>

<input type="button" value="Mostrar" onClick="contarSeleccionados()">

</form>

Cuando se presiona el botón se llama a la función Javascript contarSeleccionados(). En la misma verificamos uno a uno cada control checkbox accediendo a la propiedad checked del elemento que almacena true o false según esté o no seleccionado el control.

Como la propiedad checked almacena un true o false podemos utilizar dicho valor directamente como valor de la condición en lugar de codificar:

if (document.getElementById('checkbox1').checked==true){

cant++;

}

<script>

function mostrarSeleccionado()

{

if (document.getElementById('radio1').checked)

{

alert('no tienes estudios');

}

if (document.getElementById('radio2').checked)

{

alert('tienes estudios primarios');

}

if (document.getElementById('radio3').checked)

{

alert('tienes estudios secundarios');

}

if (document.getElementById('radio4').checked)

{

alert('tienes estudios universitarios');

}

}

</script>

<form>

<input type="radio" id="radio1" name="estudios">Sin estudios

<br>

<input type="radio" id="radio2" name="estudios">Primarios

<br>

<input type="radio" id="radio3" name="estudios">Secundarios

<br>

<input type="radio" id="radio4" name="estudios">Universitarios

<br>

<input type="button" value="Mostrar" onClick="mostrarSeleccionado()">

</form>

Es importante notar que todos los objetos de tipo RADIO tienen definida la propiedad name con el mismo valor (esto permite especificar que queremos que los radios estén relacionados entre si)

Luego podemos acceder a cada elemento mediante el método getElementById para consultar la propiedad checked:

if (document.getElementById('radio1').checked){

alert('no tienes estudios');

}

Igual que el checkbox, la propiedad checked retorna true o false, según esté o no seleccionado el control radio.

## Eventos

### onFocus y onBlur:

El evento onFocus se dispara cuando el objeto toma foco y el evento onBlur cuando el objeto pierde el foco.

<script>

function vaciar(control)

{

control.value='';

}

function verificarEntrada(control)

{

if (control.value=='')

alert('Debe ingresar datos');

}

</script>

<form>

<input type="text" id="nombre" onFocus="vaciar(this)"

onBlur="verificarEntrada(this)" value="nombre"><br>

<input type="text" id="edad" onFocus="vaciar(this)"

onBlur="verificarEntrada(this)" value="mail">

<br>

<input type="button" value="Confirmar">

</form>

### onMouseOver y onMouseOut:

El evento onMouseOver se ejecuta cuando pasamos la flecha del mouse sobre un elemento HTML y el evento onMouseOut cuando la flecha abandona el mismo.

<script>

function pintar(col)

{

document.getElementById('cuadrado1').style.backgroundColor=col;

}

</script>

<a href="#" onMouseOver="pintar('#f00')" onMouseOut="pintar('#000')">Rojo</a> -

<a href="#" onMouseOver="pintar('#0f0')" onMouseOut="pintar('#000')">Verde</a> -

<a href="#" onMouseOver="pintar('#00f')" onMouseOut="pintar('#000')">Azul</a>

<div id="cuadrado1" style="background:#000;width:200px;height:200px"></div>

<script>

function pintar(objeto,col)

{

objeto.style.backgroundColor=col;

}

</script>

<table border="1">

<tr>

<td onMouseOver="pintar(this,'#f00')"

onMouseOut="pintar(this,'#fff')">rojo</td>

<td onMouseOver="pintar(this,'#0f0')"

onMouseOut="pintar(this,'#fff')">verde</td>

<td onMouseOver="pintar(this,'#00f')"

onMouseOut="pintar(this,'#fff')">azul</td>

</tr>

</table>

### OnLoad:

El evento onLoad se ejecuta al final de la carga de la página en el navegador. Tener en cuenta que se dispara el evento cuando todos los elementos HTML están creados y se han descargado todos los recursos que hace referencia la página como podrían ser las imágenes, archivos de hoja de estilo etc. Un uso posible es para fijar el foco en algún control de un formulario, para que el operador no tenga que activar con el mouse dicho control.

Este evento está asociado a la etiqueta body.

<body onLoad="activarPrimerControl()">

<script>

function activarPrimerControl()

{

document.getElementById('nombre').focus();

}

</script>

<form>

Ingrese su nombre:

<input type="text" id="nombre"><br>

Ingrese su edad:

<input type="text" id="edad"><br>

<input type="button" value="Confirmar">

</form>

</body>

### Load y DOMContentLoaded:

* Evento 'load': El evento load se dispara cuando el contenido del archivo HTML y las referencias a todos los recursos asociados (imágenes, hojas de estilo etc.) se han cargado en memoria del navegador.
* Evento 'DOMContentLoaded': El evento DOMContentLoaded se dispara cuando el contenido del archivo HTML se ha cargado en el navegador, sin necesitar esperar imágenes, hojas de estilo etc.

Es muy conveniente en la mayoría de las veces inicializar script de JavaScript utilizando éste evento en lugar del evento 'load'. Todos los navegadores modernos disponen del evento 'DOMContentLoaded'.

### Click y dblclick:

Dos eventos que podemos capturar de cualquier elemento HTML son cuando un usuario hace un clic o un doble clic sobre el mismo.

El evento click se produce cuando el usuario hace un solo clic sobre el elemento HTML y suelta inmediatamente el botón del mouse en el mismo lugar y el dblclick cuando presiona en forma sucesiva en la misma ubicación.

<body>

<div style="width:200px;height:200px;background:#ffff00" id="recuadro1">

Prueba del evento click

</div>

<div style="width:200px;height:200px;background:#ff5500" id="recuadro2">

Prueba del evento dblclick

</div>

<script>

addEventListener('DOMContentLoaded',inicio,false);

function inicio()

{

document.getElementById('recuadro1').addEventListener('click',presion1,false);

document.getElementById('recuadro2').addEventListener('dblclick',presion2,false);

}

function presion1()

{

alert('se hizo click');

}

function presion2()

{

alert('se hizo doble click');

}

</script>

</body>

Primero registramos mediante la llamada al método addEventListener el evento DOMContentLoaded, y dentro de la función inicio registramos los eventos click y dblclick para los dos div que definimos en la página HTML:

addEventListener('DOMContentLoaded',inicio,false);

function inicio()

{

document.getElementById('recuadro1').addEventListener('click',presion1,false);

document.getElementById('recuadro2').addEventListener('dblclick',presion2,false);

}

Como vemos lo único que hacemos en los métodos que se disparan es mostrar un mensaje:

function presion1()

{

alert('se hizo click');

}

function presion2()

{

alert('se hizo doble click');

}

### Mousedown y mouseup:

Otros dos eventos relacionados con el mouse son mousedown y mouseup.

El evento mousedown se dispara inmediatamente luego que presionamos con la flecha del mouse un elemento HTML que tiene registrado dicho evento.

El evento mouseup se ejecuta luego de soltar el botón del mouse estando dentro del control HTML.

<body>

<p>Presione el recuadro amarillo sin soltar el botón del mouse.</p>

<div style="width:200px;height:200px;background:#ffff00" id="recuadro1">

</div>

<p>Presione el recuadro naranja y suelte el botón del mouse.</p>

<div style="width:200px;height:200px;background:#ff5500" id="recuadro2">

</div>

<script>

addEventListener('DOMContentLoaded',inicio,false);

function inicio()

{

document.getElementById('recuadro1').addEventListener('mousedown',presion1,false);

document.getElementById('recuadro2').addEventListener('mouseup',presion2,false);

}

function presion1()

{

document.getElementById('recuadro1').innerHTML='Se presione el mouse y todavía no se soltó';

}

function presion2()

{

document.getElementById('recuadro2').innerHTML='Se presione el mouse y se soltó';

}

</script>

</body>

### Mouseover y mouseout:

El evento mouseover se dispara cuando ingresamos con la flecha del mouse a un control HTML que tiene registrado dicho evento, y mouseout se dispara cuando sacamos la flecha del mouse del control.

Es muy común utilizar estos dos eventos para producir cambios en el elemento HTML cuando ingresamos la flecha del mouse y retornar al estado anterior cuando se saca la flecha del mouse.

Es importante hacer notar que estos eventos se disparan sin tener que presionar la flecha del mouse, solo con desplazarla al interior del elemento HTML se dispara el evento mouseover.

<body>

<div style="width:200px;height:200px;background:#0000ff" id="recuadro1">

</div>

<script>

addEventListener('DOMContentLoaded',inicio,false);

function inicio()

{

document.getElementById('recuadro1').addEventListener('mouseover',entrada,false);

document.getElementById('recuadro1').addEventListener('mouseout',salida,false);

}

function entrada()

{ document.getElementById('recuadro1').style.borderRadius='30px';

}

function salida()

{

document.getElementById('recuadro1').style.borderRadius='0px';

}

</script>

</body>

### Mousemove:

El evento mousemove se dispara cada vez que desplazamos la flecha del mouse sobre el elemento HTML que está escuchando este evento.

Este tipo de evento hay que utilizarlo con cuidado ya que puede sobrecargar el navegador, ya que este evento se dispara cada vez que desplazamos aunque sea solo un pixel.

<body>

<div style="width:200px;height:200px;background:#0000ff" id="recuadro1">

</div>

<p id="cantidad">0</p>

<script>

addEventListener('DOMContentLoaded',inicio,false);

function inicio()

{ document.getElementById('recuadro1').addEventListener('mousemove',mover,false);

}

function mover()

{

var x=parseInt(document.getElementById('cantidad').innerHTML);

x++;

document.getElementById('cantidad').innerHTML=x;

}

</script>

</body>

### keydown, keyup y keypress:

Estos tres eventos son similares a los eventos del mouse: mousedown, mouseup y click pero orientados al teclado del equipo.

El evento keydown se dispara cuando presionamos cualquier tecla del teclado, el evento keyup cuando soltamos una tecla. En cuanto el evento keypress en un principio procesa tanto cuando se la presionó y soltó, el único y gran inconveniente es que la mayoría de los navegadores no dispara el evento keypress para todas las teclas del teclado.

<body>

<input type="text" id="texto" maxlength="140" size="140">

<br>

<p>Máxima cantidad de caracteres disponibles:<span id="cantidad">140</span></p>

<script>

addEventListener('DOMContentLoaded',inicio,false);

function inicio()

{

document.getElementById('texto').addEventListener('keyup',presion,false);

}

function presion()

{

var canti=document.getElementById('texto').value.length;

var disponibles=140-parseInt(canti);

document.getElementById('cantidad').innerHTML=disponibles;

}

</script>

### Change:

El evento change se lo puede asociar a distintos elementos de formularios HTML y su comportamiento depende de cada uno.

Cuando asociamos el evento change a un control de tipo checkbox el mismo se dispara inmediatamente después que lo chequeamos o sacamos la selección. Para los controles de tipo radio también se dispara luego que cambia el estado de selección del mismo.

Para los controles select se dispara el evento change cuando cambiamos el item seleccionado.

Para los controles text y textarea se dispara cuando pierde el foco del control (porque seleccionamos otro control) y hemos cambiando el contenido del mismo.

<body>

Evento change del checkbox:

<input type="checkbox" id="checkbox1" name="checkbox1" checked>

<br>

Evento change del radio:<br>

<input type="radio" name="radio1" id="radioa"><br>

<input type="radio" name="radio1" id="radiob"><br>

<input type="radio" name="radio1" id="radioc"><br>

Evento change del select:

<select name="select1" id="select1">

<option value="argentina">Argentina</option>

<option value="chile">Chile</option>

<option value="uruguay">Uruguay</option>

<option value="paraguay">Paraguay</option>

<option value="bolivia">Bolivia</option>

</select>

<br>

<input type="text" name="text1" id="text1">

<br>

<textarea name="text1" id="textarea1" rows="6" cols="80"></textarea>

<br>

<script>

addEventListener('DOMContentLoaded',inicio,false);

function inicio()

{ document.getElementById('checkbox1').addEventListener('change',cambiocheckbox,false); document.getElementById('radioa').addEventListener('change',cambioradio,false); document.getElementById('radiob').addEventListener('change',cambioradio,false); document.getElementById('radioc').addEventListener('change',cambioradio,false); document.getElementById('select1').addEventListener('change',cambioselect,false); document.getElementById('text1').addEventListener('change',cambiotext,false); document.getElementById('textarea1').addEventListener('change',cambiotextarea,false);

}

function cambiocheckbox()

{

alert(document.getElementById('checkbox1').checked);

}

function cambioradio()

{

alert(document.getElementById('radioa').checked+' '+

document.getElementById('radiob').checked+' '+

document.getElementById('radioc').checked);

}

function cambioselect()

{

alert(document.getElementById('select1').value);

}

function cambiotext()

{

alert(document.getElementById('text1').value);

}

function cambiotextarea()

{

alert(document.getElementById('textarea1').value);

}

</script>

</body>

### Focus y blur:

El evento focus se dispara cuando se activa el control o toma foco y el evento blur se dispara cuando pierde el foco el control. Podemos capturar el evento focus y blur de un control de tipo text, textarea, button, checkbox, file, password, radio, reset y submit.

<body>

<input type="text" id="text1" name="text1" size="30">

<br>

<input type="text" id="text2" name="text2" size="30">

<script>

addEventListener('DOMContentLoaded',inicio,false);

function inicio()

{ document.getElementById('text1').addEventListener('focus',tomarfoco1,false); document.getElementById('text2').addEventListener('focus',tomarfoco2,false); document.getElementById('text1').addEventListener('blur',perderfoco1,false); document.getElementById('text2').addEventListener('blur',perderfoco2,false);

}

function tomarfoco1()

{ document.getElementById('text1').style.color='#ff0000';

}

function tomarfoco2()

{ document.getElementById('text2').style.color='#ff0000';

}

function perderfoco1()

{ document.getElementById('text1').style.color='#000000';

}

function perderfoco2()

{

document.getElementById('text2').style.color='#000000';

}

</script>

</body>

### Parámetro del método asociado al addEventListener:

Hasta ahora siempre hemos asociado una función distinta para procesar los eventos de distintos elementos HTML. Pero veremos que está permitido asociar una única función a varios eventos de distintos objetos.

Debemos definir un parámetro en la función a implementar que llega la referencia del objeto que emitió el evento.

Implementaremos un panel con un conjunto de botones que nos permitan ingresar un valor numérico presionando botones.

Asociaremos el click de cada botón con una única función.

<body>

<input type="button" id="boton0" name="boton0" value="0" class="boton">

<input type="button" id="boton1" name="boton1" value="1" class="boton">

<input type="button" id="boton2" name="boton2" value="2" class="boton">

<input type="button" id="boton3" name="boton3" value="3" class="boton">

<input type="button" id="boton4" name="boton4" value="4" class="boton">

<input type="button" id="boton5" name="boton5" value="5" class="boton">

<input type="button" id="boton6" name="boton6" value="6" class="boton">

<input type="button" id="boton7" name="boton7" value="7" class="boton">

<input type="button" id="boton8" name="boton8" value="8" class="boton">

<input type="button" id="boton9" name="boton9" value="9" class="boton">

<br>

<div id="resultado"></div>

<script>

window.addEventListener('DOMContentLoaded', inicio, false);

function inicio() {

for (var x = 0; x <= 9; x++) {

document.getElementById('boton' + x).addEventListener('click', presion, false);

}

}

function presion(evt) {

document.getElementById('resultado').innerHTML =

document.getElementById('resultado').innerHTML + evt.target.value;

}

</script>

</body>

En el body definimos 10 botones y un div donde mostraremos todos los números ingresados.

<input type="button" id="boton0" name="boton0" value="0" class="boton">

<input type="button" id="boton1" name="boton1" value="1" class="boton">

<input type="button" id="boton2" name="boton2" value="2" class="boton">

<input type="button" id="boton3" name="boton3" value="3" class="boton">

<input type="button" id="boton4" name="boton4" value="4" class="boton">

<input type="button" id="boton5" name="boton5" value="5" class="boton">

<input type="button" id="boton6" name="boton6" value="6" class="boton">

<input type="button" id="boton7" name="boton7" value="7" class="boton">

<input type="button" id="boton8" name="boton8" value="8" class="boton">

<input type="button" id="boton9" name="boton9" value="9" class="boton">

<br>

<div id="resultado"></div>

En el método inicio asociamos los diez botones con la función presion y para reducir el código disponemos un for que se repita 10 veces y llamamos en la misma al método addEventListener (debemos ir obteniendo el id del botón concatenando 'boton' y el contador del for):

function inicio() {

for (var x = 0; x <= 9; x++) {

document.getElementById('boton' + x).addEventListener('click', presion, false);

}

}

Lo nuevo aparece en la función presion:

function presion(evt) {

document.getElementById('resultado').innerHTML =

document.getElementById('resultado').innerHTML + evt.target.value;

}

La función presion tiene un parámetro llamado evt (podemos darle cualquier nombre) y el mismo es un objeto que tiene las siguientes propiedades:

* target: Referencia del objeto que generó el evento (en nuestro ejemplo alguno de los 10 botones)
* type: El nombre del evento (en nuestro caso click)
* button: El botón del mouse presionado (0 = izquierdo, 1 = medio, 2 = derecho)
* keyCode: El caracter del teclado presionado (en caso que corresponda)
* shiftKey: true o false en caso de estar presionada esta tecla.

Como en este problema debemos ir concatenando el número presionado procedemos a obtener la referencia del div y asignarle el valor actual más la propiedad value del botón presionado:

document.getElementById('resultado').innerHTML =

document.getElementById('resultado').innerHTML + evt.target.value;

### Submit:

Todo formulario se le puede capturar el evento submit que se dispara previo a enviar los datos del formulario al servidor.

Uno de los usos más extendidos es la de validar los datos ingresados al formulario y abortar el envío de los mismos al servidor (con esto liberamos sobrecargas del servidor)

El evento submit se dispara cuando presionamos un botón de tipo type="submit".

Para probar el funcionamiento del evento submit implementaremos un formulario que solicita la carga de una clave y la repetición de la misma. Luego cuando se presione un botón de tipo "submit" verificaremos que las dos claves ingresadas sean iguales.

<body>

<form method="post" action="procesar.php" id="formulario1">

Ingrese clave:

<input type="password" id="clave1" name="clave1" size="20" required>

<br>

Repita clave:

<input type="password" id="clave2" name="clave2" size="20" required>

<br>

<input type="submit" id="confirmar" name="confirmar" value="Confirmar">

</form>

<script>

window.addEventListener('DOMContentLoaded', inicio, false);

function inicio() { document.getElementById("formulario1").addEventListener('submit', validar, false);

}

function validar(evt) {

var cla1 = document.getElementById("clave1").value;

var cla2 = document.getElementById("clave2").value;

if (cla1!=cla2) {

alert('Las claves ingresadas son distintas');

evt.preventDefault();

}

}

</script>

Definimos un formulario que solicita la carga de dos claves y un botón submit para enviar los datos al servidor:

<form method="post" action="procesar.php" id="formulario1">

Ingrese clave:

<input type="password" id="clave1" name="clave1" size="20" required>

<br>

Repita clave:

<input type="password" id="clave2" name="clave2" size="20" required>

<br>

<input type="submit" id="confirmar" name="confirmar" value="Confirmar">

</form>

Registramos el evento DOMContentLoaded de la página indicando que se ejecute la función inicio donde registramos el evento submit del formulario:

window.addEventListener('DOMContentLoaded', inicio, false);

function inicio() {

document.getElementById("formulario1").addEventListener('submit', validar, false);

}

La función validar extrae los contenidos de los dos "password" y verificamos si tienen string distintos en cuyo caso llamando al método preventDefault del objeto que llega como parámetro lo cual previene que los datos se envíen al servidor:

function validar(evt) {

var cla1 = document.getElementById("clave1").value;

var cla2 = document.getElementById("clave2").value;

if (cla1!=cla2) {

alert('Las claves ingresadas son distintas');

evt.preventDefault();

}

}

## Objeto Window

window es un objeto global y tiene los siguienes métodos:

* alert: Muestra un diálogo de alerta con un mensaje
* prompt: Muestra un diálogo para la entrada de un valor de tipo string
* confirm: Muestra un diálogo de confirmación con los botones Confirmar y Cancelar.
* open y close: abre o cierra una ventana del navegador.

Podemos especificar el tamaño de la ventana, su contenido, etc.

* [Variable=][window.]open(URL, nombre, propiedades): Permite crear (y abrir) una nueva ventana. Si queremos tener acceso a ella desde la ventana donde la creamos, deberemos asignarle una variable, sino simplemente invocamos el método: el navegador automáticamente sabrá que pertenece al objeto window. El parámetro URL es una cadena que contendrá la dirección de la ventana que estamos abriendo: si está en blanco, la ventana se abrirá con una página en blanco.

Las propiedades son una lista, separada por comas, de algunos de los siguientes elementos:

* toolbar[=yes|no]
* location[=yes|no]
* directories[=yes|no]
* status[=yes|no]
* menubar[=yes|no]
* scrollbars[=yes|no]
* resizable[=yes|no]
* width=pixels
* height=pixels

Es bueno hacer notar que a todas estas funciones las podemos llamar anteponiéndole el nombre del objeto window, seguida del método o en forma resumida indicando solamente el nombre del método, esto es posible ya que el objeto window es el objeto de máximo nivel.

* valor=window.prompt("Ingrese valor","");
* valor=prompt("Ingrese valor","");

Para reducir la cantidad de caracteres que se tipean normalmente encontraremos los programas tipeados de la segunda forma.

<script>

function abrir()

{

var ventana=open();

ventana.document.write("Estoy escribiendo en la nueva ventana<br>");

ventana.document.write("Segunda linea");

}

function abrirParametros()

{

var ventana=open('','','status=yes,width=400,height=250,menubar=yes');

ventana.document.write("Esto es lo primero que aparece<br>");

}

function mostrarAlerta()

{

alert("Esta ventana de alerta ya la utilizamos en otros problemas.");

}

function confirmar()

{

var respuesta=confirm("Presione alguno de los dos botones");

if (respuesta==true)

alert("presionó aceptar");

else

alert("presionó cancelar");

}

function cargarCadena()

{

var cad=prompt("cargue una cadena:","");

alert("Usted ingreso "+cad);

}

</script>

<p>Este programa permite analizar la llamada a

distintas responsabilidades del objeto window.</p>

<form>

<br>

<input type="button" value="open()" onClick="abrir()">

<br>

<input type="button" value="open con parámetros" onClick="abrirParametros()" >

<br>

<input type="button" value="alert" onClick="mostrarAlerta()">

<br>

<input type="button" value="confirm" onClick="confirmar()">

<br>

<input type="button" value="prompt" onClick="cargarCadena()">

</form>

### Propiedad location:

Cuando le asignamos una nueva dirección a la propiedad location del objeto window, el navegador redirecciona a dicha página.

<body>

<script>

function verificarEntrada()

{

if (window.confirm('Desea salir del sitio e ingresar al periódico La Nación?'))

{

window.location='http://www.lanacion.com.ar';

}

else

{

window.alert('No hay problema');

}

}

</script>

<button onClick="verificarEntrada()">Ingresar a La Nación</button>

</body>

### Propiedad history:

El objeto history almacena todas las páginas que visitamos. Luego, con una serie de funciones, podemos extraer de la memoria de la computadora las páginas ya visitadas, sin tener que pedirlas nuevamente al servidor.

Cuenta con las siguientes funciones:

* window.history.back(); //Retrocede a la página anterior
* window.history.forward(); //Avanza a la página siguiente almacenada en la cache de la máquina.
* window.history.go(); //Avanza o retrocede en la lista de páginas visitadas.

Llamar a la función back, tiene el mismo comportamiento que presionar el botón "Atrás" del navegador.

<body>

<script>

function avanzar()

{

window.history.go(1);

}

function retornar()

{

window.history.go(-1);

}

</script>

</head>

<body>

<a href="pagina2.html">Ir a la página 2</a>

<br>

<br>

<a href="javascript:avanzar()">Extraer del cache la segunda página</a>

<a href="javascript:retornar()">Retornar</a>

</body>

Podemos mejorar el ejemplo accediendo al atributo length (almacena la cantidad de páginas de la lista) del objeto history:

if (window.history.length>0)

{

window.history.go(1);

}

else

{

alert('no hay otra página en la cache hacia adelante');

}

### Propiedad Screen:

El objeto screen ofrece información acerca del monitor donde se está ejecutando el navegador.

Las propiedades principales del objeto screen son:

* availHeight : El alto de la pantalla en pixeles disponible para el navegador.
* availWidth : El ancho de la pantalla en pixeles disponible para el navegador.
* colorDepth : Representa el número de bits usados para representar los colores.
* height : El alto de la pantalla en pixeles.
* width : El ancho de la pantalla en pixeles.

<script>

document.write('Valores de las propiedades del objeto screen:<br>');

document.write('availHeight :' + screen.availHeight + '<br>');

document.write('availWidth :' + screen.availWidth + '<br>');

document.write('height :' + screen.height + '<br>');

document.write('width :' + screen.width + '<br>');

document.write('colorDepth :' + screen.colorDepth);

</script>

### Propiedad navigator:

Contiene información sobre el navegador web. La implementación de este objeto varía entre navegadores (Chrome, Edge, IExplorer, FireFox, Opera, etc.)

Las propiedades comunes a estos navegadores son:

* appName : almacena el nombre oficial del navegador.
* appVersion : almacena la versión del navegador.
* cookieEnabled : almacena si las cookies están activas en el navegador.
* platform : almacena la plataforma donde el navegador se está ejecutando.
* plugins : almacena un array de los plugin cargados en el navegador.

<script>

document.write('Valores de las propiedades del objeto navigator:<br>');

document.write('appName :' + navigator.appName + '<br>');

document.write('appVersion :' + navigator.appVersion + '<br>');

document.write('cookieEnabled :' + navigator.cookieEnabled + '<br>');

document.write('plugins :' + navigator.plugins.length + '<br>');

</script>

## POO en JS.

Desarrollaremos una clase que represente un cliente de un banco.

La clase Cliente tiene como atributos:

* nombre
* saldo

Y las responsabilidades o métodos de la clase son:

* Constructor (inicializamos los atributos del objeto)
* depositar
* extraer

Luego debemos implementar los siguientes métodos (normalmente el constructor se utiliza el caracter mayúscula):

function Cliente(nombre,saldo)

{

this.nombre=nombre;

this.saldo=saldo;

this.depositar=depositar;

this.extraer=extraer;

}

function depositar(dinero)

{

this.saldo=this.saldo+dinero;

}

function extraer(dinero)

{

this.saldo=this.saldo-dinero;

}

El nombre de la clase coincide con el nombre de la función principal que implementamos (también llamado constructor de la clase):

function Cliente(nombre,saldo)

{

this.nombre=nombre;

this.saldo=saldo;

this.depositar=depositar;

this.extraer=extraer;

}

A ésta función llegan como parámetro los valores con que queremos inicializar los atributos. Con la palabra clave 'this' diferenciamos los atributos de los parámetros (los atributos deben llevar la palabra clave this)

this.nombre=nombre;

this.saldo=saldo;

También en el constructor inicializamos la referencia a todos los métodos que contendrá la clase (esto es muy importante y necesario para entender porque las otras dos funciones pertenecen a esta clase):

this.depositar=depositar;

this.extraer=extraer;

Por último, implementamos todos los métodos de la clase:

function depositar(dinero)

{

this.saldo=this.saldo+dinero;

}

function extraer(dinero)

{

this.saldo=this.saldo-dinero;

}

De nuevo recordemos que diferenciamos los atributos de la clase por la palabra clave this.

<script>

function Cliente(nombre,saldo)

{

this.nombre=nombre;

this.saldo=saldo;

this.depositar=depositar;

this.extraer=extraer;

}

function depositar(dinero)

{

this.saldo=this.saldo+dinero;

}

function extraer(dinero)

{

this.saldo=this.saldo-dinero;

}

var cliente1;

cliente1=new Cliente('diego',1200);

document.write('Nombre del cliente:'+cliente1.nombre+'<br>');

document.write('Saldo actual:'+cliente1.saldo+'<br>');

cliente1.depositar(120);

document.write('Saldo luego de depositar $120---->'+cliente1.saldo+'<br>');

cliente1.extraer(1000);

document.write('Saldo luego de extraer $1000---->'+cliente1.saldo+'<br>');

</script>

Para definir un objeto de la clase Cliente tenemos:

var cliente1;

cliente1=new Cliente('diego',1200);

Luego las llamadas a métodos le antecedemos el nombre del objeto llamado cliente1:

document.write('Nombre del cliente:'+cliente1.nombre+'<br>');

document.write('Saldo actual:'+cliente1.saldo+'<br>');

cliente1.depositar(120);

document.write('Saldo luego de depositar $120---->'+cliente1.saldo+'<br>');

cliente1.extraer(1000);

document.write('Saldo luego de extraer $1000---->'+cliente1.saldo+'<br>');

### Definición de varias clases:

En JavaScript podemos definir varias clases en un mismo programa.

Vamos a desarrollar un programa que contenga dos clases. Plantearemos una clase Numeroquiniela que representa una persona que elige un número de quiniela y además registra su nombre, la clase tiene por objetivo la carga por el teclado del número deseado.

Por otra parte crearemos una clase Bolillero que sortee un valor aleatorio entre 1 y 10 (que representa el valor extraído del bolillero)

<script>

//clase Numeroquiniela \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

function Numeroquiniela(nombre)

{

this.nombre=nombre;

this.cargarnumero=cargarnumero;

this.verificarsigano=verificarsigano;

}

function cargarnumero()

{

this.numero=prompt("Que número de quiniela quiere?","");

}

function verificarsigano(num)

{

if (this.numero==num)

return true;

else

return false;

}

//clase Bolillero \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

function Bolillero()

{

this.numero=-1;

this.sortear=sortear;

}

function sortear()

{

this.numero=parseInt(Math.random()\*10)+1;

}

// bloque principal

var numeroquiniela1;

numeroquiniela1=new Numeroquiniela("juan");

numeroquiniela1.cargarnumero();

var numeroquiniela2;

numeroquiniela2=new Numeroquiniela("ana");

numeroquiniela2.cargarnumero();

var bolillero;

bolillero=new Bolillero();

bolillero.sortear();

document.write('Numero sorteado:' + bolillero.numero + '<br>');

document.write(numeroquiniela1.nombre + ' eligió ' + numeroquiniela1.numero +'<br>');

document.write(numeroquiniela2.nombre + ' eligió ' + numeroquiniela2.numero +'<br>');

if (numeroquiniela1.verificarsigano(bolillero.numero))

{

document.write(numeroquiniela1.nombre + ' ha ganado <br>');

}

if (numeroquiniela2.verificarsigano(bolillero.numero))

{

document.write(numeroquiniela2.nombre + ' ha ganado <br>');

}

else

{

document.write('Nadie ha ganado <br>');

}

</script>

Al constructor de la clase Numeroquiniela llega como parámetro el nombre de la persona que la compra (podíamos cargarlo por teclado al nombre también)

Al número que selecciona lo cargamos por teclado. La clase Numeroquiniela además tiene otra responsabilidad, que es avisarnos si ganó según el número sorteado.

Por otro lado en la página html definimos dos objetos de la clase Numeroquiniela y uno de la clase Bolillero.

### Vectores con componentes de tipo objeto:

Podemos crear vectores con componente de tipo objeto.

Con un ejemplo veremos la sintaxis para trabajar con los mismos. Desarrollaremos una clase que represente un hipervínculo y luego definiremos un vector con componentes de tipo hipervinculo.

La clase hipervínculo y donde definimos objetos de dicha clase:

<script>

function Hipervinculo(direccion,titulo)

{

this.direccion=direccion;

this.titulo=titulo;

this.retornarhipervinculo=retornarhipervinculo;

}

function retornarhipervinculo()

{

var cadena;

cadena='<a href=' + this.direccion + '>' + this.titulo + '</a>';

return cadena;

}

var vector=new Array(3);

vector[0]=new Hipervinculo('http://www.google.com','google');

vector[1]=new Hipervinculo('http://www.msn.com','msn');

vector[2]=new Hipervinculo('http://www.yahoo.com','yahoo');

for(var f=0;f<vector.length;f++)

{

document.write(vector[f].retornarhipervinculo());

document.write('<br>');

}

</script>

Creamos un objeto de la clase Array y luego guardamos en cada componente un objeto de la clase hipervínculo (pasándole como parámetros al constructor, la dirección del sitio y el texto a mostrar en la página) Luego recorremos con un 'for' las componentes del vector e imprimimos en la página cada hipervínculo.

### Creación de objetos literales:

Esta metodología consiste en definir una lista de propiedades y sus valores. Veamos con un ejemplo esta técnica:

<script>

var cliente1 = {

nombre: 'Juan',

deposito: 0,

imprimir: function ()

{

document.write(this.nombre+'<br>');

document.write(this.deposito+'<br>');

},

depositar: function(monto) {

this.deposito=this.deposito+monto;

},

extraer: function(monto) {

this.deposito=this.deposito-monto;

}

};

cliente1.imprimir();

cliente1.depositar(1000);

document.write('Estado luego de depositar 1000 pesos</br>');

cliente1.imprimir();

cliente1.extraer(200);

document.write('Estado luego de extraer 200 pesos</br>');

cliente1.imprimir();

</script>

En este ejemplo hemos creado un objeto literal llamado cliente1, la sintaxis mínima para crear un objeto vacío sería:

var cliente1 = {};

Es decir creamos una variable llamada cliente1 y le asignamos un bloque encerrado entre llaves vacío. Es importante notar el punto y coma al final de la llave de cerrado (como ocurre cuando asignamos un valor a una variable)

Veamos ahora si decimos que el objeto cliente1 define la propiedad nombre, luego nuestro objeto quedará definido con la sintaxis:

var cliente1 = {

nombre: 'Juan'

};

Decimos que la propiedad nombre almacena el string 'Juan', del lado izquierdo indicamos el nombre de la propiedad y del lado derecho de los dos puntos indicamos el valor de la propiedad del objeto (el valor puede ser de cualquier tipo, en este caso es de tipo string pero podría ser de tipo number, boolean, object, Array etc.)

Ahora si agregamos una segunda propiedad a nuestro objeto cliente1 llamada deposito (que representa la cantidad de dinero que tiene depositado el cliente1) la sintaxis queda:

var cliente1 = {

nombre: 'Juan',

deposito: 0

};

Como podemos observar separamos por coma cada inicialización de propiedad del objeto (menos para la última propiedad donde aparece la "}".

Las funciones del objeto también definimos una sintaxis similar a la declaración de sus propiedades:

var cliente1 = {

nombre: 'Juan',

deposito: 0,

imprimir: function ()

{

document.write(this.nombre+'<br>');

document.write(this.deposito+'<br>');

},

depositar: function(monto) {

this.deposito=this.deposito+monto;

},

extraer: function(monto) {

this.deposito=this.deposito-monto;

}

};

Del lado izquierdo de los dos puntos indicamos el nombre de la función y del lado derecho utilizamos la palabra clave function junto con los parámetros.

En la función podemos acceder a las propiedades del objeto antecediendo la palabra clave this.

Ahora solo nos falta hacer la llamada a las funciones del objeto cliente1:

cliente1.imprimir();

cliente1.depositar(1000);

document.write('Estado luego de depositar 1000 pesos</br>');

cliente1.imprimir();

cliente1.extraer(200);

document.write('Estado luego de extraer 200 pesos</br>');

cliente1.imprimir();

## Funciones

### Objeto arguments:

Cuando llamamos a una función JavaScript crea automáticamente un objeto llamado "arguments" que almacena los valores que le pasamos a dicha función. También este objeto arguments almacena en una propiedad llamada length la cantidad de parámetros que llegaron.

<script>

function sumar(x,y)

{

var s=arguments[0]+arguments[1];

return s;

}

document.write(sumar(5,3));

</script>

<script>

function sumar()

{

var s=0;

var f;

for(f=0; f<arguments.length; f++)

{

s=s+arguments[f];

}

return s;

}

document.write(sumar(2,4));

document.write('<br>');

document.write(sumar(1,2,3,4,5));

document.write('<br>');

document.write(sumar(100,200,300));

</script>

### Variables locales y variables globales:

Las variables locales se definen dentro de una función y solo se tiene acceso dentro de la misma función. Una variable local se la define antecediendo la palabra clave var al nombre de la variable.

Luego decimos que x es una variable local que solo se la puede acceder dentro de la función imprimir. Si intentamos acceder a la variable x fuera de la función se produce un error en tiempo de ejecución.

Las variables globales son las que definimos fuera de cualquier función. Una variable global tenemos acceso fuera y dentro de las funciones.

### Funciones anidadas:

JavaScript permite definir funciones dentro de otras funciones.

<body>

<div id="div1"></div>

<script>

function generarListaOrdenada(vec)

{

function comienzo()

{

s='<ol>';

}

function fin()

{

s=s+'</ol>';

}

var s='';

comienzo();

var f;

for(f=0;f<vec.length;f++)

{

s=s+'<li>'+vec[f]+'</li>';

}

fin();

return s;

}

onload=function() {

var opciones=['Opción a','Opción b','Opción c','Opción d'];

document.getElementById('div1').innerHTML=generarListaOrdenada(opciones);

}

</script>

</body>

### Parámetros de tipo función:

<script>

var f1=function(x,y)

{

var s=x+y;

return s;

}

document.write(f1(4,6)+'<br>');

var f2=f1;

document.write(f2(3,3)+'<br>');

</script>

<script>

function sumar(v1,v2)

{

return v1+v2;

}

function restar(v1,v2)

{

return v1-v2;

}

function calcular(x,y,fu)

{

var resu=fu(x,y);

return resu;

}

var s1=calcular(10,5,sumar);

document.write(s1+'<br>');

var s2=calcular(10,5,restar);

document.write(s2+'<br>');

</script>

### Función isNaN:

La función global isNaN (is Not a Number) verifica si el valor que le pasamos es un número válido y podemos estar seguros de operar con dicho valor. Esta función puede ser empleada inmediatamente luego de llamar a las funciones parseInt y parseFloat.

Con un pequeño ejemplo podemos ver los valores que nos retorna la función isNaN si le pasamos variables con un valor entero, float, un string con caracteres numéricos y un string con caracteres alfabéticos:

var x=10;

if (isNaN(x)) //false

alert('no es un número');

var z=10.5;

if (isNaN(z)) //false

alert('no es un número');

var edad='77';

if (isNaN(edad)) //false

alert('no es un número');

var nom='juan';

if (isNaN(nom)) //true

alert('no es un número:'+nom);

<script>

var x1,x2,suma;

x1=prompt('Ingrese el primer valor:','');

x2=prompt('Ingrese el segundo valor:','');

x1=parseInt(x1);

x2=parseInt(x2);

if (isNaN(x1) || isNaN(x2))

{

document.write('Al menos uno de los dos valores ingresados no es numérico.');

}

else

{

suma=x1+x2;

document.write('La suma de los dos valores es:'+suma);

}

</script>

### Función isFinite:

La función isFinite verifica si el parámetro tiene un valor numérico finito, en caso afirmativo retorna true.

Una variable numérica puede almacenar alguno de los siguientes valores:

* El número propiamente dicho.
* El valor NaN
* El valor Infinite
* El valor –Infinite

<script>

var x1=120;

if (isFinite(x1)) //true

document.write('x1 tiene un valor finito que es el '+x1);

else

document.write('x1 no tiene un valor finito que es el '+x1);

document.write('<br>');

var x2=10e399;

if (isFinite(x2)) //false

document.write('x2 tiene un valor finito que es el '+x2);

else

document.write('x2 no tiene un valor finito que es el '+x2);

document.write('<br>');

var x3=-10e399;

if (isFinite(x3)) //false

document.write('x3 tiene un valor finito que es el '+x3);

else

document.write('x3 no tiene un valor finito que es el '+x3);

document.write('<br>');

var x4='juan';

if (isFinite(x4)) //false

document.write('x4 tiene un valor finito que es el '+x4);

else

document.write('x4 no tiene un valor finito que es el '+x4);

</script>

El primer if donde verificamos si tiene un valor finito la variable x1 se verifica verdadero:

var x1=120;

if (isFinite(x1)) //true

document.write('x1 tiene un valor finito que es el '+x1);

else

document.write('x1 no tiene un valor finito que es el '+x1);

El segundo if se verifica falso donde controlamos el valor almacenado en la variable x2 (tener en cuenta que en JavaScript el valor máximo a almacenar es 1.7976931348623157e+308), la variable almacena el valor Infinity:

var x2=10e399;

if (isFinite(x2)) //false

document.write('x2 tiene un valor finito que es el '+x2);

else

document.write('x2 no tiene un valor finito que es el '+x2);

De forma similar si almacenamos un valor negativo que supera la capacidad de JavaScript en la variable se almacena -Infinity:

var x3=-10e399;

if (isFinite(x3)) //false

document.write('x3 tiene un valor finito que es el '+x3);

else

document.write('x3 no tiene un valor finito que es el '+x3);

Por último si almacenamos un valor no numérico como un string la función isFinite retorna false:

var x4='juan';

if (isFinite(x4)) //false

document.write('x4 tiene un valor finito que es el '+x4);

else

document.write('x4 no tiene un valor finito que es el '+x4);

Para conocer el máximo y mínimo valor que puede almacenar una variable numérica podemos consultar las constantes MIN\_VALUE y MAX\_VALUE de la clase Number:

<script>

document.write(Number.MIN\_VALUE); // 5e-324

document.write('<br>');

document.write(Number.MAX\_VALUE); // 1.7976931348623157e+308

</script>

### Funciones setInterval, clearInterval, setTimeout y clearTimeout:

La función setTimeout y setInterval permiten pasar como primer parámetro el nombre de una función que se ejecutará después de una determinada cantidad de milisegundos que le pasamos en el segundo parámetro. La única diferencia entre estas funciones es que si llamamos a setInterval la función se ejecutará en forma repetida cada vez que pase dicha cantidad de milisegundos, en cambio con setTimeout solo llamará la función una única vez.

<body>

<span id="cronometro">0</span>

<br>

<input type="button" id="boton1" value="detener">

<script>

window.addEventListener('DOMContentLoaded', inicio, false);

var reloj;

function inicio() {

reloj=setInterval(procesar,1000);

document.getElementById('boton1').addEventListener('click',presionBoton,false);

}

function procesar()

{

var nro=parseInt(document.getElementById('cronometro').innerHTML);

nro++;

document.getElementById('cronometro').innerHTML=nro;

}

function presionBoton()

{

if (document.getElementById('boton1').value=='detener')

{

clearInterval(reloj); document.getElementById('boton1').value='continuar';

}

else

{

reloj=setInterval(procesar,1000);

document.getElementById('boton1').value='detener';

}

}

</script>

Definimos una variable global que guarde la referencia del timer, en la función inicio registramos la función que debe ejecutarse cada un segundo (1000 milisegundos):

window.addEventListener('DOMContentLoaded', inicio, false);

var reloj;

function inicio() {

reloj=setInterval(procesar,1000);

document.getElementById('boton1').addEventListener('click',presionBoton,false);

}

La función procesar se ejecuta cada un segundo, siempre y cuando se haya registrado esta función llamando a setInterval en la función inicio. El objetivo de esta función es extraer el valor del cronómetro, incrementarlo en uno y actualizarlo:

function procesar()

{

var nro=parseInt(document.getElementById('cronometro').innerHTML);

nro++;

document.getElementById('cronometro').innerHTML=nro;

}

La función presionBoton se ejecuta cada vez que lo presionamos y lo que hacemos es controlar la etiqueta del botón para ver si dice 'detener' con lo que llamando a la función clearInterval esto hace que se suprime la ejecución continua de la función procesar. Por el falso del if procedemos a registrar nuevamente la función procesar:

function presionBoton()

{

if (document.getElementById('boton1').value=='detener')

{

clearInterval(reloj);

document.getElementById('boton1').value='continuar';

}

else

{

reloj=setInterval(procesar,1000);

document.getElementById('boton1').value='detener';

}

}

## ES6

### Definición de constantes y variables:

ES6 o también conocido como ECMAScript 2015 es una actualización del lenguaje de programación JavaScript que actualmente gran cantidad de navegadores ya implementan.

