Tabla

Descripción generada automáticamenteDIPLOMATURA PROGRAMACIÓN WEB - FULL STACK – UTN

Clase 1 (12/06/2024)

* Instalación de VSC
* HTML 🡪 usa etiquetas

🡪 en el nombre del archivo, sin mayúscula ni espacios

🡪 se usan para dar estructura a una página web

🡪 lenguaje estructurado

🡪 apertura: < head >

🡪 cierre: </ head >

🡪 auto cierre: < img / > // html5: < img > ; meta

🡪 comentarios: <! – hola --> // alt + shift + A // ctrl + K + C

🡪 anidado o nesting: una etiqueta dentro de otra (ejemplo divs)

🡪 div: divisor

🡪 indentación o indentation

🡪 etiquetas de heading (h1 a h6)

🡪 mdn html, la documentación oficial [HTML: Lenguaje de etiquetas de hipertexto | MDN (mozilla.org)](https://developer.mozilla.org/es/docs/Web/HTML)

FORMATO (de siempre)

<html>

<head></head> <!-- lista de configuración de la pagina -->

<meta charset="UTF-8"> <!-- para acentos, configuración del idioma, caracteres, así el navegador puede interpretarlos -->

<title>Página Web</title>

<body></body>

</html>

Clase 2 (19/06/2024)

GIT 🡪 Controlador de versiones (MUY IMPORTANTE) / Sistema de versionados

Facilita poder volver a una versión **estable**

Es Open Source

Nueva versión: “commit”

Interactuamos a través de comandos en una consola o terminal

CLI 🡪 command line interface

En la terminal, git init 🡪 inicia un repositorio. Se crea dentro de la carpeta seleccionada, en una carpeta llamada .git

Initialized empty Git repository in C:/Users/chiar/OneDrive/Escritorio/Diplomatura UTN/Clase 2/.git/

Archivos trackeados (A) y no trackeados (U). ‘git add .’

Modificado (M). Deja de estar trackeado

git commit -m 'primera versión' 🡪 se creó una versión. Después puedo regresar. Se fueron los colores y letra

git log 🡪 ver los commits ya creados

git branch -M main 🡪 cambiar a la rama principal

Clase 3 (26/06/2024)

HTML 🡪 Generación automática de la parte inicial del archivo: ! + Enter / ‘html:5’

🡪 Se le llama element a las etiquetas.

🡪 Ctrl + Shift + presiona la tecla "P": Word wrap

🡪 h1, h2 solo determinan jerarquías para el navegador, NO ESTILOS

🡪 técnicas de SEO: search engine optimization

🡪 div: divider. Divide el contenido y lo separa del resto, contiene. ESTRUCTURAL, no estético

🡪 span: para pequeñas decoraciones. Es un texto para cosas chiquitas (precios, fechas)

🡪 button: etiqueta para botones.

🡪 p: párrafo

🡪 lorem + enter añade el texto en latín

🡪 ANCLA

a: etiqueta para redireccionar. <a href=”hipervículo\_o\_ruta”> Ir a Página </a>

target=”\_blank” abre en una pestaña nueva

Archivos multimedia

🡪 img src source, fuente. Alt es para el texto alternativo, personas no videntes

🡪 audio, de apertura y cierre. Src y controls para que suene; controls loop para que suene en loop; controls loop autoplay comienza a sonar solo.

🡪 video src controls

Clase 4 (01/07/2024)

HTML 🡪 SIEMPRE al final están las etiquetas de cierre de body y html

🡪 mover un bloque de text en vsc: seleccionarlo, alt, flechitas

🡪 etiqueta iframe: permite colocar una página dentro de otra página

🡪 por defecto, la unidad es pixel px

🡪 frameborder no está recomendada; usar style

🡪 etiqueta ol: order list – lista ordenada

🡪 etiqueta li: list item

🡪 etiqueta ul: unorder list – lista desordenada

Navegador simple - Navegador complejo: se combina con listas, links y sublinks

🡪 etiqueta nav: navegador con muchos div; lista de elementos navegables. Cuestión semántica de orden

🡪 etiqueta aside: colocar hacia izq o der

Seobility, hace una devolución y puntúa nuestra página. Herramienta para optimizar tu web gratis

Buscador: input

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamenteInterfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

🡪 etiqueta form: indico la creación de un formulario

🡪 etiqueta label: texto que acompaña al input, indica qué se debe escribir en el input

REGLA DE ACCESIBILIDAD - Debe tener un atributo for (con quién conecta) para enlazarlo al id del input:

<label for=”fullname”> Nombre completo:</label>

<input id=”fullname”>

El atributo type incluye las validaciones. Algunos types:

<input id=”email” type=’’email’’>

type=’’password’’ aparecen los puntitos para ocultar el texto, y el ojo para verlo

type=’’file’’ boton para seleccionar archivo

type=”date” 🡪 DD/MM/YYYY

type=”time” 🡪 HH/MM

type=”datetime-local” 🡪 DD/MM/YYYY, HH/MM

type=”tel” 🡪 abre el teclado numérico (para disp. móviles)

Botón para enviar formularios: <input type="submit">

Puedo agregarle el atributo value para cambiar el texto del botón (no se traducirá automáticamente)

No lleva label

Clase 5 (03/07/2024)

🡪 etiqueta span: texto decorativo corto sin funcionalidad

🡪 etiquetas article, aside, main, header, footer: funcionan como divs, “cajas”

En Formularios:

* Por ahora, name = id. Sirve en JS
* Required: campo obligatorio
* minlength, maxlength
* placeholder: texto gris, de fondo, en el recuadro input. Valor de prueba, ejemplo

TYPE RADIO BUTTON

<div>

                <label for="html">HTML</label>

                <input type="radio" name="lenguaje-preferido" id="html">

                <label for="css">CSS</label>

                <input type="radio" name="lenguaje-preferido" id="css">

                <label for="js">JS</label>

                <input type="radio" name="lenguaje-preferido" id="js">

            </div>

Doy el mismo valor del atributo name a todos para que se pueda seleccionar **UNO** (radio button). Una opción entre varias.

