**DESARROLLO WEB EN ENTORNO CLIENTE**

Examenes 60% Teoria (tipo test), trozos de código incompletos o erróneos y alguna de desarrollo y práctica (desarrollar aplicación o arreglar una, codigo)

Prácticas 40%

JavaScript Tutorial w3schools

Ejercicios JavaScript w3resource

**PROGRAMACION ORIENTADA A OBJETOS**

Clase Estructura

Constructor Método que introduce datos en las características de las clase

Instancia Objeto con características específicas

Propiedades de POO

* Encapsulación
* Abstracción
* Herencia
* Polimorfismo

Niveles Lenguaje

1. Lenguaje Maquina
2. Lenguaje ensamblador
3. Medio-Bajo nivel: C, C++
4. Medio-Alto nivel: Java, Python, C#, etc.
5. Librerías (código ya creado): estructuras listas para el uso
6. Frameworks

IDE’s Entornos de Desarrollo Intregrado

VISUAL STUDIO CODE

Lenguajes de programación: C++, C#, Java

Lenguajes de Scripting: Python, Php, Shell

GIT Aplicación

GITHUB Plataforma de repositorios, para poder trabajar en un proyecto desde distintos sitios en diferentes franjas horarias.

GITLAB Plataformas de repositorios privados

GIT permite tener un control de versiones y compartir y coordinar el trabajo con compañeros de la empresa, sabiendo quien y cuando se ha hecho una modificación, se puede también revertir cambios.

Comandos básicos de Git

* git init (Inicio de repositorio)
* git add <file> (Paso de [file] del WD al SA)
* git status (Muestra los estados de los archivos)
* git commit (Paso inicial del SA al repositorio)
* git push (Subir repositorio remoto)
* git pull (Descarga los cambios realizados por otros)
* git clone (Copia del cloud al local de todo)
* git diff archivo1 archivo2 (diferencias entre archivos)
* .gitignore (ignorar archivos)

Fichero, meterle una línea con commit y ver diferencia entre 1 codigo y otro.

**APUNTES DIA 2**

Arquitectura cliente/servidor

Está en un 90% a día de hoy soportada por navegadores, que son los clientes mas ligeros que hay.

En ambos entornos hay una lógica

En el servidor

* De negocio
* De almacenamiento

En la lógica de cliente

* De negocio ()

El cliente se basa en la imagen (HTML y CSS) y una parte de lógica (JavaScript)

Componentes aplicación web

* Interfaz: funcionalidad visual
* Lógica de negocio: “Inteligencia” de la aplicación. Procesos u operaciones para proveer servicios

Lógica del servidor:

* Recibe datos enviados por el cliente mediante la lógica
* Lo procesa
* Almacena
* Recupera datos y devuelve al cliente

Lógica de cliente

* Se comunica con el servidor para intercambiar datos
* Muestra los datos mediante la interfaz
* Entiende los controles incluidos en la interfaz, es decir, que antes al pulsar botones, se tenía que enviar la info al servidor, ahora se hace todo desde el cliente

No todas las apps web tienen lógica del cliente y en el servidor

Puede haber algunas que no tengan lógica diferenciada (monolítica)

**Página web ->** Contiene básicamente HTML, CSS (no siempre) y JavaScript (no siempre). Desde el punto de vista de su generación por el servidor:

* Estática, no hay lógica en el servidor, solo el archivo. Cuando hay una petición el servidor solo envía la pagina
* Dinámica, plantilla que está en el servidor, generada con código de la lógica del lado del servidor y HTML. Cuando hay petición, el servidor ejecuta el código y envía a la página.

**Sitio web ->** Conjunto de páginas web estáticas o dinámicas alojadas en un servidor. Suelen ser informativas, no ofrecen funcionalidades, como mucho formularios o autenticación para acceder al contenido privado.

**Aplicación web ->** Es un sitio web con características propias de una aplicación, permite al usuario realizar tareas específicas.

* Aplicaciones de múltiples páginas, cada acceso a una sección implica laa carga completa del HTML, CSS, JS y del contenido. Hay refresco de pantalla. (GitHub)
* Apliaciones de una página (SPA), la lógica del cliente se descarga UNA SOLA VEZ, de tal forma que el acceso a otras secciones provoca la descarga de datos. (GMail). Cuando entramos a Gmail, vemos todo lo que necesitamos ya disponible, en el momento de clickar alguna funcionalidad nueva, no se recarga todo otra vez.

Ejercicio, que son?:

FACEBOOK -> SPA

TWITTER -> SPA

GOOGLE DRIVE -> SPA

INSTAGRAM -> SPA

AMAZON -> APP WEB MULTIPLES PAGINAS

CNN -> SITIO WEB

WWW -> Servicio de internet, que permite conectarse a paginas

Websocket -> son un sistema de comunicación entre el cliente y el servidor, y realizan un handshake para establecer una conexión entre ambos

Full-duplex -> Bidireccional, ambos se comunican a la vez

Half-duplex -> Es bidireccional pero no simultáneo, se necesita apagar un extremo para que el otro empiece a comunicarse. HTTP

URL -> Identificador único, dirección

CSS dinámico -> permite modificar css de forma dinámica

Web API -> Web donde tengo una información y unos puntos de acceso, desde donde puede entrar el cliente para recoger diferentes datos.