Todos los agregados a JavaScript definidos en este nuevo estándar ya se los está aplicando desde hace un tiempo con Node.js en el servidor, y en framework de cliente como React, Angular y Vue.

Estos agregados al lenguaje JavaScript nos facilitan la programación.

**const**

Hasta que llega ES6 la única forma de definir variables en JavaScript es mediante la palabra clave 'var':

var edad=12;

var sueldo=1000.50;

var titulo='Administración';

Con ES6 si sabemos que la variable nunca cambiará luego en lugar de definir con var debemos definir una constante con la palabra clave 'const':

<script>

const titulo='Administracion';

const empleados=10;

const pi=3.1416;

alert(titulo);

alert(empleados);

alert(pi);

</script>

Todo aquel almacenamiento de datos que no cambiará durante la ejecución del programa es conveniente definirlo como constante.

**let**

Para definir variables que pueden cambiar su contenido podemos seguir utilizando 'var' y 'let'. Veremos cuales son las diferencias entre estos dos formas de definirlas.

Con 'let' se define una variable que tendrá solo alcance al bloque donde se la ha definido, veamos un ejemplo definiendo variables con let:

<script>

//Si el primer valor es mayor o igual al segundo luego sumarlos y restarlos, sin

//efectuar su producto

const v1=parseInt(prompt("Ingrese primer valor"));

const v2=parseInt(prompt("Ingrese segundo valor"));

if (v1>=v2) {

let resu;

resu=v1+v2;

document.write('La suma es:'+resu+'<br>');

resu=v1-v2;

document.write('La resta es:'+resu);

} else {

const producto=v1\*v2;

document.write('El producto es:'+producto);

}

</script>

La variable 'resu' solo se define y reserva espacio si la condición del if se verifica verdadera. Luego al finalizar el bloque del verdadero deja de existir 'resu'.

La ventaja de utilizar let y const es que si accedemos a la variable fuera del bloque de llaves donde se la define se produce un error, ya que no existe. Este error no se produce si utilizamos definiciones de variables con 'var' ya que las mismas siempre se definen como globales.

Entonces 'const' y 'let' definidas dentro de bloques de llaves evitan que se creen variables globales. Esto es una gran ventaja cuando tenemos programas grandes y el manejo de variables locales generan errores difíciles de detectar.

**let y const dentro de funciones**

El mismo concepto se aplica cuando definamos almacenamientos de datos dentro de una función:

* Con let y const se definen espacios de almacenamiento que tienen existencia en el bloque de llaves donde se los definió.
* Con var se reserva espacio y se la puede acceder dentro de cualquier parte de la función.

<script>

function mostrar(x,y) {

for(let avance=x;avance<=y;avance++) {

document.write(avance+' ');

}

document.write(avance); // error variable no definida

}

mostrar(1, 10);

</script>

Si ejecutamos este programa se genera un error en la primer línea fuera del for al tratar de acceder a la variable 'avance', esto debido a que no existe fuera del ámbito del for.

Si utilizamos la palabra clave 'var' y ejecutamos:

function mostrar(x,y) {

document.write(avance);

for(var avance=x;avance<=y;avance++) {

document.write(avance+' ');

}

document.write(avance);

}

Se muestra primero "undefined" ya que la variable 'avance' existe en la función pero no está definida. Luego muestra dentro del for los números del 1 al 10 y finalmente muestra fuera del for el 11.

La mayoría de las veces definiremos variables con la palabra clave 'let', salvo la necesidad de la funcionalidad de 'var'.

**Definición de objetos con 'const' y 'let'**

Cuando definimos por ejemplo un vector con la palabra clave const luego dicha variable no puede referenciar a otro objeto:

const dias=[3,7,12];

dias=[40,22]; //error: asignación inválida

Pero si puede modificar sus componentes:

const dias=[3,7,12];

dias[0]=4;

dias[1]=5;

dias[2]=6;

Lo mismo con una constante de tipo objeto literal podemos cambiar su contenido pero no hacer que apunte a otro objeto:

const empresa = {

nombre: 'Segui Hnos',

empleados: 34

}

empresa.nombre='Condor';

empresa.empleados=39;

Produce un error si ejecutamos:

const empresa = {

nombre: 'Segui Hnos',

empleados: 34

}

empresa = {

nombre:'Condor',

empleados: 39,

fundacion: 1947

} //error

### Plantillas de cadenas de caracteres:

El manejo de cadenas de caracteres en JavaScript tradicional se emplean las comillas simples o dobles:

var titulo1='Administración';

var titulo2="Contabilidad";

Cuando se requieren generar otra cadena formada por varias variables de tipo cadena, numérica etc. normalmente se utiliza el operador + para concatenar.

Con ES6 se incorpora la posibilidad de definir cadenas de caracteres utilizando las comillas invertidas `.

Emplearemos este tipo de cadenas cuando necesitemos sustituir el contenido de variables dentro de la cadena de caracteres sin tener que utilizar el operador de concatenación:

<script>

const valor1=parseInt(prompt('Ingrese primer valor:'));

const valor2=parseInt(prompt('Ingrese segundo valor:'));

document.write(`La suma de ${valor1} y ${valor2} es ${valor1+valor2}`);

</script>

Para interpolar el contenido de una variable dentro de una plantilla de cadena de caracteres encerramos entre llaves la variable y le antecedemos el caracter $

document.write(`La suma de ${valor1} y ${valor2} es ${valor1+valor2}`);

También es importante ver que dentro de las llaves podemos disponer una operación.

Podemos comprobar que es una sintaxis más clara a la de emplear el operador de concatenación

document.write("La suma de "+valor1+" y "+valor2+" es "+(valor1+valor2));

**Llamada a funciones**.

Cuando utilizamos una plantilla de cadenas de caracteres podemos llamar a una función dentro de la plantilla:

<script>

function mayor(v1,v2) {

if (v1>v2)

return v1;

else

return v2;

}

const valor1=parseInt(prompt('Ingrese primer valor:'));

const valor2=parseInt(prompt('Ingrese segundo valor:'));

document.write(`El mayor de ${valor1} y ${valor2} es ${mayor(valor1,valor2)}`);

</script>

**Plantillas de cadenas de caracteres de múltiples líneas**

No se requiere ninguna sintaxis especial si queremos disponer una plantilla de texto de múltiples líneas.

<script>

const dia1='Lunes';

const dia2='Miércoles';

const dia3='Jueves'

const cadena=`

<ol>

<li>${dia1}</li>

<li>${dia2}</li>

<li>${dia3}</li>

</ol>`;

document.write(cadena);

</script>

### Funciones con parámetros por defecto:

Esta característica de JavaScript nos permite asignar un valor por defecto a un parámetro para los casos en que en la llamada a la misma no se le envíe.

function sumar(x1,x2,x3=0)

Luego podemos llamar a la función sumar enviando dos o tres datos:

sumar(10, 2);

sumar(10, 2, 70);

En la primer llamada x3 queda con el valor 0, en la segunda llamada x3 almacena el valor 70.

<script>

function generarLista(vec, tipo='<ul>') {

let lista=tipo;

for(let x=0;x<vec.length;x++)

lista += `<li>${vec[x]}</li>`;

if (tipo=='<ul>')

lista += '</ul>';

else

lista += '</ol>'

return lista;

}

const opciones1=['opción 1','opción 2','opción 3','opción 4','opción 5'];

document.write(generarLista(opciones1));

const opciones2=['opción a','opción b','opción c','opción d'];

document.write(generarLista(opciones2,'<ol>'));

</script>

En la primer llamada le pasamos un solo parámetro:

document.write(generarLista(opciones1));

Luego la función generarLista almacena en el parámetro tipo el valor por defecto: '<ul>':

function generarLista(vec, tipo='<ul>') {

Pero en la segunda llamada le pasamos que queremos obtener una lista HTML ordenada:

document.write(generarLista(opciones2,'<ol>'));

Puede haber varios valores por defecto, pero deben ser los últimos. Es decir primero indicamos los parámetros que reciben datos en forma obligatoria cuando los llamamos y finalmente indicamos aquellos que tienen valores por defecto.

### Funciones con parámetros Rest:

Otra funcionalidad en JavaScript con la actualización ECMAScript 2015 (también llamada ES6) es la posibilidad de pasar una lista indefinida de valores y que los reciba un vector.

El concepto de parámetros Rest se logra antecediendo tres puntos al nombre del parámetro (el parámetro es un objeto de la clase Array con todas sus funcionalidades):

function sumar(...parametro)

Luego en la llamada podemos pasar una lista indeterminada de datos:

sumar(4, 55, 33);

<script>

function sumar(...valores) {

let suma=0;

for(let x=0;x<valores.length;x++)

suma+=valores[x];

return suma;

}

document.write(sumar(10, 2, 44, 3));

document.write('<br>');

document.write(sumar(1, 2));

document.write('<br>');

document.write(sumar());

document.write('<br>');

</script>

El parámetro valores es en realidad un vector y como tal podemos disponer un for para recorrerlo y acceder a sus elementos mediante un subíndice:

function sumar(...valores) {

let suma=0;

for(let x=0;x<valores.length;x++)

suma+=valores[x];

return suma;

}

Cuando llamamos a la función podemos pasar una cantidad variable de enteros para que sean sumados:

document.write(sumar(10, 2, 44, 3));

document.write(sumar(1, 2));

document.write(sumar());

La función con un parámetro Rest puede tener otros parámetros pero se deben declarar antes:

function operar(tipo, ...valores)

Los parámetros Rest no pueden tener un valor por defecto.

### Operador Spread (operador de propagación):

El operador Spread permite descomponer una estructura de datos en elementos individuales. Es la operación inversa de los parámetros Rest.

La sintaxis se aplica anteponiendo al nombre de la variable tres puntos

<script>

function sumar(x,y,z) {

return x+y+z;

}

const vec=[10,20,30];

const s=sumar(...vec);

document.write(s);

</script>

Es decir en la llamada a la función sumar le antecedemos tres puntos al nombre de la variable que vamos a pasar:

const vec=[10,20,30];

const s=sumar(...vec);

Luego el parámetro x recibe el primer componente, el parámetro y la segunda componente y finalmente el parámetro z recibe el tercer componente.

### Arrow functions (funciones flecha):

Las funciones flecha es una nueva sintaxis propuesta en ES6 para expresar las funciones anónimas.

Un ejemplo de definir una arrow function es:

const mayor = (x,y) => {

if (x>y)

return x;

else

return y;

}

document.write(mayor(10,2));

Como es una función anónima si queremos guardar una referencia a la misma se la debemos asignar a una constante o variable. Indicamos entre paréntesis los parámetros de la función luego la flecha => y finalmente entre llaves el algoritmo propiamente de la función.

<script>

const vec = [100, 22, 17, 50, 3];

vec.sort((x,y) => {

if (x>y)

return 1;

else

return -1;

})

document.write(vec.toString());

</script>

A la función sort le pasamos una función flecha.

**Funciones flecha con un único parámetro.**

Cuando una función flecha tiene un parámetro no es necesario encerrarlo entre paréntesis:

const vec = [100, 22, 17, 50, 3];

vec.forEach(elemento => {

document.write(elemento+' ');

})

En este ejemplo el parámetro 'elemento 'es único y por eso no lo encerramos entre paréntesis.

**Funciones flecha de una única línea.**

Podemos definir funciones flecha de una única línea, en dichas situaciones no es necesario encerrar entre llaves el algoritmo.

const doble = x => x\*2;

document.write(doble(10)); //20

document.write(doble(7)); //14

La función tiene un único parámetro llamado x y su algoritmo es multiplicar el valor que llega por 2. La función retorna el resultado sin tener que indicar la palabra clave return.

Otro ejemplo de arrow function con dos parámetros y resuelto con una única expresión:

const mayor = (x, y) => (x>y)?x:y;

document.write(mayor(7, 20)) // 20

document.write(mayor(60, 40)) // 60

Conociendo entonces que podemos definir funciones flecha de una única línea podemos plantear el algoritmo de ordenar un vector con una sintaxis más concisa:

const vec = [100, 22, 17, 50, 3];

vec.sort((x,y) => (x>y)?1:-1);

document.write(vec.toString());

<script>

const vec = [100, 22, 17, 50, 3, 78];

const pares = vec.filter( elemento => elemento%2 == 0);

document.write('Vector completo:'+vec.toString());

document.write('<br>');

document.write('Vector generado con elementos pares:'+pares.toString());

</script>

**Funciones flecha sin parámetros.**

Una función flecha que no tiene parámetros debemos disponer en forma obligatoria los paréntesis abiertos y cerrados previos a la flecha:

<script>

window.addEventListener(

'DOMContentLoaded',

setTimeout( () => alert('pasó 3 segundos desde que se cargó la página'), 3000),

false);

</script>

Registramos en el evento 'DOMContentLoaded' de la página la referencia de la llamada a la función setTimeout pasando a esta función una función de flecha que no tiene parámetros y que tiene por objetivo mostrar una ventana de alerta.

El uso de las arrow functions se está generalizando en la programación en JavaScript.

### Nombres de métodos abreviados:

Con ES6 disponemos una nueva sintaxis para definir nombres de métodos abreviados cuando definimos objetos literales. Veamos la sintaxis anterior con JavaScript:

var operacion = {

valor1: 10,

valor2: 7,

sumar: function() {

return this.valor1+this.valor2;

},

restar: function() {

return this.valor1-this.valor2;

}

}

document.write(operacion.sumar());

document.write('<br>');

document.write(operacion.restar());

La sintaxis abreviada para la definición de los dos métodos es:

const operacion = {

valor1: 10,

valor2: 7,

sumar() {

return this.valor1+this.valor2;

},

restar() {

return this.valor1-this.valor2;

}

}

document.write(operacion.sumar());

document.write('<br>');

document.write(operacion.restar());

Como vemos disponemos el nombre del método, los parámetros y finalmente el algoritmo:

sumar() {

return this.valor1+this.valor2;

},

Desaparecen los dos puntos y la palabra clave function.

### Declaración de clases:

En versiones anteriores de JavaScript si bien se pueden definir clases su sintaxis es poco clara si la comparamos con otros lenguajes.

Con ES6 se incorporó una sintaxis precisa para declarar una clase y sus métodos.

Tenemos una serie de palabras claves para declarar una clase y el constructor de la misma.

<script>

class Cliente {

constructor(nombre, saldo) {

this.nombre=nombre;

this.saldo=saldo;

}

depositar(dinero)

{

this.saldo=this.saldo+dinero;

}

extraer(dinero)

{

this.saldo=this.saldo-dinero;

}

}

const cliente1=new Cliente('diego',1200);

document.write('Nombre del cliente:'+cliente1.nombre+'<br>');

document.write('Saldo actual:'+cliente1.saldo+'<br>');

cliente1.depositar(120);

document.write('Saldo luego de depositar $120---->'+cliente1.saldo+'<br>');

cliente1.extraer(1000);

document.write('Saldo luego de extraer $1000---->'+cliente1.saldo+'<br>');

const cliente2=new Cliente('ana',100);

document.write('Nombre del cliente:'+cliente2.nombre+'<br>');

document.write('Saldo actual:'+cliente2.saldo+'<br>');

cliente2.depositar(200);

document.write('Saldo luego de depositar $200---->'+cliente2.saldo+'<br>');

</script>

Para declarar una clase en ES6 utilizamos la palabra clave class y seguidamente el nombre de la clase:

class Cliente {

Luego entre llaves encerramos todos los métodos de dicha clase.

Tenemos la palabra clave 'constructor' para definir el primer método que se ejecutará al crear el objeto:

constructor(nombre, saldo) {

this.nombre=nombre;

this.saldo=saldo;

}

Los atributos se les debe anteceder la palabra clave this

Los atributos que definamos se tiene acceso en todos los métodos de la clase:

depositar(dinero)

{

this.saldo=this.saldo+dinero;

}

Cuando creamos un objeto de la clase Cliente lo primero que se ejecuta es el constructor, debemos pasar los datos a los parámetros:

const cliente1=new Cliente('diego',1200);

A los demás métodos del objeto 'cliente1' los llamamos antecediendo primero el nombre del objeto:

cliente1.depositar(120);

### Herencia:

Con ES6 también se ha agregado al lenguaje JavaScript sintaxis para facilitar el concepto de herencia entre clases.

Si queremos que una clase herede de otra debemos indicar en el momento que la declaramos mediante la palabra clave 'extends' el nombre de la clase padre:

class Cuadrado extends Figura {

}

<script>

class Persona {

constructor(nombre, edad) {

this.nombre = nombre;

this.edad = edad;

}

imprimir() {

document.write(`Nombre: ${this.nombre}<br>`);

document.write(`Edad: ${this.edad}<br>`);

}

}

class Empleado extends Persona {

constructor(nombre, edad, sueldo) {

super(nombre, edad);

this.sueldo = sueldo;

}

imprimir() {

super.imprimir();

document.write(`Sueldo: ${this.sueldo}<br>`);

}

pagaImpuestos() {

if (this.sueldo>5000)

document.write(`${this.nombre} debe pagar impuestos<br>`);

else

document.write(`${this.nombre} no debe pagar impuestos<br>`);

}

}

const persona1=new Persona('Juan', 44);

persona1.imprimir();

document.write('<hr>');

const empleado1=new Empleado('Juan', 44, 7000);

empleado1.imprimir();

empleado1.pagaImpuestos();

</script>

La declaración de la clase Persona no tiene nada nuevo a lo visto en el concepto anterior, definimos 2 atributos y 2 métodos:

class Persona {

constructor(nombre, edad) {

this.nombre = nombre;

this.edad = edad;

}

imprimir() {

document.write(`Nombre: ${this.nombre}<br>`);

document.write(`Edad: ${this.edad}<br>`);

}

}

La clase Empleado hereda de la clase Persona (es decir que la clase empleado tiene como atributos su nombre, edad un método donde se inicializan dichos atributos y uno que los muestra):

class Empleado extends Persona {

Mediante la sintaxis de extends indicamos que Empleado hereda de Persona.

En el constructor recibe como parámetros el nombre del empleado, su edad y el sueldo:

constructor(nombre, edad, sueldo) {

super(nombre, edad);

this.sueldo = sueldo;

}

Los atributos del nombre y edad se los pasamos al constructor de la clase padre mediante la palabra clave 'super' (es obligatorio hacer esto primero sino se genera un error sintáctico)

Lo mismo en el método imprimir del Empleado llamamos al imprimir de la clase padre mediante la palabra clave super para que se muestren el nombre y edad del empleado:

imprimir() {

super.imprimir();

document.write(`Sueldo: ${this.sueldo}<br>`);

}

No es obligatorio llamar primero al imprimir de la clase Empleado, podemos primero mostrar el sueldo y luego imprimir el nombre y la edad.

Finalmente en el método pagaImpuestos controlamos el contenido del atributo 'this.sueldo' para mostrar un mensaje:

pagaImpuestos() {

if (this.sueldo>5000)

document.write(`${this.nombre} debe pagar impuestos<br>`);

else

document.write(`${this.nombre} no debe pagar impuestos<br>`);

}

### Definición de getters y setters:

En JavaScript disponemos ahora de la posibilidad de definir setters que son métodos que establecen un valor a una propiedad y getters que son métodos que rescatan un valor.

Para la definición de setters y getters disponemos de las palabras claves set y get.

La definición de propiedades con set y get nos permiten un mejor encapsulamiento de nuestra clase.

<script>

class Dado {

constructor() {

this.\_valor=1;

}

get valor() {

return this.\_valor;

}

set valor(v) {

this.\_valor=v;

}

imprimir() {

document.write(this.valor+'<br>');

}

}

const dado1=new Dado();

dado1.imprimir();

dado1.valor=6;

dado1.imprimir();

</script>

En el constructor definimos un atributo llamado \_valor que almacena el valor 1:

constructor() {

this.\_valor=1;

}

Definimos la propiedad 'valor' con sus respectivos 'set' y 'get':

get valor() {

return this.\_valor;

}

set valor(v) {

this.\_valor=v;

}

El método get retorna el valor almacenado en el atributo '\_valor' y el método set actualiza el valor almacenado en el atributo '\_valor'.

Donde definimos un objeto de la clase Dado accedemos a la propiedad dado para cambiar su valor por medio de una asignación:

dado1.valor=6;

Es decir que en dicha asignación se está ejecutando el método:

set valor(v) {

this.\_valor=v;

}

En principio podría pensarse que lo más fácil es acceder directamente al atributo \_valor y olvidarnos del método set:

dado1.\_valor=6;

Pero si queremos luego mejorar la clase 'Dado' y hacerla más robusta, por ejemplo no dejar cargar valores inválidos para un dado podemos mejorar el método 'set' con la siguiente sintaxis:

set valor(v) {

if (v>=1 && v<=6 && v%1===0)

this.\_valor=v;

else

throw "Error en asignación de valor del dado";

}

Con ésta nueva implementación del método set solo almacenaremos valores válidos en el atributo \_valor si llega un número entero comprendido entre 1 y 6. En el caso que llegue un valor con coma 1.4 que está comprendido entre 1 y 6 luego no cumple la tercer condición que el resto de dividirlo sea cero.

Cualquiera de estas asignaciones detiene el programa con un error:

dado1.valor=7;

dado1.valor=0;

dado1.valor=3.4;

dado1.valor='Hola';

Ahora podemos comprobar la ventaja de definir propiedades de acceso a una clase en lugar de acceder directamente a sus atributos.

Si accedemos directamente al atributos '\_valor' luego no se genera un error si le asignamos:

dado1.\_valor=3.4;

Esto hace que nuestro programa sea más difícil de mantener y propenso a errores.

Vimos que llamamos al método set mediante una asignación, al método get lo llamamos directamente por su nombre:

const dado1=new Dado();

dado1.valor=5;

document.write(dado1.valor);

Cuando llamamos al método write del objeto document estamos accediendo al método get del objeto dado1.

### Definición de métodos estáticos:

Con esta nueva actualización de JavaScript tenemos la posibilidad de definir métodos estáticos en una clase.

Un método estático pertenece a la clase y no a una instancia u objeto de una clase. Para llamar luego a un método estático lo hacemos directamente antecediendo el nombre de la clase.

Se las utiliza para definir funcionalidades que pertenecen a una clase pero que no dependan de instancias de dicha clase.

Los métodos estáticos no pueden acceder a los atributos de la clase mediante this.

<script>

class Matematica {

static mayor(v1,v2) {

if (v1>v2)

return v1;

else

return v2;

}

static menor(v1,v2) {

if (v1<v2)

return v1;

else

return v2;

}

static aleatorio(inicio, fin) {

return Math.trunc((Math.random()\*(fin+1-inicio))+inicio);

}

}

let x1=Matematica.aleatorio(1,10);

let x2=Matematica.aleatorio(1,10);

document.write(`El mayor entre ${x1} y ${x2} es ${Matematica.mayor(x1,x2)} <br>`);

document.write(`El menor entre ${x1} y ${x2} es ${Matematica.menor(x1,x2)}`);

</script>

Utilizamos la palabra clave 'static' para definir éste tipo de métodos:

static mayor(v1,v2) {

if (v1>v2)

return v1;

else

return v2;

}

Un método estático normalmente implementa un algoritmo para procesar los datos que llegan como parámetro.

Para la llamada a un método estático lo hacemos antecediendo el nombre de la clase y no un objeto de la clase:

let x1=Matematica.aleatorio(1,10);

### Estructura repetitiva 'for of':

Con ES6 disponemos una nueva estructura repetitiva para recorrer todos los elementos de un array, string, map, set etc.

En forma muy sencilla podemos acceder por ejemplo a cada caracter de un string:

const cadena='Hola Mundo';

for(let letra of cadena)

document.write(`${letra}<br>`);

Como resultado de ejecutar éste algoritmo de JavaScript tenemos la impresión de cada caracter en una línea distinta en la página.

Podemos recorrer también un array:

const vector=[10, 40, 60, 5];

let suma=0;

for(let elemento of vector)

suma+=elemento;

document.write(`La suma de todos los elementos del vector es:${suma}`);

<script>

const personas = [

{

nombre:'juan',

edad:45

},

{

nombre:'ana',

edad:12

},

{

nombre:'luis',

edad:16

},

{

nombre:'cristina',

edad:76

}

];

document.write('<h1>Personas mayores de edad.</h1>');

for(let persona of personas)

if (persona.edad>=18)

document.write(`${persona.nombre} y tiene ${persona.edad} años.<br>`);

</script>

Mediante un for of recorremos cada componente del array. En la variable persona se almacena en cada repetición del for of un elemento del array:

for(let persona of personas)

if (persona.edad>=18)

document.write(`${persona.nombre} y tiene ${persona.edad} años.<br>`);

**Recuperar el valor y el índice que ocupa un elemento dentro del arreglo.**

Imprimir todos los elementos de un array y la posición que ocupa dentro del mismo:

<script>

const vector=[10, 40, 60, 5];

for(let [indice, valor] of vector.entries())

document.write(`${valor} ocupa la posición ${indice}<br>`);

</script>

El método entries() devuelve un arreglo con dos valores: la posición del elemento y el valor del mismo:

for(let [indice, valor] of vector.entries())

### Módulos:

Un módulo es un archivo de JavaScript que agrupa funciones, clases, variables que luego pueden ser exportadas y utilizadas en otras partes de nuestra aplicación. Un módulo permite ocultar funcionalidad del mismo y solo exportar aquello para lo que ha sido implementado.

Esta característica agregada a JavaScript nos permite implementar programas mucho más ordenados y facilita la reutilización del código.

**export e import**

Veamos un ejemplo donde implementamos un módulo y luego lo consumimos:

modulo1.js

export function sumar(x,y) {

return x+y;

}

export function restar(x,y) {

return x-y;

}

El módulo consta de 2 funciones exportadas, podría haber dentro del módulo otras funciones, clases, variables etc. sin la palabra export que solo se pueden acceder dentro del módulo.

Para consumir un módulo desde otro módulo o una página HTML debemos utilizar la palabra clave import:

<script type="module">

import {sumar, restar} from './modulo1.js';

alert(sumar(3,8));

alert(restar(10,3));

</script>

Luego de la palabra clave import indicamos entre llaves los nombres de las funciones, clases, variables, constantes etc. que importamos, también debemos indicar entre comillas el nombre del módulo y el path o dirección donde se almacena (con ./ indicamos que se encuentra en la misma carpeta):

import {sumar, restar} from './modulo1.js';

Una vez importado los recursos del paquete los podemos acceder por su nombre:

alert(sumar(3,8));

Es muy importante indicarle al navegador que estamos utilizando la nueva tecnología de módulos asignando a la propiedad type el valor de 'module':

<script type="module">

Para exportar recursos de un módulo podemos utilizar otra sintaxis:

function sumar(x,y) {

return x+y;

}

function restar(x,y) {

return x-y;

}

export {sumar, restar};

En un principio se definen dos funciones privadas al módulo y luego mediante la palabra clave export y entre llaves indicamos los nombres de las funciones a exportar.

**default export**

Es una práctica común en JavaScript crear módulos que exporten un único valor. En estos casos tenemos una sintaxis distinta para indicar el recurso a exportar mediante las palabras claves 'default export'.

Veamos un ejemplo donde almacenamos en un módulo la clase Dado y la exportamos por defecto:

dado.js

export default class Dado {

constructor() {

this.tirar();

}

get valor() {

return this.\_valor;

}

tirar() {

this.\_valor=Math.trunc(Math.random()\*6)+1;

}

}

Luego cambia la sintaxis cuando tenemos que importar la clase:

<script type="module">

import Dado from './dado.js';

const dado1 = new Dado();

alert(dado1.valor);

dado1.tirar();

alert(dado1.valor);

</script>

No debemos indicar entre llaves el recurso a importar (si las disponemos se genera un error sintáctico):

import Dado from './dado.js';

Otra práctica común es no indicar el nombre de la clase en el módulo, luego el nombre del módulo representa la clase:

export default class {

constructor() {

this.tirar();

}

get valor() {

return this.\_valor;

}

tirar() {

this.\_valor=Math.trunc(Math.random()\*6)+1;

}

}

Hemos indicado export default para la clase, luego el archivo se llama 'dado.js' y representa dicha clase. No hay cambios para hacer uso de la clase:

import Dado from './dado.js';

**export y default export en el mismo módulo**

Un módulo solo puede exportar por defecto un solo recurso pero puede tener otros recursos que se exportan. Veamos un ejemplo:

export default class{

constructor() {

this.tirar();

}

get valor() {

return this.\_valor;

}

tirar() {

this.\_valor=Math.trunc(Math.random()\*6)+1;

}

}

export const lados=6;

En éste módulo se exporta por defecto la clase y además se exporta una constante llamada lados.

Para consumir los dos valores exportados por este módulo tenemos la siguiente sintaxis:

<script type="module">

import Dado, {lados} from './dado.js';

const dado1 = new Dado();

alert('valor del dado:'+dado1.valor);

alert('lados del dado:'+lados);

</script>

Importamos la clase por defecto y la constante lados:

import Dado, {lados} from './dado.js';

**Importar todos los recursos definidos con export.**

Si un módulo tiene varios recursos definidos con la palabra clave export luego los podemos importar a todos juntos utilizando la siguiente sintaxis:

<script type="module">

import \* as operacion from './modulo1.js';

alert(operacion.sumar(3,8));

alert(operacion.restar(10,3));

</script>

Importamos el módulo1.js que tiene dos funciones export y definimos luego de la palabra clave as el nombre de un objeto que tendrá como propiedades los recursos expotados por dicho módulo.

Llamamos ahora a cada función antecediendo el nombre del objeto creado:

alert(operacion.sumar(3,8));

**Importar un valor por defecto definiendo un alias.**

Otra sintaxis para importar un recurso definido con 'export default' es mediante la sintaxis:

<script type="module">

import {default as Dado, lados} from './dado.js';

const dado1 = new Dado();

alert('valor del dado:'+dado1.valor);

alert('lados del dado:'+lados);

</script>

Utilizamos la palabra clave default y seguidamente el nombre o alias que se asigna al recurso importado (debe ir entre llaves como cuando importamos valores definidos con 'export'):

import {default as Dado, lados} from './dado.js';

**Reexportar un recurso importado en un módulo.**

Un módulo que importa un recurso puede reexportar el mismo. Veamos un ejemplo con dos módulos y el archivo html:

dado.js

export default class{

constructor() {

this.tirar();

}

get valor() {

return this.\_valor;

}

tirar() {

this.\_valor=Math.trunc(Math.random()\*6)+1;

}

}

export const lados=6;

juegodedados.js

import Dado, {lados} from './dado.js'

export class JuegoDeDados {

constructor() {

this.dado1=new Dado();

this.dado2=new Dado();

this.dado3=new Dado();

}

sumaDados() {

return this.dado1.valor+this.dado2.valor+this.dado3.valor;

}

}

export {lados};

El módulo 'juegodedados.js' importa la clase por defecto 'Dado' y la constante 'lados'. Luego este módulo exporta la clase JuegoDeDados y reexporta 'lados':

export {lados};

pagina1.html

<script type="module"

import {JuegoDeDados, lados} from './juegodedados.js';

const juego=new JuegoDeDados();

alert('suma de los tres dados:'+juego.sumaDados());

alert('lados del dado:'+lados)

</script>

Como vemos solo importamos el módulo 'juegodedados.js' y recuperamos la clase JuegoDeDados y lados:

import {JuegoDeDados, lados} from './juegodedados.js';

No hace falta importar el módulo 'dado.js' para acceder a la constante 'lados'.

Si en el módulo 'juegodedados.js' no haremos uso de la constante 'lados' pero queremos reexportar la misma podemos utilizar ésta otra sintaxis:

import Dado from './dado.js'

export class JuegoDeDados {

constructor() {

this.dado1=new Dado();

this.dado2=new Dado();

this.dado3=new Dado();

}

sumaDados() {

return this.dado1.valor+this.dado2.valor+this.dado3.valor;

}

}

export {lados} from './dado.js';

### Map:

Con la llegada de ES6 se presenta un nuevo objeto llamado 'Map' que nos evita tener que implementar desde cero una estructura de datos tipo diccionario o mapa.

La estructura de datos tipo 'Map' utiliza una clave para acceder a un valor.

Podemos relacionarlo con conceptos que conocemos:

Un diccionario tradicional que conocemos podemos utilizar un 'Map' para representarlo. La clave sería la palabra y el valor sería la definición de dicha palabra.

Una agenda personal también la podemos representar como un diccionario. La fecha sería la clave y las actividades de dicha fecha sería el valor.

Un conjunto de usuarios de un sitio web podemos almacenarlo en un diccionario. El nombre de usuario sería la clave y como valor podríamos almacenar su mail, clave, fechas de login etc.

Como clave se puede utilizar cualquier tipo de dato primitivo como también de tipo objeto. Podemos almacenar como valor también datos primitivos como objetos.

**Creación de un Map.**

Podemos crear un Map vacío y posteriormente añadir elementos al mapa mediante el método 'set':

<script>

const mapa1 = new Map();

mapa1.set("casa", "house");

mapa1.set("rojo", "red");

mapa1.set("auto", "car");

console.log(mapa1);

</script>

Podemos crear un Map e inmediatamente pasar los datos iniciales en el constructor del mismo:

<script>

const mapa1 = new Map([

["casa", "house"],

["rojo", "red"],

["auto", "car"]

]);

console.log(mapa1);

</script>

**Recuperar el valor para una determinada clave.**

Para recuperar un valor para una determinada clave del 'Map' disponemos del método 'get'.

<script>

const mapa1 = new Map([

["casa", "house"],

["rojo", "red"],

["auto", "car"]

]);

const palcastellano = prompt("Ingrese una palabra en castellano:", "");

document.write(`La traducción de ${palcastellano} es ${mapa1.get(palcastellano)}`);

</script>

Como podemos comprobar al ejecutar el algoritmo si ingresamos una palabra que existe en el 'Map' luego el método get nos retorna el valor para dicha palabra. Si ingresamos una palabra que no existe el método nos retorna el valor 'undefined'.

Disponemos de otro método en el objeto 'Map' llamado 'has' que le debemos pasar como parámetro una clave y en el caso que exista nos retorna true, en caso negativo nos retorna false. Luego podemos modificar nuestra aplicación para que nos muestre un mensaje en el caso que no exista la traducción de la palabra ingresada:

<script>

const mapa1 = new Map([

["casa", "house"],

["rojo", "red"],

["auto", "car"]

]);

const palcastellano = prompt("Ingrese una palabra en castellano:", "");

if (mapa1.has(palcastellano))

document.write(`La traducción de ${palcastellano} es ${mapa1.get(palcastellano)}`);

else

document.write(`No existe una traducción para la palabra ${palcastellano}`)

</script>

**Tamaño del Map.**

Mediante la propiedad 'size' podemos conocer la cantidad de entradas almacenadas en el mapa:

<script>

const mapa1 = new Map([

["casa", "house"],

["rojo", "red"],

["auto", "car"]

]);

mapa1.set("ventana","window");

console.log(mapa1.size); // 4

</script>

**Eliminación de elementos del Map.**

Podemos eliminar elementos mediante el método 'delete':

<script>

const mapa1 = new Map([

["casa", "house"],

["rojo", "red"],

["auto", "car"]

]);

mapa1.delete("casa");

console.log(mapa1.size); // 2

</script>

El método 'delete' retorna true si se eliminó la entrada en el Map y false en caso que le hayamos pasado una clave que no exista.

Para eliminar todos los elementos del mapa disponemos del método 'clear':

<script>

const mapa1 = new Map([

["casa", "house"],

["rojo", "red"],

["auto", "car"]

]);

mapa1.clear();

console.log(mapa1.size); // 0

</script>

**Recorrido de un Map.**

Los elementos almacenado en el 'Map' mantienen el orden en que se agregaron.

Podemos recorrer un mapa mediante la estructura repetitiva 'for of':

<script>

const mapa1 = new Map([

["casa", "house"],

["rojo", "red"],

["auto", "car"]

]);

document.write("Claves del mapa.<br>")

for (let clave of mapa1.keys()) {

document.write(clave);

document.write("<br>");

}

document.write("Valores del mapa.<br>")

for (let valor of mapa1.values()) {

document.write(valor);

document.write("<br>");

}

document.write("Claves y valores del mapa.<br>")

for (let [clave, valor] of mapa1) {

document.write(clave + '-' + valor);

document.write("<br>");

}

</script>

El objeto Map igual que el objeto Array dispone de un método 'forEach' para iterar sobre los elementos del 'Map':

<script>

const mapa1 = new Map([

["casa", "house"],

["rojo", "red"],

["auto", "car"]

]);

document.write("Claves y valores del mapa.<br>")

mapa1.forEach((valor, clave) => document.write(`${clave} ${valor}<br>`));

</script>

### Set:

Hemos visto en conceptos anteriores que JavaScript implementa los siguientes objetos:

* Array
* Map

Con ES6 se presenta el objeto llamado 'Set' que nos evita tener que implementar desde cero una estructura de datos tipo conjunto.

El objeto 'Set' permite almacenar una colección de elementos que pueden ser de tipo primitivo u objeto y que no se pueden repetir.

**Creación de un conjunto (Set).**

Podemos crear un Set vacío y posteriormente añadir elementos al conjunto mediante el método 'add':

<script>

const conjunto1 = new Set();

conjunto1.add("C");

conjunto1.add("Pascal");

conjunto1.add("PHP");

conjunto1.add("Python");

console.log(conjunto1);

</script>

Podemos crear un Set e inmediatamente pasar los datos iniciales en el constructor del mismo mediante un arreglo:

<script>

const conjunto1 = new Set(["C", "Pascal", "PHP", "Python"]);

console.log(conjunto1);

</script>

**Verificar la existencia de un determinado valor y la cantidad de valores del Set.**

Disponemos del método 'has' que nos retorna si el conjunto almacena un determinado valor.

<script>

const conjunto1 = new Set();

for (let x = 0; x < 10; x++)

conjunto1.add(parseInt(Math.random() \* 50) + 1);

document.write(`La cantidad de números aleatorios distintos son ${conjunto1.size}<br>`)

if (conjunto1.has(12))

document.write("Si se generó el número 12")

else

document.write("No se generó el número 12")

</script>

Como un conjunto no puede almacenar valores repetidos luego si queremos saber la cantidad de números aleatorios distintos debemos acceder a la propiedad 'size' del 'Set':

document.write(`La cantidad de números aleatorios distintos son ${conjunto1.size}<br>`)

El método 'has' retorna true si el conjunto almacena el dato que le pasamos como parámetro:

if (conjunto1.has(12))

**Eliminación de elementos del Set.**

Podemos eliminar elementos mediante el método 'delete':

<script>

const conjunto1 = new Set(["C", "Pascal", "PHP", "Python"]);

console.log(conjunto1.size); // 4

conjunto1.delete("Pascal");

console.log(conjunto1.size); // 3

</script>

El método 'delete' retorna true si se eliminó la entrada en el Set y false en caso que le hayamos pasado un valor que no exista.

Para eliminar todos los elementos del conjunto disponemos del método 'clear':

<script>

const conjunto1 = new Set(["C", "Pascal", "PHP", "Python"]);

console.log(conjunto1.size); // 4

conjunto1.clear();

console.log(conjunto1.size); // 0

</script>

**Recorrido de un Set.**

Los elementos almacenado en el 'Set' mantienen el orden en que se agregaron.

Podemos recorrer un 'Set' mediante la estructura repetitiva 'for of':

<script>

const conjunto1 = new Set(["C", "Pascal", "PHP", "Python"]);

for(let elemento of conjunto1)

document.write(`${elemento}<br>`);

</script>

Al ejecutar el algoritmo podemos ver como recorrer el conjunto y recuperar en forma individual cada uno de los elementos que conforman el 'Set'.

El objeto Set igual que los objetos Array y Map dispone de un método 'forEach' para iterar sobre los elementos del 'Set':

<script>

const conjunto1 = new Set(["C", "Pascal", "PHP", "Python"]);

mapa1.forEach((elemento) => document.write(`${elemento}<br>`));

</script>

### WeakMap y WeakSet :

Son dos objetos con funcionalidades similares a Map y Set pero con la capacidad de que el recolector de basura de JavaScript elimine elementos de los mismos cuando lo considere oportuno.

**WeakSet:** Solo pueden almacenar datos de tipo objeto (es lógico ésto debido a que los tipos de datos primitivos no pueden tener la referencia de más de una variable)

Las referencias a objetos en la colección de tipo WeakSet se mantienen débilmente, ésto significa que si en algún momento durante la ejecución del algoritmo se deja de tener referencias del objeto, luego el recolector de basura de JavaScript libera espacio de la colección WeakSet.

const conjunto1 = new WeakSet();

let usuario1 = {

nombre: "juan",

puntos: 200

};

conjunto1.add(usuario1);

console.log(conjunto1.has(usuario1)); //true

usuario1 = null;

....

Una vez que la variable 'usuario1' se elimina al asignar el valor null, el "garbage collector" (recolector de basura) de JavaScript se encarga de eliminar el elemento del objeto conjunto1.

**WeakMap** Solo se puenden almacenar en la clave del mapa un valor de tipo objeto. JavaScript se encarga de eliminar elementos del mapa cuando se dejan de tener referencias al objeto que actúa como clave dentro del WeakMap.

**Acotaciones**

Los objetos WeakMap y WeakSet no disponen la capacidas de recorrer sus elementos mediante la estructura for of, tampoco podemos conocer la cantidad de elementos.

Podemos eliminar un elemento de la colección mediante el método 'delete', pero no dispone de un método para borrar todos los elementos.

El uso de estos objetos tiene sentido cuando sabemos que durante la ejecución del algoritmo se eliminarán muchos objetos que actúan como claves, en caso contrario es conveniente emplear los objetos Map y Set.

### Array:

El objeto Array existe desde la primer versión de JavaScript, pero a lo largo del tiempo ha ido incorporando nuevas funcionalidades.

Veremos los métodos que ha incorporado el objeto Array con la versión de ES6, recordando que todos los navegadores actuales los implementan en su totalidad.

**Método estático 'from'.**

El método estático 'from' retorna la referencia de un objeto de tipo Array que se crea a partir del dato que le pasamos:

<script>

const arreglo1 = [10, 20, 30];

const arreglo2 = Array.from(arreglo1);

arreglo1.fill(5);

console.log(arreglo1); // [5,5,5]

console.log(arreglo2); // [10,20,30]

</script>

Creamos un arreglo con 3 elementos:

const arreglo1 = [10, 20, 30];

Luego llamamos al método 'from' del objeto 'Array', el cual nos retorna un nuevo arreglo con el mismo contenido de 'arreglo1'.

Es importante recordar que los objetos en JavaScript (y los arreglos son objetos) almacenan la referencia del objeto y no los datos propiamente dichos como los hacen las variables de tipo primitiva. No es lo mismo hacer la asignación:

const arreglo2=arreglo1;

Con la asignación previa sigue existiendo un solo objeto y luego dos referencias (arreglo1 y arreglo2) a dicho objeto.

Si utilizamos la asignación tenemos como resultado:

<script>

const arreglo1 = [10, 20, 30];

const arreglo2 = arreglo1;

arreglo1.fill(5);

console.log(arreglo1); // [5,5,5]

console.log(arreglo2); // [5,5,5]

</script>

Como podemos comprobar si imprimimos el único arreglo ya sea con la referencia arreglo1 o arreglo2 luego se acceden al mismo contenido.

Luego veremos que el método 'fill' cambia el contenido de un arreglo con el dato que le pasamos como parámetro.

Al método 'from' podemos pasar cualquier objeto que tenga la capacidad de iterar (más adelande veremos como crear objetos iterables). Por ejemplo un string es iterable:

<script>

const titulo = "administración";

const arreglo1 = Array.from(titulo);

console.log(arreglo1); // ["a", "d", "m", "i", "n", "i", "s", "t", "r", "a", "c", "i", "ó", "n"]

console.log(arreglo1.length); // 14

</script>

Un objeto de tipo Map es iterable:

<script>

const mapa1 = new Map();

mapa1.set("casa", "house");

mapa1.set("rojo", "red");

mapa1.set("auto", "car");

const arreglo1 = Array.from(mapa1);

console.log(arreglo1); // [["casa", "house"], ["rojo", "red"], ["auto", "car"]]

console.log(arreglo1.length); // 3

</script>

Un objeto de tipo Set es iterable:

<script>

const conjunto1 = new Set();

conjunto1.add("C");

conjunto1.add("Pascal");

conjunto1.add("PHP");

conjunto1.add("Python");

const arreglo1 = Array.from(conjunto1);

console.log(arreglo1); // ["C", "Pascal", "PHP", "Python"]

console.log(arreglo1.length); // 4

</script>

Por ejemplo no son iterables los objetos WeakMap, WeapSet, Object etc.