TYPE CHECK BUTTON

            <div>

                <div>

                    <label for="sql">SQL</label>

                    <input type="checkbox" name="tecnologia-conocida" id="sql">

                </div>

                <div>

                    <label for="git">Git</label>

                    <input type="checkbox" name="tecnologia-conocida" id="git">

                </div>

                <div>

                    <label for="python">Python</label>

                    <input type="checkbox" name="tecnologia-conocida" id="python">

                </div>

            </div>

Con este, podemos seleccionar más de una opción.

ETIQUETA SELECT

        <div>

            <label for="provincia">Provincia:</label>

            <select name="provincia-elegida" id="provincia">

                <option value="bsas">Buenos Aires</option>

                <option value="sal">Salta</option>

                <option value="cba">Córdoba</option>

                <option value="mza">Mendoza</option>

                <option disabled selected value="null">No seleccionado</option>

            </select>

        </div>

Lista desplegable. Algunos atributos:

Disabled: no va a dejar seleccionar esta opción

Selected: aparece seleccionado por defecto

**CSS**

Estilo en línea: incómodo, poco escalable, mala práctica, no reutilizable. Hay excepciones. Mucha especificidad.

Otra opción: etiqueta style

 <style>

        /\* Cuando uso esta etiqueta, le indico al navegador que esta parte debe interpretarla como CSS\*/

    </style>

CSS es un lenguaje con selectores.

        h2{

            color: rgb(255, 173, 173);

            background-color: black;

            font-size: 55px;

        }

Para “excepciones”, usamos el atributo class. Para darle estilos a la clase: .nombre-clase

        /\*selector de clase\*/

        .titulo-especial{

            color: rgb(183, 146, 102);

        }

    <h2>Soy un h2</h2>

    <h2 class="titulo-especial">Soy un h2 único</h2>

    <h2>Yo también soy un h2</h2>

Este lenguaje trabaja en cascada; si dos selectores tienen el mismo nivel de jerarquía (de especificidad), la característica repetida que permanecerá será la última.

Clase 6

* Etiqueta <br> mala práctica. Se debe usar css.
* Pirámide de la especificidad
* !important; se pone en la línea del atributo.
* Estilo en línea.
* Selector de id: seleccionamos al elemento por su id. Se escribe con numeral #nombre-id { }

SELECTORES A USAR, BUENAS PRÁCTICAS:

* Selector de clases: .nombre-clase { }
* Selector de elementos.
* Selector universal: hace referencia a **todos** los elementos de la página. Se hace con un asterisco \* { }. Es el de menos valor.

Diagrama

Descripción generada automáticamente

Algunas propiedades de css:

* text-decoration: px - solid, dashed, wavy, dotted, double - underline, line-througth, overline – none;
* border-radius: px;
* background-color: #;
* color: #;
* font-family: ;
* font-size: px;
* letter-spacing: px;
* word-spacing: px;

Clase 7

Se crea un archivo .css para mayor escalabilidad y reutilización de estilos. Se lo comparte.

Para enlazarlo con el .html, en el head:

<link rel="stylesheet" href="./style.css">

Algunas reglas y carpetas útiles:

* Index.html tiene que ser hijo de la carpeta raíz
* Carpeta styles para todos los .css
* Carpeta pages para todas los .html excepto index
* Carpeta assets para imágenes
* href=”../” SALGO DE LA CARPETA ACTUAL

Elementos en bloque: h1 a h6, p, div, ul, ol, li, main, nav, body, footer, header, article, aside, section, form. **SÍ** se puede modificar su width, height, margin, padding

Elementos en línea: se sitúan a continuación. span, a, label, b, strong, i (italic), em. **NO** se puede modificar su width, height. **SÍ** border, padding horizontal (no reco), margin horizontal.

Imagen que contiene Diagrama

Descripción generada automáticamenteElementos en línea-bloque: img, input, iframe, audio, video, button, textarea, select, aquellos en línea que **SÍ** permiten modificación del width, height, margin, border y padding.

CONVERTIR UN ELEM EN LÍNEA A OTRO TIPO:

display: inline-block;

Propiedad width en css: **ancho** del elemento px;

Propiedad height en css: **alto** del elemento px;

Diagrama

Descripción generada automáticamenteImagen que contiene Forma

Descripción generada automáticamenteCaja

box-sizing: border-box; /\* ahora, se contabiliza desde el borde \*/

sombra

box-shadow:0px 0px 30px black ; /\*horizontal vertical difuminado color\*/

Generadores de css:

<https://neumorphism.io/#e0e0e0>

Imagen que contiene Forma

Descripción generada automáticamente<https://cssgenerator.org/box-shadow-css-generator.html>

para centrar textos: text-align: center;

para centrar imágenes: margin: 0 auto;

!!! box-sizing: border-box; queremos que se respeten las medidas reales

En el universal \*, margin: 0;

padding: 0;

con eso, eliminamos los espacios que vienen por defecto

Clase 8 (15/07/2024)

div.padre>div.hijo\*3 🡪

<div class="padre">

        <div class="hijo"></div>

        <div class="hijo"></div>

        <div class="hijo"></div>

</div>

Normalización: “reseteamos” los estilos predeterminados

\*{

    margin: 0;

    padding: 0;

    box-sizing: border-box;

}

Height: 100**vh**; 🡪 viewport height, medida relativa

.hijo:nth-child(1){

    background-color: rgb(61, 234, 124);

}

* pseudo element, busca todos los hijos y aplica el estilo específico al primero (en este caso).