UX/UI -> Experiencia de usuario e interfaz de usuario

**Características de el entorno cliente**

* Es multiplataforma, se puede abrir desde cualquier SO (no siempre ya que los navegadores pueden acceder a características propias del SO)
* Compatibilidad entre navegadores(No siempre, ya que la alta actualización de los estándares[ECMA para JS], pueden dejar descolgados a algún navegador)
* Universalidad Accesibles directamente desde navegadores sin necesidad de instalación.
* Actualizaciones transparentes para el usuario

Modelo de 2 capas y 3 capas

Modelo de 2 capas

No se utiliza prácticamente ya porque es difícilmente escalable, ya que cada vez que se quiera actualizar se necesita crear un nuevo cliente.

Modelo de 3 capas

* Diseñada para superar limitaciones del modelo de 2 capas
* Una capa intermedia contiene la lógica del negocio
* Las aplicaciones web actuales se ajustan a este modelo

Dentro del modelo de 3 capas están:

* Capa de presentación
  + Recoge la info del usuario y la envía al servidor para que capa de negocio la procese
  + Procesa info
  + Recibe resultados de la capa de procesos
  + Genera la presentación y la muestra (HTML, CSS, JS)
* Capa de aplicación proceso o negocio
  + Recibe los datos y los procesa
* Capa de datos (base de datos)

Componentes de un navegador

INTERFAZ DE USUARIO, permite la interacción del usuario con los elementos visuales disponibles, algunos de sus componentes son:

* Barra de herramientas
* Barra de progreso
* Gestión de descargas
* Preferencias de usuario
* Preferencias de impresión

Hay navegadores enfocados a la rapidez de la respuesta, otros mas al contenido del recurso o incluso de la seguridad.

El navegador nos da la posibilidad de conectarnos, ya sea en entorno local o en red, a un conjunto de archivos en diferentes formatos.

La arquitectura de los navegadores es siempre la misma.

* INTERFAZ DE USUARIO
* MOTOR DE BUSQUEDA (buscador)
* MOTOR DE RENDERIZADO
* SUBSISTEMA DE PERSISTENCIA DE DATOS
* INTERPRETE DE JAVASCRIPT

Lo normal es separar la parte de JavaScript, HTML y CSS en diferentes archivos.

HTML (esqueleto de la página), CSS (estilo, diseño gráfico) y JavaScript (fragmentos de código), todo para el entorno cliente.

JavaScript es un lenguaje interpretado, ya que no hay compilación, lo lee línea a línea. Aunque hace una comprobación inicial sobre si las variables están bien definidas y si hay algún problema, a esto se le llama compilado JIT (Just-In-Time), ya que hace una precompilación.

Es orientado a objetos

Es imperativo, instrucciones explícitas mediante órdenes

Es estructurado, tiene estructuras básicas, secuencia, selección e iteracion

Es débilmente tipado, no necesita indicar el tipo de variable, como php

Es de tipificación dinámica, es decir, que pueden cambiar el tipo del valor variables a lo largo de la ejecución.

En el cliente siempre hace falta un navegador con HTML para ejecutar JS, pero node permite ejecutar JavaScript sin falta de un navegador con HTML.

Como el código no es compilador, entra al navegador como texto plano, es decir, que cualquiera que entre a la página puede verlo y reutilizarlo sin permiso. Para solucionar esto se usa la ofuscación del código, que consiste en formar un código menos legible, pero no es muy eficaz.

La ubicación del Script es importante:

Si se pone en <head> se descarga el JS ANTES de empezara dibujar la pagina

Si se pone en <body> se descarga DURANTE el dibujado de la página

Antes del </body> se decarga DESPUÉS de dibujar la página. Que es lo mismo que usar defer en la etiqueta <script>

Con JS se puede escribir en la consola

Console.log(“ejemplo de consola”)

Se puede escribir en cualquier elemento HTML

Document.getElementByld(“identificador id”).innerHTML

Genera HTML con el método

Document.writer

Se puede usar un alert, por ejemplo, dentro de una etiqueta HTML siempre y cuando se quiera modificar solo un elemento.

Console.log

Console.infor

Console.warn

Console.error

Console.clear

Variable es un espacio de memoria a la que se le asigna un nombre y se guarda un valor.

Los valores son de un tipo concreto:

* Object (todos son objects, pero ya hay unos concretos, los primitivos)
* Tipos primitivos: string, number, boolean, undefinided, null, bigint y symbol

Se declaran con var(**YA NO, AHORA SE USA LET**), empiezan por letra, \_ o $. Pueden tener cadenas y números, son sensibles a mayúsculas. Se pueden declarar variables en una misma sentencia o asignar una operación a una variable.

**PARA DECLARAR LAS VARIABLES SE USA CAMEL CASE ->** miVariableNueva

Se usa let para declarar variables, porque asi se declara en ámbito local y con var es ámbito global, lo que puede causar problemas al usar la misma variable en distintos sitios con valores diferentes.