El método 'from' tiene opcionalmente un segundo parámetro al cual podemos pasar una función que reciba cada elemento del arreglo y retorne el dato a almacenar:

const arreglo1 = [10, 20, 30];

const arreglo2 = Array.from(arreglo1, (elemento) => elemento \* 2);

console.log(arreglo2); // [20, 40, 60]

</script>

La función anonima retorna que valor debe almacenar en el arreglo que se crea, en éste caso almacenamos los valores originarios del arreglo1 pero multiplicados por 2.

Recordemos que las funciones flecha son ámpliamente utilizadas a partir de ES6 ya que nos permiten generar un código muy conciso, utilizando la sintaxis anterior tenemos:

<script>

const arreglo1 = [10, 20, 30];

const arreglo2 = Array.from(arreglo1, function (elemento) { return elemento \* 2 });

console.log(arreglo2); // [20, 40, 60]

</script>

**Método estático 'of'.**

El método estático 'of' retorna la referencia de un objeto de tipo Array a partir de una lista de valores que le pasamos a dicho método:

<script>

const arreglo1 = Array.of(10, 20, 30, 40);

console.log(arreglo1); // [10, 20, 30, 40]

</script>

**Métodos 'keys', 'values' y 'entries'.**

El objeto Array implementan los métodos que permiten iterar sobre sus claves y valores:

<script>

const lenguajes = ["domingo", "lunes", "martes", "miércoles",

"jueves", "viernes", "sábado"];

for (let indice of lenguajes.keys())

console.log(indice); // 0 1 2 3 4 5 6

for (let valor of lenguajes.values())

console.log(valor); // "domingo" "lunes" "martes" "miércoles"

// "jueves" "viernes" "sábado"

for (let componente of lenguajes.entries())

console.log(componente); // [0, "domingo"] [1, "lunes"] [2, "martes"]

// [3, "miércoles"] [4, "jueves"]

// [5, "viernes"] [6, "sábado"] [7, "domingo"]

</script>

En un solo 'for of' podemos recuperar cada componente y guardarlo en un Array el índice por un lado y el valor por otro:

for (let [indice,valor] of lenguajes.entries())

console.log(indice,valor);

Método 'fill'.

El método 'fill' permite cambiar el contenido de todas o algunas componentes de un arreglo existente. En ningún caso permite redimensionarlo.

Veamos un ejemplo y los efectos del método 'fill':

<script>

const arreglo1 = [9, 9, 9, 9, 9, 9, 9];

// fijamos el valor 3 a todas las componentes del arreglo

arreglo1.fill(3);

console.log(arreglo1); // [3, 3, 3, 3, 3, 3, 3]

// fijamos el valor 0 desde la componente 3 del arreglo

// hasta el final

arreglo1.fill(0, 3);

console.log(arreglo1); // [3, 3, 3, 0, 0, 0, 0]

// fijamos el valor 5 a todas las componentes

arreglo1.fill(5);

console.log(arreglo1); // [5, 5, 5, 5, 5, 5, 5]

// fijamos el valor 5 desde la componente 3 hasta

// la componente 5 sin incluirla

arreglo1.fill(100, 3, 5);

console.log(arreglo1); // [5, 5, 5, 100, 100, 5, 5]

// fijamos el valor 5 a todas las componentes

arreglo1.fill(5);

console.log(arreglo1); // [5, 5, 5, 5, 5, 5, 5]

// fijamos el valor 100 en las dos últimas componentes

arreglo1.fill(100, -2);

console.log(arreglo1); // [5, 5, 5, 5, 5, 100, 100]

// fijamos el valor 5 a todas las componentes

arreglo1.fill(5);

console.log(arreglo1); // [5, 5, 5, 5, 5, 5, 5]

// fijamos el valor 100 en la antepenúltima y penúltima

// posición

arreglo1.fill(100, -3, -1);

console.log(arreglo1); // [5, 5, 5, 5, 100, 100, 5]

</script>

Si indicamos un solo valor se modifican todas las componentes del arreglo:

arreglo1.fill(3);

Podemos indicar desde que posición se comienza a modificar:

// fijamos el valor 0 desde la componente 3 del arreglo

// hasta el final

arreglo1.fill(0, 3);

También podemos indicar desde y hasta que componente se deben modificar:

// fijamos el valor 5 desde la componente 3 hasta

// la componente 5 sin incluirla

arreglo1.fill(100, 3, 5);

Finalmente podemos indicar con valores negativos comenzar desde el final del arreglo:

// fijamos el valor 100 en las dos últimas componentes

arreglo1.fill(100, -2);

console.log(arreglo1); // [5, 5, 5, 5, 5, 100, 100]

// fijamos el valor 100 en la antepenúltima y penúltima

// posición

arreglo1.fill(100, -3, -1);

console.log(arreglo1); // [5, 5, 5, 5, 100, 100, 5]

**Método 'copyWithin'.**

El método 'copyWithin' permite copiar algunas componentes de un arreglo en otra parte del mismo arreglo. En ningún caso permite redimensionarlo.

<script>

let arreglo1 = [1, 2, 3, 4, 5, 6, 7, 8, 9];

// Copiar a partir de la posición cero del arreglo

// los elementos comprendidos entre las posiciones

// 5 y 8

arreglo1.copyWithin(0, 5, 8);

console.log(arreglo1); // [6, 7, 8, 4, 5, 6, 7, 8, 9]

arreglo1 = [1, 2, 3, 4, 5, 6, 7, 8, 9];

// Copiar a partir de la posición cero del arreglo

// los elementos comprendidos desde la posición 5

// hasta el final

arreglo1.copyWithin(0, 5);

console.log(arreglo1); // [6, 7, 8, 9, 5, 6, 7, 8, 9]

arreglo1 = [1, 2, 3, 4, 5, 6, 7, 8, 9];

// Copiar a partir de la posición cero los dos últimos

// elementos del arreglo

arreglo1.copyWithin(0, -2);

console.log(arreglo1); // [8, 9, 3, 4, 5, 6, 7, 8, 9]

</script>

El método 'copyWithin' es muy eficiente cuando debemos copiar un bloque de un arreglo a otra parte dentro del mismo arreglo.

**Búsquedas de valores e índices en un arreglo: indexOf, lastIndexOf e include.**

Disponemos de una serie de métodos para recuperar el índice donde se encuentra un elemento en un arreglo. También mediante el método 'include' podemos identificar si un determinado valor está contenido en el Array.

<script>

let arreglo1 = [10, 20, 30, 40, 50, 60, 10, 20, 30];

// Indice donde se almacena el valor 30

console.log(arreglo1.indexOf(30)); // 2

// Indice donde se almacena un valor inexistente

console.log(arreglo1.indexOf(100)); // -1

// Indice donde se almacena el valor 20

// buscándo desde el final

console.log(arreglo1.lastIndexOf(20)); // 7

// Indice donde se almacena el valor 10

// comenzando la búsqueda en la posición 5

console.log(arreglo1.indexOf(10, 5)); // 6

// Existe el valor 50 en el arreglo?

console.log(arreglo1.includes(50)); // true

// Existe el valor 100 en el arreglo?

console.log(arreglo1.includes(100)); // false

</script>

Al método includes también podemos pasar un segundo parámetro que indica a partir de que posición dentro del arreglo comenzar a buscar:

// Existe el valor 100 en el arreglo a partir de

// la posición 2?

console.log(arreglo1.includes(100, 2));

**Métodos 'find' y 'findIndex'.**

Estos dos métodos nos permiten enviar una función anónima donde definimos el algoritmo de la búsqueda:

<script>

const personas = [

{

nombre: "pedro",

edad: 34

},

{

nombre: "ana",

edad: 54

},

{

nombre: "carlos",

edad: 12

},

{

nombre: "maría",

edad: 22

},

{

nombre: "oscar",

edad: 72

}

];

let per = personas.find((persona, indice) => persona.nombre == "maría");

if (per != undefined)

console.log(per);

else

console.log("No se encuentra maría en el vector de objetos");

let indice = personas.findIndex((persona, indice) => persona.nombre == "maría");

if (indice != -1)

console.log("Se encuentra en la posción "+indice+" del arreglo.");

else

console.log("No se encuentra maría en el vector de objetos");

</script>

El método find retorna el valor 'undefined' si no retorna true el algoritmo dispuesto en la función anónima. En el caso que se encuentra el valor buscado luego retorna el valor de la posición donde se lo ubicó.

Finalmente el método findIndex si la función anónima retorna true luego el método retorna la posición donde se encontró el valor a buscar.

Estamos utilizando funciones flecha, pero nada nos impide de utilizar la sintaxis de funciones anónimas tradicionales:

let per = personas.find(function (persona, indice) { return persona.nombre == "maría" });

if (per != undefined)

console.log(per);

else

console.log("No se encuentra maría en el vector de objetos");

let indice = personas.findIndex(function (persona, indice) { return persona.nombre == "maría" });

if (indice != -1)

console.log("Se encuentra en la posción " + indice + " del arreglo.");

else

console.log("No se encuentra maría en el vector de objetos");

**Métodos every y some.**

Al método 'every' debemos pasar una función anónima que procesa cada elemento del arreglo. Si todos los elementos cumplen la condición impuesta dentro de la función, luego el método 'every' retorna true.

En cambio con el método 'some' con que un elemento cumpla la condición impuesta luego el método retorna true.

every (todos) some (alguno)

<script>

const personas = [

{

nombre: "pedro",

edad: 34

},

{

nombre: "ana",

edad: 54

},

{

nombre: "carlos",

edad: 12

},

{

nombre: "maría",

edad: 22

},

{

nombre: "oscar",

edad: 72

}

];

// Todas las personas tienen una edad menor a 100?

console.log(personas.every((elemento, indice, arreglo) => elemento.edad < 100)); // true

// Todas las personas son mayor de edad?

console.log(personas.every((elemento, indice, arreglo) => elemento.edad >= 18)); // false

// Todas las personas tienen un nombre con 6 o menos caracteres?

console.log(personas.every((elemento, indice, arreglo) => elemento.nombre.length <= 6)); // true

// Alguna persona tiene 100 años de edad?

console.log(personas.some((elemento, indice, arreglo) => elemento.edad == 100)); // false

// Alguna persona es menor de edad?

console.log(personas.some((elemento, indice, arreglo) => elemento.edad <= 18)); // true

</script>

**Método filter.**

El método 'filter' retorna un arreglo con todos los elementos del arreglo original que cumplen una condición que definimos en la función anónima que le pasamos:

<script>

const personas = [

{

nombre: "pedro",

edad: 34

},

{

nombre: "ana",

edad: 54

},

{

nombre: "carlos",

edad: 12

},

{

nombre: "maría",

edad: 22

},

{

nombre: "oscar",

edad: 72

}

];

// Recuperar todas las personas mayores de edad

const personasmayores = personas.filter((elemento, indice, arreglo) => elemento.edad >= 18);

console.log(personasmayores); // [{nombre: "pedro",edad: 34},{nombre: "ana",edad: 54},

// {nombre: "maría",edad: 22},{nombre: "oscar",edad: 72}]

</script>

Tener en cuenta que la función anónima debe retornar true si queremos que el elemento se almacene en el arreglo generado. Si bien no tiene sentido, si queremos que se guarden todos los elementos luego podemos codificar:

const personasmayores = personas.filter((elemento, indice, arreglo) => true);

Recordar que si no utilizamos funciones flecha el algoritmo será:

const personasmayores = personas.filter(function (elemento, indice, arreglo) { return true });

**Método map.**

El método 'map' similar al método 'filter' retorna un arreglo cuyos elementos resultan de las operaciones que se efectúan dentro de la función anónima que le pasamos como parámetro:

<script>

const arreglo1 = [1, 2, 3, 4, 5];

// Generar un nuevo arreglo con los elementos del arreglo1 elevalos al cuadrado

const arreglo2 = arreglo1.map((elemento, indice, arreglo) => elemento \*\* 2);

console.log(arreglo2); //[1, 4, 9, 16, 25]

</script>

El arreglo generado siempre es del mismo tamaño que el arreglo original.

**Métodos reduce y reduceRight.**

El método 'reduce' recibe como primer parámetro una función cuyo primer parámetro es un valor que se arrastra entre cada proceso de un elemento del arreglo, el segundo parámetro de reduce es el valor inicial con el que inicia el acumulador (si no pasamos el segundo parámetro se inicia con el primer elemento del arreglo):

<script>

const arreglo1 = [1, 2, 73, 3, 4, 120, 5, 18];

// Acumular todos los elementos que tienen un solo dígito

const cantidad = arreglo1.reduce((acumulador, elemento, indice, arreglo) => {

if (elemento < 10)

acumulador += elemento;

return acumulador;

}, 0);

console.log(cantidad); // 15

</script>

El método 'reduceRight' es simila a 'reduce' con la diferencia que visita cada elemento del arreglo partiendo del final.

<script>

const arreglo1 = [1, 2, 73, 3, 4, 120, 5, 18];

// Generar un nuevo arreglo con los elementos invertidos

const arreglo2 = arreglo1.reduceRight((acumulador, elemento, indice, arreglo) => {

acumulador.push(elemento);

return acumulador;

}, []);

console.log(arreglo2); // [18, 5, 120, 4, 3, 73, 2, 1]

</script>

El valor inicial es un arreglo vacío que le pasamos al método reduceRight:

[]

Y dentro de la función anónima agregamos cada elemento del vector visitando sus componentes a partir del final:

acumulador.push(elemento);

## JQuery

Para utilizar la librería como dijimos debemos descargarla del sitio oficial y en cada página que lo requiera agregar:

<script src="jquery-3.4.1.min.js"></script>

Podemos utilizar un CDN donde se almacene la librería y liberar a nuestro servidor de esa carga:

<script src="https://code.jquery.com/jquery-3.4.1.min.js"></script>

Siempre debemos disponer esta inclusión de la librería antes de incluir los otros archivos \*.js que utilizan esta librería, es por eso el orden:

<script src="https://code.jquery.com/jquery-3.4.1.min.js"></script>

<script src="funciones2.js"></script>

La librería jQuery en resumen nos aporta las siguientes ventajas:

* Nos ahorra muchas líneas de código.
* Nos hace transparente el soporte de nuestra aplicación para los navegadores principales.
* Nos provee de un mecanismo para la captura de eventos.
* Provee un conjunto de funciones para animar el contenido de la página en forma muy sencilla.
* Integra funcionalidades para trabajar con AJAX.
* Existen un amplio abanico de extensiones.

<body>

<input type="button" id="boton1" value="presioname">

<script src="https://code.jquery.com/jquery-3.4.1.min.js"></script>

<script src="funciones2.js"></script>

</body>

funciones2.js :

let x = $(document);

x.ready(inicializarEventos);

function inicializarEventos() {

let x = $("#boton1");

x.click(presionBoton)

}

function presionBoton() {

alert("Se presionó el botón");

}

En este archivo (funciones2.js) comienza nuestro estudio de la librería jQuery

La función principal de esta librería se llama $.

A la función $ le podemos pasar distintos valores como veremos (en la primer llamada le pasamos la referencia del objeto document del DOM y en la segunda el id del control button):

let x = $(document);

...

let x = $("#boton1");

Esta función nos devuelve un objeto de tipo jQuery.

El primer método que nos interesa de esta clase es el ready:

let x = $(document);

x.ready(inicializarEventos);

El método ready tiene como parámetro un nombre de función. Esta función se ejecutará cuando todos los elementos de la página estén cargados. Es importante notar que solo pasamos el nombre de la función y no disponemos los paréntesis abiertos y cerrados al final.

El código de esta función:

function inicializarEventos() {

let x = $("#boton1");

x.click(presionBoton)

}

Utilizamos nuevamente la función $ para crear un objeto de la clase jQuery pero ahora asociándolo al botón, esto lo hacemos pasando el id del control button precediéndolo por el carácter # y encerrado entre paréntesis. Finalmente llamamos al método click pasándo como parámetro el nombre de la función que se ejecutará al presionar dicho botón.

### Selección de un elemento del documento mediante el id:

La sintaxis para seleccionar un elemento particular de la página mediante la propiedad id es:

$("#nombre del id")

<body>

<h1 id="titulo1">Primer título</h1>

<h1 id="titulo2">Segundo título</h1>

<script src="https://code.jquery.com/jquery-3.4.1.min.js"></script>

<script src="funciones.js"></script>

</body>

let x = $(document);

x.ready(inicializarEventos);

function inicializarEventos() {

let x = $("#titulo1");

x.click(presionTitulo1)

x = $("#titulo2");

x.click(presionTitulo2)

}

function presionTitulo1() {

let x = $("#titulo1");

x.css("color", "#ff0000");

x.css("background-color", "#ffff00");

x.css("font-family", "Courier");

}

function presionTitulo2() {

let x = $("#titulo2");

x.css("color", "#ffff00");

x.css("background-color", "#ff0000");

x.css("font-family", "Arial");

}

Como va ser costumbre siempre que trabajemos con esta librería primero creamos un objeto jquery a partir de la referencia a 'document' y luego llamamos al método ready indicándole el nombre de la función que debe ejecutarse luego de generarse el árbol de elementos HTML para la página:

let x = $(document);

x.ready(inicializarEventos);

La función inicializarEventos se ejecuta una vez que se cargó la página y están creados todos los elementos HTML, en esta función mediante $ accedemos a través del id a los elementos h1 respectivos, a casa uno le asignamos al evento click una función distinta que se disparará cuando presionemos con el mouse:

function inicializarEventos() {

let x = $("#titulo1");

x.click(presionTitulo1)

x = $("#titulo2");

x.click(presionTitulo2)

}

Es importante notar que cuando obtenemos la referencia de un elemento por medio del id se le antecede el caracter # al nombre del id:

x=$("#titulo1");

...

x=$("#titulo2");

Luego las dos funciones que se ejecutan al presionar los títulos:

function presionTitulo1() {

let x = $("#titulo1");

x.css("color", "#ff0000");

x.css("background-color", "#ffff00");

x.css("font-family", "Courier");

}

function presionTitulo2() {

let x = $("#titulo2");

x.css("color", "#ffff00");

x.css("background-color", "#ff0000");

x.css("font-family", "Arial");

}

Hasta ahora hemos presentado los siguientes métodos que tiene jquery:

* ready
* click

El tercer método nos permite modificar una propiedad de la hoja de estilo de un elemento HTML:

let x = $("#titulo1");

x.css("color", "#ff0000");

Una vez que hemos obtenido la referencia a un elemento HTML llamamos al método css que tiene dos parámetros: el primero indica el nombre de la propiedad y el segundo el valor a asignarle. Podemos ver las otras dos asignaciones:

x.css("background-color", "#ffff00");

x.css("font-family", "Courier");

Como podemos ver es muy fácil acceder al estilo de un elemento HTML para actualizarlo en forma dinámica luego que la página fue cargada.

### Selección de elementos por el tipo de elementos:

La sintaxis para tener la referencia de todos los elementos de cierto tipo (a, p, h1, etc.):

$("nombre del elemento")

Es decir es casi igual que obtener la referencia de un elemento particular mediante el id, solo difiere en que no le antecedemos el caracter #.

Podemos con esto definir funciones comunes a un conjunto de elementos.

<body>

<table border="1">

<tr>

<td>1111111111</td>

<td>1111111111</td>

<td>1111111111</td>

<td>1111111111</td>

</tr>

<tr>

<td>2222222222</td>

<td>2222222222</td>

<td>2222222222</td>

<td>2222222222</td>

</tr>

<tr>

<td>3333333333</td>

<td>3333333333</td>

<td>3333333333</td>

<td>3333333333</td>

</tr>

<tr>

<td>4444444444</td>

<td>4444444444</td>

<td>4444444444</td>

<td>4444444444</td>

</tr>

<tr>

<td>5555555555</td>

<td>5555555555</td>

<td>5555555555</td>

<td>5555555555</td>

</tr>

</table>

<script src="https://code.jquery.com/jquery-3.4.1.min.js"></script>

<script src="funciones.js"></script>

</body>

let x = $(document);

x.ready(inicializarEventos);

function inicializarEventos() {

let x = $("tr");

x.click(presionFila);

}

function presionFila() {

let x = $(this);

x.css("background-color", "#eeeeee");

}

Con la siguiente sintaxis obtenemos la referencia a todos los elementos HTML de tipo 'tr':

let x = $("tr");

x.click(presionFila);

y a todos ellos los enlazamos con la función presionFila

Cuando se presiona cualquiera de las filas de la tabla se ejecuta la siguiente función:

function presionFila() {

let x = $(this);

x.css("background-color", "#eeeeee");

}

Para obtener la referencia de que elemento en particular disparó el evento podemos hacerlo de la siguiente manera:

let x = $(this);

this una función de JavaScript que se usa en conjunto con jQuery mediante el selector $(this) y nos permite obtener la referencia del elemento que emitió el evento.

Y a partir de esta referencia llamar al método css.

### Selección de elementos utilizando los selectores CSS:

Una de las características más interesantes de jQuery es poder obtener la referencia de elementos del DOM utilizando para ello selectores de CSS (Significa que el manejo de Hojas de estilo nos facilita las cosas)

<body>

<input type="button" value="ocultar item primer lista" id="boton1">

<h2>Lista 1</h2>

<ul id="lista1">

<li>Opción número 1</li>

<li>Opción número 2</li>

<li>Opción número 3</li>

<li>Opción número 4</li>

</ul>

<h2>Lista 2</h2>

<ul id="lista2">

<li>Opción número 1</li>

<li>Opción número 2</li>

<li>Opción número 3</li>

<li>Opción número 4</li>

</ul>

<script src="https://code.jquery.com/jquery-3.4.1.min.js"></script>

<script src="funciones.js"></script>

</body>

Cada lista tiene definido su atributo id.

let x=$(document);

x.ready(inicializarEventos);

function inicializarEventos()

{

let x=$("#boton1");

x.click(ocultarItem);

}

function ocultarItem()

{

let x=$("#lista1 li");

x.hide();

}

En la función inicializarEventos obtenemos la referencia del botón mediante su id y llamamos al método click:

function inicializarEventos()

{

let x=$("#boton1");

x.click(ocultarItem);

}

Ahora lo nuevo es obtener la referencia de todos los elementos li que pertenecen a la primer lista:

let x=$("#lista1 li");

x.hide();

Pasamos a la función $ el selector CSS que accede a todos los elementos de tipo li contenidos en #lista1.

Todas las reglas de CSS están disponibles para seleccionar los elementos del DOM.

### Selección de elementos utilizando las clases CSS definidas:

Recordemos que definimos clases en CSS cuando una regla de estilo puede ser igual para un conjunto de marcas HTML.

Mediante este nombre de clase podemos acceder a todos los elementos utilizando la función $:

x=$(".nombre de clase");

<head>

<title>Ejemplo de jQuery</title>

<meta charset="UTF-8">

<link rel="StyleSheet" href="estilos.css" type="text/css">

</head>

<body>

<p>Cuales de los siguientes lenguajes son orientados a objetos:

<span class="resaltado">Java</span>, VB6, <span class="resaltado">C++</span>, C,

<span class="resaltado">C#</span>, Cobol ?</p>

<input type="button" id="boton1" value="ver resultado">

<script src="https://code.jquery.com/jquery-3.4.1.min.js"></script>

<script src="funciones.js"></script>

</body>

let x = $(document);

x.ready(inicializarEventos);

function inicializarEventos() {

let x = $("#boton1");

x.click(resaltar);

}

function resaltar() {

let x = $(".resaltado");

x.css("background-color", "#ffff00");

}

.resaltado{

color:#ff0000;

}

El código nuevo es:

let x = $(".resaltado");

x.css("background-color", "#ffff00");

Primero generamos un objeto jQuery que guarda la referencia a todos los elementos que tienen definida la clase .resalatado y luego fijamos como color de fondo el amarillo a dichos elementos.

### Método text(), text(valor):

Para saber el contenido de un elemento el objeto jQuery cuenta con un método llamado text(), por ejemplo:

let x = $("#parrafo1");

Luego si hacemos x.text() obtenemos el contenido del párrafo con id igual a parrafo1.

Luego si queremos cambiar el texto del párrafo deberíamos disponer:

let x = $("#parrafo1");

x.text("Este es el texto nuevo");

Pero hay que tener mucho cuidado cuando utilizamos jQuery ya que podemos cambiar el contenido de muchos elementos con una sola llamada al método text, por ejemplo:

let x=$("p");

x.text("Este texto aparece en todos los párrafos del documento");

El código anterior crea un objeto jQuery con la referencia a todos los párrafos contenidos en el documento. Luego si llamamos al método text enviándole una cadena, esta aparecerá en todo el documento remplazando el contenido de los párrafos actuales.

<body>

<input type="button" value="Obtener el texto contenido en un párrafo" id="boton1"><br>

<input type="button" value="Modificar el texto de un párrafo" id="boton2"><br>

<input type="button" value="Modificar el texto de los elementos td de una tabla" id="boton3"><br>

<p id="parrafo1">Texto del primer párrafo</p>

<table border="1">

<tr>

<td>celda 1,1</td>

<td>celda 1,2</td>

<td>celda 2,1</td>

<td>celda 2,2</td>

</tr>

</table>

<script src="https://code.jquery.com/jquery-3.4.1.min.js"></script>

<script src="funciones.js"></script>

</body>

let x = $(document);

x.ready(inicializarEventos);

function inicializarEventos() {

let x = $("#boton1");

x.click(extraerTexto);

x = $("#boton2");

x.click(modificarTexto);

x = $("#boton3");

x.click(modificarDatosTabla);

}

function extraerTexto() {

let x = $("#parrafo1");

alert(x.text());

}

function modificarTexto() {

let x = $("#parrafo1");

x.text("Nuevo texto del párrafo");

}

function modificarDatosTabla() {

let x = $("td");

x.text("texto nuevo");

}

Como vemos esta página tiene tres botones, al presionarse el primero se dispara la función:

function extraerTexto() {

let x = $("#parrafo1");

alert(x.text());

}

Obtenemos la referencia al párrafo mediante su id (recordemos que en una página todos los valores de los id son distintos), luego extraemos el contenido mediante el método text() y lo mostramos en un alert.

La segunda función cambia el contenido del párrafo:

function modificarTexto() {

let x = $("#parrafo1");

x.text("Nuevo texto del párrafo");

}

Obtenemos la referencia del párrafo mediante su id y llamamos al método text enviándole el nuevo string a mostrar.

Por último la función:

function modificarDatosTabla() {

let x = $("td");

x.text("texto nuevo");

}

Crea un objeto de la clase jQuery con la referencia a todos los elementos td del documento (es decir los td de todas las tablas) y posteriormente mediante el método text modifica el contenido de todos ellos (todos los td del documento se cambian por el string "nuevo texto")

### Métodos attr(nombre de propiedad), attr(nombre de propiedad,valor) y removeAttr(nombre de propiedad):

Estos métodos nos permiten agregar propiedades a un elemento HTML y recuperar el valor de una propiedad.

Para recuperar el valor de una propiedad (si hay muchos elementos que recupera la función $, solo retorna la propiedad del primero):

$(elemento).attr(nombre de propiedad)

Para fijar el valor de una propiedad (si hay muchos elementos que recupera la función $, se inicializan para todos):

$(elemento).attr(nombre de propiedad,valor)

Para eliminar una propiedad de un elemento o conjunto de elementos tenemos:

$(elemento).removeAttr(nombre de la propiedad)

<body>

<input type="button" id="boton1" value="Añadir propiedad border"><br>

<input type="button" id="boton2" value="Recuperar valor de la propiedad border"><br>

<input type="button" id="boton3" value="Eliminar la propiedad border">

<table id="tabla1">

<tr>

<td>1111111111</td>

<td>1111111111</td>

<td>1111111111</td>

<td>1111111111</td>

</tr>

<tr>

<td>2222222222</td>

<td>2222222222</td>

<td>2222222222</td>

<td>2222222222</td>

</tr>

<tr>

<td>3333333333</td>

<td>3333333333</td>

<td>3333333333</td>

<td>3333333333</td>

</tr>

</table>

<script src="https://code.jquery.com/jquery-3.4.1.min.js"></script>

<script src="funciones.js"></script>

</body>

let x = $(document);

x.ready(inicializarEventos);

function inicializarEventos() {

let x = $("#boton1");

x.click(agregarPropiedadBorder);

x = $("#boton2");

x.click(recuperarPropiedadBorder);

x = $("#boton3");

x.click(eliminarPropiedadBorder);

}

function agregarPropiedadBorder() {

let x = $("#tabla1");

x.attr("border", "1");

}

function recuperarPropiedadBorder() {

let x = $("#tabla1");

if (x.attr("border") != undefined)

alert(x.attr("border"));

else

alert("No está definida la propiedad border en la tabla");

}

function eliminarPropiedadBorder() {

let x = $("#tabla1");

x.removeAttr("border");

}

Cuando se presiona el primer botón:

function agregarPropiedadBorder() {

let x = $("#tabla1");

x.attr("border", "1");

}

Obtenemos la referencia de la tabla mediante su id y llamamos al método attr pasando como primer parámetro el nombre de la propiedad a agregar y como segundo parámetro el valor de la propiedad.

Cuando se presiona el segundo botón:

function recuperarPropiedadBorder() {

let x = $("#tabla1");

if (x.attr("border") != undefined)

alert(x.attr("border"));

else

alert("No está definida la propiedad border en la tabla");

}

Llamamos al método attr enviándole como parámetro el nombre de la propiedad que queremos rescatar. Si retorna el valor undefined significa que no tiene dicha propiedad el elemento HTML, en caso contrario retorna su valor y procedemos a mostrarlo mediante un alert.

Cuando se presiona el tercer botón:

function eliminarPropiedadBorder() {

let x = $("#tabla1");

x.removeAttr("border");

}

Obtenemos la referencia a la tabla mediante su id y llamamos al método removeAttr con el nombre de la propiedad a eliminar.

### Métodos addClass y removeClass:

Los métodos addClass y removeClass nos permiten asociar y desasociar una clase a un elemento o conjunto de elementos HTML.

<head>

<title>Ejemplo de jQuery</title>

<meta charset="UTF-8">

<link rel="StyleSheet" href="estilos.css" type="text/css">

</head>

<body>

<input type="button" id="boton1" value="Asociar clase">

<input type="button" id="boton2" value="Desasociar clase">

<div id="descripcion">

<p>HTML es el lenguaje que se emplea para el

desarrollo de páginas de internet.</p>

<p>Este lenguaje está constituido de

elementos que el navegador interpreta y las

despliega en la pantalla de acuerdo

a su objetivo. Veremos que hay elementos

para disponer imágenes sobre una página, hipervínculos

que nos permiten

dirigirnos a otra página, listas, tablas para

tabular datos, etc.</p>

<p>Para poder crear una página HTML se requiere

un simple editor de texto (en nuestro caso emplearemos este sitio)

y un navegador de internet (IExplorer, FireFox etc.),

emplearemos el navegador que en este preciso momento está

utilizando (recuerde que usted está viendo en este

preciso momento una página HTML con su navegador).</p>

<p>Lo más importante es que en cada concepto

desarrolle los ejercicios propuestos y modifique los que se presentan

ya resueltos.</p>

<p>Este curso lo que busca es acercar el lenguaje

HTML a una persona que nunca antes trabajó con el mismo. No

pretende mostrar todas los elementos HTML en forma alfabética.</p>

<p>Como veremos, de cada concepto se presenta una

parte teórica, en la que se da una explicación completa,

luego se pasa a la sección del ejercicio resuelto donde

podemos ver el contenido de la página HTML y cómo la

visualiza el navegador. Por último y tal vez la sección

más importante de este tutorial es donde se propone

que usted haga páginas en forma autónoma (donde realmente

podrá darse cuenta si el concepto quedó firme).

</p>

</div>

<script src="https://code.jquery.com/jquery-3.4.1.min.js"></script>

<script src="funciones.js"></script>

</body>

let x = $(document);

x.ready(inicializarEventos);

function inicializarEventos() {

let x = $("#boton1");

x.click(asociarClase);

x = $("#boton2");

x.click(desasociarClase);

}

function asociarClase() {

let x = $("#descripcion");

x.addClass("recuadro");

}

function desasociarClase() {

let x = $("#descripcion");

x.removeClass("recuadro");

}

.recuadro {

background-color:#ffffcc;

font-family:verdana;

font-size:14px;

border-top-width:1px;

border-right-width:3px;

border-bottom-width:3px;

border-left-width:1px;

border-top-style:dotted;

border-right-style:solid;

border-bottom-style:solid;

border-left-style:dotted;

border-top-color:#ffaa00;

border-right-color:#ff0000;

border-bottom-color:#ff0000;

border-left-color:#ffaa00;

}

Cuando se presiona el botón asociar hoja de estilo se ejecuta la función:

function asociarClase() {

let x = $("#descripcion");

x.addClass("recuadro");

}

Donde llamamos al método addClass con el nombre de la clase (dicha clase debe estar definida en la hoja de estilo (css)).

De forma similar para desasociar una clase se ejecuta la función:

function desasociarClase() {

let x = $("#descripcion");

x.removeClass("recuadro");

}

Donde llamamos al método removeClass a partir de un objeto jQuery.

### Método html() y html(valor):

El método:

html([bloque html])

Nos permite agregar un bloque de html a partir de un elemento de la página. Básicamente es la propiedad innerHTML del DOM.

Y el método:

html()

Nos retorna el bloque html contenido a partir del elemento html que hace referencia el objeto jQuery.

<body>

<input type="button" id="boton1" value="Mostrar formulario">

<input type="button" id="boton2" value="Mostrar elementos html del formulario"><br>

<div id="formulario">

</div>

<script src="https://code.jquery.com/jquery-3.4.1.min.js"></script>

<script src="funciones.js"></script>

</body>

let x = $(document);

x.ready(inicializarEventos);

function inicializarEventos() {

let x = $("#boton1");

x.click(presionBoton1);

x = $("#boton2");

x.click(presionBoton2);

}

function presionBoton1() {

let x = $("#formulario");

x.html('<form>Ingrese nombre:<input type="text" id = "nombre" >' +

'<br>Ingrese clave:<input type="text" id="clave"><br>' +

'<input type="submit" value="confirmar"></form>');

}

function presionBoton2() {

let x = $("#formulario");

alert(x.html());

}

Como podemos observar cuando se presiona el primer botón creamos un objeto jQuery que toma la referencia del div y mediante el método html crea un bloque en su interior:

function presionBoton1() {

let x = $("#formulario");

x.html('<form>Ingrese nombre:<input type="text" id = "nombre" >' +

'<br>Ingrese clave:<input type="text" id="clave"><br>' +

'<input type="submit" value="confirmar"></form>');

}

El segundo botón muestra en un alert el contenido HTML actual del div:

function presionBoton2() {

let x = $("#formulario");

alert(x.html());

}

Hay que diferenciar bien los métodos html(valor) y text(valor), el segundo como habíamos visto agrega texto a un elemento HTML.

### Administración de eventos con jQuery:

jQuery facilita la administración de eventos de JavaScript y lo más importante nos hace transparente la diferencia en la registración de eventos entre distintos navegadores (IExplorer, FireFox, Chrome, Safari, Opera)

En este tutorial ya hemos utilizado el manejador de eventos:

$(documento).ready(nombre de función)

Dijimos que este función que registramos mediante el método ready se ejecuta cuando el DOM del documento está en memoria. Si no utilizamos la librería jQuery hacemos esto a través del evento addEventListener.

Otro evento que vimos en conceptos anteriores es el click de un elemento, la sintaxis utilizada:

var x;

x=$("button");

x.click(presionBoton)

Con este pequeño código registramos la función presionBoton para todos los elementos de tipo button del documento.

Con este otro código:

var x;

x=$("#boton1");

x.click(presionBoton)

solo se registra para el elemento con id con valor "boton1".

Por último con este código se registra para todos los elementos "button" que dependen del div con valor de id igual a "formulario1":

var x;

x=$("#formulario1 button");

x.click(presionBoton)

Para reafirmar estos conceptos confeccionaremos una página que muestre un párrafo, luego una tabla que contenga dos párrafos y por último otra tabla que contenga dos párrafos. Capturaremos el evento click de todos los párrafos del documento y mostraremos un mensaje indicando:'se presionó un párrafo del documento', también capturaremos el evento click de los dos párrafos de la segunda tabla y mostraremos: 'se presionó un párrafo contenido en la segunda tabla.'.

<body>

<p>Párrafo fuera de la tabla</p>

<table border="1">

<tr>

<td>

<p>Párrafo dentro de la primer tabla.</p>

</td>

<td>

<p>Párrafo dentro de la primer tabla.</p>

</td>

</tr>

</table>

<br>

<table border="1" id="tabla2">

<tr>

<td>

<p>Párrafo dentro de la segunda tabla.</p>

</td>

<td>

<p>Párrafo dentro de la segunda tabla.</p>

</td>

</tr>

</table>

<script src="https://code.jquery.com/jquery-3.4.1.min.js"></script>

<script src="funciones.js"></script>

</body>

let x = $(document);

x.ready(inicializarEventos);

function inicializarEventos() {

let x = $("p");

x.click(presionParrafoDocumento);

x = $("#tabla2 p");

x.click(presionpresionParrafoSegundaTabla);

}

function presionParrafoDocumento() {

alert('se presionó un párrafo del documento');

}

function presionpresionParrafoSegundaTabla() {

alert('se presionó un párrafo contenido en la segunda tabla.');

}

El siguiente código asocia a cada elemento de tipo "p" (párrafo del documento) la función 'presionParrafoDocumento':

let x = $("p");

x.click(presionParrafoDocumento);

Luego cuando ejecutemos este documento cada vez que presionemos un párrafo de la página mostrará el mensaje 'se presionó un párrafo del documento'.

También asociamos otra función para el evento click de los párrafos contenidos dentro de la segunda tabla:

x = $("#tabla2 p");

x.click(presionpresionParrafoSegundaTabla);

Esto significa que los párrafos contenidos en la segunda tabla tienen asociados dos funciones para el evento click, luego cuando presionemos alguno de los párrafos de la segunda tabla aparecerán los dos mensajes: 'se presionó un párrafo del documento' y 'se presionó un párrafo contenido en la segunda tabla.'

### Eventos mouseover y mouseout:

Los eventos de JavaScript onmouseover y onmouseout son los equivalentes mouseover y mouseout de jQuery. Estos eventos están generalmente unidos. El primero se dispara cuando ingresamos la flecha del mouse a un elemento HTML y el segundo cuando sacamos la flecha del control.

Para probar estos dos eventos implementaremos una página que contenga tres botones y cuando ingrese la flecha del mouse al botón cambiaremos el color de texto del mismo, retornando el color original cuando retiramos la flecha del control.

<body>

<a href="http://www.lavoz.com.ar">La Voz del Interior</a>

<br>

<a href="http://www.lanacion.com.ar">La nación</a>

<br>

<a href="http://www.clarin.com.ar">El clarín</a>

<br>

<script src="https://code.jquery.com/jquery-3.4.1.min.js"></script>

<script src="funciones.js"></script>

</body>

let x = $(document);

x.ready(inicializarEventos);

function inicializarEventos() {

let x = $("a");

x.mouseover(entraMouse);

x.mouseout(saleMouse);

}

function entraMouse() {

$(this).css("background-color", "#ff0");

}

function saleMouse() {

$(this).css("background-color", "#fff");

}

Lo primero que hacemos es asignar las funciones a ejecutarse cuando ocurren los evento mouseover y mouseout:

let x = $("a");

x.mouseover(entraMouse);

x.mouseout(saleMouse);

La función entraMouse accede al elemento que recibe en su interior el mouse (la obtenemos mediante la referencia que guarda this) y cambia el color de la propiedad text-background del CSS:

function entraMouse() {

$(this).css("background-color", "#ff0");

}

De forma similar la función saleMouse retorna al color original:

function saleMouse() {

$(this).css("background-color", "#fff");

}

### Evento hover:

jQuery no solo mapea los eventos del DOM sino que provee otros que combinan estos.

Como decíamos es habitual utilizar los eventos mouseover y mouseout en común, por eso en jQuery existe un evento llamado hover que tiene dos parámetros:

$(elemento).hover([función de ingreso del mouse],[función de salida del mouse])

Es decir que al evento hover asociamos dos funciones, la primera se ejecuta cuando ingresamos la flecha del mouse dentro del elemento y la segunda cuando retiramos la flecha del mouse.

<body>

<a href="http://www.lavoz.com.ar">La Voz del Interior</a>

<br>

<a href="http://www.lanacion.com.ar">La nación</a>

<br>

<a href="http://www.clarin.com.ar">El clarín</a>

<br>

<script src="https://code.jquery.com/jquery-3.4.1.min.js"></script>

<script src="funciones.js"></script>

</body>

let x = $(document);

x.ready(inicializarEventos);

function inicializarEventos() {

let x = $("a");

x.hover(entraMouse, saleMouse);

}

function entraMouse() {

$(this).css("background-color", "#ff0");

}

function saleMouse() {

$(this).css("background-color", "#fff");

}

Utilizamos solamente el evento hover para cambiar el color de fondo del ancla cuando se ingresa la flecha del mouse y retornarla al color original cuando se sale:

let x = $("a");

x.hover(entraMouse, saleMouse);

### Evento mousemove:

Este evento se dispara cuando se mueve la flecha del mouse dentro del elemento HTML respectivo.

Si queremos recuperar la coordenada donde se encuentra en ese momento el mouse, jQuery pasa el objeto event como parámetro (con la ventaja que hace transparente las diferencias entre IE y otros navegadores)

<body>

<p id="corx">coordenada x=</p>

<p id="cory">coordenada y=</p>

<script src="https://code.jquery.com/jquery-3.4.1.min.js"></script>

<script src="funciones.js"></script>

</body>

let x = $(document);

x.ready(inicializarEventos);

function inicializarEventos() {

let x = $(document);

x.mousemove(moverMouse);

}

function moverMouse(event) {

let x = $("#corx");

x.text("coordenada x=" + event.clientX);

x = $("#cory");

x.text("coordenada y=" + event.clientY);

}

Primero obtenemos la referencia del objeto document y le registramos la función a ejecutar cuando se desplace el mouse:

let x = $(document);

x.mousemove(moverMouse);

La función que se ejecuta cada vez que se desplaza la flecha del mouse en el documento es:

function moverMouse(event) {

let x = $("#corx");

x.text("coordenada x=" + event.clientX);

x = $("#cory");

x.text("coordenada y=" + event.clientY);

}

En esta creamos un objeto jQuery a partir del id del primer párrafo y fijamos el texto del mismo con el valor del atributo event.clientX que almacena la coordenada x de la flecha del mouse:

let x = $("#corx");

x.text("coordenada x=" + event.clientX);

### Eventos mousedown y mouseup:

El evento mousedown se dispara cuando presionamos alguno de los botones del mouse y el evento mouseup cuando dejamos de presionar el botón.

<body>

<table border="1">

<tr>

<td>1111111111</td>

<td>1111111111</td>

</tr>

</table>

<script src="https://code.jquery.com/jquery-3.4.1.min.js"></script>

<script src="funciones.js"></script>

</body>

let x = $(document);

x.ready(inicializarEventos);

function inicializarEventos() {

let x = $("td");

x.mousedown(presionaMouse);

x.mouseup(sueltaMouse);

}

function presionaMouse() {

$(this).css("background-color", "#ff0");

}

function sueltaMouse() {

$(this).css("background-color", "#fff");

}

Primero asociamos los eventos mousedown y mouseup a todos los elementos td del documento:

let x = $("td");

x.mousedown(presionaMouse);

x.mouseup(sueltaMouse);

Cuando se presiona algún botón del mouse dentro de una celda de la tabla se dispara la función:

function presionaMouse() {

$(this).css("background-color", "#ff0");

}

De forma similar cuando se suelta el botón del mouse se dispara la función:

function sueltaMouse() {

$(this).css("background-color", "#fff");

}

### Evento dblclick:

El evento dblclick se dispara cuando se presiona dos veces seguidas el botón izquierdo del mouse.