Herramienta flexbox de posicionamiento: es una tecnología unidimensional. Se trabaja siempre sobre el mismo eje.

* Requiere de un padre para funcionar. En este SI O SI debemos colocar display: flex; esta propiedad activa flexbox
* Trabaja con configuraciones por defecto. Ej: flex-direction: row; (eje x)
* Gap nos permite espaciar entre los hijos. gap: 20px;
* Con justify-content justificamos los elementos en el eje x (por defecto). Si flex-direction: column; (eje y) se rotan los ejes y ahora justify-content nos deja mover los elementos en el eje y.
  + Justify-content: center; autocentrado
  + Justify-content: start; por defecto, al principio del flex
  + Justify-content: flex-end; al máximo a la derecha
* Align-items nos permite alinear los elementos respecto del eje y (f-d row) o respecto del eje x (f-d column).

section {

display: flex;

flex-direction: row;

justify-content: center;

}

justify-content: space-between; 🡪 los espacia lo máximo posible.

line-height: 24px; 🡪 interlineado

En inspeccionar, el tamaño mostrado es ANCHO X ALTO

ÍCONOS

<https://icons.getbootstrap.com/>

Importamos un css con los íconos EN el archivo css:

@import url("https://cdn.jsdelivr.net/npm/bootstrap-icons@1.11.3/font/bootstrap-icons.min.css");

Luego elegimos el ícono, copiamos el html y lo agregamos en nuestro código html con la clase correcta

FUENTES

<https://fonts.google.com/>

elegimos la fuente, Get font, Get embed code, copiamos el import y lo agregamos a nuestro archivo css.

Podemos agregar el Font-family con la fuente en el selector universal \*. Debemos agregarlo en donde sea necesario.

Las páginas no suelen tener más de 2 fuentes.

justify-content defines how the browser distributes space between and around items horizontally (also called main-axis). It has many values but only 6 are widely used:

* center align items at the center of the container.
* .wrapper {
* display: flex;
* justify-content: center;

}

1

* flex-start (default) align items at the start of the container.
* .wrapper {
* display: flex;
* justify-content: flex-start;

}

1

* flex-end align items at the end of the container.
* .wrapper {
* display: flex;
* justify-content: flex-end;

}

1

* space-between align the first and last item will be flush with the ends of the container and all of the space shared equally between the items.
* .wrapper {
* display: flex;
* justify-content: space-between;

}

1

2

* space-around align items evenly. Items have a half-size space on either end.
* .wrapper {
* display: flex;
* justify-content: space-around;

}

1

2

* space-evenly align items evenly. Items have equal space around them.
* .wrapper {
* display: flex;
* justify-content: space-evenly;

}

1

2

align-items defines how the browser distributes space between and around items vertically. Think of it as justify-content but for the vertical axis. Property values:

* center align items at the center of the container.
* .wrapper {
* display: flex;
* align-items: center;

}

1

* flex-start align items at the start of the container.
* .wrapper {
* display: flex;
* align-items: flex-start;

}

1

* flex-end align items at the end of the container.
* .wrapper {
* display: flex;
* align-items: flex-end;

}

1

* baseline items are aligned such as their baselines align.
* .wrapper {
* display: flex;
* align-items: baseline;

}

1

2

3

4

5

* stretch items are stretched to fill the container.
* .wrapper {
* display: flex;
* align-items: stretch;

}

1

2

Reminder: align-items defines how the browser distributes space between and around items vertically and justify-content defines how the browser distributes space between and around items horizontally.

The flex-direction property specifies the direction of flexible items. Property values:

* row (default) flexible items are displayed horizontally, as a row.
* .wrapper {
* display: flex;
* flex-direction: row;

}

1

2

* row-reverse same as row, but in reverse order.
* .wrapper {
* display: flex;
* flex-direction: row-reverse;

}

1

2

* column items are displayed vertically, as a column.
* .wrapper {
* display: flex;
* flex-direction: column;

}

1

2

* column-reverse items are displayed vertically, as a column.
* .wrapper {
* display: flex;
* flex-direction: column-reverse;

}

1

2

Note: when you set the direction to a reversed row or column, the start and end are also reversed.

Note: when flex direction is a column, justify-content changes to the vertical and align-items changes to horizontal.

The order property specifies the order of a flexible item relative to the rest of the flexible items inside the same container. By default, items have a value of 0. But we can set any positive or negative integer value (-2, -1, 0, 1, 2).