Const se usa para definir variables (**SE DECLARAN EN MAYUSCULAS Y CON \_ PARA LOS ESPACIOS**). Los objetos si pueden ser modificados aunque sean constantes.

Typeof e instanceof, sirve para saber el tipo de una variable o dato

Typeof -> devuelve una cadena que indica el tipo la variable.

Instanceof -> devuelve un boolean que afirma si esa clase viene instanciada de otra o no. Comprueba si contiene la propiedad prototype de un constructor.

**HOISTING**

Es un mecanismo que consiste en que el intérprete de JavaScript escanea el programa antes de ejecutarlo, buscando errores en la sintaxis.

**METODOS**

**Metodo toString() ->** Convierte número en cadena nombreVariable.toString(), (567)toString();

**Metodo toFixed ->** Ajuste de los decimales (REDONDEA)

Var numero = 99.99;

Numero.toFixed(0) -> devuelve 100. Redondea

Numero.toFixed(1) -> Devuelve 100.0. Redondea y 1 decimal, ya que tiene 2

Numero.toFixed(2) -> Devuelve 99.99. Ya que tiene 2 decimales

Numero.toFixed(3) -> Devuelve 99.990, ya que al pedir mas decimales de los tiene los rellena con 0

**Metodo toPrecision() ->** Ajusta la precisión exacta (PRECISION)

Numero.toPrecicion(0) -> DA ERROR

Numero.toPrecicion(1) -> Devuelve 1e+1

Numero.toPrecicion(2)-> Devuelve 100.

Numero.toPrecicion(3) -> Devuelve 99.99

**ENTRADA DE DATOS**

Window.prompt -> Si no se introduce valor, guarda “null” en la variable.

let libroLeido = prompt (“¿Cuál es el título del último libro que has leído?”, default);

Default se pone que en caso de que no conteste, en vez de ser null le asignará default

<p>INTRODUCE TU EDAD Y PULSA EL BOTON</p>

<input id= “edad” value=”18”/>

**FECHAS**

**`HOLA ${nombre\_variable}`** SE LLAMA LITERAL, PARA LLAMAR A VARIABLES SIN CONCATENAR NADA

var v = new Date(“11/23/2023”) //Formato corto

var v = new Date(“Ago 17 2023”) //Formato largo

var v = new Date(“Mon, Jun 20 2022 15:42:32 GMT +0100 (W. Euro Standard Time)”) //Formato complete

var v = new Date(“2022/08/17”) //Formato ISO

var diHola = (name) => `Hola ${name}`{

var diAdios = (name) => {

return `Adiós ${name}`

}

Console.log (diHola(‘Cris’));

En JavaScript switch deja evaluar tanto valores como expresiones (Los case pueden tener una expresión como < > ):

Switch (true){

Case age <18

Console.log(No puede entrar por joven)

Case age <65

Console.log(Puedes entrar)

Default

Console.log(No puedes entrar por mayor)

}

**DESESTRUCTURACION**

Separar una estructura, en js se hace con arrays u objetos

EXPANDIR (SPREAD) O AGRUPAR (AGRUPAR)

**OBJETOS NATIVOS Y DE ALTO NIVEL**

Posicion en memoria que tiene un identificador

OBJETOS NATIVOS EJEMPLOS (NO DEPENDEN DEL NAVEGADOR):

**- String**

**- Number**

**- Boolean**

**- Date**

**- Math**

**- RegExp (EXPRESIONES REGULARES)**

**- Array**

**- Function**

**- Object**

En JavaScript se define:

Var ciclista = {nombre: “Pedro”, apellido:”Gonzalez”, año: 1965};

Un objeto es una colección de variables (propiedades) y funciones (métodos).

Dentro de la definición de un objeto, se puede crear un método en donde se declaran las propiedades.

Para acceder a los valores de las propiedades se usa(con el ejemplo de ciclista):

Ciclista.nombre= Pedro

Ciclista[año]=1965 (Otra forma de mostrar el dato).

**EN JS SOLO HAY UN DATO NUMERICO (NUMBER)**

Se pueden guardar positivos, negativos, decimales, notación científica…

Metodos para cambios de base

Infinito y -infinito

NaN (NotANumber)

**MANEJO DE ERRORES**

Error -> Provoca que el programa deje de funcionar

Excepción -> situación que de alguna manera se soluciona

Try

{

Código que queremos “proteger”

}

Catch

{

Codigo que se activará cuando el código del try tenga un error o excepción

}

Finally

{

Codigo que después de todo se ejecuta siempre

}

Cuando se produce un error en JS, genera un objeto Error, que tiene 2 propiedades

* Name -> Nombre del error (EvalError, RangeError, ReferenceError, SyntaxError, TypeError, URlError)
* Message -> Cadena que describe el error

~~EvalError~~

~~InternalError~~

RangeError, valor fuera de los valores establecidos

ReferenceError, referencia a objeto invalida

SyntaxError, error de sintaxis en JS

TypeError, error de tipo de variable

URlError, error con EncodeURL o decodeURL

También se puede crear un error

Let miError=new Error (‘formato de correo incorrecto’);

Throw miError;

**OBJETOS**