Para ver el funcionamiento de este evento crearemos un div en una coordenada absoluta y lo ocultaremos al hacer doble clic en su interior.

<head>

<title>Ejemplo de jQuery</title>

<meta charset="UTF-8">

<link rel="StyleSheet" href="estilos.css" type="text/css">

</head>

<body>

<div id="recuadro">

<h1>Doble clic para ocultar este recuadro</h1>

</div>

<script src="https://code.jquery.com/jquery-3.4.1.min.js"></script>

<script src="funciones.js"></script>

</body>

let x = $(document);

x.ready(inicializarEventos);

function inicializarEventos() {

let x = $("#recuadro");

x.dblclick(dobleClic);

}

function dobleClic() {

let x = $(this);

x.hide()

}

#recuadro {

color:#aa0;

background-color:#ff0;

position:absolute;

text-align:center;

left:40px;

top:30px;

width:800px;

height:70px;

}

En la función inicializarEventos registramos el evento dblclick para el div:

let x = $("#recuadro");

x.dblclick(dobleClic);

Cuando se presiona dos veces seguidas dentro del div se dispara la función:

function dobleClic() {

let x = $(this);

x.hide()

}

donde obtenemos la referencia del elemento que emitió el evento y llamamos al método hide para que lo oculte.

### Evento focus:

El evento focus se produce cuando se activa el control. Podemos capturar el evento focus de un control de tipo text, textarea, button, checkbox, fileupload, password, radio, reset y submit.

Es común a muchos sitio donde se puede buscar información, disponer un control de tipo text con la leyenda 'Buscar...' y sin ningún botón a su lado. Cuando el control toma foco se borra automáticamente su contenido, el operador ingresa el texto a buscar y presiona la tecla ENTER.

<body>

<form action="#" method="post">

<input type="text" name="buscar" id="buscar" value="Buscar..." size="20">

</form>

<script src="https://code.jquery.com/jquery-3.4.1.min.js"></script>

<script src="funciones.js"></script>

</body>

let x = $(document);

x.ready(inicializarEventos);

function inicializarEventos() {

let x = $("#buscar");

x.focus(tomaFoco);

}

function tomaFoco() {

let x = $("#buscar");

x.attr("value", "");

}

En la función inicializarEventos creamos un objeto jQuery a partir del id del control de tipo text y asociamos el evento focus con el método tomaFoco:

let x = $("#buscar");

x.focus(tomaFoco);

El método tomaFoco obtiene la referencia del elemento tipo text y mediante el método attr modifica la propiedad value:

function tomaFoco() {

let x = $("#buscar");

x.attr("value", "");

}

### Evento blur:

El evento blur se dispara cuando pierde el foco el control.

Podemos capturar el evento blur de un control de tipo text, textarea, button, checkbox, fileupload, password, radio, reset y submit.

<head>

<title>Ejemplo de jQuery</title>

<meta charset="UTF-8">

<link rel="StyleSheet" href="estilos.css" type="text/css">

</head>

<body>

<form action="#" method="post">

<input type="text" name="text1" id="text1" value="Hola" size="20">

<br>

<input type="text" name="text2" id="text2" value="Hola" size="20">

</form>

<script src="https://code.jquery.com/jquery-3.4.1.min.js"></script>

<script src="funciones.js"></script>

</body>

let x = $(document);

x.ready(inicializarEventos);

function inicializarEventos() {

let x = $("#text1");

x.focus(tomaFoco);

x.blur(pierdeFoco);

x = $("#text2");

x.focus(tomaFoco);

x.blur(pierdeFoco);

}

function tomaFoco() {

let x = $(this);

x.css("color", "#f00");

}

function pierdeFoco() {

let x = $(this);

x.css("color", "#00f");

}

#text1,#text2 {

color:#00f;

}

En la función inicializarEventos asociamos los eventos focus y blur a los dos controles:

function inicializarEventos() {

let x = $("#text1");

x.focus(tomaFoco);

x.blur(pierdeFoco);

x = $("#text2");

x.focus(tomaFoco);

x.blur(pierdeFoco);

}

El evento tomaFoco cambia de color a rojo al texto del control seleccionado:

function tomaFoco() {

let x = $(this);

x.css("color", "#f00");

}

De forma similar pierdeFoco cambia a azul:

function pierdeFoco() {

let x = $(this);

x.css("color", "#00f");

}

### Manipulación de los elementos del DOM:

jQuery dispone de una serie de métodos que nos facilitan el tratamiento de los elementos del DOM.

<body>

<h1>Métodos para manipular nodos del DOM con jQuery.</h1>

<ul>

<li>Primer item.</li>

<li>Segundo item.</li>

<li>Tercer item.</li>

<li>Cuarto item.</li>

</ul>

<input type="button" id="boton1" value="Eliminar la lista completa."><br>

<input type="button" id="boton2" value="Restaurar Lista"><br>

<input type="button" id="boton3" value="Añadir un elemento al final de la lista"><br>

<input type="button" id="boton4" value="Añadir un elemento al principio de la lista"><br>

<input type="button" id="boton5" value="Eliminar el último elemento."><br>

<input type="button" id="boton6" value="Eliminar el primer elemento."><br>

<input type="button" id="boton7" value="Eliminar el primero y segundo elemento."><br>

<input type="button" id="boton8" value="Eliminar los dos últimos."><br>

<script src="https://code.jquery.com/jquery-3.4.1.min.js"></script>

<script src="funciones.js"></script>

</body>

let x = $(document);

x.ready(inicializarEventos);

function inicializarEventos() {

let x = $("#boton1");

x.click(eliminarElementos)

x = $("#boton2");

x.click(restaurarLista)

x = $("#boton3");

x.click(anadirElementoFinal)

x = $("#boton4");

x.click(anadirElementoPrincipio)

x = $("#boton5");

x.click(eliminarElementoFinal)

x = $("#boton6");

x.click(eliminarElementoPrincipio)

x = $("#boton7");

x.click(eliminarPrimeroSegundo)

x = $("#boton8");

x.click(eliminarDosUltimos)

}

function eliminarElementos() {

let x = $("ul");

x.empty();

}

function restaurarLista() {

$("ul").html('<li>Primer item.</li><li> Segundo item.</li > <li>Tercer item.</li><li>Cuarto item.</li>');

}

function anadirElementoFinal() {

let x = $("ul");

x.append("<li>otro item al final</li>");

}

function anadirElementoPrincipio() {

let x = $("ul");

x.prepend("<li>otro item al principio</li>");

}

function eliminarElementoFinal() {

let x = $("li");

let cantidad = x.length;

x = x.eq(cantidad - 1);

x.remove();

}

function eliminarElementoPrincipio() {

let x = $("li");

x = x.eq(0);

x.remove();

}

function eliminarPrimeroSegundo() {

let x = $("li");

x = x.slice(0, 2);

x.remove();

}

function eliminarDosUltimos() {

let x = $("li");

x = x.slice(x.length - 2);

x.remove();

}

Para borrar todos los elementos contenidos en una lista obtenemos la referencia de la lista mediante la función $ y seguidamente llamamos al método empty:

function eliminarElementos() {

let x = $("ul");

x.empty();

}

Para restaurar la lista utilizamos el método html insertando directamente los item a partir del elemento ul:

function restaurarLista() {

$("ul").html('<li>Primer item.</li><li> Segundo item.</li > <li>Tercer item.</li><li>Cuarto item.</li>');

}

Para añadir un elemento al final de la colección de elementos del objeto jQuery disponemos del método append:

function anadirElementoFinal() {

let x = $("ul");

x.append("<li>otro item al final</li>");

}

Para añadir un elemento al principio disponemos de un método llamado prepend:

function anadirElementoPrincipio() {

let x = $("ul");

x.prepend("<li>otro item al principio</li>");

}

Para eliminar un elemento del final de la lista, primero obtenemos la cantidad de elementos que almacena el objeto jQuery por medio de la propiedad length y luego mediante el método eq (equal) indicamos la posición del elemento que necesitamos (el método eq retorna otro objeto de tipo jQuery) y por último llamamos al método remove:

function eliminarElementoFinal() {

let x = $("li");

let cantidad = x.length;

x = x.eq(cantidad - 1);

x.remove();

}

Para eliminar un elemento del principio es similar a borrar el último, pero aquí no necesitamos saber la cantidad de elementos que almacena el objeto jQuery ya que debemos acceder al primero (el primero se almacena en la posición cero):

function eliminarElementoPrincipio() {

let x = $("li");

x = x.eq(0);

x.remove();

}

Para eliminar el primero y segundo elemento que almacena el objeto jQuery disponemos de un método slice que especifica un rango de índices (en nuestro ejemplo el rango es de la posición 0 hasta la 2 sin incluirla), el método slice retorna todos los elementos que se encuentran en dicho rango y posteriormente llamamos a remove:

function eliminarPrimeroSegundo() {

let x = $("li");

x = x.slice(0, 2);

x.remove();

}

Para eliminar los dos elementos finales podemos también utilizar el método slice (si no indicamos un segundo parámetro luego es hasta el final):

function eliminarDosUltimos() {

let x = $("li");

x = x.slice(x.length - 2);

x.remove();

}

### Efectos con los métodos show y hide:

Una característica muy interesante de jQuery que nos provee de un serie de efectos visuales.

Ya utilizamos los métodos hide() y show(), que tienen por objetivo ocultar y mostrar elementos HTML. Ahora veremos que podemos hacer que cuando se oculte o muestre un elemento lo haga con una pequeña animación (que se oculte o muestre lentamente)

Estas características pueden ayudar al usuario a concentrarse en una parte de la página donde sucede la animación

<head>

<title>Ejemplo de jQuery</title>

<meta charset="UTF-8">

<link rel="StyleSheet" href="estilos.css" type="text/css">

</head>

<body>

<input type="button" id="boton1" value="Ocultar recuadro">

<input type="button" id="boton2" value="Mostrar recuadro">

<div id="descripcion" class="recuadro">

<p>HTML es el lenguaje que se emplea para el desarrollo de páginas

de internet.</p>

<p>Este lenguaje está constituido de elementos que el navegador interpreta

y las despliega

en la pantalla de acuerdo a su objetivo. Veremos que hay elementos para disponer

imágenes

sobre una página, hipervínculos que nos permiten dirigirnos a otra

página, listas, tablas

para tabular datos, etc.</p>

<p>Para poder crear una página HTML se requiere un simple editor

de texto (en nuestro caso

emplearemos este sitio) y un navegador de internet (IExplorer, FireFox etc.),

emplearemos

el navegador que en este preciso momento está utilizando (recuerde que

usted está viendo

en este preciso momento una página HTML con su navegador).</p>

<p>Lo más importante es que en cada concepto desarrolle los ejercicios

propuestos y modifique

los que se presentan ya resueltos.</p>

<p>Este curso lo que busca es acercar el lenguaje HTML a una persona que

nunca antes trabajó

con el mismo. No pretende mostrar todas los elementos HTML en forma alfabética.</p>

<p>Como veremos, de cada concepto se presenta una parte teórica,

en la que se da una

explicación completa, luego se pasa a la sección del ejercicio resuelto

donde podemos

ver el contenido de la página HTML y cómo la visualiza el navegador.

Por último y

tal vez la sección más importante de este tutorial es donde se propone

que usted

haga páginas en forma autónoma (donde realmente podrá darse

cuenta si el concepto quedó firme).</p>

</div>

<script src="https://code.jquery.com/jquery-3.4.1.min.js"></script>

<script src="funciones.js"></script>

</body>

let x = $(document);

x.ready(inicializarEventos);

function inicializarEventos() {

let x = $("#boton1");

x.click(ocultarRecuadro);

x = $("#boton2");

x.click(mostrarRecuadro);

}

function ocultarRecuadro() {

let x = $("#descripcion");

x.hide("slow");

}

function mostrarRecuadro() {

let x = $("#descripcion");

x.show("fast");

}

.recuadro {

background-color:#ffffcc;

font-family:verdana;

font-size:14px;

border-top-width:1px;

border-right-width:3px;

border-bottom-width:3px;

border-left-width:1px;

border-top-style:dotted;

border-right-style:solid;

border-bottom-style:solid;

border-left-style:dotted;

border-top-color:#ffaa00;

border-right-color:#ff0000;

border-bottom-color:#ff0000;

border-left-color:#ffaa00;

}

La función ocultarRecuadro se ejecuta cuando presionamos el botón de "Ocultar recuadro":

function ocultarRecuadro() {

let x = $("#descripcion");

x.hide("slow");

}

donde obtenemos la referencia del div mediante su id y procedemos a llamar al método hide pasándole el string "slow", con esto logramos que se oculte el recuadro lentamente.

De forma similar la función mostrarRecuadro:

function mostrarRecuadro() {

let x = $("#descripcion");

x.show("fast");

}

llama a la función show pasando como parámetro el string "fast".

Hay varias formas para llamar a los métodos show y hide:

Lo muestra en forma instantanea:

show()

Lo muestra con una animación rápida:

show("fast")

Lo muestra con una animación normal:

show("normal")

Lo muestra con una animación lenta:

show("slow")

Lo muestra con una animación que tarda tantos milisegundos como le indicamos:

show([cantidad de milisegundos])

Lo muestra con una animación que tarda tantos milisegundos como le indicamos y ejecuta al final la función que le pasamos como segundo parámetro:

show([cantidad de milisegundos],[función])

De forma similar funciona el método hide.

### Efectos con los métodos fadeIn y fadeOut:

Estos dos métodos son similares a show y hide pero con la diferencia que fadeIn hace aparecer los elementos HTML con opacidad. El método fadeOut decolora hasta desaparecer, es decir reduce la opacidad del elemento en forma progresiva.

<head>

<title>Ejemplo de jQuery</title>

<meta charset="UTF-8">

<link rel="StyleSheet" href="estilos.css" type="text/css">

</head>

<body>

<input type="button" id="boton1" value="Ocultar recuadro">

<input type="button" id="boton2" value="Mostrar recuadro">

<div id="descripcion" class="recuadro">

<p>HTML es el lenguaje que se emplea para el desarrollo de páginas

de internet.</p>

<p>Este lenguaje está constituido de elementos que el navegador interpreta

y las despliega en la pantalla

de acuerdo a su objetivo. Veremos que hay elementos para disponer imágenes

sobre una página, hipervínculos

que nos permiten dirigirnos a otra página, listas, tablas para tabular

datos, etc.</b>

<p>Para poder crear una página HTML se requiere un simple editor

de texto (en nuestro caso emplearemos este

sitio) y un navegador de internet (IExplorer, FireFox etc.), emplearemos el navegador

que en este preciso

momento está utilizando (recuerde que usted está viendo en este

preciso momento una página HTML con su

navegador).</p>

<p>Lo más importante es que en cada concepto desarrolle los ejercicios

propuestos y modifique los que se

presentan ya resueltos.</p>

<p>Este curso lo que busca es acercar el lenguaje HTML a una persona que

nunca antes trabajó con el mismo.

No pretende mostrar todas los elementos HTML en forma alfabética.</p>

<p>Como veremos, de cada concepto se presenta una parte teórica,

en la que se da una explicación completa,

luego se pasa a la sección del ejercicio resuelto donde podemos ver el

contenido de la página HTML y cómo

la visualiza el navegador. Por último y tal vez la sección más

importante de este tutorial es donde se

propone que usted haga páginas en forma autónoma (donde realmente

podrá darse cuenta si el concepto quedó

firme).</p>

</div>

<script src="https://code.jquery.com/jquery-3.4.1.min.js"></script>

<script src="funciones.js"></script>

</body>

let x = $(document);

x.ready(inicializarEventos);

function inicializarEventos() {

let x = $("#boton1");

x.click(ocultarRecuadro);

x = $("#boton2");

x.click(mostrarRecuadro);

}

function ocultarRecuadro() {

let x = $("#descripcion");

x.fadeOut("slow");

}

function mostrarRecuadro() {

let x = $("#descripcion");

x.fadeIn("slow");

}

.recuadro {

background-color:#ffffcc;

font-family:verdana;

font-size:14px;

border-top-width:1px;

border-right-width:3px;

border-bottom-width:3px;

border-left-width:1px;

border-top-style:dotted;

border-right-style:solid;

border-bottom-style:solid;

border-left-style:dotted;

border-top-color:#ffaa00;

border-right-color:#ff0000;

border-bottom-color:#ff0000;

border-left-color:#ffaa00;

}

La función ocultarRecuadro se ejecuta cuando presionamos el botón de "Ocultar recuadro":

function ocultarRecuadro() {

let x = $("#descripcion");

x.fadeOut("slow");

}

donde obtenemos la referencia del div mediante su id y procedemos a llamar al método fadeOut pasándole el string "slow", con esto logramos que se oculte el recuadro y se decolore lentamente hasta desaparecer.

De forma similar la función mostrarRecuadro:

function mostrarRecuadro() {

let x = $("#descripcion");

x.fadeIn("slow");

}

llama a la función fadeIn pasando como parámetro el string "slow".

Las formas de llamar a los métodos fadeIn y fadeOut:

Lo muestra con una animación rápida:

fadeIn("fast")

Lo muestra con una animación normal:

fadeIn("normal")

Lo muestra con una animación lenta:

fadeIn("slow")

Lo muestra con una animación que tarda tantos milisegundos como le indicamos:

fadeIn([cantidad de milisegundos])

Lo muestra con una animación que tarda tantos milisegundos como le indicamos y ejecuta al final la función que le pasamos como segundo parámetro:

fadeIn([cantidad de milisegundos],[función])

De forma similar funciona el método fadeOut.

### Efecto con el método fadeTo:

El método fadeTo puede modificar la opacidad de un elemento, el efecto es llevar la opacidad actual hasta el valor que le pasamos al método fadeTo

Podemos inicializar este método de las siguientes formas:

fadeTo([velocidad],[valor de opacidad])

Indicamos la velocidad de transición del estado actual al nuevo estado (slow/normal/fast) o un valor indicado en milisegúndos.

El valor de la opacidad es un numero real entre 0 y 1. 1 significa sin opacidad y 0 es transparente.

También podemos llamar al método fadeTo con tres parámetros:

fadeTo([velocidad],[valor de opacidad],[función])

Esta segunda estructura de la función permite ejecutar una función cuando finaliza la transición.

Hay que tener en cuenta que fadeTo por más que indiquemos el valor 0 en opacidad el espacio que ocupa el elemento en la página seguirá ocupado por un recuadro vacío.

<head>

<title>Ejemplo de jQuery</title>

<meta charset="UTF-8">

<link rel="StyleSheet" href="estilos.css" type="text/css">

</head>

<body>

<input type="button" id="boton1" value="Reducir opacidad">

<input type="button" id="boton2" value="Aumentar opacidad">

<div id="descripcion" class="recuadro">

<p>HTML es el lenguaje que se emplea para el desarrollo de páginas

de internet.</p>

<p>Este lenguaje está constituido de elementos que el navegador interpreta

y las despliega en la pantalla de acuerdo a su objetivo. Veremos que hay elementos

para disponer imágenes sobre una página, hipervínculos que

nos permiten dirigirnos a otra página, listas, tablas para tabular datos, etc.</p>

<p>Para poder crear una página HTML se requiere un simple editor

de texto (en nuestro caso emplearemos este sitio) y un navegador de internet (IExplorer,

FireFox etc.), emplearemos el navegador que en este preciso momento está

utilizando (recuerde que usted está viendo en este preciso momento una

página HTML con su navegador).</p>

<p>Lo más importante es que en cada concepto desarrolle los ejercicios

propuestos y modifique los que se presentan ya resueltos.</p>

<p>Este curso lo que busca es acercar el lenguaje HTML a una persona que

nunca antes trabajó con el mismo. No pretende mostrar todas los elementos

HTML en forma alfabética.</p>

<p>Como veremos, de cada concepto se presenta una parte teórica,

en la que se da una explicación completa, luego se pasa a la sección

del ejercicio resuelto donde podemos ver el contenido de la página HTML

y cómo la visualiza el navegador. Por último y tal vez la sección

más importante de este tutorial es donde se propone que usted haga páginas

en forma autónoma (donde realmente podrá darse cuenta si el concepto

quedó firme).</p>

</div>

<script src="https://code.jquery.com/jquery-3.4.1.min.js"></script>

<script src="funciones.js"></script>

</body>

let x = $(document);

x.ready(inicializarEventos);

function inicializarEventos() {

let x = $("#boton1");

x.click(reducirOpacidadRecuadro);

x = $("#boton2");

x.click(aumentarOpacidadRecuadro);

}

function reducirOpacidadRecuadro() {

let x = $("#descripcion");

x.fadeTo("slow", 0.5);

}

function aumentarOpacidadRecuadro() {

let x = $("#descripcion");

x.fadeTo("slow", 1);

}

.recuadro {

background-color:#ffffcc;

font-family:verdana;

font-size:14px;

border-top-width:1px;

border-right-width:3px;

border-bottom-width:3px;

border-left-width:1px;

border-top-style:dotted;

border-right-style:solid;

border-bottom-style:solid;

border-left-style:dotted;

border-top-color:#ffaa00;

border-right-color:#ff0000;

border-bottom-color:#ff0000;

border-left-color:#ffaa00;

}

Cuando se presiona el botón de reducir la opacidad llamamos al método fadeTo con el valor 0.5:

function reducirOpacidadRecuadro() {

let x = $("#descripcion");

x.fadeTo("slow", 0.5);

}

y cuando presionamos el botón de aumentar la opacidad llamamos al método fadeTo con valor 1 en opacidad (significa opacidad total):

function aumentarOpacidadRecuadro() {

let x = $("#descripcion");

x.fadeTo("slow", 1);

}

### Efecto con el método toggle:

El método toggle permite cada vez que se ejecute cambiar de estado la visibilidad del elemento HTML, es decir si está visible pasa a oculto y si se encuentra oculto pasa a visible.

<head>

<title>Ejemplo de jQuery</title>

<meta charset="UTF-8">

<link rel="StyleSheet" href="estilos.css" type="text/css">

</head>

<body>

<input type="button" id="boton1" value="Mostrar/Ocultar">

<div id="descripcion" class="recuadro">

<p>HTML es el lenguaje que se emplea para el desarrollo de páginas

de internet.</p>

<p>Este lenguaje está constituido de elementos que el navegador interpreta

y las despliega en la pantalla

de acuerdo a su objetivo. Veremos que hay elementos para disponer imágenes

sobre una página, hipervínculos

que nos permiten dirigirnos a otra página, listas, tablas para tabular

datos, etc.</p>

<p>Para poder crear una página HTML se requiere un simple editor

de texto (en nuestro caso emplearemos

este sitio) y un navegador de internet (IExplorer, FireFox etc.), emplearemos

el navegador que en este preciso momento está utilizando

(recuerde que usted está viendo en este preciso momento una página HTML

con su navegador).</p>

<p>Lo más importante es que en cada concepto desarrolle los ejercicios

propuestos y modifique los que se presentan ya resueltos.</p>

<p>Este curso lo que busca es acercar el lenguaje HTML a una persona que

nunca antes trabajó con el mismo.

No pretende mostrar todas los elementos HTML en forma alfabética.</p>

<p>Como veremos, de cada concepto se presenta una parte teórica,

en la que se da una explicación completa,

luego se pasa a la sección del ejercicio resuelto donde podemos ver el

contenido de la página HTML y cómo la visualiza el navegador.

Por último y tal vez la sección más importante de este tutorial es donde se

propone que usted haga páginas en forma autónoma (donde realmente

podrá darse cuenta si el concepto quedó

firme).</p>

</div>

<script src="https://code.jquery.com/jquery-3.4.1.min.js"></script>

<script src="funciones.js"></script>

</body>

let x = $(document);

x.ready(inicializarEventos);

function inicializarEventos() {

let x = $("#boton1");

x.click(ocultarMostrarRecuadro);

}

function ocultarMostrarRecuadro() {

let x = $("#descripcion");

x.toggle("slow");

}

.recuadro {

background-color:#ffffcc;

font-family:verdana;

font-size:14px;

border-top-width:1px;

border-right-width:3px;

border-bottom-width:3px;

border-left-width:1px;

border-top-style:dotted;

border-right-style:solid;

border-bottom-style:solid;

border-left-style:dotted;

border-top-color:#ffaa00;

border-right-color:#ff0000;

border-bottom-color:#ff0000;

border-left-color:#ffaa00;

}

La función ocultarMostrarRecuadro:

function ocultarMostrarRecuadro() {

let x = $("#descripcion");

x.toggle("slow");

}

se encarga de llamar al método toggle del objeto jQuery, y este analiza si actualmente el elemento está visible u oculto, la transición se hace en forma lenta ya que le pasamos como parámetro el string "slow".

### Iteración por los elementos (each):

jQuery dispone de un método que nos permite asociar una función que se ejecutará por cada elemento que contenga la lista del objeto jQuery.

Dentro de esta función podemos manipular el elemento actual.

La sintaxis del iterador each es:

var x;

x=$([elementos]);

x.each([nombre de funcion])

<body>

<div id="parrafos">

<p>párrafo 1 - párrafo 1 - párrafo 1 - párrafo

1 - párrafo 1 - párrafo 1 - párrafo 1 - párrafo 1 -

párrafo 1 - párrafo 1 - párrafo 1 - párrafo 1 - párrafo

1 - párrafo 1 - párrafo 1 - párrafo 1 -

párrafo 1 - párrafo 1 - párrafo 1 - párrafo 1 - párrafo

1 - párrafo 1 - párrafo 1 - párrafo 1 -

párrafo 1 - párrafo 1 - párrafo 1 - párrafo 1 - párrafo

1 - párrafo 1 - párrafo 1 - párrafo 1</p>

<p>párrafo 2 - párrafo 2 - párrafo 2 - párrafo 2</p>

<p>párrafo 3 - párrafo 3 - párrafo 3 - párrafo

3 - párrafo 3 - párrafo 3 - párrafo 3 -

párrafo 3 - párrafo 3 - párrafo 3 - párrafo 3 - párrafo

3 - párrafo 3 - párrafo 3 -

párrafo 3 - párrafo 3 - párrafo 3 - párrafo 3 - párrafo

3 - párrafo 3 - párrafo 3 - párrafo 3 -

párrafo 3 - párrafo 3 - párrafo 3 - párrafo 3 - párrafo

3 - párrafo 3</p>

<p>párrafo 4 - párrafo 4 - párrafo 4 - párrafo

4 - párrafo 4</p>

</div>

<script src="https://code.jquery.com/jquery-3.4.1.min.js"></script>

<script src="funciones.js"></script>

</body>

let x = $(document);

x.ready(inicializarEventos);

function inicializarEventos() {

let x = $("#parrafos p");

x.each(resaltarParrafos);

}

function resaltarParrafos() {

let x = $(this);

if (x.text().length < 100) {

x.css("background-color", "#ff0");

}

}

En la función inicializarEventos obtenemos la lista de párrafos contenidas en el div y luego a este objeto jQuery mediante el método each indicamos el nombre de la función que procesará a cada elemento:

let x = $("#parrafos p");

x.each(resaltarParrafos);

La función resaltarParrafos se ejecuta por cada uno de los párrafos contenidos en el objeto jQuery (en nuestro caso 4 párrafos):

function resaltarParrafos() {

let x = $(this);

if (x.text().length < 100) {

x.css("background-color", "#ff0");

}

}

Lo primero que hacemos mediante la función $ es obtener la referencia del objeto a procesar. Accedemos al método text() que nos retorna el texto del párrafo propiamente dicho.

El método text() retorna un string por lo que podemos acceder al atributo length y verificar la cantidad de caracteres que tiene, en caso de ser inferior a 100 procedemos a cambiar el color de fondo de dicho párrafo.

# Bootstrap

* Bootstrap es un marco front-end gratuito para un desarrollo web más rápido y sencillo
* Bootstrap incluye plantillas de diseño basadas en HTML y CSS para tipografía, formularios, botones, tablas, navegación, modales, carruseles de imágenes y muchos otros, así como complementos de JavaScript opcionales.
* Bootstrap también le brinda la capacidad de crear fácilmente diseños receptivos

**¿Qué es el diseño web adaptable?**

El diseño web receptivo consiste en crear sitios web que se ajustan automáticamente para verse bien en todos los dispositivos, desde teléfonos pequeños hasta computadoras de escritorio grandes.

**¿Por qué utilizar Bootstrap?**

Ventajas de Bootstrap:

* Fácil de usar: cualquier persona con conocimientos básicos de HTML y CSS puede comenzar a usar Bootstrap
* Funciones receptivas : el CSS receptivo de Bootstrap se ajusta a teléfonos, tabletas y computadoras de escritorio
* Enfoque de dispositivos móviles primero: en Bootstrap 3, los estilos de dispositivos móviles primero son parte del marco central
* Compatibilidad del navegador: Bootstrap es compatible con todos los navegadores modernos (Chrome, Firefox, Internet Explorer, Edge, Safari y Opera)

**Bootstrap CDN**

Si no desea descargar y alojar Bootstrap usted mismo, puede incluirlo desde una CDN (Red de entrega de contenido).

MaxCDN proporciona soporte CDN para CSS y JavaScript de Bootstrap. También debe incluir jQuery:

<!-- Latest compiled and minified CSS -->  
<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/css/bootstrap.min.css">  
  
<!-- jQuery library -->  
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>  
  
<!-- Latest compiled JavaScript -->  
<script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/js/bootstrap.min.js"></script>

## Cree la primera página web con Bootstrap

1. Agregue el tipo de documento HTML5:

Bootstrap usa elementos HTML y propiedades CSS que requieren el tipo de documento HTML5.

Incluya siempre el tipo de documento HTML5 al principio de la página, junto con el atributo lang y el juego de caracteres correcto:

<!DOCTYPE html>  
<html lang="en">  
  <head>  
    <meta charset="utf-8">  
  </head>  
</html>

1. Bootstrap 3 es móvil primero:

Bootstrap 3 está diseñado para responder a dispositivos móviles. Los estilos de dispositivos móviles primero son parte del marco central.

Para garantizar una representación adecuada y un zoom táctil, agregue la siguiente <meta>etiqueta dentro del <head>elemento:

<meta name="viewport" content="width=device-width, initial-scale=1">

La width=device-widthparte establece el ancho de la página para seguir el ancho de la pantalla del dispositivo (que variará según el dispositivo).

La initial-scale=1parte establece el nivel de zoom inicial cuando el navegador carga la página por primera vez.

1. Contenedores:

Bootstrap también requiere un elemento contenedor para envolver el contenido del sitio.

Hay dos clases de contenedores para elegir:

* La .containerclase proporciona un contenedor de ancho fijo sensible
* La .container-fluidclase proporciona un contenedor de ancho completo, que abarca todo el ancho de la ventana gráfica.



El siguiente ejemplo muestra el código para una página de Bootstrap básica (con un contenedor de ancho fijo receptivo):

<body>  
<div class="container">  
  <h1>My First Bootstrap Page</h1>  
  <p>This is some text.</p>  
</div>  
</body>

El siguiente ejemplo muestra el código para una página Bootstrap básica (con un contenedor de ancho completo):

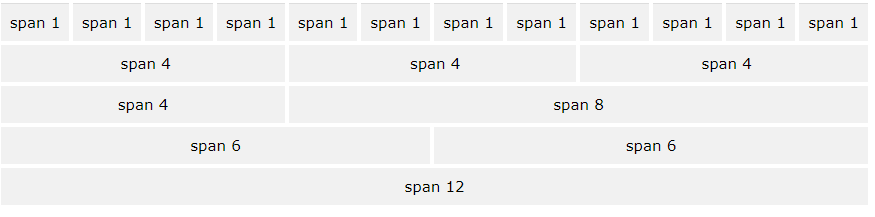
<body>  
<div class="container-fluid">  
  <h1>My First Bootstrap Page</h1>  
  <p>This is some text.</p>  
</div>  
</body>

## Cuadrículas Bootstrap

### Sistema de cuadrícula Bootstrap:

El sistema de cuadrícula de Bootstrap permite hasta 12 columnas en la página.

Si no desea utilizar las 12 columnas individualmente, puede agrupar las columnas para crear columnas más anchas:



El sistema de cuadrícula de Bootstrap responde y las columnas se reorganizarán automáticamente según el tamaño de la pantalla.

### Clases de cuadrícula:

El sistema de cuadrícula Bootstrap tiene cuatro clases:

* xs (para teléfonos: pantallas de menos de 768 px de ancho)
* sm (para tabletas: pantallas iguales o superiores a 768 px de ancho)
* md (para portátiles pequeños: pantallas iguales o superiores a 992 px de ancho)
* lg (para computadoras portátiles y de escritorio: pantallas iguales o superiores a 1200 px de ancho)

Las clases anteriores se pueden combinar para crear diseños más dinámicos y flexibles.

### Reglas del sistema de cuadrícula:

* Las filas deben colocarse dentro de un .container(ancho fijo) o .container-fluid(ancho completo) para una alineación y relleno adecuados
* Use filas para crear grupos horizontales de columnas
* El contenido debe colocarse dentro de las columnas y solo las columnas pueden ser elementos secundarios inmediatos de las filas.
* Las clases predefinidas como .row y .col-sm-4están disponibles para crear rápidamente diseños de cuadrícula
* Las columnas crean medianiles (espacios entre el contenido de la columna) mediante el relleno. Ese relleno se compensa en filas para la primera y la última columna a través de un margen negativo en.rows
* Las columnas de la cuadrícula se crean especificando el número de 12 columnas disponibles que desea abarcar. Por ejemplo, tres columnas iguales usarían tres.col-sm-4
* Los anchos de columna están en porcentaje, por lo que siempre son fluidos y tienen un tamaño relativo a su elemento principal

### Estructura básica de una cuadrícula de Bootstrap:

La siguiente es una estructura básica de una cuadrícula de Bootstrap:

<div class="row">  
  <div class="col-\*-\*"></div>  
  <div class="col-\*-\*"></div>  
</div>  
<div class="row">  
  <div class="col-\*-\*"></div>  
  <div class="col-\*-\*"></div>  
  <div class="col-\*-\*"></div>  
</div>  
<div class="row">  
  ...  
</div>

Primero; crea una fila ( <div class="row">). Luego, agregue el número deseado de columnas (etiquetas con las .col-\*-\*clases apropiadas ). Tenga en cuenta que los números en .col-\*-\*siempre deben sumar 12 para cada fila.

**Tres columnas iguales**

El siguiente ejemplo muestra cómo obtener tres columnas de igual ancho que comienzan en tabletas y escalan a escritorios grandes. En teléfonos móviles o pantallas de menos de 768 px de ancho, las columnas se apilarán automáticamente:

<div class="row">  
  <div class="col-sm-4">.col-sm-4</div>  
  <div class="col-sm-4">.col-sm-4</div>  
  <div class="col-sm-4">.col-sm-4</div>  
</div>

**Dos columnas desiguales**

El siguiente ejemplo muestra cómo obtener dos columnas de varios anchos comenzando en tabletas y escalando a escritorios grandes:

<div class="row">  
  <div class="col-sm-4">.col-sm-4</div>  
  <div class="col-sm-8">.col-sm-8</div>  
</div>

## Tipografia:

### Configuración predeterminada de Bootstrap:

El tamaño de fuente predeterminado global de Bootstrap es 14px, con una altura de línea de 1.428.

Esto se aplica al elemento<body> y a todos los párrafos (<p>).

Además, todos los elementos<p> tienen un margen inferior que equivale a la mitad de su altura de línea calculada (10 px por defecto).

### Bootstrap frente a valores predeterminados del navegador:

En este capítulo, veremos algunos elementos HTML a los que Bootstrap les aplicará un estilo un poco diferente a los valores predeterminados del navegador.

<https://www.w3schools.com/bootstrap/bootstrap_typography.asp>

### Colores y fondos contextuales:

Bootstrap también tiene algunas clases contextuales que pueden usarse para proporcionar "significado a través de colores".

Las clases para los colores de texto son: .text-muted, .text-primary, .text-success, .text-info, .text-warning, y .text-danger:

This text is muted.

This text is important.

This text indicates success.

This text represents some information.

This text represents a warning.

This text represents danger.

Las clases para los colores de fondo son: .bg-primary, .bg-success, .bg-info, .bg-warning, y .bg-danger:

This text is important.

This text indicates success.

This text represents some information.

This text represents a warning.

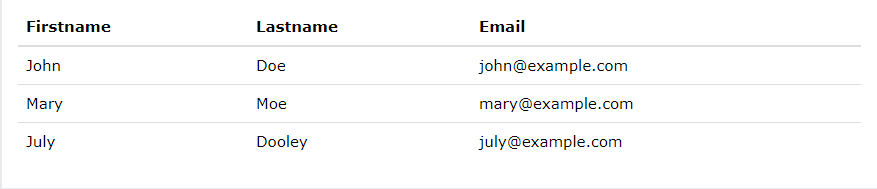
This text represents danger.

## Tablas de Bootstrap

### Tabla básica de Bootstrap:

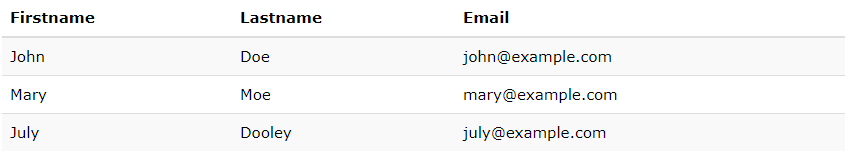
Una tabla Bootstrap básica tiene solo separadores horizontales.

La clase .table agrega estilo básico a una tabla:



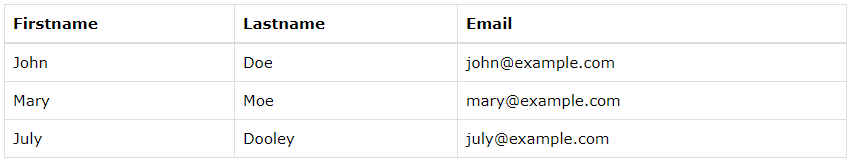
### Striped Rows:

La clase .table-striped agrega rayas de cebra a una mesa:



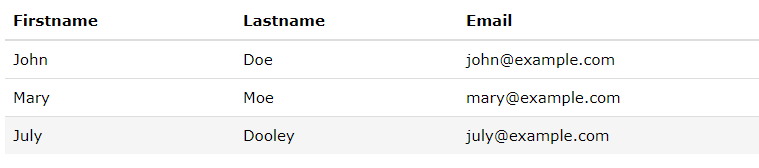
### Bordered Table:

La clase .table-bordered agrega bordes en todos los lados de la tabla y las celdas:



### Hover Rows:

La clase .table-hover agrega un efecto de desplazamiento (color de fondo gris) en las filas de la tabla donde este el cursor:



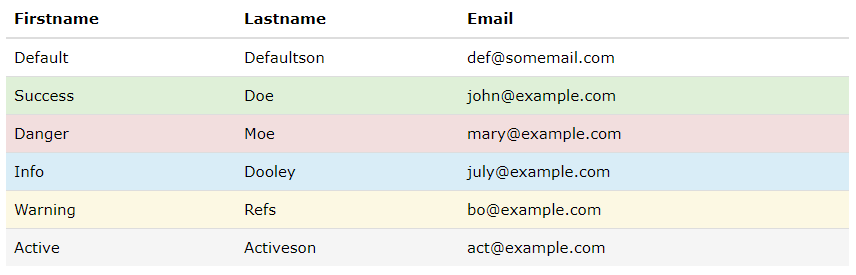
### Condensed Table:

La clase .table-condensed hace que una tabla sea más compacta al cortar el acolchado de la celda a la mitad:



### Clases contextuales:

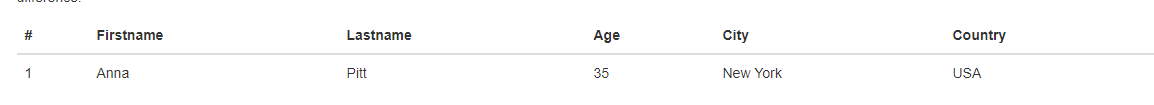
Las clases contextuales se pueden utilizar para colorear las filas de la tabla ( <tr>) o las celdas de la tabla ( <td>):

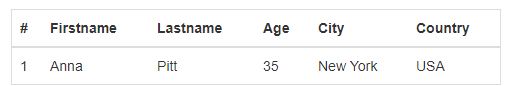


* .active: Aplica el color de desplazamiento a la fila de la tabla o la celda de la tabla
* .success: Indica una acción exitosa o positiva
* .info: Indica un cambio o acción informativa neutral
* .warning: Indica una advertencia que podría necesitar atención.
* .danger: Indica una acción peligrosa o potencialmente negativa

### Responsive Tables:

La .table-responsiveclase crea una tabla receptiva. Luego, la tabla se desplazará horizontalmente en dispositivos pequeños (menos de 768px). Cuando se ve en cualquier pantalla de más de 768 px de ancho, no hay diferencia.





## Imágenes de Bootstrap

### Esquinas redondeadas:

La clase .img-rounded agrega esquinas redondeadas a una imagen (IE8 no admite esquinas redondeadas)

<img src="cinqueterre.jpg" class="img-rounded" alt="Cinque Terre">

### Circulo:

La clase .img-circle da forma a la imagen en un círculo (IE8 no admite esquinas redondeadas)

<img src="cinqueterre.jpg" class="img-circle" alt="Cinque Terre">

### Miniatura :

La clase .img-thumbnail da forma a la imagen en una miniatura:

<img src="cinqueterre.jpg" class="img-thumbnail" alt="Cinque Terre">

### Responsive Images :

Las imágenes vienen en todos los tamaños. Lo mismo ocurre con las pantallas. Las imágenes receptivas se ajustan automáticamente para adaptarse al tamaño de la pantalla.

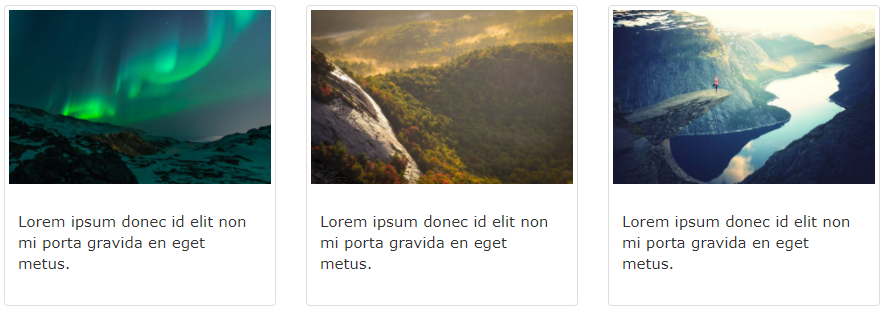
Cree imágenes receptivas agregando una .img-responsiveclase a la <img>etiqueta. Luego, la imagen se escalará bien al elemento principal.

La .img-responsiveclase se aplica display: block;y max-width: 100%;y height: auto;a la imagen:

<img class="img-responsive" src="img\_chania.jpg" alt="Chania">

### Galería de imágenes:

También puede usar el sistema de cuadrícula de Bootstrap junto con la clase .thumbnail para crear una galería de imágenes.



<div class="row">  
  <div class="col-md-4">  
    <div class="thumbnail">  
      <a href="/w3images/lights.jpg">  
        <img src="/w3images/lights.jpg" alt="Lights" style="width:100%">  
        <div class="caption">  
          <p>Lorem ipsum...</p>  
        </div>  
      </a>  
    </div>  
  </div>  
  <div class="col-md-4">  
    <div class="thumbnail">  
      <a href="/w3images/nature.jpg">  
        <img src="/w3images/nature.jpg" alt="Nature" style="width:100%">  
        <div class="caption">  
          <p>Lorem ipsum...</p>  
        </div>  
      </a>  
    </div>  
  </div>  
  <div class="col-md-4">  
    <div class="thumbnail">  
      <a href="/w3images/fjords.jpg">  
        <img src="/w3images/fjords.jpg" alt="Fjords" style="width:100%">  
        <div class="caption">  
          <p>Lorem ipsum...</p>  
        </div>  
      </a>  
    </div>  
  </div>  
</div>

### Incrustaciones receptivas:

También permita que los videos o presentaciones de diapositivas se escalen correctamente en cualquier dispositivo.