Examples:

* without order property
* .wrapper {
* display: flex;
* }

#arthur

#marilyn

* using order property
* .wrapper {
* display: flex;
* }
* #arthur {
* order: 2;
* }
* #marilyn {
* order: 1;
* }

#arthur

#marilyn

The align-self property specifies the alignment for the selected item inside the flexible container. Property values (align-self in examples applied to box number 1):

* center element is positioned at the center of the container.
* .wrapper {
* display: flex;
* }
* #box1 {
* align-self: center;
* }

1

2

* flex-start element is positioned at the beginning of the container.
* .wrapper {
* display: flex;
* }
* #box1 {
* align-self: flex-start;
* }

1

2

* flex-end element is positioned at the end of the container.
* .wrapper {
* display: flex;
* }
* #box1 {
* align-self: flex-end;
* }

1

2

* stretch element is positioned to fit the container.
* .wrapper {
* display: flex;
* }
* #box1 {
* align-self: stretch;
* }

1

2

The flex-wrap property specifies whether the flexible items should wrap or not. Property values:

* nowrap (default) specifies that the flexible items will not wrap.
* .wrapper {
* display: flex;
* flex-wrap: nowrap;
* // or the same behavior
* // without flex-wrap property
* }

1

2

3

4

* wrap specifies that the flexible items will wrap if necessary.
* .wrapper {
* display: flex;
* flex-wrap: wrap;
* }

1

2

3

4

* wrap-reverse same as wrap, but reverse.
* .wrapper {
* display: flex;
* flex-wrap: wrap-reverse;
* }

1

2

3

4

The align-content property modifies the behavior of the flex-wrap property. It is similar to align-items, but instead of aligning flex items, it aligns flex lines. Property values:

* center lines stretch to take up the remaining space.
* .wrapper {
* display: flex;
* flex-wrap: wrap;
* align-content: center;
* }

1

2

3

4

* flex-start lines are packed toward the start of the flex container.
* .wrapper {
* display: flex;
* flex-wrap: wrap;
* align-content: flex-start;
* }

1

2

3

4

* flex-end lines are packed toward the end of the flex container.
* .wrapper {
* display: flex;
* flex-wrap: wrap;
* align-content: flex-end;
* }

1

2

3

4

* space-between lines are evenly distributed in the flex container.
* .wrapper {
* display: flex;
* flex-wrap: wrap;
* align-content: space-between;
* }

1

2

3

4

* space-around lines are evenly distributed in the flex container, with half-size spaces on either end.
* .wrapper {
* display: flex;
* flex-wrap: wrap;
* align-content: space-around;
* }

1

2

3

4

CLASE 9 (17/07/2024)

**justify-content: space-between;**

Forma, Rectángulo

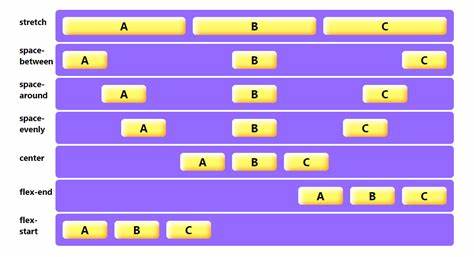
Descripción generada automáticamenteDiagrama

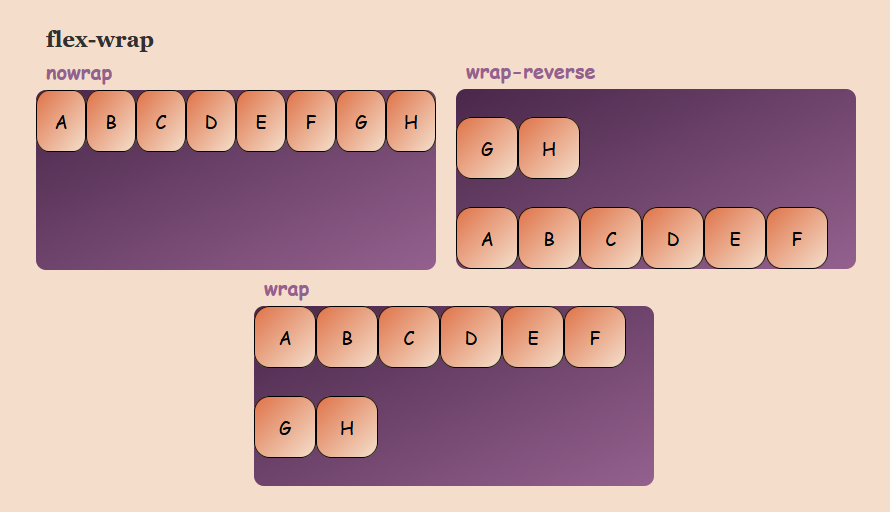
Descripción generada automáticamente

**justify-content: space-evenly;**

**Imagen que contiene Forma

Descripción generada automáticamentejustify-content: space-around;**



wrap

row-gap espacio entre filas, column-gap espacio entre columnas

Diagrama

Descripción generada automáticamenteflex fracción que usará el hijo, proporciones (el doble, el triple)

order establece un orden a los elementos.

Usar variables:

:root{

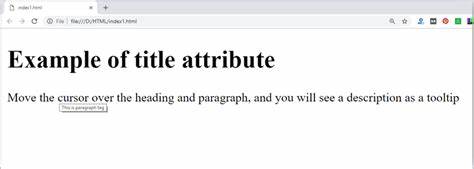
--nombre-vble: valor;

}

…

Atributo: var(--nombre-vble)

Nos sirven para reutilizar valores que pueden ser complejos.