Las clases se pueden aplicar directamente a los elementos <iframe>, <embed>, <video> y <object>.

El siguiente ejemplo crea un video receptivo agregando una .embed-responsive-itemclase a una <iframe>etiqueta (el video luego escalará bien al elemento principal). El contenido <div>define la relación de aspecto del video:

<div class="embed-responsive embed-responsive-16by9">  
  <iframe class="embed-responsive-item" src="..."></iframe>  
</div>

## Bootstrap Jumbotron y encabezado de página

### Creando un Jumbotron:

Un jumbotron indica una caja grande para llamar la atención sobre algún contenido o información especial.

Un jumbotron se muestra como un cuadro gris con esquinas redondeadas. También amplía los tamaños de fuente del texto que contiene.

Sugerencia: dentro de un jumbotron puede colocar casi cualquier HTML válido, incluidos otros elementos / clases de Bootstrap.

Usa un <div>elemento con clase .jumbotronpara crear un jumbotron

### Contenedor Jumbotron:

Coloque el jumbotron dentro del <div class="container">si desea que el jumbotron NO se extienda hasta el borde de la pantalla:

<div class="container">  
  <div class="jumbotron">  
    <h1>Bootstrap Tutorial</h1>  
    <p>Bootstrap is the most popular HTML, CSS, and JS framework for developing  
    responsive, mobile-first projects on the web.</p>  
  </div>  
  <p>This is some text.</p>  
  <p>This is another text.</p>  
</div>

Y fuera del <div class="container">si desea que el jumbotron se extienda hasta los bordes de la pantalla.

### Encabezado de página de ejemplo:

Usa un elemento <div> con clase .page-header para crear un encabezado de página:

<div class="page-header">  
  <h1>Example Page Header</h1>  
</div>

## Wells

La clase .well agrega un borde redondeado alrededor de un elemento con un color de fondo gris y algo de relleno

<div class="well">Basic Well</div>

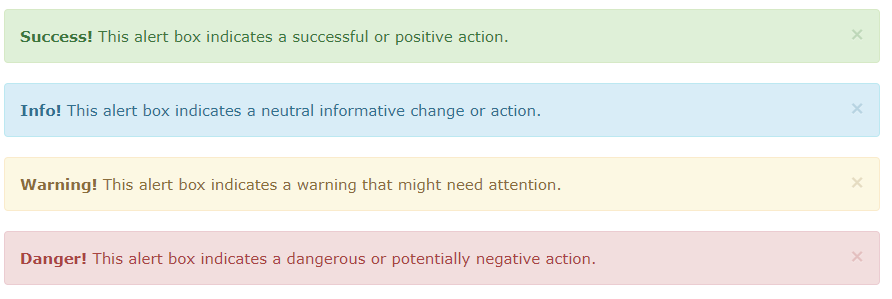
<div class="well well-sm">Small Well</div>

<div class="well well-lg">Large Well</div>



## Alertas de Bootstrap

Bootstrap proporciona una manera fácil de crear mensajes de alerta predefinidos:

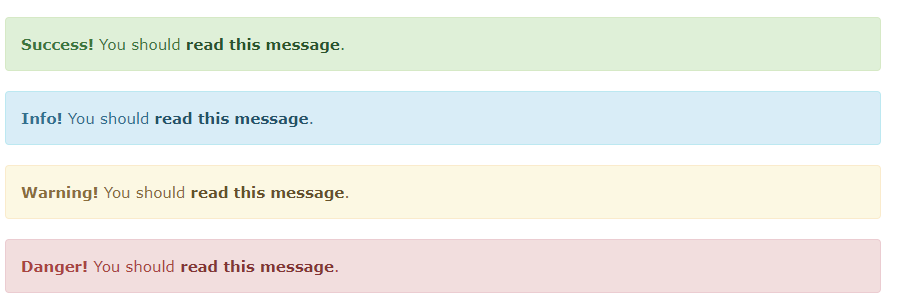


Las alertas se crean con la clase .alert, seguida de una de las cuatro clases contextuales .alert-success, .alert-info, .alert-warningo .alert-danger:

<div class="alert alert-success">  
  <strong>Success!</strong> Indicates a successful or positive action.  
</div>  
  
<div class="alert alert-info">  
  <strong>Info!</strong> Indicates a neutral informative change or action.  
</div>  
  
<div class="alert alert-warning">  
  <strong>Warning!</strong> Indicates a warning that might need attention.  
</div>  
  
<div class="alert alert-danger">  
  <strong>Danger!</strong> Indicates a dangerous or potentially negative action.  
</div>

### Enlaces de alerta:

Agregue la clase alert-link a cualquier enlace dentro del cuadro de alerta para crear "enlaces de colores coincidentes":

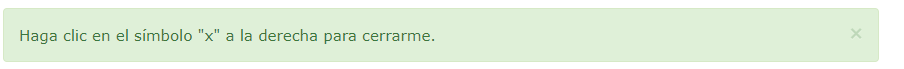


<div class="alert alert-success">  
  <strong>Success!</strong> You should <a href="#" class="alert-link">read this message</a>.  
</div>

### Alertas de cierre:

Para cerrar el mensaje de alerta, agregue una clase .alert-dismissible al contenedor de alertas. Luego agregue class="close"y data-dismiss="alert" a un enlace o un elemento de botón (cuando haga clic en este, el cuadro de alerta desaparecerá).

<div class="alert alert-success alert-dismissible">  
  <a href="#" class="close" data-dismiss="alert" aria-label="close">&times;</a>  
  <strong>Success!</strong> Indicates a successful or positive action.  
</div>



### Alertas animadas:

Las clases .fade y .in agregan un efecto de desvanecimiento al cerrar el mensaje de alerta:

<div class="alert alert-danger fade in">

## Botones Bootstrap

### Estilos de botones:

Bootstrap proporciona diferentes estilos de botones:



Para lograr los estilos de botones anteriores, Bootstrap tiene las siguientes clases:

* .btn
* .btn-default
* .btn-primary
* .btn-success
* .btn-info
* .btn-warning
* .btn-danger
* .btn-link

<button type="button" class="btn">Basic</button>  
<button type="button" class="btn btn-default">Default</button>  
<button type="button" class="btn btn-primary">Primary</button>  
<button type="button" class="btn btn-success">Success</button>  
<button type="button" class="btn btn-info">Info</button>  
<button type="button" class="btn btn-warning">Warning</button>  
<button type="button" class="btn btn-danger">Danger</button>  
<button type="button" class="btn btn-link">Link</button>

Las clases botón se puede utilizar en una <a>, <button>o <input>elemento:

<a href="#" class="btn btn-info" role="button">Link Button</a>  
<button type="button" class="btn btn-info">Button</button>  
<input type="button" class="btn btn-info" value="Input Button">  
<input type="submit" class="btn btn-info" value="Submit Button">

### Tamaños de botones:

Bootstrap proporciona cuatro tamaños de botones:



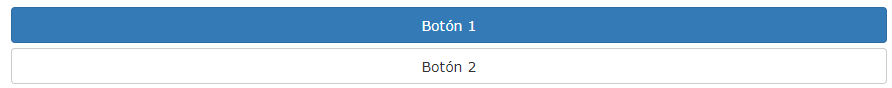
Las clases que definen los diferentes tamaños son:

* .btn-lg
* .btn-sm
* .btn-xs

<button type="button" class="btn btn-primary btn-lg">Large</button>  
<button type="button" class="btn btn-primary">Normal</button>  
<button type="button" class="btn btn-primary btn-sm">Small</button>  
<button type="button" class="btn btn-primary btn-xs">XSmall</button>

### Botones de nivel de bloque:

Un botón de nivel de bloque abarca todo el ancho del elemento principal.



Agregar clase .btn-blockpara crear un botón de nivel de bloque:

<button type="button" class="btn btn-primary btn-block">Button 1</button>

### Botones activos / desactivados:

Un botón se puede establecer en un estado activo (parecer presionado) o deshabilitado (no se puede hacer clic):



La clase .active hace que un botón aparezca presionado y la clase .disabled hace que no se pueda hacer clic en un botón:

<button type="button" class="btn btn-primary active">Active Primary</button>  
<button type="button" class="btn btn-primary disabled">Disabled Primary</button>

## Grupos de botones de Bootstrap

Bootstrap le permite agrupar una serie de botones (en una sola línea) en un grupo de botones:



Use un <div>elemento con clase .btn-grouppara crear un grupo de botones:

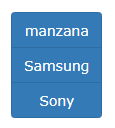
<div class="btn-group">  
  <button type="button" class="btn btn-primary">Apple</button>  
  <button type="button" class="btn btn-primary">Samsung</button>  
  <button type="button" class="btn btn-primary">Sony</button>  
</div>

Consejo: en lugar de aplicar tamaños de botón a todos los botones de un grupo, use la clase .btn-group-lg|sm|xs para cambiar el tamaño de todos los botones del grupo:

<div class="btn-group btn-group-lg">  
  <button type="button" class="btn btn-primary">Apple</button>  
  <button type="button" class="btn btn-primary">Samsung</button>  
  <button type="button" class="btn btn-primary">Sony</button>  
</div>

### Grupos de botones verticales:

Bootstrap también admite grupos de botones verticales:



Utilice la clase .btn-group-vertical para crear un grupo de botones vertical:

<div class="btn-group-vertical">  
  <button type="button" class="btn btn-primary">Apple</button>  
  <button type="button" class="btn btn-primary">Samsung</button>  
  <button type="button" class="btn btn-primary">Sony</button>  
</div>

### Grupos de botones justificados:

Para abarcar todo el ancho de la pantalla, use la clase .btn-group-justified:



<div class="btn-group btn-group-justified">  
  <a href="#" class="btn btn-primary">Apple</a>  
  <a href="#" class="btn btn-primary">Samsung</a>  
  <a href="#" class="btn btn-primary">Sony</a>  
</div>

Nota: Para los elementos <button>, debe envolver cada botón en una clase .btn-group:

<div class="btn-group btn-group-justified">  
  <div class="btn-group">  
    <button type="button" class="btn btn-primary">Apple</button>  
  </div>  
  <div class="btn-group">  
    <button type="button" class="btn btn-primary">Samsung</button>  
  </div>  
  <div class="btn-group">  
    <button type="button" class="btn btn-primary">Sony</button>  
  </div>  
</div>

### Grupos de botones anidados y menús desplegables:



Nido de grupos de botones para crear menús desplegables:

<div class="btn-group">  
  <button type="button" class="btn btn-primary">Apple</button>  
  <button type="button" class="btn btn-primary">Samsung</button>  
  <div class="btn-group">  
    <button type="button" class="btn btn-primary dropdown-toggle" data-toggle="dropdown">  
    Sony <span class="caret"></span></button>  
    <ul class="dropdown-menu" role="menu">  
      <li><a href="#">Tablet</a></li>  
      <li><a href="#">Smartphone</a></li>  
    </ul>  
  </div>  
</div>

### Menú desplegable de botones divididos:

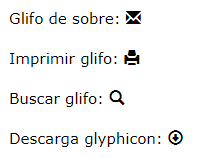


<div class="btn-group">  
  <button type="button" class="btn btn-primary">Sony</button>  
  <button type="button" class="btn btn-primary dropdown-toggle" data-toggle="dropdown">  
    <span class="caret"></span>  
  </button>  
  <ul class="dropdown-menu" role="menu">  
    <li><a href="#">Tablet</a></li>  
    <li><a href="#">Smartphone</a></li>  
  </ul>  
</div>

## Glifos de Bootstrap

Bootstrap proporciona 260 glyphicons del conjunto Glyphicons Halflings.

Los glifos se pueden utilizar en texto, botones, barras de herramientas, navegación, formularios, etc.



Se inserta un glifo con la siguiente sintaxis:

<span class="glyphicon glyphicon-name"></span>

La parte del nombre en la sintaxis anterior debe reemplazarse con el nombre propio del glyphicon.

<p>Envelope icon: <span class="glyphicon glyphicon-envelope"></span></p>  
<p>Envelope icon as a link:  
  <a href="#"><span class="glyphicon glyphicon-envelope"></span></a>  
</p>  
<p>Search icon: <span class="glyphicon glyphicon-search"></span></p>  
<p>Search icon on a button:  
  <button type="button" class="btn btn-default">  
    <span class="glyphicon glyphicon-search"></span> Search  
  </button>  
</p>  
<p>Search icon on a styled button:  
  <button type="button" class="btn btn-info">  
    <span class="glyphicon glyphicon-search"></span> Search  
  </button>  
</p>  
<p>Print icon: <span class="glyphicon glyphicon-print"></span></p>  
<p>Print icon on a styled link button:  
  <a href="#" class="btn btn-success btn-lg">  
    <span class="glyphicon glyphicon-print"></span> Print  
  </a>  
</p>

## Insignias y etiquetas de Bootstrap

### Insignias:

Las insignias son indicadores numéricos de cuántos elementos están asociados con un enlace:



Use la clase .badge dentro de los <span>elementos para crear insignias:

<a href="#">News <span class="badge">5</span></a><br>  
<a href="#">Comments <span class="badge">10</span></a><br>  
<a href="#">Updates <span class="badge">2</span></a>

Las insignias también se pueden usar dentro de otros elementos, como botones:



El siguiente ejemplo muestra cómo agregar insignias a los botones:

<button type="button" class="btn btn-primary">Primary <span class="badge">7</span></button>

### Etiquetas:

Las etiquetas se utilizan para proporcionar información adicional sobre algo:

Utilice la clase .label, seguida por una de las seis clases contextuales .label-default, .label-primary, .label-success, .label-info, .label-warningo .label-danger, dentro de un elemento <span> para crear una etiqueta:

<h1>Example <span class="label label-default">New</span></h1>  
<h2>Example <span class="label label-default">New</span></h2>  
<h3>Example <span class="label label-default">New</span></h3>  
<h4>Example <span class="label label-default">New</span></h4>  
<h5>Example <span class="label label-default">New</span></h5>  
<h6>Example <span class="label label-default">New</span></h6>

## Barras de progreso de Bootstrap

Se puede usar una barra de progreso para mostrarle a un usuario qué tan avanzado está en un proceso.

Bootstrap proporciona varios tipos de barras de progreso.

Una barra de progreso predeterminada en Bootstrap se ve así:



Para crear una barra de progreso predeterminada, agregue una clase .progress a un elemento <div>:

<div class="progress">  
  <div class="progress-bar" role="progressbar" aria-valuenow="70"  
  aria-valuemin="0" aria-valuemax="100" style="width:70%">  
    <span class="sr-only">70% Complete</span>  
  </div>  
</div>

### Barra de progreso con etiqueta:



Elimina la clase .sr-only de la barra de progreso para mostrar un porcentaje visible:

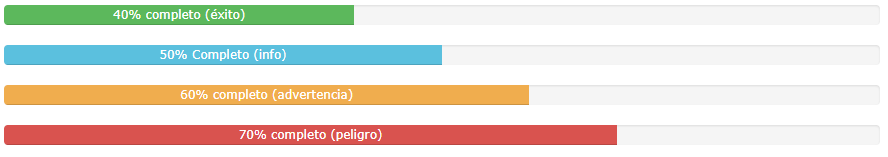
<div class="progress">  
  <div class="progress-bar" role="progressbar" aria-valuenow="70"  
  aria-valuemin="0" aria-valuemax="100" style="width:70%">  
    70%  
  </div>  
</div>

### Barras de progreso de colores:

Las clases contextuales se utilizan para proporcionar "significado a través de los colores".

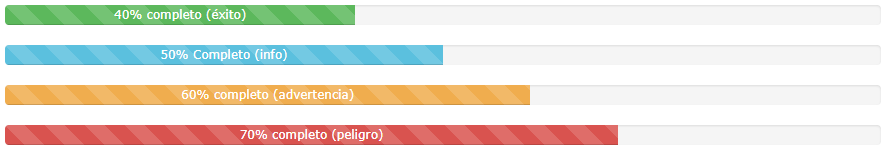
Las clases contextuales que se pueden usar con barras de progreso son:

* .progress-bar-success
* .progress-bar-info
* .progress-bar-warning
* .progress-bar-danger



<div class="progress">  
  <div class="progress-bar progress-bar-success" role="progressbar" aria-valuenow="40"  
  aria-valuemin="0" aria-valuemax="100" style="width:40%">  
    40% Complete (success)  
  </div>  
</div>  
  
<div class="progress">  
  <div class="progress-bar progress-bar-info" role="progressbar" aria-valuenow="50"  
  aria-valuemin="0" aria-valuemax="100" style="width:50%">  
    50% Complete (info)  
  </div>  
</div>  
  
<div class="progress">  
  <div class="progress-bar progress-bar-warning" role="progressbar" aria-valuenow="60"  
  aria-valuemin="0" aria-valuemax="100" style="width:60%">  
    60% Complete (warning)  
  </div>  
</div>  
  
<div class="progress">  
  <div class="progress-bar progress-bar-danger" role="progressbar" aria-valuenow="70"  
  aria-valuemin="0" aria-valuemax="100" style="width:70%">  
    70% Complete (danger)  
  </div>  
</div>

### Barras de progreso a rayas:



<div class="progress">  
  <div class="progress-bar progress-bar-success progress-bar-striped" role="progressbar"  
  aria-valuenow="40" aria-valuemin="0" aria-valuemax="100" style="width:40%">  
    40% Complete (success)  
  </div>  
</div>  
  
<div class="progress">  
  <div class="progress-bar progress-bar-info progress-bar-striped" role="progressbar"  
  aria-valuenow="50" aria-valuemin="0" aria-valuemax="100" style="width:50%">  
    50% Complete (info)  
  </div>  
</div>  
  
<div class="progress">  
  <div class="progress-bar progress-bar-warning progress-bar-striped" role="progressbar"  
  aria-valuenow="60" aria-valuemin="0" aria-valuemax="100" style="width:60%">  
    60% Complete (warning)  
  </div>  
</div>  
  
<div class="progress">  
  <div class="progress-bar progress-bar-danger progress-bar-striped" role="progressbar"  
  aria-valuenow="70" aria-valuemin="0" aria-valuemax="100" style="width:70%">  
    70% Complete (danger)  
  </div>  
</div>

### Barra de progreso animada:

Agrega una clase .active para animar la barra de progreso:

<div class="progress">  
  <div class="progress-bar progress-bar-striped active" role="progressbar"  
  aria-valuenow="40" aria-valuemin="0" aria-valuemax="100" style="width:40%">  
    40%  
  </div>  
</div>

### Barras de progreso apiladas:



Cree una barra de progreso apilada colocando varias barras en la misma <div class="progress">:

<div class="progress">  
  <div class="progress-bar progress-bar-success" role="progressbar" style="width:40%">  
    Free Space  
  </div>  
  <div class="progress-bar progress-bar-warning" role="progressbar" style="width:10%">  
    Warning  
  </div>  
  <div class="progress-bar progress-bar-danger" role="progressbar" style="width:20%">  
    Danger  
  </div>  
</div>

## Paginación Bootstrap

Si tiene un sitio web con muchas páginas, es posible que desee agregar algún tipo de paginación a cada página.

Una paginación básica en Bootstrap se ve así:



Para crear una paginación básica, agregue la clase .pagination a un elemento <ul>:

<ul class="pagination">  
  <li><a href="#">1</a></li>  
  <li><a href="#">2</a></li>  
  <li><a href="#">3</a></li>  
  <li><a href="#">4</a></li>  
  <li><a href="#">5</a></li>  
</ul>

El estado activo muestra cuál es la página actual:



Agregue una clase .active para que el usuario sepa en qué página se encuentra:

<ul class="pagination">  
  <li><a href="#">1</a></li>  
  <li class="active"><a href="#">2</a></li>  
  <li><a href="#">3</a></li>  
  <li><a href="#">4</a></li>  
  <li><a href="#">5</a></li>  
</ul>

### Enlace deshabilitado:



Agregue una clase .disabled si un enlace por alguna razón está deshabilitado:

<ul class="pagination">  
  <li><a href="#">1</a></li>  
  <li><a href="#">2</a></li>  
  <li><a href="#">3</a></li>  
  <li class="disabled"><a href="#">4</a></li>  
  <li><a href="#">5</a></li>  
</ul>

### Tamaño de la paginación:



Agregue clase .pagination-lg para bloques más grandes o .pagination-sm para bloques más pequeños:

<ul class="pagination pagination-lg">  
  <li><a href="#">1</a></li>  
  <li><a href="#">2</a></li>  
  <li><a href="#">3</a></li>  
  <li><a href="#">4</a></li>  
  <li><a href="#">5</a></li>  
</ul>  
  
<ul class="pagination pagination-sm">  
  <li><a href="#">1</a></li>  
  <li><a href="#">2</a></li>  
  <li><a href="#">3</a></li>  
  <li><a href="#">4</a></li>  
  <li><a href="#">5</a></li>  
</ul>

### Migas de pan:



La clase .breadcrumb indica la ubicación de la página actual dentro de una jerarquía de navegación:

<ul class="breadcrumb">  
  <li><a href="#">Home</a></li>  
  <li><a href="#">Private</a></li>  
  <li><a href="#">Pictures</a></li>  
  <li class="active">Vacation</li>  
</ul>

### Pager:

El pager proporciona botones (vínculos) anterior y siguiente.

Para crear botones anterior / siguiente, agregue la clase .pager a un elemento <ul>:

<ul class="pager">  
  <li><a href="#">Previous</a></li>  
  <li><a href="#">Next</a></li>  
</ul>



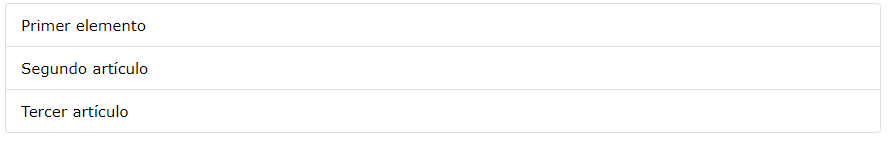
Utilice las clases .previous y .next para alinear cada botón a los lados de la página:

<ul class="pager">  
  <li class="previous"><a href="#">Previous</a></li>  
  <li class="next"><a href="#">Next</a></li>  
</ul>



## Grupos de listas de Bootstrap

El grupo de lista más básico es una lista desordenada con elementos de lista:



Para crear un grupo de lista básico, use un elemento <ul> con clase .list-group y elementos <li> con clase .list-group-item:

<ul class="list-group">  
  <li class="list-group-item">First item</li>  
  <li class="list-group-item">Second item</li>  
  <li class="list-group-item">Third item</li>  
</ul>

### Lista de grupo con insignias:

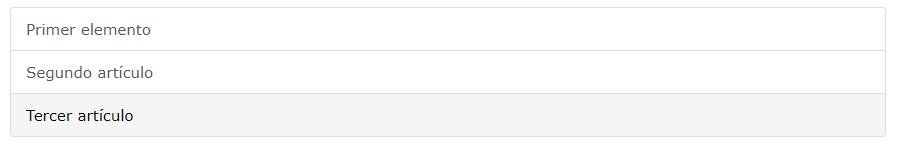


Para crear una insignia, cree un elemento <span>con clase .badge dentro del elemento de la lista:

<ul class="list-group">  
  <li class="list-group-item">New <span class="badge">12</span></li>  
  <li class="list-group-item">Deleted <span class="badge">5</span></li>  
  <li class="list-group-item">Warnings <span class="badge">3</span></li>  
</ul>

### Grupo de lista con elementos vinculados:

Los elementos de un grupo de listas también pueden ser hipervínculos. Esto agregará un color de fondo gris al pasar el mouse:

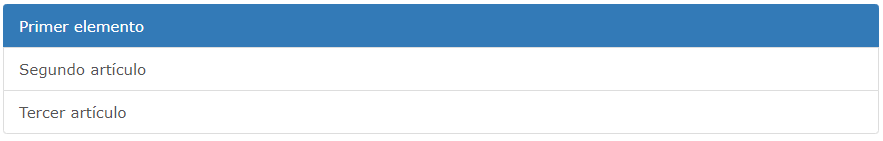


Para crear un grupo de lista con elementos vinculados, utilice en <div> lugar de <ul> y en <a> lugar de <li>:

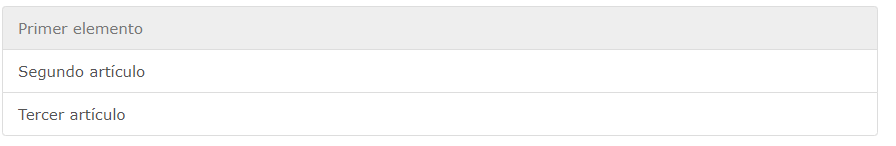
<div class="list-group">  
  <a href="#" class="list-group-item">First item</a>  
  <a href="#" class="list-group-item">Second item</a>  
  <a href="#" class="list-group-item">Third item</a>  
</div>

Utilice la clase .active para resaltar el elemento actual:

<div class="list-group">  
  <a href="#" class="list-group-item active">First item</a>  
  <a href="#" class="list-group-item">Second item</a>  
  <a href="#" class="list-group-item">Third item</a>  
</div>



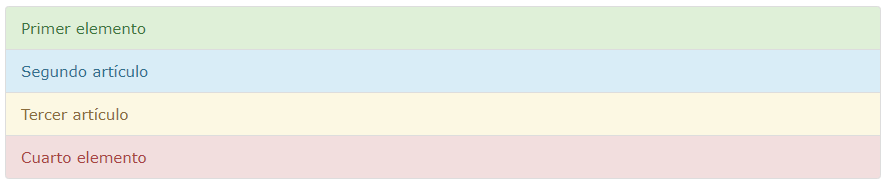
**Artículo inhabilitado**

****

Para deshabilitar un elemento, agregue la clase .disabled:

<div class="list-group">  
  <a href="#" class="list-group-item disabled">First item</a>  
  <a href="#" class="list-group-item">Second item</a>  
  <a href="#" class="list-group-item">Third item</a>  
</div>

### Clases contextuales:



<ul class="list-group">  
  <li class="list-group-item list-group-item-success">First item</li>  
  <li class="list-group-item list-group-item-info">Second item</li>  
  <li class="list-group-item list-group-item-warning">Third item</li>  
  <li class="list-group-item list-group-item-danger">Fourth item</li>  
</ul>

### Contenido personalizado:

Puede agregar casi cualquier HTML dentro de un elemento de grupo de lista.

Bootstrap proporciona las clases .list-group-item-headingy .list-group-item-textque se pueden usar de la siguiente manera:

<div class="list-group">  
  <a href="#" class="list-group-item active">  
    <h4 class="list-group-item-heading">First List Group Item Heading</h4>  
    <p class="list-group-item-text">List Group Item Text</p>  
  </a>  
  <a href="#" class="list-group-item">  
    <h4 class="list-group-item-heading">Second List Group Item Heading</h4>  
    <p class="list-group-item-text">List Group Item Text</p>  
  </a>  
  <a href="#" class="list-group-item">  
    <h4 class="list-group-item-heading">Third List Group Item Heading</h4>  
    <p class="list-group-item-text">List Group Item Text</p>  
  </a>  
</div>

## Paneles Bootstrap

Un panel en bootstrap es un cuadro bordeado con algo de relleno alrededor de su contenido:

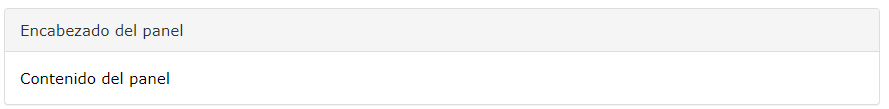


Los paneles se crean con la clase .panel y el contenido dentro del panel tiene una clase .panel-body:

<div class="panel panel-default">  
  <div class="panel-body">A Basic Panel</div>  
</div>

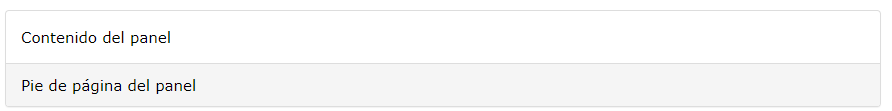
La clase .panel-heading agrega un encabezado al panel:

<div class="panel panel-default">  
  <div class="panel-heading">Panel Heading</div>  
  <div class="panel-body">Panel Content</div>  
</div>



La clase .panel-footer agrega un pie de página al panel:

<div class="panel panel-default">  
  <div class="panel-body">Panel Content</div>  
  <div class="panel-footer">Panel Footer</div>  
</div>

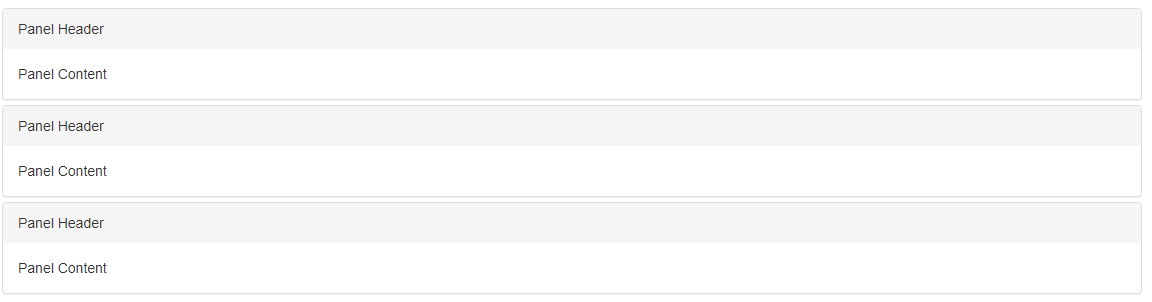


### Grupo de paneles:

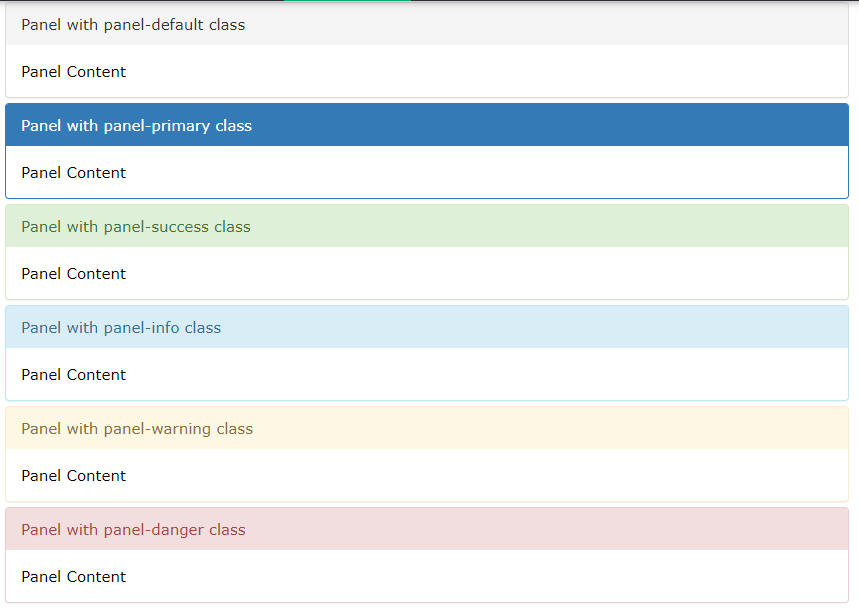
Para agrupar muchos paneles, envuélvalos <div> con una clase .panel-group.

La clase .panel-group borra el margen inferior de cada panel:

<div class="panel-group">  
  <div class="panel panel-default">  
    <div class="panel-body">Panel Content</div>  
  </div>  
  <div class="panel panel-default">  
    <div class="panel-body">Panel Content</div>  
  </div>  
</div>

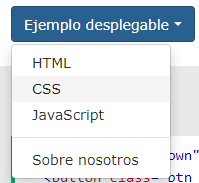


### Paneles con clases contextuales:



## Desplegables Bootstrap:

Un menú desplegable es un menú que se puede alternar que permite al usuario elegir un valor de una lista predefinida:



<div class="dropdown">  
  <button class="btn btn-primary dropdown-toggle" type="button" data-toggle="dropdown">Dropdown Example  
  <span class="caret"></span></button>  
  <ul class="dropdown-menu">  
    <li><a href="#">HTML</a></li>  
    <li><a href="#">CSS</a></li>  
    <li><a href="#">JavaScript</a></li>

<li class="divider"></li>

<li><a href="#">Sobre nosotros </a></li>  
  </ul>  
</div>

La clase .dropdown indica un menú desplegable.

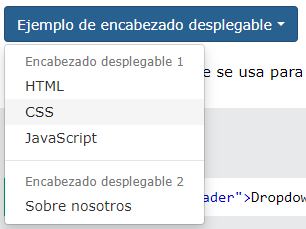
Para abrir el menú desplegable, use un botón o un enlace con una clase de .dropdown-toggle y el atributo data-toggle="dropdown".

La clase .caret crea un icono de flecha de intercalación, lo que indica que el botón es un menú desplegable.

Agregue la clase .dropdown-menu a un elemento <ul> para crear realmente el menú desplegable.

La clase .divider se usa para separar enlaces dentro del menú desplegable con un borde horizontal delgado.

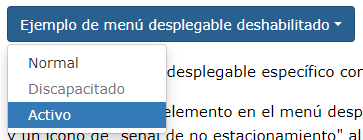
### Encabezado desplegable:



La clase .dropdown-header se usa para agregar encabezados dentro del menú desplegable:

<li class="dropdown-header">Dropdown header 1</li>

### Deshabilitar y elementos activos:



Resalte un elemento desplegable específico con la clase .active (agrega un color de fondo azul).

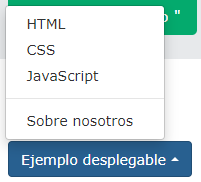
Para deshabilitar un elemento en el menú desplegable, use la clase .disabled (obtiene un color de texto gris claro y un ícono de "señal de no estacionamiento" al pasar el mouse):

<li class="disabled"><a href="#">CSS</a></li>  
<li class="active"><a href="#">HTML</a></li>

Para alinear a la derecha el menú desplegable, agregue la clase .dropdown-menu-right al elemento con .dropdown-menu:

<ul class="dropdown-menu dropdown-menu-right">

### Dropup:



Si desea que el menú desplegable se expanda hacia arriba en lugar de hacia abajo, cambie el elemento <div> con class = "dropdown" a "dropup":

<div class="dropup">

## Plegables de Bootstrap

Los plegables son útiles cuando desea ocultar y mostrar una gran cantidad de contenido

<button data-toggle="collapse" data-target="#demo">Collapsible</button>  
  
<div id="demo" class="collapse">  
Lorem ipsum dolor text....  
</div>

La clase .collapse indica un elemento plegable (un <div> en nuestro ejemplo); este es el contenido que se mostrar u ocultará con un clic de un botón.

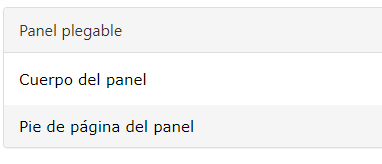
Para controlar (mostrar / ocultar) el contenido plegable, agregue el atributo data-toggle="collapse" a un elemento <a> o <button>. Luego agregue el atributo data-target="#id" para conectar el botón con el contenido plegable (<div id = "demo">).

Nota: Para los elementos <a>, puede utilizar el atributo href en lugar del data-target atributo:

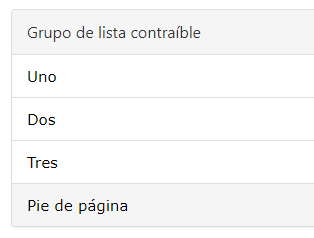
<a href="#demo" data-toggle="collapse">Collapsible</a>  
  
<div id="demo" class="collapse">  
Lorem ipsum dolor text....  
</div>

### Panel plegable:

<div class="panel-group">  
  <div class="panel panel-default">  
    <div class="panel-heading">  
      <h4 class="panel-title">  
        <a data-toggle="collapse" href="#collapse1">Collapsible panel</a>  
      </h4>  
    </div>  
    <div id="collapse1" class="panel-collapse collapse">  
      <div class="panel-body">Panel Body</div>  
      <div class="panel-footer">Panel Footer</div>  
    </div>  
  </div>  
</div>

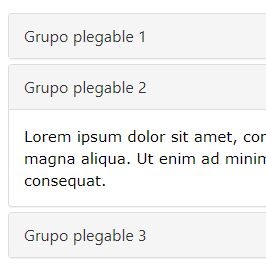
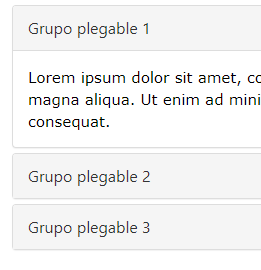


### Grupo de lista contraíble:



<div class="panel-group">  
  <div class="panel panel-default">  
    <div class="panel-heading">  
      <h4 class="panel-title">  
        <a data-toggle="collapse" href="#collapse1">Collapsible list group</a>  
      </h4>  
    </div>  
    <div id="collapse1" class="panel-collapse collapse">  
      <ul class="list-group">  
        <li class="list-group-item">One</li>  
        <li class="list-group-item">Two</li>  
        <li class="list-group-item">Three</li>  
      </ul>  
      <div class="panel-footer">Footer</div>  
    </div>  
  </div>  
</div>

### Acordeón:

Nota: Utilice el data-parentatributo para asegurarse de que todos los elementos contraíbles bajo el elemento principal especificado se cerrarán cuando se muestre uno de los elementos contraíbles.

<div class="panel-group" id="accordion">  
  <div class="panel panel-default">  
    <div class="panel-heading">  
      <h4 class="panel-title">  
        <a data-toggle="collapse" data-parent="#accordion" href="#collapse1">  
        Collapsible Group 1</a>  
      </h4>  
    </div>  
    <div id="collapse1" class="panel-collapse collapse in">  
      <div class="panel-body">Lorem ipsum dolor sit amet, consectetur adipisicing elit,  
      sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad  
      minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea  
      commodo consequat.</div>  
    </div>  
  </div>  
  <div class="panel panel-default">  
    <div class="panel-heading">  
      <h4 class="panel-title">  
        <a data-toggle="collapse" data-parent="#accordion" href="#collapse2">  
        Collapsible Group 2</a>  
      </h4>  
    </div>  
    <div id="collapse2" class="panel-collapse collapse">  
      <div class="panel-body">Lorem ipsum dolor sit amet, consectetur adipisicing elit,  
      sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad  
      minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea  
      commodo consequat.</div>  
    </div>  
  </div>  
  <div class="panel panel-default">  
    <div class="panel-heading">  
      <h4 class="panel-title">  
        <a data-toggle="collapse" data-parent="#accordion" href="#collapse3">  
        Collapsible Group 3</a>  
      </h4>  
    </div>  
    <div id="collapse3" class="panel-collapse collapse">  
      <div class="panel-body">Lorem ipsum dolor sit amet, consectetur adipisicing elit,  
      sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad  
      minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea  
      commodo consequat.</div>  
    </div>  
  </div>  
</div>

## Píldoras y pestañas Bootstrap

### Menús:

La mayoría de las páginas web tienen algún tipo de menú.

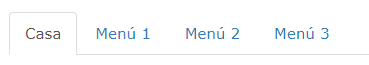
En HTML, un menú a menudo se define en una lista desordenada <ul>(y luego se le aplica un estilo), así:

<ul>  
  <li><a href="#">Home</a></li>  
  <li><a href="#">Menu 1</a></li>  
  <li><a href="#">Menu 2</a></li>  
  <li><a href="#">Menu 3</a></li>  
</ul>

Si desea crear un menú horizontal de la lista anterior, agregue la clase .list-inline a <ul>:

<ul class="list-inline">

### Pestañas:



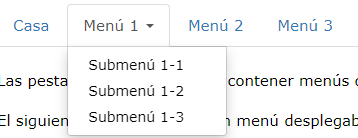
Las pestañas se crean con <ul class="nav nav-tabs">:

Sugerencia: marque también la página actual con <li class="active">.

El siguiente ejemplo crea pestañas de navegación:

<ul class="nav nav-tabs">  
  <li class="active"><a href="#">Home</a></li>  
  <li><a href="#">Menu 1</a></li>  
  <li><a href="#">Menu 2</a></li>  
  <li><a href="#">Menu 3</a></li>  
</ul>

### Pestañas con menú desplegable:



<ul class="nav nav-tabs">  
  <li class="active"><a href="#">Home</a></li>  
  <li class="dropdown">  
    <a class="dropdown-toggle" data-toggle="dropdown" href="#">Menu 1  
    <span class="caret"></span></a>  
    <ul class="dropdown-menu">  
      <li><a href="#">Submenu 1-1</a></li>  
      <li><a href="#">Submenu 1-2</a></li>  
      <li><a href="#">Submenu 1-3</a></li>  
    </ul>  
  </li>  
  <li><a href="#">Menu 2</a></li>  
  <li><a href="#">Menu 3</a></li>  
</ul>

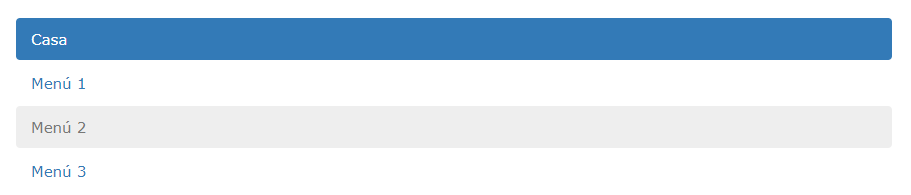
### Pastillas:



Las pastillas se crean con <ul class="nav nav-pills">. También marque la página actual con <li class="active">:

<ul class="nav nav-pills">  
  <li class="active"><a href="#">Home</a></li>  
  <li><a href="#">Menu 1</a></li>  
  <li><a href="#">Menu 2</a></li>  
  <li><a href="#">Menu 3</a></li>  
</ul>

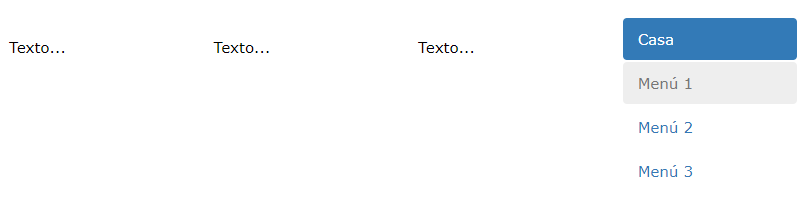
### Pastillas verticals:



Las píldoras también se pueden mostrar verticalmente. Solo agrega la .nav-stackedclase:

<ul class="nav nav-pills nav-stacked">  
  <li class="active"><a href="#">Home</a></li>  
  <li><a href="#">Menu 1</a></li>  
  <li><a href="#">Menu 2</a></li>  
  <li><a href="#">Menu 3</a></li>  
</ul>

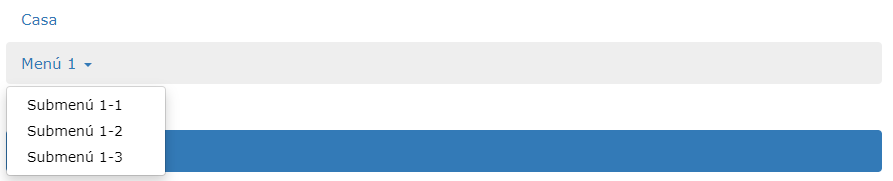
### Pastillas verticales seguidas:



El siguiente ejemplo coloca el menú de píldoras vertical dentro de la última columna. Entonces, en una pantalla grande, el menú se mostrará a la derecha. Pero en una pantalla pequeña, el contenido se ajustará automáticamente a un diseño de una sola columna:

<div class="col-md-3">  
  <ul class="nav nav-pills nav-stacked">  
    <li class="active"><a href="#">Home</a></li>  
    <li><a href="#">Menu 1</a></li>  
    <li><a href="#">Menu 2</a></li>  
    <li><a href="#">Menu 3</a></li>  
  </ul>  
</div>

### Pastillas con menú desplegable:



<ul class="nav nav-pills nav-stacked">  
  <li class="active"><a href="#">Home</a></li>  
  <li class="dropdown">  
    <a class="dropdown-toggle" data-toggle="dropdown" href="#">Menu 1  
    <span class="caret"></span></a>  
    <ul class="dropdown-menu">  
      <li><a href="#">Submenu 1-1</a></li>  
      <li><a href="#">Submenu 1-2</a></li>  
      <li><a href="#">Submenu 1-3</a></li>  
    </ul>  
  </li>  
  <li><a href="#">Menu 2</a></li>  
  <li><a href="#">Menu 3</a></li>  
</ul>

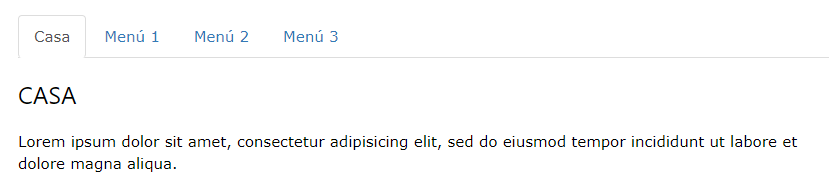
### Centradas:

Para centrar / justificar las pestañas y las píldoras, use la clase .nav-justified.

Tenga en cuenta que en las pantallas de menos de 768 px, los elementos de la lista se apilan (el contenido permanecerá centrado):

<!-- Centered Tabs -->  
<ul class="nav nav-tabs nav-justified">  
  <li class="active"><a href="#">Home</a></li>  
  <li><a href="#">Menu 1</a></li>  
  <li><a href="#">Menu 2</a></li>  
  <li><a href="#">Menu 3</a></li>  
</ul>  
  
<!-- Centered Pills -->  
<ul class="nav nav-pills nav-justified">  
  <li class="active"><a href="#">Home</a></li>  
  <li><a href="#">Menu 1</a></li>  
  <li><a href="#">Menu 2</a></li>  
  <li><a href="#">Menu 3</a></li>  
</ul>

### Pestañas dinámicas / alternables:



Para que las pestañas se puedan alternar, agregue el atributo data-toggle="tab" a cada enlace. Luego agregue una clase .tab-pane con una ID única para cada pestaña y envuélvalas dentro de un elemento <div> con class .tab-content.

Si desea que las pestañas aparezcan y desaparezcan gradualmente al hacer clic en ellas, agregue la clase .fade a .tab-pane:

<ul class="nav nav-tabs">  
  <li class="active"><a data-toggle="tab" href="#home">Home</a></li>  
  <li><a data-toggle="tab" href="#menu1">Menu 1</a></li>  
  <li><a data-toggle="tab" href="#menu2">Menu 2</a></li>  
</ul>  
  
<div class="tab-content">  
  <div id="home" class="tab-pane fade in active">  
    <h3>HOME</h3>  
    <p>Some content.</p>  
  </div>  
  <div id="menu1" class="tab-pane fade">  
    <h3>Menu 1</h3>  
    <p>Some content in menu 1.</p>  
  </div>  
  <div id="menu2" class="tab-pane fade">  
    <h3>Menu 2</h3>  
    <p>Some content in menu 2.</p>  
  </div>  
</div>

### Pastillas alternas / dinámicas:

El mismo código se aplica a las píldoras; solo cambie el atributo de alternancia de datos a data-toggle="pill":

<ul class="nav nav-pills">  
  <li class="active"><a data-toggle="pill" href="#home">Home</a></li>  
  <li><a data-toggle="pill" href="#menu1">Menu 1</a></li>  
  <li><a data-toggle="pill" href="#menu2">Menu 2</a></li>  
</ul>  
  
<div class="tab-content">  
  <div id="home" class="tab-pane fade in active">  
    <h3>HOME</h3>  
    <p>Some content.</p>  
  </div>  
  <div id="menu1" class="tab-pane fade">  
    <h3>Menu 1</h3>  
    <p>Some content in menu 1.</p>  
  </div>  
  <div id="menu2" class="tab-pane fade">  
    <h3>Menu 2</h3>  
    <p>Some content in menu 2.</p>  
  </div>  
</div>

## Barra de navegación Bootstrap

Una barra de navegación es un encabezado de navegación que se coloca en la parte superior de la página:



Con Bootstrap, una barra de navegación puede extenderse o contraerse, según el tamaño de la pantalla.

Se crea una barra de navegación estándar con <nav class="navbar navbar-default">.

El siguiente ejemplo muestra cómo agregar una barra de navegación en la parte superior de la página:

<nav class="navbar navbar-default">  
  <div class="container-fluid">  
    <div class="navbar-header">  
      <a class="navbar-brand" href="#">WebSiteName</a>  
    </div>  
    <ul class="nav navbar-nav">  
      <li class="active"><a href="#">Home</a></li>  
      <li><a href="#">Page 1</a></li>  
      <li><a href="#">Page 2</a></li>  
      <li><a href="#">Page 3</a></li>  
    </ul>  
  </div>  
</nav>

### Barra de navegación invertida:

Si no le gusta el estilo de la barra de navegación predeterminada, Bootstrap proporciona una barra de navegación negra alternativa:



Simplemente cambie la clase .navbar-default a .navbar-inverse:

<nav class="navbar navbar-inverse">  
  <div class="container-fluid">  
    <div class="navbar-header">  
      <a class="navbar-brand" href="#">WebSiteName</a>  
    </div>  
    <ul class="nav navbar-nav">  
      <li class="active"><a href="#">Home</a></li>  
      <li><a href="#">Page 1</a></li>  
      <li><a href="#">Page 2</a></li>  
      <li><a href="#">Page 3</a></li>  
    </ul>  
  </div>  
</nav>

### Barra de navegación con menú desplegable:



<nav class="navbar navbar-inverse">  
  <div class="container-fluid">  
    <div class="navbar-header">  
      <a class="navbar-brand" href="#">WebSiteName</a>  
    </div>  
    <ul class="nav navbar-nav">  
      <li class="active"><a href="#">Home</a></li>  
      <li class="dropdown">  
        <a class="dropdown-toggle" data-toggle="dropdown" href="#">Page 1  
        <span class="caret"></span></a>  
        <ul class="dropdown-menu">  
          <li><a href="#">Page 1-1</a></li>  
          <li><a href="#">Page 1-2</a></li>  
          <li><a href="#">Page 1-3</a></li>  
        </ul>  
      </li>  
      <li><a href="#">Page 2</a></li>  
      <li><a href="#">Page 3</a></li>  
    </ul>  
  </div>  
</nav>

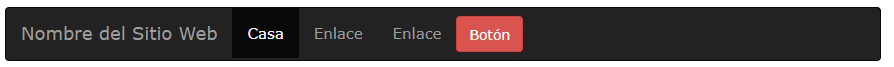
### Barra de navegación alineada a la derecha:



La clase .navbar-right se usa para alinear a la derecha los botones de la barra de navegación.

<nav class="navbar navbar-inverse">  
  <div class="container-fluid">  
    <div class="navbar-header">  
      <a class="navbar-brand" href="#">WebSiteName</a>  
    </div>  
    <ul class="nav navbar-nav">  
      <li class="active"><a href="#">Home</a></li>  
      <li><a href="#">Page 1</a></li>  
      <li><a href="#">Page 2</a></li>  
    </ul>  
    <ul class="nav navbar-nav navbar-right">  
      <li><a href="#"><span class="glyphicon glyphicon-user"></span> Sign Up</a></li>  
      <li><a href="#"><span class="glyphicon glyphicon-log-in"></span> Login</a></li>  
    </ul>  
  </div>  
</nav>

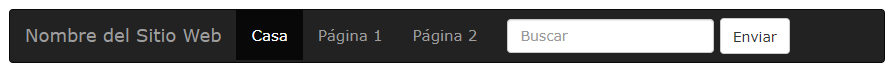
### Botones de la barra de navegación:



Para agregar botones dentro de la barra de navegación, agregue la clase .navbar-btn en un botón de Bootstrap:

<nav class="navbar navbar-inverse">  
  <div class="container-fluid">  
    <div class="navbar-header">  
      <a class="navbar-brand" href="#">WebSiteName</a>  
    </div>  
    <ul class="nav navbar-nav">  
      <li class="active"><a href="#">Home</a></li>  
      <li><a href="#">Link</a></li>  
      <li><a href="#">Link</a></li>  
    </ul>  
    <button class="btn btn-danger navbar-btn">Button</button>  
  </div>  
</nav>

### Formularios de barra de navegación:

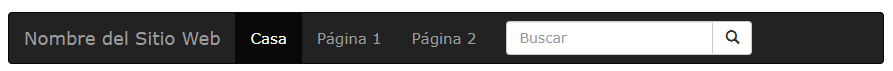


Para agregar elementos de formulario dentro de la barra de navegación, agregue la clase .navbar-form a un elemento de formulario y agregue una entrada (s). Tenga en cuenta que hemos agregado una clase .form-group al contenedor div que contiene la entrada. Esto agrega un relleno adecuado si tiene más de una entrada (aprenderá más sobre esto en el capítulo de Formularios).

<nav class="navbar navbar-inverse">  
  <div class="container-fluid">  
    <div class="navbar-header">  
      <a class="navbar-brand" href="#">WebSiteName</a>  
    </div>  
    <ul class="nav navbar-nav">  
      <li class="active"><a href="#">Home</a></li>  
      <li><a href="#">Page 1</a></li>  
      <li><a href="#">Page 2</a></li>  
    </ul>  
    <form class="navbar-form navbar-left" action="/action\_page.php">  
      <div class="form-group">  
        <input type="text" class="form-control" placeholder="Search">  
      </div>  
      <button type="submit" class="btn btn-default">Submit</button>  
    </form>  
  </div>  
</nav>

También puede utilizar las clases .input-groupy .input-group-addon para adjuntar un icono o texto de ayuda junto al campo de entrada. Aprenderá más sobre estas clases en el capítulo Entradas de Bootstrap.

<form class="navbar-form navbar-left" action="/action\_page.php">  
  <div class="input-group">  
    <input type="text" class="form-control" placeholder="Search">  
    <div class="input-group-btn">  
      <button class="btn btn-default" type="submit">  
        <i class="glyphicon glyphicon-search"></i>  
      </button>  
    </div>  
  </div>  
</form>



### Texto de la barra de navegación:



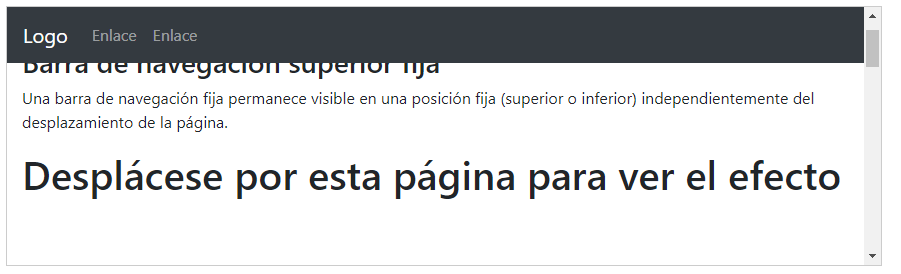
Utilice la clase .navbar-text para alinear verticalmente cualquier elemento dentro de la barra de navegación que no sean enlaces (asegura el relleno y el color del texto adecuados).

<nav class="navbar navbar-inverse">  
  <ul class="nav navbar-nav">  
    <li><a href="#">Link</a></li>  
    <li><a href="#">Link</a></li>  
  </ul>  
  <p class="navbar-text">Some text</p>  
</nav>

### Barra de navegación fija:

La barra de navegación también se puede fijar en la parte superior o inferior de la página.

Una barra de navegación fija permanece visible en una posición fija (superior o inferior) independientemente del desplazamiento de la página.



La clase .navbar-fixed-top hace que la barra de navegación esté fija en la parte superior:

<nav class="navbar navbar-inverse navbar-fixed-top">  
  <div class="container-fluid">  
    <div class="navbar-header">  
      <a class="navbar-brand" href="#">WebSiteName</a>  
    </div>  
    <ul class="nav navbar-nav">  
      <li class="active"><a href="#">Home</a></li>  
      <li><a href="#">Page 1</a></li>  
      <li><a href="#">Page 2</a></li>  
      <li><a href="#">Page 3</a></li>  
    </ul>  
  </div>  
</nav>

La clase .navbar-fixed-bottom hace que la barra de navegación permanezca en la parte inferior:

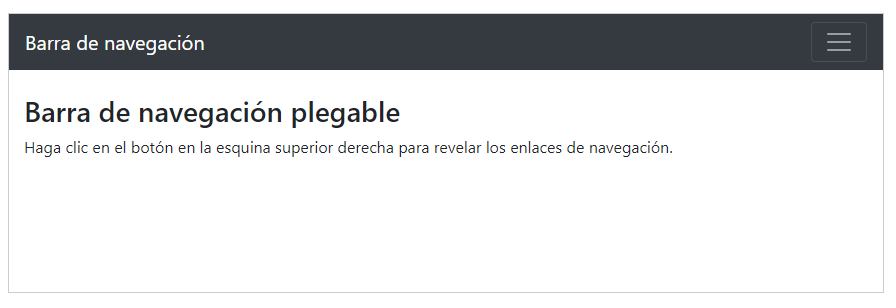
<nav class="navbar navbar-inverse navbar-fixed-bottom">  
  <div class="container-fluid">  
    <div class="navbar-header">  
      <a class="navbar-brand" href="#">WebSiteName</a>  
    </div>  
    <ul class="nav navbar-nav">  
      <li class="active"><a href="#">Home</a></li>  
      <li><a href="#">Page 1</a></li>  
      <li><a href="#">Page 2</a></li>  
      <li><a href="#">Page 3</a></li>  
    </ul>  
  </div>  
</nav>

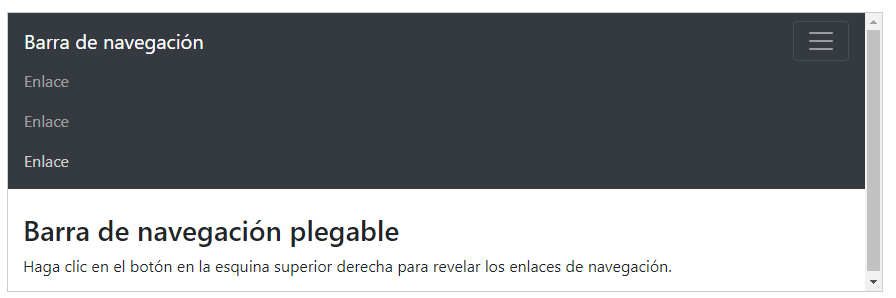
### Colapsar la barra de navegación:

La barra de navegación a menudo ocupa demasiado espacio en una pantalla pequeña.

Deberíamos ocultar la barra de navegación; y solo mostrarlo cuando sea necesario.

En el siguiente ejemplo, la barra de navegación se reemplaza por un botón en la esquina superior derecha. Solo cuando se hace clic en el botón, se mostrará la barra de navegación:





<nav class="navbar navbar-inverse">  
  <div class="container-fluid">  
    <div class="navbar-header">  
      <button type="button" class="navbar-toggle" data-toggle="collapse" data-target="#myNavbar">  
        <span class="icon-bar"></span>  
        <span class="icon-bar"></span>  
        <span class="icon-bar"></span>  
      </button>  
      <a class="navbar-brand" href="#">WebSiteName</a>  
    </div>  
    <div class="collapse navbar-collapse" id="myNavbar">  
      <ul class="nav navbar-nav">  
        <li class="active"><a href="#">Home</a></li>  
        <li><a href="#">Page 1</a></li>  
        <li><a href="#">Page 2</a></li>  
        <li><a href="#">Page 3</a></li>  
      </ul>  
      <ul class="nav navbar-nav navbar-right">  
        <li><a href="#"><span class="glyphicon glyphicon-user"></span> Sign Up</a></li>  
        <li><a href="#"><span class="glyphicon glyphicon-log-in"></span> Login</a></li>  
      </ul>  
    </div>  
  </div>  
</nav>

## Formularios de Bootstrap

Los controles de formulario reciben automáticamente un estilo global con Bootstrap:

Todo elemento textual <input>, <textarea>y <select> con clase .form-control tiene una anchura de 100%

Bootstrap proporciona tres tipos de diseños de formularios:

Forma vertical (esto es predeterminado)

Forma horizontal

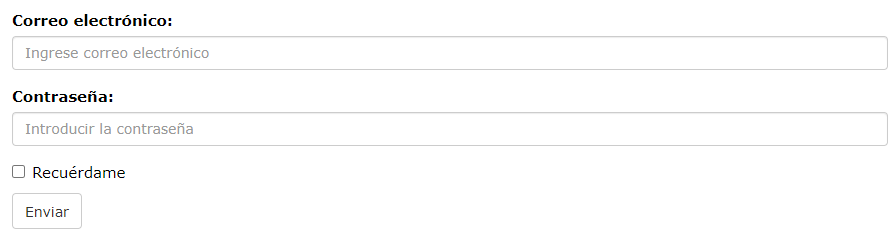
Formulario en línea

Reglas estándar para los tres diseños de formularios:

Envuelva las etiquetas y los controles de formulario <div class="form-group">(necesarios para un espaciado óptimo)

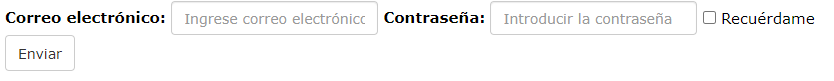
Agregar la clase .form-controla todo elemento textual <input>, <textarea>y <select>

### Forma vertical de Bootstrap (predeterminado):



<form action="/action\_page.php">  
  <div class="form-group">  
    <label for="email">Email address:</label>  
    <input type="email" class="form-control" id="email">  
  </div>  
  <div class="form-group">  
    <label for="pwd">Password:</label>  
    <input type="password" class="form-control" id="pwd">  
  </div>  
  <div class="checkbox">  
    <label><input type="checkbox"> Remember me</label>  
  </div>  
  <button type="submit" class="btn btn-default">Submit</button>  
</form>

### Formulario en línea de Bootstrap:



En un formulario en línea, todos los elementos están alineados, alineados a la izquierda y las etiquetas están al lado.

Nota: ¡Esto solo se aplica a los formularios dentro de las ventanas gráficas que tienen al menos 768 px de ancho!

Regla adicional para un formulario en línea:

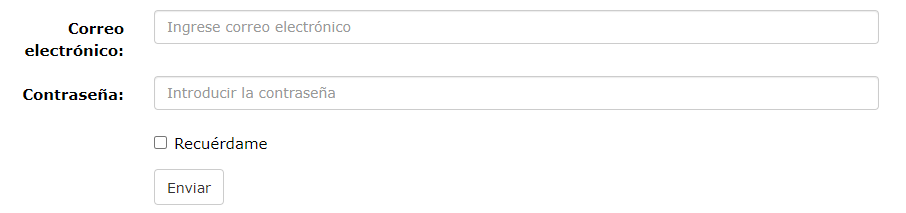
Agregar clase .form-inline al elemento <form>

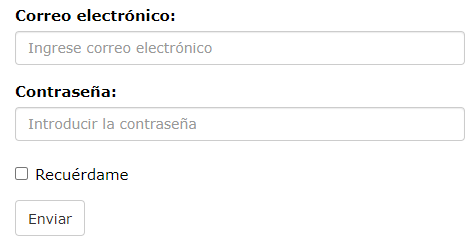
<form class="form-inline" action="/action\_page.php">  
  <div class="form-group">  
    <label for="email">Email address:</label>  
    <input type="email" class="form-control" id="email">  
  </div>  
  <div class="form-group">  
    <label for="pwd">Password:</label>  
    <input type="password" class="form-control" id="pwd">  
  </div>  
  <div class="checkbox">  
    <label><input type="checkbox"> Remember me</label>  
  </div>  
  <button type="submit" class="btn btn-default">Submit</button>  
</form>

Consejo: si no incluye una etiqueta para cada entrada, los lectores de pantalla tendrán problemas con sus formularios. Puede ocultar las etiquetas de todos los dispositivos, excepto los lectores de pantalla, mediante la clase .sr-only.

<form class="form-inline" action="/action\_page.php">  
  <div class="form-group">  
    <label class="sr-only" for="email">Email address:</label>  
    <input type="email" class="form-control" id="email">  
  </div>  
  <div class="form-group">  
    <label class="sr-only" for="pwd">Password:</label>  
    <input type="password" class="form-control" id="pwd">  
  </div>  
  <div class="checkbox">  
    <label><input type="checkbox"> Remember me</label>  
  </div>  
  <button type="submit" class="btn btn-default">Submit</button>  
</form>

### Forma horizontal Bootstrap:





Una forma horizontal significa que las etiquetas están alineadas junto al campo de entrada (horizontal) en pantallas grandes y medianas. En pantallas pequeñas (767px e inferiores), se transformará en una forma vertical (las etiquetas se colocan en la parte superior de cada entrada).

Reglas adicionales para una forma horizontal:

* Agregar clase .form-horizontal al <form>elemento
* Agregar clase .control-labela todos los elementos <label>

Consejo: use las clases de cuadrícula predefinidas de Bootstrap para alinear etiquetas y grupos de controles de formulario en un diseño horizontal.

<form class="form-horizontal" action="/action\_page.php">  
  <div class="form-group">  
    <label class="control-label col-sm-2" for="email">Email:</label>  
    <div class="col-sm-10">  
      <input type="email" class="form-control" id="email" placeholder="Enter email">  
    </div>  
  </div>  
  <div class="form-group">  
    <label class="control-label col-sm-2" for="pwd">Password:</label>  
    <div class="col-sm-10">  
      <input type="password" class="form-control" id="pwd" placeholder="Enter password">  
    </div>  
  </div>  
  <div class="form-group">  
    <div class="col-sm-offset-2 col-sm-10">  
      <div class="checkbox">  
        <label><input type="checkbox"> Remember me</label>  
      </div>  
    </div>  
  </div>  
  <div class="form-group">  
    <div class="col-sm-offset-2 col-sm-10">  
      <button type="submit" class="btn btn-default">Submit</button>  
    </div>  
  </div>  
</form>

## Entradas de formulario Bootstrap

Bootstrap admite los siguientes controles de formulario:

* input
* textarea
* checkbox
* radio
* select

### Bootstrap Input:

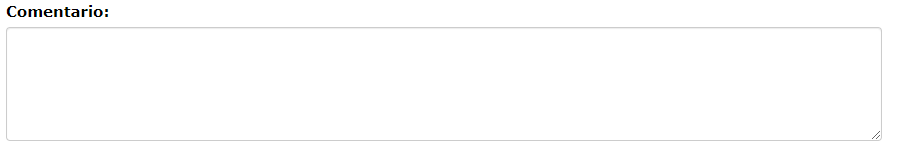
Bootstrap admite todos los tipos de entrada HTML5: texto, contraseña, fecha y hora, fecha y hora local, fecha, mes, hora, semana, número, correo electrónico, url, búsqueda, teléfono y color.

Nota: ¡Las entradas NO tendrán un estilo completo si su tipo no se declara correctamente!



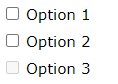
<div class="form-group">  
  <label for="usr">Name:</label>  
  <input type="text" class="form-control" id="usr">  
</div>  
<div class="form-group">  
  <label for="pwd">Password:</label>  
  <input type="password" class="form-control" id="pwd">  
</div>

### Bootstrap Textarea:



<div class="form-group">  
  <label for="comment">Comment:</label>  
  <textarea class="form-control" rows="5" id="comment"></textarea>  
</div>

### Bootstrap Checkboxes:



<div class="checkbox">  
  <label><input type="checkbox" value="">Option 1</label>  
</div>  
<div class="checkbox">  
  <label><input type="checkbox" value="">Option 2</label>  
</div>  
<div class="checkbox disabled">  
  <label><input type="checkbox" value="" disabled>Option 3</label>  
</div>

Utilice la clase .checkbox-inline si desea que las casillas de verificación aparezcan en la misma línea:



<label class="checkbox-inline"><input type="checkbox" value="">Option 1</label>  
<label class="checkbox-inline"><input type="checkbox" value="">Option 2</label>  
<label class="checkbox-inline"><input type="checkbox" value="">Option 3</label>

### Bootstrap Radio Buttons:

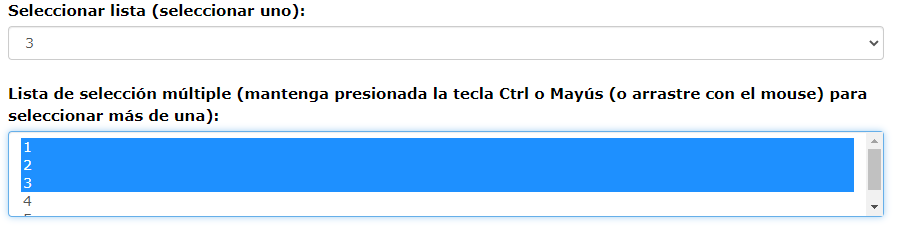
<div class="radio">  
  <label><input type="radio" name="optradio" checked>Option 1</label>  
</div>  
<div class="radio">  
  <label><input type="radio" name="optradio">Option 2</label>  
</div>  
<div class="radio disabled">  
  <label><input type="radio" name="optradio" disabled>Option 3</label>  
</div>

Utilice la clase .radio-inline si desea que los botones de opción aparezcan en la misma línea:



<label class="radio-inline"><input type="radio" name="optradio" checked>Option 1</label>  
<label class="radio-inline"><input type="radio" name="optradio">Option 2</label>  
<label class="radio-inline"><input type="radio" name="optradio">Option 3</label>

### Bootstrap Select List:

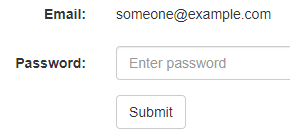


<div class="form-group">  
  <label for="sel1">Select list:</label>  
  <select class="form-control" id="sel1">  
    <option>1</option>  
    <option>2</option>  
    <option>3</option>  
    <option>4</option>  
  </select>  
</div>

### Control estático:

Si necesita insertar texto sin formato junto a una etiqueta de formulario dentro de un formulario horizontal, use la clase .form-control-static en un elemento <p>:

<form class="form-horizontal">  
  <div class="form-group">  
    <label class="control-label col-sm-2">Email:</label>  
    <div class="col-sm-10">  
      <p class="form-control-static">someone@example.com</p>  
    </div>  
  </div>  
</form>



### Grupos de entrada de Bootstrap:

La clase .input-group es un contenedor para mejorar una entrada agregando un icono, texto o un botón delante o detrás como "texto de ayuda".

La clase .input-group-addon adjunta un icono o texto de ayuda junto al campo de entrada.



<form>  
  <div class="input-group">  
    <span class="input-group-addon"><i class="glyphicon glyphicon-user"></i></span>  
    <input id="email" type="text" class="form-control" name="email" placeholder="Email">  
  </div>  
  <div class="input-group">  
    <span class="input-group-addon"><i class="glyphicon glyphicon-lock"></i></span>  
    <input id="password" type="password" class="form-control" name="password" placeholder="Password">  
  </div>  
  <div class="input-group">  
    <span class="input-group-addon">Text</span>  
    <input id="msg" type="text" class="form-control" name="msg" placeholder="Additional Info">  
  </div>  
</form>

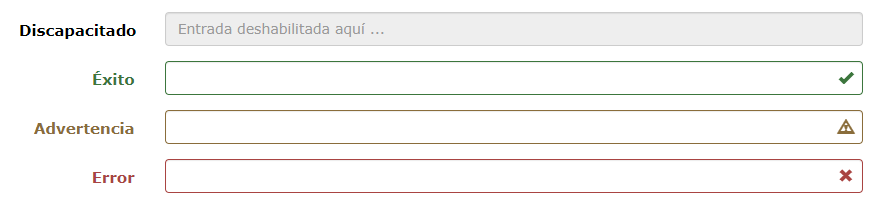


El .input-group-btn adjunta un botón al lado de una entrada. Esto se usa a menudo junto con una barra de búsqueda:



<form>  
  <div class="input-group">  
    <input type="text" class="form-control" placeholder="Search">  
    <div class="input-group-btn">  
      <button class="btn btn-default" type="submit">  
        <i class="glyphicon glyphicon-search"></i>  
      </button>  
    </div>  
  </div>  
</form>

### Estados de control de formulario de Bootstrap:



* ENFOQUE DE ENTRADA : se elimina el contorno de la entrada y se aplica una sombra de cuadro en el enfoque
* ENTRADAS DESHABILITADAS : agregue un atributo disabled para deshabilitar un campo de entrada
* CONJUNTOS DE CAMPOS DESHABILITADOS : agregue un atributo disabled a un conjunto de campos para deshabilitar todos los controles dentro
* ENTRADAS DE LECTURA : agregue un atributo readonly a una entrada para evitar la entrada del usuario
* ESTADOS DE VALIDACIÓN: Bootstrap incluye estilos de validación para mensajes de error, advertencia y éxito. Para utilizar, añadir .has-warning, .has-erroro .has-successal elemento padre
* ICONOS : puedes agregar íconos de comentarios con la clase .has-feedback y un ícono
* ETIQUETAS OCULTAS : agregue una clase .sr-only en etiquetas no visibles

<form class="form-horizontal">  
  <div class="form-group">  
    <label class="col-sm-2 control-label">Focused</label>  
    <div class="col-sm-10">  
      <input class="form-control" id="focusedInput" type="text" value="Click to focus">  
    </div>  
  </div>  
  <div class="form-group">  
    <label for="disabledInput" class="col-sm-2 control-label">Disabled</label>  
    <div class="col-sm-10">  
      <input class="form-control" id="disabledInput" type="text" disabled>  
    </div>  
  </div>  
  <fieldset disabled>  
    <div class="form-group">  
      <label for="disabledTextInput" class="col-sm-2 control-label">Fieldset disabled</label>  
      <div class="col-sm-10">  
        <input type="text" id="disabledTextInput" class="form-control">  
      </div>  
    </div>  
    <div class="form-group">  
      <label for="disabledSelect" class="col-sm-2 control-label"></label>  
      <div class="col-sm-10">  
        <select id="disabledSelect" class="form-control">  
          <option>Disabled select</option>  
        </select>  
      </div>  
    </div>  
  </fieldset>  
  <div class="form-group has-success has-feedback">  
    <label class="col-sm-2 control-label" for="inputSuccess">  
    Input with success and icon</label>  
    <div class="col-sm-10">  
      <input type="text" class="form-control" id="inputSuccess">  
      <span class="glyphicon glyphicon-ok form-control-feedback"></span>  
    </div>  
  </div>  
  <div class="form-group has-warning has-feedback">  
    <label class="col-sm-2 control-label" for="inputWarning">  
    Input with warning and icon</label>  
    <div class="col-sm-10">  
      <input type="text" class="form-control" id="inputWarning">  
      <span class="glyphicon glyphicon-warning-sign form-control-feedback"></span>  
    </div>  
  </div>  
  <div class="form-group has-error has-feedback">  
    <label class="col-sm-2 control-label" for="inputError">  
    Input with error and icon</label>  
    <div class="col-sm-10">  
      <input type="text" class="form-control" id="inputError">  
      <span class="glyphicon glyphicon-remove form-control-feedback"></span>  
    </div>  
  </div>  
</form>

## Tamaño de inputs de Bootstrap

### Tamaño de inputs en formularios:

Establezca las alturas de los elementos de entrada utilizando clases como .input-lgy .input-sm.

Establezca los anchos de los elementos utilizando clases de columnas de cuadrícula como .col-lg-\*y .col-sm-\*.

### Tamaño de altura:



<form>  
  <div class="form-group">  
    <label for="inputsm">Small input</label>  
    <input class="form-control input-sm" id="inputsm" type="text">  
  </div>  
   <div class="form-group">  
    <label for="inputdefault">Default input</label>  
    <input class="form-control" id="inputdefault" type="text">  
  </div>  
  <div class="form-group">  
    <label for="inputlg">Large input</label>  
    <input class="form-control input-lg" id="inputlg" type="text">  
  </div>  
</form>

Puede cambiar rápidamente el tamaño de las etiquetas y los controles de formulario dentro de un formulario horizontal agregando .form-group-\*al elemento <div class="form-group">:

<div class="form-group form-group-lg">

También puede dimensionar rápidamente todas las entradas y otros elementos dentro de una .input-group con las clases .input-group-smo .input-group-lg:

<div class="input-group input-group-lg">

### Tamaño de columna:



<div class="form-group row">  
  <div class="col-xs-2">  
    <label for="ex1">col-xs-2</label>  
    <input class="form-control" id="ex1" type="text">  
  </div>  
  <div class="col-xs-3">  
    <label for="ex2">col-xs-3</label>  
    <input class="form-control" id="ex2" type="text">  
  </div>  
  <div class="col-xs-4">  
    <label for="ex3">col-xs-4</label>  
    <input class="form-control" id="ex3" type="text">  
  </div>  
</div>

### texto de ayuda:

Use la clase .help-block para agregar un texto de ayuda a nivel de bloque en formularios:

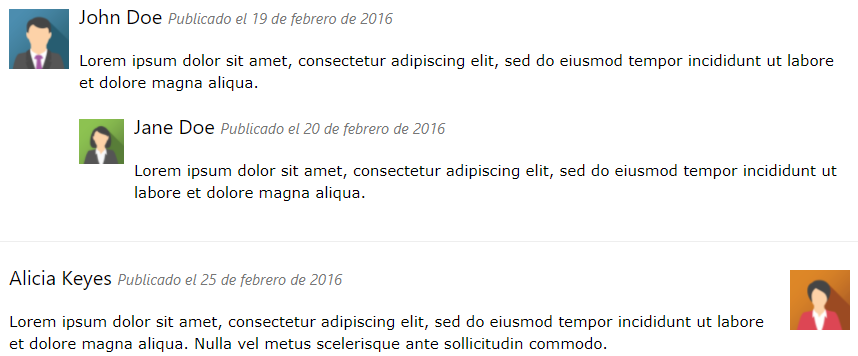
<div class="form-group">  
  <label for="pwd">Password:</label>  
  <input type="password" class="form-control" id="pwd" placeholder="Enter password">  
  <span class="help-block">This is some help text...</span>  
</div>



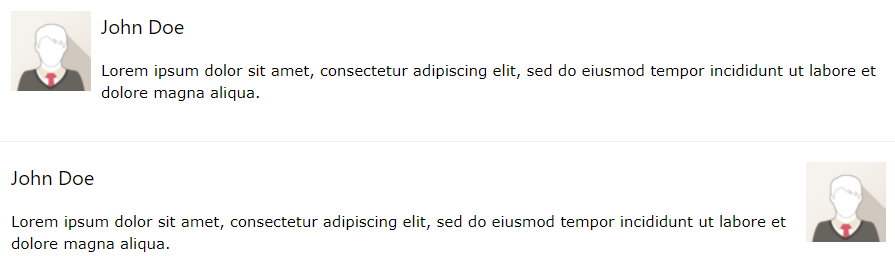
## Bootstrap Media Objects

### Objetos multimedia:

Bootstrap proporciona una manera fácil de alinear objetos multimedia (como imágenes o videos) a la izquierda o a la derecha de algún contenido. Esto se puede utilizar para mostrar comentarios de blogs, tweets, etc.



### Objeto multimedia básico:



<!-- Left-aligned -->  
<div class="media">  
  <div class="media-left">  
    <img src="img\_avatar1.png" class="media-object" style="width:60px">  
  </div>  
  <div class="media-body">  
    <h4 class="media-heading">John Doe</h4>  
    <p>Lorem ipsum...</p>  
  </div>  
</div>  
  
<!-- Right-aligned -->  
<div class="media">  
  <div class="media-body">  
    <h4 class="media-heading">John Doe</h4>  
    <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.</p>  
  </div>  
  <div class="media-right">  
    <img src="img\_avatar1.png" class="media-object" style="width:60px">  
  </div>  
</div>

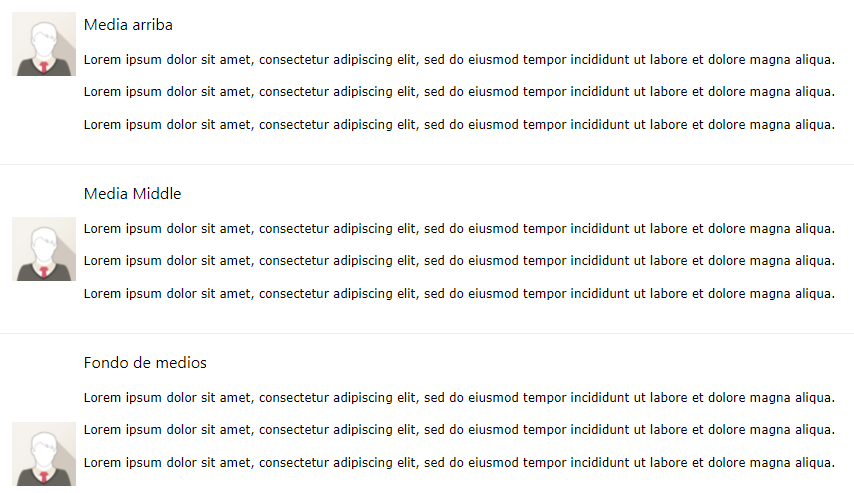
Utilice un elemento <div> con la clase .media para crear un contenedor para objetos multimedia.

Utilice la clase .media-left para alinear el objeto multimedia (imagen) a la izquierda, o la clase .media-right para alinearlo a la derecha.

El texto que debería aparecer junto a la imagen, se coloca dentro de un contenedor con class = " media-body".

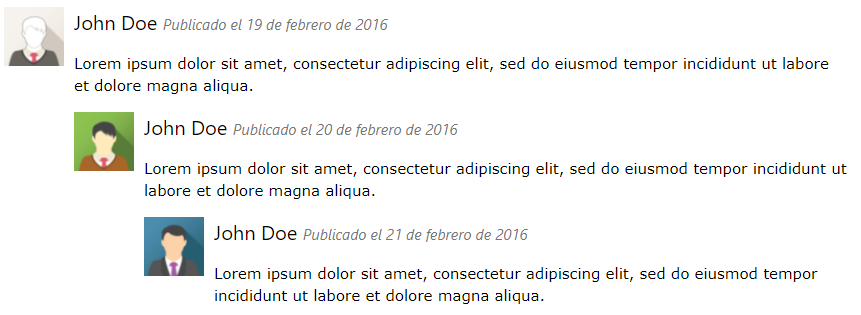
Además, puede utilizar .media-heading para los encabezados.

### Alineación superior, media o inferior:



<!-- Media top -->  
<div class="media">  
  <div class="media-left media-top">  
    <img src="img\_avatar1.png" class="media-object" style="width:60px">  
  </div>  
  <div class="media-body">  
    <h4 class="media-heading">Media Top</h4>  
    <p>Lorem ipsum...</p>  
  </div>  
</div>  
  
<!-- Media middle -->  
<div class="media">  
  <div class="media-left media-middle">  
    <img src="img\_avatar1.png" class="media-object" style="width:60px">  
  </div>  
  <div class="media-body">  
    <h4 class="media-heading">Media Middle</h4>  
    <p>Lorem ipsum...</p>  
  </div>  
</div>  
  
<!-- Media bottom -->  
<div class="media">  
  <div class="media-left media-bottom">  
    <img src="img\_avatar1.png" class="media-object" style="width:60px">  
  </div>  
  <div class="media-body">  
    <h4 class="media-heading">Media Bottom</h4>  
    <p>Lorem ipsum...</p>  
  </div>  
</div>

### Anidar objetos multimedia:



<div class="media">

<div class="media-left">

<img src="img\_avatar1.png" class="media-object" style="width:45px">

</div>

<div class="media-body">

<h4 class="media-heading">John Doe <small><i>Posted on February 19, 2016</i></small></h4>

<p>Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.</p>

<!-- Nested media object -->

<div class="media">

<div class="media-left">

<img src="img\_avatar2.png" class="media-object" style="width:45px">

</div>

<div class="media-body">

<h4 class="media-heading">John Doe <small><i>Posted on February 19, 2016</i></small></h4>

<p>Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.</p>

<!-- Nested media object -->

<div class="media">

<div class="media-left">

<img src="img\_avatar3.png" class="media-object" style="width:45px">

</div>

<div class="media-body">

<h4 class="media-heading">John Doe <small><i>Posted on February 19, 2016</i></small></h4>

<p>Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.</p>

</div>

</div>

</div>

</div>

</div>

</div>

</div>

## Bootstrap Carousel

El complemento Carousel es un componente para recorrer elementos, como un carrusel (presentación de diapositivas).

<div id="myCarousel" class="carousel slide" data-ride="carousel">  
  <!-- Indicators -->  
  <ol class="carousel-indicators">  
    <li data-target="#myCarousel" data-slide-to="0" class="active"></li>  
    <li data-target="#myCarousel" data-slide-to="1"></li>  
    <li data-target="#myCarousel" data-slide-to="2"></li>  
  </ol>  
  
  <!-- Wrapper for slides -->  
  <div class="carousel-inner">  
    <div class="item active">  
      <img src="la.jpg" alt="Los Angeles">  
    </div>  
  
    <div class="item">  
      <img src="chicago.jpg" alt="Chicago">  
    </div>  
  
    <div class="item">  
      <img src="ny.jpg" alt="New York">  
    </div>  
  </div>  
  
  <!-- Left and right controls -->  
  <a class="left carousel-control" href="#myCarousel" data-slide="prev">  
    <span class="glyphicon glyphicon-chevron-left"></span>  
    <span class="sr-only">Previous</span>  
  </a>  
  <a class="right carousel-control" href="#myCarousel" data-slide="next">  
    <span class="glyphicon glyphicon-chevron-right"></span>  
    <span class="sr-only">Next</span>  
  </a>  
</div>

**El <div> más externo:**

Los carruseles requieren el uso de una identificación (en este caso id="myCarousel") para que los controles del carrusel funcionen correctamente.

Las class="carousel" especifica que este <div>contiene un carrusel.

La clase .slide agrega una transición CSS y un efecto de animación, que hace que los elementos se deslicen cuando se muestra un elemento nuevo. Omita esta clase si no desea este efecto.

El atributo data-ride="carousel" le dice a Bootstrap que comience a animar el carrusel inmediatamente cuando se carga la página.

**La parte "Indicadores":**

Los indicadores son los pequeños puntos en la parte inferior de cada diapositiva (que indican cuántas diapositivas hay en el carrusel y qué diapositiva está viendo el usuario en ese momento).

Los indicadores se especifican en una lista ordenada con clase .carousel-indicators.

El atributo data-target apunta a la identificación del carrusel.

El atributo data-slide-to específica a qué diapositiva ir, al hacer clic en el punto específico.

**La parte "Envoltorio para diapositivas":**

Las diapositivas se especifican en una clase <div> con .carousel-inner.

El contenido de cada diapositiva se define en una clase <div> con .item. Puede ser texto o imágenes.

La clase .active debe agregarse a una de las diapositivas. De lo contrario, el carrusel no será visible.

**La parte "Controles izquierdo y derecho":**

Este código agrega botones "izquierda" y "derecha" que permiten al usuario ir y venir entre las diapositivas manualmente.

El atributo data-slide acepta las palabras clave "prev"o "next", que altera la posición de la diapositiva en relación con su posición actual.