En html, si usamos la propiedad title, aparece el texto descriptivo por encima. Se llama Tooltip

CLASE 10 (22/07/2024)

**Position**

Por defecto: position: static;

position: relative; 🡪 activa el posicionamiento (ahora tenemos acceso a 4 propiedades nuevas: top, bottom, left, right). Top vale más que bottom y left más que right. Reco: usar bottom o top; usar left o right. **No** usar ambas al mismo tiempo. una vertical y una horizontal.

top: px; 🡪 se colocan por encima x cantidad de pixeles. SE POSICIONA X CANTIDAD MÁS ABAJO

bottom: px; 🡪 SE POSICIONA X CANTIDAD MÁS ARRIBA. se le agregan pixeles abajo

puede usarse para superposicionarse.

usar right y bottom lo menos posible; es mejor usar valores negativos en top y left.

relativo hace referencia a que dependerá de su posicionamiento inicial.

z-index 🡪 posición de los elementos en el eje z. qué tan adelante va a estar. no permite valores negativos distintos de -1. Con este valor se va a ir atrás del body y de todo. no muy impo el -1

relative vale más que static

position: absolute; 🡪 queda tapado por los demás hijos. rebelde. busca al padre con position distinto de static; si no encuentra, se sale del padre y se hace hijo de toda la página.

position: fixed; 🡪 permanece quieto. ejemplo navegadores, buscadores. es como pegar algo, dejarlo fijo en un lugar específico

position: sticky; 🡪

position: ; 🡪

CLASE 11 (24/07/2024)

**Media Queries**

Son condiciones; si … entonces …

Hay muchos casos; uno es el del header:

🡪 “cuando el ancho tenga 320px o menos entonces el background color cambia a blueviolet”

@media screen and (max-width: 320px){

.padre{ 🡪 este bloque se ejecuta solamente si la condición es Verdadera

backgruound-color: blueviolet;

}

}

display: none; 🡪 oculta un elemento

svg: formato de imagen vectorial; no pierde calidad; pesa poco

recomendación: particionar css. Crear una carpeta “styles” y hacemos varios archivos .css, después los importamos:

@import url('./normalize.css');

Mínimo de pantallas en los celulares: 320px

Overflow 🡪 un hijo sobrepasa las capacidades del padre

Overflow: hidden;

Max-width, max-height. Idem min

CLASE 12 (29/07/2024)

Introducción a JAVASCRIPT

* Nos permite dar interacción a la página web, funcionalidad
* Conexión con apis
* Funcionamiento de botones
* Etiqueta <script> </script> para contener todo lo que el navegador indentificará como JS
* Comentarios: multilínea /\*\*/ - una sola línea //
* Archivo script.js
* Importo el js al html con: <script src=’’./script.js’’> </script> ANTES DE CERRAR EL BODY.
* Otra forma: <script src=’’./script.js’’ defer> </script> EN HEAD
* Otra forma: <script src=’’./script.js’’ async> </script> EN HEAD
* Tipos de datos primitivos:
  + strings
  + numbers, infinity, NaN not a number
  + booleans
  + null
  + undefined
* typeof(). Con null no funciona
* Constructores:
  + Number(valor)
  + String(valor)
  + Boolean(valor)

CLASE 13 (31/07/2024)

JS

No necesita ; al final de las sentencias

Valores truthy y falsy. Los no falsos, son verdaderos. Estos son falsos: null, undefined, NaN, 0, false, ‘’’’. Podemos verificarlos con el contructor Boolean(valor)

Boolean(NaN) es false

Operadores aritméticos

+ concatenación, unión entre dos cadenas. Relacionado con los strings, ocurre cuando hay uno o más strings en la operación. 1 + ‘1’ = ‘11’. Transforma el dato no string a string y los une. Devuelve siempre un string.

+ suma, ocurre cuando no hay strings en la operación. Todo lo transforma a number (null=0, true=1, false=0). Devuelve siempre un number.

- resta, ocurre cuando no hay strings en la operación. Todo lo transforma a number. Devuelve siempre un number.

\* multiplicación, ocurre cuando no hay strings en la operación. Todo lo transforma a number. Devuelve siempre un number.

/ división, ocurre cuando no hay strings en la operación. Todo lo transforma a number. Devuelve siempre un number.

% módulo, ocurre cuando no hay strings en la operación. Todo lo transforma a number. Devuelve siempre un number.

Cualquier valor operado con NaN (EXCEPTO LA CONCATENACIÓN), la operación devolverá Nan.

Comparadores

== igualdad, excepción: NaN == NaN SIEMPRE devuelve false.

=== estricta igualdad, recomendado. Compara valor y tipo de dato

!= distinto,devuelve boolean

<, >, <=, >=, menor a, mayor a, menor o igual a, mayor o igual a

* NaN siempre es distinto de cualquier valor, incluso él mismo
* Null solo puede ser igual a null o undefined

Operadores lógicos

! not, devuelve el valor booleano opuesto

x || y or, verifica si el primer valor es true o false. Si es verdadero, devuelve el primer valor;

si es falso, devuelve el segundo valor.

x && y and, verifica si el primer valor es true o false.

Si es verdadero, devuelve el segundo valor;

si es falso, devuelve el primer valor.