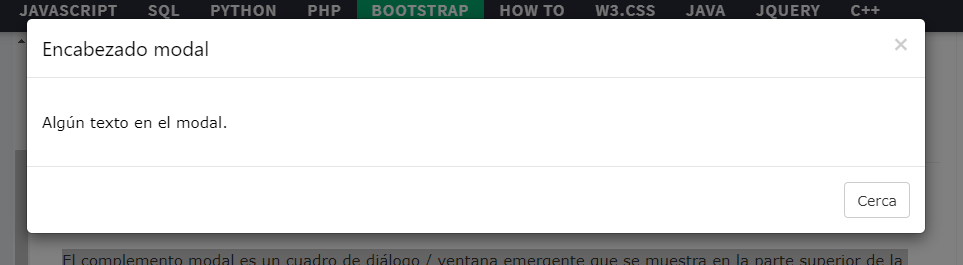
### Agregar subtítulos a las diapositivas:

Agregue <div class="carousel-caption">dentro de cada uno <div class="item">para crear un título para cada diapositiva:

<div id="myCarousel" class="carousel slide" data-ride="carousel">  
  <!-- Indicators -->  
  <ol class="carousel-indicators">  
    <li data-target="#myCarousel" data-slide-to="0" class="active"></li>  
    <li data-target="#myCarousel" data-slide-to="1"></li>  
    <li data-target="#myCarousel" data-slide-to="2"></li>  
  </ol>  
  
  <!-- Wrapper for slides -->  
  <div class="carousel-inner">  
    <div class="item active">  
      <img src="la.jpg" alt="Chania">  
      <div class="carousel-caption">  
        <h3>Los Angeles</h3>  
        <p>LA is always so much fun!</p>  
      </div>  
    </div>  
  
    <div class="item">  
      <img src="chicago.jpg" alt="Chicago">  
      <div class="carousel-caption">  
        <h3>Chicago</h3>  
        <p>Thank you, Chicago!</p>  
      </div>  
    </div>  
  
    <div class="item">  
      <img src="ny.jpg" alt="New York">  
      <div class="carousel-caption">  
        <h3>New York</h3>  
        <p>We love the Big Apple!</p>  
      </div>  
    </div>  
  </div>  
  
  <!-- Left and right controls -->  
  <a class="left carousel-control" href="#myCarousel" data-slide="prev">  
    <span class="glyphicon glyphicon-chevron-left"></span>  
    <span class="sr-only">Previous</span>  
  </a>  
  <a class="right carousel-control" href="#myCarousel" data-slide="next">  
    <span class="glyphicon glyphicon-chevron-right"></span>  
    <span class="sr-only">Next</span>  
  </a>  
</div>

## Modal Bootstrap

El complemento modal es un cuadro de diálogo / ventana emergente que se muestra en la parte superior de la página actual



<!-- Trigger the modal with a button -->  
<button type="button" class="btn btn-info btn-lg" data-toggle="modal" data-target="#myModal">Open Modal</button>  
  
<!-- Modal -->  
<div id="myModal" class="modal fade" role="dialog">  
  <div class="modal-dialog">  
  
    <!-- Modal content-->  
    <div class="modal-content">  
      <div class="modal-header">  
        <button type="button" class="close" data-dismiss="modal">&times;</button>  
        <h4 class="modal-title">Modal Header</h4>  
      </div>  
      <div class="modal-body">  
        <p>Some text in the modal.</p>  
      </div>  
      <div class="modal-footer">  
        <button type="button" class="btn btn-default" data-dismiss="modal">Close</button>  
      </div>  
    </div>  
  
  </div>  
</div>

**La parte "Trigger":**

Para activar la ventana modal, debe usar un botón o un enlace.

Luego incluya los dos atributos data- \*:

* data-toggle="modal" abre la ventana modal
* data-target="#myModal" apunta al id del modal

**La parte "modal":**

El padre <div>del modal debe tener un ID que sea el mismo que el valor del atributo de destino de datos utilizado para activar el modal ("myModal").

La clase .modal identifica el contenido de <div> como modal y lo enfoca.

La clase .fade agrega un efecto de transición que desvanece el modo de entrada y salida. Elimine esta clase si no desea este efecto.

El atributo role="dialog" mejora la accesibilidad para las personas que utilizan lectores de pantalla.

La clase .modal-dialog establece el ancho y el margen adecuados del modal.

**La parte "Contenido modal":**

El " <div> con class="modal-content" aplica estilo al modal (borde, color de fondo, etc.). Dentro de esto <div>, agregue el encabezado, el cuerpo y el pie de página del modal.

La clase .modal-header se usa para definir el estilo del encabezado del modal. El <button>interior del encabezado tiene un atributo data-dismiss="modal" que cierra el modal si hace clic en él. La clase .close le da estilo al botón de cierre y la clase .modal-title le da estilo al encabezado con una altura de línea adecuada.

La clase .modal-body se usa para definir el estilo del cuerpo del modal. Agregue aquí cualquier marcado HTML; párrafos, imágenes, videos, etc.

La clase .modal-footer se usa para definir el estilo del pie de página del modal. Tenga en cuenta que esta área está alineada a la derecha de forma predeterminada.

### Tamaño modal:

Cambie el tamaño del modal agregando la clase .modal-sm para modales pequeños o la clase .modal-lg para modales grandes.

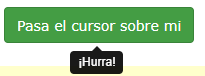
Agregue la clase de tamaño al <div>elemento con clase .modal-dialog:

<div class="modal-dialog modal-sm">

<div class="modal-dialog modal-lg">

## Bootstrap Tooltip

El complemento Tooltip es un pequeño cuadro emergente que aparece cuando el usuario mueve el puntero del mouse sobre un elemento:



Para crear una información sobre herramientas, agregue el atributo data-toggle="tooltip" a un elemento.

Utilice el atributo title para especificar el texto que debe mostrarse dentro de la información sobre herramientas:

<a href="#" data-toggle="tooltip" title="Hooray!">Hover over me</a>

Nota: La información sobre herramientas debe inicializarse con jQuery: seleccione el elemento especificado y llame al método tooltip().

El siguiente código habilitará todas las descripciones emergentes del documento:

<script>  
$(document).ready(function(){  
  $('[data-toggle="tooltip"]').tooltip();  
});  
</script>

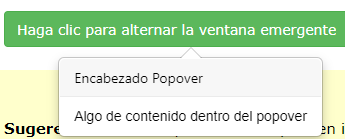
De forma predeterminada, la información sobre herramientas aparecerá en la parte superior del elemento.

Utilice el atributo data-placement para establecer la posición de la información sobre herramientas en la parte superior, inferior, izquierda o derecha del elemento:

<a href="#" data-toggle="tooltip" data-placement="top" title="Hooray!">Hover</a>  
<a href="#" data-toggle="tooltip" data-placement="bottom" title="Hooray!">Hover</a>  
<a href="#" data-toggle="tooltip" data-placement="left" title="Hooray!">Hover</a>  
<a href="#" data-toggle="tooltip" data-placement="right" title="Hooray!">Hover</a>

## Bootstrap Popover

El complemento Popover es similar a la información sobre herramientas; es un cuadro emergente que aparece cuando el usuario hace clic en un elemento. La diferencia es que el popover puede contener mucho más contenido.



Para crear un popover, agregue el atributo data-toggle="popover" a un elemento.

Use el atributo title para especificar el texto del encabezado del popover y use el atributo data-content para especificar el texto que debe mostrarse dentro del cuerpo del popover:

<a href="#" data-toggle="popover" title="Popover Header" data-content="Some content inside the popover">Toggle popover</a>

Nota: Los popovers deben inicializarse con jQuery: seleccione el elemento especificado y llame al método popover().

El siguiente código habilitará todos los popovers en el documento:

<script>  
$(document).ready(function(){  
  $('[data-toggle="popover"]').popover();  
});  
</script>

De forma predeterminada, la ventana emergente aparecerá en el lado derecho del elemento.

Utilice el atributo data-placement para establecer la posición de la ventana emergente en la parte superior, inferior, izquierda o derecha del elemento:

<a href="#" title="Header" data-toggle="popover" data-placement="top" data-content="Content">Click</a>  
<a href="#" title="Header" data-toggle="popover" data-placement="bottom" data-content="Content">Click</a>  
<a href="#" title="Header" data-toggle="popover" data-placement="left" data-content="Content">Click</a>  
<a href="#" title="Header" data-toggle="popover" data-placement="right" data-content="Content">Click</a>

De forma predeterminada, la ventana emergente se cierra cuando vuelve a hacer clic en el elemento. Sin embargo, puede usar el atributo data-trigger="focus" que cerrará la ventana emergente al hacer clic fuera del elemento:

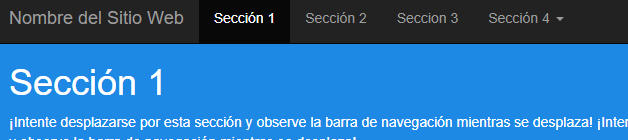
<a href="#" title="Dismissible popover" data-toggle="popover" data-trigger="focus" data-content="Click anywhere in the document to close this popover">Click me</a>

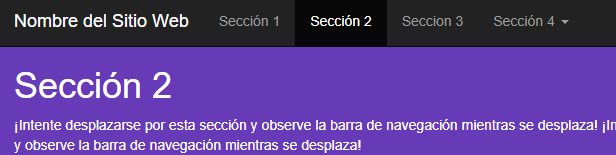
Sugerencia: Si desea que se muestre el popover cuando mueva el puntero del mouse sobre el elemento, use el atributo data-trigger con un valor de "hover":

<a href="#" title="Header" data-toggle="popover" data-trigger="hover" data-content="Some content">Hover over me</a>

## Bootstrap Scrollspy

El complemento Scrollspy se utiliza para actualizar automáticamente los enlaces en una lista de navegación según la posición de desplazamiento.





<!-- The scrollable area -->  
<body data-spy="scroll" data-target=".navbar" data-offset="50">  
  
<!-- The navbar - The <a> elements are used to jump to a section in the scrollable area -->  
<nav class="navbar navbar-inverse navbar-fixed-top">  
...  
  <ul class="nav navbar-nav">  
    <li><a href="#section1">Section 1</a></li>  
    ...  
</nav>  
  
<!-- Section 1 -->  
<div id="section1">  
  <h1>Section 1</h1>  
  <p>Try to scroll this page and look at the navigation bar while scrolling!</p>  
</div>  
...  
  
</body>

Agregue data-spy="scroll" al elemento que debe usarse como área desplazable (a menudo, este es el elemento <body>).

Luego agregue el atributo data-target con un valor de id o el nombre de clase de la barra de navegación ( .navbar). Esto es para asegurarse de que la barra de navegación esté conectada con el área desplazable.

Tenga en cuenta que los elementos desplazables deben coincidir con el ID de los enlaces dentro de los elementos de la lista de la barra de navegación ( <div id="section1">coincidencias <a href="#section1">).

El atributo data-offset opcional específica el número de píxeles a compensar desde la parte superior al calcular la posición del desplazamiento. Esto es útil cuando cree que los enlaces dentro de la barra de navegación cambian el estado activo demasiado pronto o demasiado pronto al saltar a los elementos desplazables. El valor predeterminado es 10 píxeles.

## Bootstrap Affix

El complemento Affix permite que un elemento se adhiera (bloquee) en un área de la página. Esto se usa a menudo con menús de navegación o botones de íconos sociales, para que se "peguen" en un área específica mientras se desplaza hacia arriba y hacia abajo en la página.

El plugin cambia este comportamiento dentro y fuera (cambia el valor de CSS position partir statica fixed), dependiendo de la posición de desplazamiento.

Con Affix, cuando nos desplazamos hacia arriba y hacia abajo por la página, el menú siempre está visible y bloqueado en su posición.

El siguiente ejemplo muestra cómo crear un menú de navegación fijo horizontal:

<nav class="navbar navbar-inverse" data-spy="affix" data-offset-top="197">

El siguiente ejemplo muestra cómo crear un menú de navegación vertical fijo:

<ul class="nav nav-pills nav-stacked" data-spy="affix" data-offset-top="205">

Agregue data-spy="affix"al elemento que desea colocar.

Opcionalmente, agregue el atributo data-offset-top|bottom para calcular la posición del desplazamiento.

El afijo complemento alterna entre tres clases: .affix, .affix-top, y .affix-bottom. Cada clase representa un estado particular. Debe agregar propiedades CSS para manejar las posiciones reales, con la excepción de position:fixed en la clase .affix.

* El complemento agrega la clase .affix-topo .affix-bottom para indicar que el elemento está en su posición más alta o más baja. El posicionamiento con CSS no es necesario en este momento.
* Al desplazarse más allá del elemento adjunto, se activa la colocación real: aquí es donde el complemento reemplaza la clase .affix-topo .affix-bottom con la clase .affix (conjuntos position:fixed). En este punto, debe agregar el CSS topo la propiedad bottom para colocar el elemento adjunto en la página.
* Si se define un desplazamiento inferior, desplazarse más allá reemplaza la clase .affix con .affix-bottom. Dado que las compensaciones son opcionales, configurar una requiere que configure el CSS apropiado. En este caso, agregue position:absolute cuando sea necesario.

En el primer ejemplo anterior, el complemento Affix agrega la clase .affix (posición: fija) al elemento <nav> cuando nos hemos desplazado 197 píxeles desde la parte superior. Si abre el ejemplo, también verá que agregamos la propiedad top CSS con un valor de 0 a la clase .affix. Esto es para asegurarnos de que la barra de navegación permanezca en la parte superior de la página en todo momento, cuando hayamos desplazado 197 píxeles desde la parte superior.

### Scrollspy & Affix:

Usando el complemento Affix junto con el complemento Scrollspy:

Horizontal:

<body data-spy="scroll" data-target=".navbar" data-offset="50">  
  
<nav class="navbar navbar-inverse" data-spy="affix" data-offset-top="197">  
...  
</nav>  
  
</body>

Vertical:

<body data-spy="scroll" data-target="#myScrollspy" data-offset="15">  
  
<nav class="col-sm-3" id="myScrollspy">  
  <ul class="nav nav-pills nav-stacked" data-spy="affix" data-offset-top="205">  
  ...  
</nav>  
  
</body>

## Filtros Bootstrap

Bootstrap no tiene un componente que permita el filtrado. Sin embargo, podemos usar jQuery para filtrar / buscar elementos.

### Tablas de filtros:

Realice una búsqueda que no distinga entre mayúsculas y minúsculas de elementos en una tabla:

<script>  
$(document).ready(function(){  
  $("#myInput").on("keyup", function() {  
    var value = $(this).val().toLowerCase();  
    $("#myTable tr").filter(function() {  
      $(this).toggle($(this).text().toLowerCase().indexOf(value) > -1)  
    });  
  });  
});  
</script>

Usamos jQuery para recorrer cada fila de la tabla para verificar si hay algún valor de texto que coincida con el valor del campo de entrada. El toggle()método oculta la fila ( display:none) que no coincide con la búsqueda. Usamos el toLowerCase()método para convertir el texto a minúsculas, lo que hace que la búsqueda no distinga entre mayúsculas y minúsculas (permite "john", "John" e incluso "JOHN" en la búsqueda).

### Listas de filtros:

Realice una búsqueda que no distinga entre mayúsculas y minúsculas de los elementos de una lista:

<script>

$(document).ready(function(){

$("#myInput").on("keyup", function() {

var value = $(this).val().toLowerCase();

$("#myList li").filter(function() {

$(this).toggle($(this).text().toLowerCase().indexOf(value) > -1)

});

});

});

</script>

### Filtrar menús desplegables:

Realice una búsqueda que no distinga entre mayúsculas y minúsculas de elementos en un menú desplegable:

<script>

$(document).ready(function(){

$("#myInput").on("keyup", function() {

var value = $(this).val().toLowerCase();

$(".dropdown-menu li").filter(function() {

$(this).toggle($(this).text().toLowerCase().indexOf(value) > -1)

});

});

});

</script>

# PHP

PHP es un lenguaje de programación de servidores y una poderosa herramienta para crear páginas web dinámicas e interactivas.

PHP es una alternativa eficiente, gratuita y ampliamente utilizada frente a competidores como ASP de Microsoft.

* PHP es un acrónimo de "PHP: Preprocesador de hipertexto"
* PHP es un lenguaje de programación de código abierto ampliamente utilizado
* Los scripts PHP se ejecutan en el servidor.
* PHP se puede descargar y usar gratis

**Archivo php**

* Los archivos PHP pueden contener texto, HTML, CSS, JavaScript y código PHP
* El código PHP se ejecuta en el servidor y el resultado se devuelve al navegador como HTML sin formato.
* Los archivos PHP tienen la extensión " .php"

**¿Qué puede hacer PHP?**

* PHP puede generar contenido de página dinámico
* PHP puede crear, abrir, leer, escribir, eliminar y cerrar archivos en el servidor
* PHP puede recopilar datos de formularios
* PHP puede enviar y recibir cookies
* PHP puede agregar, eliminar, modificar datos en su base de datos
* PHP se puede utilizar para controlar el acceso de los usuarios
* PHP puede cifrar datos

## Sintaxis PHP

Se puede colocar un script PHP en cualquier lugar del documento.

Un script PHP comienza <?phpy termina con ?>:

La extensión de archivo predeterminada para los archivos PHP es " .php".

Un archivo PHP normalmente contiene etiquetas HTML y algo de código de programación PHP.

A continuación, tenemos un ejemplo de un archivo PHP simple, con un script PHP que usa una función PHP incorporada " echo" para generar el texto "¡Hola mundo!" en una página web:

<!DOCTYPE html>  
<html>  
<body>  
  
<h1>My first PHP page</h1>  
  
<?php  
echo "Hello World!";  
?>  
  
</body>  
</html>

Nota: las declaraciones de PHP terminan con un punto y coma ( ;).

En PHP, las palabras clave (por ejemplo if, else, while, echo, etc.), las clases, funciones, y funciones definidos por el usuario no entre mayúsculas y minúsculas.

En el siguiente ejemplo, las tres declaraciones de eco a continuación son iguales y legales:

<!DOCTYPE html>  
<html>  
<body>  
  
<?php  
ECHO "Hello World!<br>";  
echo "Hello World!<br>";  
EcHo "Hello World!<br>";  
?>  
  
</body>  
</html>

Nota: Sin embargo; ¡Todos los nombres de variables distinguen entre mayúsculas y minúsculas!

Mira el ejemplo de abajo; ¡solo la primera declaración mostrará el valor de la variable $color! Esto se debe a que $color, $COLOR y $coLOR se tratan como tres variables diferentes:

<!DOCTYPE html>  
<html>  
<body>  
  
<?php  
$color = "red";  
echo "My car is " . $color . "<br>";  
echo "My house is " . $COLOR . "<br>";  
echo "My boat is " . $coLOR . "<br>";  
?>  
  
</body>  
</html>

### Comentarios en PHP:

Sintaxis para comentarios de una sola línea:

<!DOCTYPE html>  
<html>  
<body>  
  
<?php  
// This is a single-line comment  
  
# This is also a single-line comment  
?>  
  
</body>  
</html>

Sintaxis para comentarios en varias líneas:

<!DOCTYPE html>  
<html>  
<body>  
  
<?php  
/\*  
This is a multiple-lines comment block  
that spans over multiple  
lines  
\*/  
?>  
  
</body>  
</html>

Usando comentarios para omitir partes del código:

<!DOCTYPE html>  
<html>  
<body>  
  
<?php  
// You can also use comments to leave out parts of a code line  
$x = 5 /\* + 15 \*/ + 5;  
echo $x;  
?>  
  
</body>  
</html>

### Variables PHP:

En PHP, una variable comienza con el $signo, seguido del nombre de la variable:

<?php  
$txt = "Hello world!";  
$x = 5;  
$y = 10.5;  
?>

Después de la ejecución de las declaraciones anteriores, la variable $txt mantendrá el valor Hello world!, la variable $x mantendrá el valor 5y la variable $y mantendrá el valor 10.5.

Nota: cuando asigne un valor de texto a una variable, coloque comillas alrededor del valor.

Nota: a diferencia de otros lenguajes de programación, PHP no tiene ningún comando para declarar una variable. Se crea en el momento en que le asigna un valor por primera vez.

Una variable puede tener un nombre corto (como x e y) o un nombre más descriptivo (edad, nombre del coche, volumen\_total).

Reglas para variables PHP:

* Una variable comienza con el signo $, seguido del nombre de la variable.
* El nombre de una variable debe comenzar con una letra o el carácter de subrayado
* Un nombre de variable no puede comenzar con un número
* El nombre de una variable solo puede contener caracteres alfanuméricos y guiones bajos (Az, 0-9 y \_)
* Los nombres de las variables distinguen entre mayúsculas y minúsculas ( $agey $AGEson dos variables diferentes)

**Variables de salida**

La declaración echo de PHP se usa a menudo para enviar datos a la pantalla.

<?php  
$txt = "W3Schools.com";  
echo "I love $txt!";  
?>

La concatenación es con punto (.)

<?php  
$txt = "W3Schools.com";  
echo "I love " . $txt . "!";  
?>

El siguiente ejemplo generará la suma de dos variables:

<?php  
$x = 5;  
$y = 4;  
echo $x + $y;  
?>

### Alcance de las variables de PHP:

En PHP, las variables se pueden declarar en cualquier parte del script.

El alcance de una variable es la parte del script donde se puede hacer referencia / usar la variable.

PHP tiene tres ámbitos variables diferentes:

* local
* global
* estático

### Alcance global y local:

Una variable declarada fuera de una función tiene un ALCANCE GLOBAL y solo se puede acceder fuera de una función:

<?php  
$x = 5; // global scope  
  
function myTest() {  
  // using x inside this function will generate an error  
  echo "<p>Variable x inside function is: $x</p>";  
}  
myTest();  
  
echo "<p>Variable x outside function is: $x</p>";  
?>

Una variable declarada dentro de una función tiene un ALCANCE LOCAL y solo se puede acceder dentro de esa función:

<?php  
function myTest() {  
  $x = 5; // local scope  
  echo "<p>Variable x inside function is: $x</p>";  
}  
myTest();  
  
// using x outside the function will generate an error  
echo "<p>Variable x outside function is: $x</p>";  
?>

### PHP La palabra clave global:

La palabra global clave se utiliza para acceder a una variable global desde dentro de una función.

Para hacer esto, use la palabra global clave antes de las variables (dentro de la función):

<?php  
$x = 5;  
$y = 10;  
  
function myTest() {  
  global $x, $y;  
  $y = $x + $y;  
}  
  
myTest();  
echo $y; // outputs 15  
?>

PHP también almacena todas las variables globales en una matriz llamada $GLOBALS[index]index . El contiene el nombre de la variable. Esta matriz también es accesible desde funciones y se puede utilizar para actualizar variables globales directamente.

<?php  
$x = 5;  
$y = 10;  
  
function myTest() {  
  $GLOBALS['y'] = $GLOBALS['x'] + $GLOBALS['y'];  
}  
  
myTest();  
echo $y; // outputs 15  
?>

### PHP La palabra clave static:

Normalmente, cuando se completa / ejecuta una función, se eliminan todas sus variables. Sin embargo, a veces queremos que NO se elimine una variable local. Lo necesitamos para otro trabajo.

Para hacer esto, use la palabra static clave cuando declare la variable por primera vez:

<?php  
function myTest() {  
  static $x = 0;  
  echo $x;  
  $x++;  
}  
  
myTest();  
myTest();  
myTest();  
?>

Entonces, cada vez que se llama a la función, esa variable seguirá teniendo la información que contenía desde la última vez que se llamó a la función.

Nota: la variable sigue siendo local a la función.

### Declaraciones echo y print en PHP:

Con PHP, hay dos formas básicas de obtener resultados: echo y print

Echo y print son más o menos iguales. Ambos se utilizan para enviar datos a la pantalla.

Las diferencias son pequeñas: echo no tiene valor de retorno, mientras que print tiene un valor de retorno de 1, por lo que se puede usar en expresiones. Echo puede tomar múltiples parámetros (aunque tal uso es raro) mientras que print puede tomar un argumento. Echo es ligeramente más rápido que print.

**Echo:**

La declaración echo se puede utilizar con o sin paréntesis: echo o echo().

<?php  
echo "<h2>PHP is Fun!</h2>";  
echo "Hello world!<br>";  
echo "I'm about to learn PHP!<br>";  
echo "This ", "string ", "was ", "made ", "with multiple parameters.";  
?>

El siguiente ejemplo muestra cómo generar texto y variables con la declaración echo:

<?php  
$txt1 = "Learn PHP";  
$txt2 = "W3Schools.com";  
$x = 5;  
$y = 4;  
  
echo "<h2>" . $txt1 . "</h2>";  
echo "Study PHP at " . $txt2 . "<br>";  
echo $x + $y;  
?>

**Print:**

La declaración print se puede utilizar con o sin paréntesis: print o print().

<?php  
print "<h2>PHP is Fun!</h2>";  
print "Hello world!<br>";  
print "I'm about to learn PHP!";  
?>

El siguiente ejemplo muestra cómo generar texto y variables con la declaración print:

<?php  
$txt1 = "Learn PHP";  
$txt2 = "W3Schools.com";  
$x = 5;  
$y = 4;  
  
print "<h2>" . $txt1 . "</h2>";  
print "Study PHP at " . $txt2 . "<br>";  
print $x + $y;  
?>

## Tipos de datos PHP

PHP admite los siguientes tipos de datos:

* String

<?php  
$x = "Hello world!";  
$y = 'Hello world!';  
  
echo $x;  
echo "<br>";  
echo $y;  
?>

* Integer

Reglas para enteros:

* Un número entero debe tener al menos un dígito
* Un número entero no debe tener un punto decimal
* Un número entero puede ser positivo o negativo
* Los números enteros se pueden especificar en: notación decimal (base 10), hexadecimal (base 16), octal (base 8) o binaria (base 2)

En el siguiente ejemplo, $ x es un número entero. La función PHP var\_dump () devuelve el tipo y valor de datos:

<?php  
$x = 5985;  
var\_dump($x);  
?>

PHP tiene las siguientes constantes predefinidas para enteros:

* PHP\_INT\_MAX: el entero más grande admitido
* PHP\_INT\_MIN: el número entero más pequeño admitido
* PHP\_INT\_SIZE: el tamaño de un número entero en bytes

PHP tiene las siguientes funciones para verificar si el tipo de una variable es entero:

* + is\_int ()
  + is\_integer () - alias de is\_int ()
  + is\_long () - alias de is\_int ()
* Float (floating point numbers - also called double)

<?php  
$x = 10.365;  
var\_dump($x);  
?>

PHP tiene las siguientes constantes predefinidas para flotantes (de PHP 7.2):

* PHP\_FLOAT\_MAX: el número de punto flotante más grande representable
* PHP\_FLOAT\_MIN - El número de punto flotante positivo representable más pequeño
* PHP\_FLOAT\_MAX - El número de coma flotante negativo representable más pequeño
* PHP\_FLOAT\_DIG: el número de dígitos decimales que se pueden redondear a un flotante y viceversa sin pérdida de precisión
* PHP\_FLOAT\_EPSILON: el número positivo representable más pequeño x, de modo que x + 1.0! = 1.0

PHP tiene las siguientes funciones para verificar si el tipo de variable es flotante:

* is\_float ()
* is\_double () - alias de is\_float ()
* Boolean

$x = true;  
$y = false;

* Array

<?php  
$cars = array("Volvo","BMW","Toyota");  
var\_dump($cars);  
?>

* Object

Las clases y los objetos son los dos aspectos principales de la programación orientada a objetos.

Una clase es una plantilla para objetos y un objeto es una instancia de una clase.

Cuando se crean los objetos individuales, heredan todas las propiedades y comportamientos de la clase, pero cada objeto tendrá valores diferentes para las propiedades.

Supongamos que tenemos una clase llamada Car. Un automóvil puede tener propiedades como modelo, color, etc. Podemos definir variables como $ modelo, $ color, etc., para mantener los valores de estas propiedades.

Cuando se crean los objetos individuales (Volvo, BMW, Toyota, etc.), heredan todas las propiedades y comportamientos de la clase, pero cada objeto tendrá valores diferentes para las propiedades.

Si crea una función \_\_construct (), PHP llamará automáticamente a esta función cuando cree un objeto a partir de una clase.

<?php  
class Car {  
  public $color;  
  public $model;  
  public function \_\_construct($color, $model) {  
    $this->color = $color;  
    $this->model = $model;  
  }  
  public function message() {  
    return "My car is a " . $this->color . " " . $this->model . "!";  
  }  
}  
  
$myCar = new Car("black", "Volvo");  
echo $myCar -> message();  
echo "<br>";  
$myCar = new Car("red", "Toyota");  
echo $myCar -> message();  
?>

* NULL

Null es un tipo de datos especial que solo puede tener un valor: NULL.

Una variable de tipo de datos NULL es una variable que no tiene ningún valor asignado.

Sugerencia: Si se crea una variable sin un valor, se le asigna automáticamente un valor de NULL.

Las variables también se pueden vaciar estableciendo el valor en NULL:

<?php  
$x = "Hello world!";  
$x = null;  
var\_dump($x);  
?>

* Resource

El tipo de recurso especial no es un tipo de datos real. Es el almacenamiento de una referencia a funciones y recursos externos a PHP.

Un ejemplo común de uso del tipo de datos de recursos es una llamada a la base de datos.

## Strings PHP

* La función PHP strlen()devuelve la longitud de una cadena:

<?php  
echo strlen("Hello world!"); // outputs 12  
?>

* La función PHP str\_word\_count() cuenta el número de palabras en una cadena.

<?php  
echo str\_word\_count("Hello world!"); // outputs 2  
?>

* La función PHP strrev() invierte una cadena.

<?php  
echo strrev("Hello world!"); // outputs !dlrow olleH  
?>

* La función PHP strpos() busca un texto específico dentro de una cadena. Si se encuentra una coincidencia, la función devuelve la posición del carácter de la primera coincidencia. Si no se encuentra ninguna coincidencia, devolverá FALSE.

<?php  
echo strpos("Hello world!", "world"); // outputs 6  
?>

* La función PHP str\_replace() reemplaza algunos caracteres con otros caracteres en una cadena.

<?php  
echo str\_replace("world", "Dolly", "Hello world!"); // outputs Hello Dolly!  
?>

## Matemáticas PHP

* La función pi()devuelve el valor de PI:

<?php  
echo(pi()); // returns 3.1415926535898  
?>

* Las funciones min() y max()se pueden usar para encontrar el valor más bajo o más alto en una lista de argumentos:

<?php  
echo(min(0, 150, 30, 20, -8, -200));  // returns -200  
echo(max(0, 150, 30, 20, -8, -200));  // returns 150  
?>

* La función abs()devuelve el valor absoluto (positivo) de un número:

<?php  
echo(abs(-6.7));  // returns 6.7  
?>

* La función sqrt()devuelve la raíz cuadrada de un número:

<?php  
echo(sqrt(64));  // returns 8  
?>

* La función round()redondea un número de punto flotante a su entero más cercano:

<?php  
echo(round(0.60));  // returns 1  
echo(round(0.49));  // returns 0  
?>

* La función rand()genera un número aleatorio:

<?php  
echo(rand());  
?>

Para obtener más control sobre el número aleatorio, puede añadir la opción min y max parámetros para especificar el número entero más bajo y el más alto número entero a ser devueltos.

<?php  
echo(rand(10, 100));  
?>

## Constantes PHP

Para crear una constante, use la función define().

define(*name*, *value*, *case-insensitive*)

* name : especifica el nombre de la constante
* value : especifica el valor de la constante
* case-insensitive: especifica si el nombre de la constante no debe distinguir entre mayúsculas y minúsculas. El valor predeterminado es falso.

Las constantes son automáticamente globales y se pueden utilizar en todo el script.

Distingue entre mayúsculas y minúsculas:

<?php  
define("GREETING", "Welcome to W3Schools.com!");  
echo GREETING;  
?>

No distingue entre mayúsculas y minúsculas:

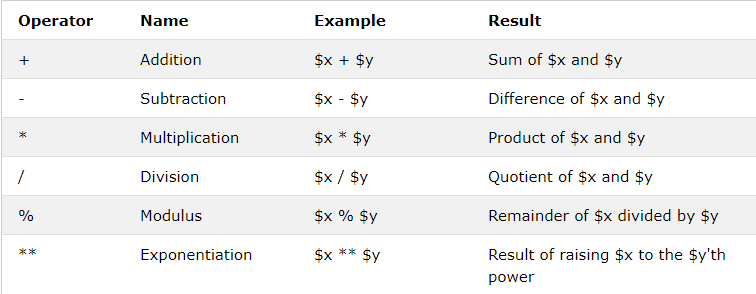
<?php  
define("GREETING", "Welcome to W3Schools.com!", true);  
echo greeting;  
?>

### Matrices constantes PHP:

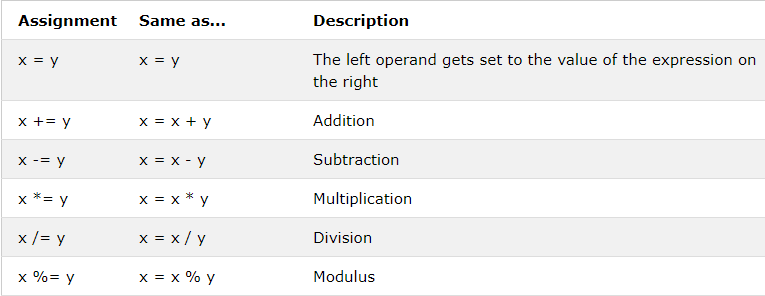
<?php  
define("cars", [  
  "Alfa Romeo",  
  "BMW",  
  "Toyota"  
]);  
echo cars[0];  
?>

## Operadores PHP

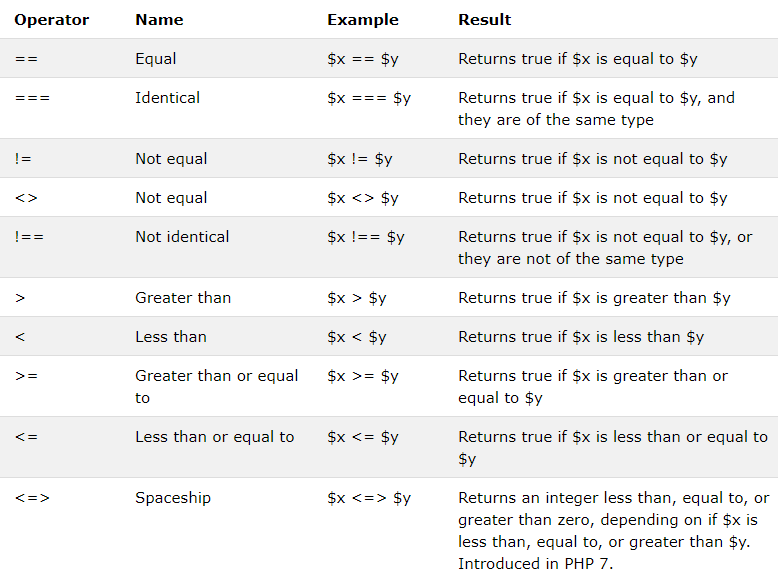
**Operadores aritméticos PHP:**



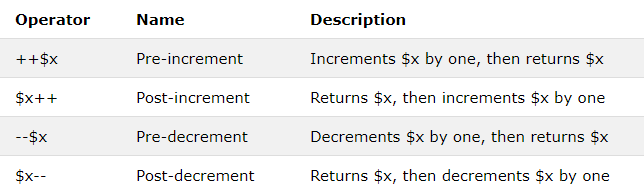
**Operadores de asignación de PHP:**

****

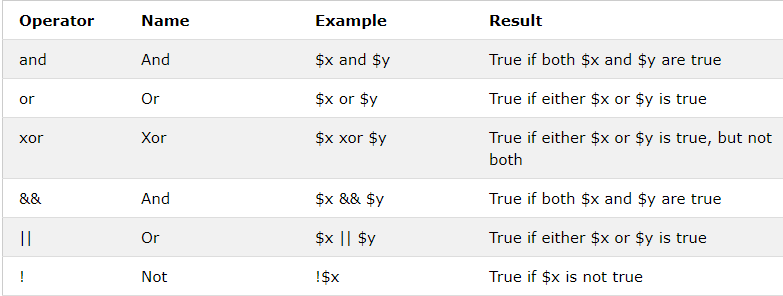
**Operadores de comparación de PHP:**

****

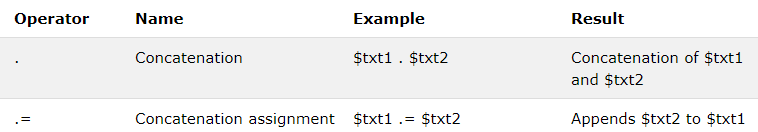
**Operadores de incremento / decremento de PHP:**

****

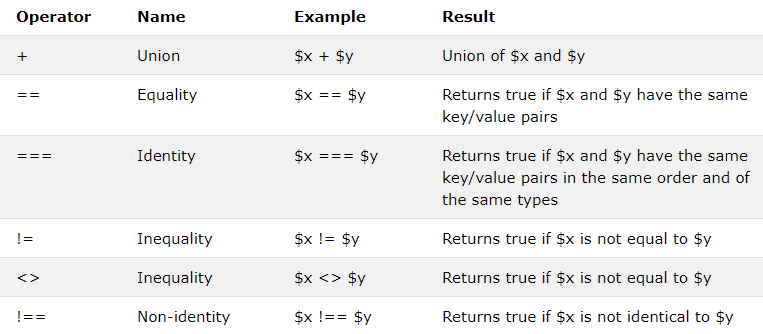
**Operadores lógicos PHP:**

****

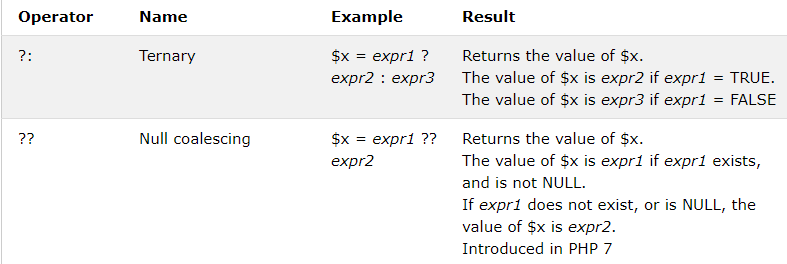
**Operadores de strings PHP:**

****

**Operadores de matriz PHP:**

****

**Operadores de asignación condicional de PHP:**

****

## Declaraciones condicionales de PHP

### PHP - La declaración if:

<?php  
$t = date("H");  
  
if ($t < "20") {  
  echo "Have a good day!";  
}  
?>

### PHP - La declaración if ... else:

<?php  
$t = date("H");  
  
if ($t < "20") {  
  echo "Have a good day!";  
} else {  
  echo "Have a good night!";  
}  
?>

### PHP - La declaración if ... elseif ... else:

<?php  
$t = date("H");  
  
if ($t < "10") {  
  echo "Have a good morning!";  
} elseif ($t < "20") {  
  echo "Have a good day!";  
} else {  
  echo "Have a good night!";  
}  
?>

## La declaración de switch de PHP

<?php  
$favcolor = "red";  
  
switch ($favcolor) {  
  case "red":  
    echo "Your favorite color is red!";  
    break;  
  case "blue":  
    echo "Your favorite color is blue!";  
    break;  
  case "green":  
    echo "Your favorite color is green!";  
    break;  
  default:  
    echo "Your favorite color is neither red, blue, nor green!";  
}  
?>

## Bucles PHP

### While:

El bucle while ejecuta un bloque de código siempre que la condición especificada sea verdadera.

<?php  
$x = 1;  
  
while($x <= 5) {  
  echo "The number is: $x <br>";  
  $x++;  
}  
?>

### Do while:

El ciclo do... while siempre ejecutará el bloque de código una vez, luego comprobará la condición y repetirá el ciclo mientras la condición especificada sea verdadera.

<?php  
$x = 1;  
  
do {  
  echo "The number is: $x <br>";  
  $x++;  
} while ($x <= 5);  
?>

### For:

El bucle for recorre un bloque de código un número específico de veces.

Parámetros:

* contador de inicialización: inicializa el valor del contador de bucle
* contador de prueba: evaluado para cada iteración del ciclo. Si se evalúa como VERDADERO, el ciclo continúa. Si se evalúa como FALSE, el ciclo termina.
* contador de incrementos : aumenta el valor del contador de bucle

<?php  
for ($x = 0; $x <= 10; $x++) {  
  echo "The number is: $x <br>";  
}  
?>

### Foreach:

El bucle foreach recorre un bloque de código para cada elemento de una matriz.

El bucle foreach funciona solo en matrices y se utiliza para recorrer cada par clave / valor en una matriz.

Para cada iteración del ciclo, el valor del elemento de la matriz actual se asigna $ a value y el puntero de la matriz se mueve en uno, hasta que alcanza el último elemento de la matriz.

El siguiente ejemplo generará los valores de la matriz dada ($ colores):

<?php  
$colors = array("red", "green", "blue", "yellow");  
  
foreach ($colors as $value) {  
  echo "$value <br>";  
}  
?>

El siguiente ejemplo generará tanto las claves como los valores de la matriz dada ($ age):

<?php  
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");  
  
foreach($age as $x => $val) {  
  echo "$x = $val<br>";  
}  
?>

### Break & Continue:

**Break:**

Ya ha visto la declaración break utilizada en un capítulo anterior de este tutorial. Se utilizó para "saltar" de una declaración switch.

La declaración break también se puede usar para saltar de un bucle.

<?php  
for ($x = 0; $x < 10; $x++) {  
  if ($x == 4) {  
    break;  
  }  
  echo "The number is: $x <br>";  
}  
?>

**Continue:**

La declaración continue rompe una iteración (en el ciclo), si ocurre una condición especificada, y continúa con la siguiente iteración en el ciclo.

<?php  
for ($x = 0; $x < 10; $x++) {  
  if ($x == 4) {  
    continue;  
  }  
  echo "The number is: $x <br>";  
}  
?>

## Funciones PHP

* Una función es un bloque de declaraciones que se pueden usar repetidamente en un programa.
* Una función no se ejecutará automáticamente cuando se cargue una página.
* Una función se ejecutará mediante una llamada a la función.

<?php  
function writeMsg() {  
  echo "Hello world!";  
}  
  
writeMsg(); // call the function  
?>

### Argumentos de funciones PHP:

La información se puede pasar a funciones a través de argumentos. Un argumento es como una variable.

Los argumentos se especifican después del nombre de la función, entre paréntesis. Puede agregar tantos argumentos como desee, solo sepárelos con una coma.

<?php  
function familyName($fname) {  
  echo "$fname Refsnes.<br>";  
}  
  
familyName("Jani");  
familyName("Hege");  
familyName("Stale");  
familyName("Kai Jim");  
familyName("Borge");  
?>

<?php  
function familyName($fname, $year) {  
  echo "$fname Refsnes. Born in $year <br>";  
}  
  
familyName("Hege", "1975");  
familyName("Stale", "1978");  
familyName("Kai Jim", "1983");  
?>

En el ejemplo anterior, observe que no tuvimos que decirle a PHP qué tipo de datos es la variable.

PHP asocia automáticamente un tipo de datos a la variable, dependiendo de su valor. Dado que los tipos de datos no se establecen en un sentido estricto, puede hacer cosas como agregar una cadena a un número entero sin causar un error.

En PHP 7, se agregaron declaraciones de tipo. Esto nos da una opción para especificar el tipo de datos esperado al declarar una función, y al agregar la declaración strict, arrojará un "Error fatal" si el tipo de datos no coincide.

En el siguiente ejemplo, intentamos enviar un número y una cadena a la función sin usar strict:

<?php  
function addNumbers(int $a, int $b) {  
  return $a + $b;  
}  
echo addNumbers(5, "5 days");  
// since strict is NOT enabled "5 days" is changed to int(5), and it will return 10  
?>

Para especificar strict necesitamos establecer declare(strict\_types=1);. Debe estar en la primera línea del archivo PHP.

En el siguiente ejemplo intentamos enviar tanto un número como una cadena a la función, pero aquí hemos agregado la declaración strict:

<?php declare(strict\_types=1); // strict requirement  
  
function addNumbers(int $a, int $b) {  
  return $a + $b;  
}  
echo addNumbers(5, "5 days");  
// since strict is enabled and "5 days" is not an integer, an error will be thrown  
?>

### Valor de argumento predeterminado de PHP:

El siguiente ejemplo muestra cómo utilizar un parámetro predeterminado. Si llamamos a la función setHeight () sin argumentos, toma el valor predeterminado como argumento:

<?php declare(strict\_types=1); // strict requirement  
function setHeight(int $minheight = 50) {  
  echo "The height is : $minheight <br>";  
}  
  
setHeight(350);  
setHeight(); // will use the default value of 50  
setHeight(135);  
setHeight(80);  
?>

### Devolución de valores:

Para permitir que una función devuelva un valor, use la declaración return:

<?php declare(strict\_types=1); // strict requirement  
function sum(int $x, int $y) {  
  $z = $x + $y;  
  return $z;  
}  
  
echo "5 + 10 = " . sum(5, 10) . "<br>";  
echo "7 + 13 = " . sum(7, 13) . "<br>";  
echo "2 + 4 = " . sum(2, 4);  
?>

PHP 7 también admite declaraciones de tipo para la declaración return. Al igual que con la declaración de tipo para argumentos de función, al habilitar el requisito estricto, arrojará un "Error fatal" en una falta de coincidencia de tipos.