Variables

* var
* var nombre = ‘valor’
* console.log(lo que queremos registrar en la consola)
* cuando una vble no tiene valor definido, devolverá undefined
* se puede reasignar, declarar sin valor inicial (será un undefined)
* se puede redeclarar
* tiene hoisting ( JS tiene la capacidad de ‘mover’ las cosas), puede ser accedida antes de su declaración o inicialización. Coloca todas las declaraciones hacia arriba, al inicio.
* Tiene ámbitos
* let
* let nombre = ‘valor’
* se puede reasignar, declarar sin valor inicial (será un undefined)
* NO es posible redeclarar la misma variable en el mismo bloque (crearla dos veces); puede ser redeclarada en diferentes ámbitos.
* No puede ser accedida antes de su declaración o inicialización. NO TIENE HOISTING
* Tiene ámbitos
* const
  + const nombre = ‘valor’
  + son constantes
  + NO se puede reasignar, NO se puede declarar sin valor inicial (será un undefined)
  + NO es posible redeclarar la misma variable en el mismo bloque (crearla dos veces)
  + No puede ser accedida antes de su declaración o inicialización. NO TIENE HOISTING
  + Tiene ámbitos

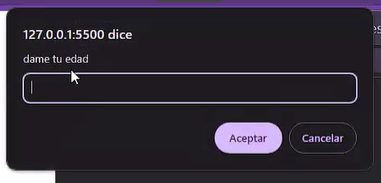
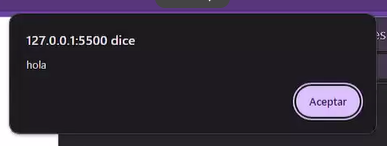
Condiciones

Nos permiten modificar el flujo lineal

Funciones nativas: alert() 🡪 envía una alerta

(ya las trae el lenguaje) prompt() 🡪 solicita un dato. Siempre retorna string (si hacemos click en Aceptar) o null (si hacemos click en Cancelar)

alert(‘hola’) prompt(‘dame tu edad’)



? significa opcional

En las funciones, retorno void significa que no devolverá ningún tipo de dato (en realidad sí, un undefined)

CLASE 14 (05/08/2024)

JS

Es un lenguaje keysensitive

Función nativa: isNaN() le paso un dato y la funcion intentara pasarlo a numero, si el resultado es NaN entonces isNaN retorna true sino retorna false

CLASE 15 (07/08/2024)

let tiene ámbitos y puede ser redeclarada en diferentes ámbitos

Ámbitos: global y local, se limitan los alcances

Bloques de código: líneas de código entre dos llaves {}

Sólo const y let tienen ámbitos, var no tiene (desventaja)

Ciclo for: for(desde; hasta/condición; incremento)

Funciones: bloques de códigos almacenados en la memoria

Palabra reservada function: function nombre\_funcion() {}

Deben representar acciones reutilizables

CLASE 16 (12/08/2024)

Principio de la programación “DRY”: Don’t repeat yourself

Arrays:

También llamados vectores, listas, arreglos, colecciones

listan elementos

Indice/index: la posicion de un elemento en array

0 1 2

const nombres = ['pepe', 'juan', 'maria']

Si quiero llamar a maria deberia conocer su posicion

console.log(nombres[2])

Array multinivel: array de arrays, matriz

i 🡪 iterator, index

escalabilidad

propiedad length [gallina tiene huevos]: cantidad de elementos del array, longitud, comienza en 1

a las propiedades se acceden con punto: array.length

JSDoc, documentación – documentacion.md

Bucle FOR OF:

for(let element of array){

accion

}

for (let nombre of nombres){

console.log(‘Nombre: ’ + nombre}

Buenas prácticas: arrays con nombres en plural (hay excepciones)

la vble creada para el for of, en singular o ‘elemento’

funciones nombradas como verbos

Métodos de arrays: acciones que se implementan para arrays

array.push(element) 🡪 **agregar** elemento al **final** del array

array.pop() 🡪 **eliminar** elemento al **final** del array

array.unshift(element) 🡪 **agregar** elemento al **PRINCIPIO** del array

array.shift() 🡪 **eliminar** elemento al **principio** del array

array.includes(element) 🡪 saber si determinado elemento forma parte del array; devuelve un booleano

Una función sin retorno devolverá undefined por defecto

CLASE 17 (14/08/2024)

Otros métodos de arrays:

array.indexOf(‘element’) 🡪 primera posición del elemento

array.splice() 🡪 1. Start (desde qué posición quiero eliminar), 2. Delete count (cantidad a eliminar) 3. Nuevos elementos que agregaremos. Como empalmar un cable

MALA PRACTICA:

delete nombres[0]

console.log(nombres)

Los **objetos** nos permiten almacenar datos complejos

* Trabajan con entidades compuestas por propiedades (nombre, apellido, edad)

Objeto literal (no es un bloque de código entre llaves)

const persona = {

'nombre': 'pepe',

'apellido': 'suarez',

'edad': 30,

'nro telefono': '12313213',

'amigo': {

'nombre': 'maria',

'edad': 35,

'numeros preferidos': [3, 7, 4]

}

}

objeto.propiedad

¡!

arrays, objetos, funciones = CONST

string, null, number, boolean = LET

Variables y referencias

Trabajar los objetos con const facilita que este NO pueda ser roto y transformado en algo distinto de objeto. Protege al dato. La referencia es constante

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Acceder al valor de la propiedad del objeto:

NOTACION DE CORCHETES

console.log(persona['nombre'])

console.log(persona['nro telefono']) !! se pueden usar espacios

console.log(persona['amigo']['nombre'])

console.log(persona['amigo']['numeros preferidos'][1])

Tabla

Descripción generada automáticamente con confianza mediaPodemos ver los objetos como tablas:

Las propiedades SIEMPRE son strings

Pueden ser sin comillas al menos que tenga un espacio

NOTACION DE PUNTOS

console.log(persona.nombre)

console.log(persona.nroTelefono)

console.log(persona.amigo.nombre)

console.log(persona.amigo["numeros preferidos"][1])

usar corchetes o puntos? Depende de la situación. Reco: usar siempre de puntos a menos que necesite corchetes (excepciones como cambiar algún valor)

FOR IN

for(let propiedad in objeto){

console.log('el valor de la propiedad: ' + propiedad + ' es ' + objeto[propiedad])

}

const formularioLogin = {

'username': {

tipo: 'text',

placeholder: 'ingresa un nombre',

label: 'Ingresa el nombre:'

},

'email': {

tipo: 'email',

placeholder: 'Ingresa tu email',

label: 'Ingresa el email:'

},

'password': {

tipo: 'password',

placeholder: 'ingresa la contraseña',

label: 'Ingresa la contraseña:'

}

}

let formulario = ''

for( let propiedad in formularioLogin ){

//Template string

formulario = formulario + `

<label class='label'>${formularioLogin[propiedad].label}</label>

<input type="${formularioLogin[propiedad].tipo}" placeholder="${formularioLogin[propiedad].placeholder}">

`

}

Template string permite saltos de línea

ARRAY DE OBJETOS

const productos = [

{

nombre: 'tv samsung',

id: 1,

descripcion: 'lorem',

precio: 10

},

{

nombre: 'tv LG',

id: 2,

descripcion: 'lorem',

precio: 15

},

{

nombre: 'tv noblex',

id: 3,

descripcion: 'lorem',

precio: 7

}

]

console.log(productos[1]) 🡪 uso la posición del elemento al que quiero acceder

for(const producto of productos ){

console.log(`El producto con id ${producto.id} se llama ${producto.nombre}`)

}

const porque estoy trabajando con OBJETOS

Template strings: (`La variable es ${nom\_vble}.`)

Interpolación (entre strins, necesito escribir JS)

CLASE 18 (19-08-2024)

Objeto:

const objeto = {

propiedad1: ‘valor1’,

’ propiedad 2’: valor2,

propiedad3: ‘valor3’

}

console.log(`El valor de la propiedad 1 es ${objeto.propiedad1 } y la segunda es ${objeto[‘propiedad 2’]}.`)

console.log(Object.keys(cliente)) 🡪 trae las propiedades del objeto

console.log(Object.values(cliente)) 🡪 trae los valores del objeto

RECORRER UN ARRAY DE OBJETOS:

for(const usuario of usuarios\_conectados){

console.log(`${usuario.nombre} está conectado`)

}

Métodos de strings (y algunos compartidos con arrays)

Se pueden concatenar métodos (siempre y cuando tengan sentido los retornos)

Ej. let resultado = mensaje.toLowerCase().includes(palabraBuscada.toLowerCase())

* string.includes(valor a buscar) (retorna booleano)
* string.toUpperCase(), toLowerCase() 🡪 nueva cadena, no modifica al original
* string.replace(qué reemplazo, nuevo valor) 🡪 solo reemplaza la primera aparición
* string.replaceAll(qué reemplazo, nuevo valor) 🡪 reemplaza TODAS las apariciones
* string.trim() 🡪 elimina los espacios en blanco que están al inicio del string
* string.repeat(cantidad de veces) 🡪 multiplica el string
* string.charAt(posición) 🡪 devuelve el caracter de la posición solicitada. Idem corchete
* string.split(string especificado) 🡪 transforma string a array. Cuando encuentre el string especificado, termina el elemento. Ej
  + let mensajeArray = mensaje.split(' ') va a devolver un array con todas las palabras separadas porque le pedí que haga un split en cada espacio
  + Puede servir para obtener la cantidad de palabras en un párrafo

Arrow function, formato en flecha para las funciones:

Se declara como una variable const y los valores son los parámetros

|  |  |
| --- | --- |
| function verificarNumero (numero){  return !isNaN(numero)  } | const verificarNumero = (numero) => {  return !isNaN(numero)  } |

Ventajas de usar arrow function:

No podemos declarar funciones con el mismo nombre en el mismo bloque

No funciona si la llamo antes de su inicialización (hosting 🡪 MALA PRÁCTICA)

CLASE 19 (21-08-2024)

‘\n’ 🡪 saltos de línea

type = ‘module’ en nuestro script, permite que pueda hacer exportaciones

continue, fuerza y pasa a la siguiente iteración

break, termina el bucle

CALLBACK 🡪 Funciones recibidas como parámetros de otra función ej. validaciones

Se pasa como referencia Y NO COMO UNA INVOCACIÓN, sin paréntesis

Parámetro mensaje

**Funciones nombradas y funciones anónimas**

Para usar una sola vez y si no tenemos la necesidad de guardarla en memoria Se crea la callback directamente en la invocación de la otra función

La flecha funciona como return

console.log(filtrar(productos, (producto) => producto.nombre.includes('TV')))

console.log(filtrar(productos, (producto) => producto.precio > 10000))

console.log(filtrar(personas, (persona) => persona.edad > 32))

const filtrar = (array, condicion) => {

    const resultado = []

    for(const elemento of array){

        if(condicion(elemento)){

            resultado.push(elemento)

        }

    }

    return resultado

}

**Métodos nativos avanzados de arrays 🡪 reciben callbacks**

**filter**

La función filtar ya existe en JS 🡪 productos.filter(condición o callback anónima)

Esta itera el array buscando elementos que cumplan la callback

Siempre retorna un array compuesto por los elementos que hayan cumplido con la callback

**forEach**

recibe una callback y **POR CADA** elemento ejecuta la callback

NO RETORNA NADA (Undefined)