Para declarar un tipo para el retorno de la función, agregue dos puntos ( :) y el tipo justo antes del {corchete de apertura ( ) al declarar la función.

<?php declare(strict\_types=1); // strict requirement  
function addNumbers(float $a, float $b) : float {  
  return $a + $b;  
}  
echo addNumbers(1.2, 5.2);  
?>

Puede especificar un tipo de retorno diferente al de los tipos de argumento, pero asegúrese de que el retorno sea del tipo correcto:

<?php declare(strict\_types=1); // strict requirement  
function addNumbers(float $a, float $b) : int {  
  return (int)($a + $b);  
}  
echo addNumbers(1.2, 5.2);  
?>

### Pasar argumentos por referencia:

En PHP, los argumentos generalmente se pasan por valor, lo que significa que se usa una copia del valor en la función y la variable que se pasó a la función no se puede cambiar.

Cuando un argumento de función se pasa por referencia, los cambios en el argumento también cambian la variable que se pasó. Para convertir un argumento de función en una referencia, & se usa el operador:

<?php  
function add\_five(&$value) {  
  $value += 5;  
}  
  
$num = 2;  
add\_five($num);  
echo $num;  
?>

## Matrices PHP

Una matriz almacena varios valores en una sola variable:

<?php  
$cars = array("Volvo", "BMW", "Toyota");  
echo "I like " . $cars[0] . ", " . $cars[1] . " and " . $cars[2] . ".";  
?>

En PHP, hay tres tipos de matrices:

* Matrices indexadas : matrices con un índice numérico
* Matrices asociativas : matrices con claves con nombre
* Matrices multidimensionales : matrices que contienen una o más matrices

La función count()se usa para devolver la longitud (el número de elementos) de una matriz:

<?php  
$cars = array("Volvo", "BMW", "Toyota");  
echo count($cars);  
?>

### Matrices indexadas PHP:

Hay dos formas de crear matrices indexadas:

El índice se puede asignar automáticamente (el índice siempre comienza en 0), así:

$cars = array("Volvo", "BMW", "Toyota");

o el índice se puede asignar manualmente:

$cars[0] = "Volvo";  
$cars[1] = "BMW";  
$cars[2] = "Toyota";

Para recorrer e imprimir todos los valores de una matriz indexada, puede usar un bucle for, como este:

<?php  
$cars = array("Volvo", "BMW", "Toyota");  
$arrlength = count($cars);  
  
for($x = 0; $x < $arrlength; $x++) {  
  echo $cars[$x];  
  echo "<br>";  
}  
?>

### Matrices asociativas PHP:

Las matrices asociativas son matrices que utilizan claves con nombre que les asigna.

Hay dos formas de crear una matriz asociativa:

$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");

O

$age['Peter'] = "35";  
$age['Ben'] = "37";  
$age['Joe'] = "43";

Las claves nombradas se pueden usar en un script:

<?php  
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");  
echo "Peter is " . $age['Peter'] . " years old.";  
?>

Para recorrer e imprimir todos los valores de una matriz asociativa, puede usar un bucle foreach, como este:

<?php  
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");  
  
foreach($age as $x => $x\_value) {  
  echo "Key=" . $x . ", Value=" . $x\_value;  
  echo "<br>";  
}  
?>

### Matrices multidimensionales:

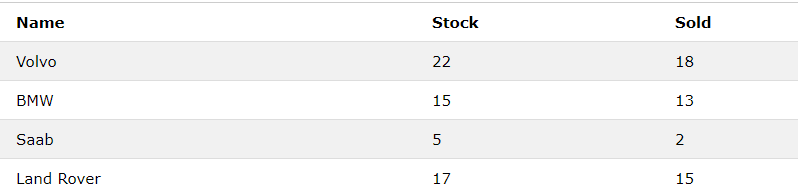
Una matriz multidimensional es una matriz que contiene una o más matrices.

PHP admite matrices multidimensionales que tienen dos, tres, cuatro, cinco o más niveles de profundidad. Sin embargo, las matrices de más de tres niveles de profundidad son difíciles de administrar para la mayoría de las personas.

La dimensión de una matriz indica el número de índices que necesita para seleccionar un elemento.

* Para una matriz bidimensional, necesita dos índices para seleccionar un elemento
* Para una matriz tridimensional, necesita tres índices para seleccionar un elemento

### Matrices bidimensionales:



Podemos almacenar los datos de la tabla anterior en una matriz bidimensional, como esta:

$cars = array (  
  array("Volvo",22,18),  
  array("BMW",15,13),  
  array("Saab",5,2),  
  array("Land Rover",17,15)  
);

Ahora, la matriz bidimensional $ cars contiene cuatro matrices y tiene dos índices: fila y columna.

Para acceder a los elementos de la matriz $ cars debemos apuntar a los dos índices (fila y columna):

<?php  
echo $cars[0][0].": In stock: ".$cars[0][1].", sold: ".$cars[0][2].".<br>";  
echo $cars[1][0].": In stock: ".$cars[1][1].", sold: ".$cars[1][2].".<br>";  
echo $cars[2][0].": In stock: ".$cars[2][1].", sold: ".$cars[2][2].".<br>";  
echo $cars[3][0].": In stock: ".$cars[3][1].", sold: ".$cars[3][2].".<br>";  
?>

También podemos poner un bucle for dentro de otro bucle for para obtener los elementos de la matriz $ cars (todavía tenemos que apuntar a los dos índices):

<?php  
for ($row = 0; $row < 4; $row++) {  
  echo "<p><b>Row number $row</b></p>";  
  echo "<ul>";  
  for ($col = 0; $col < 3; $col++) {  
    echo "<li>".$cars[$row][$col]."</li>";  
  }  
  echo "</ul>";  
}  
?>

## Funciones de ordenación para matrices

En este capítulo, repasaremos las siguientes funciones de ordenación de matrices de PHP:

* sort() - ordenar matrices en orden ascendente

El siguiente ejemplo ordena los elementos de la matriz $ cars en orden alfabético ascendente:

<?php  
$cars = array("Volvo", "BMW", "Toyota");  
sort($cars);  
?>

El siguiente ejemplo ordena los elementos de la matriz $ números en orden numérico ascendente:

<?php  
$numbers = array(4, 6, 2, 22, 11);  
sort($numbers);  
?>

* rsort() - ordenar matrices en orden descendente

El siguiente ejemplo ordena los elementos de la matriz $ cars en orden alfabético descendente:

<?php  
$cars = array("Volvo", "BMW", "Toyota");  
rsort($cars);  
?>

El siguiente ejemplo ordena los elementos de la matriz $ números en orden numérico descendente:

<?php  
$numbers = array(4, 6, 2, 22, 11);  
rsort($numbers);  
?>

* asort() - ordenar matrices asociativas en orden ascendente, de acuerdo con el valor

<?php  
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");  
asort($age);  
?>

* ksort() - ordenar matrices asociativas en orden ascendente, de acuerdo con la clave

<?php  
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");  
ksort($age);  
?>

* arsort() - ordenar matrices asociativas en orden descendente, de acuerdo con el valor

<?php  
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");  
arsort($age);  
?>

* krsort() - ordenar matrices asociativas en orden descendente, de acuerdo con la clave

<?php  
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");  
krsort($age);  
?>

## Variables globales de PHP – Superglobales

Algunas variables predefinidas en PHP son "superglobales", lo que significa que siempre son accesibles, independientemente del alcance, y puede acceder a ellas desde cualquier función, clase o archivo sin tener que hacer nada especial.

Las variables superglobales de PHP son:

* $ GLOBALES

$ GLOBALS es una variable súper global de PHP que se utiliza para acceder a variables globales desde cualquier lugar del script PHP (también desde dentro de funciones o métodos).

PHP almacena todas las variables globales en una matriz llamada $ GLOBALS [ índice ]. El índice contiene el nombre de la variable.

<?php  
$x = 75;  
$y = 25;  
   
function addition() {  
  $GLOBALS['z'] = $GLOBALS['x'] + $GLOBALS['y'];  
}  
   
addition();  
echo $z;  
?>

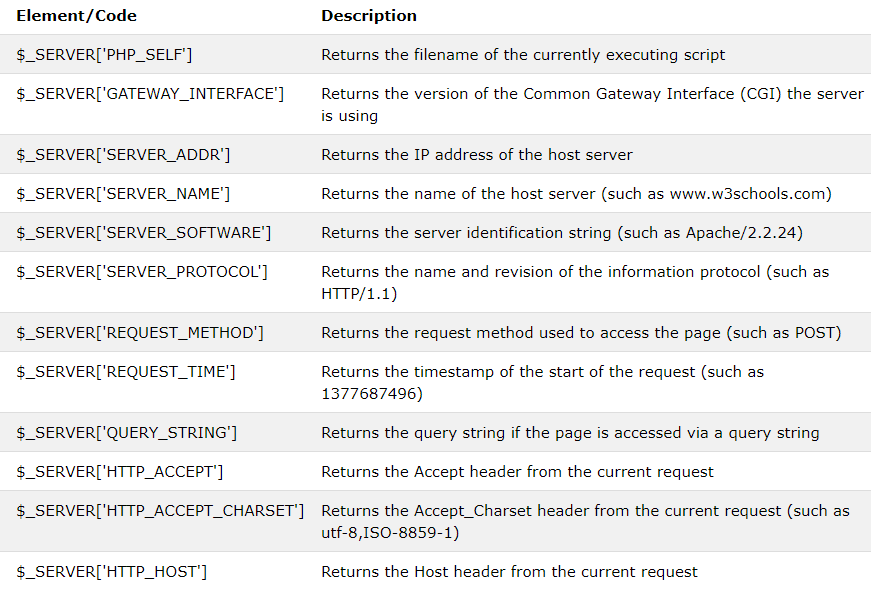
En el ejemplo anterior, dado que z es una variable presente dentro de la matriz $ GLOBALS, ¡también es accesible desde fuera de la función!

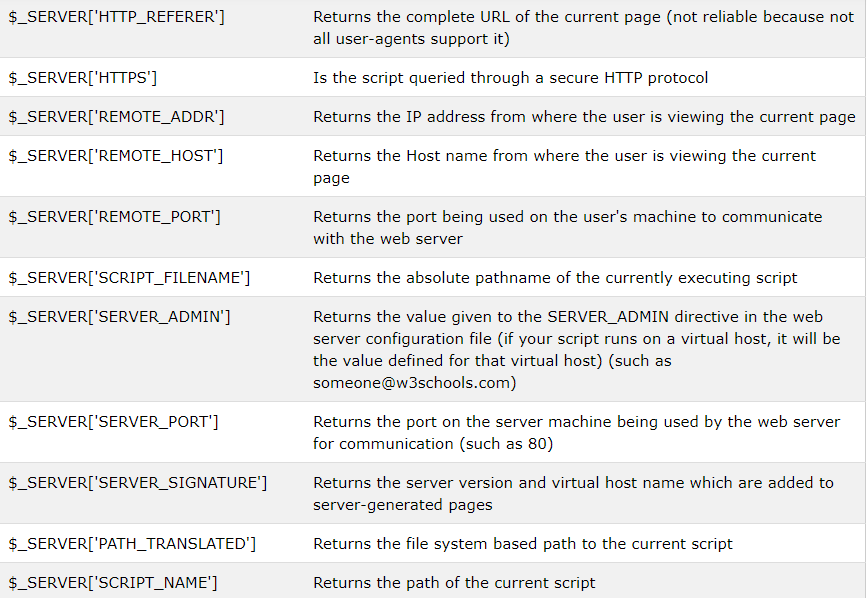
* $ \_SERVER

$ \_SERVER es una variable súper global de PHP que contiene información sobre encabezados, rutas y ubicaciones de scripts.

<?php  
echo $\_SERVER['PHP\_SELF'];  
echo "<br>";  
echo $\_SERVER['SERVER\_NAME'];  
echo "<br>";  
echo $\_SERVER['HTTP\_HOST'];  
echo "<br>";  
echo $\_SERVER['HTTP\_REFERER'];  
echo "<br>";  
echo $\_SERVER['HTTP\_USER\_AGENT'];  
echo "<br>";  
echo $\_SERVER['SCRIPT\_NAME'];  
?>

La siguiente tabla enumera los elementos más importantes que pueden ir dentro de $ \_SERVER:





* $ \_REQUEST

PHP $ \_REQUEST es una variable súper global de PHP que se utiliza para recopilar datos después de enviar un formulario HTML.

El siguiente ejemplo muestra un formulario con un campo de entrada y un botón de envío. Cuando un usuario envía los datos haciendo clic en "Enviar", los datos del formulario se envían al archivo especificado en el atributo de acción de la etiqueta <form>. En este ejemplo, apuntamos a este archivo para procesar los datos del formulario. Si desea utilizar otro archivo PHP para procesar los datos del formulario, reemplácelo con el nombre de archivo de su elección. Luego, podemos usar la variable súper global $ \_REQUEST para recopilar el valor del campo de entrada:  
<form method="post" action="<?php echo $\_SERVER['PHP\_SELF'];?>">  
  Name: <input type="text" name="fname">  
  <input type="submit">  
</form>  
  
<?php  
if ($\_SERVER["REQUEST\_METHOD"] == "POST") {  
  // collect value of input field  
  $name = $\_REQUEST['fname'];  
  if (empty($name)) {  
    echo "Name is empty";  
  } else {  
    echo $name;  
  }  
}  
?>

* $ \_POST

PHP $ \_POST es una variable súper global de PHP que se utiliza para recopilar datos de formularios después de enviar un formulario HTML con method = "post". $ \_POST también se usa ampliamente para pasar variables.

El siguiente ejemplo muestra un formulario con un campo de entrada y un botón de envío. Cuando un usuario envía los datos haciendo clic en "Enviar", los datos del formulario se envían al archivo especificado en el atributo de acción de la etiqueta <form>. En este ejemplo, apuntamos al archivo en sí para procesar los datos del formulario. Si desea utilizar otro archivo PHP para procesar los datos del formulario, reemplácelo con el nombre de archivo de su elección. Luego, podemos usar la variable súper global $ \_POST para recopilar el valor del campo de entrada:

<form method="post" action="<?php echo $\_SERVER['PHP\_SELF'];?>">  
  Name: <input type="text" name="fname">  
  <input type="submit">  
</form>  
  
<?php  
if ($\_SERVER["REQUEST\_METHOD"] == "POST") {  
  // collect value of input field  
  $name = $\_POST['fname'];  
  if (empty($name)) {  
    echo "Name is empty";  
  } else {  
    echo $name;  
  }  
}  
?>

* $ \_GET

PHP $ \_GET es una variable súper global de PHP que se utiliza para recopilar datos de formularios después de enviar un formulario HTML con method = "get".

$ \_GET también puede recopilar datos enviados en la URL.

Supongamos que tenemos una página HTML que contiene un hipervínculo con parámetros:

<a href="test\_get.php?subject=PHP&web=W3schools.com">Test $GET</a>

Cuando un usuario hace clic en el enlace "Test $ GET", los parámetros "subject" y "web" se envían a "test\_get.php", y luego puede acceder a sus valores en "test\_get.php" con $ \_GET.

<?php  
echo "Study " . $\_GET['subject'] . " at " . $\_GET['web'];  
?>

* $ \_FILES
* $ \_ENV
* $ \_GALLETA
* $ \_SESIÓN

## Expresiones regulares PHP

Una expresión regular es una secuencia de caracteres que forma un patrón de búsqueda. Cuando busca datos en un texto, puede utilizar este patrón de búsqueda para describir lo que está buscando.

Una expresión regular puede ser un solo carácter o un patrón más complicado.

Las expresiones regulares se pueden utilizar para realizar todo tipo de operaciones de búsqueda y reemplazo de texto.

En PHP, las expresiones regulares son cadenas compuestas por delimitadores, un patrón y modificadores opcionales.

$exp = "/w3schools/i";

En el ejemplo anterior, /es el delimitador , w3schools es el patrón que se está buscando y i es un modificador que hace que la búsqueda no distinga entre mayúsculas y minúsculas.

El delimitador puede ser cualquier carácter que no sea una letra, un número, una barra invertida o un espacio. El delimitador más común es la barra inclinada (/), pero cuando su patrón contiene barras inclinadas, es conveniente elegir otros delimitadores como # o ~.

### Funciones de expresión regular:

PHP proporciona una variedad de funciones que le permiten utilizar expresiones regulares. Las funciones preg\_match(), preg\_match\_all()y preg\_replace()son algunas de las más utilizadas:

* La función preg\_match()le dirá si una cadena contiene coincidencias de un patrón.

Utilice una expresión regular para realizar una búsqueda que no distinga entre mayúsculas y minúsculas de "w3schools" en una cadena:

<?php  
$str = "Visit W3Schools";  
$pattern = "/w3schools/i";  
echo preg\_match($pattern, $str); // Outputs 1  
?>

* La función preg\_match\_all() le dirá cuántas coincidencias se encontraron para un patrón en una cadena.

Utilice una expresión regular para hacer un recuento que no distinga entre mayúsculas y minúsculas del número de apariciones de "ain" en una cadena:

<?php  
$str = "The rain in SPAIN falls mainly on the plains.";  
$pattern = "/ain/i";  
echo preg\_match\_all($pattern, $str); // Outputs 4  
?>

* La función preg\_replace()reemplazará todas las coincidencias del patrón en una cadena con otra cadena.

Use una expresión regular que no distinga entre mayúsculas y minúsculas para reemplazar Microsoft con W3Schools en una cadena:

<?php  
$str = "Visit Microsoft!";  
$pattern = "/microsoft/i";  
echo preg\_replace($pattern, "W3Schools", $str); // Outputs "Visit W3Schools!"  
?>

<https://www.w3schools.com/php/php_regex.asp>

## Manejo de formularios PHP

El siguiente ejemplo muestra un formulario HTML simple con dos campos de entrada y un botón de envío:

<html>  
<body>  
  
<form action="welcome.php" method="post">  
Name: <input type="text" name="name"><br>  
E-mail: <input type="text" name="email"><br>  
<input type="submit">  
</form>  
  
</body>  
</html>

Cuando el usuario completa el formulario anterior y hace clic en el botón enviar, los datos del formulario se envían para su procesamiento a un archivo PHP llamado "welcome.php". Los datos del formulario se envían con el método HTTP POST.

Para mostrar los datos enviados, simplemente puede repetir todas las variables. El "welcome.php" se ve así:

<html>  
<body>  
  
Welcome <?php echo $\_POST["name"]; ?><br>  
Your email address is: <?php echo $\_POST["email"]; ?>  
  
</body>  
</html>

La salida podría ser algo como esto:

Welcome John  
Your email address is [john.doe@example.com](mailto:john.doe@example.com)

También se podría lograr el mismo resultado utilizando el método HTTP GET:

<html>  
<body>  
  
<form action="welcome\_get.php" method="get">  
Name: <input type="text" name="name"><br>  
E-mail: <input type="text" name="email"><br>  
<input type="submit">  
</form>  
  
</body>  
</html>

y "welcome\_get.php" se ve así:

<html>  
<body>  
  
Welcome <?php echo $\_GET["name"]; ?><br>  
Your email address is: <?php echo $\_GET["email"]; ?>  
  
</body>  
</html>

### GET VS POST:

Tanto GET como POST crean una matriz (por ejemplo, matriz (clave1 => valor1, clave2 => valor2, clave3 => valor3, ...)). Esta matriz contiene pares clave / valor, donde las claves son los nombres de los controles de formulario y los valores son los datos de entrada del usuario.

Tanto GET como POST se tratan como $ \_GET y $ \_POST. Estos son superglobales, lo que significa que siempre son accesibles, independientemente del alcance, y puede acceder a ellos desde cualquier función, clase o archivo sin tener que hacer nada especial.

$ \_GET es una matriz de variables que se pasan al script actual a través de los parámetros de URL.

$ \_POST es una matriz de variables que se pasan al script actual a través del método HTTP POST.

**¿Cuándo usar GET?**

La información enviada desde un formulario con el método GET es visible para todos (todos los nombres y valores de las variables se muestran en la URL). GET también tiene límites en la cantidad de información para enviar. La limitación es de unos 2000 caracteres. Sin embargo, debido a que las variables se muestran en la URL, es posible marcar la página. Esto puede resultar útil en algunos casos.

GET se puede utilizar para enviar datos no confidenciales.

Nota: ¡NUNCA se debe utilizar GET para enviar contraseñas u otra información confidencial!

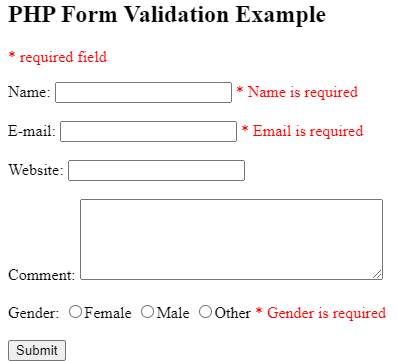
**¿Cuándo usar POST?**

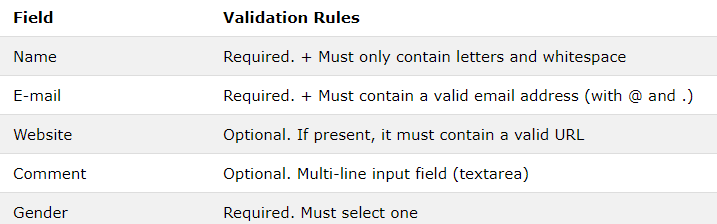
La información enviada desde un formulario con el método POST es invisible para los demás (todos los nombres / valores están incrustados en el cuerpo de la solicitud HTTP) y no tiene límites en la cantidad de información a enviar.

Además, POST admite funciones avanzadas, como la compatibilidad con entradas binarias de varias partes mientras se cargan archivos en el servidor.

Sin embargo, debido a que las variables no se muestran en la URL, no es posible marcar la página.

## Validación de formularios PHP





Name: <input type="text" name="name">  
E-mail: <input type="text" name="email">  
Website: <input type="text" name="website">  
Comment: <textarea name="comment" rows="5" cols="40"></textarea>

Gender:  
<input type="radio" name="gender" value="female">Female  
<input type="radio" name="gender" value="male">Male  
<input type="radio" name="gender" value="other">Other

El código HTML del formulario se ve así:

<form method="post" action="<?php echo htmlspecialchars($\_SERVER["PHP\_SELF"]);?>">

$ \_SERVER ["PHP\_SELF"] es una variable súper global que devuelve el nombre de archivo del script que se está ejecutando actualmente.

Entonces, $ \_SERVER ["PHP\_SELF"] envía los datos del formulario enviado a la página misma, en lugar de saltar a una página diferente. De esta forma, el usuario recibirá mensajes de error en la misma página que el formulario.

La función htmlspecialchars () convierte caracteres especiales en entidades HTML. Esto significa que reemplazará los caracteres HTML como <y> por & lt; y & gt ;. Esto evita que los atacantes exploten el código inyectando código HTML o Javascript (ataques de secuencias de comandos entre sitios) en los formularios.

### Nota importante sobre la seguridad de los formularios PHP:

¡Los piratas informáticos pueden utilizar la variable $ \_SERVER ["PHP\_SELF"]!

Si PHP\_SELF se usa en su página, entonces un usuario puede ingresar una barra (/) y luego algunos comandos Cross Site Scripting (XSS) para ejecutar.

La secuencia de comandos entre sitios (XSS) es un tipo de vulnerabilidad de seguridad informática que normalmente se encuentra en aplicaciones web. XSS permite a los atacantes inyectar secuencias de comandos del lado del cliente en las páginas web que ven otros usuarios.

Supongamos que tenemos el siguiente formulario en una página llamada "test\_form.php":

<form method="post" action="<?php echo $\_SERVER["PHP\_SELF"];?>">

Ahora, si un usuario ingresa la URL normal en la barra de direcciones como "http://www.example.com/test\_form.php", el código anterior se traducirá a:

<form method="post" action="test\_form.php">

Hasta aquí todo bien.

Sin embargo, considere que un usuario ingresa la siguiente URL en la barra de direcciones:

<http://www.example.com/test_form.php/%22%3E%3Cscript%3Ealert('hacked')%3C/script%3E>

En este caso, el código anterior se traducirá a:

<form method="post" action="test\_form.php/"><script>alert('hacked')</script>

Este código agrega una etiqueta de secuencia de comandos y un comando de alerta. Y cuando se cargue la página, se ejecutará el código JavaScript (el usuario verá un cuadro de alerta). Este es solo un ejemplo simple e inofensivo de cómo se puede explotar la variable PHP\_SELF.

¡Tenga en cuenta que se puede agregar cualquier código JavaScript dentro de la etiqueta <script>! Un pirata informático puede redirigir al usuario a un archivo en otro servidor, y ese archivo puede contener código malicioso que puede alterar las variables globales o enviar el formulario a otra dirección para guardar los datos del usuario, por ejemplo.

### ¿Cómo evitar las vulnerabilidades de $ \_SERVER ["PHP\_SELF"]?

Las vulnerabilidades de $ \_SERVER ["PHP\_SELF"] se pueden evitar usando la función htmlspecialchars ().

El código del formulario debería verse así:

<form method="post" action="<?php echo htmlspecialchars($\_SERVER["PHP\_SELF"]);?>">

La función htmlspecialchars () convierte caracteres especiales en entidades HTML. Ahora, si el usuario intenta explotar la variable PHP\_SELF, dará como resultado el siguiente resultado:

<form method="post" action="test\_form.php/&quot;&gt;&lt;script&gt;alert('hacked')&lt;/script&gt;">

¡El intento de explotación falla y no se hace daño!

### Validar datos de formulario con PHP:

Lo primero que haremos es pasar todas las variables a través de la función htmlspecialchars () de PHP.

Cuando usamos la función htmlspecialchars (); luego, si un usuario intenta enviar lo siguiente en un campo de texto:

<script> location.href ('http://www.hacked.com') </script>

- esto no se ejecutaría, porque se guardaría como código HTML de escape, así:

& lt; script & gt; location.href ('http://www.hacked.com') & lt; / script & gt;

El código ahora es seguro para mostrarse en una página o dentro de un correo electrónico.

También haremos dos cosas más cuando el usuario envíe el formulario:

1. Elimine los caracteres innecesarios (espacio adicional, tabulación, nueva línea) de los datos de entrada del usuario (con la función PHP trim ())
2. Elimina las barras invertidas (\) de los datos de entrada del usuario (con la función PHP stripslashes ())

El siguiente paso es crear una función que haga todas las comprobaciones por nosotros (lo cual es mucho más conveniente que escribir el mismo código una y otra vez).

Denominaremos la función test\_input ().

Ahora, podemos verificar cada variable $ \_POST con la función test\_input (), y el script se ve así:

<?php  
// define variables and set to empty values  
$name = $email = $gender = $comment = $website = "";  
  
if ($\_SERVER["REQUEST\_METHOD"] == "POST") {  
  $name = test\_input($\_POST["name"]);  
  $email = test\_input($\_POST["email"]);  
  $website = test\_input($\_POST["website"]);  
  $comment = test\_input($\_POST["comment"]);  
  $gender = test\_input($\_POST["gender"]);  
}  
  
function test\_input($data) {  
  $data = trim($data);  
  $data = stripslashes($data);  
  $data = htmlspecialchars($data);  
  return $data;  
}  
?>

Tenga en cuenta que al comienzo del script, verificamos si el formulario se ha enviado utilizando $ \_SERVER ["REQUEST\_METHOD"]. Si REQUEST\_METHOD es POST, entonces el formulario ha sido enviado y debe ser validado. Si no se ha enviado, omita la validación y muestre un formulario en blanco.

Sin embargo, en el ejemplo anterior, todos los campos de entrada son opcionales. El script funciona bien incluso si el usuario no ingresa ningún dato.

El siguiente paso es hacer que los campos de entrada sean obligatorios y crear mensajes de error si es necesario.

## Formularios PHP - Campos obligatorios

En el siguiente código, hemos agregado algunas variables nuevas: $ nameErr, $ emailErr, $ genderErr y $ websiteErr. Estas variables de error contendrán mensajes de error para los campos obligatorios. También hemos agregado una declaración if else para cada variable $ \_POST. Esto verifica si la variable $ \_POST está vacía (con la función PHP empty()). Si está vacío se almacena un mensaje de error en las diferentes variables de error, y si no está vacío envía los datos de entrada del usuario a través de la función test\_input():

<?php  
// define variables and set to empty values  
$nameErr = $emailErr = $genderErr = $websiteErr = "";  
$name = $email = $gender = $comment = $website = "";  
  
if ($\_SERVER["REQUEST\_METHOD"] == "POST") {  
  if (empty($\_POST["name"])) {  
    $nameErr = "Name is required";  
  } else {  
    $name = test\_input($\_POST["name"]);  
  }  
  
  if (empty($\_POST["email"])) {  
    $emailErr = "Email is required";  
  } else {  
    $email = test\_input($\_POST["email"]);  
  }  
  
  if (empty($\_POST["website"])) {  
    $website = "";  
  } else {  
    $website = test\_input($\_POST["website"]);  
  }  
  
  if (empty($\_POST["comment"])) {  
    $comment = "";  
  } else {  
    $comment = test\_input($\_POST["comment"]);  
  }  
  
  if (empty($\_POST["gender"])) {  
    $genderErr = "Gender is required";  
  } else {  
    $gender = test\_input($\_POST["gender"]);  
  }  
}  
?>

### PHP - Mostrar los mensajes de error:

Luego, en el formulario HTML, agregamos un pequeño script después de cada campo obligatorio, que genera el mensaje de error correcto si es necesario (es decir, si el usuario intenta enviar el formulario sin completar los campos obligatorios):

<form method="post" action="<?php echo htmlspecialchars($\_SERVER["PHP\_SELF"]);?>">  
  
Name: <input type="text" name="name">  
<span class="error">\* <?php echo $nameErr;?></span>  
<br><br>  
E-mail:  
<input type="text" name="email">  
<span class="error">\* <?php echo $emailErr;?></span>  
<br><br>  
Website:  
<input type="text" name="website">  
<span class="error"><?php echo $websiteErr;?></span>  
<br><br>  
Comment: <textarea name="comment" rows="5" cols="40"></textarea>  
<br><br>  
Gender:  
<input type="radio" name="gender" value="female">Female  
<input type="radio" name="gender" value="male">Male  
<input type="radio" name="gender" value="other">Other  
<span class="error">\* <?php echo $genderErr;?></span>  
<br><br>  
<input type="submit" name="submit" value="Submit">  
  
</form>

## Formularios PHP : validar correo electrónico y URL

El siguiente código muestra una forma sencilla de verificar si el campo del nombre solo contiene letras, guiones, apóstrofos y espacios en blanco. Si el valor del campo de nombre no es válido, guarde un mensaje de error:

$name = test\_input($\_POST["name"]);  
if (!preg\_match("/^[a-zA-Z-' ]\*$/",$name)) {  
  $nameErr = "Only letters and white space allowed";  
}

La función preg\_match () busca un patrón en una cadena, devolviendo verdadero si el patrón existe y falso en caso contrario.

La forma más fácil y segura de verificar si una dirección de correo electrónico está bien formada es usar la función filter\_var () de PHP.

En el siguiente código, si la dirección de correo electrónico no está bien formada, guarde un mensaje de error:

$email = test\_input($\_POST["email"]);  
if (!filter\_var($email, FILTER\_VALIDATE\_EMAIL)) {  
  $emailErr = "Invalid email format";  
}

El siguiente código muestra una forma de verificar si la sintaxis de una dirección URL es válida (esta expresión regular también permite guiones en la URL). Si la sintaxis de la dirección URL no es válida, guarde un mensaje de error:

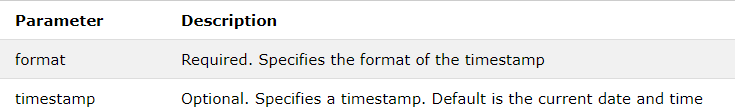
$website = test\_input($\_POST["website"]);  
if (!preg\_match("/\b(?:(?:https?|ftp):\/\/|www\.)[-a-z0-9+&@#\/%?=~\_|!:,.;]\*[-a-z0-9+&@#\/%=~\_|]/i",$website)) {  
  $websiteErr = "Invalid URL";  
}

<?php  
// define variables and set to empty values  
$nameErr = $emailErr = $genderErr = $websiteErr = "";  
$name = $email = $gender = $comment = $website = "";  
  
if ($\_SERVER["REQUEST\_METHOD"] == "POST") {  
  if (empty($\_POST["name"])) {  
    $nameErr = "Name is required";  
  } else {  
    $name = test\_input($\_POST["name"]);  
    // check if name only contains letters and whitespace  
    if (!preg\_match("/^[a-zA-Z-' ]\*$/",$name)) {  
      $nameErr = "Only letters and white space allowed";  
    }  
  }  
  
  if (empty($\_POST["email"])) {  
    $emailErr = "Email is required";  
  } else {  
    $email = test\_input($\_POST["email"]);  
    // check if e-mail address is well-formed  
    if (!filter\_var($email, FILTER\_VALIDATE\_EMAIL)) {  
      $emailErr = "Invalid email format";  
    }  
  }  
  
  if (empty($\_POST["website"])) {  
    $website = "";  
  } else {  
    $website = test\_input($\_POST["website"]);  
    // check if URL address syntax is valid (this regular expression also allows dashes in the URL)  
    if (!preg\_match("/\b(?:(?:https?|ftp):\/\/|www\.)[-a-z0-9+&@#\/%?=~\_|!:,.;]\*[-a-z0-9+&@#\/%=~\_|]/i",$website)) {  
      $websiteErr = "Invalid URL";  
    }  
  }  
  
  if (empty($\_POST["comment"])) {  
    $comment = "";  
  } else {  
    $comment = test\_input($\_POST["comment"]);  
  }  
  
  if (empty($\_POST["gender"])) {  
    $genderErr = "Gender is required";  
  } else {  
    $gender = test\_input($\_POST["gender"]);  
  }  
}  
?>

## Fecha y hora de PHP

La función date()PHP formatea una marca de tiempo en una fecha y hora más legibles.

date(*format*,*timestamp*)



Una marca de tiempo es una secuencia de caracteres que indica la fecha y / o la hora en la que ocurrió un determinado evento.

El parámetro de formato requerido de la función date () especifica cómo formatear la fecha (u hora).

A continuación, se muestran algunos caracteres que se utilizan comúnmente para las fechas:

* d - Representa el día del mes (01 a 31)
* m - Representa un mes (01 a 12)
* Y: representa un año (en cuatro dígitos)
* l ('L' minúscula): representa el día de la semana

También se pueden insertar otros caracteres, como "/", "." O "-" entre los caracteres para agregar formato adicional.

<?php  
echo "Today is " . date("Y/m/d") . "<br>";  
echo "Today is " . date("Y.m.d") . "<br>";  
echo "Today is " . date("Y-m-d") . "<br>";  
echo "Today is " . date("l");  
?>

Utilice la función date()para actualizar automáticamente el año de copyright en su sitio web:

&copy; 2010-<?php echo date("Y");?>

A continuación, se muestran algunos caracteres que se usan comúnmente para los tiempos:

* H - formato de 24 horas de una hora (00 a 23)
* h - formato de 12 horas de una hora con ceros a la izquierda (01 a 12)
* i - Minutos con ceros a la izquierda (00 a 59)
* s - Segundos con ceros a la izquierda (00 a 59)
* a - Ante meridiem y Post meridiem en minúsculas (am o pm)

<?php  
echo "The time is " . date("h:i:sa");  
?>

Tenga en cuenta que la función PHP date () devolverá la fecha / hora actual del servidor.

Si la hora a la que regresó del código no es correcta, probablemente se deba a que su servidor se encuentra en otro país o está configurado para una zona horaria diferente.

Por lo tanto, si necesita que la hora sea correcta de acuerdo con una ubicación específica, puede establecer la zona horaria que desea usar.

El siguiente ejemplo establece la zona horaria en "America / New\_York", luego muestra la hora actual en el formato especificado:

<?php  
date\_default\_timezone\_set("America/New\_York");  
echo "The time is " . date("h:i:sa");  
?>

### Crear una fecha con mktime ():

El parámetro de marca de tiempo opcional en la función date () especifica una marca de tiempo. Si se omite, se utilizarán la fecha y hora actuales (como en los ejemplos anteriores).

La función PHP mktime()devuelve la marca de tiempo de Unix para una fecha. La marca de tiempo de Unix contiene el número de segundos entre la época de Unix (1 de enero de 1970 a las 00:00:00 GMT) y la hora especificada.

mktime(hour, minute, second, month, day, year)

<?php  
$d=mktime(11, 14, 54, 8, 12, 2014);  
echo "Created date is " . date("Y-m-d h:i:sa", $d);  
?>

### Crear una fecha a partir de una cadena con strtotime ():

La función PHP strtotime()se utiliza para convertir una cadena de fecha legible por humanos en una marca de tiempo Unix (el número de segundos desde el 1 de enero de 1970 a las 00:00:00 GMT).

strtotime(time, now)

<?php  
$d=strtotime("10:30pm April 15 2014");  
echo "Created date is " . date("Y-m-d h:i:sa", $d);  
?>

## PHP Incluir archivos

La declaración include() o require() toma todo el texto / código / marcado que existe en el archivo especificado y lo copia en el archivo que usa la declaración de inclusión.

La inclusión de archivos es muy útil cuando desea incluir el mismo PHP, HTML o texto en varias páginas de un sitio web.

Es posible insertar el contenido de un archivo PHP en otro archivo PHP (antes de que el servidor lo ejecute), con la declaración include o require.

Las declaraciones include y require son idénticas, excepto en caso de falla:

* require producirá un error fatal (E\_COMPILE\_ERROR) y detendrá el script
* include solo producirá una advertencia (E\_WARNING) y el script continuará

Por lo tanto, si desea que la ejecución continúe y muestre a los usuarios la salida, incluso si falta el archivo de inclusión, use la declaración de inclusión. De lo contrario, en el caso de FrameWork, CMS o una codificación de aplicación PHP compleja, utilice siempre la instrucción require para incluir un archivo de clave para el flujo de ejecución. Esto ayudará a evitar comprometer la seguridad e integridad de su aplicación, en caso de que un archivo clave se pierda accidentalmente.

La inclusión de archivos ahorra mucho trabajo. Esto significa que puede crear un encabezado, pie de página o archivo de menú estándar para todas sus páginas web. Luego, cuando sea necesario actualizar el encabezado, solo puede actualizar el archivo de inclusión del encabezado.

include '*filename*';  
  
or  
  
require '*filename*';

footer.php:

<?php  
echo "<p>Copyright &copy; 1999-" . date("Y") . " W3Schools.com</p>";  
?>

<html>  
<body>  
  
<h1>Welcome to my home page!</h1>  
<p>Some text.</p>  
<p>Some more text.</p>  
<?php include 'footer.php';?>  
  
</body>  
</html>

Menu.php:

<?php  
echo '<a href="/default.asp">Home</a> -  
<a href="/html/default.asp">HTML Tutorial</a> -  
<a href="/css/default.asp">CSS Tutorial</a> -  
<a href="/js/default.asp">JavaScript Tutorial</a> -  
<a href="default.asp">PHP Tutorial</a>';  
?>

<html>  
<body>  
  
<div class="menu">  
<?php include 'menu.php';?>  
</div>  
  
<h1>Welcome to my home page!</h1>  
<p>Some text.</p>  
<p>Some more text.</p>  
  
</body>  
</html>

### PHP incluye vs.requiere:

La declaración require también se usa para incluir un archivo en el código PHP.

Sin embargo, hay una gran diferencia entre incluir y requerir; cuando se incluye un archivo con la declaración include y PHP no puede encontrarlo, el script continuará ejecutándose.

Úselo cuando require la aplicación requiera el archivo.

Úselo cuando include el archivo no sea necesario y la aplicación debería continuar cuando no se encuentre el archivo.

## Manejo de archivos PHP

La función readfile() lee un archivo y lo escribe en el búfer de salida.

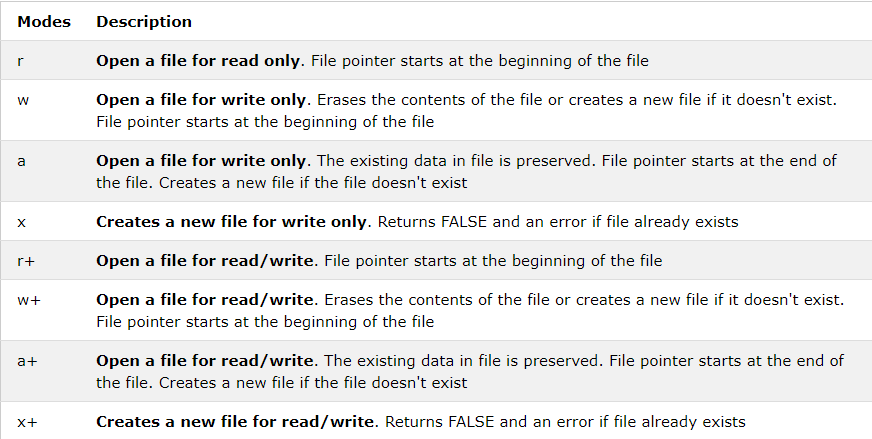
<?php  
echo readfile("webdictionary.txt");  
?>

La función readfile()es útil si todo lo que desea hacer es abrir un archivo y leer su contenido.

Un mejor método para abrir archivos es con la función fopen(). Esta función le ofrece más opciones que la función readfile().

El primer parámetro de fopen() contiene el nombre del archivo que se abrirá y el segundo parámetro especifica en qué modo se debe abrir el archivo. El siguiente ejemplo también genera un mensaje si la función fopen () no puede abrir el archivo especificado:

<?php  
$myfile = fopen("webdictionary.txt", "r") or die("Unable to open file!");  
echo fread($myfile,filesize("webdictionary.txt"));  
fclose($myfile);  
?>



La función fread()lee de un archivo abierto.

El primer parámetro de fread() contiene el nombre del archivo para leer y el segundo parámetro especifica el número máximo de bytes para leer.

El siguiente código PHP lee el archivo "webdictionary.txt" hasta el final:

fread($myfile,filesize("webdictionary.txt"));

La función fclose()se utiliza para cerrar un archivo abierto.

Es una buena práctica de programación cerrar todos los archivos una vez que haya terminado con ellos. ¡No desea que un archivo abierto se ejecute en su servidor ocupando recursos!

El fclose() requiere el nombre del archivo (o una variable que contiene el nombre del archivo) que desea cerrar:

<?php  
$myfile = fopen("webdictionary.txt", "r");  
// some code to be executed....  
fclose($myfile);  
?>

La función fgets()se usa para leer una sola línea de un archivo.

<?php  
$myfile = fopen("webdictionary.txt", "r") or die("Unable to open file!");  
echo fgets($myfile);  
fclose($myfile);  
?>

Nota: Después de llamar a la fgets()función, el puntero del archivo se ha movido a la siguiente línea.

La función feof() comprueba si se ha alcanzado el "fin de archivo" (EOF).

La funciób feof() es útil para recorrer datos de longitud desconocida.

El siguiente ejemplo lee el archivo "webdictionary.txt" línea por línea, hasta que se llega al final del archivo:

<?php  
$myfile = fopen("webdictionary.txt", "r") or die("Unable to open file!");  
// Output one line until end-of-file  
while(!feof($myfile)) {  
  echo fgets($myfile) . "<br>";  
}  
fclose($myfile);  
?>

La función fgetc() se utiliza para leer un solo carácter de un archivo.

El siguiente ejemplo lee el archivo "webdictionary.txt" carácter por carácter, hasta que se llega al final del archivo:

<?php  
$myfile = fopen("webdictionary.txt", "r") or die("Unable to open file!");  
// Output one character until end-of-file  
while(!feof($myfile)) {  
  echo fgetc($myfile);  
}  
fclose($myfile);  
?>