Similar al FOR OF

**find**

recibe una callback y por cada elemento verificará si al pasarlo por la callback devuelve verdadero

cuando devuelva verdadero la callback, find retornara ese elemento

Solo retorna un valor, el primero que encuentra y cumple

SI NADIE CUMPLE CON LA CONDICION ENTONCES RETORNA UNDEFINED

**findIndex**

recibe una callback y por cada elemento verificará si al pasarlo por la callback devuelve verdadero

cuando devuelva verdadero la callback, find retornara LA POSICIÓN del elemento

Solo retorna una posicion, el primero que encuentra y cumple

SI NADIE CUMPLE CON LA CONDICION ENTONCES RETORNA UNDEFINED

CLASE 20 (26-08-2024)

Callbacks

Al hacerlas, no se sabe cuándo serán invocadas

A veces se utiliza para la abstracción

**DOM**: Document Object Model – Modelo de Objetos de Documento (html)

* Cómo manipular el archivo HTML desde JS
* JS NO lee html. El DOM funciona como una representación del html como si fuese un objeto
* ‘Traduce’ el html a objeto
* Variable **document**: global que representa al DOM. Se crea cuando se conecta el script <script src="./script.js"></script>. Objeto.
* console.dir(document) es exacto para mostrar objetos
* document nos permite hacer las modificaciones
* se trabaja con métodos

getElementById recibe un id y devuelve un elemento HTML como objeto. Si no lo encuentra, devuelve null

const titulo = document.getElementById('titulo')

propiedad .innerText

ejemplo de modificación:

console.log(titulo.innerText)

titulo.innerText = 'pepe'

.innerHTML(‘texto’) 🡪 interpreta al string dado como HTML

Inicialmente, puede ser un empty string

Se utilizan estas modificaciones para NO caer en un html estático. JS permite lógica dinámica

Mucha escalabilidad, ‘plantilla’

**JSON** 🡪 JavaScript Object Notation. Es un array de objetos especial

Las comillas invertidas permiten hacer saltos de línea

Se les llama a las imágenes como thumbnail

La reasignación en document (con innerHTML por ejemplo) genera un problema de rendimiento. Se busca evitarla lo más posible

React se encarga de los problemas de rendimiento

querySelector recibe un string que selecciona un css o id:

Llamar a un elemento por ID

const cajaProductos = document.querySelector('#caja')

Llamar a un elemento por clase

const cajaUsuarios = document.querySelector('.usuarios')

getElementById consume menos que querySelector

SessionStorage & LocalStorage (diferencia de permanencia)

* Para hacer que las variables persistan y los valores de las claves no se pierdan
* Ej. localStorage.setItem('theme', 'blanco')
* Inspeccionar 🡪 Aplicación 🡪 Almacenamiento local 🡪 dirección de nuestra página
* Podemos verlo como una tabla. Modelo key value
* SOLO puede guardar strings

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamenteInterfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

1. Da valor a la clave
2. Devuelve el valor de la clave
3. Elimina un valor por su clave
4. Borra toda la tabla

Añadir una clase al body: body.classList.add('mode-' + theme)

CLASE 21 (28-08-2024)

Función MAP: la usamos cuando queremos obtener un **array** en base a otro array

RETORNA SIEMPRE ARRAY

EL LENGTH DE EL ARRAY RETORNADO SERA IGUAL AL DEL ORIGINAL

El valor que tomara cada elemento del nuevo array, será determinado por el retorno de la callback

por detrás tiene un for of

const nombresHTML = nombres.map(

(nombre) => {

return `<div>Nombre: ${nombre}</div>`

}

)

el valor de retorno de una callback es undefined

**Eventos**: parte interactiva de un usuario con la página

onclick es un atributo que sirve para asociar una función al evento click

en el html, asigno un id al elemento. Después en el script uso getElementById(id) para guardar el elemento en una variable y poder trabajar con sus atributos:

HTML

<button id="btn-2" >Click 2</button>

JS

const accion\_1 = () => {

alert('hola has dado click en un boton de la pagina')

}

const button2HTML = document.getElementById('btn-2')

button2HTML.onclick = accion\_1

Sin embargo no es la más usada

OPCIÓN MÁS USADA:

const button3HTML = document.getElementById('btn-3')

button3HTML.addEventListener('click', accion\_1) 🡪 ‘oyente’, espera a que alguien genere el evento

Obtengo el id del elemento. El primer parámetro es uno de los atributos del elemento

addEventListener(cuando esto suceda, hacer esto) callback, es una función pasada por parámetro, se va a ejecutar por detrás

página con listado y clasificación de eventos: [HTML Event Attributes (w3schools.com)](https://www.w3schools.com/tags/ref_eventattributes.asp)

renderizar es la acción de poder mostrar el valor nuevo, ‘actualizar el estado’. renderContador()

estado: valor que al ser modificado, debe ser si o si actualizado en la página

la función setter se encarga de hacer la actualización del html. Los estados están ligados a esto.

CLASE 22 (02-09-2024)

**React**

🡪 Node.js es un entorno de ejecución

🡪 JS se ejecuta SIEMPRE en un navegador porque este es quien lo interpreta

🡪 Node.js nos permite ejecutar js en la computadora local: podremos usar bases de datos

🡪 React es una herramienta de frontend. Para ejecutarlo, NECESITAMOS Node.js

* idem Vite

El proceso de transformación lo hace quien hostee (levante) la página ej vercel, hostinger

Node interpreta y lo devuelve como html, css y js

No suele escribirse React desde cero, por ej se usa Vite

Live Server no sabrá leer React. Vite será el encargado de levantar el código en una pag web (reemplazará a LS)